ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ
Khoa Công nghệ Thông tin

PHẠM HỒNG THÁI

Bài giảng
NGÔN NGỮ LẬP TRÌNH C/C++

Hà Nội – 2003

LỜI NÓI ĐẦU

Ngôn ngữ lập trình (NNLT) C/C++ là một trong những ngôn ngữ lập trình hướng đối tượng mạnh và phổ biến hiện nay do tính mềm dẻo và đa năng của nó. Không chỉ các ứng dụng được viết trên C/C++ mà cả những chương trình hệ thống lớn đều được viết hầu hết trên C/C++. C++ là ngôn ngữ lập trình hướng đối tượng được phát triển trên nền tảng của C, không những khắc phục một số nhược điểm của ngôn ngữ C mà quan trọng hơn, C++ cung cấp cho người sử dụng (NSD) một phương tiện lập trình theo kỹ thuật mới: lập trình hướng đối tượng. Đây là kỹ thuật lập trình được sử dụng hầu hết trong các ngôn ngữ mạnh hiện nay, đặc biệt là các ngôn ngữ hoạt động trong môi truờng Windows như Microsoft Access, Visual Basic, Visual Foxpro …

Hiện nay NNLT C/C++ đã được đưa vào giảng dạy trong hầu hết các trường Đại học, Cao đẳng để thay thế một số NNLT đã cũ như FORTRAN, Pascal … Tập bài giảng này được viết ra với mục đích đó, trang bị kiến thức và kỹ năng thực hành cho sinh viên bắt đầu học vào NNLT C/C++ tại Khoa Công nghệ, Đại học Quốc gia Hà Nội. Để phù hợp với chương trình, tập bài giảng này chỉ đề cập một phần nhỏ đến kỹ thuật lập trình hướng đối tượng trong C++, đó là các kỹ thuật đóng gói dữ liệu, phương thức và định nghĩa mới các toán tử. Tên gọi của tập bài giảng này nói lên điều đó, có nghĩa nội dung của bài giảng thực chất là NNLT C được mở rộng với một số đặc điểm mới của C++. Về kỹ thuật lập trình hướng đối tượng (trong C++) sẽ được trang bị bởi một giáo trình khác. Tuy nhiên để ngắn gọn, trong tập bài giảng này tên gọi C/C++ sẽ được chúng tôi thay bằng C++.

Nội dung tập bài giảng này gồm 8 chương. Phần đầu gồm các chương từ 1 đến 6 chủ yếu trình bày về NNLT C++ trên nền tảng của kỹ thuật lập trình cấu trúc. Các chương còn lại (chương 7 và 8) sẽ trình bày các cấu trúc cơ bản trong C++ đó là kỹ thuật đóng gói (lớp và đối tượng) và định nghĩa phép toán mới cho lớp.

Tuy đã có nhiều cố gắng nhưng do thời gian và trình độ người viết có hạn nên chắc chắn không tránh khỏi sai sót, vì vậy rất mong nhận được sự góp ý của bạn đọc để bài giảng ngày càng một hoàn thiện hơn.

Tác giả.

CHƯƠNG 1

CÁC KHÁI NIỆM CƠ BẢN CỦA C++

Các yếu tố cơ bản
Môi trường làm việc của C++
Các bước để tạo và thực hiện một chương trình
Vào/ra trong C++

I. CÁC YẾU TỐ CƠ BẢN

Một ngôn ngữ lập trình (NNLT) bậc cao cho phép người sử dụng (NSD) biểu hiện ý tưởng của mình để giải quyết một vấn đề, bài toán bằng cách diễn đạt gần với ngôn ngữ thông thường thay vì phải diễn đạt theo ngôn ngữ máy (dãy các kí hiệu 0,1). Hiển nhiên, các ý tưởng NSD muốn trình bày phải được viết theo một cấu trúc chặt chẽ thường được gọi là *thuật toán* hoặc *giải thuật* và theo đúng các qui tắc của ngôn ngữ gọi là *cú pháp* hoặc *văn phạm*. Trong giáo trình này chúng ta bàn đến một ngôn ngữ lập trình như vậy, đó là ngôn ngữ lập trình C++ và làm thế nào để thể hiện các ý tưởng giải quyết vấn đề bằng cách viết thành chương trình trong C++.

Trước hết, trong mục này chúng ta sẽ trình bày về các qui định bắt buộc đơn giản và cơ bản nhất. Thông thường các qui định này sẽ được nhớ dần trong quá trình học ngôn ngữ, tuy nhiên để có một vài khái niệm tương đối hệ thống về NNLT C++ chúng ta trình bày sơ lược các khái niệm cơ bản đó. Người đọc đã từng làm quen với các NNLT khác có thể đọc lướt qua phần này.

1. Bảng ký tự của C++

Hầu hết các ngôn ngữ lập trình hiện nay đều sử dụng các kí tự tiếng Anh, các kí hiệu thông dụng và các con số để thể hiện chương trình. Các kí tự của những ngôn ngữ khác không được sử dụng (ví dụ các chữ cái tiếng Việt). Dưới đây là bảng kí tự được phép dùng để tạo nên những câu lệnh của ngôn ngữ C++.

- Các chữ cái la tinh (viết thường và viết hoa): a .. z và A .. Z. Cùng một chữ cái nhưng viết thường phân biệt với viết hoa. Ví dụ chữ cái 'a' là khác với 'A'.
- Dấu gạch dưới: _
- Các chữ số thập phân: 0, 1, . ., 9.

- Các ký hiệu toán học: +, -, *, /, % , &, ||, !, >, <, = ...
- Các ký hiệu đặc biệt khác: , ;: [ ], {}, #, dấu cách, ...

2. Từ khoá

Một từ khoá là một từ được qui định trước trong NNLT với một ý nghĩa cố định, thường dùng để chỉ các loại dữ liệu hoặc kết hợp thành câu lệnh. NSD có thể tạo ra những từ mới để chỉ các đối tượng của mình nhưng không được phép trùng với từ khoá. Dưới đây chúng tôi liệt kê một vài từ khoá thường gặp, ý nghĩa của các từ này, sẽ được trình bày dần trong các đề mục liên quan.

auto, break, case, char, continue, default, do, double, else, externe, float,

for, goto, if, int, long, register, return, short, sizeof, static, struct, switch,

typedef, union, unsigned, while ...

Một đặc trưng của C++ là các từ khoá luôn luôn được viết bằng chữ thường.

3. Tên gọi

Để phân biệt các đối tượng với nhau chúng cần có một tên gọi. Hầu hết một đối tượng được viết ra trong chương trình thuộc 2 dạng, một dạng đã có sẵn trong ngôn ngữ (ví dụ các từ khoá, tên các hàm chuẩn ...), một số do NSD tạo ra dùng để đặt tên cho hằng, biến, kiểu, hàm ... các tên gọi do NSD tự đặt phải tuân theo một số qui tắc sau:

- Là dãy ký tự liên tiếp (không chứa dấu cách) và phải bắt đầu bằng chữ cái hoặc gạch dưới.
- Phân biệt kí tự in hoa và thường.
- Không được trùng với từ khóa.
- Số lượng chữ cái dùng để phân biệt tên gọi có thể được đặt tuỳ ý.
- Chú ý các tên gọi có sẵn của C++ cũng tuân thủ theo đúng qui tắc trên.

Trong một chương trình nếu NSD đặt tên sai thì trong quá trình xử lý sơ bộ (trước khi chạy chương trình) máy sẽ báo lỗi (gọi là lỗi văn phạm).

Ví dụ 1 :

- Các tên gọi sau đây là đúng (được phép): i, i1, j, tinhoc, tin_hoc, luu_luong
- Các tên gọi sau đây là sai (không được phép): 1i, tin hoc, luu-luong-nuoc
- Các tên gọi sau đây là khác nhau: ha_noi, Ha_noi, HA_Noi, HA_NOI, ...

4. Chú thích trong chương trình

Một chương trình thường được viết một cách ngắn gọn, do vậy thông thường bên cạnh các câu lệnh chính thức của chương trình, NSD còn được phép viết vào chương trình các câu ghi chú, giải thích để làm rõ nghĩa hơn chương trình. Một chú thích có thể ghi chú về nhiệm vụ, mục đích, cách thức của thành phần đang được chú thích như biến, hằng, hàm hoặc công dụng của một đoạn lệnh ... Các chú thích sẽ làm cho chương trình sáng sủa, dễ đọc, dễ hiểu và vì vậy dễ bảo trì, sửa chữa về sau.

Có 2 cách báo cho chương trình biết một đoạn chú thích:

- Nếu chú thích là một đoạn kí tự bất kỳ liên tiếp nhau (trong 1 dòng hoặc trên nhiều dòng) ta đặt đoạn chú thích đó giữa cặp dấu đóng mở chú thích /* (mở) và */ (đóng).
- Nếu chú thích bắt đầu từ một vị trí nào đó cho đến hết dòng, thì ta đặt dấu // ở vị trí đó. Như vậy // sử dụng cho các chú thích chỉ trên 1 dòng.

Như đã nhắc ở trên, vai trò của đoạn chú thích là làm cho chương trình dễ hiểu đối với người đọc, vì vậy đối với máy các đoạn chú thích sẽ được bỏ qua. Lợi dụng đặc điểm này của chú thích đôi khi để tạm thời bỏ qua một đoạn lệnh nào đó trong chương trình (nhưng không xoá hẳn để khỏi phải gõ lại khi cần dùng đến) ta có thể đặt các dấu chú thích bao quanh đoạn lệnh này (ví dụ khi chạy thử chương trình, gỡ lỗi ...), khi cần sử dụng lại ta có thể bỏ các dấu chú thích.

Chú ý: Cặp dấu chú thích /* ... */ không được phép viết lồng nhau, ví dụ dòng chú thích sau là không được phép

```
/* Đây là đoạn chú thích /* chứa đoạn chú thích này */ như đoạn chú thích con */
```

cần phải sửa lại như sau:

- hoặc chỉ giữ lại cặp dấu chú thích ngoài cùng

```
/* Đây là đoạn chú thích chứa đoạn chú thích này như đoạn chú thích con */
```

- hoặc chia thành các đoạn chú thích liên tiếp nhau

```
/* Đây là đoạn chú thích */ /*chứa đoạn chú thích này*/ /*như đoạn chú thích con */
```

II. MÔI TRƯỜNG LÀM VIỆC CỦA C++

1. Khởi động - Thoát khỏi C++

Khởi động C++ cũng như mọi chương trình khác bằng cách nhấp đúp chuột lên biểu tượng của chương trình. Khi chương trình được khởi động sẽ hiện ra giao diện gồm có menu công việc và một khung cửa sổ bên dưới phục vụ cho soạn thảo. Một con

trỏ nhấp nháy trong khung cửa sổ và chúng ta bắt đầu nhập nội dung (văn bản) chương trình vào trong khung cửa sổ soạn thảo này. Mục đích của giáo trình này là trang bị những kiến thức cơ bản của lập trình thông qua NNLT C++ cho các sinh viên mới bắt đầu nên chúng tôi vẫn chọn trình bày giao diện của các trình biên dịch quen thuộc là Turbo C hoặc Borland C. Về các trình biên dịch khác độc giả có thể tự tham khảo trong các tài liệu liên quan.

Để kết thúc làm việc với C++ (soạn thảo, chạy chương trình ...) và quay về môi trường Windows chúng ta ấn **Alt-X**.

2. Giao diện và cửa sổ soạn thảo

a. Mô tả chung

Khi gọi chạy C++ trên màn hình sẽ xuất hiện một menu xổ xuống và một cửa sổ soạn thảo. Trên menu gồm có các nhóm chức năng: **File, Edit, Search, Run, Compile, Debug, Project, Options, Window, Help**. Để kích hoạt các nhóm chức năng, có thể ấn **Alt+chữ cái** biểu thị cho menu của chức năng đó (là chữ cái có gạch dưới). Ví dụ để mở nhóm chức năng **File** ấn **Alt+F**, sau đó dịch chuyển hộp sáng đến mục cần chọn rồi ấn Enter. Để thuận tiện cho NSD, một số các chức năng hay dùng còn được gắn với một tổ hợp các phím cho phép người dùng có thể chọn nhanh chức năng này mà không cần thông qua việc mở menu như đã mô tả ở trên. Một số tổ hợp phím cụ thể đó sẽ được trình bày vào cuối phần này. Các bộ chương trình dịch hỗ trợ người lập trình một môi trường tích hợp tức ngoài chức năng soạn thảo, nó còn cung cấp nhiều chức năng, tiện ích khác giúp người lập trình vừa có thể soạn thảo văn bản chương trình vừa gọi chạy chương trình vừa gỡ lỗi …

Các chức năng liên quan đến soạn thảo phần lớn giống với các bộ soạn thảo khác (như WinWord) do vậy chúng tôi chỉ trình bày tóm tắt mà không trình bày chi tiết ở đây.

b. Các chức năng soạn thảo

Giống hầu hết các bộ soạn thảo văn bản, bộ soạn thảo của Turbo C hoặc Borland C cũng sử dụng các phím sau cho quá trình soạn thảo:

- Dịch chuyển con trỏ: các phím mũi tên cho phép dịch chuyển con trỏ sang trái, phải một kí tự hoặc lên trên, xuống dưới 1 dòng. Để dịch chuyển nhanh có các phím như Home (về đầu dòng), End (về cuối dòng), PgUp, PgDn (lên, xuống một trang màn hình). Để dịch chuyển xa hơn có thể kết hợp các phím này cùng phím Control (Ctrl, ^) như ^PgUp: về đầu tệp, ^PgDn: về cuối tệp.
- Chèn, xoá, sửa: Phím Insert cho phép chuyển chế độ soạn thảo giữa chèn và đè. Các phím Delete, Backspace cho phép xoá một kí tự tại vị trí con trỏ và

trước vị trí con trỏ (xoá lùi).

- Các thao tác với khối dòng: Để đánh dấu khối dòng (thực chất là khối kí tự liền nhau bất kỳ) ta đưa con trỏ đến vị trí đầu ấn Ctrl-KB và Ctrl-KK tại vị trí cuối. Cũng có thể thao tác nhanh hơn bằng cách giữ phím Shift và dùng các phím dịch chuyển con trỏ quét từ vị trí đầu đến vị trí cuối, khi đó khối kí tự được đánh dấu sẽ chuyển mầu nền. Một khối được đánh dấu có thể dùng để cắt, dán vào một nơi khác trong văn bản hoặc xoá khỏi văn bản. Để thực hiện thao tác cắt dán, đầu tiên phải đưa khối đã đánh dấu vào bộ nhớ đệm bằng nhóm phím Shift-Delete (cắt), sau đó dịch chuyển con trỏ đến vị trí mới cần hiện nội dung vừa cắt và ấn tổ hợp phím Shift-Insert. Một đoạn văn bản được ghi vào bộ nhớ đệm có thể được dán nhiều lần vào nhiều vị trí khác nhau bằng cách lặp lại tổ hợp phím Shift-Insert tại các vị trí khác nhau trong văn bản. Để xoá một khối dòng đã đánh dấu mà không ghi vào bộ nhớ đệm, dùng tổ hợp phím Ctrl-Delete. Khi một nội dung mới ghi vào bộ nhớ đệm thì nó sẽ xoá (ghi đè) nội dung cũ đã có, do vậy cần cân nhắc để sử dụng phím Ctrl-Delete (xoá và không lưu lại nội dung vừa xoá vào bộ đệm) và Shift-Delete (xoá và lưu lại nội dung vừa xoá) một cách phù hợp.
- Tổ hợp phím Ctrl-A rất thuận lợi khi cần đánh dấu nhanh toàn bộ văn bản.

c. Chức năng tìm kiếm và thay thế

Chức năng này dùng để dịch chuyển nhanh con trỏ văn bản đến từ cần tìm. Để thực hiện tìm kiếm bấm Ctrl-QF, tìm kiếm và thay thế bấm Ctrl-QA. Vào từ hoặc nhóm từ cần tìm vào cửa sổ Find, nhóm thay thế (nếu dùng Ctrl-QA) vào cửa sổ Replace và đánh dấu vào các tuỳ chọn trong cửa sổ bên dưới sau đó ấn Enter. Các tuỳ chọn gồm: không phân biệt chữ hoa/thường, tìm từ độc lập hay đứng trong từ khác, tìm trong toàn văn bản hay chỉ trong phần được đánh dấu, chiều tìm đi đến cuối hay ngược về đầu văn bản, thay thế có hỏi lại hay không hỏi lại … Để dịch chuyển con trỏ đến các vùng khác nhau trong một menu hay cửa sổ chứa các tuỳ chọn ta sử dụng phím Tab.

d. Các chức năng liên quan đến tệp

- Ghi tệp lên đĩa: Chọn menu File\Save hoặc phím F2. Nếu tên tệp chưa có (còn mang tên Noname.cpp) máy sẽ yêu cầu cho tên tệp. Phần mở rộng của tên tệp được mặc định là CPP.
- Soạn thảo tệp mới: Chọn menu File\New. Hiện ra cửa sổ soạn thảo trắng và tên file tạm thời lấy là Noname.cpp.
- Soạn thảo tệp cũ: Chọn menu File\Open hoặc ấn phím F3, nhập tên tệp hoặc dịch chuyển con trỏ trong vùng danh sách tệp bên dưới đến tên tệp cần soạn rồi ấn Enter. Cũng có thể áp dụng cách này để soạn tệp mới khi không nhập

vào tên tệp cụ thể.

- Ghi tệp đang soạn thảo lên đĩa với tên mới: Chọn menu File\Save As và nhập tên tệp mới vào rồi ấn Enter.

e. Chức năng dịch và chạy chương trình

- Ctrl-F9: Khởi động chức năng dịch và chạy toàn bộ chương trình.
- F4: Chạy chương trình từ đầu đến dòng lệnh hiện tại (đang chứa con trỏ)
- F7: Chạy từng lệnh một của hàm main(), kể cả các lệnh con trong hàm.
- F8: Chạy từng lệnh một của hàm main(). Khi đó mỗi lời gọi hàm được xem là một lệnh (không chạy từng lệnh trong các hàm được gọi).

Các chức năng liên quan đến dịch chương trình có thể được chọn thông qua menu Compile (Alt-C).

f. Tóm tắt một số phím nóng hay dùng

- Các phím kích hoạt menu: Alt+chữ cái đại diện cho nhóm menu đó. Ví dụ Alt-F mở menu File để chọn các chức năng cụ thể trong nó như Open (mở file), Save (ghi file lên đĩa), Print (in nội dung văn bản chương trình ra máy in), … Alt-C mở menu Compile để chọn các chức năng dịch chương trình.
- Các phím dịch chuyển con trỏ khi soạn thảo.
- F1: mở cửa sổ trợ giúp. Đây là chức năng quan trọng giúp người lập trình nhớ tên lệnh, cú pháp và cách sử dụng.
- F2: ghi tệp lên đĩa.
- F3: mở tệp cũ ra sửa chữa hoặc soạn thảo tệp mới.
- F4: chạy chương trình đến vị trí con trỏ.
- F5: Thu hẹp/mở rộng cửa sổ soạn thảo.
- F6: Chuyển đổi giữa các cửa sổ soạn thảo.
- F7: Chạy chương trình theo từng lệnh, kể cả các lệnh trong hàm con.
- F8: Chạy chương trình theo từng lệnh trong hàm chính.
- F9: Dịch và liên kết chương trình. Thường dùng chức năng này để tìm lỗi cú pháp của chương trình nguồn trước khi chạy.
- Alt-F7: Chuyển con trỏ về nơi gây lỗi trước đó.
- Alt-F8: Chuyển con trỏ đến lỗi tiếp theo.

- Ctrl-F9: Chạy chương trình.
- Ctrl-Insert: Lưu khối văn bản được đánh dấu vào bộ nhớ đệm.
- Shift-Insert: Dán khối văn bản trong bộ nhớ đệm vào văn bản tại vị trí con trỏ.
- Shift-Delete: Xoá khối văn bản được đánh dấu, lưu nó vào bộ nhớ đệm.
- Ctrl-Delete: Xoá khối văn bản được đánh dấu (không lưu vào bộ nhớ đệm).
- Alt-F5: Chuyển sang cửa sổ xem kết quả của chương trình vừa chạy xong.
- Alt-X: thoát C++ về lại Windows.

3. Cấu trúc một chương trình trong C++

Một chương trình C++ có thể được đặt trong một hoặc nhiều file văn bản khác nhau. Mỗi file văn bản chứa một số phần nào đó của chương trình. Với những chương trình đơn giản và ngắn thường chỉ cần đặt chúng trên một file.

Một chương trình gồm nhiều hàm, mỗi hàm phụ trách một công việc khác nhau của chương trình. Đặc biệt trong các hàm này có một hàm duy nhất có tên hàm là main(). Khi chạy chương trình, các câu lệnh trong hàm main() sẽ được thực hiện đầu tiên. Trong hàm main() có thể có các câu lệnh gọi đến các hàm khác khi cần thiết, và các hàm này khi chạy lại có thể gọi đến các hàm khác nữa đã được viết trong chương trình (trừ việc gọi quay lại hàm main()). Sau khi chạy đến lệnh cuối cùng của hàm main() chương trình sẽ kết thúc.

Cụ thể, thông thường một chương trình gồm có các nội dung sau:

- Phần khai báo các tệp nguyên mẫu: khai báo tên các tệp chứa những thành phần có sẵn (như các hằng chuẩn, kiểu chuẩn và các hàm chuẩn) mà NSD sẽ dùng trong chương trình.
- Phần khai báo các kiểu dữ liệu, các biến, hằng ... do NSD định nghĩa và được dùng chung trong toàn bộ chương trình.
- Danh sách các hàm của chương trình (do NSD viết, bao gồm cả hàm main()). Cấu trúc chi tiết của mỗi hàm sẽ được đề cập đến trong chương 4.

Dưới đây là một đoạn chương trình đơn giản chỉ gồm 1 hàm chính là hàm main(). Nội dung của chương trình dùng in ra màn hình dòng chữ: *Chào các bạn, bây giờ là 2 giờ*.

```
#include <iostream.h>                // khai báo tệp nguyên mẫu để
void main()                          // được sử dụng toán tử in cout <<
{
```

```
    int h = 2,                                  // Khai báo và khởi tạo biến h = 2
    cout << “Chào các bạn, bây giờ là ” << h << " giờ" ;          // in ra màn hình
}
```

Dòng đầu tiên của chương trình là khai báo tệp nguyên mẫu iostream.h. Đây là khai báo bắt buộc vì trong chương trình có sử dụng phương thức chuẩn “cout <<” (in ra màn hình), phương thức này được khai báo và định nghĩa sẵn trong iostream.h.

Không riêng hàm main(), mọi hàm khác đều phải bắt đầu tập hợp các câu lệnh của mình bởi dấu { và kết thúc bởi dấu }. Tập các lệnh bất kỳ bên trong cặp dấu này được gọi là khối lệnh. Khối lệnh là một cú pháp cần thiết trong các câu lệnh có cấu trúc như ta sẽ thấy trong các chương tiếp theo.

III. CÁC BƯỚC ĐỂ TẠO VÀ THỰC HIỆN MỘT CHƯƠNG TRÌNH

1. Qui trình viết và thực hiện chương trình

Trước khi viết và chạy một chương trình thông thường chúng ta cần:

1. Xác định yêu cầu của chương trình. Nghĩa là xác định dữ liệu đầu vào (input) cung cấp cho chương trình và tập các dữ liệu cần đạt được tức đầu ra (output). Các tập hợp dữ liệu này ngoài các tên gọi còn cần xác định kiểu của nó.Ví dụ để giải một phương trình bậc 2 dạng: $ax^2 + bx + c = 0$, cần báo cho chương trình biết dữ liệu đầu vào là a, b, c và đầu ra là nghiệm x1 và x2 của phương trình. Kiểu của a, b, c, x1, x2 là các số thực.
2. Xác định thuật toán giải.
3. Cụ thể hoá các khai báo kiểu và thuật toán thành dãy các lệnh, tức viết thành chương trình thông thường là trên giấy, sau đó bắt đầu soạn thảo vào trong máy. Quá trình này được gọi là soạn thảo chương trình nguồn.
4. Dịch chương trình nguồn để tìm và sửa các lỗi gọi là lỗi cú pháp.
5. Chạy chương trình, kiểm tra kết quả in ra trên màn hình. Nếu sai, sửa lại chương trình, dịch và chạy lại để kiểm tra. Quá trình này được thực hiện lặp đi lặp lại cho đến khi chương trình chạy tốt theo yêu cầu đề ra của NSD.

2. Soạn thảo tệp chương trình nguồn

Soạn thảo chương trình nguồn là một công việc đơn giản: gõ nội dung của chương trình (đã viết ra giấy) vào trong máy và lưu lại nó lên đĩa. Thông thường khi đã lưu lại chương trình lên đĩa lần sau sẽ không cần phải gõ lại. Có thể soạn chương trình nguồn trên các bộ soạn thảo (editor) khác nhưng phải chạy trong môi trường tích hợp

C++ (Borland C, Turbo C). Mục đích của soạn thảo là tạo ra một văn bản chương trình và đưa vào bộ nhớ của máy. Văn bản chương trình cần được trình bày sáng sủa, rõ ràng. Các câu lệnh cần gióng thẳng cột theo cấu trúc của lệnh (các lệnh chứa trong một lệnh cấu trúc được trình bày thụt vào trong so với điểm bắt đầu của lệnh). Các chú thích nên ghi ngắn gọn, rõ nghĩa và phù hợp.

3. Dịch chương trình

Sau khi đã soạn thảo xong chương trình nguồn, bước tiếp theo thường là dịch (ấn tổ hợp phím Alt-F9) để tìm và sửa các lỗi gọi là lỗi cú pháp. Trong khi dịch C++ sẽ đặt con trỏ vào nơi gây lỗi (viết sai cú pháp) trong văn bản. Sau khi sửa xong một lỗi NSD có thể dùng Alt-F8 để chuyển con trỏ đến lỗi tiếp theo hoặc dịch lại. Để chuyển con trỏ về ngược lại lỗi trước đó có thể dùng Alt-F7. Quá trình sửa lỗi – dịch được lặp lại cho đến khi văn bản đã được sửa hết lỗi cú pháp.

Sản phẩm sau khi dịch là một tệp mới gọi là chương trình đích có đuôi EXE tức là tệp mã máy để thực hiện.Tệp này có thể lưu tạm thời trong bộ nhớ phục vụ cho quá trình chạy chương trình hoặc lưu lại trên đĩa tuỳ theo tuỳ chọn khi dịch của NSD. Trong và sau khi dịch, C++ sẽ hiện một cửa sổ chứa thông báo về các lỗi (nếu có), hoặc thông báo chương trình đã được dịch thành công (không còn lỗi). Các lỗi này được gọi là lỗi cú pháp.

Để dịch chương trình ta chọn menu \Compile\Compile hoặc \Compile\Make hoặc nhanh chóng hơn bằng cách ấn tổ hợp phím Alt-F9.

4. Chạy chương trình

Ấn Ctrl-F9 để chạy chương trình, nếu chương trình chưa dịch sang mã máy, máy sẽ tự động dịch lại trước khi chạy. Kết quả của chương trình sẽ hiện ra trong một cửa sổ kết quả để NSD kiểm tra. Nếu kết quả chưa được như mong muốn, quay lại văn bản để sửa và lại chạy lại chương trình. Quá trình này được lặp lại cho đến khi chương trình chạy đúng như yêu cầu đã đề ra. Khi chương trình chạy, cửa sổ kết quả sẽ hiện ra tạm thời che khuất cửa sổ soạn thảo. Sau khi kết thúc chạy chương trình cửa sổ soạn thảo sẽ tự động hiện ra trở lại và che khuất cửa sổ kết quả. Để xem lại kết quả đã hiện ấn Alt-F5. Sau khi xem xong để quay lại cửa sổ soạn thảo ấn phím bất kỳ.

IV. VÀO/RA TRONG C++

Trong phần này chúng ta làm quen một số lệnh đơn giản cho phép NSD nhập dữ liệu vào từ bàn phím hoặc in kết quả ra màn hình. Trong phần sau của giáo trình chúng ta sẽ khảo sát các câu lệnh vào/ra phức tạp hơn

1. Vào dữ liệu từ bàn phím

Để nhập dữ liệu vào cho các biến có tên **biến_1**, **biến_2**, **biến_3** chúng ta sử dụng câu lệnh:

```
cin >> biến_1 ;
cin >> biến_2 ;
cin >> biến_3 ;
```

hoặc:

```
cin >> biến_1 >> biến_2 >> biến_3 ;
```

biến_1, biến_2, biến_3 là các **biến** được sử dụng để lưu trữ các giá trị NSD nhập vào từ bàn phím. Khái niệm biến sẽ được mô tả cụ thể hơn trong chương 2, ở đây biến_1, biến_2, biến_3 được hiểu là các tên gọi để chỉ 3 giá trị khác nhau. Hiển nhiên có thể nhập dữ liệu nhiều hơn 3 biến bằng cách tiếp tục viết tên biến vào bên phải sau dấu >> của câu lệnh.

Khi chạy chương trình nếu gặp các câu lệnh trên chương trình sẽ "tạm dừng" để chờ NSD nhập dữ liệu vào cho các biến. Sau khi NSD nhập xong dữ liệu, chương trình sẽ tiếp tục chạy từ câu lệnh tiếp theo sau của các câu lệnh trên.

Cách thức nhập dữ liệu của NSD phụ thuộc vào loại giá trị của biến cần nhập mà ta gọi là **kiểu,** ví dụ nhập một số có cách thức khác với nhập một chuỗi kí tự. Giả sử cần nhập độ dài hai cạnh của một hình chữ nhật, trong đó cạnh dài được qui ước bằng tên biến cd và chiều rộng được qui ước bởi tên biến cr. Câu lệnh nhập sẽ như sau:

```
cin >> cd >> cr ;
```

Khi máy dừng chờ nhập dữ liệu NSD sẽ gõ giá trị cụ thể của các chiều dài, rộng theo đúng thứ tự trong câu lệnh. Các giá trị này cần cách nhau bởi ít nhất một dấu trắng (ta qui ước gọi dấu trắng là một trong 3 loại dấu được nhập bởi các phím sau: phím spacebar (dấu cách), phím tab (dấu tab) hoặc phím Enter (dấu xuống dòng)). Các giá trị NSD nhập vào cũng được hiển thị trên màn hình để NSD dễ theo dõi.

Ví dụ nếu NSD nhập vào 23 11 ↵ thì chương trình sẽ gán giá trị 23 cho biến cd và 11 cho biến cr.

Chú ý: giả sử NSD nhập 2311 ↵ (không có dấu cách giữa 23 và 11) thì chương trình sẽ xem 2311 là một giá trị và gán cho cd. Máy sẽ tạm dừng chờ NSD nhập tiếp giá trị cho biến cr.

2. In dữ liệu ra màn hình

Để in giá trị của các **biểu thức** ra màn hình ta dùng câu lệnh sau:

```
cout << bt_1 ;
```

```
cout << bt_2 ;
cout << bt_3 ;
```

hoặc:

```
cout << bt_1 << bt_2 << bt_3 ;
```

cũng giống câu lệnh nhập ở đây chúng ta cũng có thể mở rộng lệnh in với nhiều hơn 3 biểu thức. Câu lệnh trên cho phép in giá trị của các biểu thức bt_1, bt_2, bt_3. Các giá trị này có thể là tên của biến hoặc các kết hợp tính toán trên biến.

Ví dụ để in câu "Chiều dài là " và số 23 và tiếp theo là chữ "mét", ta có thể sử dụng 3 lệnh sau đây:

```
cout << "Chiều dài là" ;
cout << 23 ;
cout << "mét";
```

hoặc có thể chỉ bằng 1 lệnh:

```
cout << "Chiều dài là 23 mét" ;
```

Trường hợp chưa biết giá trị cụ thể của chiều dài, chỉ biết hiện tại giá trị này đã được lưu trong biến cd (ví dụ đã được nhập vào là 23 từ bàn phím bởi câu lệnh cin >> cd trước đó) và ta cần biết giá trị này là bao nhiêu thì có thể sử dụng câu lệnh in ra màn hình.

```
cout << "Chiều dài là" << cd << "mét" ;
```

Khi đó trên màn hình sẽ hiện ra dòng chữ: "Chiều dài là 23 mét". Như vậy trong trường hợp này ta phải dùng đến ba lần dấu phép toán << chứ không phải một như câu lệnh trên. Ngoài ra phụ thuộc vào giá trị hiện được lưu trong biến cd, chương trình sẽ in ra số chiều dài thích hợp chứ không chỉ in cố định thành "chiều dài là 23 mét". Ví dụ nếu cd được nhập là 15 thì lệnh trên sẽ in câu "chiều dài là 15 mét".

Một giá trị cần in không chỉ là một biến như cd, cr, ... mà còn có thể là một biểu thức, điều này cho phép ta dễ dàng yêu cầu máy in ra diện tích và chu vi của hình chữ nhật khi đã biết cd và cr bằng các câu lệnh sau:

```
cout << "Diện tích = " << cd * cr ;
cout << "Chu vi = " << 2 * (cd + cr) ;
```

hoặc gộp tất cả thành 1 câu lệnh:

```
cout << Diện tích = " << cd * cr << '\n' << " Chu vi = " << 2 * (cd + cr) ;
```

ở đây có một kí tự đặc biệt: đó là kí tự '\n' kí hiệu cho kí tự xuống dòng, khi gặp kí tự này chương trình sẽ in các phần tiếp theo ở đầu dòng kế tiếp. Do đó kết quả của câu lệnh trên là 2 dòng sau đây trên màn hình:

```
Diện tích = 253
Chu vi = 68
```

ở đây 253 và 68 lần lượt là các giá trị mà máy tính được từ các biểu thức cd * cr, và 2 * (cd + cr) trong câu lệnh in ở trên.

Chú ý: để sử dụng các câu lệnh nhập và in trong phần này, đầu chương trình phải có dòng khai báo #include <iostream.h>.

Thông thường ta hay sử dụng lệnh in để in câu thông báo nhắc NSD nhập dữ liệu trước khi có câu lệnh nhập. Khi đó trên màn hình sẽ hiện dòng thông báo này rồi mới tạm dừng chờ dữ liệu nhập vào từ bàn phím. Nhờ vào thông báo này NSD sẽ biết phải nhập dữ liệu, nhập nội dung gì và như thế nào ... ví dụ:

```
cout << "Hãy nhập chiều dài: "; cin >> cd;
cout << "Và nhập chiều rộng: "; cin >> cr;
```

khi đó máy sẽ in dòng thông báo "Hãy nhập chiều dài: " và chờ sau khi NSD nhập xong 23 ↵, máy sẽ thực hiện câu lệnh tiếp theo tức in dòng thông báo "Và nhập chiều rộng: " và chờ đến khi NSD nhập xong 11 ↵ chương trình sẽ tiếp tục thực hiện các câu lệnh tiếp theo.

Ví dụ 2 : Từ các thảo luận trên ta có thể viết một cách đầy đủ chương trình tính diện tích và chu vi của một hình chữ nhật. Để chương trình có thể tính với các bộ giá trị khác nhau của chiều dài và rộng ta cần lưu giá trị này vào trong các biến (ví dụ cd, cr).

```
#include <iostream.h>      // khai báo tệp nguyên mẫu để dùng được cin, cout
void main()                // đây là hàm chính của chương trình
{
    float cd, cr ;      // khai báo các biến có tên cd, cr để chứa độ dài các cạnh
    cout << "Hãy nhập chiều dài: " ; cin >> cd ;          // nhập dữ liệu
    cout << "Hãy nhập chiều rộng: " ; cin >> cr ;
    cout << "Diện tích = " << cd * cr << '\n' ;           // in kết quả
    cout << "Chu vi = " << 2 * (cd + cr) << '\n';
    return ;
}
```

Chương trình này có thể gõ vào máy và chạy. Khi chạy đến câu lệnh nhập, chương trình dừng để chờ nhận chiều dài và chiều rộng, NSD nhập các giá trị cụ thể, chương trình sẽ tiếp tục thực hiện và in ra kết quả. Thông qua câu lệnh nhập dữ liệu và 2 biến cd, cr NSD có thể yêu cầu chương trình cho kết quả của một hình chữ nhật bất

kỳ chứ không chỉ trong trường hợp hình có chiều dài 23 và chiều rộng 11 như trong ví dụ cụ thể trên.

3. Định dạng thông tin cần in ra màn hình

Một số định dạng đơn giản được chúng tôi trình bày trước ở đây. Các định dạng chi tiết và phức tạp hơn sẽ được trình bày trong các phần sau của giáo trình. Để sử dụng các định dạng này cần khai báo file nguyên mẫu <iomanip.h> ở đầu chương trình bằng chỉ thị #include <iomanip.h>.

- endl: Tương đương với kí tự xuống dòng '\n'.
- setw(n): Bình thường các giá trị được in ra bởi lệnh cout << sẽ thẳng theo lề trái với độ rộng phụ thuộc vào độ rộng của giá trị đó. Phương thức này qui định độ rộng dành để in ra các giá trị là n cột màn hình. Nếu n lớn hơn độ dài thực của giá trị, giá trị sẽ in ra theo lề phải, để trống phần thừa (dấu cách) ở trước.
- setprecision(n): Chỉ định số chữ số của phần thập phân in ra là n. Số sẽ được làm tròn trước khi in ra.
- setiosflags(ios::showpoint): Phương thức setprecision chỉ có tác dụng trên một dòng in. Để cố định các giá trị đã đặt cho mọi dòng in (cho đến khi đặt lại giá trị mới) ta sử dụng phương thức setiosflags(ios::showpoint).

Ví dụ sau minh hoạ cách sử dụng các phương thức trên.

Ví dụ 3 :

```
#include <iostream.h>                    // để sử dụng cout <<
#include <iomanip.h>                     // để sử dụng các định dạng
#include <conio.h>                       // để sử dụng các hàm clrscr() và getch()
void main()
{
      clrscr();                          // xoá màn hình
      cout << "CHI TIÊU" << endl << "=======" << endl ;
      cout << setiosflags(ios::showpoint) << setprecision(2) ;
      cout << "Sách vở" << setw(20) << 123.456 << endl;
      cout << "Thức ăn" << setw(20) << 2453.6 << endl;
      cout << "Quần áo lạnh" << setw(15) << 3200.0 << endl;
```

```
    getch();                        // tạm dừng (để xem kết quả)
    return ;                        // kết thúc thực hiện hàm main()
}
```

Chương trình này khi chạy sẽ in ra bảng sau:

```
CHI TIÊU
========
Sách vở          123.46
Thức ăn          2453.60
Quần áo lạnh     3200.00
```

Chú ý: toán tử nhập >> chủ yếu làm việc với dữ liệu kiểu số. Để nhập kí tự hoặc xâu kí tự, C++ cung cấp các phương thức (hàm) sau:

- **cin.get(c)**: cho phép nhập một kí tự vào biến kí tự c,
- **cin.getline(s,n)**: cho phép nhập tối đa n-1 kí tự vào xâu s.

các hàm trên khi thực hiện sẽ lấy các kí tự còn lại trong bộ nhớ đệm (của lần nhập trước) để gán cho c hoặc s. Do toán tử cin >> x sẽ để lại kí tự xuống dòng trong bộ đệm nên kí tự này sẽ làm trôi các lệnh sau đó như cin.get(c), cin.getline(s,n) (máy không dừng để nhập cho c hoặc s). Vì vậy trước khi sử dụng các phương thức cin.get(c) hoặc cin.getline(s,n) nên sử dụng phương thức cin.ignore(1) để lấy ra kí tự xuống dòng còn sót lại trong bộ đệm. Ví dụ đoạn lệnh sau cho phép nhập một số nguyên x (bằng toán tử >>) và một kí tự c (bằng phương thức cin.get(c)):

```
int x;
char c;
cin >> x; cin.ignore(1);
cin.get(c);
```

4. Vào/ra trong C

Trong phần trên chúng tôi đã trình bày 2 toán tử vào/ra và một số phương thức, hàm nhập và định dạng trong C++. Phần này chúng tôi trình bày các câu lênh nhập xuất theo khuôn dạng cũ trong C. Hiển nhiên các câu lệnh này vẫn dùng được trong chương trình viết bằng C++, tuy nhiên chỉ nên sử dụng hoặc các câu lệnh của C++ hoặc của C, không nên dùng lẫn lộn cả hai vì dễ gây nhầm lẫn. Do đó mục này chỉ có

giá trị tham khảo để bạn đọc có thể hiểu được các câu lệnh vào/ra trong các chương trình viết theo NNLT C cũ.

a. In kết quả ra màn hình

Để in các giá trị bt_1, bt_2, …, bt_n ra màn hình theo một khuôn dạng mong muốn ta có thể sử dụng câu lệnh sau đây:

printf(dòng định dạng, bt_1, bt_2, ..., bt_n) ;

trong đó dòng định dạng là một dãy kí tự đặt trong cặp dấu nháy kép (“”) qui định khuôn dạng cần in của các giá trị bt_1, bt_2, …, bt_n. Các bt_i có thể là các hằng, biến hay các biểu thức tính toán. Câu lệnh trên sẽ in giá trị của các bt_i này theo thứ tự xuất hiện của chúng và theo qui định được cho trong dòng định dạng.

Ví dụ, giả sử x = 4, câu lệnh:

printf(“%d %0.2f”, 3, x + 1) ;

sẽ in các số 3 và 5.00 ra màn hình, trong đó 3 được in dưới dạng số nguyên (được qui định bởi “%d”) và x + 1 (có giá trị là 5) được in dưới dạng số thực với 2 số lẻ thập phân (được qui định bởi “%0.2f”). Cụ thể, các kí tự đi sau kí hiệu % dùng để định dạng việc in gồm có:

d	in số nguyên dưới dạng hệ thập phân
o	in số nguyên dạng hệ 8
x, X	in số nguyên dạng hệ 16
u	in số nguyên dạng không dấu
c	in kí tự
s	in xâu kí tự
e, E	in số thực dạng dấu phẩy động
f	in số thực dạng dấu phẩy tĩnh

– Các kí tự trên phải đi sau dấu %. Các kí tự nằm trong dòng định dạng nếu không đi sau % thì sẽ được in ra màn hình. Muốn in % phải viết 2 lần (tức %%).

Ví dụ câu lệnh: printf(“Tỉ lệ học sinh giỏi: %0.2f %%”, 32.486) ;

sẽ in câu “Tỉ lệ học sinh giỏi: “, tiếp theo sẽ in số 32.486 được làm tròn đến 2 số lẻ thập phân lấp vào vị trí của “%0.2f”, và cuối cùng sẽ in dấu “%” (do có %% trong dòng định dạng). Câu được in ra màn hình sẽ là:

Tỉ lệ học sinh giỏi: 32.49%

Chú ý: Mỗi bt_i cần in phải có một định dạng tương ứng trong dòng định dạng.

Ví dụ câu lệnh trên cũng có thể viết:

```
printf("%s %0.2f" , "Tỉ lệ học sinh giỏi: ", 32.486);
```

trong câu lệnh này có 2 biểu thức cần in. Biểu thức thứ nhất là xâu kí tự "Tỉ lệ học sinh giỏi:" được in với khuôn dạng %s (in xâu kí tự) và biểu thức thứ hai là 32.486 được in với khuôn dạng %0.2f (in số thực với 2 số lẻ phần thập phân).

- Nếu giữa kí tự % và kí tự định dạng có số biểu thị độ rộng cần in thì giá trị in ra sẽ được gióng cột sang lề phải, để trống các dấu cách phía trước. Nếu độ rộng âm (thêm dấu trừ – phía trước) sẽ gióng cột sang lề trái. Nếu không có độ rộng hoặc độ rộng bằng 0 (ví dụ %0.2f) thì độ rộng được tự điều chỉnh đúng bằng độ rộng của giá trị cần in.
- Dấu + trước độ rộng để in giá trị số kèm theo dấu (dương hoặc âm)
- Trước các định dạng số cần thêm kí tự l (ví dụ ld, lf) khi in số nguyên dài long hoặc số thực với độ chính xác gấp đôi double.

Ví dụ 4 :

```
main()
{
    int i = 2, j = 3 ;
    printf("Chương trình tính tổng 2 số nguyên:\ni + j = %d", i+j);
}
```

sẽ in ra:

```
Chương trình tính tổng 2 số nguyên:
i + j = 5.
```

b. Nhập dữ liệu từ bàn phím

scanf(dòng định dạng, biến_1, biến_2, ..., biến_n) ;

Lệnh này cho phép nhập dữ liệu vào cho các biến biến_1, …, biến_n. Trong đó dòng định dạng chứa các định dạng về kiểu biến (nguyên, thực, kí tự …) được viết như trong mô tả câu lệnh printf. Các biến được viết dưới dạng địa chỉ của chúng tức có dấu & trước mỗi tên biến. Ví dụ câu lệnh:

```
scanf("%d %f %ld", &x, &y, &z) ;
```

cho phép nhập giá trị cho các biến x, y, z trong đó x là biến nguyên, y là biến thực và z là biến nguyên dài (long). Câu lệnh:

```
scanf("%2d %f %lf %3s", &i, &x, &d, s);
```

cho phép nhập giá trị cho các biến i, x, d, s, trong đó i là biến nguyên có 2 chữ số, f là biến thực (độ dài tùy ý), d là biến nguyên dài và s là xâu kí tự có 3 kí tự. Giả sử NSD nhập vào dãy dữ liệu: 12345 67abcd ↵ thì các biến trên sẽ được gán các giá trị như sau: i = 12, x = 345, d = 67 và s = "abc". Kí tự d và dấu enter (↵) sẽ được lưu lại trong bộ nhớ và tự động gán cho các biến của lần nhập sau.

Cuối cùng, chương trình trong ví dụ 3 được viết lại với printf() và scanf() như sau:

Ví dụ 5 :

```
#include <stdio.h>              // để sử dụng các hàm printf() và scanf()
#include <conio.h>              // để sử dụng các hàm clrscr() và getch()
void main()
{
    clrscr();                   // xoá màn hình
    printf("CHI TIÊU\n=======\n") ;
    printf("Sách vở %20.2f\n" , 123.456) ;
    printf("Thức ăn %20.2f\n" , 2453.6) ;
    printf("Quần áo lạnh %15.2f\n" , 3200.0) ;
    getch();                    // tạm dừng (để xem kết quả)
    return ;                    // kết thúc thực hiện hàm main()
}
```

BÀI TẬP

1. Những tên gọi nào sau đây là hợp lệ:

– x – 123variabe – tin_hoc – toan tin – so-dem

– RADIUS – one.0 – number# – Radius – nam2000

2. Bạn hãy thử viết một chương trình ngắn nhất có thể được.

3. Tìm các lỗi cú pháp trong chương trình sau:

```
#include (iostream.h)
void main();                    / Giải phương trình bậc 1
{
    cout << 'Day la chương trình: Gptb1.\nXin chao cac ban';
    getch();
}
```

4. Viết chương trình in nội dung một bài thơ nào đó.

5. Viết chương trình in ra 4 dòng, 2 cột gồm các số sau và gióng cột:

– thẳng theo lề trái	0.63	64.1
– thẳng theo lề phải	12.78	-11.678
– thẳng theo dấu chấm thập phân	-124. 6	59.002
	65.7	-1200.654

6. Hãy viết và chạy các chương trình trong các ví dụ 3, 5.

7. Chương trình sau khai báo 5 biến kí tự a, b, c, d, e và một biến số nam. Hãy điền thêm các câu lệnh vào các dòng … để chương trình thực hiện nhiệm vụ sau:

- Nhập giá trị cho biến nam
- Nhập giá trị cho các biến kí tự a, b, c, d, e.
- In ra màn hình dòng chữ được ghép bởi 5 kí tự đã nhập và chữ "năm" sau đó in số đã nhập (nam). Ví dụ nếu 5 chữ cái đã nhập là 'H', 'A', 'N', 'O', 'I' và nam được nhap là 2000, thì màn hình in ra dòng chữ: HANOI năm 2000.
- Nhập chương trình đã sửa vào máy và chạy để kiểm tra kết quả.

```
#include <iostream.h>
#include <conio.h>
main()
{
    int nam;
    char a, b, c, d, e;
    clrscr();
    cin >> nam ;
    ... ;
    cin.get(a); cin.get(b); cin.get(c); ... ; ... ;
```

```
        // in kết quả
        cout << a << ... << ... << ... << ... << " nam " << ... ;
        getch();
}
```

CHƯƠNG 2

KIỂU DỮ LIỆU, BIỂU THỨC VÀ CÂU LỆNH

Kiểu dữ liệu đơn giản
Hằng - khai báo và sử dụng hằng
Biến - khai báo và sử dụng biến
Phép toán, biểu thức và câu lệnh
Thư viện các hàm toán học

I. KIỂU DỮ LIỆU ĐƠN GIẢN

1. Khái niệm về kiểu dữ liệu

Thông thường dữ liệu hay dùng là số và chữ. Tuy nhiên việc phân chia chỉ 2 loai dữ liệu là không đủ. Để dễ dàng hơn cho lập trình, hầu hết các NNLT đều phân chia dữ liệu thành nhiều kiểu khác nhau được gọi là các kiểu cơ bản hay chuẩn. Trên cơ sở kết hợp các kiểu dữ liệu chuẩn, NSD có thể tự đặt ra các kiểu dữ liệu mới để phục vụ cho chương trình giải quyết bài toán của mình. Có nghĩa lúc đó mỗi đối tượng được quản lý trong chương trình sẽ là một tập hợp nhiều thông tin hơn và được tạo thành từ nhiều loại (kiểu) dữ liệu khác nhau. Dưới đây chúng ta sẽ xét đến một số kiểu dữ liệu chuẩn được qui định sẵn bởi C++.

Một biến như đã biết là một số ô nhớ liên tiếp nào đó trong bộ nhớ dùng để lưu trữ dữ liệu (vào, ra hay kết quả trung gian) trong quá trình hoạt động của chương trình. Để quản lý chặt chẽ các biến, NSD cần khai báo cho chương trình biết trước tên biến và kiểu của dữ liệu được chứa trong biến. Việc khai báo này sẽ làm chương trình quản lý các biến dễ dàng hơn như trong việc phân bố bộ nhớ cũng như quản lý các tính toán trên biến theo nguyên tắc: chỉ có các dữ liệu cùng kiểu với nhau mới được phép làm toán với nhau. Do đó, khi đề cập đến một kiểu chuẩn của một NNLT, thông thường chúng ta sẽ xét đến các yếu tố sau:

- tên kiểu: là một từ dành riêng để chỉ định kiểu của dữ liệu.
- số byte trong bộ nhớ để lưu trữ một đơn vị dữ liệu thuộc kiểu này: Thông thường số byte này phụ thuộc vào các trình biên dịch và hệ thống máy khác nhau, ở đây ta chỉ xét đến hệ thống máy PC thông dụng hiện nay.
- Miền giá trị của kiểu: Cho biết một đơn vị dữ liệu thuộc kiểu này sẽ có thể lấy

giá trị trong miền nào, ví dụ nhỏ nhất và lớn nhất là bao nhiêu. Hiển nhiên các giá trị này phụ thuộc vào số byte mà hệ thống máy qui định cho từng kiểu. NSD cần nhớ đến miền giá trị này để khai báo kiểu cho các biến cần sử dụng một cách thích hợp.

Dưới đây là bảng tóm tắt một số kiểu chuẩn đơn giản và các thông số của nó được sử dụng trong C++.

Loại dữ liệu	Tên kiểu	Số ô nhớ	Miền giá trị
Kí tự	char	1 byte	– 128 .. 127
	unsigned char	1 byte	0 .. 255
Số nguyên	int	2 byte	– 32768 .. 32767
	unsigned int	2 byte	0 .. 65535
	short	2 byte	– 32768 .. 32767
	long	4 byte	$-2^{15} .. 2^{15} - 1$
Số thực	float	4 byte	$\pm 10^{-37} . . \pm 10^{+38}$
	double	8 byte	$\pm 10^{-307} . . \pm 10^{+308}$

Bảng 1. Các loại kiểu đơn giản

Trong chương này chúng ta chỉ xét các loại kiểu đơn giản trên đây. Các loại kiểu có cấu trúc do người dùng định nghĩa sẽ được trình bày trong các chương sau.

2. Kiểu ký tự

Một kí tự là một kí hiệu trong bảng mã ASCII. Như đã biết một số kí tự có mặt chữ trên bàn phím (ví dụ các chữ cái, chữ số) trong khi một số kí tự lại không (ví dụ kí tự biểu diễn việc lùi lại một ô trong văn bản, kí tự chỉ việc kết thúc một dòng hay kết thúc một văn bản). Do vậy để biểu diễn một kí tự người ta dùng chính mã ASCII của kí tự đó trong bảng mã ASCII và thường gọi là giá trị của kí tự. Ví dụ phát biểu "Cho kí tự 'A'" là cũng tương đương với phát biểu "Cho kí tự 65" (65 là mã ASCII của kí tự 'A'), hoặc "Xoá kí tự xuống dòng" là cũng tương đương với phát biểu "Xoá kí tự 13" vì 13 là mã ASCII của kí tự xuống dòng.

Như vậy một biến kiểu kí tự có thể được nhận giá trị theo 2 cách tương đương - chữ hoặc giá trị số: ví dụ giả sử `c` là một biến kí tự thì câu lệnh gán `c = 'A'` cũng tương đương với câu lệnh gán `c = 65`. Tuy nhiên để sử dụng giá trị số của một kí tự `c` nào đó ta phải yêu cầu đổi `c` sang giá trị số bằng câu lệnh `int(c)`.

Theo bảng trên ta thấy có 2 loại kí tự là `char` với miền giá trị từ -128 đến 127 và

unsigned char (kí tự không dấu) với miền giá trị từ 0 đến 255. Trường hợp một biến được gán giá trị vượt ra ngoài miền giá trị của kiểu thì giá trị của biến sẽ được tính theo mã bù – (256 – c). Ví dụ nếu gán cho char c giá trị 179 (vượt khỏi miền giá trị đã được qui định của char) thì giá trị thực sự được lưu trong máy sẽ là – (256 – 179) = –77.

Ví dụ 1 :

```
char c, d ;                  // c, d được phép gán giá trị từ -128 đến 127
unsigned e ;                 // e được phép gán giá trị từ 0 đến 255
c = 65 ; d = 179 ;           // d có giá trị ngoài miền cho phép
e = 179; f = 330 ;           // f có giá trị ngoài miền cho phép
cout << c << int(c) ;        // in ra chữ cái 'A' và giá trị số 65
cout << d << int(d) ;        // in ra là kí tự '|' và giá trị số -77
cout << e << int(e)          // in ra là kí tự '|' và giá trị số 179
cout << f << int(f)          // in ra là kí tự 'J' và giá trị số 74
```

Chú ý: Qua ví dụ trên ta thấy một biến nếu được gán giá trị ngoài miền cho phép sẽ dẫn đến kết quả không theo suy nghĩ thông thường. Do vậy nên tuân thủ qui tắc chỉ gán giá trị cho biến thuộc miền giá trị mà kiểu của biến đó qui định. Ví dụ nếu muốn sử dụng biến có giá trị từ 128 .. 255 ta nên khai báo biến dưới dạng kí tự không dấu (unsigned char), còn nếu giá trị vượt quá 255 ta nên chuyển sang kiểu nguyên (int) chẳng hạn.

3. Kiểu số nguyên

Các số nguyên được phân chia thành 4 loại kiểu khác nhau với các miền giá trị tương ứng được cho trong bảng 1. Đó là kiểu số nguyên ngắn (short) tương đương với kiểu số nguyên (int) sử dụng 2 byte và số nguyên dài (long int) sử dụng 4 byte. Kiểu số nguyên thường được chia làm 2 loại có dấu (int) và không dấu (unsigned int hoặc có thể viết gọn hơn là unsigned). Qui tắc mã bù cũng được áp dụng nếu giá trị của biến vượt ra ngoài miền giá trị cho phép, vì vậy cần cân nhắc khi khai báo kiểu cho các biến. Ta thường sử dụng kiểu int cho các số nguyên trong các bài toán với miền giá trị vừa phải (có giá trị tuyệt đối bé hơn 32767), chẳng hạn các biến đếm trong các vòng lặp, ...

4. Kiểu số thực

Để sử dụng số thực ta cần khai báo kiểu float hoặc double mà miền giá trị của chúng được cho trong bảng 1. Các giá trị số kiểu double được gọi là số thực với độ chính xác gấp đôi vì với kiểu dữ liệu này máy tính có cách biểu diễn khác so với kiểu

float để đảm bảo số số lẻ sau một số thực có thể tăng lên đảm bảo tính chính xác cao hơn so với số kiểu float. Tuy nhiên, trong các bài toán thông dụng thường ngày độ chính xác của số kiểu float là đủ dùng.

Như đã nhắc đến trong phần các lệnh vào/ra ở chương 1, liên quan đến việc in ấn số thực ta có một vài cách thiết đặt dạng in theo ý muốn, ví dụ độ rộng tối thiểu để in một số hay số số lẻ thập phân cần in ...

Ví dụ 2 : Chương trình sau đây sẽ in diện tích và chu vi của một hình tròn có bán kính 2cm với 3 số lẻ.

```
#include <iostream.h>
#include <iomanip.h>
void main()
{
        float r = 2 ;                    // r là tên biến dùng để chứa bán kính
        cout << "Diện tích = " << setiosflags(ios::showpoint) ;
        cout << setprecision(3) << r * r * 3.1416 ;
        getch() ;
}
```

II. HẰNG - KHAI BÁO VÀ SỬ DỤNG HẰNG

Hằng là một giá trị cố định nào đó ví dụ 3 (hằng nguyên), 'A' (hằng kí tự), 5.0 (hằng thực), "Ha noi" (hằng xâu kí tự). Một giá trị có thể được hiểu dưới nhiều kiểu khác nhau, do vậy khi viết hằng ta cũng cần có dạng viết thích hợp.

1. Hằng nguyên

- kiểu short, int: 3, -7, ...
- kiểu unsigned: 3, 123456, ...
- kiểu long, long int: 3L, -7L, 123456L, ... (viết L vào cuối mỗi giá trị)

Các cách viết trên là thể hiện của số nguyên trong hệ thập phân, ngoài ra chúng còn được viết dưới các hệ đếm khác như hệ cơ số 8 hoặc hệ cơ số 16. Một số nguyên trong cơ số 8 luôn luôn được viết với số 0 ở đầu, tương tự với cơ số 16 phải viết với 0x ở đầu. Ví dụ ta biết 65 trong cơ số 8 là 101 và trong cơ số 16 là 41, do đó 3 cách viết 65, 0101, 0x41 là như nhau, cùng biểu diễn giá trị 65.

2. Hằng thực

Một số thực có thể được khai báo dưới dạng kiểu float hoặc double và các giá trị của nó có thể được viết dưới một trong hai dạng.

a. Dạng dấu phảy tĩnh

Theo cách viết thông thường. Ví dụ: 3.0, -7.0, 3.1416, ...

b. Dạng dấu phảy động

Tổng quát, một số thực x có thể được viết dưới dạng: men hoặc mEn, trong đó m được gọi là phần định trị, n gọi là phần bậc (hay mũ). Số men biểu thị giá trị $x = m \times 10^n$. Ví dụ số $\pi = 3.1416$ có thể được viết:

$\pi = \ldots = 0.031416e2 = 0.31416e1 = 3.1416e0 = 31.416e{-}1 = 314.16e{-}2 = \ldots$

vì $\pi = 0.031416 \times 10^2 = 0.31416 \times 10^1 = 3.1416 \times 10^0 = \ldots$

Như vậy một số x có thể được viết dưới dạng mEn với nhiều giá trị m, n khác nhau, phụ thuộc vào dấu phảy ngăn cách phần nguyên và phần thập phân của số. Do vậy cách viết này được gọi là dạng dấu phảy động.

3. Hằng kí tự

a. Cách viết hằng

Có 2 cách để viết một hằng kí tự. Đối với các kí tự có mặt chữ thể hiện ta thường sử dụng cách viết thông dụng đó là đặt mặt chữ đó giữa 2 dấu nháy đơn như: 'A', '3', ' ' (dấu cách) ... hoặc sử dụng trực tiếp giá trị số của chúng. Ví dụ các giá trị tương ứng của các kí tự trên là 65, 51 và 32. Với một số kí tự không có mặt chữ ta buộc phải dùng giá trị (số) của chúng, như viết 27 thay cho kí tự được nhấn bởi phím Escape, 13 thay cho kí tự được nhấn bởi phím Enter ...

Để biểu diễn kí tự bằng giá trị số ta có thể viết trực tiếp (không dùng cặp dấu nháy đơn) giá trị đó dưới dạng hệ số 10 (như trên) hoặc đặt chúng vào cặp dấu nháy đơn, trường hợp này chỉ dùng cho giá trị viết dưới dạng hệ 8 hoặc hệ 16 theo mẫu sau:

- '\kkk': không quá 3 chữ số trong hệ 8. Ví dụ '\11' biểu diễn kí tự có mã 9.
- '\xkk': không quá 2 chữ số trong hệ 16. Ví dụ '\x1B' biểu diễn kí tự có mã 27.

Tóm lại, một kí tự có thể có nhiều cách viết, chẳng hạn 'A' có giá trị là 65 (hệ 10) hoặc 101 (hệ 8) hoặc 41 (hệ 16), do đó kí tự 'A' có thể viết bởi một trong các dạng sau:

65, 0101, 0x41 hoặc 'A' , '\101' , '\x41'

Tương tự, dấu kết thúc xâu có giá trị 0 nên có thể viết bởi 0 hoặc '\0' hoặc '\x0', trong các cách này cách viết '\0' được dùng thông dụng nhất.

b. Một số hằng thông dụng

Đối với một số hằng kí tự thường dùng nhưng không có mặt chữ tương ứng, hoặc các kí tự được dành riêng với nhiệm vụ khác, khi đó thay vì phải nhớ giá trị của chúng ta có thể viết theo qui ước sau:

'\n'	:	biểu thị kí tự xuống dòng (cũng tương đương với endl)
'\t'	:	kí tự tab
'\a'	:	kí tự chuông (tức thay vì in kí tự, loa sẽ phát ra một tiếng 'bíp')
'\r'	:	xuống dòng
'\f'	:	kéo trang
'\\'	:	dấu \
'\?'	:	dấu chấm hỏi ?
'\''	:	dấu nháy đơn '
'\"'	:	dấu nháy kép "
'\kkk'	:	kí tự có mã là kkk trong hệ 8
'\xkk'	:	kí tự có mã là kk trong hệ 16

Ví dụ:

```
cout << "Hôm nay trời \t nắng \a \a \a \n" ;
```

sẽ in ra màn hình dòng chữ "Hôm nay trời" sau đó bỏ một khoảng cách bằng một tab (khoảng 8 dấu cách) rồi in tiếp chữ "nắng", tiếp theo phát ra 3 tiếng chuông và cuối cùng con trỏ trên màn hình sẽ nhảy xuống đầu dòng mới.

Do dấu cách (phím spacebar) không có mặt chữ, nên trong một số trường hợp để tránh nhầm lẫn chúng tôi qui ước sử dụng kí hiệu <> để biểu diễn dấu cách. Ví dụ trong giáo trình này dấu cách (có giá trị là 32) được viết ' ' (dấu nháy đơn bao một dấu cách) hoặc rõ ràng hơn bằng cách viết theo qui ước <>.

4. Hằng xâu kí tự

Là dãy kí tự bất kỳ đặt giữa cặp dấu nháy kép. Ví dụ: "Lớp K43*", "12A4", "A", "<>", "" là các hằng xâu kí tự, trong đó "" là xâu không chứa kí tự nào, các xâu "<>", "A" chứa 1 kí tự ... Số các kí tự giữa 2 dấu nháy kép được gọi là độ dài của xâu. Ví dụ xâu "" có độ dài 0, xâu "<>" hoặc "A" có độ dài 1 còn xâu "Lớp K43*" có độ dài 8.

Chú ý phân biệt giữa 2 cách viết 'A' và "A", tuy chúng cùng biểu diễn chữ cái A nhưng chương trình sẽ hiểu 'A' là một kí tự còn "A" là một xâu kí tự (do vậy chúng được bố trí khác nhau trong bộ nhớ cũng như cách sử dụng chúng là khác nhau). Tương tự ta không được viết " (2 dấu nháy đơn liền nhau) vì không có khái niệm kí tự

"rỗng". Để chỉ xâu rỗng (không có kí tự nào) ta phải viết "" (2 dấu nháy kép liền nhau).

Tóm lại một giá trị có thể được viết dưới nhiều kiểu dữ liệu khác nhau và do đó cách sử dụng chúng cũng khác nhau. Ví dụ liên quan đến khái niệm 3 đơn vị có thể có các cách viết sau tuy nhiên chúng hoàn toàn khác nhau:

- 3 : số nguyên 3 đơn vị
- 3L : số nguyên dài 3 đơn vị
- 3.0 : số thực 3 đơn vị
- '3' : chữ số 3
- "3" : xâu chứa kí tự duy nhất là 3

5. Khai báo hằng

Một giá trị cố định (hằng) được sử dụng nhiều lần trong chương trình đôi khi sẽ thuận lợi hơn nếu ta đặt cho nó một tên gọi, thao tác này được gọi là khai báo hằng. Ví dụ một chương trình quản lý sinh viên với giả thiết số sinh viên tối đa là 50. Nếu số sinh viên tối đa không thay đổi trong chương trình ta có thể đặt cho nó một tên gọi như SOSV chẳng hạn. Trong suốt chương trình bất kỳ chỗ nào xuất hiện giá trị 50 ta đều có thể thay nó bằng SOSV. Tương tự C++ cũng có những tên hằng được đặt sẵn, được gọi là các hằng chuẩn và NSD có thể sử dụng khi cần thiết. Ví dụ hằng π được đặt sẵn trong C++ với tên gọi M_PI. Việc sử dụng tên hằng thay cho hằng có nhiều điểm thuận lợi như sau:

- Chương trình dễ đọc hơn, vì thay cho các con số ít có ý nghĩa, một tên gọi sẽ làm NSD dễ hình dung vai trò, nội dung của nó. Ví dụ, khi gặp tên gọi SOSV NSD sẽ hình dung được chẳng hạn, "đây là số sinh viên tối đa trong một lớp", trong khi số 50 có thể là số sinh viên mà cũng có thể là tuổi của một sinh viên nào đó.
- Chương trình dễ sửa chữa hơn, ví dụ bây giờ nếu muốn thay đổi chương trình sao cho bài toán quản lý được thực hiện với số sinh viên tối đa là 60, khi đó ta cần tìm và thay thế hàng trăm vị trí xuất hiện của 50 thành 60. Việc thay thế như vậy dễ gây ra lỗi vì có thể không tìm thấy hết các số 50 trong chương trình hoặc thay nhầm số 50 với ý nghĩa khác như tuổi của một sinh viên nào đó chẳng hạn. Nếu trong chương trình sử dụng hằng SOSV, bây giờ việc thay thế trở nên chính xác và dễ dàng hơn bằng thao tác khai báo lại giá trị hằng SOSV bằng 60. Lúc đó trong chương trình bất kỳ nơi nào gặp tên hằng SOSV đều được chương trình hiểu với giá trị 60.

Để khai báo hằng ta dùng các câu khai báo sau:

#define tên_hằng giá_trị_hằng ;

hoặc:

const tên_hằng = giá_trị_hằng ;

Ví dụ:

```
#define sosv 50 ;
#define MAX 100 ;
const sosv = 50 ;
```

Như trên đã chú ý một giá trị hằng chưa nói lên kiểu sử dụng của nó vì vậy ta cần khai báo rõ ràng hơn bằng cách thêm tên kiểu trước tên hằng trong khai báo const, các hằng khai báo như vậy được gọi là hằng có kiểu.

Ví dụ:

```
const int sosv = 50 ;
const float nhiet_do_soi = 100.0 ;
```

III. BIẾN - KHAI BÁO VÀ SỬ DỤNG BIẾN

1. Khai báo biến

Biến là các tên gọi để lưu giá trị khi làm việc trong chương trình. Các giá trị được lưu có thể là các giá trị dữ liệu ban đầu, các giá trị trung gian tạm thời trong quá trình tính toán hoặc các giá trị kết quả cuối cùng. Khác với hằng, giá trị của biến có thể thay đổi trong quá trình làm việc bằng các lệnh đọc vào từ bàn phím hoặc gán. Hình ảnh cụ thể của biến là một số ô nhớ trong bộ nhớ được sử dụng để lưu các giá trị của biến.

Mọi biến phải được khai báo trước khi sử dụng. Một khai báo như vậy sẽ báo cho chương trình biết về một biến mới gồm có: tên của biến, kiểu của biến (tức kiểu của giá trị dữ liệu mà biến sẽ lưu giữ). Thông thường với nhiều NNLT tất cả các biến phải được khai báo ngay từ đầu chương trình hay đầu của hàm, tuy nhiên để thuận tiện C++ cho phép khai báo biến ngay bên trong chương trình hoặc hàm, có nghĩa bất kỳ lúc nào NSD thấy cần thiết sử dụng biến mới, họ có quyền khai báo và sử dụng nó từ đó trở đi.

Cú pháp khai báo biến gồm tên kiểu, tên biến và có thể có hay không khởi tạo giá trị ban đầu cho biến. Để khởi tạo hoặc thay đổi giá trị của biến ta dùng lệnh gán (=).

a. Khai báo không khởi tạo

```
tên_kiểu tên_biến_1 ;
tên_kiểu tên_biến_2 ;
```

```
tên_kiểu  tên_biến_3 ;
```

Nhiều biến cùng kiểu có thể được khai báo trên cùng một dòng:

```
tên_kiểu  tên_biến_1, tên_biến_2, tên_biến_3 ;
```

Ví dụ:

```
void main()
{
    int i, j ;                  // khai báo 2 biến i, j có kiểu nguyên
    float x ;                   // khai báo biến thực x
    char c, d[100] ;            // biến kí tự c, xâu d chứa tối đa 100 kí tự
    unsigned int u ;            // biến nguyên không dấu u
    …
}
```

b. Khai báo có khởi tạo

Trong câu lệnh khai báo, các biến có thể được gán ngay giá trị ban đầu bởi phép toán gán (=) theo cú pháp:

tên_kiểu tên_biến_1 = gt_1, tên_biến_2 = gt_2, tên_biến_3 = gt_3 ;

trong đó các giá trị gt_1, gt_2, gt_3 có thể là các hằng, biến hoặc biểu thức.

Ví dụ:

```
const int n = 10 ;
void main()
{
    int i = 2, j , k = n + 5;       // khai báo i và khởi tạo bằng 2, k bằng 15
    float eps = 1.0e-6 ;            // khai báo biến thực epsilon khởi tạo bằng 10^-6
    char c = 'Z';                   // khai báo biến kí tự c và khởi tạo bằng 'A'
    char d[100] = "Tin học";        // khai báo xâu kí tự d chứa dòng chữ "Tin học"
    …
}
```

2. Phạm vi của biến

Như đã biết chương trình là một tập hợp các hàm, các câu lệnh cũng như các khai báo. Phạm vi tác dụng của một biến là nơi mà biến có tác dụng, tức hàm nào, câu lệnh

nào được phép sử dụng biến đó. Một biến xuất hiện trong chương trình có thể được sử dụng bởi hàm này nhưng không được bởi hàm khác hoặc bởi cả hai, điều này phụ thuộc chặt chẽ vào vị trí nơi biến được khai báo. Một nguyên tắc đầu tiên là biến sẽ có tác dụng kể từ vị trí nó được khai báo cho đến hết khối lệnh chứa nó. Chi tiết cụ thể hơn sẽ được trình bày trong chương 4 khi nói về hàm trong C++.

3. Gán giá trị cho biến (phép gán)

Trong các ví dụ trước chúng ta đã sử dụng phép gán dù nó chưa được trình bày, đơn giản một phép gán mang ý nghĩa tạo giá trị mới cho một biến. Khi biến được gán giá trị mới, giá trị cũ sẽ được tự động xoá đi bất kể trước đó nó chứa giá trị nào (hoặc chưa có giá trị, ví dụ chỉ mới vừa khai báo xong). Cú pháp của phép gán như sau:

tên_biến = biểu thức ;

Khi gặp phép gán chương trình sẽ tính toán giá trị của biểu thức sau đó gán giá trị này cho biến. Ví dụ:

```
int n, i = 3;                    // khởi tạo i bằng 3
n = 10;                          // gán cho n giá trị 10
cout << n <<", " << i << endl;   // in ra: 10, 3
i = n / 2;                       // gán lại giá trị của i bằng n/2 = 5
cout << n <<", " << i << endl;   // in ra: 10, 5
```

Trong ví dụ trên n được gán giá trị bằng 10; trong câu lệnh tiếp theo biểu thức n/2 được tính (bằng 5) và sau đó gán kết quả cho biến i, tức i nhận kết quả bằng 5 dù trước đó nó đã có giá trị là 2 (trong trường hợp này việc khởi tạo giá trị 2 cho biến i là không có ý nghĩa).

Một khai báo có khởi tạo cũng tương đương với một khai báo và sau đó thêm lệnh gán cho biến (ví dụ int i = 3 cũng tương đương với 2 câu lệnh int i; i = 3) tuy nhiên về mặt bản chất khởi tạo giá trị cho biến vẫn khác với phép toán gán như ta sẽ thấy trong các phần sau.

4. Một số điểm lưu ý về phép gán

Với ý nghĩa thông thường của phép toán (nghĩa là tính toán và cho lại một giá trị) thì phép toán gán còn một nhiệm vụ nữa là trả lại một giá trị. Giá trị trả lại của phép toán gán chính là giá trị của biểu thức sau dấu bằng. Lợi dụng điều này C++ cho phép chúng ta gán "kép" cho nhiều biến nhận cùng một giá trị bởi cú pháp:

biến_1 = biến_2 = … = biến_n = gt ;

với cách gán này tất cả các biến sẽ nhận cùng giá trị gt. Ví dụ:

```
int i, j, k ;
i = j = k = 1;
```

Biểu thức gán trên có thể được viết lại như (i = (j = (k = 1))), có nghĩa đầu tiên để thực hiện phép toán gán giá trị cho biến i chương trình phải tính biểu thức (j = (k = 1)), tức phải tính k = 1, đây là phép toán gán, gán giá trị 1 cho k và trả lại giá trị 1, giá trị trả lại này sẽ được gán cho j và trả lại giá trị 1 để tiếp tục gán cho i.

Ngoài việc gán kép như trên, phép toán gán còn được phép xuất hiện trong bất kỳ biểu thức nào, điều này cho phép trong một biểu thức có phép toán gán, nó không chỉ tính toán mà còn gán giá trị cho các biến, ví dụ n = 3 + (i = 2) sẽ cho ta i = 2 và n = 5. Việc sử dụng nhiều chức năng của một câu lệnh làm cho chương trình gọn gàng hơn (trong một số trường hợp) nhưng cũng trở nên khó đọc, chẳng hạn câu lệnh trên có thể viết tách thành 2 câu lệnh i = 2; n = 3 + i; sẽ dễ đọc hơn ít nhất đối với các bạn mới bắt đầu tìm hiểu về lập trình.

IV. PHÉP TOÁN, BIỂU THỨC VÀ CÂU LỆNH

1. Phép toán

C++ có rất nhiều phép toán loại 1 ngôi, 2 ngôi và thậm chí cả 3 ngôi. Để hệ thống, chúng tôi tạm phân chia thành các lớp và trình bày chỉ một số trong chúng. Các phép toán còn lại sẽ được tìm hiểu dần trong các phần sau của giáo trình. Các thành phần tên gọi tham gia trong phép toán được gọi là hạng thức hoặc toán hạng, các kí hiệu phép toán được gọi là toán tử. Ví dụ trong phép toán a + b; a, b được gọi là toán hạng và + là toán tử. Phép toán 1 ngôi là phép toán chỉ có một toán hạng, ví dụ –a (đổi dấu số a), &x (lấy địa chỉ của biến x) … Một số kí hiệu phép toán cũng được sử dụng chung cho cả 1 ngôi lẫn 2 ngôi (hiển nhiên với ngữ nghĩa khác nhau), ví dụ kí hiệu – được sử dụng cho phép toán trừ 2 ngôi a – b, hoặc phép & còn được sử dụng cho phép toán lấy hội các bit (a & b) của 2 số nguyên a và b …

*a. Các phép toán số học: +, -, *, /, %*

- Các phép toán + (cộng), – (trừ), * (nhân) được hiểu theo nghĩa thông thường trong số học.
- Phép toán a / b (chia) được thực hiện theo kiểu của các toán hạng, tức nếu cả hai toán hạng là số nguyên thì kết quả của phép chia chỉ lấy phần nguyên, ngược lại nếu 1 trong 2 toán hạng là thực thì kết quả là số thực. Ví dụ:

```
13/5 = 2                              // do 13 và 5 là 2 số nguyên
13.0/5 = 13/5.0 = 13.0/5.0 = 2.6      // do có ít nhất 1 toán hạng là thực
```

– Phép toán a % b (lấy phần dư) trả lại phần dư của phép chia a/b, trong đó a và b là 2 số nguyên. Ví dụ:

```
13%5 = 3                    // phần dư của 13/5
5%13 = 5                    // phần dư của 5/13
```

b. Các phép toán tự tăng, giảm: i++, ++i, i--, --i

– Phép toán ++i và i++ sẽ cùng tăng i lên 1 đơn vị tức tương đương với câu lệnh i = i+1. Tuy nhiên nếu 2 phép toán này nằm trong câu lệnh hoặc biểu thức thì ++i khác với i++. Cụ thể ++i sẽ tăng i, sau đó i mới được tham gia vào tính toán trong biểu thức. Ngược lại i++ sẽ tăng i sau khi biểu thức được tính toán xong (với giá trị i cũ). Điểm khác biệt này được minh hoạ thông qua ví dụ sau, giả sử i = 3, j = 15.

Phép toán	Tương đương	Kết quả
i = ++j ; // tăng trước	j = j + 1 ; i = j ;	i = 16 , j = 16
i = j++ ; // tăng sau	i = j ; j = j + 1 ;	i = 15 , j = 16
j = ++i + 5 ;	i = i + 1 ; j = i + 5 ;	i = 4, j = 9
j = i++ + 5 ;	j = i + 5; i = i + 1;	i = 4, j = 8

Ghi chú: Việc kết hợp phép toán tự tăng giảm vào trong biểu thức hoặc câu lệnh (như ví dụ trong phần sau) sẽ làm chương trình gọn nhưng khó hiểu hơn.

c. Các phép toán so sánh và lôgic

Đây là các phép toán mà giá trị trả lại là đúng hoặc sai. Nếu giá trị của biểu thức là đúng thì nó nhận giá trị 1, ngược lại là sai thì biểu thức nhận giá trị 0. Nói cách khác 1 và 0 là giá trị cụ thể của 2 khái niệm "đúng", "sai". Mở rộng hơn C++ quan niệm một giá trị bất kỳ khác 0 là "đúng" và giá trị 0 là "sai".

- Các phép toán so sánh

  == (bằng nhau), != (khác nhau), > (lớn hơn), < (nhỏ hơn), >= (lớn hơn hoặc bằng), <= (nhỏ hơn hoặc bằng).

Hai toán hạng của các phép toán này phải cùng kiểu. Ví dụ:

```
3 == 3 hoặc 3 == (4 -1)     // nhận giá trị 1 vì đúng
3 == 5                      // = 0 vì sai
3 != 5                      // = 1
3 + (5 < 2)                 // = 3 vì 5<2 bằng 0
```

```
3 + (5 >= 2)                    // = 4 vì 5>=2 bằng 1
```

Chú ý: cần phân biệt phép toán gán (=) và phép toán so sánh (==). Phép gán vừa gán giá trị cho biến vừa trả lại giá trị bất kỳ (là giá trị của toán hạng bên phải), trong khi phép so sánh luôn luôn trả lại giá trị 1 hoặc 0.

- Các phép toán lôgic:

  && (và), || (hoặc), ! (không, phủ định)

Hai toán hạng của loại phép toán này phải có kiểu lôgic tức chỉ nhận một trong hai giá trị "đúng" (được thể hiện bởi các số nguyên khác 0) hoặc "sai" (thể hiện bởi 0). Khi đó giá trị trả lại của phép toán là 1 hoặc 0 và được cho trong bảng sau:

a	b	a && b	a || b	! a
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Tóm lại:

– Phép toán "và" đúng khi và chỉ khi hai toán hạng cùng đúng

– Phép toán "hoặc" sai khi và chỉ khi hai toán hạng cùng sai

– Phép toán "không" (hoặc "phủ định") đúng khi và chỉ khi toán hạng của nó sai.

Ví dụ:

```
3 && (4 > 5)            // = 0 vì có hạng thức (4>5) sai
(3 >= 1) && (7)         // = 1 vì cả hai hạng thức cùng đúng
!1                      // = 0
! (4 + 3 < 7)           // = 1 vì (4+3<7) bằng 0
5 || (4 >= 6)           // = 1 vì có một hạng thức (5) đúng
(5 < !0) || (4 >= 6)    // = 0 vì cả hai hạng thức đều sai
```

Chú ý: việc đánh giá biểu thức được tiến hành từ trái sang phải và sẽ dừng khi biết kết quả mà không chờ đánh giá hết biểu thức. Cách đánh giá này sẽ cho những kết quả phụ khác nhau nếu trong biểu thức ta "tranh thủ" đưa thêm vào các phép toán tự tăng giảm. Ví dụ cho i = 2, j = 3, xét 2 biểu thức sau đây:

```
x = (++i < 4 && ++j > 5)        cho kết quả x = 0 , i = 3 , j = 4
```

```
y = (++j > 5 && ++i < 4)          cho kết quả y = 0 , i = 2 , j = 4
```

cách viết hai biểu thức là như nhau (ngoại trừ hoán đổi vị trí 2 toán hạng của phép toán &&). Với giả thiết i = 2 và j = 3 ta thấy cả hai biểu thức trên cùng nhận giá trị 0. Tuy nhiên các giá trị của i và j sau khi thực hiện xong hai biểu thức này sẽ có kết quả khác nhau. Cụ thể với biểu thức đầu vì ++i < 4 là đúng nên chương trình phải tiếp tục tính tiếp ++j > 5 để đánh giá được biểu thức. Do vậy sau khi đánh giá xong cả i và j đều được tăng 1 (i=3, j=4). Trong khi đó với biểu thức sau do ++j > 5 là sai nên chương trình có thể kết luận được toàn bộ biểu thức là sai mà không cần tính tiếp ++i < 4. Có nghĩa chương trình sau khi đánh giá xong ++j > 5 sẽ dừng và vì vậy chỉ có biến j được tăng 1, từ đó ta có i = 2, j = 4 khác với kết quả của biểu thức trên. Ví dụ này một lần nữa nhắc ta chú ý kiểm soát kỹ việc sử dụng các phép toán tự tăng giảm trong biểu thức và trong câu lệnh.

2. Các phép gán

- Phép gán thông thường: Đây là phép gán đã được trình bày trong mục trước.
- Phép gán có điều kiện:

biến = (điều_kiện) ? a: b ;

điều_kiện là một biểu thức logic, a, b là các biểu thức bất kỳ cùng kiểu với kiểu của biến. Phép toán này gán giá trị a cho biến nếu điều kiện đúng và b nếu ngược lại.

Ví dụ:

```
x = (3 + 4 < 7) ? 10: 20      // x = 20 vì 3+4<7 là sai
x = (3 + 4) ? 10: 20          // x = 10 vì 3+4 khác 0, tức điều kiện đúng
x = (a > b) ? a: b            // x = số lớn nhất trong 2 số a, b.
```

- Cách viết gọn của phép gán: Một phép gán dạng x = x @ a ; có thể được viết gọn dưới dạng x @= a trong đó @ là các phép toán số học, xử lý bit ... Ví dụ:

  thay cho viết x = x + 2 có thể viết x += 2;

  hoặc x = x/2 ; x = x*2 có thể được viết lại như x /= 2; x *= 2;

Cách viết gọn này có nhiều thuận lợi khi viết và đọc chương trình nhất là khi tên biến quá dài hoặc đi kèm nhiều chỉ số … thay vì phải viết hai lần tên biến trong câu lệnh thì chỉ phải viết một lần, điều này tránh viết lặp lại tên biến dễ gây ra sai sót. Ví dụ thay vì viết:

```
ngay_quoc_te_lao_dong = ngay_quoc_te_lao_dong + 365;
```

có thể viết gọn hơn bởi:

```
ngay_quoc_te_lao_dong += 365;
```

hoặc thay cho viết :

```
Luong[Nhanvien[3][2*i+1]] = Luong[Nhanvien[3][2*i+1]] * 290 ;
```

có thể được viết lại bởi:

```
Luong[Nhanvien[3][2*i+1]] *= 290;
```

3. Biểu thức

Biểu thức là dãy kí hiệu kết hợp giữa các toán hạng, phép toán và cặp dấu () theo một qui tắc nhất định. Các toán hạng là hằng, biến, hàm. Biểu thức cung cấp một cách thức để tính giá trị mới dựa trên các toán hạng và toán tử trong biểu thức. Ví dụ:

```
(x + y) * 2 - 4 ;       3 - x + sqrt(y) ;  (-b + sqrt(delta)) / (2*a) ;
```

a. Thứ tự ưu tiên của các phép toán

Để tính giá trị của một biểu thức cần có một trật tự tính toán cụ thể và thống nhất. Ví dụ xét biểu thức x = 3 + 4 * 2 + 7

- nếu tính theo đúng trật tự từ trái sang phải, ta có x = ((3+4) * 2) + 7 = 21,
- nếu ưu tiên dấu + được thực hiện trước dấu *, x = (3 + 4) * (2 + 7) = 63,
- nếu ưu tiên dấu * được thực hiện trước dấu +, x = 3 + (4 * 2) + 7 = 18.

Như vậy cùng một biểu thức tính x nhưng cho 3 kết quả khác nhau theo những cách hiểu khác nhau. Vì vậy cần có một cách hiểu thống nhất dựa trên thứ tự ưu tiên của các phép toán, tức những phép toán nào sẽ được ưu tiên tính trước và những phép toán nào được tính sau ...

C++ qui định trật tự tính toán theo các mức độ ưu tiên như sau:

1. Các biểu thức trong cặp dấu ngoặc ()
2. Các phép toán 1 ngôi (tự tăng, giảm, lấy địa chỉ, lấy nội dung con trỏ …)
3. Các phép toán số học.
4. Các phép toán quan hệ, logic.
5. Các phép gán.

Nếu có nhiều cặp ngoặc lồng nhau thì cặp trong cùng (sâu nhất) được tính trước. Các phép toán trong cùng một lớp có độ ưu tiên theo thứ tự: lớp nhân (*, /, &&), lớp cộng (+, –, ||). Nếu các phép toán có cùng thứ tự ưu tiên thì chương trình sẽ thực hiện từ trái sang phải. Các phép gán có độ ưu tiên cuối cùng và được thực hiện từ phải sang trái. Ví dụ. theo mức ưu tiên đã qui định, biểu thức tính x trong ví dụ trên sẽ được tính như x = 3 + (4 * 2) + 7 = 18.

Phần lớn các trường hợp muốn tính toán theo một trật tự nào đó ta nên sử dụng cụ thể các dấu ngoặc (vì các biểu thức trong dấu ngoặc được tính trước). Ví dụ:

- Để tính $\Delta = b^2 - 4ac$ ta viết delta = b * b – 4 * a * c ;
- Để tính nghiệm phương trình bậc 2: $x = \frac{-b+\sqrt{\Delta}}{2a}$ viết : x = –b + sqrt(delta) / 2*a; là sai vì theo mức độ ưu tiên x sẽ được tính như –b + ((sqrt(delta)/2) * a) (thứ tự tính sẽ là phép toán 1 ngôi đổi dấu –b, đến phép chia, phép nhân và cuối cùng là phép cộng). Để tính chính xác cần phải viết (–b + sqrt(delta)) / (2*a).
- Cho a = 1, b = 2, c = 3. Biểu thức a += b += c cho giá trị c = 3, b = 5, a = 6. Thứ tự tính sẽ là từ phải sang trái, tức câu lệnh trên tương đương với các câu lệnh sau:

```
a = 1 ; b = 2 ; c = 3 ;
b = b + c ;                    // b = 5
a = a + b ;                    // a = 6
```

Để rõ ràng, tốt nhất nên viết biểu thức cần tính trước trong các dấu ngoặc.

b. Phép chuyển đổi kiểu

Khi tính toán một biểu thức phần lớn các phép toán đều yêu cầu các toán hạng phải cùng kiểu. Ví dụ để phép gán thực hiện được thì giá trị của biểu thức phải có **cùng kiểu** với biến. Trong trường hợp kiểu của giá trị biểu thức khác với kiểu của phép gán thì hoặc là chương trình sẽ tự động chuyển kiểu giá trị biểu thức về thành kiểu của biến được gán (nếu được) hoặc sẽ báo lỗi. Do vậy khi cần thiết NSD phải sử dụng các câu lệnh để chuyển kiểu của biểu thức cho phù hợp với kiểu của biến.

- Chuyển kiểu tự động: về mặt nguyên tắc, khi cần thiết các kiểu có giá trị thấp sẽ được chương trình tự động chuyển lên kiểu cao hơn cho phù hợp với phép toán. Cụ thể phép chuyển kiểu có thể được thực hiện theo sơ đồ như sau:

  char $\leftrightarrow$ int $\rightarrow$ long int $\rightarrow$ float $\rightarrow$ double

Ví dụ:

```
int i = 3;
float f ;
f = i + 2;
```

trong ví dụ trên i có kiểu nguyên và vì vậy i+2 cũng có kiểu nguyên trong khi f có kiểu thực. Tuy vậy phép toán gán này là hợp lệ vì chương trình sẽ tự động chuyển kiểu

cuả i+2 (bằng 5) sang kiểu thực (bằng 5.0) rồi mới gán cho f.

– Ép kiểu: trong chuyển kiểu tự động, chương trình chuyển các kiểu từ thấp đến cao, tuy nhiên chiều ngược lại không thể thực hiện được vì nó có thể gây mất dữ liệu. Do đó nếu cần thiết NSD phải ra lệnh cho chương trình. Ví dụ:

```
int i;
float f = 3 ;          // tự động chuyển 3 thành 3.0 và gán cho f
i = f + 2 ;            // sai vì mặc dù f + 2 = 5 nhưng không gán được cho i
```

Trong ví dụ trên để câu lệnh i = f+2 thực hiện được ta phải ép kiểu của biểu thức f+2 về thành kiểu nguyên. Cú pháp tổng quát như sau:

```
(tên_kiểu)biểu_thức          // cú pháp cũ trong C
```

hoặc:

```
tên_kiểu(biểu_thức)          // cú pháp mới trong C++
```

trong đó tên_kiểu là kiểu cần được chuyển sang. Như vậy câu lệnh trên phải được viết lại:

```
i = int(f + 2) ;
```

khi đó f+2 (bằng 5.0) được chuyển thành 5 và gán cho i.

Dưới đây ta sẽ xét một số ví dụ về lợi ích của việc ép kiểu.

- Phép ép kiểu từ một số thực về số nguyên sẽ cắt bỏ tất cả phần thập phân của số thực, chỉ để lại phần nguyên. Như vậy để tính phần nguyên của một số thực x ta chỉ cần ép kiểu của x về thành kiểu nguyên, có nghĩa int(x) là phần nguyên của số thực x bất kỳ. Ví dụ để kiểm tra một số nguyên n có phải là số chính phương, ta cần tính căn bậc hai của n. Nếu căn bậc hai x của n là số nguyên thì n là số chính phương, tức nếu int(x) = x thì x nguyên và n là chính phương, ví dụ:

```
int n = 10 ;
float x = sqrt(n) ;          // hàm sqrt(n) trả lại căn bậc hai của số n
if (int(x) == x) cout << "n chính phương" ;
else cout << "n không chính phương" ;
```

- Để biết mã ASCII của một kí tự ta chỉ cần chuyển kí tự đó sang kiểu nguyên.

```
char c ;
cin >> c ;
cout << "Mã của kí tự vừa nhập là " << int(c) ;
```

Ghi chú: Xét ví dụ sau:

```
int i = 3 , j = 5 ;
float x ;
x = i / j * 10;              // x = 6 ?
cout << x ;
```

trong ví dụ này mặc dù x được khai báo là thực nhưng kết quả in ra sẽ là 0 thay vì 6 như mong muốn. Lý do là vì phép chia giữa 2 số nguyên i và j sẽ cho lại số nguyên, tức i/j = 3/5 = 0. Từ đó x = 0*10 = 0. Để phép chia ra kết quả thực ta cần phải ép kiểu hoặc i hoặc j hoặc cả 2 thành số thực, khi đó phép chia sẽ cho kết quả thực và x được tính đúng giá trị. Cụ thể câu lệnh x = i/j*10 được đổi thành:

```
x = float(i) / j * 10 ;              // đúng
x = i / float(j) * 10      ;         // đúng
x = float(i) / float(j) * 10 ;       // đúng
x = float(i/j) * 10 ;                // sai
```

Phép ép kiểu: x = float(i/j) * 10 ; vẫn cho kết quả sai vì trong dấu ngoặc phép chia i/j vẫn là phép chia nguyên, kết quả x vẫn là 0.

4. Câu lệnh và khối lệnh

Một **câu lệnh** trong C++ được thiết lập từ các từ khoá và các biểu thức … và luôn luôn được kết thúc bằng dấu chấm phẩy. Các ví dụ vào/ra hoặc các phép gán tạo thành những câu lệnh đơn giản như:

```
cin >> x >> y ;
x = 3 + x ; y = (x = sqrt(x)) + 1 ;
cout << x ;
cout << y ;
```

Các câu lệnh được phép viết trên cùng một hoặc nhiều dòng. Một số câu lệnh được gọi là lệnh có cấu trúc, tức bên trong nó lại chứa dãy lệnh khác. Dãy lệnh này phải được bao giữa cặp dấu ngoặc {} và được gọi là *khối lệnh*. Ví dụ tất cả các lệnh trong một hàm (như hàm main()) luôn luôn là một khối lệnh. Một đặc điểm của khối lệnh là các biến được khai báo trong khối lệnh nào thì chỉ có tác dụng trong khối lệnh đó. Chi tiết hơn về các đặc điểm của lệnh và khối lệnh sẽ được trình bày trong các chương tiếp theo của giáo trình.

V. THƯ VIỆN CÁC HÀM TOÁN HỌC

Trong phần này chúng tôi tóm tắt một số các hàm toán học hay dùng. Các hàm này đều được khai báo trong file nguyên mẫu math.h.

1. Các hàm số học

- abs(x), labs(x), fabs(x) : trả lại giá trị tuyệt đối của một số nguyên, số nguyên dài và số thực.
- pow(x, y) : hàm mũ, trả lại giá trị x lũy thừa y (x^y).
- exp(x) : hàm mũ, trả lại giá trị e mũ x (e^x).
- log(x), log10(x) : trả lại lôgarit cơ số e và lôgarit thập phân của x (lnx, logx) .
- sqrt(x) : trả lại căn bậc 2 của x.
- atof(s_number) : trả lại số thực ứng với số viết dưới dạng xâu kí tự s_number.

2. Các hàm lượng giác

- sin(x), cos(x), tan(x) : trả lại các giá trị sinx, cosx, tgx.

BÀI TẬP

1. Viết câu lệnh khai báo biến để lưu các giá trị sau:

- Tuổi của một người
- Số lượng cây trong thành phố
- Độ dài cạnh một tam giác
- Khoảng cách giữa các hành tinh
- Một chữ số
- Nghiệm x của phương trình bậc 1
- Một chữ cái
- Biệt thức Δ của phương trình bậc 2

2. Viết câu lệnh nhập vào 4 giá trị lần lượt là số thực, nguyên, nguyên dài và kí tự. In ra màn hình các giá trị này để kiểm tra.

3. Viết câu lệnh in ra màn hình các dòng sau (không kể các số thứ tự và dấu: ở đầu mỗi dòng)

1: Bộ Giáo dục và Đào tạo Cộng hoà xã hội chủ nghĩa Việt Nam

2:

3: Sở Giáo dục Hà Nội Độc lập - Tự do - Hạnh phúc

Chú ý: khoảng trống giữa chữ Đào tạo và Cộng hoà (dòng 1) là 2 tab. Dòng 2: để trống.

4. Viết chương trình nhập vào một kí tự. In ra kí tự đó và mã ascii của nó.

5. Viết chương trình nhập vào hai số thực. In ra hai số thực đó với 2 số lẻ và cách nhau 5 cột.

6. Nhập, chạy và giải thích kết quả đạt được của đoạn chương trình sau:

```
#include <iostream.h>
void main()
{
    char c1 = 200; unsigned char c2 = 200 ;
    cout << "c1 = " << c1 << ", c2 = " << c2 << "\n" ;
    cout << "c1+100 = " << c1+100 << ", c2+100 = " << c2+100 ;
}
```

7. Nhập a, b, c. In ra màn hình dòng chữ phương trình có dạng ax^2 + bx + c = 0, trong đó các giá trị a, b, c chỉ in 2 số lẻ (ví dụ với a = 5.141, b = –2, c = 0.8 in ra 5.14 x^2 –2.00 x + 0.80).

8. Viết chương trình tính và in ra giá trị các biểu thức sau với 2 số lẻ:

a. $\sqrt{3+\sqrt{3+\sqrt{3}}}$ b. $\dfrac{1}{2+\dfrac{1}{2+\dfrac{1}{2}}}$

9. Nhập a, b, c là các số thực. In ra giá trị của các biểu thức sau với 3 số lẻ:

a. $a^2 - 2b + ab/c$ c. $3a - b^3 - 2\sqrt{c}$

b. $\dfrac{b^2-4ac}{2a}$ d. $\sqrt{a^2/b-4a/bc+1}$

10. In ra tổng, tích, hiệu và thương của 2 số được nhập vào từ bàn phím.

11. In ra trung bình cộng, trung bình nhân của 3 số được nhập vào từ bàn phím.

12. Viết chương trình nhập cạnh, bán kính và in ra diện tích, chu vi của các hình: vuông, chữ nhật, tròn.

13. Nhập a, b, c là độ dài 3 cạnh của tam giác (chú ý đảm bảo tổng 2 cạnh phải lớn

hơn cạnh còn lại). Tính chu vi, diện tích, độ dài 3 đường cao, 3 đường trung tuyến, 3 đường phân giác, bán kính đường tròn nội tiếp, ngoại tiếp lần lượt theo các công thức sau:

C = 2p = a + b + c ; S = $\sqrt{p(p-a)(p-b)(p-c)}$;

$h_a = \frac{2S}{a}$; ma = $\frac{1}{2}\sqrt{2b^2+2c^2-a^2}$; ga = $\frac{2}{b+c}\sqrt{bcp(p-a)}$;

$r = \frac{S}{p}$; $R = \frac{abc}{4S}$;

14. Tính diện tích và thể tích của hình cầu bán kính R theo công thức:

S = $4\pi R^2$; V = RS/3

15. Nhập vào 4 chữ số. In ra tổng của 4 chữ số này và chữ số hàng chục, hàng đơn vị của tổng (ví dụ 4 chữ số 3, 1, 8, 5 có tổng là 17 và chữ số hàng chục là 1 và hàng đơn vị là 7, cần in ra 17, 1, 7).

16. Nhập vào một số nguyên (có 4 chữ số). In ra tổng của 4 chữ số này và chữ số đầu, chữ số cuối (ví dụ số 3185 có tổng các chữ số là 17, đầu và cuối là 3 và 5, kết quả in ra là: 17, 3, 5).

17. Hãy nhập 2 số a và b. Viết chương trình đổi giá trị của a và b theo 2 cách:

- dùng biến phụ t: t = a; a = b; b = t;
- không dùng biến phụ: a = a + b; b = a - b; a = a - b;

In kết quả ra màn hình để kiểm tra.

18. Viết chương trình đoán số của người chơi đang nghĩ, bằng cách yêu cầu người chơi nghĩ một số, sau đó thực hiện một loạt các tính toán trên số đã nghĩ rồi cho biết kết quả. Máy sẽ in ra số mà người chơi đã nghĩ. (ví dụ yêu cầu người chơi lấy số đã nghĩ nhân đôi, trừ 4, bình phương, chia 2 và trừ 7 rồi cho biết kết quả, máy sẽ in ra số người chơi đã nghĩ).

19. Một sinh viên gồm có các thông tin: họ tên, tuổi, điểm toán (hệ số 2), điểm tin (hệ số 1). Hãy nhập các thông tin trên cho 2 sinh viên. In ra bảng điểm gồm các chi tiết nêu trên và điểm trung bình của mỗi sinh viên.

20. Một nhân viên gồm có các thông tin: họ tên, hệ số lương, phần trăm phụ cấp (theo lương) và phần trăm phải đóng BHXH. Hãy nhập các thông tin trên cho 2 nhân viên. In ra bảng lương gồm các chi tiết nêu trên và tổng số tiền cuối cùng mỗi nhân viên được nhận.

CHƯƠNG 3

CẤU TRÚC ĐIỀU KHIỂN VÀ DỮ LIỆU KIỂU MẢNG

Cấu trúc rẽ nhánh
Cấu trúc lặp
Mảng dữ liệu
Mảng hai chiều

I. CẤU TRÚC RẼ NHÁNH

Nói chung việc thực hiện chương trình là hoạt động tuần tự, tức thực hiện từng lệnh một từ câu lệnh bắt đầu của chương trình cho đến câu lệnh cuối cùng. Tuy nhiên, để việc lập trình hiệu quả hơn hầu hết các NNLT bậc cao đều có các câu lệnh rẽ nhánh và các câu lệnh lặp cho phép thực hiện các câu lệnh của chương trình không theo trình tự tuần tự như trong văn bản.

Phần này chúng tôi sẽ trình bày các câu lệnh cho phép rẽ nhánh như vậy. Để thống nhất mỗi câu lệnh được trình bày về cú pháp (tức cách viết câu lệnh), cách sử dụng, đặc điểm, ví dụ minh hoạ và một vài điều cần chú ý khi sử dụng lệnh.

1. Câu lệnh điều kiện if

a. Ý nghĩa

Một câu lệnh **if** cho phép chương trình có thể thực hiện khối lệnh này hay khối lệnh khác phụ thuộc vào một điều kiện được viết trong câu lệnh là đúng hay sai. Nói cách khác câu lệnh **if** cho phép chương trình rẽ nhánh (chỉ thực hiện 1 trong 2 nhánh).

b. Cú pháp

– **if (điều kiện) { khối lệnh 1; } else { khối lệnh 2; }**

– **if (điều kiện) { khối lệnh 1; }**

Trong cú pháp trên câu lệnh if có hai dạng: có else và không có else. điều kiện là một biểu thức lôgic tức nó có giá trị đúng (khác 0) hoặc sai (bằng 0).

Khi chương trình thực hiện câu lệnh if nó sẽ tính biểu thức điều kiện. Nếu điều kiện đúng chương trình sẽ tiếp tục thực hiện các lệnh trong khối lệnh 1, ngược lại nếu

điều kiện sai chương trình sẽ thực hiện khối lệnh 2 (nếu có else) hoặc không làm gì (nếu không có else).

c. Đặc điểm

- Đặc điểm chung của các câu lệnh có cấu trúc là bản thân nó chứa các câu lệnh khác. Điều này cho phép các câu lệnh if có thể lồng nhau.
- Nếu nhiều câu lệnh if (có else và không else) lồng nhau việc hiểu if và else nào đi với nhau cần phải chú ý. Qui tắc là else sẽ đi với if gần nó nhất mà chưa được ghép cặp với else khác. Ví dụ câu lệnh

```
if (n>0) if (a>b) c = a;
else c = b;
```

là tương đương với

```
if (n>0) { if (a>b) c = a; else c = b;}
```

d. Ví dụ minh hoạ

Ví dụ 1 : Bằng phép toán gán có điều kiện có thể tìm số lớn nhất max trong 2 số a, b như sau: `max = (a > b) ? a: b ;`

hoặc max được tìm bởi dùng câu lệnh if:

```
if (a > b) max = a; else max = b;
```

Ví dụ 2 : Tính năm nhuận. Năm thứ n là nhuận nếu nó chia hết cho 4, nhưng không chia hết cho 100 hoặc chia hết 400. Chú ý: một số nguyên a là chia hết cho b nếu phần dư của phép chia bằng 0, tức a%b == 0.

```
#include <iostream.h>
void main()
{
    int nam;
    cout << “Nam = “ ; cin >> nam ;
    if (nam%4 == 0 && year%100 !=0 || nam%400 == 0)
        cout << nam << "la nam nhuan” ;
    else
        cout << nam << "la nam khong nhuan” ;
}
```

Ví dụ 3 : Giải phương trình bậc 2. Cho phương trình $ax^2 + bx + c = 0$ ($a \neq 0$), tìm x.

```
#include <iostream.h>                    // tệp chứa các phương thức vào/ra
#include <math.h>                        // tệp chứa các hàm toán học
void main()
{
      float a, b, c;                     // khai báo các hệ số
      float delta;
      float x1, x2;                      // 2 nghiem
      cout << “Nhap a, b, c:\n” ; cin >> a >> b >> c ;    // qui ước nhập a ≠ 0
      delta = b*b - 4*a*c ;
      if (delta < 0) cout << “ph. trình vô nghiệm\n” ;
      else if (delta==0) cout<<“ph. trình có nghiệm kép:" << -b/(2*a) << '\n';
      else
      {
            x1 = (-b+sqrt(delta))/(2*a);
            x2 = (-b-sqrt(delta))/(2*a);
            cout << “nghiem 1 = " << x1 << " và nghiem 2 = " << x2 ;
      }
}
```

Chú ý: do C++ quan niệm "đúng" là một giá trị khác 0 bất kỳ và "sai" là giá trị 0 nên thay vì viết if (x != 0) hoặc if (x == 0) ta có thể viết gọn thành if (x) hoặc if (!x) vì nếu (x != 0) đúng thì ta có x ≠ 0 và vì x ≠ 0 nên (x) cũng đúng. Ngược lại nếu (x) đúng thì x ≠ 0, từ đó (x != 0) cũng đúng. Tương tự ta dễ dàng thấy được (x == 0) là tương đương với (!x).

2. Câu lệnh lựa chọn switch

a. Ý nghĩa

Câu lệnh if cho ta khả năng được lựa chọn một trong hai nhánh để thực hiện, do đó nếu sử dụng nhiều lệnh if lồng nhau sẽ cung cấp khả năng được rẽ theo nhiều nhánh. Tuy nhiên trong trường hợp như vậy chương trình sẽ rất khó đọc, do vậy C++ còn cung cấp một câu lệnh cấu trúc khác cho phép chương trình có thể chọn một trong nhiều nhánh để thực hiện, đó là câu lệnh switch.

b. Cú pháp

```
switch (biểu thức điều khiển)
{
    case biểu_thức_1: dãy lệnh 1 ;
    case biểu_thức_2: dãy lệnh 2 ;
    case ……...: ..... ;
    case biểu_thức_n: dãy lệnh n ;
    default: dãy lệnh n+1;
}
```

- biểu thức điều khiển: phải có kiểu nguyên hoặc kí tự,
- các biểu_thức_i: được tạo từ các hằng nguyên hoặc kí tự,
- các dãy lệnh có thể rỗng. Không cần bao dãy lệnh bởi cặp dấu {},
- nhánh default có thể có hoặc không và vị trí của nó có thể nằm bất kỳ trong câu lệnh (giữa các nhánh case), không nhất thiết phải nằm cuối cùng.

c. Cách thực hiện

Để thực hiện câu lệnh **switch** đầu tiên chương trình tính giá trị của biểu thức điều khiển (btđk), sau đó so sánh kết quả của btđk với giá trị của các biểu_thức_i bên dưới lần lượt từ biểu thức đầu tiên (thứ nhất) cho đến biểu thức cuối cùng (thứ n), nếu giá trị của btđk bằng giá trị của biểu thức thứ i đầu tiên nào đó thì chương trình sẽ thực hiện dãy lệnh thứ i và tiếp tục thực hiện tất cả dãy lệnh còn lại (từ dãy lệnh thứ i+1) cho đến hết (gặp dấu ngoặc đóng } của lệnh switch). Nếu quá trình so sánh không gặp biểu thức (nhánh case) nào bằng với giá trị của btđk thì chương trình thực hiện dãy lệnh trong default và tiếp tục cho đến hết (sau default có thể còn những nhánh case khác). Trường hợp câu lệnh switch không có nhánh default và btđk không khớp với bất cứ nhánh case nào thì chương trình không làm gì, coi như đã thực hiện xong lệnh switch.

Nếu muốn lệnh switch chỉ thực hiện nhánh thứ i (khi btđk = biểu_thức_i) mà không phải thực hiện thêm các lệnh còn lại thì cuối dãy lệnh thứ i thông thường ta đặt thêm lệnh **break;** đây là lệnh cho phép thoát ra khỏi một lệnh cấu trúc bất kỳ.

d. Ví dụ minh hoạ

Ví dụ 1 : In số ngày của một tháng bất kỳ nào đó được nhập từ bàn phím.

```
int th;
cout << “Cho biết tháng cần tính: “ ; cin >> th ;
switch (th)
```

```
{
    case 1: case 3: case 5: case 7: case 8: case 10:
    case 12: cout << "tháng này có 31 ngày" ; break ;
    case 2: cout << "tháng này có 28 ngày" ; break;
    case 4: case 6: case 9:
    case 11: cout << "tháng này có 30 ngày" ; break;
    default: cout << "Bạn đã nhập sai tháng, không có tháng này" ;
}
```

Trong chương trình trên giả sử NSD nhập tháng là 5 thì chương trình bắt đầu thực hiện dãy lệnh sau case 5 (không có lệnh nào) sau đó tiếp tục thực hiện các lệnh còn lại, cụ thể là bắt đầu từ dãy lệnh trong case 7, đến case 12 chương trình gặp lệnh in kết quả "tháng này có 31 ngày", sau đó gặp lệnh break nên chương trình thoát ra khỏi câu lệnh switch (đã thực hiện xong). Việc giải thích cũng tương tự cho các trường hợp khác của tháng. Nếu NSD nhập sai tháng (ví dụ tháng nằm ngoài phạm vi 1..12), chương trình thấy th không khớp với bất kỳ nhánh case nào nên sẽ thực hiện câu lệnh trong default, in ra màn hình dòng chữ "Bạn đã nhập sai tháng, không có tháng này" và kết thúc lệnh.

Ví dụ 2 : Nhập 2 số a và b vào từ bàn phím. Nhập kí tự thể hiện một trong bốn phép toán: cộng, trừ, nhân, chia. In ra kết quả thực hiện phép toán đó trên 2 số a, b.

```
void main()
{
    float a, b, c ;                 // các toán hạng a, b và kết quả c
    char dau ;                      // phép toán được cho dưới dạng kí tự
    cout << "Hãy nhập 2 số a, b: " ; cin >> a >> b ;
    cout << "và dấu phép toán: " ; cin >> dau ;
    switch (dau)
    {
        case '+': c = a + b ; break ;
        case '–': c = a - b ; break ;
        case 'x': case '.': case '*': c = a * b ; break ;
        case ':': case '/': c = a / b ; break ;
    }
    cout << setiosflags(ios::showpoint) << setprecision(4) ;      // in 4 số lẻ
```

```
        cout << "Kết quả là: " << c ;
}
```

Trong chương trình trên ta chấp nhận các kí tự x, ., * thể hiện cho phép toán nhân và :, / thể hiện phép toán chia.

3. Câu lệnh nhảy goto

a. Ý nghĩa

Một dạng khác của rẽ nhánh là câu lệnh nhảy **goto** cho phép chương trình chuyển đến thực hiện một đoạn lệnh khác bắt đầu từ một điểm được đánh dấu bởi một nhãn trong chương trình. Nhãn là một tên gọi do NSD tự đặt theo các qui tắt đặt tên gọi. Lệnh goto thường được sử dụng để tạo vòng lặp. Tuy nhiên việc xuất hiện nhiều lệnh goto dẫn đến việc khó theo dõi trình tự thực hiện chương trình, vì vậy lệnh này thường được sử dụng rất hạn chế.

b. Cú pháp

Goto <nhãn> ;

Vị trí chương trình chuyển đến thực hiện là đoạn lệnh đứng sau nhãn và dấu hai chấm (:).

c. Ví dụ minh hoạ

Ví dụ 3 : Nhân 2 số nguyên theo phương pháp Ấn độ.

Phương pháp Ấn độ cho phép nhân 2 số nguyên bằng cách chỉ dùng các phép toán nhân đôi, chia đôi và cộng. Các phép nhân đôi và chia đôi thực chất là phép toán dịch bit về bên trái (nhân) hoặc bên phải (chia) 1 bit. Đây là các phép toán cơ sở trong bộ xử lý, do vậy dùng phương pháp này sẽ làm cho việc nhân các số nguyên được thực hiện rất nhanh. Có thể tóm tắt phương pháp như sau: Giả sử cần nhân m với n. Kiểm tra m nếu lẻ thì cộng thêm n vào kq (đầu tiên kq được khởi tạo bằng 0), sau đó lấy m chia 2 và n nhân 2. Quay lại kiểm tra m và thực hiện như trên. Quá trình dừng khi không thể chia đôi m được nữa (m = 0), khi đó kq là kết quả cần tìm (tức kq = m*n). Để dễ hiểu phương pháp này chúng ta tiến hành tính trên ví dụ với các số m, n cụ thể. Giả sử m = 21 và n = 11. Các bước tiến hành được cho trong bảng dưới đây:

Bước	m (chia 2)	n (nhân 2)	kq (khởi tạo kq = 0)
1	21	11	m lẻ, cộng thêm 11 vào kq = 0 + 11 = 11
2	10	22	m chẵn, bỏ qua
3	5	44	m lẻ, cộng thêm 44 vào kq = 11 + 44 = 55

4	2	88	m chẵn, bỏ qua
5	1	176	m lẻ, cộng thêm 176 vào kq = 55 + 176 = 231
6	0		m = 0, dừng cho kết quả kq = 231

Sau đây là chương trình được viết với câu lệnh goto.

```
void main()
{
    long m, n, kq = 0;              // Các số cần nhân và kết quả kq
    cout << "Nhập m và n: " ; cin >> m >> n ;
    lap:                            // đây là nhãn để chương trình quay lại
    if (m%2) kq += n;               // nếu m lẻ thì cộng thêm n vào kq
    m = m >> 1;                     // dịch m sang phải 1 bit tức m = m / 2
    n = n << 1;                     // dịch m sang trái 1 bit tức m = m * 2
    if (m) goto lap;                // quay lại nếu m ≠ 0
    cout << "m nhân n =" << kq ;
}
```

II. CẤU TRÚC LẶP

Một trong những cấu trúc quan trọng của lập trình cấu trúc là các câu lệnh cho phép lặp nhiều lần một đoạn lệnh nào đó của chương trình. Chẳng hạn trong ví dụ về bài toán nhân theo phương pháp Ấn độ, để lặp lại một đoạn lệnh chúng ta đã sử dụng câu lệnh goto. Tuy nhiên như đã lưu ý việc dùng nhiều câu lệnh này làm chương trình rất khó đọc. Do vậy cần có những câu lệnh khác trực quan hơn và thực hiện các phép lặp một cách trực tiếp. C++ cung cấp cho chúng ta 3 lệnh lặp như vậy. Về thực chất 3 lệnh này là tương đương (cũng như có thể dùng goto thay cho cả 3 lệnh lặp này), tuy nhiên để chương trình viết được sáng sủa, rõ ràng, C++ đã cung cấp nhiều phương án cho NSD lựa chọn câu lệnh khi viết chương trình phù hợp với tính chất lặp. Mỗi bài toán lặp có một đặc trưng riêng, ví dụ lặp cho đến khi đã đủ số lần định trước thì dừng hoặc lặp cho đến khi một điều kiện nào đó không còn thoả mãn nữa thì dừng … việc sử dụng câu lệnh lặp phù hợp sẽ làm cho chương trình dễ đọc và dễ bảo trì hơn. Đây là ý nghĩa chung của các câu lệnh lặp, do vậy trong các trình bày về câu lệnh tiếp theo sau đây chúng ta sẽ không cần phải trình bày lại ý nghĩa của chúng.

1. Lệnh lặp for

a. Cú pháp

for (dãy biểu thức 1 ; điều kiện lặp ; dãy biểu thức 2) { khối lệnh lặp; }

- Các biểu thức trong các dãy biểu thức 1, 2 cách nhau bởi dấu phảy (,). Có thể có nhiều biểu thức trong các dãy này hoặc dãy biểu thức cũng có thể trống.
- Điều kiện lặp: là biểu thức lôgic (có giá trị đúng, sai).
- Các dãy biểu thức và/hoặc điều kiện có thể trống tuy nhiên vẫn giữ lại các dấu chấm phảy (;) để ngăn cách các thành phần với nhau.

b. Cách thực hiện

Khi gặp câu lệnh **for** trình tự thực hiện của chương trình như sau:

- Thực hiện dãy biểu thức 1 (thông thường là các lệnh khởi tạo cho một số biến),
- Kiểm tra điều kiện lặp, nếu đúng thì thực hiện khối lệnh lặp → thực hiện dãy biểu thức 2 → quay lai kiểm tra điều kiện lặp và lặp lại quá trình trên cho đến bước nào đó việc kiểm tra điều kiện lặp cho kết quả sai thì dừng.

Tóm lại, biểu thức 1 sẽ được thực hiện 1 lần duy nhất ngay từ đầu quá trình lặp sau đó thực hiện các câu lệnh lặp và dãy biểu thức 2 cho đến khi nào không còn thoả điều kiện lặp nữa thì dừng.

c. Ví dụ minh hoạ

Ví dụ 1 : Nhân 2 số nguyên theo phương pháp Ấn độ

```
void main()
{
    long m, n, kq;                    // Các số cần nhân và kết quả kq
    cout << “Nhập m và n: “ ; cin >> m >> n ;
    for (kq = 0 ; m ; m >>= 1, n <<= 1) if (m%2) kq += n ;
    cout << “m nhân n =” << kq ;
}
```

So sánh ví dụ này với ví dụ dùng goto ta thấy chương trình được viết rất gọn. Để bạn đọc dễ hiểu câu lệnh for, một lần nữa chúng ta nhắc lại cách hoạt động của nó thông qua ví dụ này, trong đó các thành phần được viết trong cú pháp là như sau:

- Dãy biểu thức 1: kq = 0,
- Điều kiện lặp: m. Ở đây điều kiện là đúng nếu $m \neq 0$ và sai nếu m = 0.

– Dãy biểu thức 2: m >>= 1 và n <<= 1. 2 biểu thức này có nghĩa m = m >> 1 (tương đương với m = m / 2) và n = n << 1 (tương đương với n = n * 2).

– Khối lệnh lặp: chỉ có một lệnh duy nhất if (m%2) kq += n ; (nếu phần dư của m chia 2 là khác 0, tức m lẻ thì cộng thêm n vào kq).

Cách thực hiện của chương trình như sau:

- Đầu tiên thực hiện biểu thức 1 tức gán kq = 0. Chú ý rằng nếu kq đã được khởi tạo trước bằng 0 trong khi khai báo (giống như trong ví dụ 6) thì thành phần biểu thức 1 ở đây có thể để trống (nhưng vẫn giữ lại dấu ; để phân biệt với các thành phần khác).
- Kiểm tra điều kiện: giả sử m ≠ 0 (tức điều kiện đúng) for sẽ thực hiện lệnh lặp tức kiểm tra nếu m lẻ thì cộng thêm n vào cho kq.
- Quay lại thực hiện các biểu thức 2 tức chia đôi m và nhân đôi n và vòng lặp được tiếp tục lại bắt đầu bằng việc kiểm tra m …
- Đến một bước lặp nào đó m sẽ bằng 0 (vì bị chia đôi liên tiếp), điều kiện không thoả, vòng lặp dừng và cho ta kết quả là kq.

Ví dụ 2 : Tính tổng của dãy các số từ 1 đến 100.

Chương trình dùng một biến đếm i được khởi tạo từ 1, và một biến kq để chứa tổng. Mỗi bước lặp chương trình cộng i vào kq và sau đó tăng i lên 1 đơn vị. Chương trình còn lặp khi nào i còn chưa vượt qua 100. Khi i lớn hơn 100 chương trình dừng. Sau đây là văn bản chương trình.

```
void main()
{
    int i, kq = 0;
    for (i = 1 ; i <= 100 ; i ++) kq += i ;
    cout << "Tổng = " << kq;
}
```

Ví dụ 3 : In ra màn hình dãy số lẻ bé hơn một số n nào đó được nhập vào từ bàn phím.

Chương trình dùng một biến đếm i được khởi tạo từ 1, mỗi bước lặp chương trình sẽ in i sau đó tăng i lên 2 đơn vị. Chương trình còn lặp khi nào i còn chưa vượt qua n. Khi i lớn hơn n chương trình dừng. Sau đây là văn bản chương trình.

```
void main()
{
    int n, i ;
```

```
    cout << "Hãy nhập n = " ; cin >> n ;
    for (i = 1 ; i < n ; i += 2) cout << i << '\n' ;
}
```

d. Đặc điểm

Thông qua phần giải thích cách hoạt động của câu lệnh for trong ví dụ 7 có thể thấy các thành phần của for có thể để trống, tuy nhiên các dấu chấm phẩy vẫn giữ lại để ngăn cách các thành phần với nhau. Ví dụ câu lệnh for (kq = 0 ; m ; m >>= 1, n <<= 1) if (m%2) kq += n ; trong ví dụ 7 có thể được viết lại như sau:

```
kq = 0;
for ( ; m ; ) { if (m%2) kq += n; m >>= 1; n <<= 1; }
```

Tương tự, câu lệnh for (i = 1 ; i <= 100 ; i ++) kq += i ; trong ví dụ 8 cũng có thể được viết lại như sau:

```
i = 1;
for ( ; i <= 100 ; ) kq += i ++ ;
```

(câu lệnh kq += i++; được thực hiện theo 2 bước: cộng i vào kq và tăng i (tăng sau)).

Trong trường hợp điều kiện trong for cũng để trống chương trình sẽ ngầm định là điều kiện luôn luôn đúng, tức vòng lặp sẽ lặp vô hạn lần (!). Trong trường hợp này để dừng vòng lặp trong khối lệnh cần có câu lệnh kiểm tra dừng và câu lệnh break.

Ví dụ câu lệnh for (i = 1 ; i <= 100 ; i ++) kq += i ; được viết lại như sau:

```
i = 1;
for ( ; ; )
{
    kq += i++;
    if (i > 100) break;
}
```

Tóm lại, việc sử dụng dạng viết nào của for phụ thuộc vào thói quen của NSD, tuy nhiên việc viết đầy đủ các thành phần của for làm cho việc đọc chương trình trở nên dễ dàng hơn.

e. Lệnh for lồng nhau

Trong dãy lệnh lặp có thể chứa cả lệnh for, tức các lệnh for cũng được phép lồng nhau như các câu lệnh có cấu trúc khác.

Ví dụ 4 : Bài toán cổ: vừa gà vừa chó bó lại cho tròn đếm đủ 100 chân. Hỏi có mấy gà

và mấy con chó, biết tổng số con là 36.

Để giải bài toán này ta gọi g là số gà và c là số chó. Theo điều kiện bài toán ta thấy g có thể đi từ 0 (không có con nào) và đến tối đa là 50 (vì chỉ có 100 chân), tương tự c có thể đi từ 0 đến 25. Như vậy ta có thể cho g chạy từ 0 đến 50 và với mỗi giá trị cụ thể của g lại cho c chạy từ 0 đến 25, lần lượt với mỗi cặp (g, c) cụ thể đó ta kiểm tra 2 điều kiện: g + c == 36 ? (số con) và 2g + 4c == 100 ? (số chân). Nếu cả 2 điều kiện đều thoả thì cặp (g, c) cụ thể đó chính là nghiệm cần tìm. Từ đó ta có chương trình với 2 vòng for lồng nhau, một vòng for cho g và một vòng cho c.

```
void main()
{
    int g, c ;
    for (g = 0 ; g <= 50 ; g++)
    for (c = 0 ; c <= 25 ; c++)
    if (g+c == 36 && 2*g+4*c == 100) cout << "gà=" << g << ", chó=" << c ;
}
```

Chương trình trên có thể được giải thích một cách ngắn gọn như sau: Đầu tiên cho g = 0, thực hiện lệnh for bên trong tức lần lượt cho c = 0, 1, …, 25, với c=0 và g=0 kiểm tra điều kiện, nếu thoả thì in kết quả nếu không thì bỏ qua, quay lại tăng c, cho đến khi nào c>25 thì kết thúc vòng lặp trong quay về vòng lặp ngoài tăng g lên 1, lại thực hiện vòng lặp trong với g=1 này (tức lại cho c chạy từ 0 đến 25). Khi g của vòng lặp ngoài vượt quá 50 thì dừng. Từ đó ta thấy số vòng lặp của chương trình là 50 x 25 = 1000 lần lặp.

Chú ý: Có thể giảm bớt số lần lặp bằng nhận xét số gà không thể vượt quá 36 (vì tổng số con là 36). Một vài nhận xét khác cũng có thể làm giảm số vòng lặp, tiết kiệm thời gian chạy của chương trình. Bạn đọc tự nghĩ thêm các phương án giải khác để giảm số vòng lặp đến ít nhất.

Ví dụ 5 : Tìm tất cả các phương án để có 100đ từ các tờ giấy bạc loại 10đ, 20đ và 50đ.

```
main()
{
    int t10, t20, t50;                    // số tờ 10đ, 20đ, 50đ
    sopa = 0;                             // số phương án
    for (t10 = 0 ; t10 <= 10 ; t10++)
    for (t20 = 0 ; t20 <= 5 ; t20++)
    for (t50 = 0 ; t50 <= 2 ; t50++)
```

```
        if (t10*10 + t20*20 + t50*50 == 100)          // nếu thoả thì
        {
              sopa++;                                 // tăng số phương án
              if (t10) cout << t10 << "tờ 10đ " ;     // in số tờ 10đ nếu ≠ 0
              if (t20) cout << "+" << t20 << "tờ 20đ " ;  // thêm số tờ 20đ nếu≠0
              if (t50) cout << "+" << t50 << "tờ 50đ " ;  // thêm số tờ 50đ nếu≠0
              cout << '\n' ;                          // xuống dòng
        }
        cout << "Tong so phuong an = " << sopa ;
  }
```

2. Lệnh lặp while

a. Cú pháp

while (điều kiện) { khối lệnh lặp ; }

b. Thực hiện

Khi gặp lệnh while chương trình thực hiện như sau: đầu tiên chương trình sẽ kiểm tra điều kiện, nếu đúng thì thực hiện khối lệnh lặp, sau đó quay lại kiểm tra điều kiện và tiếp tục. Nếu điều kiện sai thì dừng vòng lặp. Tóm lại có thể mô tả một cách ngắn gọn về câu lệnh while như sau: *lặp lại các lệnh trong khi điều kiện vẫn còn đúng*.

c. Đặc điểm

- Khối lệnh lặp có thể không được thực hiện lần nào nếu điều kiện sai ngay từ đầu.
- Để vòng lặp không lặp vô hạn thì trong khối lệnh thông thường phải có ít nhất một câu lệnh nào đó gây ảnh hưởng đến kết quả của điều kiện, ví dụ làm cho điều kiện đang đúng trở thành sai.
- Nếu điều kiện luôn luôn nhận giá trị đúng (ví dụ biểu thức điều kiện là 1) thì trong khối lệnh lặp phải có câu lệnh kiểm tra dừng và lệnh break.

d. Ví dụ minh hoạ

Ví dụ 1 : Nhân 2 số nguyên theo phương pháp Ấn độ

```
void main()
{
```

```
        long m, n, kq;                  // Các số cần nhân và kết quả kq
        cout << “Nhập m và n: “ ; cin >> m >> n ;
        kq = 0 ;
        while (m)
        {
            if (m%2) kq += n ;
            m >>= 1;
            n <<= 1;
        }
        cout << “m nhân n =” << kq ;
    }
```

Trong chương trình trên câu lệnh while (m) … được đọc là "trong khi m còn khác 0 thực hiện …", ta thấy trong khối lệnh lặp có lệnh m >>= 1, lệnh này sẽ ảnh hưởng đến điều kiện (m), đến lúc nào đó m bằng 0 tức (m) là sai và chương trình sẽ dừng lặp.

Câu lệnh while (m) … cũng có thể được thay bằng while (1) … như sau:

```
    void main()
    {
        long m, n, kq;                  // Các số cần nhân và kết quả kq
        cout << “Nhập m và n: “ ; cin >> m >> n ;
        kq = 0 ;
        while (1) {
            if (m%2) kq += n ;
            m >>= 1;
            n <<= 1;
            if (!m) break ;             // nếu m = 0 thì thoát khỏi vòng lặp
        }
        cout << “m nhân n =” << kq ;
    }
```

Ví dụ 2 : Bài toán cổ: vừa gà vừa chó bó lại cho tròn đếm dủ 100 chân. Hỏi có mấy gà và mấy con chó, biết tổng số con là 36.

```
    void main()
```

```
{
    int g, c ;
    g = 0 ;
    while (g <= 36) {
        c = 0 ;
        while (c <= 50) {
            if (g + c == 36 && 2*g + 4*c == 100) cout << g << c ;
            c++;
        }
        g++;
    }
}
```

Ví dụ 3 : Tìm ước chung lớn nhất (UCLN) của 2 số nguyên m và n.

Áp dụng thuật toán Euclide bằng cách liên tiếp lấy số lớn trừ đi số nhỏ khi nào 2 số bằng nhau thì đó là UCLN. Trong chương trình ta qui ước m là số lớn và n là số nhỏ. Thêm biến phụ r để tính hiệu của 2 số. Sau đó đặt lại m hoặc n bằng r sao cho m > n và lặp lại. Vòng lặp dừng khi m = n.

```
void main()
{
    int m, n, r;
    cout << "Nhập m, n: " ; cin >> m >> n ;
    if (m < n) { int t = m; m = n; n = t; }      // nếu m < n thì đổi vai trò hai số
    while (m != n) {
        r = m - n ;
        if (r > n) m = r; else { m = n ; n = r ; }
    }
    cout << "UCLN = " << m ;
}
```

Ví dụ 4 : Tìm nghiệm xấp xỉ của phương trình $e^x - 1.5 = 0$, trên đoạn [0, 1] với độ chính xác 10^{-6} bằng phương pháp chia đôi.

Để viết chương trình này chúng ta nhắc lại phương pháp chia đôi. Cho hàm f(x) liên tục và đổi dấu trên một đoạn [a, b] nào đó (tức f(a), f(b) trái dấu nhau hay f(a)*f(b)

< 0). Ta đã biết với điều kiện này chắc chắn đồ thị của hàm f(x) sẽ cắt trục hoành tại một điểm x_0 nào đó trong đoạn [a, b], tức x_0 là nghiệm của phương trình f(x) = 0. Tuy nhiên việc tìm chính xác x_0 là khó, vì vậy ta có thể tìm xấp xỉ x' của nó sao cho x' càng gần x_0 càng tốt. Lấy c là điểm giữa của đoạn [a, b], c sẽ chia đoạn [a, b] thành 2 đoạn con [a, c] và [c, b] và do f(a), f(b) trái dấu nên chắc chắn một trong hai đoạn con cũng phải trái dấu, tức nghiệm x_0 sẽ nằm trong đoạn này. Tiếp tục quá trình bằng cách chia đôi đoạn vừa tìm được … cho đến khi ta nhận được một đoạn con (trái dấu, chứa x_0) sao cho độ dài của đoạn con này bé hơn độ xấp xỉ cho trước thì dừng. Khi đó lấy bất kỳ điểm nào trên đoạn con này (ví dụ hai điểm mút hoặc điểm giữa của a và b) thì chắc chắn khoảng cách của nó đến x_0 cũng bé hơn độ xấp xỉ cho trước, tức có thể lấy điểm này làm nghiệm xấp xỉ của phương trình f(x) = 0.

Trong ví dụ này hàm f(x) chính là e^x - 1.5 và độ xấp xỉ là 10^{-6}. Đây là hàm liên tục trên toàn trục số và đổi dấu trên đoạn [0, 1] (vì f(0) = 1 – 1.5 < 0 còn f(1) = e - 1.5 > 0). Sau đây là chương trình.

```
void main()
{
    float a = 0, b = 1, c;              // các điểm mút a, b và điểm giữa c
    float fa, fc;                       // giá trị của f(x) tại các điểm a, c
    while (b-a > 1.0e-6)                // trong khi độ dài đoạn còn lớn hơn ε
    {
        c = (a + b)/2;                          // tìm điểm c giữa đoạn [a,b]
        fa = exp(a) - 1.5; fc = exp(c) - 1.5;   // tính f(a) và f(c)
        if (fa*fc == 0) break;                  // f(c) = 0 tức c là nghiệm
        if (fa*fc > 0) a = c; else b = c;
    }
    cout << "Nghiem xap xi cua phuong trinh = " << c ;
}
```

Trong chương trình trên câu lệnh if (fa*fc > 0) a = c; else b = c; dùng để kiểm tra f(a) và f(c), nếu cùng dấu (f(a)*f(c) > 0) thì hàm f(x) phải trái dấu trên đoạn con [c, b] do đó đặt lại đoạn này là [a, b] (để quay lại vòng lặp) tức đặt a = c và b giữ nguyên, ngược lại nếu hàm f(x) trái dấu trên đoạn con [a, c] thì đặt lại b = c còn a giữ nguyên. Sau đó vòng lặp quay lại kiểm tra độ dài đoạn [a, b] (mới) nếu đã bé hơn độ xấp xỉ thì dừng và lấy c làm nghiệm xấp xỉ, nếu không thì tính lại c và tiếp tục quá trình.

Để tính f(a) và f(c) chương trình đã sử dụng hàm exp(x), đây là hàm cho lại kết quả e^x, để dùng hàm này hoặc các hàm toán học nói chung, cần khai báo file nguyên

mẫu math.h.

3. Lệnh lặp do ... while

a. Cú pháp

do { khối lệnh lặp } while (điều kiện) ;

b. Thực hiện

Đầu tiên chương trình sẽ thực hiện khối lệnh lặp, tiếp theo kiểm tra điều kiện, nếu điều kiện còn đúng thì quay lại thực hiện khối lệnh và quá trình tiếp tục cho đến khi điều kiện trở thành sai thì dừng.

c. Đặc điểm

Các đặc điểm của câu lệnh do … while cũng giống với câu lệnh lặp while trừ điểm khác biệt, đó là khối lệnh trong do … while sẽ được thực hiện ít nhất một lần, trong khi trong câu lệnh while có thể không được thực hiện lần nào (vì lệnh while phải kiểm tra điều kiện trước khi thực hiện khối lệnh, do đó nếu điều kiện sai ngay từ đầu thì lệnh sẽ dừng, khối lệnh không được thực hiện lần nào. Trong khi đó lệnh do … while sẽ thực hiện khối lệnh rồi mới kiểm tra điều kiện lặp để cho phép thực hiện tiếp hoặc dừng).

d. Ví dụ minh hoạ

Ví dụ 1 : Tính xấp xỉ số pi theo công thức Euler $\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + ... + \frac{1}{n^2}$, với $\frac{1}{n^2} < 10^{-6}$.

```
void main()
{
    int n = 1; float S = 0;
    do S += 1.0/(n*n) while 1.0/(n*n) < 1.0e-6;
    float pi = sqrt(6*S);
    cout << "pi = " << pi ;
}
```

Ví dụ 2 : Kiểm tra một số n có là số nguyên tố.

Để kiểm tra một số n > 3 có phải là số nguyên tố ta lần lượt chia n cho các số i đi

từ 2 đến một nửa của n. Nếu có i sao cho n chia hết cho i thì n là hợp số ngược lại n là số nguyên tố.

```
void main()
{
    int i, n ;                                  // n: số cần kiểm tra
    cout << "Cho biết số cần kiểm tra: " ; cin >> n ;
    i = 2 ;
    do {
        if (n%i == 0) {
            cout << n << "là hợp số" ;
            return ;                            // dừng chương trình
        }
        i++;
    } while (i <= n/2);
    cout << n << "là số nguyên tố" ;
}
```

Ví dụ 3 : Nhập dãy kí tự và thống kê các loại chữ hoa, thường, chữ số và các loại khác còn lại đến khi gặp ENTER thì dừng.

```
void main()
{
    char c;                                     // kí tự dùng cho nhập
    int n1, n2, n3, n4 ;                        // số lượng các loại kí tự
    n1 = n2 = n3 = n4 = 0;
    cout << “Hãy nhập dãy kí tự: \n” ;
    do
    {
        cin >> c;
        if (‘a’ <= c && c <= ‘z’) n1++;         // nếu c là chữ thường thì tăng n1
        else if (‘A’ <= c && c <= ‘Z’) n2++;    // chữ hoa, tăng n2
        else if (‘0’ <= c && c <= ‘9’) n3++;    // chữ số, tăng n3
        else n4++;                              // loại khác, tăng n4
```

```
            cout << n1 << n2 << n3 << n4 ;          // in kết quả
        } while (c != 10) ;                         // còn lặp khi c còn khác kí tự ↵
    }
```

4. Lối ra của vòng lặp: break, continue

a. Lệnh break

Công dụng của lệnh dùng để thoát ra khỏi (chấm dứt) các câu lệnh cấu trúc, chương trình sẽ tiếp tục thực hiện các câu lệnh tiếp sau câu lệnh vừa thoát. Các ví dụ minh hoạ bạn đọc có thể xem lại trong các ví dụ về câu lệnh switch, for, while.

b. Lệnh continue

Lệnh dùng để quay lại đầu vòng lặp mà không chờ thực hiện hết các lệnh trong khối lệnh lặp.

Ví dụ 1 : Giả sử với mỗi i từ 1 đến 100 ta cần thực hiện một loạt các lệnh nào đó trừ những số i là số chính phương. Như vậy để tiết kiệm thời gian, vòng lặp sẽ kiểm tra nếu i là số chính phương thì sẽ quay lại ngay từ đầu để thực hiện với i tiếp theo.

```
int i ;
for (i = 1; i <= 100; i++) {
    if (i là số chính phương) continue;
    {                                       // dãy lệnh khác
        .
        .
        .
    }
}
```

(Để kiểm tra i có là số chính phương chúng ta so sánh căn bậc hai của i với phần nguyên của nó. Nếu hai số này bằng nhau thì i là số chính phương. Cụ thể nếu sqrt(i) = int(sqrt(i)) thì i là số chính phương. Ở đây sqrt(x) là hàm trả lại căn bậc hai của x. Để sử dụng hàm này cần phải khai báo file nguyên mẫu math.h.)

5. So sánh cách dùng các câu lệnh lặp

Thông qua các ví dụ đã trình bày bạn đọc có thể thấy rằng về mặt thực chất để tổ chức một vòng lặp chúng ta có thể chọn một trong các câu lệnh goto, for, while, do … while, có nghĩa về mặt khả năng thực hiện các câu lệnh này là như nhau. Tuy nhiên,

trong một ngữ cảnh cụ thể việc sử dụng câu lệnh phù hợp trong chúng làm cho chương trình sáng sủa, rõ ràng và tăng độ tin cậy lên cao hơn. Theo thói quen lập trình trong một số ngôn ngữ có trước và dựa trên đặc trưng riêng của từng câu lệnh, các lệnh lặp thường được dùng trong các ngữ cảnh cụ thể như sau:

- FOR thường được sử dụng trong những vòng lặp mà số lần lặp được biết trước, nghĩa là vòng lặp thường được tổ chức dưới dạng một (hoặc nhiều) biến đếm chạy từ một giá trị nào đó và đến khi đạt được đến một giá trị khác cho trước thì dừng. Ví dụ dạng thường dùng của câu lệnh for là như sau:
- for (i = gt1 ; i <= gt2 ; i++) … tức i tăng từ gt1 đến gt2 hoặc
- for (i = gt2 ; i >= gt1 ; i--) … tức i giảm từ gt2 xuống gt1
- Ngược lại với FOR, WHILE và DO … WHILE thường dùng trong các vòng lặp mà số lần lặp không biết trước, chúng thường được sử dụng khi việc lặp hay dừng phụ thuộc vào một biểu thức lôgic.
- WHILE được sử dụng khi khả năng thực hiện khối lặp không xảy ra lần nào, tức nếu điều kiện lặp có giá trị sai ngay từ đầu, trong khi đó DO … WHILE được sử dụng khi ta biết chắc chắn khối lệnh lặp phải được thực hiện ít nhất một lần.

III. MẢNG DỮ LIỆU

1. Mảng một chiều

a. Ý nghĩa

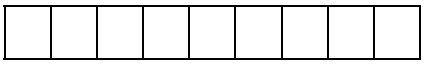
Khi cần lưu trữ một dãy n phần tử dữ liệu chúng ta cần khai báo n biến tương ứng với n tên gọi khác nhau. Điều này sẽ rất khó khăn cho người lập trình để có thể nhớ và quản lý hết được tất cả các biến, đặc biệt khi n lớn. Trong thực tế, hiển nhiên chúng ta gặp rất nhiều dữ liệu có liên quan đến nhau về một mặt nào đó, ví dụ chúng có cùng kiểu và cùng thể hiện một đối tượng: như các toạ độ của một vectơ, các số hạng của một ma trận, các sinh viên của một lớp hoặc các dòng kí tự của một văn bản … Lợi dụng đặc điểm này toàn bộ dữ liệu (cùng kiểu và cùng mô tả một đối tượng) có thể chỉ cần chung một tên gọi để phân biệt với các đối tượng khác, và để phân biệt các dữ liệu trong cùng đối tượng ta sử dụng cách đánh số thứ tự cho chúng, từ đó việc quản lý biến sẽ dễ dàng hơn, chương trình sẽ gọn và có tính hệ thống hơn.

Giả sử ta có 2 vectơ trong không gian ba chiều, mỗi vec tơ cần 3 biến để lưu 3 toạ độ, vì vậy để lưu toạ độ của 2 vectơ chúng ta phải dùng đến 6 biến, ví dụ x1, y1, z1 cho vectơ thứ nhất và x2, y2, z2 cho vectơ thứ hai. Một kiểu dữ liệu mới được gọi là mảng một chiều cho phép ta chỉ cần khai báo 2 biến v1 và v2 để chỉ 2 vectơ, trong đó mỗi v1

hoặc v2 sẽ chứa 3 dữ liệu được đánh số thứ tự từ 0 đến 2, trong đó ta có thể ngầm định thành phần 0 biểu diễn toạ độ x, thành phần 1 biểu diễn toạ độ y và thành phần có số thứ tự 2 sẽ biểu diễn toạ độ z.

Tóm lại, mảng là một dãy các thành phần có cùng kiểu được sắp kề nhau liên tục trong bộ nhớ. Tất cả các thành phần đều có cùng tên là tên của mảng. Để phân biệt các thành phần với nhau, các thành phần sẽ được đánh số thứ tự từ 0 cho đến hết mảng. Khi cần nói đến thành phần cụ thể nào của mảng ta sẽ dùng tên mảng và kèm theo số thứ tự của thành phần đó.

Dưới đây là hình ảnh của một mảng gồm có 9 thành phần, các thành phần được đánh số từ 0 đến 8.



0 1 2 3 4 5 6 7 8

b. Khai báo

<tên kiểu> <tên mảng>[số thành phần] ; // không khởi tạo

<tên kiểu> <tên mảng>[số thành phần] = { dãy giá trị } ; // có khởi tạo

<tên kiểu> <tên mảng>[ ] = { dãy giá trị } ; // có khởi tạo

- Tên kiểu là kiểu dữ liệu của các thành phần, các thành phần này có kiểu giống nhau. Thỉnh thoảng ta cũng gọi các thành phần là phần tử.
- Cách khai báo trên giống như khai báo tên biến bình thường nhưng thêm số thành phần trong mảng giữa cặp dấu ngoặc vuông [] còn được gọi là kích thước của mảng. Mỗi tên mảng là một biến và để phân biệt với các biến thông thường ta còn gọi là biến mảng.
- Một mảng dữ liệu được lưu trong bộ nhớ bởi dãy các ô liên tiếp nhau. Số lượng ô bằng với số thành phần của mảng và độ dài (byte) của mỗi ô đủ để chứa thông tin của mỗi thành phần. Ô đầu tiên được đánh thứ tự bởi 0, ô tiếp theo bởi 1, và tiếp tục cho đến hết. Như vậy nếu mảng có n thành phần thì ô cuối cùng trong mảng sẽ được đánh số là n - 1.
- Dạng khai báo thứ 2 cho phép khởi tạo mảng bởi dãy giá trị trong cặp dấu {}, mỗi giá trị cách nhau bởi dấu phảy (,), các giá trị này sẽ được gán lần lượt cho các phần tử của mảng bắt đầu từ phần tử thứ 0 cho đến hết dãy. Số giá trị có thể bé hơn số phần tử. Các phần tử mảng chưa có giá trị sẽ không được xác định cho đến khi trong chương trình nó được gán một giá trị nào đó.
- Dạng khai báo thứ 3 cho phép vắng mặt số phần tử, trường hợp này số phần tử được xác định bởi số giá trị của dãy khởi tạo. Do đó nếu vắng mặt cả dãy khởi

tạo là không được phép (chẳng hạn khai báo int a[] là sai).

Ví dụ:

- Khai báo biến chứa 2 vectơ a, b trong không gian 3 chiều:

```
float a[3] , b[3] ;
```

- Khai báo 3 phân số a, b, c; trong đó a = 1/3 và b = 3/5:

```
int a[2] = {1, 3} , b[2] = {3, 5} , c[2] ;
```

ở đây ta ngầm qui ước thành phần đầu tiên (số thứ tự 0) là tử và thành phần thứ hai (số thứ tự 1) là mẫu của phân số.

- Khai báo mảng L chứa được tối đa 100 số nguyên dài:

```
long L[100] ;
```

- Khai báo mảng dong (dòng), mỗi dòng chứa được tối đa 80 kí tự:

```
char dong[80] ;
```

- Khai báo dãy Data chứa được 5 số thực độ chính xác gấp đôi:

```
double Data[] = { 0,0,0,0,0 };                    // khởi tạo tạm thời bằng 0
```

c. Cách sử dụng

i. Để chỉ thành phần thứ i (hay chỉ số i) của một mảng ta viết tên mảng kèm theo chỉ số trong cặp ngoặc vuông []. Ví dụ với các phân số trên a[0], b[0], c[0] để chỉ tử số và a[1], b[1], c[1] để chỉ mẫu số của 3 phân số a,b,c.

ii. Tuy mỗi mảng biểu diễn một đối tượng nhưng chúng ta không thể áp dụng các thao tác lên toàn bộ mảng mà phải thực hiện thao tác thông qua từng thành phần của mảng. Ví dụ chúng ta không thể nhập dữ liệu cho mảng a[10] bằng câu lệnh:

```
cin >> a ;                                        // sai
```

mà phải nhập cho từng phần tử từ a[0] đến a[9] của a. Dĩ nhiên trong trường hợp này chúng ta phải cần đến lệnh lặp for:

```
int i ;
for (i = 0 ; i < 10 ; i++) cin >> a[i] ;
```

Tương tự, giả sử chúng ta cần cộng 2 phân số a, b và đặt kết quả vào c.

Không thể viết:

```
c = a + b ;                                       // sai
```

mà cần phải tính từng phần tử của c:

```
c[0] = a[0] * b[1] + a[1] * b[0] ;    // tử số
c[1] = a[1] * b[1] ;                  // mẫu số
```

Để khắc phục nhược điểm này, trong các chương sau C++ cung cấp một kiểu dữ liệu mới gọi là lớp, và cho phép NSD có thể định nghĩa riêng phép cộng cho 2 mảng tuỳ ý, khi đó có thể viết một cách đơn giản và quen thuộc c = a + b để cộng 2 phân số.

d. Ví dụ minh hoạ

Ví dụ 1 : Tìm tổng, tích 2 phân số.

```
void main()
{
    int a[2], b[2], tong[2], tich[2] ;
    cout << "Nhập a. Tử = " ; cin >> a[0] ; cout << "mẫu = " ; cin >> a[1] ;
    cout << "Nhập b. Tử = " ; cin >> b[0] ; cout << "mẫu = " ; cin >> b[1] ;
    tong[0] = a[0]*b[1] + a[1]*b[0] ; tong[1] = a[1] * b[1] ;
    tich[0] = a[0]*b[0]; tich[1] = a[1] * b[1] ;
    cout << "Tổng = " << tong[0] << '/' << tong[1] ;
    cout << "Tích = " << tich[0] << '/' << tich[1] ;
}
```

Ví dụ 2 : Nhập dãy số nguyên, tính: số số hạng dương, âm, bằng không của dãy.

```
void main()
{
    float a[50], i, n, sd, sa, s0;                    // a chứa tối đa 50 số
    cout << "Nhập số phần tử của dãy: " ; cin >> n;       // nhập số phần tử
    for (i=0; i<n; i++) { cout << "a[" << i << "] = " ; cin >> a[i]; }  // nhập dãy số
    sd = sa = s0 = 0 ;
    for (i=1; i<n; i++) {
        if (a[i] > 0 ) sd++;
        if (a[i] < 0 ) sa++;
        if (a[i] == 0 ) s0++;
    }
    cout << "Số số dương = " << sd << " số số âm = " << sa ;
```

```
        cout << "Số số bằng 0 = " << s0 ;
}
```

Ví dụ 3 : Tìm số bé nhất của một dãy số. In ra số này và vị trí của nó trong dãy.

Chương trình sử dụng mảng a để lưu dãy số, n là số phần tử thực sự trong dãy, min lưu số bé nhất tìm được và k là vị trí của min trong dãy. min được khởi tạo bằng giá trị đầu tiên (a[0]), sau đó lần lượt so sánh với các số hạng còn lại, nếu gặp số hạng nhỏ hơn, min sẽ nhận giá trị của số hạng này. Quá trình so sánh tiếp tục cho đến hết dãy. Vì số số hạng của dãy là biết trước (n), nên số lần lặp cũng được biết trước (n-1 lần lặp), do vậy chúng ta sẽ sử dụng câu lệnh for cho ví dụ này.

```
void main()
{
        float a[100], i, n, min, k;                     // a chứa tối đa 100 số
        cout << "Nhập số phần tử của dãy: " ; cin >> n;
        for (i=0; i<n; i++) { cout << "a[" << i << "] = " ; cin >> a[i]; }
        min = a[0]; k = 0;
        for (i=1; i<n; i++) if (a[i] < min ) { min = a[i]; k = i; }
cout << "Số bé nhất là " << min << "tại vị trí " << k;
}
```

Ví dụ 4 : Nhập và sắp xếp tăng dần một dãy số. Thuật toán được tiến hành bằng cách sắp xếp dần từng số hạng bé nhất lên đầu dãy. Giả sử đã sắp được i-1 vị trí, ta sẽ tìm số bé nhất trong dãy còn lại (từ vị trí thứ i đến n-1) và đưa số này lắp vào vị trí thứ i. Để thực hiện, chúng ta so sánh a[i] lần lượt với từng số a[j] trong dãy còn lại (tức j đi từ i+1 đến n), nếu gặp a[j] bé hơn a[i] thì đổi chỗ hai số này với nhau.

```
void main()
{
        float a[100], i, j, n, tam;
        cout << "Cho biết số phần tử n = " ; cin >> n ;
        for (i=0; i<n; i++) {cout<<"a[" <<i<< "] = "; cin >> a[i] ;}    // nhập dữ liệu
        for (i=0; i<n; i++) {
                for (j=i+1; j<n; j++)
                if (a[i] > a[j]) { tam = a[i]; a[i] = a[j]; a[j] = tam; }          // đổi chỗ
        }
```

```
    for (i=0; i<n; i++) cout << a[i] ;                    // in kết quả
    getch();
}
```

2. Xâu kí tự

Một xâu kí tự là một dãy bất kỳ các kí tự (kể cả dấu cách) do vậy nó có thể được lưu bằng mảng kí tự. Tuy nhiên để máy có thể nhận biết được mảng kí tự này là một xâu, cần thiết phải có kí tự kết thúc xâu, theo qui ước là kí tự có mã 0 (tức '\0') tại vị trí nào đó trong mảng. Khi đó xâu là dãy kí tự bắt đầu từ phần tử đầu tiên (thứ 0) đến kí tự kết thúc xâu đầu tiên (không kể các kí tự còn lại trong mảng).

0	1	2	3	4	5	6	7
H	E	L	L	O	\0		
H	E	L	\0	L	O	\0	
\0	H	E	L	L	O	\0	

Hình vẽ trên minh hoạ 3 xâu, mỗi xâu được chứa trong mảng kí tự có độ dài tối đa là 8. Nội dung xâu thứ nhất là "Hello" có độ dài thực tế là 5 kí tự, chiếm 6 ô trong mảng (thêm ô chứa kí tự kết thúc '\0'). Xâu thứ hai có nội dung "Hel" với độ dài 3 (chiếm 4 ô) và xâu cuối cùng biểu thị một xâu rỗng (chiếm 1 ô). Chú ý mảng kí tự được khai báo với độ dài 8 tuy nhiên các xâu có thể chỉ chiếm một số kí tự nào đó trong mảng này và tối đa là 7 kí tự.

a. Khai báo

```
char <tên xâu>[độ dài] ;                    // không khởi tạo
char <tên xâu>[độ dài] = xâu kí tự ;        // có khởi tạo
char <tên xâu>[] = xâu kí tự ;              // có khởi tạo
```

- Độ dài mảng là số kí tự tối đa có thể có trong xâu. Độ dài thực sự của xâu chỉ tính từ đầu mảng đến dấu kết thúc xâu (không kể dấu kết thúc xâu '\0').
- Do một xâu phải có dấu kết thúc xâu nên trong khai báo độ dài của mảng cần phải khai báo thừa ra một phần tử. Thực chất độ dài tối đa của xâu = độ dài mảng - 1. Ví dụ nếu muốn khai báo mảng s chứa được xâu có độ dài tối đa 80 kí tự, ta cần phải khai báo `char s[81]`.
- Cách khai báo thứ hai có kèm theo khởi tạo xâu, đó là dãy kí tự đặt giữa cặp dấu nháy kép. Ví dụ:

```
char hoten[26] ;                    // xâu họ tên chứa tối đa 25 kí tự
```

```
char monhoc[31] = "NNLT C++" ;
```

xâu môn học chứa tối đa 30 kí tự, được khởi tạo với nội dung "NNLT C++" với độ dài thực sự là 10 kí tự (chiếm 11 ô đầu tiên trong mảng monhoc[31]).

– Cách khai báo thứ 3 tự chương trình sẽ quyết định độ dài của mảng bởi xâu khởi tạo (bằng độ dài xâu + 1). Ví dụ:

```
char thang[] = "Mười hai" ;          // độ dài mảng = 9
```

b. Cách sử dụng

Tương tự như các mảng dữ liệu khác, xâu kí tự có những đặc trưng như mảng, tuy nhiên chúng cũng có những điểm khác biệt. Dưới đây là các điểm giống và khác nhau đó.

- Truy cập một kí tự trong xâu: cú pháp giống như mảng. Ví dụ:

```
char s[50] = "I\'m a student" ;      // chú ý kí tự ' phải được viết là \'
cout << s[0] ;                       // in kí tự đầu tiên, tức kí tự 'I'
s[1] = 'a' ;                         // đặt lại kí tự thứ 2 là 'a'
```

- Không được thực hiện các phép toán trực tiếp trên xâu như:

```
char s[20] = "Hello", t[20] ;        // khai báo hai xâu s và t
t = "Hello" ;                        // sai, chỉ gán được khi khai báo
t = s ;                              // sai, không gán được toàn bộ mảng
if (s < t) …                         // sai, không so sánh được hai mảng
…
```

- Toán tử nhập dữ liệu >> vẫn dùng được nhưng có nhiều hạn chế. Ví dụ

```
char s[60] ;
cin >> s ;
cout << s ;
```

nếu xâu nhập vào là "Tin học hoá" chẳng hạn thì toán tử >> chỉ nhập "Tin" cho s (bỏ tất cả các kí tự đứng sau dấu trắng), vì vậy khi in ra trên màn hình chỉ có từ "Tin".

Vì các phép toán không dùng được trực tiếp trên xâu nên các chương trình dịch đã viết sẵn các hàm thư viện được khai báo trong file nguyên mẫu string.h. Các hàm này giải quyết được hầu hết các công việc cần thao tác trên xâu. Nó cung cấp cho NSD phương tiện để thao tác trên xâu như gán, so sánh, sao chép, tính độ dài xâu, nhập, in, … Để sử dụng được các hàm này đầu chương trình cần có khai báo string.h. Phần lớn các hàm này sẽ được giới thiệu trong phần tiếp sau.

c. Phương thức nhập xâu (#include <iostream.h>)

Do toán tử nhập >> có hạn chế đối với xâu kí tự nên C++ đưa ra hàm riêng (còn gọi là phương thức) **cin.getline(s,n)** để nhập xâu kí tự. Hàm có 2 đối với s là xâu cần nhập nội dung và n-1 là số kí tự tối đa của xâu. Giống phương thức nhập kí tự cin.get(c), khi gặp hàm cin.getline(s,n) chương trình sẽ nhìn vào bộ đệm bàn phím lấy ra n-1 kí tự (nếu đủ hoặc lấy tất cả kí tự còn lại, trừ kí tự enter) và gán cho s. Nếu tại thời điểm đó bộ đệm đang rỗng, chương trình sẽ tạm dừng chờ NSD nhập dữ liệu (dãy kí tự) vào từ bàn phím. NSD có thể nhập vào dãy với độ dài bất kỳ cho đến khi nhấn Enter, chương trình sẽ lấy ra n-1 kí tự đầu tiên gán cho s, phần còn lại vẫn được lưu trong bộ đệm (kể cả kí tự Enter) để dùng cho lần nhập sau. Hiển nhiên, sau khi gán các kí tự cho s, chương trình sẽ tự động đặt kí tự kết thúc xâu vào ô tiếp theo của xâu s.

Ví dụ 1 : Xét đoạn lệnh sau

```
char s[10] ;
cin.getline(s, 10) ;
cout << s << endl ;
cin.getline(s, 10) ;
cout << s << endl ;
```

giả sử ta nhập vào bàn phím dòng kí tự: 1234567890abcd ↵. Khi đó lệnh cin.getline(s,10) đầu tiên sẽ gán xâu "123456789" (9 kí tự) cho s, phần còn lại vẫn lưu trong bộ đệm bàn phím. Tiếp theo s được in ra màn hình. Đến lệnh cin.getline(s,10) thứ hai NSD không phải nhập thêm dữ liệu, chương trình tự động lấy nốt số dữ liệu còn lại (vì chưa đủ 9 kí tự) "0abcd" để gán cho s. Sau đó in ra màn hình. Như vậy trên màn hình sẽ xuất hiện hai dòng:

```
123456789
0abcd
```

Ví dụ 2 : Nhập một ngày tháng dạng Mỹ (mm/dd/yy), đổi sang ngày tháng dạng Việt Nam rồi in ra màn hình.

```
#include <iostream.h>
main()
{
    char US[9], VN[9] = " / / " ;          // khởi tạo trước hai dấu /
    cin.getline(US, 9) ;              // nhập ngày tháng, ví dụ "05/01/99"
    VN[0] = US[3]; VN[1] = US[4] ;        // ngày
    VN[3] = US[0]; VN[4] = US[1] ;        // tháng
```

```
        VN[6] = US[6]; VN[7] = US[7] ;          // năm
        cout << VN << endl ;
}
```

d. Một số hàm xử lí xâu (#include <string.h>)

- **strcpy(s, t) ;**

Gán nội dung của xâu t cho xâu s (thay cho phép gán = không được dùng). Hàm sẽ sao chép toàn bộ nội dung của xâu t (kể cả kí tự kết thúc xâu) vào cho xâu s. Để sử dụng hàm này cần đảm bảo độ dài của mảng s ít nhất cũng bằng độ dài của mảng t. Trong trường hợp ngược lại kí tự kết thúc xâu sẽ không được ghi vào s và điều này có thể gây treo máy khi chạy chương trình.

Ví dụ:

```
char s[10], t[10] ;
t = "Face" ;                        // không được dùng
s = t ;                             // không được dùng
strcpy(t, "Face") ;                 // được, gán "Face" cho t
strcpy(s, t) ;                      // được, sao chép t sang s
cout << s << " to " << t ;          // in ra: Face to Face
```

- **strncpy(s, t, n) ;**

Sao chép n kí tự của t vào s. Hàm này chỉ làm nhiệm vụ sao chép, không tự động gắn kí tự kết thúc xâu cho s. Do vậy NSD phải thêm câu lệnh đặt kí tự '\0' vào cuối xâu s sau khi sao chép xong.

Ví dụ:

```
char s[10], t[10] = "Steven";
strncpy(s, t, 5) ;                  // copy 5 kí tự "Steve" vào s
s[5] = '\0' ;                       // đặt dấu kết thúc xâu
// in câu: Steve is young brother of Steven
cout << s << " is young brother of " << t ;
```

Một sử dụng có ích của hàm này là copy một xâu con bất kỳ của t và đặt vào s. Ví dụ cần copy xâu con dài 2 kí tự bắt đầu từ kí tự thứ 3 của xâu t và đặt vào s, ta viết **strncpy(s, t+3, 2)**. Ngoài ra xâu con được copy có thể được đặt vào vị trí bất kỳ của s (không nhất thiết phải từ đầu xâu s) chẳng hạn đặt vào từ vị trí thứ 5, ta viết: **strncpy(s+5, t+3, 2)**. Câu lệnh này có nghĩa: lấy 2 kí tự thứ 3 và thứ 4 của xâu t đặt vào 2 ô thứ 5 và thứ 6 của xâu s. Trên cơ sở này chúng ta có thể viết các đoạn chương

trình ngắn để thay thế một đoạn con bất kỳ nào đó trong s bởi một đoạn con bất kỳ (có độ dài tương đương) trong t. Ví dụ các dòng lệnh chuyển đổi ngày tháng trong ví dụ trước có thể viết lại bằng cách dùng hàm strncpy như sau:

```
strncpy(VN+0, US+3, 2) ;          // ngày
strncpy(VN+3, US+0, 2) ;          // tháng
strncpy(VN+6, US+6, 2);           // năm
```

- **strcat(s, t);**

Nối một bản sao của t vào sau s (thay cho phép +). Hiển nhiên hàm sẽ loại bỏ kí tự kết thúc xâu s trước khi nối thêm t. Việc nối sẽ đảm bảo lấy cả kí tự kết thúc của xâu t vào cho s (nếu s đủ chỗ) vì vậy NSD không cần thêm kí tự này vào cuối xâu. Tuy nhiên, hàm không kiểm tra xem liệu độ dài của s có đủ chỗ để nối thêm nội dung, việc kiểm tra này phải do NSD đảm nhiệm. Ví dụ:

```
char a[100] = "Mẫn", b[4] = "tôi";
strcat(a, " và ");
strcat(a, b);
cout << a                                  // Mẫn và tôi

char s[100] , t[100] = "Steve" ;
strncpy(s, t, 3); s[3] = '\0';             // s = "Ste"
strcat(s, "p");                            // s = "Step"
cout << t << " goes "<< s << " by " <<s    // Steve goes Step by Step
```

- **strncat(s, t, n);**

Nối bản sao n kí tự đầu tiên của xâu t vào sau xâu s. Hàm tự động đặt thêm dấu kết thúc xâu vào s sau khi nối xong (tương phản với strncpy()). Cũng giống strcat hàm đòi hỏi độ dài của s phải đủ chứa kết quả. Tương tự, có thể sử dụng cách viết **strncat(s, t+k, n)** để nối n kí tự từ vị trí thứ k của xâu t cho s.

Ví dụ:

```
char s[20] = "Nhà " ;
char t[] = "vua chúa"
strncat(s, t, 3) ;                 // s = "Nhà vua"
hoặc:
strncat(s, t+4, 4) ;               // s = "Nhà chúa"
```

- **strcmp(s, t);**

Hàm so sánh 2 xâu s và t (thay cho các phép toán so sánh). Giá trị trả lại là hiệu 2 kí tự khác nhau đầu tiên của s và t. Từ đó, nếu s1 < s2 thì hàm trả lại giá trị âm, bằng 0 nếu s1==s2, và dương nếu s1 > s2. Trong trường hợp chỉ quan tâm đến so sánh bằng, nếu hàm trả lại giá trị 0 là 2 xâu bằng nhau và nếu giá trị trả lại khác 0 là 2 xâu khác nhau.

Ví dụ:

```
if (strcmp(s,t)) cout << "s khác t"; else cout << "s bằng t" ;
```

- **strncmp(s, t) ;**

Giống hàm strcmp(s, t) nhưng chỉ so sánh tối đa n kí tự đầu tiên của hai xâu.

Ví dụ:

```
char s[] = "Hà Nội" , t[] = "Hà nội" ;
cout << strcmp(s,t) ;                  // -32 (vì 'N' = 78, 'n' = 110)
cout << strncmp(s, t, 3) ;             // 0 (vì 3 kí tự đầu của s và t là như nhau)
```

- **strcmpi(s, t) ;**

Như strcmp(s, t) nhưng không phân biệt chữ hoa, thường.

Ví dụ:

```
char s[] = "Hà Nội" , t[] = "hà nội" ;
cout << strcmpi(s, t) ;                // 0 (vì s = t)
```

- **strupr(s);**

Hàm đổi xâu s thành in hoa, và cũng trả lại xâu in hoa đó.

Ví dụ:

```
char s[10] = "Ha noi" ;
cout << strupr(s) ;                    // HA NOI
cout << s ;                            // HA NOI (s cũng thành in hoa)
```

- **strlwr(s);**

Hàm đổi xâu s thành in thường, kết quả trả lại là xâu s.

Ví dụ:

```
char s[10] = "Ha Noi" ;
cout << strlwr(s) ;                    // ha noi
cout << s ;                            // ha noi (s cũng thành in thường)
```

- **strlen(s) ;**

Hàm trả giá trị là độ dài của xâu s.

Ví dụ:

```
char s[10] = "Ha Noi" ;
cout << strlen(s) ;                    // 5
```

Sau đây là một số ví dụ sử dụng tổng hợp các hàm trên.

Ví dụ 1 : Thống kê số chữ 'a' xuất hiện trong xâu s.

```
main()
{
    const int MAX = 100;
    char s[MAX+1];
    int sokitu = 0;
    cin.getline(s, MAX+1);
    for (int i=0; i < strlen(s); i++)   if (s[i] = 'a ') sokitu++;
    cout << "Số kí tự = " << sokitu << endl ;
}
```

Ví dụ 2 : Tính độ dài xâu bằng cách đếm từng kí tự (tương đương với hàm strlen())

```
main()
{
    char s[100];                       // độ dài tối đa là 99 kí tự
    cin.getline(s, 100);               // nhập xâu s
    for (int i=0 ; s[i] != '\0' ; i++) ;   // chạy từ đầu đến cuối xâu
    cout << "Độ dài xâu = " << i ;
}
```

Ví dụ 3 : Sao chép xâu s sang xâu t (tương đương với hàm strcpy(t,s))

```
void main()
{
    char s[100], t[100];
    cin.getline(s, 100);               // nhập xâu s
    int i=0;
```

```
        while ((t[i] = s[i]) != '\0') i++;          // copy cả dấu kết thúc xâu '\0'
        cout << t << endl ;
}
```

Ví dụ 4 : Cắt dấu cách 2 đầu của xâu s. Chương trình sử dụng biến i chạy từ đầu xâu đến vị trí đầu tiên có kí tự khác dấu trắng. Từ vị trí này sao chép từng kí tự còn lại của xâu về đầu xâu bằng cách sử dụng thêm biến j để làm chỉ số cho xâu mới. Kết thúc sao chép j sẽ ở vị trí cuối xâu (mới). Cho j chạy ngược về đầu xâu cho đến khi gặp kí tự đầu tiên khác dấu trắng. Đặt dấu kết thúc xâu tại đây.

```
main()
{
        char s[100];
        cin.getline(s, 100);                        // nhập xâu s
        int i, j ;
        i = j = 0;
        while (s[i++] == ' '); i-- ;                // bỏ qua các dấu cách đầu tiên
        while (s[i] != '\0') s[j++] = s[i++] ;      // sao chép phần còn lại vào s
        while (s[--j] == ' ') ;                     // bỏ qua các dấu cách cuối
        s[j+1] = '\0' ;                             // đặt dấu kết thúc xâu
        cout << s ;
}
```

Ví dụ 5 : Chạy dòng chữ quảng cáo vòng tròn từ phải sang trái giữa màn hình.

Giả sử hiện 30 kí tự của xâu quảng cáo. Ta sử dụng vòng lặp. Cắt 30 kí tự đầu tiên của xâu cho vào biến hien, hiện biến này ra màn hình. Bước lặp tiếp theo cắt ra 30 kí tự của xâu nhưng dịch sang phải 1 kí tự cho vào biến hien và hiện ra màn hình. Quá trình tiếp tục, mỗi bước lặp ta dịch chuyển nội dung cần hiện ra màn hình 1 kí tự, do hiệu ứng của mắt ta thấy dòng chữ sẽ chạy từ biên phải về biên trái của màn hình. Để quá trình chạy theo vòng tròn (khi hiện đến kí tự cuối của xâu sẽ hiện quay lại từ kí tự đầu của xâu) chương trình sử dụng biến i đánh dấu điểm đầu của xâu con cần cắt cho vào hien, khi i bằng độ dài của xâu chương trình đặt lại i = 0 (cắt lại từ đầu xâu). Ngoài ra, để phần cuối xâu nối với phần đầu (tạo thành vòng tròn) ngay từ đầu chương trình, xâu quảng cáo sẽ được nối thành gấp đôi.

Vòng lặp tiếp tục đến khi nào NSD ấn phím bất kỳ (chương trình nhận biết điều này nhờ vào hàm kbhit() thuộc file nguyên mẫu conio.h) thì dừng. Để dòng chữ chạy

không quá nhanh chương trình sử dụng hàm trễ delay(n) (thuộc dos.h, tạm dừng trong n phần nghìn giây) với n được điều chỉnh thích hợp theo tốc độ của máy. Hàm gotoxy(x, y) (thuộc conio.h) trong chương trình đặt con trỏ màn hình tại vị trí cột x dòng y để đảm bảo dòng chữ luôn luôn hiện ra tại đúng một vị trí trên màn hình.

```
#include <iostream.h>
#include <conio.h>
#include <dos.h>
main()
{
    char qc[100] = "Quảng cáo miễn phí: Không có tiền thì không có kem. ";
    int dd = strlen(qc);
    char tam[100] ; strcpy(tam, qc) ;
    strcat(qc, tam) ;                       // nhân đôi dòng quảng cáo
    clrscr();                               // xoá màn hình
    char hien[31] ;                         // chứa xâu dài 30 kí tự để hiện
    i = 0;
    while (!kbhit()) {                      // trong khi chưa ấn phím bất kỳ
        strncpy(hien, s+i, 30);
        hien[30] = '\0';                    // copy 30 kí tự từ qc[i] sang hien
        gotoxy(20,10); cout << hien ;       // in hien tại dòng 10 cot 20
        delay(100);                         // tạm dừng 1/10 giây
        i++; if (i==dd) i = 0;              // tăng i
    }
}
```

Ví dụ 6 : Nhập mật khẩu (không quá 10 kí tự). In ra "đúng" nếu là "HaNoi2000", "sai" nếu ngược lại. Chương trình cho phép nhập tối đa 3 lần. Nhập riêng rẽ từng kí tự (bằng hàm getch()) cho mật khẩu. Hàm getch() không hiện kí tự NSD gõ vào, thay vào đó chương trình chỉ hiện kí tự 'X' để che giấu mật khẩu. Sau khi NSD đã gõ xong (9 kí tự) hoặc đã Enter, chương trình so sánh xâu vừa nhập với "HaNoi2000", nếu đúng chương trình tiếp tuc, nếu sai tăng số lần nhập (cho phép không quá 3 lần).

```
#include <iostream.h>
#include <conio.h>
```

```
#include <string.h>
void main()
{
        char pw[11]; int solan = 0;                 // Cho phep nhap 3 lan
        do {
                clrscr(); gotoxy(30,12) ;
                int i = 0;
                while ((pw[i]=getch()) != 13 && ++i < 10) cout << 'X' ; // 13 = Enter
                pw[i] = '\0' ;
                cout << endl ;
                if (!strcmp(pw, "HaNoi2000")) { cout << "Mời vào" ; break; }
                else { cout << "Sai mật khẩu. Nhập lại") ; solan++ ; }
        } while (solan < 3);
}
```

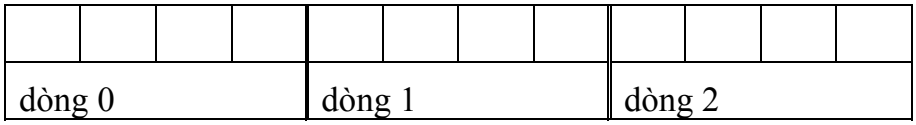
IV. MẢNG HAI CHIỀU

Để thuận tiện trong việc biểu diễn các loại dữ liệu phức tạp như ma trận hoặc các bảng biểu có nhiều chỉ tiêu, C++ đưa ra kiểu dữ liệu mảng nhiều chiều. Tuy nhiên, việc sử dụng mảng nhiều chiều rất khó lập trình vì vậy trong mục này chúng ta chỉ bàn đến mảng hai chiều. Đối với mảng một chiều m thành phần, nếu mỗi thành phần của nó lại là mảng một chiều n phần tử thì ta gọi mảng là hai chiều với số phần tử (hay kích thước) mỗi chiều là m và n. Ma trận là một minh hoạ cho hình ảnh của mảng hai chiều, nó gồm m dòng và n cột, tức chứa m x n phần tử, và hiển nhiên các phần tử này có cùng kiểu. Tuy nhiên, về mặt bản chất mảng hai chiều không phải là một tập hợp với m x n phần tử cùng kiểu mà là tập hợp với m thành phần, trong đó mỗi thành phần là một mảng một chiều với n phần tử. Điểm nhấn mạnh này sẽ được giải thích cụ thể hơn trong các phần trình bày về con trỏ của chương sau.

	0	1	2	3
0				
1				
2				

Hình trên minh hoạ hình thức một mảng hai chiều với 3 dòng, 4 cột. Thực chất

trong bộ nhớ tất cả 12 phần tử của mảng được sắp liên tiếp theo từng dòng của mảng như minh hoạ trong hình dưới đây.



a. Khai báo

<kiểu thành phần > <tên mảng>[m][n] ;

- m, n là số hàng, số cột của mảng.
- kiểu thành phần là kiểu của m x n phần tử trong mảng.
- Trong khai báo cũng có thể được khởi tạo bằng dãy các dòng giá trị, các dòng cách nhau bởi dấu phẩy, mỗi dòng được bao bởi cặp ngoặc {} và toàn bộ giá trị khởi tạo nằm trong cặp dấu {}.

b. Sử dụng

- Tương tự mảng một chiều các chiều trong mảng cũng được đánh số từ 0.
- Không sử dụng các thao tác trên toàn bộ mảng mà phải thực hiện thông qua từng phần tử của mảng.
- Để truy nhập phần tử của mảng ta sử dụng tên mảng kèm theo 2 chỉ số chỉ vị trí hàng và cột của phần tử. Các chỉ số này có thể là các biểu thức thực, khi đó C++ sẽ tự chuyển kiểu sang nguyên.

Ví dụ:

- Khai báo 2 ma trận 4 hàng 5 cột A, B chứa các số nguyên:

```
int A[3][4], B[3][4] ;
```

- Khai báo có khởi tạo:

```
int A[3][4] = { {1,2,3,4}, {3,2,1,4}, {0,1,1,0} };
```

với khởi tạo này ta có ma trận:

1	2	3	4
3	2	1	4
0	1	1	0

trong đó: A[0][0] = 1, A[0][1] = 2, A[1][0] = 3, A[2][3] = 0 …

– Trong khai báo có thể vắng số hàng (không được vắng số cột), số hàng này được xác định thông qua khởi tạo.

```
float A[][3] = { {1,2,3}, {0,1,0} } ;
```

trong khai báo này chương trình tự động xác định số hàng là 2.

– Phép khai báo và khởi tạo sau đây là cũng hợp lệ:

```
float A[][3] = { {1,2}, {0} } ;
```

chương trình cũng xác định số hàng là 2 và số cột (bắt buộc phải khai báo) là 3 mặc dù trong khởi tạo không thể xác định được số cột. Các phần tử chưa khởi tạo sẽ chưa được xác định cho đến khi nào nó được nhập hoặc gán giá trị cụ thể. Trong ví dụ trên các phần tử A[0][2], A[1][1] và A[1][2] là chưa được xác định.

c. Ví dụ minh hoạ

Ví dụ 1 : Nhập, in và tìm phần tử lớn nhất của một ma trận.

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
main()
{
    float a[10][10] ;
    int m, n ;                          // số hàng, cột của ma trận
    int i, j ;                          // các chỉ số trong vòng lặp
    int amax, imax, jmax ;              // số lớn nhất và chỉ số của nó
    clrscr(); cout << "Nhập số hàng và cột: " ; cin >> m >> n ;
    for (i=0; i<m; i++)
    for (j=0; j<n; j++)
    {
        cout << "a[" << i << "," << j << "] = " ; cin >> a[i][j] ;
    }
    amax = a[0][0]; imax = 0; jmax = 0;
    for (i=0; i<m; i++)
    for (j=0; j<n; j++)
    if (amax < a[i][j])
```

```
        {
                amax = a[i][j]; imax = i; jmax = j;
        }
        cout << "Ma trận đã nhập\n" ;
        cout << setiosflags(ios::showpoint) << setprecision(1) ;
        for (i=0; i<m; i++)
        for (j=0; j<n; j++)
        {
                if (j==0) cout << endl;
                cout << setw(6) << a[i][j] ;
        }
        cout << "Số lớn nhất là " << setw(6) << amax << endl;
        cout << "tại vị trí (" << imax << "," << jmax << ")" ;
        getch();
}
```

Ghi chú: Khi làm việc với mảng (1 chiều, 2 chiều) do thói quen chúng ta thường tính chỉ số từ 1 (thay vì 0), do vậy trong mảng ta có thể bỏ qua hàng 0, cột 0 bằng cách khai báo số hàng và cột tăng lên 1 so với số hàng, cột thực tế của mảng và từ đó có thể làm việc từ hàng 1, cột 1 trở đi.

Ví dụ 2 : Nhân 2 ma trận. Cho 2 ma trận A (m x n) và B (n x p). Tính ma trận C = A x B, trong đó C có kích thước là m x p. Ta lập vòng lặp tính từng phần tử của C. Giá trị của phần tử C tại hàng i, cột j chính là tích vô hướng của hàng i ma trận A với cột j ma trận B. Để tránh nhầm lẫn ta qui ước bỏ các hàng, cột 0 của các ma trận A, B, C (tức các chỉ số được tính từ 1 trở đi).

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
main()
{
        float A[10][10], B[10], C[10][10] ;
        int m, n, p ;                           // số hàng, cột của ma trận
        int i, j, k ;                           // các chỉ số trong vòng lặp
```

```
        clrscr();
        cout << "Nhập số hàng và cột của 2 ma trận: " ; cin >> m >> n >> p;
        // Nhập ma trận A
        for (i=1; i<=m; i++)
        for (j=1; j<=n; j++)
        {
                cout << "A[" << i << "," << j << "] = " ; cin >> A[i][j] ;
        }
        // Nhập ma trận B
        for (i=1; i<=n; i++)
        for (j=1; j<=p; j++)
        {
                cout << "B[" << i << "," << j << "] = " ; cin >> B[i][j] ;
        }
        // Tính ma trận C = A x B
        for (i=1; i<=m; i++)
        for (j=1; j<=p; j++)
        {
                C[i][j] = 0; for (k=1; k<=n; k++) C[i][j] += A[i][k]*B[k][j] ;
        }
        // In kết quả
        cout << "Ma trận kết quả\n" ;
        cout << setiosflags(ios::showpoint) << setprecision(2) ;
        for (i=1; i<m; i++)
        for (j=1; j<n; j++)
        {
                if (j==1) cout << endl;
                cout << setw(6) << a[i][j] ;
        }
        getch();
}
```

BÀI TẬP

Lệnh rẽ nhánh

1. Nhập một kí tự. Cho biết kí tự đó có phải là chữ cái hay không.

2. Nhập vào một số nguyên. Trả lời số nguyên đó: âm hay dương, chẵn hay lẻ ?

3. Cho n = x = y và bằng: a. 1 b. 2 c. 3 d. 4

Hãy cho biết giá trị của x, y sau khi chạy xong câu lệnh:

```
if (n % 2 == 0) if (x > 3) x = 0;
else y = 0;
```

4. Tính giá trị hàm

a. $f(x) = \begin{cases} 3x+\sqrt{x} & , x>0 \\ e^x+4 & , x\le 0 \end{cases}$

b. $f(x) = \begin{cases} \sqrt{x^2+1} & , x\ge 1 \\ 3x+5 & , -1<x<1 \\ x^2+2x-1 & , x\le -1 \end{cases}$

5. Viết chương trình giải hệ phương trình bậc nhất 2 ẩn: $\begin{cases} ax+by=c \\ dx+ey=f \end{cases}$

6. Nhập 2 số a, b. In ra max, min của 2 số đó. Mở rộng với 3 số, 4 số ?

7. Nhập 3 số a, b, c. Hãy cho biết 3 số trên có thể là độ dài 3 cạnh của một tam giác ? Nếu là một tam giác thì đó là tam giác gì: vuông, đều, cân, vuông cân hay tam giác thường ?

8. Nhập vào một số, in ra thứ tương ứng với số đó (qui ước 2 là thứ hai, …, 8 là chủ nhật).

9. Nhập 2 số biểu thị tháng và năm. In ra số ngày của tháng năm đó (có kiểm tra năm nhuận).

10. Lấy ngày tháng hiện tại làm chuẩn. Hãy nhập một ngày bất kỳ trong tháng. Cho biết thứ của ngày vừa nhập ?

Lệnh lặp

11. Giá trị của i bằng bao nhiêu sau khi thực hiện cấu trúc for sau:

```
for (i = 0; i < 100; i++);
```

12. Giá trị của x bằng bao nhiêu sau khi thực hiện cấu trúc for sau:

```
for (x = 2; i < 10; x+=3) ;
```

13. Bạn bổ sung gì vào lệnh for sau:

```
for ( ; nam < 1997 ; ) ;
```

để khi kết thúc nam có giá trị 2000.

14. Bao nhiêu kí tự 'X' được in ra màn hình khi thực hiện đoạn chương trình sau:

```
for (x = 0; x < 10; x ++) for (y = 5; y > 0; y --) cout << 'X';
```

15. Nhập vào tuổi cha và tuổi con hiện nay sao cho tuổi cha lớn hơn 2 lần tuổi con. Tìm xem bao nhiêu năm nữa tuổi cha sẽ bằng đúng 2 lần tuổi con (ví dụ 30 và 12, sau 6 năm nữa tuổi cha là 36 gấp đôi tuổi con là 18).

16. Nhập số nguyên dương N. Tính:

a. $S_1 = \dfrac{1+2+3+...+N}{N}$

b. $S_2 = \sqrt{1^2+2^2+3^2+...+N^2}$

17. Nhập số nguyên dương n. Tính:

a. $S_1 = \sqrt{3+\sqrt{3+\sqrt{3+...+\sqrt{3}}}}$ n dấu căn

b. $S_2 = \dfrac{1}{2+\dfrac{1}{2+\dfrac{1}{2+...\dfrac{1}{2}}}}$ n dấu chia

18. Nhập số tự nhiên n. In ra màn hình biểu diễn của n ở dạng nhị phân.

19. In ra màn hình các số có 2 chữ số sao cho tích của 2 chữ số này bằng 2 lần tổng của 2 chữ số đó (ví dụ số 36 có tích 3*6 = 18 gấp 2 lần tổng của nó là 3 + 6 = 9).

20. Số *hoàn chỉnh* là số bằng tổng mọi ước của nó (không kể chính nó). Ví dụ 6 = 1 + 2 + 3 là một số hoàn chỉnh. Hãy in ra màn hình tất cả các số hoàn chỉnh < 1000.

21. Các số *sinh đôi* là các số nguyên tố mà khoảng cách giữa chúng là 2. Hãy in tất cả cặp số sinh đôi < 1000.

22. Nhập dãy kí tự đến khi gặp kí tự '.' thì dừng. Thống kê số chữ cái viết hoa, viết thường, số chữ số và tổng số các kí tự khác đã nhập. Loại kí tự nào nhiều nhất ?

23. Tìm số nguyên dương n lớn nhất thoả mãn điều kiện:

a. $1+\frac{1}{3}+\frac{1}{5}+\ldots+\frac{1}{2n-1} < 2.101999$.

b. $e^n - 1999 log_{10} n < 2000$.

24. Cho ε = 1e–6. Tính gần đúng các số sau:

a. Số pi theo công thức Euler: $\frac{\pi^2}{6}=\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\ldots+\frac{1}{n^2}$ dừng lặp khi $\frac{1}{n^2}<10^{-6}$.

b. e^x theo công thức: $e^x = 1+\frac{x^1}{1!}+\frac{x^2}{2!}+\ldots+\frac{x^n}{n!}$ dừng lặp khi $\left|\frac{x^n}{n!}\right| < 10^{-6}$.

c. $sin\, x = x-\frac{x^3}{3!}+\frac{x^5}{5!}+\ldots+(-1)^n \frac{x^{2n+1}}{(2n+1)!}$, dừng lặp khi $\left|\frac{x^{2n+1}}{(2n+1)!}\right| < 10^{-6}$.

d. $\sqrt{a}\ (a>0)$ theo công thức: $s_n = \begin{cases} a & n=0 \\ (s_{n-1}^2+a)/2s_{n-1} & n>0 \end{cases}$, dừng khi $|s_n - s_{n-1}|<10^{-6}$.

25. In ra mã của phím bất kỳ được nhấn. Chương trình lặp cho đến khi nhấn ESC để thoát.

26. Bằng phương pháp chia đôi, hãy tìm nghiệm xấp xỉ (độ chính xác 10^{-6}) của các phương trình sau:

a. $e^x - 1.5 = 0$, trên đoạn [0, 1].

b. $x2^x - 1 = 0$, trên đoạn [0, 1].

c. $a_0x^n + a_1x^{n-1} + \ldots + a_n = 0$, trên đoạn [a, b]. Các số thực a_i, a, b được nhập từ bàn phím sao cho f(a) và f(b) trái dấu.

Mảng

27. Nhập vào dãy n số thực. Tính tổng dãy, trung bình dãy, tổng các số âm, dương và tổng các số ở vị trí chẵn, vị trí lẻ trong dãy. Tìm phần tử gần số trung bình nhất

của dãy.

28. Tìm và chỉ ra vị trí xuất hiện đầu tiên của phần tử x trong dãy.

29. Nhập vào dãy n số. Hãy in ra số lớn nhất, bé nhất của dãy.

30. Nhập vào dãy số. In ra dãy đã được sắp xếp tăng dần, giảm dần.

31. Cho dãy đã được sắp tăng dần. Chèn thêm vào dãy phần tử x sao cho dãy vẫn sắp xếp tăng dần.

32. Hãy nhập vào 16 số nguyên. In ra thành 4 dòng, 4 cột.

33. Nhập ma trận A và in ra ma trận đối xứng của nó.

34. Cho một ma trận nguyên kích thước m*n. Tính:

- Tổng tất cả các phần tử của ma trận.
- Tổng tất cả các phần tử dương của ma trận.
- Tổng tất cả các phần tử âm của ma trận.
- Tổng tất cả các phần tử chẵn của ma trận.
- Tổng tất cả các phần tử lẻ của ma trận.

35. Cho một ma trận thực kích thước m*n. Tìm:

- Số nhỏ nhất, lớn nhất (kèm chỉ số) của ma trận.
- Số nhỏ nhất, lớn nhất (kèm chỉ số) của từng hàng của ma trận.
- Số nhỏ nhất, lớn nhất (kèm chỉ số) của từng cột của ma trận.
- Số nhỏ nhất, lớn nhất (kèm chỉ số) của đường chéo chính của ma trận.
- Số nhỏ nhất, lớn nhất (kèm chỉ số) của đường chéo phụ của ma trận.

36. Nhập 2 ma trận vuông cấp n A và B. Tính A + B, A – B, A * B và A^2 - B^2.

Xâu kí tự

37. Hãy nhập một xâu kí tự. In ra màn hình đảo ngược của xâu đó.

38. Nhập xâu. Thống kê số các chữ số '0', số chữ số '1', …, số chữ số '9' trong xâu.

39. In ra vị trí kí tự trắng đầu tiên từ bên trái (phải) một xâu kí tự.

40. Nhập xâu. In ra tất các các vị trí của chữ 'a' trong xâu và tổng số lần xuất hiện của nó.

41. Nhập xâu. Tính số từ có trong xâu. In mỗi dòng một từ.

42. Nhập xâu họ tên, in ra họ, tên dưới dạng viết hoa.

43. Thay kí tự x trong xâu s bởi kí tự y (s, x, y được đọc vào từ bàn phím)

44. Xoá mọi kí tự x có trong xâu s (s, x được đọc vào từ bàn phím). (Gợi ý: nên xoá ngược từ cuối xâu về đầu xâu).

45. Nhập xâu. Không phân biệt viết hoa hay viết thường, hãy in ra các kí tự có mặt trong xâu và số lần xuất hiện của nó (ví dụ xâu "Trach – Van – Doanh" có chữ a xuất hiện 3 lần, c(1), d(1), h(2), n(2), o(1), r(1), t(1), –(2), space(4)).

CHƯƠNG 4

HÀM VÀ CHƯƠNG TRÌNH

Con trỏ và số học địa chỉ
Hàm
Đệ qui
Tổ chức chương trình

I. CON TRỎ VÀ SỐ HỌC ĐỊA CHỈ

Trước khi bàn về hàm và chương trình, trong phần này chúng ta sẽ nói về một loại biến mới gọi là con trỏ, ý nghĩa, công dụng và sử dụng nó như thế nào. Biến con trỏ là một đặc trưng mạnh của C++, nó cho phép chúng ta thâm nhập trực tiếp vào bộ nhớ để xử lý các bài toán khó bằng chỉ vài câu lệnh đơn giản của chương trình. Điều này cũng góp phần làm cho C++ trở thành ngôn ngữ gần gũi với các ngôn ngữ cấp thấp như hợp ngữ. Tuy nhiên, vì tính đơn giản, ngắn gọn nên việc sử dụng con trỏ đòi hỏi tính cẩn thận cao và giàu kinh nghiệm của người lập trình.

1. Địa chỉ, phép toán &

Mọi chương trình trước khi chạy đều phải bố trí các biến do NSD khai báo vào đâu đó trong bộ nhớ. Để tạo điều kiện truy nhập dễ dàng trở lại các biến này, bộ nhớ được đánh số, mỗi byte sẽ được ứng với một số nguyên, được gọi là địa chỉ của byte đó từ 0 đến hết bộ nhớ. Từ đó, mỗi biến (với tên biến) được gắn với một số nguyên là địa chỉ của byte đầu tiên mà biến đó được phân phối. Số lượng các byte phân phối cho biến là khác nhau (nhưng đặt liền nhau từ thấp đến cao) tuỳ thuộc kiểu dữ liệu của biến (và tuỳ thuộc vào quan niệm của từng NNLT), tuy nhiên chỉ cần biết tên biến hoặc địa chỉ của biến ta có thể đọc/viết dữ liệu vào/ra các biến đó. Từ đó ngoài việc thông qua tên biến chúng ta còn có thể thông qua địa chỉ của chúng để truy nhập vào nội dung. Tóm lại biến, ô nhớ và địa chỉ có quan hệ khăng khít với nhau. C++ cung cấp một toán tử một ngôi & để lấy địa chỉ của các biến (ngoại trừ biến mảng và xâu kí tự). Nếu x là một biến thì &x là địa chỉ của x. Từ đó câu lệnh sau cho ta biết x được bố trí ở đâu trong bộ nhớ:

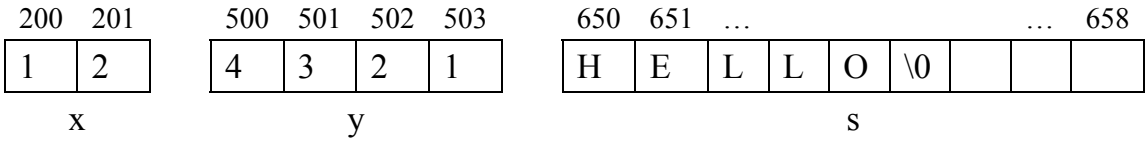
```
int x ;
cout << &x ;          // địa chỉ sẽ được hiện dưới dạng cơ số 16. Ví dụ 0xfff4
```

Đối với biến kiểu mảng, thì tên mảng chính là địa chỉ của mảng, do đó không cần dùng đến toán tử &. Ví dụ địa chỉ của mảng a chính là a (không phải &a). Mặt khác địa chỉ của mảng a cũng chính là địa chỉ của byte đầu tiên mà mảng a chiếm và nó cũng chính là địa chỉ của phần tử đầu tiên của mảng a. Do vậy địa chỉ của mảng a là địa chỉ của phần tử a[0] tức &a[0]. Tóm lại, địa chỉ của mảng a là a hoặc &a[0].

Tóm lại, cần nhớ:

```
int x;                  // khai báo biến nguyên x
long y;                 // khai báo biến nguyên dài y
cout << &x << &y;       // in địa chỉ các biến x, y
char s[9];              // khai báo mảng kí tự s
cout << a;              // in địa chỉ mảng s
cout << &a[0];          // in địa chỉ mảng s (tức địa chỉ s[0])
cout << &a[2];          // in địa chỉ kí tự s[2]
```

Hình vẽ sau đây minh hoạ một vài biến và địa chỉ của nó trong bộ nhớ.



Biến x chiếm 2 byte nhớ, có địa chỉ là 200, biến y có địa chỉ là 500 và chiếm 4 byte nhớ. Xâu s chiếm 9 byte nhớ tại địa chỉ 650. Các byte nhớ của một biến là liền nhau.

Các phép toán liên quan đến địa chỉ được gọi là số học địa chỉ. Tuy nhiên, chúng ta vẫn không được phép thao tác trực tiếp trên các địa chỉ như đặt biến vào địa chỉ này hay khác (công việc này do chương trình dịch đảm nhiệm), hay việc cộng, trừ hai địa chỉ với nhau là vô nghĩa … Các thao tác được phép trên địa chỉ vẫn phải thông qua các biến trung gian chứa địa chỉ, được gọi là biến con trỏ.

2. Con trỏ

a. Ý nghĩa

- Con trỏ là một biến chứa địa chỉ của biến khác. Nếu p là con trỏ chứa địa chỉ của biến x ta gọi p trỏ tới x và x được trỏ bởi p. Thông qua con trỏ ta có thể làm việc được với nội dung của những ô nhớ mà p trỏ đến.
- Để con trỏ p trỏ tới x ta phải gán địa chỉ của x cho p.

– Để làm việc với địa chỉ của các biến cần phải thông qua các biến con trỏ trỏ đến biến đó.

b. Khai báo biến con trỏ

<kiểu được trỏ> <*tên biến> ;

Địa chỉ của một biến là địa chỉ byte nhớ đầu tiên của biến đó. Vì vậy để lấy được nội dung của biến, con trỏ phải biết được số byte của biến, tức kiểu của biến mà con trỏ sẽ trỏ tới. Kiểu này cũng được gọi là kiểu của con trỏ. Như vậy khai báo biến con trỏ cũng giống như khai báo một biến thường ngoại trừ cần thêm dấu * trước tên biến (hoặc sau tên kiểu). Ví dụ:

```
int *p ;          // khai báo biến p là biến con trỏ trỏ đến kiểu dữ liệu nguyên.
float *q, *r ;    // hai con trỏ thực q và r.
```

*c. Sử dụng con trỏ, phép toán **

- Để con trỏ p trỏ đến biến x ta phải dùng phép gán p = địa chỉ của x.
  - Nếu x không phải là mảng ta viết: p = &x.
  - Nếu x là mảng ta viết: p = x hoặc p = &x[0].
- Không gán p cho một hằng địa chỉ cụ thể. Ví dụ viết p = 200 là sai.
- Phép toán * cho phép lấy nội dung nơi p trỏ đến, ví dụ để gán nội dung nơi p trỏ đến cho biến f ta viết f = *p.
- & và * là 2 phép toán ngược nhau. Cụ thể nếu p = &x thì x = *p. Từ đó nếu p trỏ đến x thì bất kỳ nơi nào xuất hiện x đều có thể thay được bởi *p và ngược lại.

Ví dụ 1 :

```
int i, j ;                  // khai báo 2 biến nguyên i, j
int *p, *q ;                // khai báo 2 con trỏ nguyên p, q
p = &i;                     // cho p trỏ tới i
q = &j;                     // cho q trỏ tới j
cout << &i ;                // hỏi địa chỉ biến i
cout << q ;                 // hỏi địa chỉ biến j (thông qua q)
i = 2;                      // gán i bằng 2
*q = 5;                     // gán j bằng 5 (thông qua q)
i++ ; cout << i ;           // tăng i và hỏi i, i = 3
```

```
(*q)++ ; cout << j ;          // tăng j (thông qua q) và hỏi j, j = 6
(*p) = (*q) * 2 + 1;          // gán lại i (thông qua p)
cout << i ;                   // 13
```

Qua ví dụ trên ta thấy mọi thao tác với i là tương đương với *p, với j là tương đương với *q và ngược lại.

3. Các phép toán với con trỏ

Trên đây ta đã trình bày về 2 phép toán một ngôi liên quan đến địa chỉ và con trỏ là & và *. Phần này chúng ta tiếp tục xét với các phép toán khác làm việc với con trỏ.

a. Phép toán gán

- Gán con trỏ với địa chỉ một biến: p = &x ;
- Gán con trỏ với con trỏ khác: p = q ; (sau phép toán gán này p, q chứa cùng một địa chỉ, cùng trỏ đến một nơi).

Ví dụ 2 :

```
int i = 10 ;                  // khai báo và khởi tạo biến i = 10
int *p, *q, *r ;              // khai báo 3 con trỏ nguyên p, q, r
p = q = r = &i ;              // cùng trỏ tới i
*p = q**q + 2**r + 1 ;        // i = 10*10 + 2*10 + 1
cout << i ;                   // 121
```

b. Phép toán tăng giảm địa chỉ

p ± n: con trỏ trỏ đến thành phần thứ n sau (trước) p.

Một đơn vị tăng giảm của con trỏ bằng kích thước của biến được trỏ. Ví dụ giả sử p là con trỏ nguyên (2 byte) đang trỏ đến địa chỉ 200 thì p+1 là con trỏ trỏ đến địa chỉ 202. Tương tự, p + 5 là con trỏ trỏ đến địa chỉ 210. p – 3 chứa địa chỉ 194.

194	195	196	197	198	199	200	201	202
p – 3						p		p + 1

Như vậy, phép toán tăng, giảm con trỏ cho phép làm việc thuận lợi trên mảng. Nếu con trỏ đang trỏ đến mảng (tức đang chứa địa chỉ đầu tiên của mảng), việc tăng con trỏ lên 1 đơn vị sẽ dịch chuyển con trỏ trỏ đến phần tử thứ hai, … Từ đó ta có thể cho con trỏ chạy từ đầu đến cuối mảng bằng cách tăng con trỏ lên từng đơn vị như trong câu lệnh for dưới đây.

Ví dụ 3 :

```
int a[100] = { 1, 2, 3, 4, 5, 6, 7 }, *p, *q;
p = a; cout << *p ;                    // cho p trỏ đến mảng a, *p = a[0] = 1
p += 5; cout << *p ;                                  // *p = a[5] = 6 ;
q = p - 4 ; cout << *q ;                              // q = a[1] = 2 ;
for (int i=0; i<100; i++) cout << *(p+i) ;            // in toàn bộ mảng a
```

c. Phép toán tự tăng giảm

p++, p--, ++p, --p: tương tự p+1 và p-1, có chú ý đến tăng (giảm) trước, sau.

Ví dụ 4 : Ví dụ sau minh hoạ kết quả kết hợp phép tự tăng giảm với lấy giá trị nơi con trỏ trỏ đến. a là một mảng gồm 2 số, p là con trỏ trỏ đến mảng a. Các lệnh dưới đây được qui ước là độc lập với nhau (tức lệnh sau không bị ảnh hưởng bởi lệnh trước, đối với mỗi lệnh p luôn luôn trỏ đến phần tử đầu (a[0]) của a.

```
int a[2] = {3, 7}, *p = a;
(*p)++ ;        // tăng (sau) giá trị nơi p trỏ ≡ tăng a[0] thành 4
++(*p) ;        // tăng (trước) giá trị nơi p trỏ ≡ tăng a[0] thành 4
*(p++) ;        // lấy giá trị nơi p trỏ (3) và tăng trỏ p (tăng sau), p → a[1]
*(++p) ;        // tăng trỏ p (tăng trước), p → a[1] và lấy giá trị nơi p trỏ (7)
```

Chú ý:

- Phân biệt p+1 và p++ (hoặc ++p):
- p+1 được xem như một con trỏ khác với p. p+1 trỏ đến phần tử sau p.
- p++ là con trỏ p nhưng trỏ đến phần tử khác. p++ trỏ đến phần tử đứng sau phần tử p trỏ đến ban đầu.
- Phân biệt *(p++) và *(++p):

Các phép toán tự tăng giảm cũng là một ngôi, mức ưu tiên của chúng là cao hơn các phép toán hai ngôi khác và cao hơn phép lấy giá trị (*). Cụ thể:

```
*p++        ≡    *(p++)
*++p        ≡    *(++p)
++*p        ≡    ++(*p)
```

Cũng giống các biến nguyên việc kết hợp các phép toán này với nhau rất dễ gây nhầm lẫn, do vậy cần sử dụng cặp dấu ngoặc để qui định trình tự tính toán.

d. Hiệu của 2 con trỏ

Phép toán này chỉ thực hiện được khi p và q là 2 con trỏ cùng trỏ đến các phần tử của một dãy dữ liệu nào đó trong bộ nhớ (ví dụ cùng trỏ đến 1 mảng dữ liệu). Khi đó hiệu p - q là số thành phần giữa p và q (chú ý p - q không phải là hiệu của 2 địa chỉ mà là số thành phần giữa p và q).

Ví dụ: giả sử p và q là 2 con trỏ nguyên, p có địa chỉ 200 và q có địa chỉ 208. Khi đó p - q = –4 và q - p = 4 (4 là số thành phần nguyên từ địa chỉ 200 đến 208).

e. Phép toán so sánh

Các phép toán so sánh cũng được áp dụng đối với con trỏ, thực chất là so sánh giữa địa chỉ của hai nơi được trỏ bởi các con trỏ này. Thông thường các phép so sánh <, <=, >, >= chỉ áp dụng cho hai con trỏ trỏ đến phần tử của cùng một mảng dữ liệu nào đó. Thực chất của phép so sánh này chính là so sánh chỉ số của 2 phần tử được trỏ bởi 2 con trỏ đó.

Ví dụ 5 :

```
float a[100], *p, *q ;
p = a ;                                  // p trỏ đến mảng (tức p trỏ đến a[0])
q = &a[3] ;                              // q trỏ đến phần tử thứ 3 (a[3]) của mảng
cout << (p < q) ;                        // 1
cout << (p + 3 == q) ;                   // 1
cout << (p > q - 1) ;                    // 0
cout << (p >= q - 2) ;                   // 0
for (p=a ; p < a+100; p++) cout << *p ;      // in toàn bộ mảng a
```

4. Cấp phát động, toán tử cấp phát, thu hồi new, delete

Khi tiến hành chạy chương trình, chương trình dịch sẽ bố trí các ô nhớ cụ thể cho các biến được khai báo trong chương trình. Vị trí cũng như số lượng các ô nhớ này tồn tại và cố định trong suốt thời gian chạy chương trình, chúng xem như đã bị chiếm dụng và sẽ không được sử dụng vào mục đích khác và chỉ được giải phóng sau khi chấm dứt chương trình. Việc phân bổ bộ nhớ như vậy được gọi là cấp phát tĩnh (vì được cấp sẵn trước khi chạy chương trình và không thể thay đổi tăng, giảm kích thước hoặc vị trí trong suốt quá trình chạy chương trình). Ví dụ nếu ta khai báo một mảng nguyên chứa 1000 số thì trong bộ nhớ sẽ có một vùng nhớ liên tục 2000 bytes để chứa dữ liệu của mảng này. Khi đó dù trong chương trình ta chỉ nhập vào mảng và làm việc với một vài số thì phần mảng rỗi còn lại vẫn không được sử dụng vào việc khác. Đây là hạn chế thứ nhất của kiểu mảng. Ở một hướng khác, một lần nào đó chạy chương trình ta lại

cần làm việc với hơn 1000 số nguyên. Khi đó vùng nhớ mà chương trình dịch đã dành cho mảng là không đủ để sử dụng. Đây chính là hạn chế thứ hai của mảng được khai báo trước.

Khắc phục các hạn chế trên của kiểu mảng, bây giờ chúng ta sẽ không khai báo (bố trí) trước mảng dữ liệu với kích thước cố định như vậy. Kích thước cụ thể sẽ được cấp phát trong quá trình chạy chương trình theo đúng yêu cầu của NSD. Nhờ vậy chúng ta có đủ số ô nhớ để làm việc mà vẫn tiết kiệm được bộ nhớ, và khi không dùng nữa ta có thể thu hồi (còn gọi là giải phóng) số ô nhớ này để chương trình sử dụng vào việc khác. Hai công việc cấp phát và thu hồi này được thực hiện thông qua các toán tử new, delete và con trỏ p. Thông qua p ta có thể làm việc với bất kỳ địa chỉ nào của vùng được cấp phát. Cách thức bố trí bộ nhớ như thế này được gọi là cấp phát động. Sau đây là cú pháp của câu lệnh new.

```
p = new <kiểu> ;                    // cấp phát 1 phần tử
p = new <kiểu>[n] ;                 // cấp phát n phần tử
```

Ví dụ:

```
int *p ;
p = new int       ;          // cấp phát vùng nhớ chứa được 1 số nguyên
p = float int[100] ;         // cấp phát vùng nhớ chứa được 100 số thực
```

Khi gặp toán tử new, chương trình sẽ tìm trong bộ nhớ một lượng ô nhớ còn rỗi và liên tục với số lượng đủ theo yêu cầu và cho p trỏ đến địa chỉ (byte đầu tiên) của vùng nhớ này. Nếu không có vùng nhớ với số lượng như vậy thì việc cấp phát là thất bại và p = NULL (NULL là một địa chỉ rỗng, không xác định). Do vậy ta có thể kiểm tra việc cấp phát có thành công hay không thông qua kiểm tra con trỏ p bằng hay khác NULL. Ví dụ:

```
float *p ;
int n ;
cout << "Số lượng cần cấp phát = "; cin >> n;
p = new double[n];
if (p == NULL) {
      cout << "Không đủ bộ nhớ" ;
      exit(0) ;
}
```

Ghi chú: lệnh exit(0) cho phép thoát khỏi chương trình, để sử dụng lệnh này cần khai báo file tiêu đề <process.h>.

Để giải phóng bộ nhớ đã cấp phát cho một biến (khi không cần sử dụng nữa) ta sử dụng câu lệnh delete.

delete p ; // p là con trỏ được sử dụng trong new

và để giải phóng toàn bộ mảng được cấp pháp thông qua con trỏ p ta dùng câu lệnh:

delete[] p ; // p là con trỏ trỏ đến mảng

Dưới đây là ví dụ sử dụng tổng hợp các phép toán trên con trỏ.

Ví dụ 1 : Nhập dãy số (không dùng mảng). Sắp xếp và in ra màn hình.

Trong ví dụ này chương trình xin cấp phát bộ nhớ đủ chứa n số nguyên và được trỏ bởi con trỏ head. Khi đó địa chỉ của số nguyên đầu tiên và cuối cùng sẽ là head và head+n-1. p và q là 2 con trỏ chạy trên dãy số này, so sánh và đổi nội dung của các số này với nhau để sắp thành dãy tăng dần và cuối cùng in kết quả.

```
main()
{
    int *head, *p, *q, n, tam;          // head trỏ đến (đánh dấu) đầu dãy
    cout << "Cho biết số số hạng của dãy: "); cin >> n ;
    head = new int[n] ;                 // cấp phát bộ nhớ chứa n số nguyên
    for (p=head; p<head+n; p++)         // nhập dãy
    {
        cout << "So thu " << p-head+1 << ": " ; cin >> *p ;
    }
    for (p=head; p<head+n-1; p++)                       // sắp xếp
    for (q=p+1; q<head+n; q++)
        if (*q < *p) { tam = *p; *p = *q; *q = tam; }   // đổi chỗ
    for (p=head; p<head+n; p++) cout << *p ;            // in kết quả
}
```

5. Con trỏ và mảng, xâu kí tự

a. Con trỏ và mảng 1 chiều

Việc cho con trỏ trỏ đến mảng cũng tương tự trỏ đến các biến khác, tức gán địa chỉ của mảng (chính là tên mảng) cho con trỏ. Chú ý rằng địa chỉ của mảng cũng là địa chỉ của thành phần thứ 0 nên a+i sẽ là địa chỉ thành phần thứ i của mảng. Tương tự, nếu p trỏ đến mảng a thì p+i là địa chỉ thành phần thứ i của mảng a và do đó *(p+i) =

a[i] = *(a+i).

Chú ý khi viết *(p+1) = *(a+1) ta thấy vai trò của p và a trong biểu thức này là như nhau, cùng truy cập đến giá trị của phần tử a[1]. Tuy nhiên khi viết *(p++) thì lại khác với *(a++), cụ thể viết p++ là hợp lệ còn a++ là không được phép. Lý do là tuy p và a cùng thể hiện địa chỉ của mảng a nhưng p thực sự là một biến, nó có thể thay đổi được giá trị còn a là một hằng, giá trị không được phép thay đổi. Ví dụ viết x = 3 và sau đó có thể tăng x bởi x++ nhưng không thể viết x = 3++.

Ví dụ 1 : In toàn bộ mảng thông qua con trỏ.

```
int a[5] = {1,2,3,4,5}, *p, i;
1:  p = a; for (i=1; i<=5; i++) cout << *(p+i);          // p không thay đổi
```

hoặc:

```
2: for (p=a; p<=a+4; p++) cout << *p ;                  // thay đổi p
```

Trong phương án 1, con trỏ p không thay đổi trong suốt quá trình làm việc của lệnh for, để truy nhập đến phần tử thứ i của mảng a ta sử dụng cú pháp *(p+i).

Đối với phương án 2 con trỏ sẽ dịch chuyển dọc theo mảng a bắt đầu từ địa chỉ a (phần tử đầu tiên) đến phần tử cuối cùng. Tại bước thứ i, p sẽ trỏ vào phần tử a[i], do đó ta chỉ cần in giá trị *p. Để kiểm tra khi nào p đạt đến phần tử cuối cùng, ta có thể so sánh p với địa chỉ cuối mảng chính là địa chỉ đầu mảng cộng thêm số phần tử trong a và trừ 1 (tức a+4 trong ví dụ trên).

b. Con trỏ và xâu kí tự

Một con trỏ kí tự có thể xem như một biến xâu kí tự, trong đó xâu chính là tất cả các kí tự kể từ byte con trỏ trỏ đến cho đến byte '\0' gặp đầu tiên. Vì vậy ta có thể khai báo các xâu dưới dạng con trỏ kí tự như sau.

```
char *s ;
char *s = "Hello" ;
```

Các hàm trên xâu vẫn được sử dụng như khi ta khai báo nó dưới dạng mảng kí tự. Ngoài ra khác với mảng kí tự, ta được phép sử dụng phép gán cho 2 xâu dưới dạng con trỏ, ví dụ:

```
char *s, *t = "Tin học" ; s = t;            // thay cho hàm strcpy(s, t) ;
```

Thực chất phép gán trên chỉ là gán 2 con trỏ với nhau, nó cho phép s bây giờ cũng được trỏ đến nơi mà t trỏ (tức dãy kí tự "Tin học" đã bố trí sẵn trong bộ nhớ)

Khi khai báo xâu dạng con trỏ nó vẫn chưa có bộ nhớ cụ thể, vì vậy thông thường kèm theo khai báo ta cần phải xin cấp phát bộ nhớ cho xâu với độ dài cần thiết. Ví dụ:

```
char *s = new char[30], *t ;
```

```
strcpy(s, "Hello") ;          // trong trường hợp này không cần cấp phát bộ
t = s ;                       // nhớ cho t vì t và s cùng sử dụng chung vùng nhớ
```

nhưng:

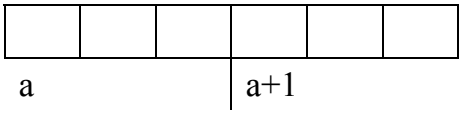
```
char *s = new char[30], *t ;
strcpy(s, "Hello") ;
t = new char[30];             // trong trường hợp này phải cấp bộ nhớ cho t vì
strcpy(t, s) ;                // có chỗ để strcpy sao chép sang nội dung của s.
```

c. Con trỏ và mảng hai chiều

Để dễ hiểu việc sử dụng con trỏ trỏ đến mảng hai chiều, chúng ta nhắc lại về mảng 2 chiều thông qua ví dụ. Giả sử ta có khai báo:

```
float a[2][3], *p;
```

khi đó a được bố trí trong bộ nhớ như là một dãy 6 phần tử float như sau



tuy nhiên a không được xem là mảng 1 chiều với 6 phần tử mà được quan niệm như mảng một chiều gồm 2 phần tử, mỗi phần tử là 1 bộ 3 số thực. Do đó địa chỉ của mảng a chính là địa chỉ của phần tử đầu tiên a[0][0], và a+1 không phải là địa chỉ của phần tử tiếp theo a[0][1] mà là địa chỉ của phần tử a[1][0]. Nói cách khác a+1 cũng là tăng địa chỉ của a lên một thành phần, nhưng 1 thành phần ở đây được hiểu là toàn bộ một dòng của mảng.

Mặt khác, việc lấy địa chỉ của từng phần tử (float) trong a thường là không chính xác. Ví dụ: viết &a[i][j] (địa chỉ của phần tử dòng i cột j) là được đối với mảng nguyên nhưng lại không đúng đối với mảng thực.

Từ các thảo luận trên, phép gán p = a là dễ gây nhầm lẫn vì p là con trỏ float còn a là địa chỉ mảng (1 chiều). Do vậy trước khi gán ta cần ép kiểu của a về kiểu float. Tóm lại cách gán địa chỉ của a cho con trỏ p được thực hiện như sau:

Cách sai:

```
p = a ;                       // sai vì khác kiểu
```

Các cách đúng:

```
p = (float*)a;                // ép kiểu của a về con trỏ float (cũng là kiểu của p)
p = a[0];                     // gán với địa chỉ của mảng a[0]
p = &a[0][0];                 // gán với địa chỉ số thực đầu tiên trong a
```

trong đó cách dùng p = (float*)a; là trực quan và đúng trong mọi trường hợp nên được dùng thông dụng hơn cả.

Sau khi gán a cho p (p là con trỏ thực), việc tăng giảm p chính là dịch chuyển con trỏ trên từng phần tử (thực) của a. Tức:

```
p      trỏ tới a[0][0]
p+1    trỏ tới a[0][1]
p+2    trỏ tới a[0][2]
p+3    trỏ tới a[1][0]
p+4    trỏ tới a[1][1]
p+5    trỏ tới a[1][2]
```

Tổng quát, đối với mảng m x n phần tử:

```
p + i*n + j trỏ tới a[i][j]      hoặc      a[i][j] = *(p + i*n + j)
```

Từ đó để truy nhập đến phần tử a[i][j] thông qua con trỏ p ta nên sử dụng cách viết sau:

```
p = (float*)a;
cin >> *(p+i*n+j) ;          // nhập cho a[i][j]
cout << *(p+i*n+j);          // in a[i][j]
```

Ví dụ sau đây cho phép nhập và in một mảng 2 chiều m*n (m dòng, n cột) thông qua con trỏ p. Nhập liên tiếp m*n số vào mảng và in thành ma trận m dòng, n cột.

```
main()
{
    clrscr();
    float a[m][n], *p;
    int i, j;
    p = (float*) a;
    for (i=0; i<m*n; i++) cin >> *(p+i);          // nhập như dãy mxn phần tử
    *(p+2*n+3) = 100; *(p+4*n) = 100;             // gán a[2,3] = a[4][0] = 100
    for (i=0; i<m; i++)                           // in lại dưới dạng ma trận
    {
        for (j=0; j<n; j++) cout << *(p+i*n+j);
        cout << endl;
```

```
        }
        getch();
}
```

Chú ý: việc lấy địa chỉ phần tử a[i][j] của mảng thực a là không chính xác. Tức: viết p = &a[i][j] có thể dẫn đến kết quả sai.

6. Mảng con trỏ

a. Khái niệm chung

Thực chất một con trỏ cũng là một biến thông thường có tên gọi (ví dụ p, q, …), do đó cũng giống như biến, nhiều biến cùng kiểu có thể tổ chức thành một mảng với tên gọi chung, ở đây cũng vậy nhiều con trỏ cùng kiểu cũng được tổ chức thành mảng. Như vậy mỗi phần tử của mảng con trỏ là một con trỏ trỏ đến một mảng nào đó. Nói cách khác một mảng con trỏ cho phép quản lý nhiều mảng dữ liệu cùng kiểu. Cách khai báo:

<kiểu> *a[size];

Ví dụ:

```
int *a[10];
```

khai báo một mảng chứa 10 con trỏ. Mỗi con trỏ a[i] chứa địa chỉ của một mảng nguyên nào đó.

b. Mảng xâu kí tự

Là trường hợp riêng của mảng con trỏ nói chung, trong đó kiểu cụ thể là char. Mỗi thành phần mảng là một con trỏ trỏ đến một xâu kí tự, có nghĩa các thao tác tiến hành trên *a[i] như đối với một xâu kí tự.

Ví dụ 1 : Nhập vào và in ra một bài thơ.

```
main()
{
        clrscr();
        char *dong[100];                    // khai báo 100 con trỏ kí tự (100 dòng)
        int i, n;
        cout << "so dong = "; cin >> n ;  // nhập số dòng thực sự
        cin.ignore();                       // loại dấu ↵ trong lệnh cin ở trên
        for (i=0; i<n; i++)
```

```
    {
        dong[i] = new char[80];      // cấp bộ nhớ cho dòng i
        cin.getline(dong[i],80);     // nhập dòng i
    }
    for (i=0; i<n; i++) cout << dong[i] << endl;   // in kết quả
    getch();
}
```

II. HÀM

Hàm là một chương trình con trong chương trình lớn. Hàm nhận (hoặc không) các đối số và trả lại (hoặc không) một giá trị cho chương trình gọi nó. Trong trường hợp không trả lại giá trị, hàm hoạt động như một thủ tục trong các NNLT khác. Một chương trình là tập các hàm, trong đó có một hàm chính với tên gọi main(), khi chạy chương trình, hàm main() sẽ được chạy đầu tiên và gọi đến hàm khác. Kết thúc hàm main() cũng là kết thúc chương trình.

Hàm giúp cho việc phân đoạn chương trình thành những môđun riêng rẽ, hoạt động độc lập với ngữ nghĩa của chương trình lớn, có nghĩa một hàm có thể được sử dụng trong chương trình này mà cũng có thể được sử dụng trong chương trình khác, dễ cho việc kiểm tra và bảo trì chương trình. Hàm có một số đặc trưng:

- Nằm trong hoặc ngoài văn bản có chương trình gọi đến hàm. Trong một văn bản có thể chứa nhiều hàm,
- Được gọi từ chương trình chính (main), từ hàm khác hoặc từ chính nó (đệ quy),
- Không lồng nhau.
- Có 3 cách truyền giá trị: Truyền theo tham trị, tham biến và tham trỏ.

1. Khai báo và định nghĩa hàm

a. Khai báo

Một hàm thường làm chức năng: tính toán trên các tham đối và cho lại giá trị kết quả, hoặc chỉ đơn thuần thực hiện một chức năng nào đó, không trả lại kết quả tính toán. Thông thường kiểu của giá trị trả lại được gọi là kiểu của hàm. Các hàm thường được khai báo ở đầu chương trình. Các hàm viết sẵn được khai báo trong các file nguyên mẫu *.h. Do đó, để sử dụng được các hàm này, cần có chỉ thị #include <*.h> ở ngay đầu chương trình, trong đó *.h là tên file cụ thể có chứa khai báo của các hàm

được sử dụng (ví dụ để sử dụng các hàm toán học ta cần khai báo file nguyên mẫu math.h). Đối với các hàm do NSD tự viết, cũng cần phải khai báo. Khai báo một hàm như sau:

<kiểu giá trị trả lại> <tên hàm>(d/s kiểu đối) ;

trong đó, kiểu giá trị trả lại còn gọi là kiểu hàm và có thể nhận kiểu bất kỳ chuẩn của C++ và cả kiểu của NSD tự tạo. Đặc biệt nếu hàm không trả lại giá trị thì kiểu của giá trị trả lại được khai báo là **void**. Nếu kiểu giá trị trả lại được bỏ qua thì chương trình ngầm định hàm có kiểu là **int** (phân biệt với void !).

Ví dụ 1 :

```
int bp(int);              // Khai báo hàm bp, có đối kiểu int và kiểu hàm là int
int rand100();            // Không đối, kiểu hàm (giá trị trả lại) là int
void alltrim(char[]) ;    // đối là xâu kí tự, hàm không trả lại giá trị (không kiểu).
cong(int, int);           // Hai đối kiểu int, kiểu hàm là int (ngầm định).
```

Thông thường để chương trình được rõ ràng chúng ta nên tránh lạm dụng các ngầm định. Ví dụ trong khai báo cong(int, int); nên khai báo rõ cả kiểu hàm (trong trường hợp này kiểu hàm ngầm định là int) như sau : int cong(int, int);

b. Định nghĩa hàm

Cấu trúc một hàm bất kỳ được bố trí cũng giống như hàm main() trong các phần trước. Cụ thể:

- Hàm có trả về giá trị

```
<kiểu hàm> <tên hàm>(danh sách tham đối hình thức)
{
    khai báo cục bộ của hàm ;     // chỉ dùng riêng cho hàm này
    dãy lệnh của hàm ;
    return (biểu thức trả về);    // có thể nằm đâu đó trong dãy lệnh.
}
```

– Danh sách tham đối hình thức còn được gọi ngắn gọn là danh sách đối gồm dãy các đối cách nhau bởi dấu phẩy, đối có thể là một biến thường, biến tham chiếu hoặc biến con trỏ, hai loại biến sau ta sẽ trình bày trong các phần tới. Mỗi đối được khai báo giống như khai báo biến, tức là cặp gồm <kiểu đối> <tên đối>.

– Với hàm có trả lại giá trị cần có câu lệnh **return** kèm theo sau là một biểu thức. Kiểu của giá trị biểu thức này chính là kiểu của hàm đã được khai báo ở

phần tên hàm. Câu lênh return có thể nằm ở vị trí bất kỳ trong phần câu lệnh, tuỳ thuộc mục đích của hàm. Khi gặp câu lệnh return chương trình tức khắc thoát khỏi hàm và trả lại giá trị của biểu thức sau return như giá trị của hàm.

Ví dụ 2 : Ví dụ sau định nghĩa hàm tính luỹ thừa n (với n nguyên) của một số thực bất kỳ. Hàm này có hai đầu vào (đối thực x và số mũ nguyên n) và đầu ra (giá trị trả lại) kiểu thực với độ chính xác gấp đôi là x^n.

```
double luythua(float x, int n)
{
    int i ;                     // biến chỉ số
    double kq = 1 ;             // để lưu kết quả
    for (i=1; i<=n; i++) kết quả *= x ;
    return kq;
}
```

- Hàm không trả về giá trị

Nếu hàm không trả lại giá trị (tức kiểu hàm là void), khi đó có thể có hoặc không có câu lệnh return, nếu có thì đằng sau return sẽ không có biểu thức giá trị trả lại.

Ví dụ 3 : Hàm xoá màn hình 100 lần, hàm chỉ làm công việc cẩn thận xoá màn hình nhiều lần để màn hình thật sạch, nên không có giá trị gì để trả lại.

```
void xmh()
{
    int i;
    for (i=1; i<=100; i++) clrscr();
    return ;
}
```

Hàm main() thông thường có hoặc không có giá trị trả về cho hệ điều hành khi chương trình chạy xong, vì vậy ta thường khai báo kiểu hàm là int main() hoặc void main() và câu lệnh cuối cùng trong hàm thường là return 1 hoặc return. Trường hợp bỏ qua từ khoá void nhưng trong thân hàm không có câu lệnh return (giống phần lớn ví dụ trong giáo trình này) chương trình sẽ ngầm hiểu hàm main() trả lại một giá trị nguyên nhưng vì không có nên khi dịch chương trình ta sẽ gặp lời cảnh báo "Cần có giá trị trả lại cho hàm" (một lời cảnh báo không phải là lỗi, chương trình vẫn chạy bình thường). Để tránh bị quấy rầy về những lời cảnh báo "không mời" này chúng ta có thể đặt thêm câu lệnh return 0; (nếu không khai báo void main()) hoặc khai báo kiểu hàm là void main() và đặt câu lệnh return vào cuối hàm.

c. Chú ý về khai báo và định nghĩa hàm

- Danh sách đối trong khai báo hàm có thể chứa hoặc không chứa tên đối, thông thường ta chỉ khai báo kiểu đối chứ không cần khai báo tên đối, trong khi ở dòng đầu tiên của định nghĩa hàm phải có tên đối đầy đủ.
- Cuối khai báo hàm phải có dấu chấm phẩy (;), trong khi cuối dòng đầu tiên của định nghĩa hàm không có dấu chấm phẩy.
- Hàm có thể không có đối (danh sách đối rỗng), tuy nhiên cặp dấu ngoặc sau tên hàm vẫn phải được viết. Ví dụ clrscr(), lamtho(), vietgiaotrinh(), …
- Một hàm có thể không cần phải khai báo nếu nó được định nghĩa trước khi có hàm nào đó gọi đến nó. Ví dụ có thể viết hàm main() trước (trong văn bản chương trình), rồi sau đó mới viết đến các hàm "con". Do trong hàm main() chắc chắn sẽ gọi đến hàm con này nên danh sách của chúng phải được khai báo trước hàm main(). Trường hợp ngược lại nếu các hàm con được viết (định nghĩa) trước thì không cần phải khai báo chúng nữa (vì trong định nghĩa đã hàm ý khai báo). Nguyên tắc này áp dụng cho hai hàm A, B bất kỳ chứ không riêng cho hàm main(), nghĩa là nếu B gọi đến A thì trước đó A phải được định nghĩa hoặc ít nhất cũng có dòng khai báo về A.

2. Lời gọi và sử dụng hàm

Lời gọi hàm được phép xuất hiện trong bất kỳ biểu thức, câu lệnh của hàm khác … Nếu lời gọi hàm lại nằm trong chính bản thân hàm đó thì ta gọi là đệ quy. Để gọi hàm ta chỉ cần viết tên hàm và danh sách các giá trị cụ thể truyền cho các đối đặt trong cặp dấu ngoặc tròn ().

tên hàm(danh sách tham đối thực sự) ;

- Danh sách tham đối thực sự còn gọi là danh sách giá trị gồm các giá trị cụ thể để gán lần lượt cho các đối hình thức của hàm. Khi hàm được gọi thực hiện thì tất cả những vị trí xuất hiện của đối hình thức sẽ được gán cho giá trị cụ thể của đối thực sự tương ứng trong danh sách, sau đó hàm tiến hành thực hiện các câu lệnh của hàm (để tính kết quả).
- Danh sách tham đối thực sự truyền cho tham đối hình thức có số lượng bằng với số lượng đối trong hàm và được truyền cho đối theo thứ tự tương ứng. Các tham đối thực sự có thể là các hằng, các biến hoặc biểu thức. Biến trong giá trị có thể trùng với tên đối. Ví dụ ta có hàm in n lần kí tự c với tên hàm inkitu(int n, char c); và lời gọi hàm inkitu(12, 'A'); thì n và c là các đối hình thức, 12 và 'A' là các đối thực sự hoặc giá trị. Các đối hình thức n và c sẽ lần lượt được gán bằng các giá trị tương ứng là 12 và 'A' trước khi tiến hành các câu lệnh trong phần thân hàm. Giả sử hàm in kí tự được khai báo lại thành inkitu(char

c, int n); thì lời gọi hàm cũng phải được thay lại thành inkitu('A', 12).

- Các giá trị tương ứng được truyền cho đối phải có kiểu cùng với kiểu đối (hoặc C++ có thể tự động chuyển kiểu được về kiểu của đối).
- Khi một hàm được gọi, nơi gọi tạm thời chuyển điều khiển đến thực hiện dòng lệnh đầu tiên trong hàm được gọi. Sau khi kết thúc thực hiện hàm, điều khiển lại được trả về thực hiện tiếp câu lệnh sau lệnh gọi hàm của nơi gọi.

Ví dụ 4 : Giả sử ta cần tính giá trị của biểu thức $2x^3 - 5x^2 - 4x + 1$, thay cho việc tính trực tiếp x^3 và x^2, ta có thể gọi hàm luythua() trong ví dụ trên để tính các giá trị này bằng cách gọi nó trong hàm main() như sau:

```
#include <iostream.h>
#include <iomanip.h>
double luythua(float x, int n)                  // trả lại giá trị x^n
{
      int i ;                                   // biến chỉ số
      double kq = 1 ;                           // để lưu kết quả
      for (i=1; i<=n; i++) kết quả *= x ;
      return kq;
}

void xmh(int n)                                 // xoá màn hình n lần
{
      int i;
      for (i=1; i<=n; i++) clrscr();
      return ;
}

main()                                          // tính giá trị 2x^3 - 5x^2 - 4x + 1
{
      float x ;                                 // tên biến có thể trùng với đối của hàm
      double f ;                                // để lưu kết quả
      cout << "x = " ; cin >> x
      f = 2*luythua(x,3) - 5*luythua(x,2) - 4*x + 1;
      xmh(100);                                 // xoá thật sạch màn hình 100 lần
```

```
        cout << setprecision(2) << f << endl ;
}
```

Qua ví dụ này ta thấy lợi ích của lập trình cấu trúc, chương trình trở nên gọn hơn, chẳng hạn hàm luythua() chỉ được viết một lần nhưng có thể sử dụng nó nhiều lần (2 lần trong ví dụ này) chỉ bằng một câu lệnh gọi đơn giản cho mỗi lần sử dụng thay vì phải viết lại nhiều lần đoạn lệnh tính luỹ thừa.

3. Hàm với đối mặc định

Mục này và mục sau chúng ta bàn đến một vài mở rộng thiết thực của C++ đối với C có liên quan đến hàm, đó là hàm với đối mặc định và cách tạo, sử dụng các hàm có chung tên gọi. Một mở rộng quan trọng khác là cách truyền đối theo tham chiếu sẽ được bàn chung trong mục truyền tham đối thực sự cho hàm.

Trong phần trước chúng ta đã khẳng định số lượng tham đối thực sự phải bằng số lượng tham đối hình thức khi gọi hàm. Tuy nhiên, trong thực tế rất nhiều lần hàm được gọi với các giá trị của một số tham đối hình thức được lặp đi lặp lại. Trong trường hợp như vậy lúc nào cũng phải viết một danh sách dài các tham đối thực sự giống nhau cho mỗi lần gọi là một công việc không mấy thú vị. Từ thực tế đó C++ đưa ra một cú pháp mới về hàm sao cho một danh sách tham đối thực sự trong lời gọi không nhất thiết phải viết đầy đủ nếu một số trong chúng đã có sẵn những giá trị định trước. Cú pháp này được gọi là hàm với tham đối mặc định và được khai báo với cú pháp như sau:

<kiểu hàm> <tên hàm>(đ1, …, đn, đmđ1 = gt1, …, đmđm = gtm) ;

- Các đối đ1, …, đn và đối mặc định đmđ1, …, đmđm đều được khai báo như cũ nghĩa là gồm có kiểu đối và tên đối.
- Riêng các đối mặc định đmđ1, …, đmđm có gán thêm các giá trị mặc định gt1, …, gtm. Một lời gọi bất kỳ khi gọi đến hàm này đều phải có đầy đủ các tham đối thực sự ứng với các đ1, …, đm nhưng có thể có hoặc không các tham đối thực sự ứng với các đối mặc định đmđ1, …, đmđm. Nếu tham đối nào không có tham đối thực sự thì nó sẽ được tự động gán giá trị mặc định đã khai báo.

Ví dụ 5 :

- Xét hàm xmh(int n = 100), trong đó n mặc định là 100, nghĩa là nếu gọi xmh(99) thì màn hình được xoá 99 lần, còn nếu gọi xmh(100) hoặc gọn hơn xmh() thì chương trình sẽ xoá màn hình 100 lần.
- Tương tự, xét hàm int luythua(float x, int n = 2); Hàm này có một tham đối mặc định là số mũ n, nếu lời gọi hàm bỏ qua số mũ này thì chương trình hiểu là tính bình phương của x (n = 2). Ví dụ lời gọi luythua(4, 3) được hiểu là 4^3

còn luythua(4) được hiểu là 4^2.

– Hàm tính tổng 4 số nguyên: int tong(int m, int n, int i = 0; int j = 0); khi đó có thể tính tổng của 5, 2, 3, 7 bằng lời gọi hàm **tong(5,2,3,7)** hoặc có thể chỉ tính tổng 3 số 4, 2, 1 bằng lời gọi **tong(4,2,1)** hoặc cũng có thể gọi **tong(6,4)** chỉ để tính tổng của 2 số 6 và 4.

Chú ý: Các đối ngầm định phải được khai báo liên tục và xuất hiện cuối cùng trong danh sách đối. Ví dụ:

```
int tong(int x, int y=2, int z, int t=1);     // sai vì các đối mặc định không liên tục
void xoa(int x=0, int y)                      // sai vì đối mặc định không ở cuối
```

4. Khai báo hàm trùng tên

Hàm trùng tên hay còn gọi là hàm chồng (đè). Đây là một kỹ thuật cho phép sử dụng cùng một tên gọi cho các hàm "giống nhau" (cùng mục đích) nhưng xử lý trên các kiểu dữ liệu khác nhau hoặc trên số lượng dữ liệu khác nhau. Ví dụ hàm sau tìm số lớn nhất trong 2 số nguyên:

```
int max(int a, int b) { return (a > b) ? a: b ; }
```

Nếu đặt c = max(3, 5) ta sẽ có c = 5. Tuy nhiên cũng tương tự như vậy nếu đặt c = max(3.0, 5.0) chương trình sẽ bị lỗi vì các giá trị (float) không phù hợp về kiểu (int) của đối trong hàm max. Trong trường hợp như vậy chúng ta phải viết hàm mới để tính max của 2 số thực. Mục đích, cách làm việc của hàm này hoàn toàn giống hàm trước, tuy nhiên trong C và các NNLT cổ điển khác chúng ta buộc phải sử dụng một tên mới cho hàm "mới" này. Ví dụ:

```
float fmax(float a, float b) { return (a > b) ? a: b ; }
```

Tương tự để tuận tiện ta sẽ viết thêm các hàm

```
char cmax(char a, char b) { return (a > b) ? a: b ; }
long lmax(long a, long b) { return (a > b) ? a: b ; }
double dmax(double a, double b) { return (a > b) ? a: b ; }
```

Tóm lại ta sẽ có 5 hàm: max, cmax, fmax, lmax, dmax, việc sử dụng tên như vậy sẽ gây bất lợi khi cần gọi hàm. C++ cho phép ta có thể khai báo và định nghĩa cả 5 hàm trên với cùng 1 tên gọi ví dụ là max chẳng hạn. Khi đó ta có 5 hàm:

```
1:  int max(int a, int b) { return (a > b) ? a: b ; }
2:  float max(float a, float b) { return (a > b) ? a: b ; }
3:  char max(char a, char b) { return (a > b) ? a: b ; }
4:  long max(long a, long b) { return (a > b) ? a: b ; }
```

```
5:    double max(double a, double b) { return (a > b) ? a: b ; }
```

Và lời gọi hàm bất kỳ dạng nào như max(3,5), max(3.0,5), max('O', 'K') đều được đáp ứng. Chúng ta có thể đặt ra vấn đề: với cả 5 hàm cùng tên như vậy, chương trình gọi đến hàm nào. Vấn đề được giải quyết dễ dàng vì chương trình sẽ dựa vào kiểu của các đối khi gọi để quyết định chạy hàm nào. Ví dụ lời gọi max(3,5) có 2 đối đều là kiểu nguyên nên chương trình sẽ gọi hàm 1, lời gọi max(3.0,5) hướng đến hàm số 2 và tương tự chương trình sẽ chạy hàm số 3 khi gặp lời gọi max('O','K'). Như vậy một đặc điểm của các hàm trùng tên đó là trong danh sách đối của chúng phải có ít nhất một cặp đối nào đó khác kiểu nhau. Một đặc trưng khác để phân biệt thông qua các đối đó là số lượng đối trong các hàm phải khác nhau (nếu kiểu của chúng là giống nhau).

Ví dụ việc vẽ các hình: thẳng, tam giác, vuông, chữ nhật trên màn hình là giống nhau, chúng chỉ phụ thuộc vào số lượng các điểm nối và toạ độ của chúng. Do vậy ta có thể khai báo và định nghĩa 4 hàm vẽ nói trên với cùng chung tên gọi. Chẳng hạn:

```
void ve(Diem A, Diem B) ;                    // vẽ đường thẳng AB
void ve(Diem A, Diem B, Diem C) ;            // vẽ tam giác ABC
void ve(Diem A, Diem B, Diem C, Diem D) ;    // vẽ tứ giác ABCD
```

trong ví dụ trên ta giả thiết Diem là một kiểu dữ liệu lưu toạ độ của các điểm trên màn hình. Hàm ve(Diem A, Diem B, Diem C, Diem D) sẽ vẽ hình vuông, chữ nhật, thoi, bình hành hay hình thang phụ thuộc vào toạ độ của 4 điểm ABCD, nói chung nó được sử dụng để vẽ một tứ giác bất kỳ.

Tóm lại nhiều hàm có thể được định nghĩa chồng (với cùng tên gọi giống nhau) nếu chúng thoả các điều kiện sau:

- Số lượng các tham đối trong hàm là khác nhau, hoặc
- Kiểu của tham đối trong hàm là khác nhau.

Kỹ thuật chồng tên này còn áp dụng cả cho các toán tử. Trong phần lập trình hướng đối tượng, ta sẽ thấy NSD được phép định nghĩa các toán tử mới nhưng vẫn lấy tên cũ như +, -, *, / …

5. Biến, đối tham chiếu

Một biến có thể được gán cho một bí danh mới, và khi đó chỗ nào xuất hiện biến thì cũng tương đương như dùng bí danh và ngược lại. Một bí danh như vậy được gọi là một biến tham chiếu, ý nghĩa thực tế của nó là cho phép "tham chiếu" tới một biến khác cùng kiểu của nó, tức sử dụng biến khác nhưng bằng tên của biến tham chiếu.

Giống khai báo biến bình thường, tuy nhiên trước tên biến ta thêm dấu và (&). Có thể tạm phân biến thành 3 loại: biến thường với tên thường, biến con trỏ với dấu * trước tên và biến tham chiếu với dấu &.

<kiểu biến> &<tên biến tham chiếu> = <tên biến được tham chiếu>;

Cú pháp khai báo này cho phép ta tạo ra một biến tham chiếu mới và cho nó tham chiếu đến biến được tham chiếu (cùng kiểu và phải được khai báo từ trước). Khi đó biến tham chiếu còn được gọi là bí danh của biến được tham chiếu. Chú ý không có cú pháp khai báo chỉ tên biến tham chiếu mà không kèm theo khởi tạo.

Ví dụ:

```
int hung, dung ;          // khai báo các biến nguyên hung, dung
int &ti = hung;           // khai báo biến tham chiếu ti, teo tham chieu đến
int &teo = dung;          // hung dung. ti, teo là bí danh của hung, dung
```

Từ vị trí này trở đi việc sử dụng các tên hung, ti hoặc dung, teo là như nhau.

Ví dụ:

```
hung = 2 ;
ti ++;                    // tương đương hung ++;
cout << hung << ti ;      // 3 3
teo = ti + hung ;         // tương đương dung = hung + hung
dung ++ ;                 // tương đương teo ++
cout << dung << teo ;           // 7 7
```

Vậy sử dụng thêm biến tham chiếu để làm gì ?

Cách tổ chức bên trong của một biến tham chiếu khác với biến thường ở chỗ nội dung của nó là địa chỉ của biến mà nó đại diện (giống biến con trỏ), ví dụ câu lệnh

```
cout << teo ;             // 7
```

in ra giá trị 7 nhưng thực chất đây không phải là nội dung của biến teo, nội dung của teo là địa chỉ của dung, khi cần in teo, chương trình sẽ tham chiếu đến dung và in ra nội dung của dung (7). Các hoạt động khác trên teo cũng vậy (ví dụ teo++), thực chất là tăng một đơn vị nội dung của dung (chứ không phải của teo). Từ cách tổ chức của biến tham chiếu ta thấy chúng giống con trỏ nhưng thuận lợi hơn ở chỗ khi truy cập đên giá trị của biến được tham chiếu (dung) ta chỉ cần ghi tên biến tham chiếu (teo) chứ không cần thêm toán tử (*) ở trước như trường hợp dùng con trỏ. Điểm khác biệt này có ích khi được sử dụng để truyền đối cho các hàm với mục đích làm thay đổi nội dung của biến ngoài. Tư tưởng này được trình bày rõ ràng hơn trong mục 6 của chương.

Chú ý:

- Biến tham chiếu phải được khởi tạo khi khai báo.

- Tuy giống con trỏ nhưng không dùng được các phép toán con trỏ cho biến tham chiếu. Nói chung chỉ nên dùng trong truyền đối cho hàm.

6. Các cách truyền tham đối

Có 3 cách truyền tham đối thực sự cho các tham đối hình thức trong lời gọi hàm. Trong đó cách ta đã dùng cho đến thời điểm hiện nay được gọi là truyền theo tham trị, tức các đối hình thức sẽ nhận các giá trị cụ thể từ lời gọi hàm và tiến hành tính toán rồi trả lại giá trị. Để dễ hiểu các cách truyền đối chúng ta sẽ xem qua cách thức chương trình thực hiện với các đối khi thực hiện hàm.

a. Truyền theo tham trị

Ta xét lại ví dụ hàm luythua(float x, int n) tính x^n. Giả sử trong chương trình chính ta có các biến a, b, f đang chứa các giá trị a = 2, b = 3, và f chưa có giá trị. Để tính a^b và gán giá trị tính được cho f, ta có thể gọi f = luythua(a,b). Khi gặp lời gọi này, chương trình sẽ tổ chức như sau:

- Tạo 2 biến mới (tức 2 ô nhớ trong bộ nhớ) có tên x và n. Gán nội dung các ô nhớ này bằng các giá trị trong lời gọi, tức gán 2 (a) cho x và 3 (b) cho n.
- Tới phần khai báo (của hàm), chương trình tạo thêm các ô nhớ mang tên kq và i.
- Tiến hành tính toán (gán lại kết quả cho kq).
- Cuối cùng lấy kết quả trong kq gán cho ô nhớ f (là ô nhớ có sẵn đã được khai báo trước, nằm bên ngoài hàm).
- Kết thúc hàm quay về chương trình gọi. Do hàm luythua đã hoàn thành xong việc tính toán nên các ô nhớ được tạo ra trong khi thực hiện hàm (x, n, kq, i) sẽ được xoá khỏi bộ nhớ. Kết quả tính toán được lưu giữ trong ô nhớ f (không bị xoá vì không liên quan gì đến hàm).

Trên đây là truyền đối theo cách thông thường. Vấn đề đặt ra là giả sử ngoài việc tính f, ta còn muốn thay đổi các giá trị của các ô nhớ a, b (khi truyền nó cho hàm) thì có thể thực hiện được không ? Để giải quyết bài toán này ta cần theo một kỹ thuật khác, nhờ vào vai trò của biến con trỏ và tham chiếu.

b. Truyền theo dẫn trỏ

Xét ví dụ tráo đổi giá trị của 2 biến. Đây là một yêu cầu nhỏ nhưng được gặp nhiều lần trong chương trình, ví dụ để sắp xếp một danh sách. Do vậy cần viết một hàm để thực hiện yêu cầu trên. Hàm không trả kết quả. Do các biến cần trao đổi là chưa được biết trước tại thời điểm viết hàm, nên ta phải đưa chúng vào hàm như các tham đối, tức hàm có hai tham đối x, y đại diện cho các biến sẽ thay đổi giá trị sau này.

Từ một vài nhận xét trên, theo thông thường hàm tráo đổi sẽ được viết như sau:

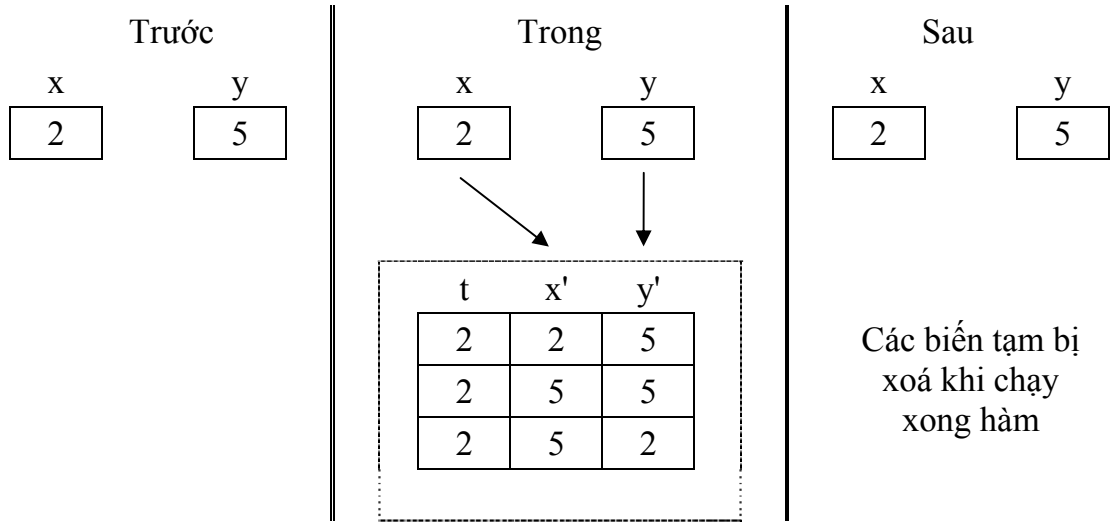
```
void swap(int x, int y)
{
    int t ;  t = x ; x = y ; y = t ;
}
```

Giả sử trong chương trình chính ta có 2 biến x, y chứa các giá trị lần lượt là 2, 5. Ta cần đổi nội dung 2 biến này sao cho x = 5 còn y = 2 bằng cách gọi đến hàm swap(x, y).

```
main()
{
    int x = 2; int y = 5;
    swap(x, y) ;
    cout << x << y ;                    // 2, 5 (x, y vẫn không đổi)
}
```

Thực sự sau khi chạy xong chương trình ta thấy giá trị của x và y vẫn không thay đổi !?.

Như đã giải thích trong mục trên (gọi hàm luythua), việc đầu tiên khi chương trình thực hiện một hàm là tạo ra các biến mới (các ô nhớ mới, độc lập với các ô nhớ x, y đã có sẵn) tương ứng với các tham đối, trong trường hợp này cũng có tên là x, y và gán nội dung của x, y (ngoài hàm) cho x, y (mới). Và việc cuối cùng của chương trình sau khi thực hiện xong hàm là xoá các biến mới này. Do vậy nội dung của các biến mới thực sự là có thay đổi, nhưng không ảnh hưởng gì đến các biến x, y cũ. Hình vẽ dưới đây minh hoạ cách làm việc của hàm swap, trước, trong và sau khi gọi hàm.



Như vậy hàm swap cần được viết lại sao cho việc thay đổi giá trị không thực hiện trên các biến tạm mà phải thực sự thực hiện trên các biến ngoài. Muốn vậy thay vì truyền giá trị của các biến ngoài cho đối, bây giờ ta sẽ truyền địa chỉ của nó cho đối, và các thay đổi sẽ phải thực hiện trên nội dung của các địa chỉ này. Đó chính là lý do ta phải sử dụng con trỏ để làm tham đối thay cho biến thường. Cụ thể hàm swap được viết lại như sau:

```
void swap(int *p, int *q)
{
    int t;              // khai báo biến tạm t
    t = *p ;            // đặt giá trị của t bằng nội dung nơi p trỏ tới
    *p = *q ;           // thay nội dung nơi p trỏ bằng nội dung nơi q trỏ
    *q = t ;            // thay nội dung nơi q trỏ tới bằng nội dung của t
}
```

Với cách tổ chức hàm như vậy rõ ràng nếu ta cho p trỏ tới biến x và q trỏ tới biến y thì hàm swap sẽ thực sự làm thay đổi nội dung của x, y chứ không phải của p, q.

Từ đó lời gọi hàm sẽ là swap(&x, &y) (tức truyền địa chỉ của x cho p, p trỏ tới x và tương tự q trỏ tới y).

Như vậy có thể tóm tắt 3 đặc trưng để viết một hàm làm thay đổi giá trị biến ngoài như sau:

- Đối của hàm phải là con trỏ (ví dụ int *p)
- Các thao tác liên quan đến đối này (trong thân hàm) phải thực hiện tại nơi nó trỏ đến (ví dụ *p = …)
- Lời gọi hàm phải chuyển địa chỉ cho p (ví dụ &x).

Ngoài hàm swap đã trình bày, ở đây ta đưa thêm ví dụ để thấy sự cần thiết phải có hàm cho phép thay đổi biến ngoài. Ví dụ hàm giải phương trình bậc 2 rất hay gặp trong các bài toán khoa học kỹ thuật. Tức cho trước 3 số a, b, c như 3 hệ số của phương trình, cần tìm 2 nghiệm x1, x2 của nó. Không thể lấy giá trị trả lại của hàm để làm nghiệm vì giá trị trả lại chỉ có 1 trong khi ta cần đến 2 nghiệm. Do vậy ta cần khai báo 2 biến "ngoài" trong chương trình để chứa các nghiệm, và hàm phải làm thay đổi 2 biến này (tức chứa giá trị nghiệm giải được). Như vậy hàm được viết cần phải có 5 đối, trong đó 3 đối a, b, c đại diện cho các hệ số, không thay đổi và 2 biến x1, x2 đại diện cho nghiệm, 2 đối này phải được khai báo dạng con trỏ. Ngoài ra, phương trình có thể vô nghiệm, 1 nghiệm hoặc 2 nghiệm do vậy hàm sẽ trả lại giá trị là số nghiệm của phương trình, trong trường hợp 1 nghiệm (nghiệm kép), giá trị nghiệm sẽ được cho vào x1.

Ví dụ 6 : Dưới đây là một dạng đơn giản của hàm giải phương trình bậc 2.

```
int gptb2(float a, float b, float c, float *p, float *q)
{
    float d ;                                   // để chứa Δ
    d = (b*b) - 4*a*c ;
    if (d < 0) return 0 ;
    else if (d == 0) { *p = -b/(2*a) ; return 1 ; }
    else {
        *p = (-b + sqrt(d))/(2*a) ;
        *q = (-b - sqrt(d))/(2*a) ;
        return 2 ;
    }
}
```

Một ví dụ của lời gọi hàm trong chương trình chính như sau:

```
main()
{
    float a, b, c ;                             // các hệ số
    float x1, x2 ;                              // các nghiệm
    cout << "Nhập hệ số: " ;
    cin >> a >> b >> c;
    switch (gptb2(a, b, c, &x1, &x2))
    {
        case 0: cout << "Phương trình vô nghiệm" ; break;
        case 1: cout << "Phương trình có nghiệm kép x = " << x1 ; break ;
        case 2: cout << "Phương trình có 2 nghiệm phân biệt:" << endl ;
        cout << "x1 = " << x1 << " và x2 = " << x2 << endl ; break;
    }
}
```

Trên đây chúng ta đã trình bày cách xây dựng các hàm cho phép thay đổi giá trị của biến ngoài. Một đặc trưng dễ nhận thấy là cách viết hàm tương đối phức tạp. Do vậy C++ đã phát triển một cách viết khác dựa trên đối tham chiếu và việc truyền đối cho hàm được gọi là truyền theo tham chiếu.

c. Truyền theo tham chiếu

Một hàm viết dưới dạng đối tham chiếu sẽ đơn giản hơn rất nhiều so với đối con trỏ và giống với cách viết bình thường (truyền theo tham trị), trong đó chỉ có một khác biệt đó là các đối khai báo dưới dạng tham chiếu.

Để so sánh 2 cách sử dụng ta nhắc lại các điểm khi viết hàm theo con trỏ phải chú ý đến, đó là:

- Đối của hàm phải là con trỏ (ví dụ int *p)
- Các thao tác liên quan đến đối này trong thân hàm phải thực hiện tại nơi nó trỏ đến (ví dụ *p = …)
- Lời gọi hàm phải chuyển địa chỉ cho p (ví dụ &x).

Hãy so sánh với đối tham chiếu, cụ thể:

- Đối của hàm phải là tham chiếu (ví dụ int &p)
- Các thao tác liên quan đến đối này phải thực hiện tại nơi nó trỏ đến, tức địa chỉ cần thao tác. Vì một thao tác trên biến tham chiếu thực chất là thao tác trên biến được nó tham chiếu nên trong hàm chỉ cần viết p trong mọi thao tác (thay vì *p như trong con trỏ)
- Lời gọi hàm phải chuyển địa chỉ cho p. Vì bản thân p khi tham chiếu đến biến nào thì sẽ chứa địa chỉ của biến đó, do đó lời gọi hàm chỉ cần ghi tên biến, ví dụ x (thay vì &x như đối với dẫn trỏ).

Tóm lại, đối với hàm viết theo tham chiếu chỉ thay đổi ở đối (là các tham chiếu) còn lại mọi nơi khác đều viết đơn giản như cách viết truyền theo tham trị.

Ví dụ 7 : Đổi giá trị 2 biến

```
void swap(int &x, int &y)
{
    int t = x; x = y; y = t;
}
```

và lời gọi hàm cũng đơn giản như trong truyền đối theo tham trị. Ví dụ:

```
int a = 5, b = 3;
swap(a, b);
cout << a << b;
```

Bảng dưới đây minh hoạ tóm tắt 3 cách viết hàm thông qua ví dụ đổi biến ở trên.

	Tham trị	Tham chiếu	Dẫn trỏ
Khai báo đối	void swap(int x, int y)	void swap(int &x, int &y)	void swap(int *x, int *y)
Câu lệnh	t = x; x = y; y = t;	t = x; x = y; y = t;	t = *x; *x = *y; *y = t;
Lời gọi	swap(a, b);	swap(a, b);	swap(&a, &b);
Tác dụng	a, b không thay đổi	a, b có thay đổi	a, b có thay đổi

7. Hàm và mảng dữ liệu

a. Truyền mảng 1 chiều cho hàm

Thông thường chúng ta hay xây dựng các hàm làm việc trên mảng như vectơ hay ma trận các phần tử. Khi đó tham đối thực sự của hàm sẽ là các mảng dữ liệu này. Trong trường hợp này ta có 2 cách khai báo đối. Cách thứ nhất đối được khai báo bình thường như khai báo biến mảng nhưng không cần có số phần tử kèm theo, ví dụ:

- int x[];
- float x[];

Cách thứ hai khai báo đối như một con trỏ kiểu phần tử mảng, ví dụ:

- int *p;
- float *p

Trong lời gọi hàm tên mảng a sẽ được viết vào danh sách tham đối thực sự, vì a là địa chỉ của phần tử đầu tiên của mảng a, nên khi hàm được gọi địa chỉ này sẽ gán cho con trỏ p. Vì vậy giá trị của phần tử thứ i của a có thể được truy cập bởi x[i] (theo khai báo 1) hoặc *(p+i) (theo khai báo 2) và nó cũng có thể được thay đổi thực sự (do đây cũng là cách truyền theo dẫn trỏ).

Sau đây là ví dụ đơn giản, nhập và in vectơ, minh hoạ cho cả 2 kiểu khai báo đối.

Ví dụ 8 : Hàm nhập và in giá trị 1 vectơ

```
void nhap(int x[], int n)                    // n: số phần tử
{
      int i;
      for (i=0; i<n; i++) cin >> x[i];       // hoặc cin >> *(x+i)
}
void in(int *p, int n)
{
```

```
    int i;
    for (i=0; i<n; i++) cout << *(p+i);
}
main()
{
    int a[10] ;                    // mảng a chứa tối đa 10 phần tử
    nhap(a,7);                     // vào 7 phần tử đầu tiên cho a
    in(a,3);                       // ra 3 phần tử đầu tiên của a
}
```

b. Truyền mảng 2 chiều cho hàm

Đối với mảng 2 chiều khai báo đối cũng như lời gọi là phức tạp hơn nhiều so với mảng 1 chiều. Ta có hai cách khai báo đối như sau:

– Khai báo theo đúng bản chất của mảng 2 chiều float x[m][n] do C++ qui định, tức x là mảng 1 chiều m phần tử, mỗi phần tử của nó có kiểu float[n]. Từ đó, đối được khai báo như một mảng hình thức 1 chiều (không cần số phần tử - ở đây là số dòng) của kiểu float[n]. Tức có thể khai báo như sau:

```
float x[][n] ;      // mảng với số phần tử không định trước, mỗi phần tử là n số
float (*x)[n] ;     // một con trỏ, có kiểu là mảng n số (float[n])
```

Để truy nhập đến đến phần tử thứ i, j ta vẫn sử dụng cú pháp x[i][j]. Tên của mảng a được viết bình thường trong lời gọi hàm. Nói chung theo cách khai báo này việc truy nhập là đơn giản nhưng phương pháp cũng có hạn chế đó là số cột của mảng truyền cho hàm phải cố định bằng n.

– Xem mảng float x[m][n] thực sự là mảng một chiều float x[m*n] và sử dụng cách khai báo như trong mảng một chiều, đó là sử dụng con trỏ float *p để truy cập được đến từng phần tử của mảng. Cách này có hạn chế trong lời gọi: địa chỉ truyền cho hàm không phải là mảng a mà cần phải ép kiểu về (float*) (để phù hợp với p). Với cách này gọi k là thứ tự của phần tử a[i][j] trong mảng một chiều (m*n), ta có quan hệ giữa k, i, j như sau:

  - k = *(p + i*n + j)
  - i = k/n
  - j = k%n

trong đó n là số cột của mảng truyền cho hàm. Điều này có nghĩa để truy cập đến a[i][j] ta có thể viết *(p+i*n+j), ngược lại biết chỉ số k có thể tính được dòng i, cột j

của phần tử này. Ưu điểm của cách khai báo này là ta có thể truyền mảng với kích thước bất kỳ (số cột không cần định trước) cho hàm.

Sau đây là các ví dụ minh hoạ cho 2 cách khai báo trên.

Ví dụ 9 : Tính tổng các số hạng trong ma trận

```
float tong(float x[][10], int m, int n)      // hoặc float tong(float (*x)[10], int m, int n)
{                                            // m: số dòng, n: số cột
    float t = 0;
    int i, j ;
    for (i=0; i<m; i++)
    for (j=0; j<n; j++) t += x[i][j] ;
    return t;
}

main()
{
    float a[8][10], b[5][7] ;
    int i, j, ma, na, mb, nb;
    cout << "nhập số dòng, số cột ma trận a: " ; cin >> ma >> na;
    for (i=0; i<ma; i++)                     // nhập ma trận a
    for (j=0; j<na; j++)
    { cout << "a[" << i << "," << j << "] = " ; cin >> a[i][j] ; }
    cout << "nhập số dòng, số cột ma trận b: " ; cin >> mb >> nb;
    for (i=0; i<mb; i++)                     // nhập ma trận b
    for (j=0; j<nb; j++)
    { cout << "b[" << i << "," << j << "] = " ; cin >> b[i][j] ; }
    cout << tong(a, ma, na);                 // in tổng các số trong ma trận
    cout << tong(b, mb, nb);                 // sai vì số cột của b khác 10
}
```

Ví dụ 10 : Tìm phần tử bé nhất của ma trận

```
void minmt(float *x, int m, int n)           // m: số dòng, n: số cột
{
```

```
        float min = *x;                         // gán phần tử đầu tiên cho min
        int k, kmin;
        for (k=1; k<m*n; k++)
        if (min > *(x+k)) { min = *(x+k) ; kmin = k; }
        cout << "Giá trị min la: " << min << " tại dòng " << k/n << " cột " << k%n;
}

main()
{
        float a[8][10], b[5][7] ;
        int i, j ;
        for (i=0; i<8; i++)                     // nhập ma trận a
        for (j=0; j<10; j++)
        { cout << "a[" << i << "," << j << "] = " ; cin >> a[i][j] ; }
        for (i=0; i<5; i++)                     // nhập ma trận b
        for (j=0; j<7; j++)
        { cout << "b[" << i << "," << j << "] = " ; cin >> b[i][j] ; }
        minmt((float*)a, 8, 10) ;       // in giá trị và vị trí số bé nhất trong a
        minmt((float*)b, 5, 7) ;        // in giá trị và vị trí số bé nhất trong b
}
```

Ví dụ 11 : Cộng 2 ma trận và in kết quả.

```
void inmt(float *x, int m, int n)
{
        int i, j;
        for (i=0; i<m; i++)
        {
                for (j=0; j<n; j++) cout << *(x+i*n+j);
                cout << endl;
        }
}

void cong(float *x, float *y, int m, int n)
```

```
{
    float *t = new float[m*n];          // t là ma trận kết quả (xem như dãy số)
    int k, i, j ;
    for (k = 0; k < m*n; k++) *(t+k) = *(x+k) + *(y+k) ;
    inmt((float*)t, m, n);
}

main()
{
    float a[8][10], b[5][7] ;
    int i, j, m, n;
    cout << "nhập số dòng, số cột ma trận: " ; cin >> m >> n;
    for (i=0; i<m; i++)                          // nhập ma trận a, b
    for (j=0; j<n; j++)
    {
        cout << "a[" << i << "," << j << "] = " ; cin >> a[i][j] ;
        cout << "b[" << i << "," << j << "] = " ; cin >> b[i][j] ;
    }
    cong((float*)a, (float*)b, m, n);            // cộng và in kết quả a+b
}
```

Xu hướng chung là chúng ta xem mảng (1 hoặc 2 chiều) như là một dãy liên tiếp các số trong bộ nhớ, tức một ma trận là một đối con trỏ trỏ đến thành phần của mảng. Đối với mảng 2 chiều m*n khi truyền đối địa chỉ của ma trận cần phải ép kiểu về kiểu con trỏ. Ngoài ra bước chạy k của con trỏ (từ 0 đến m*n-1) tương ứng với các toạ độ của phần tử a[i][j] trong mảng như sau:

- k = *(p + i*n + j)
- i = k/n
- j = k%n

từ đó, chúng ta có thể viết các hàm mà không cần phải băn khoăn gì về kích thước của ma trận sẽ truyền cho hàm.

c. Giá trị trả lại của hàm là một mảng

Không có cách nào để giá trị trả lại của một hàm là mảng. Tuy nhiên thực sự mỗi

mảng cũng chính là một con trỏ, vì vậy việc hàm trả lại một con trỏ trỏ đến dãy dữ liệu kết quả là tương đương với việc trả lại mảng. Ngoài ra còn một cách dễ dùng hơn đối với mảng 2 chiều là mảng kết quả được trả lại vào trong tham đối của hàm (giống như nghiệm của phương trình bậc 2 được trả lại vào trong các tham đối). Ở đây chúng ta sẽ lần lượt xét 2 cách làm việc này.

1. Giá trị trả lại là con trỏ trỏ đến mảng kết quả. Trước hết chúng ta xét ví dụ nhỏ sau đây:

```
int* tragiatri1()                    // giá trị trả lại là con trỏ trỏ đến dãy số nguyên
{
      int kq[3] = { 1, 2, 3 };       // tạo mảng kết quả với 3 giá trị 1, 2, 3
      return kq ;                    // trả lại địa chỉ cho con trỏ kết quả hàm
}

int* tragiatri2()                    // giá trị trả lại là con trỏ trỏ đến dãy số nguyên
{
      int *kq = new int[4];          // cấp phát 3 ô nhớ nguyên
      *kq = *(kq+1) = *(kq+2) = 0 ;  // tạo mảng kết quả với 3 giá trị 1, 2, 3
      return kq ;                    // trả lại địa chỉ cho con trỏ kết quả hàm
}

main()
{
      int *a, i;
      a = tragiatri1();
      for (i=0; i<3; i++) cout *(a+i);     // không phải là 1, 2, 3
      a = tragiatri2();
      for (i=0; i<3; i++) cout *(a+i);     // 1, 2, 3
}
```

Qua ví dụ trên ta thấy hai hàm trả giá trị đều tạo bên trong nó một mảng 3 số nguyên và trả lại địa chỉ mảng này cho con trỏ kết quả hàm. Tuy nhiên, chỉ có tragiatri2() là cho lại kết quả đúng. Tại sao ? Xét mảng kq được khai báo và khởi tạo trong tragiatri1(), đây là một mảng cục bộ (được tạo bên trong hàm) như sau này chúng ta sẽ thấy, các loại biến "tạm thời" này (và cả các tham đối) chỉ tồn tại trong quá trình hàm hoạt động. Khi hàm kết thúc các biến này sẽ mất đi. Do vậy tuy hàm đã trả lại địa

chỉ của kq trước khi nó kết thúc, thế nhưng sau khi hàm thực hiện xong, toàn bộ kq sẽ được xoá khỏi bộ nhớ và vì vậy con trỏ kết quả hàm đã trỏ đến vùng nhớ không còn các giá trị như kq đã có. Từ điều này việc sử dụng hàm trả lại con trỏ là phải hết sức cẩn thận. Muốn trả lại con trỏ cho hàm thì con trỏ này phải trỏ đến dãy dữ liệu nào sao cho nó không mất đi sau khi hàm kết thúc, hay nói khác hơn đó phải là những dãy dữ liệu được khởi tạo bên ngoài hàm hoặc có thể sử dụng theo phương pháp trong hàm tragiatri2(). Trong tragiatri2() một mảng kết quả 3 số cũng được tạo ra nhưng bằng cách xin cấp phát vùng nhớ. Vùng nhớ được cấp phát này sẽ vẫn còn tồn tại sau khi hàm kết thúc (nó chỉ bị xoá đi khi sử dụng toán tử delete). Do vậy hoạt động của tragiatri2() là chính xác.

Tóm lại, ví dụ trên cho thấy nếu muốn trả lại giá trị con trỏ thì vùng dữ liệu mà nó trỏ đến phải được cấp phát một cách tường minh (bằng toán tử new), chứ không để chương trình tự động cấp phát và tự động thu hồi. Ví dụ sau minh hoạ hàm cộng 2 vectơ và trả lại vectơ kết quả (thực chất là con trỏ trỏ đến vùng nhớ đặt kết quả)

```
int* congvt(int *x, int *y, int n)                  // n số phần tử của vectơ
{
      int* z = new int[n];                          // xin cấp phát bộ nhớ
      for (int i=0; i<n; i++) z[i] = x[i] + y[i];
      return c;
}

main()
{
      int i, n, a[10], b[10], c[10] ;
      cout << "n = " ; cin >> n;                    // nhập số phần tử
      for (i=0; i<n; i++) cin >> a[i] ;             // nhập vectơ a
      for (i=0; i<n; i++) cin >> b[i] ;             // nhập vectơ b
      c = congvt(a, b, n);
      for (i=0; i<n; i++) cout << c[i] ;            // in kết quả
}
```

Chú ý: a[i], b[i], c[i] còn được viết dưới dạng tương đương *(a+i), *(b+i), *(c+i).

2. Trong cách này, mảng cần trả lại được khai báo như một tham đối trong danh sách đối của hàm. Tham đối này là một con trỏ nên hiển nhiên khi truyền mảng đã khai báo sẵn (để chứa kết quả) từ ngoài vào cho hàm thì mảng sẽ thực sự nhận được nội dung kết quả (tức có thay đổi trước và sau khi gọi hàm

- xem mục truyền tham đối thực sự theo dẫn trỏ). Ở đây ta xét 2 ví dụ: bài toán cộng 2 vectơ trong ví dụ trước và nhân 2 ma trận.

Ví dụ 12 : Cộng 2 vectơ, vectơ kết quả trả lại trong tham đối của hàm. So với ví dụ trước giá trị trả lại là void (không trả lại giá trị) còn danh sách đối có thêm con trỏ z để chứa kết quả.

```
void congvt(int *x, int *y, int *z, int n)          // z lưu kết quả
{
    for (int i=0; i<n; i++) z[i] = x[i] + y[i];
}

main()
{
    int i, n, a[10], b[10], c[10] ;
    cout << "n = " ; cin >> n;                      // nhập số phần tử
    for (i=0; i<n; i++) cin >> a[i] ;              // nhập vectơ a
    for (i=0; i<n; i++) cin >> b[i] ;              // nhập vectơ b
    congvt(a, b, c, n);
    for (i=0; i<n; i++) cout << c[i] ;             // in kết quả
}
```

Ví dụ 13 : Nhân 2 ma trận kích thước m*n và n*p. Hai ma trận đầu vào và ma trận kết quả (kích thước m*p) đều được khai báo dưới dạng con trỏ và là đối của hàm nhanmt(). Nhắc lại, trong lời gọi hàm địa chỉ của 3 mảng cần được ép kiểu về (int*) để phù hợp với các con trỏ tham đối.

```
void nhanmt(int *x, int *y, int *z, int m, int n, int p)    // z lưu kết quả
{
    int i, j, k ;
    for (i=0; i<m; i++)
    for (j=0; j<p; j++)
    {
        *(z+i*p+j) = 0;                            // tức z[i][j] = 0
        for (k=0; k<n; k++)
        *(z+i*p+j) += *(x+i*n+k)**(y+k*p+j) ;      // tức z[i][j] += x[i][k]*y[k][j]
```

```
    }
}

main()
{
    int a[10][10], b[10][10], c[10][10] ;          // khai báo 3 mảng a, b, c
    int m, n, p ;                                  // kích thước các mảng
    cout << "m, n, p = " ; cin >> m >> n >> p ;    // nhập số phần tử
    for (i=0; i<m; i++)                            // nhập ma trận a
    for (j=0; j<n; j++)
        cout << "a[" << i << "," << j << "] = " ; cin >> a[i][j] ;
    for (i=0; i<n; i++)                            // nhập ma trận b
    for (j=0; j<p; j++)
        cout << "b[" << i << "," << j << "] = " ; cin >> b[i][j] ;
    nhanmt((int*)a, (int*)b, (int*)c, m, n, p);    // gọi hàm
    for (i=0; i<m; i++)                            // in kết quả
    {
        for (j=0; j<p; j++) cout << c[i][j] ;
        cout << endl;
    }
}
```

d. Đối và giá trị trả lại là xâu kí tự

Giống các trường hợp đã xét với mảng 1 chiều, đối của các hàm xâu kí tự có thể khai báo dưới 2 dạng: mảng kí tự hoặc con trỏ kí tự. Giá trị trả lại luôn luôn là con trỏ kí tự. Ngoài ra hàm cũng có thể trả lại giá trị vào trong các đối con trỏ trong danh sách đối.

Ví dụ sau đây dùng để tách họ, tên của một xâu họ và tên. Ví dụ gồm 3 hàm. Hàm họ trả lại xâu họ (con trỏ kí tự) với đối là xâu họ và tên được khai báo dạng mảng. Hàm tên trả lại xâu tên (con trỏ kí tự) với đối là xâu họ và tên được khai báo dạng con trỏ kí tự. Thực chất đối họ và tên trong hai hàm họ, tên có thể được khai báo theo cùng cách thức, ở đây chương trình muốn minh hoạ các cách khai báo đối khác nhau (đã đề cập đến trong phần đối mảng 1 chiều). Hàm thứ ba cũng trả lại họ, tên nhưng cho vào trong danh sách tham đối, do vậy hàm không trả lại giá trị (void). Để đơn giản ta qui ước xâu

họ và tên không chứa các dấu cách đầu và cuối xâu, trong đó họ là dãy kí tự từ đầu cho đến khi gặp dấu cách đầu tiên và tên là dãy kí tự từ sau dấu cách cuối cùng đến kí tự cuối xâu.

```
char* ho(char hoten[])                          // hàm trả lại họ
{
    char* kq = new char[10];                    // cấp bộ nhớ để chứa họ
    int i=0;
    while (hoten[i] != '\40') i++;              // i dừng tại dấu cách đầu tiên
    strncpy(kq, hoten, i) ;                     // copy i kí tự của hoten vào kq
    return kq;
}

char* ten(char* hoten)                          // hàm trả lại tên
{
    char* kq = new char[10];                    // cấp bộ nhớ để chứa tên
    int i=strlen(hoten);
    while (hoten[i] != '\40') i--;              // i dừng tại dấu cách cuối cùng
    strncpy(kq, hoten+i+1, strlen(hoten)-i-1) ; // copy tên vào kq
    return kq;
}

void tachht(char* hoten, char* ho, char* ten)
{
    int i=0;
    while (hoten[i] != '\40') i++;              // i dừng tại dấu cách đầu tiên
    strncpy(ho, hoten, i) ;                     // copy i kí tự của hoten vào ho
    i=strlen(hoten);
    while (hoten[i] != '\40') i--;              // i dừng tại dấu cách cuối cùng
    strncpy(ten, hoten+i+1, strlen(hoten)-i-1) ; // copy tên vào ten
}

main()
{
```

```
    char ht[30], *h, *t ;                          // các biến họ tên, họ, tên
    cout << "Họ và tên = " ; cin.getline(ht,30) ; // nhập họ tên
    h = ho(ht); t = ten(ht);
    cout << "Họ = " << h << ", tên = " << t << endl;
    tachht(ht, h, t);
    cout << "Họ = " << h << ", tên = " << t << endl;
}
```

e. Đối là hằng con trỏ

Theo phần truyền đối cho hàm ta đã biết để thay đổi biến ngoài đối tương ứng phải được khai báo dưới dạng con trỏ. Tuy nhiên, trong nhiều trường hợp các biến ngoài không có nhu cầu thay đổi nhưng đối tương ứng với nó vẫn phải khai báo dưới dạng con trỏ (ví dụ đối là mảng hoặc xâu kí tự). Điều này có khả năng do nhầm lẫn, các biến ngoài này sẽ bị thay đổi ngoài ý muốn. Trong trường hợp như vậy để cẩn thận, các đối con trỏ nếu không muốn thay đổi (chỉ lấy giá trị) cần được khai báo như là một hằng con trỏ bằng cách thêm trước khai báo kiểu của chúng từ khoá const. Từ khoá này khẳng định biến tuy là con trỏ nhưng nó là một hằng không thay đổi được giá trị. Nếu trong thân hàm ta cố tình thay đổi chúng thì chương trình sẽ báo lỗi. Ví dụ đối hoten trong cả 3 hàm ở trên có thể được khai báo dạng const char* hoten.

Ví dụ 14 : Đối là hằng con trỏ. In hoa một xâu kí tự

```
void inhoa(const char* s)
{
    char *t;
    strcpy(t, s);
    cout << s << strupr(t);                    // không dùng được strupr(s)
}
main()
{
    char *s = "abcde" ;
    inhoa(s);                                  // abcdeABCDE
}
```

8. Con trỏ hàm

Một hàm (tập hợp các lệnh) cũng giống như dữ liệu: có tên gọi , có địa chỉ lưu

trong bộ nhớ và có thể truy nhập đến hàm thông qua tên gọi hoặc địa chỉ của nó. Để truy nhập (gọi hàm) thông qua địa chỉ chúng ta phải khai báo một con trỏ chứa địa chỉ này và sau đó gọi hàm bằng cách gọi tên con trỏ.

a. Khai báo

<kiểu giá trị> (*tên biến hàm)(d/s tham đối);

<kiểu giá trị> (*tên biến hàm)(d/s tham đối) = <tên hàm>;

Ta thấy cách khai báo con trỏ hàm cũng tương tự khai báo con trỏ biến (chỉ cần đặt dấu * trước tên), ngoài ra còn phải bao *tên hàm giữa cặp dấu ngoặc (). Ví dụ:

```
– float (*f)(int);          // khai báo con trỏ hàm có tên là f trỏ đến hàm
                            // có một tham đối kiểu int và cho giá trị kiểu float.
– void (*f)(float, int);    // con trỏ trỏ đến hàm với cặp đối (float, int).
```

hoặc phức tạp hơn:

```
– char* (*m[10])(int, char) // khai báo một mảng 10 con trỏ hàm trỏ đến
                            // các hàm có cặp tham đối (int, char), giá trị trả
                            // lại của các hàm này là xâu kí tự.
```

Chú ý: phân biệt giữa 2 khai báo: float (*f)(int) và float *f(int). Cách khai báo trước là khai báo con trỏ hàm có tên là f. Cách khai báo sau có thể viết lại thành float* f(int) là khai báo hàm f với giá trị trả lại là một con trỏ float.

b. Khởi tạo

Một con trỏ hàm cũng giống như các con trỏ, được phép khởi tạo trong khi khai báo hoặc gán với một địa chỉ hàm cụ thể sau khi khai báo. Cũng giống như kiểu dữ liệu mảng, tên hàm chính là một hằng địa chỉ trỏ đến bản thân nó. Do vậy cú pháp của khởi tạo cũng như phép gán là như sau:

biến con trỏ hàm = tên hàm;

trong đó f và tên hàm được trỏ phải giống nhau về kiểu trả lại và danh sách đối. Nói cách khác với mục đích sử dụng con trỏ f trỏ đến hàm (lớp hàm) nào đó thì f phải được khai báo với kiểu trả lại và danh sách đối giống như hàm đó. Ví dụ:

```
float luythua(float, int);  // khai báo hàm luỹ thừa
float (*f)(float, int);     // khai báo con trỏ f tương thích với hàm luythua
f = luythua;                // cho f trỏ đến hàm luỹ thừa
```

c. Sử dụng con trỏ hàm

Để sử dụng con trỏ hàm ta phải gán nó với tên hàm cụ thể và sau đó bất kỳ nơi nào được phép xuất hiện tên hàm thì ta đều có thể thay nó bằng tên con trỏ. Ví dụ như các thao tác gọi hàm, đưa hàm vào làm tham đối hình thức cho một hàm khác … Sau đây là các ví dụ minh hoạ.

Ví dụ 15 : Dùng tên con trỏ để gọi hàm

```
float bphuong(float x)                    // hàm trả lại x2
{
      return x*x;
}
void main()
{
      float (*f)(float);
      f = bphuong;
      cout << "Bình phương của 3.5 là " << f(3.5) ;
}
```

Ví dụ 16 : Dùng hàm làm tham đối. Tham đối của hàm ngoài các kiểu dữ liệu đã biết còn có thể là một hàm. Điều này có tác dụng rất lớn trong các bài toán tính toán trên những đối tượng là hàm toán học như tìm nghiệm, tính tích phân của hàm trên một đoạn ... Hàm đóng vai trò tham đối sẽ được khai báo dưới dạng con trỏ hàm. Ví dụ sau đây trình bày hàm tìm nghiệm xấp xỉ của một hàm liên tục và đổi dấu trên đoạn [a, b]. Để hàm tìm nghiệm này sử dụng được trên nhiều hàm toán học khác nhau, trong hàm sẽ chứa một biến con trỏ hàm và hai cận a, b, cụ thể bằng khai báo **float timnghiem(float (*f)(float), float a, float b).** Trong lời gọi hàm f sẽ được thay thế bằng tên hàm cụ thể cần tìm nghiệm.

```
#define EPS 1.0e-6
float timnghiem(float (*f)(float), float a, float b);
float emu(float);
float loga(float);
void main()
{
      clrscr();
      cout << "Nghiệm của e mũ x - 2 trên đoạn [0,1] = ";
      cout << timnghiem(emu,0,1));
```

```
        cout << "Nghiệm của loga(x) - 1 trên đoạn [2,3] = ";
        cout << timnghiem(loga,2,3));
        getch();
}

float timnghiem(float (*f)(float), float a, float b)
{
        float c = (a+b)/2;
        while (fabs(a-b)>EPS && f(c)!=0)
        {
                if (f(a)*f(c)>0) a = c ; else b = c;
                c = (a+b)/2;
        }
        return c;
}

float emux(float x)     { return (exp(x)-2); }
float logx(float x)     { return (log(x)-1); }
```

d. Mảng con trỏ hàm

Tương tự như biến bình thường các con trỏ hàm giống nhau có thể được gộp lại vào trong một mảng, trong khai báo ta chỉ cần thêm [n] vào sau tên mảng với n là số lượng tối đa các con trỏ. Ví dụ sau minh hoạ cách sử dụng này. Trong ví dụ chúng ta xây dựng 4 hàm cộng, trừ, nhân, chia 2 số thực. Các hàm này giống nhau về kiểu, số lượng đối, … Chúng ta có thể sử dụng 4 con trỏ hàm riêng biệt để trỏ đến các hàm này hoặc cũng có thể dùng mảng 4 con trỏ để trỏ đến các hàm này. Chương trình sẽ in ra kết quả cộng, trừ, nhân, chia của 2 số nhập vào từ bàn phím.

Ví dụ 17 :

```
void cong(int a, int b){ cout << a << " + " << b << " = " << a+b ; }
void tru(int a, int b)   { cout << a << " - " << b << " = " << a-b ; }
void nhan(int a, int b){ cout << a << " x " << b << " = " << a*b ; }
void chia(int a, int b)  { cout << a << ": " << b << " = " << a/b ; }
main()
{
```

```
    clrscr();
    void (*f[4])(int, int) = {cong, tru, nhan, chia}; // khai báo, khởi tạo 4 con trỏ
    int m, n;
    cout "Nhập m, n " ; cin >> m >> n ;
    for (int i=0; i<4; i++) f[i](m,n);
    getch();
}
```

III. ĐỆ QUI

1. Khái niệm đệ qui

Một hàm gọi đến hàm khác là bình thường, nhưng nếu hàm lại gọi đến chính nó thì ta gọi hàm là đệ qui. Khi thực hiện một hàm đệ qui, hàm sẽ phải chạy rất nhiều lần, trong mỗi lần chạy chương trình sẽ tạo nên một tập biến cục bộ mới trên ngăn xếp (các đối, các biến riêng khai báo trong hàm) độc lập với lần chạy trước đó, từ đó dễ gây tràn ngăn xếp. Vì vậy đối với những bài toán có thể giải được bằng phương pháp lặp thì không nên dùng đệ qui.

Để minh hoạ ta hãy xét hàm tính n giai thừa. Để tính n! ta có thể dùng phương pháp lặp như sau:

```
main()
{
    int n; doule kq = 1;
    cout << "n = " ; cin >> n;
    for (int i=1; i<=n; i++) kq *= i;
    cout << n << "! = " << kq;
}
```

Mặt khác, n! giai thừa cũng được tính thông qua (n-1)! bởi công thức truy hồi

```
n! = 1          nếu n = 0
n! = (n-1)!n    nếu n > 0
```

do đó ta có thể xây dựng hàm đệ qui tính n! như sau:

```
double gt(int n)
{
```

```
        if (n==0) return 1;
        else return gt(n-1)*n;
}

main()
{
        int n;
        cout << "n = " ; cin >> n;
        cout << gt(n);
}
```

Trong hàm main() giả sử ta nhập 3 cho n, khi đó để thực hiện câu lệnh cout << gt(3) để in 3! đầu tiên chương trình sẽ gọi chạy hàm gt(3). Do 3 ≠ 0 nên hàm gt(3) sẽ trả lại giá trị gt(2)*3, tức lại gọi hàm gt với tham đối thực sự ở bước này là n = 2. Tương tự gt(2) = gt(1)*2 và gt(1) = gt(0)*1. Khi thực hiện gt(0) ta có đối n = 0 nên hàm trả lại giá trị 1, từ đó gt(1) = 1*1 = 1 và suy ngược trở lại ta có gt(2) = gt(1)*2 = 1*2 = 2, gt(3) = gt(2)*3 = 2*3 = 6, chương trình in ra kết quả 6.

Từ ví dụ trên ta thấy hàm đệ qui có đặc điểm:

- Chương trình viết rất gọn,
- Việc thực hiện gọi đi gọi lại hàm rất nhiều lần phụ thuộc vào độ lớn của đầu vào. Chẳng hạn trong ví dụ trên hàm được gọi n lần, mỗi lần như vậy chương trình sẽ mất thời gian để lưu giữ các thông tin của hàm gọi trước khi chuyển điều khiển đến thực hiện hàm được gọi. Mặt khác các thông tin này được lưu trữ nhiều lần trong ngăn xếp sẽ dẫn đến tràn ngăn xếp nếu n lớn.

Tuy nhiên, đệ qui là cách viết rất gọn, dễ viết và đọc chương trình, mặt khác có nhiều bài toán hầu như tìm một thuật toán lặp cho nó là rất khó trong khi viết theo thuật toán đệ qui thì lại rất dễ dàng.

2. Lớp các bài toán giải được bằng đệ qui

Phương pháp đệ qui thường được dùng để giải các bài toán có đặc điểm:

- Giải quyết được dễ dàng trong các trường hợp riêng gọi là trường hợp *suy biến* hay *cơ sở*, trong trường hợp này hàm được tính bình thường mà không cần gọi lại chính nó,
- Đối với trường hợp *tổng quát*, bài toán có thể giải được bằng bài toán cùng dạng nhưng với tham đối khác có kích thước nhỏ hơn tham đối ban đầu. Và sau một số bước hữu hạn biến đổi cùng dạng, bài toán đưa được về trường hợp

suy biến.

Như vậy trong trường hợp tính n! nếu n = 0 hàm cho ngay giá trị 1 mà không cần phải gọi lại chính nó, đây chính là trường hợp suy biến. Trường hợp n > 0 hàm sẽ gọi lại chính nó nhưng với n giảm 1 đơn vị. Việc gọi này được lặp lại cho đến khi n = 0.

Một lớp rất rộng của bài toán dạng này là các bài toán có thể định nghĩa được dưới dạng đệ qui như các bài toán lặp với số bước hữu hạn biết trước, các bài toán UCLN, tháp Hà Nội, ...

3. Cấu trúc chung của hàm đệ qui

Dạng thức chung của một chương trình đệ qui thường như sau:

```
if (trường hợp suy biến)
{
    trình bày cách giải          // giả định đã có cách giải
}
else                             // trường hợp tổng quát
{
    gọi lại hàm với tham đối "bé" hơn
}
```

4. Các ví dụ

Ví dụ 1 : Tìm UCLN của 2 số a, b. Bài toán có thể được định nghĩa dưới dạng đệ qui như sau:

- nếu a = b thì UCLN = a
- nếu a > b thì UCLN(a, b) = UCLN(a-b, b)
- nếu a < b thì UCLN(a, b) = UCLN(a, b-a)

Từ đó ta có chương trình đệ qui để tính UCLN của a và b như sau.

```
int UCLN(int a, int b)                  // qui uoc a, b > 0
{
    if (a < b) UCLN(a, b-a);
    if (a == b) return a;
    if (a > b) UCLN(a-b, b);
}
```

Ví dụ 2 : Tính số hạng thứ n của dãy Fibonaci là dãy f(n) được định nghĩa:

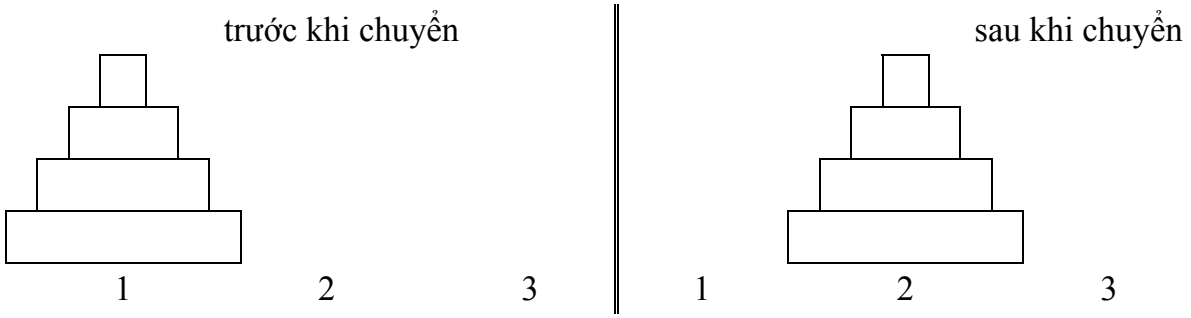
- f(0) = f(1) = 1
- f(n) = f(n-1) + f(n-2) với $\forall n \geq 2$.

```
long Fib(int n)
{
  long kq;
  if (n==0 || n==1) kq = 1; else kq = Fib(n-1) + Fib(n-2);
  return kq;
}
```

Ví dụ 3 : Chuyển tháp là bài toán cổ nổi tiếng, nội dung như sau: Cho một tháp n tầng, đang xếp tại vị trí 1. Yêu cầu bài toán là hãy chuyển toàn bộ tháp sang vị trí 2 (cho phép sử dụng vị trí trung gian 3) theo các điều kiện sau đây

- mỗi lần chỉ được chuyển một tầng trên cùng của tháp,
- tại bất kỳ thời điểm tại cả 3 vị trí các tầng tháp lớn hơn phải nằm dưới các tầng tháp nhỏ hơn.

Bài toán chuyển tháp được minh hoạ bởi hình vẽ dưới đây.



Bài toán có thể được đặt ra tổng quát hơn như sau: chuyển tháp từ vị trí di đến vị trí den, trong đó di, den là các tham số có thể lấy giá trị là 1, 2, 3 thể hiện cho 3 vị trí. Đối với 2 vị trí di và den, dễ thấy vị trí trung gian (vị trí còn lại) sẽ là vị trí 6-di-den (vì di+den+tg = 1+2+3 = 6). Từ đó để chuyển tháp từ vị trí di đến vị trí den, ta có thể xây dựng một cách chuyển đệ qui như sau:

- chuyển 1 tầng từ di sang tg,
- chuyển n-1 tầng còn lại từ di sang den,
- chuyển trả tầng tại vị trí tg về lại vị trí den

hiển nhiên nếu số tầng là 1 thì ta chỉ phải thực hiện một phép chuyển từ di sang den.

Mỗi lần chuyển 1 tầng từ vị trí i đến j ta kí hiệu i → j. Chương trình sẽ nhập vào input là số tầng và in ra các bước chuyển theo kí hiệu trên.

Từ đó ta có thể xây dựng hàm đệ qui sau đây ;

```
void chuyen(int n, int di, int den)              // n: số tầng, di, den: vị trí đi, đến
{
    if (n==1) cout << di << " → " << den << endl;
    else {
        cout << di << "→" << 6-di-den << endl; // 1 tầng từ di qua trung gian
        chuyen(n-1, di, den) ;                  // n-1 tầng từ di qua den
        cout << 6-di-den << "→" den << endl;    // 1 tầng từ tg về lại den
    }
}

main()
{
    int sotang ;
    cout << "Số tầng = " ; cin >> sotang;
    chuyen(sotang, 1, 2);
}
```

Ví dụ nếu số tầng bằng 3 thì chương trình in ra kết quả là dãy các phép chuyển sau đây:

$$1 \to 2\ ,\ 1 \to 3\ ,\ 2 \to 3\ ,\ 1 \to 2\ ,\ 3 \to 1\ ,\ 3 \to 2\ ,\ 1 \to 2.$$

có thể tính được số lần chuyển là $2^n - 1$ với n là số tầng.

IV. TỔ CHỨC CHƯƠNG TRÌNH

1. Các loại biến và phạm vi

a. Biến cục bộ

Là các biến được khai báo trong thân của hàm và chỉ có tác dụng trong hàm này, kể cả các biến khai báo trong hàm main() cũng chỉ có tác dụng riêng trong hàm main(). Từ đó, tên biến trong các hàm là được phép trùng nhau. Các biến của hàm nào sẽ chỉ

tồn tại trong thời gian hàm đó hoạt động. Khi bắt đầu hoạt động các biến này được tự động sinh ra và đến khi hàm kết thúc các biến này sẽ mất đi. Tóm lại, một hàm được xem như một đơn vị độc lập, khép kín.

Tham đối của các hàm cũng được xem như biến cục bộ.

Ví dụ 1 : Dưới đây ta nhắc lại một chương trình nhỏ gồm 3 hàm: luỹ thừa, xoá màn hình và main(). Mục đích để minh hoạ biến cục bộ.

```
float luythua(float x, int n)                    // hàm trả giá trị x^n
{
      int i ;
      float kq = 1;
      for (i=1; i<=n; i++) kq *= x;
      return kq;
}

void xmh(int n)                                  // xoá màn hình n lần
{
      int i;
      for (i=1; i<=n; i++) clrscr();
}

main()
{
      float x; int n;
      cout << "Nhập x và n: "; cin >> x >> n;
      xmh(5);                                    // xoá màn hình 5 lần
      cout << luythua(x, n);                     // in x^n
}
```

Qua ví dụ trên ta thấy các biến i, đối n được khai báo trong hai hàm: luythua() và xmh(). kq được khai báo trong luythua và main(), ngoài ra các biến x và n trùng với đối của hàm luythua(). Tuy nhiên, tất cả khai báo trên đều hợp lệ và đều được xem như khác nhau. Có thể giải thích như sau:

– Tất cả các biến trên đều cục bộ trong hàm nó được khai báo.

- x và n trong main() có thời gian hoạt động dài nhất: trong suốt quá trình chạy chương trình. Chúng chỉ mất đi khi chương trình chấm dứt. Đối x và n trong luythua() chỉ tạm thời được tạo ra khi hàm luythua() được gọi đến và độc lập với x, n trong main(), nói cách khác tại thời điểm đó trong bộ nhớ có hai biến x và hai biến n. Khi hàm luythua chay xong biến x và n của nó tự động biến mất.
- Tương tự 2 đối n, 2 biến i trong luythua() và xoá màn hình cũng độc lập với nhau, chúng chỉ được tạo và tồn tại trong thời gian hàm của chúng được gọi và hoạt động.

b. Biến ngoài

Là các biến được khai báo bên ngoài của tất cả các hàm. Vị trí khai báo của chúng có thể từ đầu văn bản chương trình hoặc tại một một vị trí bất kỳ nào đó giữa văn bản chương trình. Thời gian tồn tại của chúng là từ lúc chương trình bắt đầu chạy đến khi kết thúc chương trình giống như các biến trong hàm main(). Tuy nhiên về phạm vi tác dụng của chúng là bắt đầu từ điểm khai báo chúng đến hết chương trình, tức tất cả các hàm khai báo sau này đều có thể sử dụng và thay đổi giá trị của chúng. Như vậy các biến ngoài được khai báo từ đầu chương trình sẽ có tác dụng lên toàn bộ chương trình. Tất cả các hàm đều sử dụng được các biến này nếu trong hàm đó không có biến khai báo trùng tên. Một hàm nếu có biến trùng tên với biến ngoài thì biến ngoài bị che đối với hàm này. Có nghĩa nếu i được khai báo như một biến ngoài và ngoài ra trong một hàm nào đó cũng có biến i thì như vậy có 2 biến i độc lập với nhau và khi hàm truy nhập đến i thì có nghĩa là i của hàm chứ không phải i của biến ngoài.

Dưới đây là ví dụ minh hoạ cho các giải thích trên.

Ví dụ 2 : Chúng ta xét lại các hàm luythua() và xmh(). Chú ý rằng trong cả hai hàm này đều có biến i, vì vậy chúng ta có thể khai báo i như một biến ngoài (để dùng chung cho luythua() và xmh()), ngoài ra x, n cũng có thể được khai báo như biến ngoài. Cụ thể:

```
#include <iostream.h>
#include <iomanip.h>
float x; int n; int i ;
float luythua(float x, int n)
{
      float kq = 1;
      for (i=1; i<=n; i++) kq *= x;
}
```

```
void xmh()
{
        for (i=1; i<=n; i++) clrscr();
}

main()
{
        cout << "Nhập x và n: "; cin >> x >> n;
        xmh(5);                                    // xoá màn hình 5 lần
        cout << luythua(x, n);                     // in x^n
}
```

Trong ví dụ này ta thấy các biến x, n, i đều là các biến ngoài. Khi ta muốn sử dụng biến ngoài ví dụ i, thì biến i sẽ không được khai báo trong hàm sử dụng nó. Chẳng hạn, luythua() và xmh() đều sử dụng i cho vòng lặp for của mình và nó không được khai báo lại trong 2 hàm này. Các đối x và n trong luythua() là độc lập với biến ngoài x và n. Trong luythua() khi sử dụng đến x và n (ví dụ câu lệnh kq *= x) thì đây là x của hàm chứ không phải biến ngoài, trong khi trong main() không có khai báo về x và n nên ví dụ câu lệnh cout << luythua(x, n); là sử dụng x, n của biến ngoài.

Nói chung trong 2 ví dụ trên chương trình đều chạy tốt và như nhau. Tuy nhiên, việc khai báo khác nhau như vậy có ảnh hưởng hoặc gây nhầm lẫn gì cho người lập trình ? Liệu chúng ta có nên tự đặt ra một nguyên tắc nào đó trong khai báo biến ngoài và biến cục bộ để tránh những nhầm lẫn có thể xảy ra. Chúng ta hãy xét tiếp cũng ví dụ trên nhưng thay đổi một số khai báo và tính 2^3 (có thể bỏ bớt biến n) như sau:

```
#include <iostream.h>
#include <iomanip.h>
float x; int i ;                                   // không dùng n
float luythua(float x, int n)
{
        float kq = 1;
        for (i=1; i<=n; i++) kq *= x;
}

void xmh()
{
```

```
        for (i=1; i<=n; i++) clrscr();
}

main()
{
        x = 2;
        i = 3;
        xmh(5);                                  // xoá màn hình 5 lần
        cout << luythua(x, i);                   // in x^i, kết quả x = 2^3 = 8 ?
}
```

Nhìn vào hàm main() ta thấy giá trị 2^3 được tính bằng cách đặt x = 2, i = 3 và gọi hàm luythua(x,i). Kết quả ta mong muốn sẽ là giá trị 8 hiện ra màn hình, tuy nhiên không đúng như vậy. Trước khi in kết quả này ra màn hình hàm xmh() đã được gọi đến để xoá màn hình. Hàm này sử dụng một biến ngoài i để làm biến đếm cho mình trong vòng lặp for và sau khi ra khỏi for (cũng là kết thúc xmh()) i nhận giá trị 6. Biến i ngoài này lại được sử dụng trong lời gọi luythua(x,i) của hàm main(), tức tại thời điểm này x = 2 và i = 6, kết quả in ra màn hình sẽ là $2^6 = 64$ thay vì 8 như mong muốn.

Tóm lại "điểm yếu" dẫn đến sai sót của chương trình trên là ở chỗ lập trình viên đã "tranh thủ" sử dụng biến i cho 2 hàm xmh() và main() (bằng cách khai báo nó như biến ngoài) nhưng lại với mục đích khác nhau. Do vậy sau khi chạy xong hàm xmh() i bị thay đổi khác với giá trị i được khởi tạo lúc ban đầu. Để khắc phục lỗi trong chương trình trên ta cần khai báo lại biến i: hoặc trong main() khai báo thêm i (nó sẽ che biến i ngoài), hoặc trong cả hai xmh() và main() đều có biến i (cục bộ trong từng hàm).

Từ đó, ta nên đề ra một vài nguyên tắc lập trình sao cho nó có thể tránh được những lỗi không đáng có như vậy:

- **nếu một biến chỉ sử dụng vì mục đích riêng của một hàm thì nên khai báo biến đó như biến cục bộ trong hàm**. Ví dụ các biến đếm của vòng lặp, thông thường chúng chỉ được sử dụng thậm chí chỉ riêng trong vòng lặp chứ cũng chưa phải cho toàn bộ cả hàm, vì vậy không nên khai báo chúng như biến ngoài. Những biến cục bộ này sau khi hàm kết thúc chúng cũng sẽ kết thúc, không gây ảnh hưởng đến bất kỳ hàm nào khác. Một đặc điểm có lợi nữa cho khai báo cục bộ là chúng tạo cho hàm tính cách hoàn chỉnh, độc lập với mọi hàm khác, chương trình khác. Ví dụ hàm xmh() có thể mang qua chạy ở chương trình khác mà không phải sửa chữa gì nếu i đã được khai báo bên trong hàm. Trong khi ở ví dụ này hàm xmh() vẫn hoạt động được nhưng trong chương trình khác nếu không có i như một biến ngoài (để xmh() sử dụng) thì hàm sẽ gây lỗi.

- với các biến mang tính chất **sử dụng chung** rõ nét (đặc biệt với những biến kích thước lớn) mà nhiều hàm cùng sử dụng chúng với mục đích giống nhau thì nên khai báo chúng như biến ngoài. Điều này tiết kiệm được thời gian cho người lập trình vì không phải khai báo chúng nhiều lần trong nhiều hàm, tiết kiệm bộ nhớ vì không phải tạo chúng tạm thời mỗi khi chạy các hàm, tiết kiệm được thời gian chạy chương trình vì không phải tổ chức bộ nhớ để lưu trữ và giải phóng chúng. Ví dụ trong chương trình quản lý sinh viên (chương 6), biến sinh viên được dùng chung và thống nhất trong hầu hết các hàm (xem, xoá, sửa, bổ sung, thống kê …) nên có thể khai báo chúng như biến ngoài, điều này cũng tăng tính thống nhất của chương trình (mọi biến sinh viên là như nhau cho mọi hàm con của chương trình).

Tóm lại, nguyên tắc tổng quát nhất là cố gắng tạo hàm một cách độc lập, khép kín, không chịu ảnh hưởng của các hàm khác và không gây ảnh hưởng đến hoạt động của các hàm khác đến mức có thể.

2. Biến với mục đích đặc biệt

a. Biến hằng và từ khoá const

Để sử dụng hằng có thể khai báo thêm từ khoá const trước khai báo biến. Phạm vi và miền tác dụng cũng như biến, có nghĩa biến hằng cũng có thể ở dạng cục bộ hoặc toàn thể. Biến hằng luôn luôn được khởi tạo trước.

Có thể khai báo từ khoá const trước các tham đối hình thức để không cho phép thay đổi giá trị của các biến ngoài (đặc biệt đối với với mảng và xâu kí tự, vì bản thân các biến này được xem như con trỏ do đó hàm có thể thay đổi được giá trị của các biến ngoài truyền cho hàm này).

Ví dụ sau thể hiện hằng cũng có thể được khai báo ở các phạm vi khác nhau.

```
const int MAX = 30;                          // toàn thể
void vidu(const int *p)                      // cục bộ
{
      const MAX = 10;                        // cục bộ
…
}
void main()
{
      const MAX = 5;                         // cục bộ
…
```

```
}
…
```

Trong Turbo C, BorlandC và các chương trình dịch khác có nhiều hằng số khai báo sẵn trong tệp values.h như MAXINT, M_PI hoặc các hằng đồ hoạ trong graphics.h như WHITE, RED, …

b. Biến tĩnh và từ khoá static

Được khai báo bằng từ khoá static. Là biến cục bộ nhưng vẫn giữ giá trị sau khi ra khỏi hàm. Phạm vi tác dụng như biến cục bộ, nghĩa là nó chỉ được sử dụng trong hàm khai báo nó. Tuy nhiên thời gian tác dụng được xem như biến toàn thể, tức sau khi hàm thực hiện xong biến vẫn còn tồn tại và vẫn lưu lại giá trị sau khi ra khỏi hàm. Giá trị này này được tiếp tục sử dụng khi hàm được gọi lại, tức biến static chỉ được khởi đầu một lần trong lần chạy hàm đầu tiên. Nếu không khởi tạo, C++ tự động gán giá trị 0 (ngầm định = 0). Ví dụ:

```
int i = 1;
void bp()
{
        static int lanthu = 0;
        lanthu++;
        i = 2 * i;
        cout << "Hàm chạy lần thứ " << lanthu << ", i = " << i ;
…
}

main()
{
        ham();                          // Hàm chạy lần thứ 1, i = 1
        ham();                          // Hàm chạy lần thứ 2, i = 2
        ham();                          // Hàm chạy lần thứ 3, i = 4
…
}
```

c. Biến thanh ghi và từ khoá register

Để tăng tốc độ tính toán C++ cho phép một số biến được đặt trực tiếp vào thanh ghi thay vì ở bộ nhớ. Khai báo bằng từ khoá **register** đứng trước khai báo biến. Tuy

nhiên khai báo này chỉ có tác dụng đối với các biến có kích thước nhỏ như biến char, int.

Ví dụ: register char c; register int dem;

d. Biến ngoài và từ khoá extern

Như đã biết một chương trình có thể được đặt trên nhiều file văn bản khác nhau. Một biến không thể được khai báo nhiều lần với cùng phạm vi hoạt động. Do vậy nếu một hàm sử dụng biến được khai báo trong file văn bản khác thì biến này phải được khai báo với từ khoá extern. Từ khoá này cho phép chương trình dịch tìm và liên kết biến này từ bên ngoài file đang chứa biến. Chúng ta hãy xét ví dụ gây lỗi sau đây và tìm phương án khắc phục chúng.

```
void in();
void main()
{
        int i = 1;
        in();
}

void in()
{
        cout << i ;
}
```

- Lỗi (cú pháp) vì i là biến cục bộ trong main(), trong in() không nhận biết i, nếu trong hoặc trước in() khai báo thêm i thì lỗi ngữ nghĩa (tức chương trình in giá trị i khác không theo ý muốn của lập trình viên).
- Giả thiết khai báo lại như sau:

```
void in();
void main() { ... }          // Bỏ khai báo i trong main()
int i;                       // Đưa khai báo i ra trước in() và sau main()
void in() { ... }
```

cách khai báo này cũng gây lỗi vì main() không nhận biết i. Cuối cùng để main() có thể nhận biết i thì i phải được khai báo dưới dạng biến extern. Thông thường trong trường hợp này cách khắc phục hay nhất là khai báo trước main() để bỏ các extern (không cần thiết).

Giả thiết 2 chương trình trên nằm trong 2 tệp khác nhau. Để liên kết (link) biến i giữa 2 chương trình cần định nghĩa tổng thể i trong một và khai báo extern trong chương trình kia.

```
/* program1.cpp*/
void in();
int i;
void main()
{
        i = 1;
        in();
}
/* program2.cpp */
void in()
{
        extern i;
        cout << i ;
}
```

Hàm in() nằm trong tệp văn bản program2.cpp, được dùng để in giá trị của biến i khai báo trong program1.cpp, tạm gọi là tệp gốc (hai tệp này khi dịch sẽ được liên kết với nhau). Từ đó trong tệp gốc, i phải được khai báo là biến ngoài, và bất kỳ hàm ở tệp khác muốn sử dụng biến i này đều phải có câu lệnh khai báo extern int i (nếu không có từ khoá extern thì biến i lại được xem là biến cục bộ, khác với biến i trong tệp gốc).

Để liên kết các tệp nguồn có thể tạo một dự án (project) thông qua menu PROJECT (Alt-P). Các phím nóng cho phép mở dự án, thêm bớt tệp vào danh sách tệp của dự án … được hướng dẫn ở dòng cuối của cửa sổ dự án.

3. Các chỉ thị tiền xử lý

Như đã biết trước khi chạy chương trình (bắt đầu từ văn bản chương trình tức chương trình nguồn) C++ sẽ dịch chương trình ra tệp mã máy còn gọi là chương trình đích. Thao tác dịch chương trình nói chung gồm có 2 phần: xử lý sơ bộ chương trình và dịch. Phần xử lý sơ bộ được gọi là tiền xử lý, trong đó có các công việc liên quan đến các chỉ thị được đặt ở đầu tệp chương trình nguồn như #include, #define …

a. Chỉ thị bao hàm tệp #include

Cho phép ghép nội dung các tệp đã có khác vào chương trình trước khi dịch. Các tệp cần ghép thêm vào chương trình thường là các tệp chứa khai báo nguyên mẫu của các hằng, biến, hàm … có sẵn trong C hoặc các hàm do lập trình viên tự viết. Có hai dạng viết chỉ thị này.

1: #include <tệp>

2: #include "đường dẫn\tệp"

Dạng khai báo 1 cho phép C++ ngầm định tìm tệp tại thư mục định sẵn (khai báo thông qua menu Options\Directories) thường là thư mục TC\INCLUDE và tệp là các tệp nguyên mẫu của thư viện C++.

Dạng khai báo 2 cho phép tìm tệp theo đường dẫn, nếu không có đường dẫn sẽ tìm trong thư mục hiện tại. Tệp thường là các tệp (thư viện) được tạo bởi lập trình viên và được đặt trong cùng thư mục chứa chương trình. Cú pháp này cho phép lập trình viên chia một chương trình thành nhiều môđun đặt trên một số tệp khác nhau để dễ quản lý. Nó đặc biệt hữu ích khi lập trình viên muốn tạo các thư viện riêng cho mình.

b. Chỉ thị macro #define

#define tên_macro xaukitu

Trước khi dịch bộ tiền xử lý sẽ tìm trong chương trình và thay thế bất kỳ vị trí xuất hiện nào của tên_macro bởi xâu kí tự. Ta thường sử dụng macro để định nghĩa các hằng hoặc thay cụm từ này bằng cụm từ khác dễ nhớ hơn, ví dụ:

```
#define then                    // thay then bằng dấu cách
#define begin {                 // thay begin bằng dấu {
#define end }                   // thay end bằng dấu }
#define MAX 100                 // thay MAX bằng 100
#define TRUE 1                  // thay TRUE bằng 1
```

từ đó trong chương trình ta có thể viết những đoạn lệnh như:

```
if (i < MAX) then
begin
    Ok = TRUE;
    cout << i ;
end
```

trước khi dịch bộ tiền xử lý sẽ chuyển đoạn chương trình trên thành

```
if (i < 100)
{
```

```
        Ok = 1;
        cout << i ;
    }
```

theo đúng cú pháp của C++ và rồi mới tiến hành dịch.

Ngoài việc chỉ thị #define cho phép thay tên_macro bởi một xâu kí tự bất kỳ, nó còn cũng được phép viết dưới dạng có đối. Ví dụ, để tìm số lớn nhất của 2 số, thay vì ta phải viết nhiều hàm max (mỗi hàm ứng với một kiểu số khác nhau), bây giờ ta chỉ cần thay chúng bởi một macro có đối đơn giản như sau:

#define max(A,B) ((A) > (B) ? (A): (B))

khi đó trong chương trình nếu có dòng x = max(a, b) thì nó sẽ được thay bởi: x = ((a) > (b) ? (a): (b))

Chú ý:

- Tên macro phải được viết liền với dấu ngoặc của danh sách đối. Ví dụ không viết max (A,B).
- #define bp(x) (x*x) viết sai vì bp(5) đúng nhưng bp(a+b) sẽ thành (a+b*a+b) (tức a+b+ab).
- Cũng tương tự viết #define max(A,B) (A > B ? A: B) là sai (?) vì vậy luôn luôn bao các đối bởi dấu ngoặc.
- #define bp(x) ((x)*(x)) viết đúng nhưng nếu giả sử lập trình viên muốn tính bình phương của 2 bằng đoạn lệnh sau:

```
int i = 1;
cout << bp(++i);                    // 6
```

thì kết quả in ra sẽ là 6 thay vì kết quả đúng là 4. Lí do là ở chỗ chương trình dịch sẽ thay bp(++i) bởi ((++i)*(++i)), và với i = 1 chương trình sẽ thực hiện như 2*3 = 6. Do vậy cần cẩn thận khi sử dụng các phép toán tự tăng giảm trong các macro có đối. Nói chung, nên hạn chế việc sử dụng các macro phức tạp, vì nó có thể gây nên những hiệu ứng phụ khó kiểm soát.

c. Các chỉ thị biên dịch có điều kiện #if, #ifdef, #ifndef

- Chỉ thị:

  #if dãy lệnh … #endif

  #if dãy lệnh … #else dãy lệnh … #endif,

Các chỉ thị này giống như câu lệnh if, mục đích của nó là báo cho chương trình dịch biết đoạn lệnh giữa #if (điều kiện) và #endif chỉ được dịch nếu điều kiện đúng. Ví

dụ:

```
const int M = 1;
void main() {
int i = 5;
#if M==1
        cout << i ;
#endif
}
```

hoặc:

```
const int M = 10;
void main() {
int i = 5;
#if M > 8
        cout << i+i ;
#else
        cout << i*i ;
#endif
}
```

- **Chỉ thị #ifdef và #ifndef**

Chỉ thị này báo cho chương trình dịch biết đoạn lệnh có được dịch hay không khi một tên gọi đã được định nghĩa hay chưa. #ifdef được hiểu là nếu tên đã được định nghĩa thì dịch, còn #ifndef được hiểu là nếu tên chưa được định nghĩa thì dịch. Để định nghĩa một tên gọi ta dùng chỉ thị #define tên.

Chỉ thị này đặc biệt có ích khi chèn các tệp thư viện vào để sử dụng. Một tệp thư viện có thể được chèn nhiều lần trong văn bản do vậy nó có thể sẽ được dịch nhiều lần, điều này sẽ gây ra lỗi vì các biến được khai báo nhiều lần. Để tránh việc này, ta cần sử dụng chỉ thị trên như ví dụ minh hoạ sau: Giả sử ta đã viết sẵn 2 tệp thư viện là mylib.h và mathfunc.h, trong đó mylib.h chứa hàm max(a,b) tìm số lớn nhất giữa 2 số, mathfunc.h chứa hàm max(a,b,c) tìm số lớn nhất giữa 3 số thông qua sử dụng hàm max(a,b). Do vậy mathfunc.h phải có chỉ thị #include mylib.h để sử dụng được hàm max(a,b).

– Thư viện 1. tên tệp: MYLIB.H

```
int max(int a, int b)
```

```
{
  return (a>b? a: b);
}
```

– Thư viện 2. tên tệp: MATHFUNC.H

```
#include "mylib.h"
int max(int a, int b)
{
  return (a>b? a: b);
}
```

Hàm main của chúng ta nhập 3 số, in ra max của từng cặp số và max của cả 3 số. Chương trình cần phải sử dụng cả 2 thư viện.

```
#include "mylib.h"
#include "mathfunc.h"
main()
{
      int a, b, c;
      cout << "a, b, c = " ; cin >> a >> b >> c;
      cout << max(a,b) << max(b,c) << max(a,c) << max(a,b,c) ;
}
```

Trước khi dịch chương trình, bộ tiền xử lý sẽ chèn các thư viện vào trong tệp chính (chứa main()) trong đó mylib.h được chèn vào 2 lần (một lần của tệp chính và một lần của mathfunc.h), do vậy khi dịch chương trình, C++ sẽ báo lỗi (do hàm int max(inta, int b) được khai báo hai lần). Để khắc phục tình trạng này trong mylib.h ta thêm chỉ thị mới như sau:

```
// tệp mylib.h
#ifndef _MYLIB_                    // nếu chưa định nghĩa tên gọi
_MYLIB_
#define _MYLIB_                    // thì định nghĩa nó
int max(int a, int b)              // và các hàm khác
{
      return (a>b? a: b);
}
```

```
#endif
```

Như vậy khi chương trình dịch xử lý mylib.h lần đầu do _MYLIB_ chưa định nghĩa nên máy sẽ định nghĩa từ này, và dịch đoạn chương trình tiếp theo cho đến #endif. Lần thứ hai khi gặp lại đoạn lệnh này do _MYLIB_ đã được định nghĩa nên chương trình bỏ qua đoạn lệnh này không dịch.

Để cẩn thận trong cả mathfunc.h ta cũng sử dụng cú pháp này, vì có thể trong một chương trình khác mathfunc.h lại được sử dụng nhiều lần.

BÀI TẬP

Con trỏ

1. Hãy khai báo biến kí tự ch và con trỏ kiểu kí tự pc trỏ vào biến ch. Viết ra các cách gán giá trị 'A' cho biến ch.

2. Cho mảng nguyên cost. Viết ra các cách gán giá trị 100 cho phần tử thứ 3 của mảng.

3. Cho p, q là các con trỏ cùng trỏ đến kí tự c. Đặt *p = *q + 1. Có thể khẳng định: *q = *p - 1 ?

4. Cho p, q là các con trỏ trỏ đến biến nguyên x = 5. Đặt *p = *q + 1; Hỏi *q ?

5. Cho p, q, r, s là các con trỏ trỏ đến biến nguyên x = 10. Đặt *q = *p + 1; *r = *q + 1; *s = *r + 1. Hỏi giá trị của biến x ?

6. Chọn câu đúng nhất trong các câu sau:

A: Địa chỉ của một biến là số thứ tự của byte đầu tiên máy dành cho biến đó.

B: Địa chỉ của một biến là một số nguyên.

C: Số học địa chỉ là các phép toán làm việc trên các số nguyên biểu diễn địa chỉ của biến

D: a và b đúng

7. Chọn câu sai trong các câu sau:

A: Các con trỏ có thể phân biệt nhau bởi kiểu của biến mà nó trỏ đến.

B: Hai con trỏ trỏ đến các kiểu khác nhau sẽ có kích thước khác nhau.

C: Một con trỏ kiểu void có thể được gán bởi con trỏ có kiểu bất kỳ (cần ép kiểu).

D: Hai con trỏ cùng trỏ đến kiểu cấu trúc có thể gán cho nhau.

8. Cho con trỏ p trỏ đến biến x kiểu float. Có thể khẳng định ?

A: p là một biến và *p cũng là một biến

B: p là một biến và *p là một giá trị hằng

C: Để sử dụng được p cần phải khai báo float *p; và gán *p = x;

D: Cũng có thể khai báo void *p; và gán (float)p = &x;

9. Cho khai báo float x, y, z, *px, *py; và các lệnh px = &x; py = &y; Có thể khẳng định ?

A: Nếu x = *px thì y = *py B: Nếu x = y + z thì *px = y + z

C: Nếu *px = y + z thì *px = *py + z D: a, b, c đúng

10. Cho khai báo float x, y, z, *px, *py; và các lệnh px = &x; py = &y; Có thể khẳng định ?

A: Nếu *px = x thì *py = y B: Nếu *px = *py - z thì *px = y - z

C: Nếu *px = y - z thì x = y - z D: a, b, c đúng

11. Không dùng mảng, hãy nhập một dãy số nguyên và in ngược dãy ra màn hình.

12. Không dùng mảng, hãy nhập một dãy số nguyên và chỉ ra vị trí của số bé nhất, lớn nhất.

13. Không dùng mảng, hãy nhập một dãy số nguyên và in ra dãy đã được sắp xếp.

14. Không dùng mảng, hãy nhập một dãy kí tự. Thay mỗi kí tự ‘a’ trong dãy thành kí tự ‘b’ và in kết quả ra màn hình.

Con trỏ và xâu kí tự

15. Giả sử p là một con trỏ kiểu kí tự trỏ đến xâu "Tin học". Chọn câu đúng nhất trong các câu sau:

A: cout << p sẽ in ra dòng "Tin học" B: cout << p sẽ in ra dòng "Tin học"

C: cout << p sẽ in ra chữ cái 'T' D: b và c đúng

16. Xét chương trình (không kể các khai báo file nguyên mẫu):

```
char st[] = "tin học";
main() {
    char *p; p = new char[10];
    for (int i=0; st[i] != '\0'; i++) p[i] = st[i];
}
```

Chương trình trên chưa hoàn chỉnh vì:

A: Sử dụng sai cú pháp toán tử new

B: Sử dụng sai cú pháp p[i] (đúng ra là *(p+i))

C: Xâu p chưa có kết thúc

D: Cả a, b, c, đều sai

17. Để tính độ dài xâu một sinh viên viết đoạn chương trình sau:

```
char *st;
main()
{
        int len = 0; gets(st); while (st++ != '\0') len++; printf("%d",len);
}
```

Hãy chọn câu đúng nhất:

A: Chương trình trên là hoàn chỉnh B: Cần thay len++ bởi ++len

C: Cần thay st++ bởi *st++ D: Cần thay st++ != '\0' bởi st++ == '\0'

18. Cho xâu kí tự (dạng con trỏ) s. Hãy in ngược xâu ra màn hình.

19. Cho xâu kí tự (dạng con trỏ) s. Hãy copy từ s sang xâu t một đoạn bắt đầu tại vị trí m với độ dài n.

20. Cho xâu kí tự (dạng con trỏ) s. Hãy thống kê tần xuất xuất hiện của các kí tự có trong s. In ra màn hình theo thứ tự giảm dần của các tần xuất (tần xuất là tỉ lệ % số lần xuất hiện của x trên tổng số kí tự trong s).

Hàm

21. Chọn câu sai trong các câu sau đây:

A: Hàm không trả lại giá trị thì không cần khai báo kiểu giá trị của hàm.

B: Các biến được khai báo trong hàm là cục bộ, tự xoá khi hàm thực hiện xong

C: Hàm không trả lại giá trị sẽ có kiểu giá trị ngầm định là void.

D: Hàm là đơn vị độc lập, không được khai báo hàm lồng nhau.

22. Chọn câu đúng nhất trong các câu sau đây:

A: Hàm phải được kết thúc với 1 câu lệnh return

B: Phải có ít nhất 1 câu lệnh return cho hàm

C: Các câu lệnh return được phép nằm ở vị trí bất kỳ trong thân hàm

D: Không cần khai báo kiểu giá trị trả lại của hàm nếu hàm không có lệnh return

23. Chọn câu sai trong các câu sau đây:

A: Số tham số thực sự phải bằng số tham số hình thức trong lời gọi hàm

B: Các biến cục bộ trong thân hàm được chương trình dịch cấp phát bộ nhớ

C: Các tham số hình thức sẽ được cấp phát bộ nhớ tạm thời khi hàm được gọi

D: Kiểu của tham số thực sự phải bằng kiểu của tham số hình thức tương ứng với nó trong lời gọi hàm

24. Để thay đổi giá trị của tham biến, các đối của hàm cần khai báo dưới dạng:

A: biến bình thường và tham đối được truyền theo giá trị

B: biến con trỏ và tham đối được truyền theo giá trị

C: biến bình thường và tham đối được truyền theo địa chỉ

D: biến tham chiếu và tham đối được truyền theo giá trị

25. Viết hàm tìm UCLN của 2 số. áp dụng hàm này (AD:) để tìm UCLN của 4 số nhập từ bàn phím.

26. Viết hàm kiểm tra một số nguyên n có là số nguyên tố. AD: In ra các số nguyên tố bé hơn 1000.

27. Viết hàm kiểm tra một số nguyên n có là số nguyên tố. AD: In các cặp số sinh đôi < 1000. (Các số "sinh đôi" là các số nguyên tố mà khoảng cách giữa chúng là 2).

28. Viết hàm kiểm tra một năm có là năm nhuận. AD: In ra các năm nhuận từ năm 1000 đến 2000.

29. Viết hàm xoá dấu cách đầu tiên trong một xâu. AD: Xoá tất cả dấu cách trong xâu.

30. Viết hàm thay 2 dấu cách bởi 1 dấu cách. AD: Cắt các dấu cách giữa 2 từ của một xâu về còn 1 dấu cách.

31. Viết hàm tráo đổi giá trị của 2 số. AD: sắp xếp dãy số.

32. Viết hàm giải phương trình bậc 2. Dùng chương trình con này tìm nghiệm của một phương trình chính phương bậc 4.

33. Số hoàn chỉnh là số bằng tổng mọi ước của nó (Ví dụ 6 = 1 + 2 + 3). Hãy in ra mọi số hoàn chỉnh từ 1 đến 100.

34. Tính tổng của dãy phân số. In ra màn hình kết quả dưới dạng phân số tối giản.

35. Nhập số tự nhiên chẵn n > 2. Hãy kiểm tra số này có thể biểu diễn được dưới dạng tổng của 2 số nguyên tố hay không ?.

36. In tổng của n số nguyên tố đầu tiên.

37. Tính phần diện tích giới hạn bởi hình tròn bán kính R và hình vuông ngoại tiếp của nó.

38. Chuẩn hoá một xâu (cắt kí tự trắng 2 đầu, cắt bớt các dấu trắng (chỉ để lại 1) giữa các từ, viết hoa đầu từ).

39. Viết chương trình nhập số nguyên lớn (không quá một tỷ), hãy đọc giá trị của nó bằng cách in ra xâu kí tự tương ứng với giá trị của nó. Ví dụ 1094507 là "Một triệu, (không trăm) chín tư nghìn, năm trăm linh bảy đơn vị".

40. Viết chương trình sắp xếp theo tên một mảng họ tên nào đó.

41. Tìm tất cả số tự nhiên có 4 chữ số mà trong mỗi số không có 2 chữ số nào giống nhau.

42. Nhập số tự nhiên n. Hãy biểu diễn n dưới dạng tích của các thừa số nguyên tố.

Đệ qui

43. Nhập số nguyên dương N. Viết hàm đệ qui tính:

a. $S_1 = \dfrac{1+2+3+...+N}{N}$

b. $S_2 = \sqrt{1^2+2^2+3^2+...+N^2}$

1. Nhập số nguyên dương n. Viết hàm đệ qui tính:

a. $S_1 = \sqrt{3+\sqrt{3+\sqrt{3+...+\sqrt{3}}}}$ n dấu căn

b. $S_2 = \dfrac{1}{2+\dfrac{1}{2+\dfrac{1}{2+...\dfrac{1}{2}}}}$ n dấu chia

44. Viết hàm đệ qui tính n!. áp dụng chương trình con này tính tổ hợp chập k theo công thức truy hồi: C(n, k) = n!/(k! (n-k)!)

45. Viết hàm đệ qui tính số fibonaci thứ n. Dùng chương trình con này tính f(2) + f(4) + f(6) + f(8).

46. Viết dưới dạng đệ qui các hàm

a. daoxau b. UCLN c. Fibonaci d. Tháp Hà Nội

47. Viết macro tráo đổi nội dung 2 biến. AD: Sắp xếp dãy số.

CHƯƠNG 5

DỮ LIỆU KIỂU CẤU TRÚC VÀ HỢP

Kiểu cấu trúc
Cấu trúc tự trỏ và danh sách liên kết
Kiểu hợp
Kiểu liệt kê

Để lưu trữ các giá trị gồm nhiều thành phần dữ liệu giống nhau ta có kiểu biến mảng. Thực tế rất nhiều dữ liệu là tập các kiểu dữ liệu khác nhau tập hợp lại, để quản lý dữ liệu kiểu này C++ đưa ra kiểu dữ liệu cấu trúc. Một ví dụ của dữ liệu kiểu cấu trúc là một bảng lý lịch trong đó mỗi nhân sự được lưu trong một bảng gồm nhiều kiểu dữ liệu khác nhau như họ tên, tuổi, giới tính, mức lương …

I. KIỂU CẤU TRÚC

1. Khai báo, khởi tạo

Để tạo ra một kiểu cấu trúc NSD cần phải khai báo tên của kiểu (là một tên gọi do NSD tự đặt), tên cùng với các thành phần dữ liệu có trong kiểu cấu trúc này. Một kiểu cấu trúc được khai báo theo mẫu sau:

```
struct <tên kiểu>
{
    các thành phần ;
} <danh sách biến>;
```

- Mỗi thành phần giống như một biến riêng của kiểu, nó gồm kiểu và tên thành phần. Một thành phần cũng còn được gọi là trường.
- Phần tên của kiểu cấu trúc và phần danh sách biến có thể có hoặc không. Tuy nhiên trong khai báo kí tự kết thúc cuối cùng phải là dấu chấm phẩy (;).
- Các kiểu cấu trúc được phép khai báo lồng nhau, nghĩa là một thành phần của kiểu cấu trúc có thể lại là một trường có kiểu cấu trúc.
- Một biến có kiểu cấu trúc sẽ được phân bố bộ nhớ sao cho các thực hiện của nó được sắp liên tục theo thứ tự xuất hiện trong khai báo.

– Khai báo biến kiểu cấu trúc cũng giống như khai báo các biến kiểu cơ sở dưới dạng:

struct <tên cấu trúc> <danh sách biến> ; // kiểu cũ trong C

hoặc

<tên cấu trúc> <danh sách biến> ; // trong C++

Các biến được khai báo cũng có thể đi kèm khởi tạo:

<tên cấu trúc> biến = { giá trị khởi tạo } ;

Ví dụ:

– Khai báo kiểu cấu trúc chứa phân số gồm 2 thành phần nguyên chứa tử số và mẫu số.

```
struct Phanso
{
        int tu ;
        int mau ;
} ;
```

hoặc:

```
struct Phanso { int tu, mau ; }
```

– Kiểu ngày tháng gồm 3 thành phần nguyên chứa ngày, tháng, năm.

```
struct Ngaythang {
        int ng ;
        int th ;
        int nam ;
} holiday = { 1,5,2000 } ;
```

một biến holiday cũng được khai báo kèm cùng kiểu này và được khởi tạo bởi bộ số 1. 5. 2000. Các giá trị khởi tạo này lần lượt gán cho các thành phần theo đúng thứ tự trong khai báo, tức ng = 1, th = 5 và nam = 2000.

– Kiểu Lop dùng chứa thông tin về một lớp học gồm tên lớp và sĩ số sinh viên. Các biến kiểu Lop được khai báo là daihoc và caodang, trong đó daihoc được khởi tạo bởi bộ giá trị {"K41T", 60} với ý nghĩa tên lớp đại học là K41T và sĩ số là 60 sinh viên.

```
struct Lop {
        char tenlop[10],
```

```
        int soluong;
    } ;
    struct Lop daihoc = {"K41T", 60}, caodang ;
```

hoặc:

```
    Lop daihoc = {"K41T", 60}, caodang ;
```

– Kiểu Sinhvien gồm có các trường hoten để lưu trữ họ và tên sinh viên, ns lưu trữ ngày sinh, gt lưu trữ giới tính dưới dạng số (qui ước 1: nam, 2: nữ) và cuối cùng trường diem lưu trữ điểm thi của sinh viên. Các trường trên đều có kiểu khác nhau.

```
    struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
    } x, *p, K41T[60];
    Sinhvien y = {"NVA", {1,1,1980}, 1} ;
```

Khai báo cùng với cấu trúc Sinhvien có các biến x, con trỏ p và mảng K41T với 60 phần tử kiểu Sinhvien. Một biến y được khai báo thêm và kèm theo khởi tạo giá trị {"NVA", {1,1,1980}, 1}, tức họ tên của sinh viên y là "NVA", ngày sinh là 1/1/1980, giới tính nam và điểm thi để trống. Đây là kiểu khởi tạo thiếu giá trị, giống như khởi tạo mảng, các giá trị để trống phải nằm ở cuối bộ giá trị khởi tạo (tức các thành phần bỏ khởi tạo không được nằm xen kẽ giữa những thành phần được khởi tạo).Ví dụ này còn minh hoạ cho các cấu trúc lồng nhau, cụ thể trong kiểu cấu trúc Sinhvien có một thành phần cũng kiểu cấu trúc là thành phần ns.

2. Truy nhập các thành phần kiểu cấu trúc

Để truy nhập vào các thành phần kiểu cấu trúc ta sử dụng cú pháp: tên biến.tên thành phần hoặc tên biến → tên thành phần đối với biến con trỏ cấu trúc. Cụ thể:

– Đối với biến thường: **tên biến.tên thành phần**

Ví dụ:

```
    struct Lop {
        char tenlop[10];
        int siso;
```

```
} ;
Lop daihoc = "K41T", caodang ;
caodang.tenlop = daihoc.tenlop ;      // gán tên lớp cđẳng bởi tên lớp đhọc
caodang.siso++;                       // tăng sĩ số lớp caodang lên 1
```

– Đối với biến con trỏ: **tên biến → tên thành phần**

Ví dụ:

```
struct Sinhvien {
      char hoten[25] ;
      Ngaythang ns;
      int gt;
      float diem ;
} x, *p, K41T[60];
Sinhvien y = {"NVA", {1,1,1980}, 1} ;
y.diem = 5.5 ;                        // gán điểm thi cho sinh viên y
p = new Sinhvien ;                    // cấp bộ nhớ chứa 1 sinh viên
strcpy(p→hoten, y.hoten) ;            // gán họ tên của y cho sv trỏ bởi p
cout << p→hoten << y.hoten;           // in hoten của y và con trỏ p
```

– Đối với biến mảng: truy nhập thành phần mảng rồi đến thành phần cấu trúc.

Ví dụ:

```
strcpy(K41T[1].hoten, p→hoten) ;      // gán họ tên cho sv đầu tiên của lớp
K41T[1].diem = 7.0 ;                  // gán điểm cho sv đầu tiên
```

– Đối với cấu trúc lồng nhau. Truy nhập thành phần ngoài rồi đến thành phần của cấu trúc bên trong, sử dụng các phép toán . hoặc → (các phép toán lấy thành phần) một cách thích hợp.

```
x.ngaysinh.ng = y.ngaysinh.ng ;       // gán ngày,
x.ngaysinh.th = y.ngaysinh.th ;       // tháng,
x.ngaysinh.nam = y.ngaysinh.nam ;     // năm sinh của y cho x.
```

3. Phép toán gán cấu trúc

Cũng giống các biến mảng, để làm việc với một biến cấu trúc chúng ta phải thực hiện thao tác trên từng thành phần của chúng. Ví dụ vào/ra một biến cấu trúc phải viết

câu lệnh vào/ra từng cho từng thành phần. Nhận xét này được minh họa trong ví dụ sau:

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} x, y;
cout << " Nhập dữ liệu cho sinh viên x:" << endl ;
cin.getline(x.hoten, 25);
cin >> x.ns.ng >> x.ns.th >> x.ns.nam;
cin >> x.gt;
cin >> x.diem
cout << "Thông tin về sinh viên x là:" << endl ;
cout << "Họ và tên: " << x.hoten << endl;
cout << "Sinh ngày: " << x.ns.ng << "/" << x.ns.th << "/" << x.ns.nam ;
cout << "Giới tính: " << (x.gt == 1) ? "Nam": "Nữ ;
cout << x.diem
```

Tuy nhiên, khác với biến mảng, **đối với cấu trúc chúng ta có thể gán giá trị của 2 biến cho nhau**. Phép gán này cũng tương đương với việc gán từng thành phần của cấu trúc. Ví dụ:

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} x, y, *p ;
cout << " Nhập dữ liệu cho sinh viên x:" << endl ;
cin.getline(x.hoten, 25);
cin >> x.ns.ng >> x.ns.th >> x.ns.nam;
cin >> x.gt;
```

```
cin >> x.diem

y = x ;       // Đối với biến mảng, phép gán này là không thực hiện được
p = new Sinhvien[1] ; *p = x ;

cout << "Thông tin về sinh viên y là:" << endl ;
cout << "Họ và tên: " << y.hoten << endl;
cout << "Sinh ngày: " << y.ns.ng << "/" << y.ns.th << "/" << y.ns.nam ;
cout << "Giới tính: " << (y.gt = 1) ? "Nam": "Nữ ;
cout << y.diem
```

Chú ý: không gán bộ giá trị cụ thể cho biến cấu trúc. Cách gán này chỉ thực hiện được khi khởi tạo. Ví dụ:

```
Sinhvien x = { "NVA", {1,1,1980}, 1, 7.0}, y ;          // được
y = { "NVA", {1,1,1980}, 1, 7.0};                       // không được
y = x;                                                  // được
```

4. Các ví dụ minh hoạ

Dưới đây chúng ta đưa ra một vài ví dụ minh hoạ cho việc sử dụng kiểu cấu trúc.

Ví dụ 1 : Cộng, trừ, nhân chia hai phân số được cho dưới dạng cấu trúc.

```
#include <iostream.h>
#include <conio.h>
struct Phanso {
      int tu ;
      int mau ;
} a, b, c ;

void main()
{
      clrscr();
      cout << "Nhập phân số a:" << endl ;          // nhập a
      cout << "Tử:"; cin >> a.tu;
      cout << "Mẫu:"; cin >> a.mau;
```

```
        cout << "Nhập phân số b:" << endl ;              // nhập b
        cout << "Tử:"; cin >> b.tu;
        cout << "Mẫu:"; cin >> b.mau;
        c.tu = a.tu*b.mau + a.mau*b.tu;                 // tính và in a+b
        c.mau = a.mau*b.mau;
        cout << "a + b = " << c.tu << "/" << c.mau;
        c.tu = a.tu*b.mau - a.mau*b.tu;                 // tính và in a-b
        c.mau = a.mau*b.mau;
        cout << "a - b = " << c.tu << "/" << c.mau;
        c.tu = a.tu*b.tu;                               // tính và in axb
        c.mau = a.mau*b.mau;
        cout << "a + b = " << c.tu << "/" << c.mau;
        c.tu = a.tu*b.mau;                              // tính và in a/b
        c.mau = a.mau*b.tu;
        cout << "a + b = " << c.tu << "/" << c.mau;
        getch();
}
```

Ví dụ 2 : Nhập mảng K41T. Tính tuổi trung bình của sinh viên nam, nữ. Hiện danh sách của sinh viên có điểm thi cao nhất.

```
#include <iostream.h>
#include <conio.h>
void main()
{
        struct Sinhvien {
                char hoten[25] ;
                Ngaythang ns;
                int gt;
                float diem ;
        } x, K41T[60];
        int i, n;
```

```
        // nhập dữ liệu
        cout << "Cho biết số sinh viên: "; cin >> n;
        for (i=1, i<=n, i++)
        {
                cout << "Nhap sinh vien thu " << i);
                cout << "Ho ten: " ; cin.getline(x.hoten);
                cout << "Ngày sinh: " ; cin >> x.ns.ng >> x.ns.th >>x.ns.nam ;
                cout << "Giới tính: " ; cin >> x.gt ;
                cout << "Điểm: " ; cin >> x.diem ;
                cin.ignore();
                K41T[i] = x ;
        }
}

// Tính điểm trung bình
float tbnam = 0, tbnu = 0;
int sonam = 0, sonu = 0 ;
for (i=1; i<=n; i++)
        if (K41T[i].gt == 1) { sonam++ ; tbnam += K41T[1].diem ; }
else { sonu++ ; tbnu += K41T[1].diem ; }
cout << "Điểm trung bình của sinh viên nam là " << tbnam/sonam ;
cout << "Điểm trung bình của sinh viên nữ là " << tbnu/sonu ;

// In danh sách sinh viên có điểm cao nhất
float diemmax = 0;
for (i=1; i<=n; i++)                                    // Tìm điểm cao nhất
if (diemmax < K41T[i].diem) diemmax = K41T[i].diem ;

for (i=1; i<=n; i++)                                    // In danh sách
{
        if (K41T[i].diem < diemmax) continue ;
```

```
            x = K41T[1] ;
            cout << x.hoten << '\t' ;
            cout << x.ns.ng << "/" << x.ns.th << "/" << x.ns.nam << '\t' ;
            cout << (x.gt == 1) ? "Nam": "Nữ" << '\t' ;
            cout << x.diem << endl;
        }
    }
```

5. Hàm với cấu trúc

a. Con trỏ và địa chỉ cấu trúc

Một con trỏ cấu trúc cũng giống như con trỏ trỏ đến các kiểu dữ liệu khác, có nghĩa nó chứa địa chỉ của một biến cấu trúc hoặc một vùng nhớ có kiểu cấu trúc nào đó. Một con trỏ cấu trúc được khởi tạo bởi:

- Gán địa chỉ của một biến cấu trúc, một thành phần của mảng, tương tự nếu địa chỉ của mảng (cũng là địa chỉ của phần tử đầu tiên của mảng) gán cho con trỏ thì ta cũng gọi là con trỏ mảng cấu trúc. Ví dụ:

```
struct Sinhvien {
  char hoten[25] ;
  Ngaythang ns;
  int gt;
  float diem ;
} x, y, *p, lop[60];

p = &x ;                    // cho con trỏ p trỏ tới biến cấu trúc x
p→diem = 5.0;               // gán giá trị 5.0 cho điểm của biến x
p = &lop[10] ;              // cho p trỏ tới sinh viên thứ 10 của lớp
cout << p→hoten;            // hiện họ tên của sinh viên này
*p = y ;                    // gán lại sinh viên thứ 10 là y
(*p).gt = 2;                // sửa lại giới tính của sinh viên thứ 10 là nữ
```

Chú ý: thông qua ví dụ này ta còn thấy một cách khác nữa để truy nhập các thành phần của x được trỏ bởi con trỏ p. Khi đó *p là tương đương với x, do vậy ta dùng lại cú

pháp sử dụng toán tử . sau *p để lấy thành phần như (*p).hoten, (*p).diem, …

- Con trỏ được khởi tạo do xin cấp phát bộ nhớ. Ví dụ:

```
Sinhvien *p, *q ;
p = new Sinhvien[1];
q = new Sinhvien[60];
```

trong ví dụ này *p có thể thay cho một biến kiểu sinh viên (tương đương biến x ở trên) còn q có thể được dùng để quản lý một danh sách có tối đa là 60 sinh viên (tương đương biến lop[60], ví dụ khi đó *(p+10) là sinh viên thứ 10 trong danh sách).

- Đối với con trỏ p trỏ đến mảng a, chúng ta có thể sử dụng một số cách sau để truy nhập đến các trường của các thành phần trong mảng, ví dụ để truy cập hoten của thành phần thứ i của mảng a ta có thể viết:
  - p[i].hoten
  - (p+i)→hoten
  - *(p+i).hoten

Nói chung các cách viết trên đều dễ nhớ do suy từ kiểu mảng và con trỏ mảng. Cụ thể trong đó p[i] là thành phần thứ i của mảng a, tức a[i]. (p+i) là con trỏ trỏ đến thành phần thứ i và *(p+i) chính là a[i]. Ví dụ sau gán giá trị cho thành phần thứ 10 của mảng sinh viên lop, sau đó in ra màn hình các thông tin này. Ví dụ dùng để minh hoạ các cách truy nhập trường dữ liệu của thành phần trong mảng lop.

Ví dụ 3 :

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} lop[60] ;

strcpy(lop[10].hoten, "NVA");
lop[10].gt = 1; lop[10].diem = 9.0 ;
Sinhvien *p ;                    // khai báo thêm biến con trỏ Sinh viên
p = &lop ;                       // cho con trỏ p trỏ tới mảng lop
```

```
cout << p[10].hoten ;          // in họ tên sinh viên thứ 10
cout << (p+10) → gt ;          // in giới tính của sinh viên thứ 10
cout << (*(p+10)).diem ;       // in điểm của sinh viên thứ 10
```

Chú ý: Độ ưu tiên của toán tử lấy thành phần (dấu chấm) là cao hơn các toán tử lấy địa chỉ (&) và lấy giá trị (*) nên cần phải viết các dấu ngoặc đúng cách.

b. Địa chỉ của các thành phần của cấu trúc

Các thành phần của một cấu trúc cũng giống như các biến, do vậy cách lấy địa chỉ của các thành phần này cũng tương tự như đối với biến bình thường. Chẳng hạn địa chỉ của thành phần giới tính của biến cấu trúc x là &x.gt (lưu ý độ ưu tiên của . cao hơn &, nên &x.gt là cũng tương đương với &(x.gt)), địa chỉ của trường hoten của thành phần thứ 10 trong mảng lớp là lop[10].hoten (hoten là xâu kí tự), tương tự địa chỉ của thành phần diem của biến được trỏ bởi p là &(p→diem).

Ví dụ:

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} lop[60], *p, x = { "NVA", {1,1,1980}, 1, 9.0) };
        lop[10] = x; p = &lop[10] ;     // p trỏ đến sinh viên thứ 10 trong lop
        char *ht; int *gt; float *d;    // các con trỏ kiểu thành phần
        ht = x.ht ;                     // cho ht trỏ đến thành phần hoten của x
        gt = &(lop[10].gt) ;            // gt trỏ đến gt của sinh viên thứ 10
        d = &(p→diem) ;                 // p trỏ đến diem của sv p đang trỏ
        cout << ht ;                    // in họ tên sinh viên x
        cout << *gt ;                   // in giới tính của sinh viên thứ 10
        cout << *d ;                    // in điểm của sinh viên p đang trỏ.
```

c. Đối của hàm là cấu trúc

Một cấu trúc có thể được sử dụng để làm đối của hàm dưới các dạng sau đây:

– Là một biến cấu trúc, khi đó tham đối thực sự là một cấu trúc.

- Là một con trỏ cấu trúc, tham đối thực sự là địa chỉ của một cấu trúc.
- Là một tham chiếu cấu trúc, tham đối thực sự là một cấu trúc.
- Là một mảng cấu trúc hình thức hoặc con trỏ mảng, tham đối thực sự là tên mảng cấu trúc.

Ví dụ 4 : Ví dụ sau đây cho phép tính chính xác khoảng cách của 2 ngày tháng bất kỳ, từ đó có thể suy ra thứ của một ngày tháng bất kỳ. Đối của các hàm là một biến cấu trúc.

- Khai báo

```
struct DATE {                                   // Kiểu ngày tháng
  int ngay ;
  int thang;
  int nam ;
};
// Số ngày của mỗi tháng
int n[13] = {0,31,28,31,30,31,30,31,31,30,31,30,31};
```

- Hàm tính năm nhuận hay không nhuận, trả lại 1 nếu năm nhuận, ngược lại trả 0.

```
int Nhuan(int nm)
{
  return (nam%4==0 && nam%100!=0 || nam%400==0)? 1: 0;
}
```

- Hàm trả lại số ngày của một tháng bất kỳ. Nếu năm nhuận và là tháng hai số ngày của tháng hai (28) được cộng thêm 1.

```
int Ngayct(int thang, int nam)
{
  return n[thang] + ((thang==2) ? Nhuan(nam): 0);
}
```

- Hàm trả lại số ngày tính từ ngày 1 tháng 1 năm 1 bằng cách cộng dồn số ngày của từng năm từ năm 1 đến năm hiện tại, tiếp theo cộng dồn số ngày từng tháng của năm hiện tại cho đến tháng hiện tại và cuối cùng cộng thêm số ngày hiện tại.

```
long Tongngay(DATE d)
{
  long i, kq = 0;
  for (i=1; i<d.nam; i++) kq += 365 + Nhuan(i);
  for (i=1; i<d.thang; i++) kq += Ngayct(i,d.nam);
  kq += d.ngay;
  return kq;
}
```

- Hàm trả lại khoảng cách giữa 2 ngày bất kỳ.

```
long Khoangcach(DATE d1, DATE d2)
{
  return Tongngay(d1)-Tongngay(d2);
}
```

- Hàm trả lại thứ của một ngày bất kỳ. Qui ước 1 là chủ nhật, 2 là thứ hai, … Để tính thứ hàm dựa trên một ngày chuẩn nào đó (ở đây là ngày 1/1/2000, được biết là thứ bảy). Từ ngày chuẩn này nếu cộng hoặc trừ 7 sẽ cho ra ngày mới cũng là thứ bảy. Từ đó, tính khoảng cách giữa ngày cần tính thứ và ngày chuẩn. Tìm phần dư của phép chia khoảng cách này với 7, nếu phần dư là 0 thì thứ là bảy, phần dư là 1 thì thứ là chủ nhật …

```
int Thu(DATE d)
{
  DATE curdate = {1,1,2000};                    // ngày 1/1/2000 là thứ bảy
  long kc = Khoangcach(d, curdate);
  int du = kc % 7; if (du < 0) du += 7;
  return du;
}
```

- Hàm dịch một số dư sang thứ

```
char* Dich(int t)
{
  char* kq = new char[10];
  switch (t) {
  case 0: strcpy(kq, "thứ bảy"); break;
        case 1: strcpy(kq, "chủ nhật"); break;
  case 2: strcpy(kq, "thứ hai"); break;
        case 3: strcpy(kq, "thứ ba"); break;
  case 4: strcpy(kq, "thứ tư"); break;
  case 5: strcpy(kq, "thứ năm"); break;
        case 6: strcpy(kq, "thứ sáu"); break;
  }
  return kq;
}
```

- Hàm main()

```
void main()
{
  DATE d;
  cout << "Nhap ngay thang nam: " ;
  cin >> d.ngay >> d.thang >> d.nam ;
  cout << "Ngày " << d.ngay << "/" << d.thang << "/" << d.nam ;
  cout << " là " << Dich(Thu(d));
}
```

Ví dụ 5 : Chương trình đơn giản về quản lý sinh viên.

- Khai báo.

```
struct Sinhvien {                    // cấu trúc sinh viên
  char hoten[25] ;
  Ngaythang ns;
  int gt;
```

```
    float diem ;
} ;
Sinhvien lop[3];                    // lớp chứa tối đa 3 sinh viên
```

- Hàm in thông tin về sinh viên sử dụng biến cấu trúc làm đối. Trong lời gọi sử dụng biến cấu trúc để truyền cho hàm.

```
void in(Sinhvien x)
{
    cout << x.hoten << "\t" ;
    cout << x.ns.ng << "/" << x.ns.th << "/" << x.ns.nam << "\t" ;
    cout << x.gt << "\t";
    cout << x.diem << endl;
}
```

- Hàm nhập thông tin về sinh viên sử dụng con trỏ sinh viên làm đối. Trong lời gọi sử dụng địa chỉ của một cấu trúc để truyền cho hàm.

```
void nhap(Sinhvien *p)
{
    cin.ignore();
    cout << "Họ tên: "; cin.getline(p→hoten, 25) ;
    cout << "Ngày sinh: ";
    cin >> (p→ns).ng >> (p→ns).th >> (p→ns).nam ;
    cout << "Giới tính: "; cin >> (p→gt) ;
    cout << "Điểm: "; cin >> (p→diem) ;
}
```

- Hàm sửa thông tin về sinh viên sử dụng tham chiếu cấu trúc làm đối. Trong lời gọi sử dụng biến cấu trúc để truyền cho hàm.

```
void sua(Sinhvien &r)
{
    int chon;
    do {
        cout << "1: Sửa họ tên" << endl ;
        cout << "2: Sửa ngày sinh" << endl ;
```

```
            cout << "3: Sửa giới tính" << endl ;
            cout << "4: Sửa điểm" << endl ;
            cout << "0: Thôi" << endl ;
            cout << "Sửa (0/1/2/3/4) ? ; cin >> chon ; cin.ignore();
            switch (chon) {
                case 1: cin.getline(r.hoten, 25) ; break;
                case 2: cin >> r.ns.ng >> r.ns.th >> r.ns.nam ; break;
                case 3: cin >> r.gt ; break;
                case 4: cin >> r.diem ; break;
            }
        } while (chon) ;
    }
```

- Hàm nhapds nhập thông tin của mọi sinh viên trong mảng, sử dụng con trỏ mảng Sinhvien làm tham đối hình thức. Trong lời gọi sử dụng tên mảng để truyền cho hàm.

```
void nhapds(Sinhvien *a)
{
    int sosv = sizeof(lop) / sizeof(Sinhvien) -1;        // bỏ phần tử 0
    for (int i=1; i<=sosv; i++) nhap(&a[i]) ;
}
```

- Hàm suads cho phép sửa thông tin của sinh viên trong mảng, sử dụng con trỏ mảng Sinhvien làm tham đối hình thức. Trong lời gọi sử dụng tên mảng để truyền cho hàm.

```
void suads(Sinhvien *a)
{
    int chon;
    cout << "Chọn sinh viên cần sửa: " ; cin >> chon ; cin.ignore();
    sua(a[chon]) ;
}
```

- Hàm inds hiện thông tin của mọi sinh viên trong mảng, sử dụng hằng con trỏ mảng Sinhvien làm tham đối hình thức. Trong lời gọi sử dụng tên mảng để truyền cho hàm.

```
void hien(const Sinhvien *a)
{
        int sosv = sizeof(lop) / sizeof(Sinhvien) -1;        // bỏ phần tử 0
        for (int i=1; i<=sosv; i++) in(a[i]) ;
}
```

- Hàm main() gọi chạy các hàm trên để nhập, in, sửa danh sách sinh viên.

```
void main()
{
    nhapds(lop) ;
    inds(lop);
    suads(lop);
    inds(lop);
}
```

d. Giá trị hàm là cấu trúc

Cũng tương tự như các kiểu dữ liệu cơ bản, giá trị trả lại của một hàm cũng có thể là các cấu trúc dưới các dạng sau:

- là một biến cấu trúc.
- là một con trỏ cấu trúc.
- là một tham chiếu cấu trúc.

Sau đây là các ví dụ minh hoạ giá trị cấu trúc của hàm.

Ví dụ 6 : Đối và giá trị của hàm là cấu trúc: Cộng, trừ hai số phức.

– Khai báo kiểu số phức

```
struct Sophuc                        // Khai báo kiểu số phức dùng chung
{
        float thuc;
        float ao;
};
```

- Hàm cộng 2 số phức, trả lại một số phức

```
Sophuc Cong(Sophuc x, Sophuc y)
{
```

```
        Sophuc kq;
        kq.thuc = x.thuc + y.thuc ;
        kq.ao = x.ao + y.ao ;
        return kq;
}
```

- Hàm trừ 2 số phức, trả lại một số phức

```
Sophuc Tru(Sophuc x, Sophuc y)
{
        Sophuc kq;
        kq.thuc = x.thuc + y.thuc ;
        kq.ao = x.ao + y.ao ;
        return kq;
}
```

- Hàm in một số phức dạng (r + im)

```
void In(Sophuc x)
{
        cout << "(" << x.thuc << "," << x.ao << ")" << endl ;
}
```

- Hàm chính

```
void main()
{
        Sophuc x, y, z ;
        cout << "x = " ; cin >> x.thuc >> x.ao ;
        cout << "y = " ; cin >> y.thuc >> y.ao ;
        cout << "x + y = " ; In(Cong(x,y)) ;
        cout << "x - y = " ; In(Tru(x,y)) ;
}
```

Ví dụ 7 : Chương trình nhập và in thông tin về một lớp cùng sinh viên có điểm cao nhất lớp.

- Khai báo kiểu dữ liệu Sinh viên và biến mảng lop.

```
struct Sinhvien {
        char *hoten ;
        float diem ;
} lop[4] ;
```

- Hàm nhập sinh viên, giá trị trả lại là một con trỏ trỏ đến dữ liệu vừa nhập.

```
Sinhvien* nhap()
{
        Sinhvien* kq = new Sinhvien[1];          // nhớ cấp phát vùng nhớ
        kq->hoten = new char[15];                // cho cả con trỏ hoten
        cout << "Họ tên: "; cin.getline(kq->hoten,30);
        cout << "Điểm: "; cin >> kq->diem; cin.ignore();
        return kq;                               // trả lại con trỏ kq
}
```

- Hàm tìm sinh viên có điểm cao nhất, giá trị trả lại là một tham chiếu đến sinh viên tìm được.

```
Sinhvien& svmax()
{
        int sosv = sizeof(lop)/sizeof(Sinhvien)-1;   // bỏ thành phần thứ 0
        float maxdiem = 0;
        int kmax;                                    // chỉ số sv có điểm max
        for (int i=1; i<sosv; i++)
        if (maxdiem < lop[i].diem)
        {
                maxdiem = lop[i].diem ;
                kmax = i;
        }
        return lop[kmax];                            // trả lại sv có điểm max
```

```
}
```

- Hàm in thông tin của một sinh viên x

```
void in(Sinhvien x)
{
        cout << x.hoten << "\t";
        cout << x.diem << endl;
}
```

- Hàm chính

```
void main()
{
        clrscr();
        int i;
        int sosv = sizeof(lop)/sizeof(Sinhvien)-1;    // bỏ thành phần thứ 0
        for (i=1; i<=sosv; i++) lop[i] = *nhap();       // nhập danh sách lớp
        for (i=1; i<=sosv; i++) in(lop[i]);             // in danh sách lớp
        Sinhvien &b = svmax();          // khai báo tham chiếu b và cho
                                        // tham chiếu đến sv có điểm max
        in(b);                          // in sinh viên có điểm max
        getch();
}
```

6. Cấu trúc với thành phần kiểu bit

a. Trường bit

Thông thường các trường trong một cấu trúc thường sử dụng ít nhất là 2 byte tức 16 bit. Trong nhiều trường hợp một số trường có thể chỉ cần đến số bit ít hơn, ví dụ trường gioitinh thông thường chỉ cần đến 1 bit để lưu trữ. Những trường hợp như vậy ta có thể khai báo kiểu bit cho các trường này để tiết kiệm bộ nhớ. Tuy nhiên, cách khai báo này ít được sử dụng trừ khi cần thiết phải truy nhập đến mức bit của dữ liệu trong các chương trình liên quan đến hệ thống.

Một trường bit là một khai báo trường int và thêm dấu: cùng số bit n theo sau,

trong đó $0 \le n < 15$. Ví dụ do độ lớn của ngày không vượt quá 31, tháng không vượt quá 12 nên 2 trường này trong cấu trúc ngày tháng có thể khai báo tiết kiệm hơn bằng 5 và 4 bit như sau:

```
struct Date {
        int ng: 5;
        int th: 4;
        int nam;
} ;
```

b. Đặc điểm

Cần chú ý các đặc điểm sau của một cấu trúc có chứa trường bit:

- Các bit được bố trí liên tục trên dãy các byte.
- Kiểu trường bit phải là int (signed hoặc unsigned).
- Độ dài mỗi trường bit không quá 16 bit.
- Có thể bỏ qua một số bit nếu bỏ trống tên trường, ví dụ:

```
struct tu {
        int: 8;
        int x:8;
}
```

mỗi một biến cấu trúc theo khai báo trên gồm 2 byte, bỏ qua không sử dụng byte thấp và trường x chiếm byte (8 bit) cao.

- Không cho phép lấy địa chỉ của thành phần kiểu bit.
- Không thể xây dựng được mảng kiểu bit.
- Không được trả về từ hàm một thành phần kiểu bit. Ví dụ nếu b là một thành phần của biến cấu trúc x có kiểu bit thì câu lệnh sau là sai:

```
return x.b ;                    // sai
```

tuy nhiên có thể thông qua biến phụ như sau:

```
int tam = x.b ; return tam ;
```

- Tiết kiệm bộ nhớ
- Dùng trong kiểu union để lấy các bit của một từ (xem ví dụ trong phần kiểu hợp).

7. Câu lệnh typedef

Để thuận tiện trong sử dụng, thông thường các kiểu được NSD tạo mới sẽ được gán cho một tên kiểu bằng câu lệnh typedef như sau:

typedef <kiểu> <tên_kiểu> ;

Ví dụ: Để tạo kiểu mới có tên Bool và chỉ chứa giá trị nguyên (thực chất chỉ cần 2 giá trị 0, 1), ta có thể khai báo:

```
typedef int Bool;
```

khai báo này cho phép xem Bool như kiểu số nguyên.

hoặc có thể đặt tên cho kiểu ngày tháng là Date với khai báo sau:

```
typedef struct Date {
    int ng;
    int th;
    int nam;
};
```

khi đó ta có thể sử dụng các tên kiểu này trong các khai báo (ví dụ tên kiểu của đối, của giá trị hàm trả lại …).

8. Hàm sizeof()

Hàm trả lại kích thước của một biến hoặc kiểu. Ví dụ:

```
Bool a, b;
Date x, y, z[50];
cout << sizeof(a) << sizeof(b) << sizeof(Bool) ;                    // in 2 2 2
cout << "Số phần tử của z = " << sizeof(z) / sizeof(Date)           // in 50
```

II. CẤU TRÚC TỰ TRỎ VÀ DANH SÁCH LIÊN KẾT

Thông thường khi thiết kế chương trình chúng ta chưa biết được số lượng dữ liệu cần dùng là bao nhiêu để khai báo số biến cho phù hợp. Đặc biệt là biến dữ liệu kiểu mảng. Số lượng các thành phần của biến mảng cần phải khai báo trước và chương trình dịch sẽ bố trí vùng nhớ cố định cho các biến này. Do buộc phải khai báo trước số lượng thành phần nên kiểu mảng thường dẫn đến hoặc là lãng phí bộ nhớ (khi chương trình không dùng hết) hoặc là không đủ để chứa dữ liệu (khi chương trình cần chứa dữ liệu với số lượng thành phần lớn hơn).

Để khắc phục tình trạng này C++ cho phép cấp phát bộ nhớ động, nghĩa là số

lượng các thành phần không cần phải khai báo trước. Bằng toán tử new chúng ta có thể xin cấp phát vùng nhớ theo nhu cầu mỗi khi chạy chương trình. Tuy nhiên, cách làm này dẫn đến việc dữ liệu của một danh sách sẽ không còn nằm liên tục trong bộ nhớ như đối với biến mảng. Mỗi lần xin cấp phát bởi toán tử new, chương trình sẽ tìm một vùng nhớ đang rỗi bất kỳ để cấp phát cho biến và như vậy dữ liệu sẽ nằm rải rác trong bộ nhớ. Từ đó, để dễ dàng quản lý trật tự của một danh sách các dữ liệu, mỗi thành phần của danh sách cần phải chứa địa chỉ của thành phần tiếp theo hoặc trước nó (điều này là không cần thiết đối với biến mảng vì các thành phần của chúng sắp xếp liên tục, kề nhau). Từ đó, mỗi thành phần của danh sách sẽ là một cấu trúc, ngoài các thành phần chứa thông tin của bản thân, chúng còn phải có thêm một hoặc nhiều con trỏ để trỏ đến các thành phần tiếp theo hay đứng trước. Các cấu trúc như vậy được gọi là cấu trúc tự trỏ vì các thành phần con trỏ trong cấu trúc này sẽ trỏ đến các vùng dữ liệu có kiểu chính là kiểu của chúng.

1. Cấu trúc tự trỏ

Một cấu trúc có chứa ít nhất một thành phần con trỏ có kiểu của chính cấu trúc đang định nghĩa được gọi là cấu trúc tự trỏ. Có thể khai báo cấu trúc tự trỏ bởi một trong những cách sau:

Cách 1:

```
typedef struct <tên cấu trúc> <tên kiểu> ;          // định nghĩa tên cấu trúc
struct <tên cấu trúc>
{
      các thành phần chứa thông tin … ;
      <tên kiểu> *con trỏ ;
} ;
```

Ví dụ:

```
typedef struct Sv Sinhvien ;                    // lưu ý phải có từ khoá struct
struct Sv
{
      char hoten[30] ;        // thành phần chứa thông tin
      float diem ;            // thành phần chứa thông tin
      Sinhvien *tiep ;        // thành phần con trỏ chứa địa chỉ tiếp theo
} ;
```

Cách 2:

```
struct <tên cấu trúc>
{
    các thành phần chứa thông tin … ;
    <tên cấu trúc> *con trỏ ;
} ;
typedef <tên cấu trúc> <tên kiểu> ;          // định nghĩa tên cấu trúc tự trỏ
```

Ví dụ:

```
struct Sv
{
    char hoten[30] ;            // thành phần chứa thông tin
    float diem ;                // thành phần chứa thông tin
    Sv *tiep ;                  // thành phần con trỏ chứa địa chỉ tiếp theo
} ;
typedef Sv Sinhvien ;
```

Cách 3:

```
typedef struct <tên kiểu>       // định nghĩa tên cấu trúc tự trỏ
{
    các thành phần chứa thông tin … ;
    <tên kiểu> *con trỏ ;
} ;
```

Ví dụ:

```
typedef struct Sinhvien
{
    char hoten[30] ;            // thành phần chứa thông tin
    float diem ;                // thành phần chứa thông tin
    Sinhvien *tiep ;            // con trỏ chứa địa chỉ thành phần tiếp theo
} ;
```

Cách 4:

```
struct <tên kiểu>
{
        các thành phần chứa thông tin … ;
        <tên kiểu> *con trỏ ;
} ;
```

Ví dụ:

```
struct Sinhvien
{
        char hoten[30] ;            // thành phần chứa thông tin
        float diem ;                // thành phần chứa thông tin
        Sinhvien *tiep ;            // con trỏ chứa địa chỉ của phần tử tiếp.
} ;
```

Trong các cách trên ta thấy 2 cách khai báo cuối cùng là đơn giản nhất. C++ quan niệm các tên gọi đứng sau các từ khoá struct, union, enum là các tên kiểu (dù không có từ khoá typedef), do vậy có thể sử dụng các tên này để khai báo.

2. Khái niệm danh sách liên kết

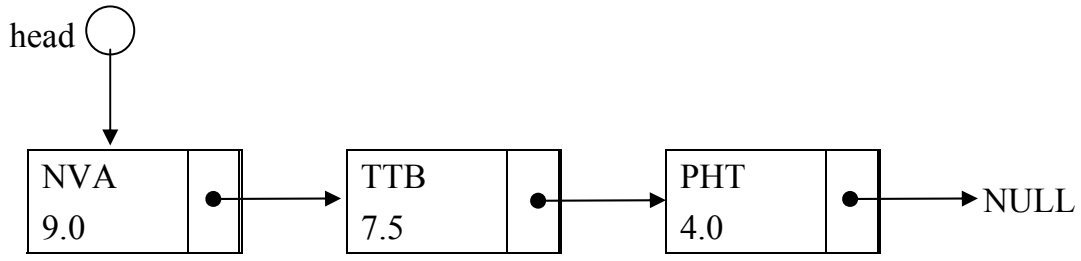
Danh sách liên kết là một cấu trúc dữ liệu cho phép thể hiện và quản lý danh sách bằng các cấu trúc liên kết với nhau thông qua các con trỏ trong cấu trúc. Có nhiều dạng danh sách liên kết phụ thuộc vào các kết nối, ví dụ:

- Danh sách liên kết đơn, mỗi cấu trúc chứa một con trỏ trỏ đến cấu trúc tiếp theo hoặc trước đó. Đối với danh sách con trỏ trỏ về trước, cấu trúc đầu tiên của danh sách sẽ trỏ về hằng con trỏ NULL, cấu trúc cuối cùng được đánh dấu bởi con trỏ last là con trỏ trỏ vào cấu trúc này. Đối với danh sách con trỏ trỏ về cấu trúc tiếp theo, cấu trúc đầu sẽ được đánh dấu bằng con trỏ head và cấu trúc cuối cùng chưa con trỏ NULL.
- Danh sách liên kết kép gồm 2 con trỏ, một trỏ đến cấu trúc trước và một trỏ đến cấu trúc sau, 2 đầu của danh sách được đánh dấu bởi các con trỏ head, last.
- Danh sách liên kết vòng gồm 1 con trỏ trỏ về sau (hoặc trước), hai đầu của danh sách được nối với nhau tạo thành vòng tròn. Chỉ cần một con trỏ head để đánh dấu đầu danh sách.

Do trong cấu trúc có chứa các con trỏ trỏ đến cấu trúc tiếp theo và/hoặc cấu trúc đứng trước nên từ một cấu trúc này chúng ta có thể truy cập đến một cấu trúc khác

(trước và/hoặc sau nó). Kết hợp với các con trỏ đánh dấu 2 đầu danh sách (head, last) chúng ta sẽ dễ dàng làm việc với bất kỳ phần tử nào của danh sách. Có thể kể một số công việc thường thực hiện trên một danh sách như: bổ sung phần tử vào cuối danh sách, chèn thêm một phần tử mới, xoá một phần tử của danh sách, tìm kiếm, sắp xếp danh sách, in danh sách …

Hình vẽ bên dưới minh hoạ một danh sách liên kết đơn quản lý sinh viên, thông tin chứa trong mỗi phần tử của danh sách gồm có họ tên sinh viên, điểm. Ngoài ra mỗi phần tử còn chứa con trỏ tiep để nối với phần tử tiếp theo của nó. Phần tử cuối cùng nối với cấu trúc rỗng (NULL).



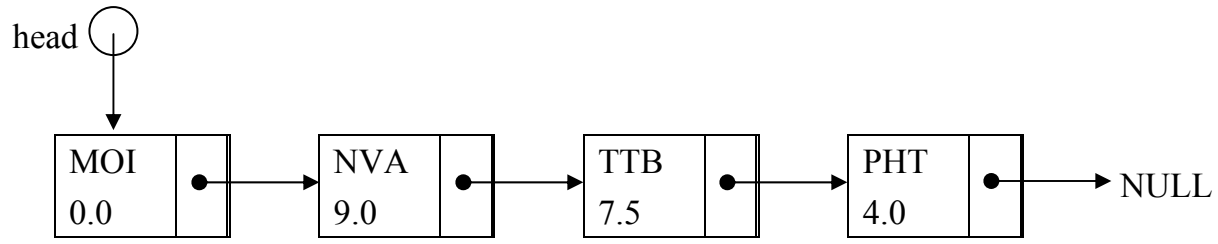
3. Các phép toán trên danh sách liên kết

Dưới đây chúng ta mô tả tóm tắt cách thức thực hiện một số thao tác trên danh sách liên kết đơn.

a. Tạo phần tử mới

Để tạo phần tử mới thông thường chúng ta thực hiện theo các bước sau đây:

- dùng toán tử new xin cấp phát một vùng nhớ đủ chứa một phần tử của danh sách.
- nhập thông tin cần lưu trữ vào phần tử mới. Con trỏ tiep được đặt bằng NULL.
- gắn phần tử vừa tạo được vào danh sách. Có hai cách:
  - hoặc gắn vào đầu danh sách, khi đó vị trí của con trỏ head (chỉ vào đầu danh sách) được điều chỉnh lại để chỉ vào phần tử mới.
  - hoặc gắn vào cuối danh sách bằng cách cho con trỏ tiep của phần tử cuối danh sách (đang trỏ vào NULL) trỏ vào phần tử mới. Nếu danh sách có con trỏ last để chỉ vào cuối danh sách thì last được điều chỉnh để trỏ vào phần tử mới. Nếu danh sách không có con trỏ last thì để tìm được phần tử cuối chương trình phải duyệt từ đầu, bắt đầu từ con trỏ head cho đến khi gặp phần tử trỏ vào NULL, đó là phần tử cuối của danh sách.

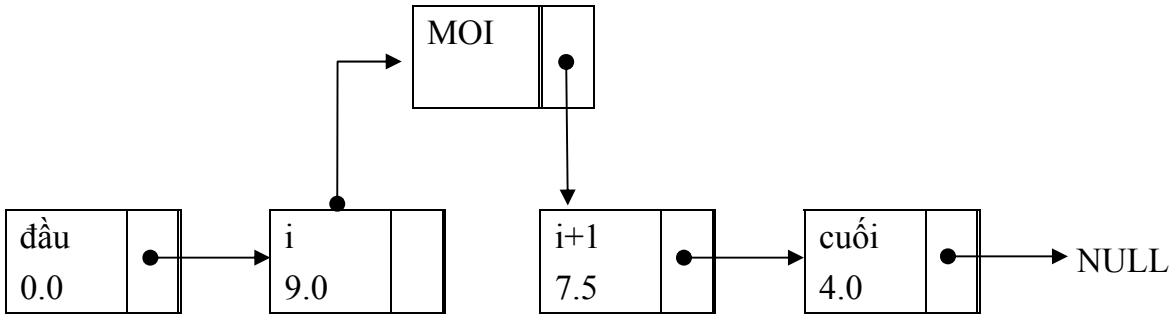


Gắn phần tử mới vào đầu danh sách

b. Chèn phần tử mới vào giữa

Giả sử phần tử mới được chèn vào giữa phần tử thứ i và i+1. Để chèn ta nối phần tử thứ i vào phần tử mới và phần tử mới nối vào phần tử thứ i+1. Thuật toán sẽ như sau:

- Cho con trỏ p chạy đến phần tử thứ i.
- Cho con trỏ tiep của phần tử mới trỏ vào phần tử thứ i+1 (tuc p->tiep).
- Cho con trỏ tiep của phần tử thứ i (hiện được trỏ bởi p) thay vì trỏ vào phần tử thứ i+1 bây giờ sẽ trỏ vào phần tử mới.



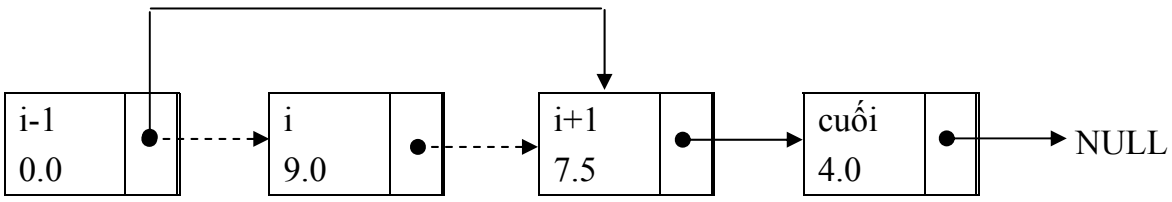
Chèn phần tử mới vào giữa phần tử i và i+1

a. Xoá phần tử thứ i khỏi danh sách

Việc xoá một phần tử ra khỏi danh sách rất đơn giản bởi chỉ việc thay đổi các con trỏ. Cụ thể giả sử cần xoá phần tử thứ i ta chỉ cần cho con trỏ tiep của phần tử thứ i-1 trỏ ("vòng qua" phần tử thứ i) vào phần tử thứ i+1. Như vậy bây giờ khi chạy trên danh sách đến phần tử thứ i-1, phần tử tiếp theo là phần tử thứ i+1 chứ không còn là phần tử thư i. Nói cách khác phần tử thứ i không được nối bởi bất kỳ phần tử nào nên nó sẽ

không thuộc danh sách. Có thể thực hiện các bước như sau:

- Cho con trỏ p chạy đến phần tử thứ i-1.
- Đặt phần tử thứ i vào biến x.
- Cho con trỏ tiep của phần tử thứ i-1 trỏ vào phần tử thứ i+1 bằng cách đặt tiep = x.tiep.
- Giải phóng bộ nhớ được trỏ bởi x bằng câu lệnh delete x.



Xóa phần tử thứ i

c. Duyệt danh sách

Duyệt là thao tác đi qua từng phần tử của danh sách, tại mỗi phần tử chương trình thực hiện một công việc gì đó trên phần tử mà ta gọi là thăm phần tử đó. Một phép thăm có thể đơn giản là hiện nội dung thông tin của phần tử đó ra màn hình chẳng hạn. Để duyệt danh sách ta chỉ cần cho một con trỏ p chạy từ đầu đến cuối danh sách đến khi phần tử cuối có con trỏ tiep = NULL thì dừng. Câu lệnh cho con trỏ p chuyển đến phần tử tiếp theo của nó là:

p = p → tiep ;

d. Tìm kiếm

Cho một danh sách trong đó mỗi phần tử của danh sách đều chứa một trường gọi là trường khoá, thường là các trường có kiểu cơ sở hoặc kết hợp của một số trường như vậy. Bài toán đặt ra là tìm trên danh sách phần tử có giá trị của trường khoá bằng với một giá trị cho trước. Tiến trình thực hiện nhiệm vụ thựcchất cũng là bài toán duyệt, trong đó thao tác "thăm" chính là so sánh trường khoá của phần tử với giá trị cho trước, nếu trùng nhau ta in kết quả và dừng, Nếu đã duyệt hết mà không có phần tử nào có trường khoá trùng với giá trị cho trước thì xem danh sách không chứa giá trị này.

Ngoài các thao tác trên, nói chung còn nhiều các thao tác quen thuộc khác tuy nhiên chúng ta không trình bày ở đây vì nó không thuộc phạm vi của giáo trình này.

Dưới đây là một ví dụ minh hoạ cho cấc cấu trúc tự trỏ, danh sách liên kết và một vài thao tác trên danh sách liên kết thông qua bài toán quản lý sinh viên.

- Khai báo

```
struct DATE
{
        int day, month, year;                   // ngày, tháng, năm
};
struct Sinhvien {                               // cấu trúc tự trỏ
        char hoten[31];
        DATE ns;
        float diem;
        Sinhvien *tiep ;
};
Sinhvien *dau = NULL, *cuoi = NULL;     // Các con trỏ tới đầu và cuối ds
Sinhvien *cur = NULL;                   // Con trỏ tới sv hiện tại
int sosv = 0;                           // Số sv của danh sách
```

- Tạo sinh viên mới và nhập thông tin, trả lại con trỏ trỏ đến sinh viên mới.

```
Sinhvien* Nhap1sv()                     // Tạo 1 khối dữ liệu cho sv mới
{
        Sinhvien *kq = new Sinhvien[1] ;        // Cấp phát bộ nhớ cho kq
        cout << "\nSinh vien thu ", sosv+1 ;
        cout << "Ho ten = " ; cin.getline(kq->hoten);
        cout << "Ns = " ; cin >> kq->ns.day >> kq->ns.month >> kq->ns.year;
        cout << "Diem = " ; cin >> kq->diem ; cin.ignore() ;
        kq->tiep = NULL;
        return kq ;
}
```

- Bổ sung sinh viên mới vào cuối danh sách.

```
void Bosung()                           // Bổ sung sv mới vào cuối ds
{
        cur = Nhap1sv();
```

```
        if (sosv == 0) {dau = cuoi = cur;}
        else { cuoi->tiep = cur; cuoi = cur; }
        sosv++;
    }
```

- Chèn sv mới vào trước sinh viên thứ n.

```
    void Chentruoc(int n)                    // Chèn sv mới vào trước sv thứ n
    {
        cur = Nhap1sv();
        if (sosv==0) { dau = cuoi = cur; sosv++; return; }
        if (sosv==1 || n==1) {cur->tiep = dau; dau = cur; sosv++; return;}
        Sinhvien *truoc, *sau;
        truoc = dau;
        sau = dau -> tiep;
        for (int i=1; i<n-1; i++) truoc = truoc->tiep;
        sau = truoc->tiep;
        truoc->tiep = cur;
        cur -> tiep = sau;
        sosv ++;
    }
```

- Chèn sv mới vào sau sinh viên thứ n.

```
    void Chensau(int n)                      // Chèn sv mới vào sau sv thứ n
    {
        cur = Nhap1sv();
        if (sosv==0 || sosv<n) { dau = cuoi = cur; sosv++; return; }
        Sinhvien *truoc, *sau;
        truoc = dau; sau = dau -> tiep;
        for (int i=1; i<n; i++) truoc = truoc->tiep;
        sau = truoc->tiep;
        truoc->tiep = cur;
```

```
        cur -> tiep = sau;
        sosv ++;
    }
```

- Xoá sinh viên thứ n.

```
    void Xoa(int n)                                    // Xoá sinh viên thứ n
    {
        if (sosv==1&&n==1) { delete dau ; dau = cuoi = NULL; sosv--; return; }
        if (n==1) { cur = dau; dau = cur->tiep; delete cur; sosv--; return; }
        Sinhvien *truoc, *sau;
        truoc = dau;
        sau = dau -> tiep;
        for (int i=1; i<n-1; i++) truoc = truoc->tiep;
        cur = truoc->tiep; sau = cur->tiep; truoc->tiep = sau;
        delete cur ;
        sosv --;
    }
```

- Tạo danh sách sinh viên.

```
    void Taods()                                       // Tạo danh sách
    {
        int tiep = 1;
        while (tiep) {
            Bosung();
            cout << "Tiep (0/1) ? " ; cin >> tiep ;
        }
    }
```

- In danh sách sinh viên.

```
    void Inds()                                        // In danh sách
    {
```

```
        cur = dau;        int i=1;
        while (cur != NULL) {
            cout << "\nSinh vien thu " << i << " ----------\n") ;
            cout << "Hoten:" << cur->hoten ;
            cout << "Ngay sinh: "
            cout << cur -> ns.day << "/" ;
            cout << cur -> ns.month << "/" ;
            cout << cur -> ns.year ;
            cout << "Diem: " << cur->diem ;
            cur = cur->tiep; i++;
        }
    }
```

- Hàm chính.

```
    void main()
    {
        clrscr();
        Taods();
        Inds();
        getch();
    }
```

III. KIỂU HỢP

1. Khai báo

Giống như cấu trúc, kiểu hợp cũng có nhiều thành phần nhưng các thành phần của chúng sử dụng chung nhau một vùng nhớ. Do vậy kích thước của một kiểu hợp là độ dài của trường lớn nhất và việc thay đổi một thành phần sẽ ảnh hưởng đến tất cả các thành phần còn lại.

```
union <tên kiểu> {
    Danh sách các thành phần;
};
```

2. Truy cập

Cú pháp truy cập đến các thành phần của hợp cũng tương tự như kiểu cấu trúc, tức cũng sử dụng toán tử lấy thành phần (dấu chấm . hoặc → cho biến con trỏ kiểu hợp).

Dưới đây là một ví dụ minh hoạ việc sử dụng khai báo kiểu hợp để tách byte thấp, byte cao của một số nguyên.

Ví dụ 1 :

```
void main()
{
    union songuyen {
        int n;
        unsigned char c[2];
    } x;
    cout << "Nhập số nguyên: " ; cin >> x.n ;
    cout << "Byte thấp của x = " << x.c[0] << endl ;
    cout << "Byte cao của x = " << x.c[1] << endl;
}
```

Ví dụ 2 : Kết hợp cùng kiểu nhóm bit trong cấu trúc, chúng ta có thể tìm được các bit của một số như chương trình sau. Trong chương trình ta sử dụng một biến u có kiểu hợp. Trong kiểu hợp này có 2 thành phần là 2 cấu trúc lần lượt có tên s và f.

```
union {
    struct { unsigned a, b ; } s;
    struct {
        unsigned n1: 1;
        unsigned: 15;
        unsigned n2: 1;
        unsigned: 7;
        unsigned n3: 8;
    } t ;
} u;
```

với khai báo trên đây khi nhập u.s thì nó cũng ảnh hưởng đến u.t, cụ thể

- u.t.n1 là bit đầu tiên (0) của thành phần u.s.a
- u.t.n2 là bit 0 của thành phần u.s.b
- u.t.n3 là byte cao của u.s.b

IV. KIỂU LIỆT KÊ

Có thể gán các giá trị nguyên liên tiếp (tính từ 0) cho các tên gọi cụ thể bằng kiểu liệt kê theo khai báo sau đây:

enum tên_kiểu { d/s tên các giá trị };

Ví dụ:

```
enum Bool {false, true};
```

khai báo kiểu mới đặt tên Bool chỉ nhận 1 trong 2 giá trị đặt tên false và true, trong đó false ứng với giá trị 0 và true ứng với giá trị 1. Cách khai báo kiểu enum trên cũng tương đương với dãy các macro sau:

```
#define false 0
#define true 1
```

Với kiểu Bool ta có thể khai báo một số biến như sau:

```
Bool Ok, found;
```

hai biến Ok và found sẽ chỉ nhận 1 trong 2 giá trị false (thay cho 0) hoặc true (thay cho 1). Có nghĩa có thể gán:

```
Ok = true;
```

hoặc: `found = false;`

Tuy nhiên không thể gán các giá trị nguyên trực tiếp cho các biến enum mà phải thông qua ép kiểu. Ví dụ:

```
Ok = 0;              // sai
Ok = Bool(0) ;       // đúng
hoặc Ok = false ;    // đúng
```

BÀI TẬP

1. Có thể truy nhập thành phần của cấu trúc thông qua con trỏ như sau (với p là con trỏ cấu trúc và a là thành phần của cấu trúc):

A: (*p).a B: *p→a C: a và b sai D: a và b đúng

2. Cho khai báo struct T {int x; float y;} t, *p, a[10]; Câu lệnh nào trong các câu sau là không hợp lệ:

(1) p = &t; (2) p = &t.x; (3) p = a;

(4) p = &a (5) p = &a[5]; (6) p = &a[5].y;

A: 1, 2 và 3 B: 4, 5 và 6 C: 1, 3 và 5 D: 2, 4 và 6

3. Cho các khai báo sau:

```
struct ngay {int ng, th, nam;} vaotruong, ratruong;
typedef struct {char hoten[25]; ngay ngaysinh;} sinhvien;
```

Hãy chọn câu đúng nhất

A: Không được phép gán: ratruong = vaotruong;

B: sinhvien là tên cấu trúc, vaotruong, ratruong là biến cấu trúc

C: Có thể viết: vaotruong.ng, ratruong.th, sinhvien.vaotruong.nam để truy nhập đến các thành phần tương ứng.

D: a, b, c đúng

4. Trong các khởi tạo giá trị cho các cấu trúc sau, khởi tạo nào đúng:

```
struct S1 {
        int ngay, thang, nam;
} s1 = {2,3};
struct S2 {
        char hoten[10];
        struct S1 ngaysinh;
} s2 = {"Ly Ly",1,2,3};
struct S3 {
        struct S2 sinhvien;
        float diem;
} s3 = {{{"Cốc cốc", {4,5,6}}, 7};
```

A: S1 và S2 đúng B: S2 và S3 đúng C: S3 và S1 đúng D: Cả 3 cùng đúng

5. Đối với kiểu cấu trúc, cách gán nào dưới đây là không được phép:

A: Gán hai biến cho nhau.

B: Gán hai phần tử mảng (kiểu cấu trúc) cho nhau

C: Gán một phần tử mảng (kiểu cấu trúc) cho một biến và ngược lại

D: Gán hai mảng cấu trúc cùng số phần tử cho nhau

6. Cho đoạn chương trình sau:

```
struct {
    int to ;
    float soluong;
} x[10];
for (int i = 0; i < 10; i++) cin >> x[i].to >> x[i].soluong ;
```

Chọn câu đúng nhất trong các câu sau:

A: Đoạn chương trình trên có lỗi cú pháp

B: Không được phép sử dụng toán tử lấy địa chỉ đối với các thành phần to và soluong

C: Lấy địa chỉ thành phần soluong dẫn đến chương trình hoạt động không đúng đắn

D: Cả a, b, c đều sai

7. Chọn câu đúng nhất trong các câu sau:

A: Các thành phần của kiểu hợp (union) được cấp phát một vùng nhớ chung

B: Kích thước của kiểu hợp bằng kích thước của thành phần lớn nhất

C: Một biến kiểu hợp có thể được tổ chức để cho phép thay đổi được kiểu dữ liệu của biến trong qua trình chạy chương trình

D: a, b, c đúng

8. Cho khai báo:

```
union {
    unsigned x;
    unsigned char y[2];
} z = {0xabcd};
```

Chọn câu đúng nhất trong các câu sau:

A: Khai báo trên là sai vì thiếu tên kiểu

B: Khởi tạo biến z là sai vì chỉ có một giá trị (0xabcd)

C: z.y[0] = 0xab

D: z.y[1] = 0xab

9. Cho kiểu hợp:

```
union U {
    char x[1];
    int y[2]; float z[3];
} u;
```

Chọn câu đúng nhất trong các câu sau:

A: sizeof(U) = 1+2+3 = 6

B: sizeof(U) = max(sizeof(char), sizeof(int), sizeof(float))

C: sizeof(u) = max(sizeof(u.x), sizeof(u.y), sizeof(u.z))

D: b và c đúng

10. Cho khai báo:

```
union {
    unsigned x;
    struct {
        unsigned char a, b;
    } y;
} z = {0xabcd};
```

Giá trị của z.y.a và z.y.b tương ứng:

A: 0xab, 0xcd B: 0xcd, 0xab C: 0xabcd, 0 D: 0, 0xabcd

11. Cho khai báo:

```
union {
    struct {
        unsigned char a, b;
    } y;
    unsigned x;
```

```
} z = {{1,2}};
```

Giá trị của z.x bằng:

A: 513 B: 258

C: Không xác định vì khởi tạo sai D: Khởi tạo đúng nhưng z.x chưa có giá trị

12. Xét đoạn lệnh:

```
union U {
    int x; char y;
} u;
u.x = 0; u.y = 200;
```

Tìm giá trị của u.x + u.y ?

A: 122 B: 144 C: 200 D: 400

13. Cho số phức dưới dạng cấu trúc gồm 2 thành phần là thực và ảo. Viết chương trình nhập 2 số phức và in ra tổng, tích, hiệu, thương của chúng.

14. Cho phân số dưới dạng cấu trúc gồm 2 thành phần là tử và mẫu. Viết chương trình nhập 2 phân số, in ra tổng, tích, hiệu, thương của chúng dưới dạng tối giản.

15. Tính số ngày đã qua kể từ đầu năm cho đến ngày hiện tại. Qui ước ngày được khai báo dưới dạng cấu trúc và để đơn giản một năm bất kỳ được tính 365 ngày và tháng bất kỳ có 30 ngày.

16. Nhập một ngày tháng năm dưới dạng cấu trúc. Tính chính xác (kể cả năm nhuận) số ngày đã qua kể từ ngày 1/1/1 cho đến ngày đó.

17. Tính khoảng cách giữa 2 ngày tháng bất kỳ.

18. Hiện thứ của một ngày bất kỳ nào đó, biết rằng ngày 1/1/1 là thứ hai.

19. Hiện thứ của một ngày bất kỳ nào đó, lấy ngày thứ hiện tại để làm chuẩn.

20. Viết chương trình nhập một mảng sinh viên, thông tin về mỗi sinh viên gồm họ tên và ngày sinh (kiểu cấu trúc). Sắp xếp mảng theo tuổi và in ra màn hình

21. Để biểu diễn số phức có thể sử dụng định nghĩa sau:

```
typedef struct {  float re, im; } sophuc;
```

Cần bổ sung thêm trường nào vào cấu trúc để có thể lập được một danh sách liên kết các số phức.

22. Để tạo danh sách liên kết, theo bạn sinh viên nào dưới đây khai báo đúng cấu trúc tự trỏ sẽ được dùng:

Sinh viên 1: struct SV {char ht[25]; int tuoi; struct SV *tiep;};

Sinh viên 2: typedef struct SV node; struct SV {char ht[25]; int tuoi; node *tiep;};

Sinh viên 3: typedef struct SV {char ht[25]; int tuoi; struct SV *tiep;} node;

A: Sinh viên 1 B: Sinh viên 2 C: Sinh viên 2 và 3 D: Sinh viên 1, 2 và 3

23. Lập danh sách liên kết chứa bảng chữ cái A, B, C … Hãy đảo phần đầu từ A .. M xuống cuối thành N, O, … Z, A, …M.

24. Viết chương trình tìm người cuối cùng trong trò chơi: 30 người xếp vòng tròn. Đếm vòng tròn (bắt đầu từ người số 1) cứ đến người thứ 7 thì người này bị loại ra khỏi vòng. Hỏi người còn lại cuối cùng ?

25. Giả sử có danh sách liên kết mà mỗi nốt của nó lưu một giá trị nguyên. Viết chương trình sắp xếp danh sách theo thứ tự giảm dần.

26. Giả sử có danh sách liên kết mà mỗi nốt của nó lưu một giá trị nguyên được sắp giảm dần. Viết chương trình cho phép chèn thêm một phần tử vào danh sách sao cho danh sách vẫn được sắp giảm dần.

27. Tạo danh sách liên kết các số thực x_1, x_2, ..., xn. Gọi m là trung bình cộng: $m = \frac{x_1 + x_2 + ... + x_n}{n}$.

Hãy in lần lượt ra màn hình các giá trị: m, x_1 - m, x_2 - m, ..., xn - m.

28. Sử dụng kiểu union để in ra byte thấp, byte cao của một số nguyên.

CHƯƠNG 6

ĐỒ HOẠ VÀ ÂM THANH

Đồ hoạ
Âm thanh

I. ĐỒ HOẠ

1. Khái niệm đồ hoạ

a. Điểm ảnh và độ phân giải

Màn hình ở chế độ đồ hoạ là tập hợp các điểm (pixel-picture elements) ảnh. Số điểm ảnh và cách bố trí theo chiều ngang, dọc của màn hình được gọi là độ phân giải (resolution). Vì vậy độ phân giải thường được đặc trưng bởi một cặp số chỉ định số điểm ảnh theo chiều ngang và chiều dọc của màn hình. Ví dụ màn hình VGA ở mode 2 có độ phân giải là 640x480, tức trên mỗi dòng ngang của màn hình có thể vẽ được 640 điểm ảnh và trên mỗi cột dọc vẽ được 480 điểm ảnh. Các cột và dòng được đánh số từ 0, theo chiều từ trái sang phải (đối với cột) và từ trên xuống dưới (đối với dòng). Một điểm ảnh hay còn gọi là pixel là giao điểm của một cột và một dòng nào đó trên màn hình và vị trí của nó được thể hiện bởi cặp toạ độ (x,y) với x biểu diễn cho cột và y biểu diễn cho dòng. Ví dụ với màn hình trên điểm ảnh "đầu tiên" nằm ở góc trên bên trái của màn hình có toạ độ (0,0) và điểm "cuối cùng" ở góc dưới bên phải có toạ độ (639,479). Điểm có toạ độ (150,200) là giao điểm của cột thứ 150 và dòng 200.

b. Trình điều khiển đồ hoạ

Màn hình đồ hoạ có nhiều loại khác nhau. Mỗi loại màn hình cần có trình điều khiển tương ứng. C cung cấp các trình điều khiển màn hình trong thư mục BGI đặt dưới thư mục gốc của C (TC hoặc BC) gồm có:

Tên trình điều khiển	**Kiểu màn hình đồ hoạ**
ATT.BGI	ATT & T6300 (400 dòng)
CGA.BGI	IBMCGA, MCGA và các máy tương thích
EGAVGA.BGI	IBM EGA, VGA và các máy tương thích

HERC.BGI	Hercules mono và các máy tương thích
IBM8514.BGI	IBM 8514 và các máy tương thích
PC3270.BGI	IBM 3270 PC

Ngoài các trình điều khiển trong thư mục BGI còn chứa các file font chữ có đuôi CHR gồm:

GOTH.CHR

LITT.CHR

SANS.CHR

TRIP.CHR

c. Mốt (mode) đồ hoạ

Mỗi màn hình đồ hoạ có thể hoạt động dưới nhiều mốt khác nhau. Độ phân giải của màn hình phụ thuộc vào từng mốt. Ví dụ màn hình VGA có thể hoạt động dưới các mốt 0 (VGALO: độ phân giải thấp 640x200), 1 (VGAMED: độ phân giải trung bình 640x350), 2 (VGAHI: độ phân giải cao 640x480).

2. Vào/ra chế độ đồ hoạ

Trong C++ các hàm liên quan đến đồ hoạ được khai báo trong tệp <graphics.h>

a. Khởi động chế độ đồ hoạ

void initgraph(int *graphdriver, int *graphmode, char *drivepath)

- drivepath: đường dẫn của thư mục chứa các trình điều khiển đồ hoạ. Nếu rỗng sẽ tìm trong thư mục hiện tại.
- graphdriver, graphmode: Chỉ định trình quản lý và mốt màn hình cần sử dụng. Trong đó graphdriver có thể nhận 1 trong các giá trị sau:

DETECT	0
CGA	1
EGA	3
EGA64	4
EGAMONO	5
VGA	9
.....	..

Hiển nhiên việc chọn giá trị của graphdriver phải tương ứng với màn hình thực tế. Trong trường hợp ta không biết chủng loại thực tế của màn hình có thể sử dụng giá trị DETECT (hoặc 0) là giá trị chỉ định cho chương trình tự tìm hiểu về màn hình và gọi trình điều khiển tương ứng. Trong trường hợp này graphmode sẽ được gán giá trị tự động với mode có độ phân giải cao nhất có thể. Về graphmode có thể nhận các giá trị sau:

CGAC0	0	320 x 200	
CGAC1	1	320 x 200	
CGAC2	2	320 x 200	
CGAC3	3	320 x 200	
CGAHI	4	640 x 200	2 color
EGALO	0	640 x 200	16 color
EGAHI	1	640 x 350	16 color
EGA64LO	0	640 x 200	16 color
EGA64HI	1	640 x 350	4 color
VGALO	0	640 x 200	16 color
VGAMED	0	640 x 350	16 color
VGAHI	0	640 x 480	16 color

Trong quá trình sử dụng để xoá màn hình đồ hoạ ta dùng hàm **cleardevice();**

b. Kết thúc chế độ đồ hoạ

Để kết thúc chế độ đồ hoạ về lại chế độ văn bản ta sử dụng hàm **closegraph();**

c. Lỗi đồ hoạ

- Sau mỗi thao tác đồ hoạ, hàm **graphresult()** sẽ cho giá trị 0 nếu không có lỗi, hoặc các giá trị âm (-1 .. -18) tương ứng với lỗi. Hàm **grapherrormsg(n)** trả lại nội dung lỗi và mã lỗi.

Mã lỗi	Hằng lỗi (graphresult())	Nội dung lỗi (grapherrormsg())
0	grOk	No error
-1	grNoInitGraph	(BGI) Không có BGI
-2	grNotDetected	Graphics hardware not detected

-3	grFileNotFound	Device driver file not found

.....

Ví dụ 1 :

Ví dụ sau đây khởi tạo chế độ đồ hoạ với graphdriver = 0 (DETECT) và thông báo lỗi nếu không thành công hoặc thông báo chế độ đồ hoạ cũng như mode màn hình. Để biết độ phân giải của màn hình có thể dùng các hàm **getmaxx()** (số cột) và **getmaxy()** (số dòng)

```
void main()
{
    int gd = DETECT, gm, maloi;
    initgraph(&gd, &gm, "C:\\BC\\BGI");
    maloi = graphresult();
    if (maloi != grOk)
    {
        cout << "Lỗi: " << grapherrormsg(maloi)) << endl;
        cout << "An phím bất kỳ để dừng "; getch();
        exit(1);
    } else {
        cout << "Chế độ màn hình = " << gd << endl;
        cout << "Mode màn hình = " << gm << endl;
        cout << "Độ phân giải: " << getmaxx() << "," << getmaxy() << endl;
        getch();
    }
    closegraph();
}
```

Các phần tiếp theo sau đây sẽ cung cấp các câu lệnh để vẽ trong chế độ đồ họa.

3. Vẽ điểm, đường, khối, màu sắc

a. Màu sắc

- **getmaxcolor()**: Trả lại số hiệu (hằng) tương ứng với màu tối đa của màn hình hiện tại. Do các hằng màu được tính từ 0 nên số màu sẽ bằng hằng trả lại cộng

thêm 1.

- **setbkcolor(màu):** Đặt màu nền. Có tác dụng với văn bản và các nét vẽ.
- **setcolor(màu):** Đặt màu vẽ. Có tác dụng với văn bản và các nét vẽ.
- **getbkcolor():** Trả lại màu nền hiện tại.
- **getcolor():** Trả lại màu vẽ hiện tại (từ 0 đến getmaxcolor()).

Ví dụ: In toạ độ tại vị trí hiện tại của con trỏ màn hình. Trong ví dụ này chúng ta sử dụng câu lệnh sprintf(xâu s, "dòng đk", các biểu thức cần in), câu lệnh này sẽ thay việc in các biểu thức ra màn hình thành in ra xâu s (tức tạo xâu s từ các biểu thức). Ngoài ra hàm outtextxy(x, y, s) sẽ in xâu s tại vị trí (x,y).

```
void intoado(dx, dy, color)
{
    int oldcolor;
    oldcolor = getcolor();
    setcolor(color);
    char td[10];
    sprintf(td, "(%d, %d)", getx(), gety());
    outtextxy(getx()+dx, gety()+dy, td);
    setcolor(oldcolor);
}
```

b. Vẽ điểm

- **putpixel(x, y, c)**: Vẽ điểm (x, y) với màu c.
- **getpixel (x, y)**: Trả lại màu tại điểm (x, y).

Ví dụ 2 : Vẽ bầu trời sao

```
void sky()
{
    int maxx, maxy, maxc;
    int i, xarr[3001], yarr[3001];
    maxx = getmaxx();
    maxy = getmaxy();
    maxc = getmaxcolor();
```

```
        randomize();
        for (i=1;i<3001;i++) {xarr[i]=random(maxx);yarr[i]=random(maxy);}
        while (!kbhit()) {
            for (i=1;i<3001;i++)
            {
                putpixel(xarr[i], yarr[i], random(maxc));delay(1);
            }
        for (i=1;i<3001;i++)
        if (getpitxel(xarr[i], yarr[i]) == random(maxc))
            putpitxel(xarr[i], yarr[i], 0);
        }
    }
```

c. Vẽ đường thẳng và gấp khúc

- **line(x1, y1, x2, y2):** Vẽ đường thẳng từ (x1, y1) đến (x2, y2). Con trỏ màn hình vẫn đứng tại vị trí cũ.
- **lineto(x, y):** Vẽ đường thẳng từ vị trí hiện tại của con trỏ đến vị trí (x, y). con trỏ chuyển về (x, y).
- **linerel(dx, dy):** Gọi (x, y) là vị trí hiện tại của con trỏ, lệnh này sẽ vẽ đường thẳng nối (x, y) với điểm mới có tọa độ (x+dx, y+dy). Tức lệnh này cũng tương đương với lệnh lineto(getx()+dx, gety()+dy), trong đó getx() và gety() là hai hàm trả lại vị trí x, y hiện tại của con trỏ. Lệnh linerel sau khi thực hiện xong sẽ đặt con trỏ tại vị trí cuối của đường thẳng vừa vẽ.

Ví dụ 3 : Vẽ hình bao thư bằng 1 nét.

```
void baothu()
{
    setbkcolor(1);
    setcolor(YELLOW);
    moveto(100, 100);
    lineto(300, 100); lineto(300, 200); lineto(100, 200); lineto(100, 100);
    lineto(200, 50); lineto(300, 100);
}
```

- **rectangle(x1, y1, x2, y2)**: Vẽ hình khung chữ nhật với góc trên bên trái có tọa độ (x1, y1) và góc dưới bên phải có tọa độ (x2, y2).
- **bar(x1, y1, x2, y2):** Vẽ hình chữ nhật đặc. Màu khung được đặt bởi *setcolor* và màu nền lẫn mẫu tô nền được đặt bởi lệnh *setlinestyle*. Mẫu nền ngầm định là đặc và màu là getmaxcolor.
- **bar3d(x1, y1, x2, y2, c, top):** Vẽ hình trụ chữ nhật với đáy là (x1, y1, x2, y2) và độ cao c, nếu top = 1 hình sẽ có nắp và nếu top = 0 hình không có nắp.

Ví dụ : Vẽ các hình khối chữ nhật với mầu nền và mẫu tô khác nhau.

```
void main()
{
        int gdriver = DETECT, gmode;
        initgraph(&gdriver, &gmode, "c:\\borlandc\\bgi");
        int midx = getmaxx() / 2;
        int midy = getmaxy() / 2;
        for (int i=SOLID_FILL; i<USER_FILL; i++)
        {
                setfillstyle(i, i);
                bar3d(midx-50, midy-50, midx+50, midy+50, 100, 0);
                getch();
        }
        closegraph();
}
```

Ghi chú: để xoá điểm hoặc đường ta vẽ lại điểm hoặc đường đó bằng màu nền hiện tại. Để biết màu nền hiện tại ta sử dụng hàm getbkcolor().

d. Các thuộc tính về đường (kiểu đường, độ rộng)

- **setlinestyle(style, pattern, width)**: đặt các thuộc tính về đường vẽ, trong đó style là kiểu đường, pattern là mẫu tô và width là độ đậm của đường vẽ. Các thuộc tính này được giải thích bên dưới.
- **getlinesettings(struct linesettingstype *info)**: Lấy các thuộc tính về đường vẽ hiện tại cho vào biến được trỏ bởi info.
- Kiểu của biến chứa các thuộc tính đường vẽ:

```
struct linesettingstype {
    int linetsyle;
    int upattern;
    int thickness;
}
```

- Các hằng số qui định các kiểu đường (style):

```
style:    SOLID_LINE = 0
          DOTTED_LINE = 1
          CENTER_LINE = 2
          DASHED_LINE = 3
          USERBIT_LINE = 4,        // Kiểu đường do NSD định nghĩa
```

- pattern: Do NSD định nghĩa theo 2 byte cho một đường. Chỉ có tác dụng khi style = 4.
- Các hằng số qui định độ đậm (độ dày) của đường (width):

```
          NORM_WIDTH = 1
          THICK_WIDTH = 3
```

Ví dụ 4 :

```
void netve()
{
    char *lname[] = {"Duong lien net", "Duong cham cham",
    "Duong trung tam", "Duong dut net", "Duong do NSD dinh nghia" };
    int style, midx, midy, mauNSD;
    midx = getmaxx() / 2; midy = getmaxy() / 2;
    // Mẫu đường được định nghĩa bởi NSD "0000000000000001"
    mauNSD = 1;
    for (style=SOLID_LINE; style<=USERBIT_LINE; style++) {
        setlinestyle(style, mauNSD, 1);
        line(0, 0, midx-10, midy);
        rectangle(0, 0, getmaxx(), getmaxy());
        outtextxy(midx, midy, lname[style]);
```

```
            line(midx, midy+10, midx+8*strlen(lname[style]), midy+10);
            getch();
            cleardevice();
        }
    }
```

e. Các thuộc tính về hình (mẫu tô, màu tô)

- setfillstyle(mẫu tô, màu tô): Đặt mẫu tô, màu tô
- setfillpattern(mẫu tô, màu tô): Định nghĩa mẫu tô.
- getfillsettings(struct fillsettingstype *info): Lấy mẫu tô hiện tại

```
struct fillsettingstype {
        int pattern;
        int color;
};
```

- **getfillpattern(mẫu tô)**: Trả lại mẫu tô hiện do NSD định nghĩa. Là một con trỏ trỏ đến mảng 8 kí tự. Sau đây là một số mẫu tô và các hằng tương ứng

EMPTY_FILL	0
SOLID_FILL	1
LINE_FILL	2
LTSLASH_FILL	3
SLASH_FILL	4
BKSLASH_FILL	5
LTBKSLASH_FILL	6
HATCH_FILL	7
XHATCH_FILL	8
INTERLEAVE_FILL	9
WIDE_DOT_FILL	10
CLOSE_DOT_FILL	11
USER_FILL	12

Ví dụ 5 : Đặt mẫu tô.

```
char caro[8] = {0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55};
```

```
maxx = getmaxx();
maxy = getmaxy();
setfillpattern(caro, getmaxcolor());
// Tô màn hình theo mẫu
bar(0, 0, maxx, maxy);
getch();
```

f. Vẽ đa giác

- **drawpoly(số đỉnh, vị trí đỉnh)**: Vẽ đường đa giác theo setlinestyle;
- **fillpoly(số đỉnh, vị trí đỉnh)**: Vẽ hình đa giác đặc theo setfillstyle;

  Vị trí đỉnh là con trỏ trỏ đến dãy các toạ độ, thông thường dùng mảng.

  Để vẽ đa giác đóng phải đưa ra n+1 toạ độ trong đó toạ độ n = toạ độ 0.

Ví dụ 6 :

```
int poly[10];
poly[0] = 20; poly[1] = maxy / 2;              // đỉnh thứ nhất
poly[2] = maxx - 20; poly[3] = 20;             // đỉnh thứ hai
poly[4] = maxx - 50; poly[5] = maxy - 20;      // đỉnh thứ ba
poly[6] = maxx / 2; poly[7] = maxy / 2;        // đỉnh thứ tư
poly[8] = poly[0]; poly[9] = poly[1];
// vẽ đa giác
drawpoly(5, poly);
```

g. Vẽ đường cong

- **arc(x, y, góc đầu, góc cuối, bán kính):** Vẽ cung tròn có tâm (x, y) với các góc và bán kính tương ứng.
- **circle(x, y, bán kính):** Vẽ đường tròn có tâm tại (x, y).
- **pieslice(x, y, góc đầu, góc cuối, bán kính):** Vẽ hình quạt tròn đặc với mẫu hiện tại;
- **ellipse(x, y, góc đầu, góc cuối, bkx, bky):** Vẽ cung elip với tâm, các góc và các bán kính theo hoàng độ và tung độ tương ứng.
- **fillellipse(x, y, bkx, bky):** Vẽ hình elip đặc.
- sector(x, y, góc đầu, góc cuối, bkx, bky): Vẽ hình quạt elip.

Chú ý: Nếu góc đầu = 0 và góc cuối = 360 cung, lệnh trên sẽ vẽ đường tròn hoặc elip.

Ví dụ 7 : Vẽ đường tròn và elip.

```
arc(200, 200, 45, 135, 100) ;          // cung tròn
arc(200, 200, 0, 360, 100) ;           // đường tròn
circle(200, 200, 100) ;                // đường tròn
ellipse(200, 200, 45, 135, 100, 80) ;  // cung elip
ellipse(200, 200, 0, 360, 100, 80) ;   // đường elip;
setfillstyle(EMPTY_FILL, getmaxcolor());
pieslice(200, 200, 45, 135, 100) ;     // đường quạt tròn
fillellipse(200, 200, 0, 360, 100, 80) ;  // đường elip
setfillstyle(SOLID_FILL, getmaxcolor());
pieslice(200, 200, 45, 135, 100);      // hình quạt tròn;
circle(200, 200, 100);                 // hình tròn;
fillellipse(200, 200, 0, 360, 100, 80);  //  hình elip;
sector(200, 200, 45, 135, 100, 80);    // hình quạt elip
```

h. Tô mầu

- **floodfill(x, y, c)**: Tô màu một hình kín chứa điểm x, y và màu viền c. Mầu dùng để tô được đặt bởi hàm setfillstyle(kiểu tô, màu tô). Ví dụ:

```
void fill()
{
    rectangle(100, 100, 180, 140);          // Vẽ hình chữ nhật
    setfillstyle(1, BLUE);                  // Mẫu tô đặc, màu xanh
    floodfill(120, 120, 15);                // Tô hình chữ nhật đã vẽ
    int tg[8] = {150, 120, 180, 280, 350, 180, 150, 120};
    drawpoly(4, tg);
    setfillstyle(2, RED);
    floodfill(180, 200, 15);
    circle(380, 210, 100);
    setfillstyle(3, GREEN);
    floodfill(380, 210, 15);
```

```
}

void fill2()                          // Vẽ và tô màu dãy đường tròn liên tiếp
{
      int i, x = 0, y = 0, r = 0;
      for (i=1;i<10;i++) {
            r = 10*i;
            y = x += r;
            circle(x, y, r);
            setfillstyle(i, i);
            floodfill(x, y, 15);
      }
}
```

4. Viết văn bản trong màn hình đồ họa

a. Viết văn bản

outtext(s) ;

outtextxy(x, y, s) ;

Câu lệnh trên cho phép viết xâu kí tự tại vị trí con trỏ trên màn hình đồ họa. Câu lệnh tiếp theo cho phép viết s ra tại vị trí (x, y). Vị trí con trỏ sau khi thực hiện outtext(s) sẽ đặt tại vị trí cuối của xâu được in trong khi vị trí con trỏ sau khi thực hiện lệnh outtextxy(x, y, s) là không thay đổi. Ví dụ sau in ra màn hình đồ họa dòng chữ "Đây là chương trình minh họa lệnh outtext(s)" tại vị trí (100, 20):

```
moveto(100, 20) ;                     // chuyen con tro den cot 100, dong 20
outtext("Đây là chương trình minh họa lệnh outtext(s)") ;  hoặc
outtext("Đây là chương trình ") ;
outtext("minh họa lệnh ") ;
outtext("outtext(s)") ;
```

hoặc dòng văn bản trên cũng có thể được in bởi lệnh outtextxy(x, y, s);

```
outtextxy(100, 20, "Đây là chương trình minh họa lệnh outtextxy(x, y, s)");
```

b. Điều chỉnh font, hướng và cỡ chữ

settextstyle(Font, Hướng, Cỡ chữ);

a. Font : Gồm các loại font tương ứng với các hằng sau đây:

DEFAULT_FONT	0
SMALL_FONT	1
TRIPLEX_FONT	2
SANS_SERIF_FONT	3
GOTHIC_FONT	4

- Hướng : hướng viết theo kiểu nằm ngang hay thẳng đứng, tương ứng với các hằng:

HOIRIZ_DIR	0
VERT_DIR	1

- Cỡ chữ : Gồm các cỡ chữ đánh số tăng dần từ 1. Cỡ chữ ngầm định là 1.

Ví dụ sau lần lượt in tại tâm màn hình tên của các font với các cỡ chữ lớn dần, theo hướng nằm ngang.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
void main()
{
    char *fname[] = {"ngầm định", "Triplex", "Small", "Sans Serif", "Gothic" };
    int gdriver = DETECT, gmode;
    int font, midx, midy;
    int size = 1;
    initgraph(&gdriver, &gmode, "C:\\Borlandc\\BGI");
    midx = getmaxx() / 2; midy = getmaxy() / 2;
    for (font = DEFAULT_FONT; font <= GOTHIC_FONT; font++)
    {
        cleardevice();
        size = font;
        settextstyle(font, HORIZ_DIR, size);
        outtextxy(midx, midy, fname[font]);
```

```
        getch();
    }
    closegraph();
}
```

c. Điều chỉnh cách viết

Theo mỗi hướng (nằm ngang hay thẳng đứng) có 3 cách viết tương ứng với các hằng số sau:

1. Theo hướng nằm ngang:

   LEFT_TEXT = 0 : Viết từ trái sang phải.

   CENTER_TEXT = 1 : Viết từ vị trí con trỏ sang hai bên.

   RIGHT_TEXE = 2 : Viết từ phải sang trái.

2. Theo hướng thẳng đứng:

   BOTTOM_TEXT = 0 : Viết từ dưới lên.

   CENTER_TEXT = 1 : Viết từ vị trí con trỏ lên trên và xuống dưới.

   TOP_TEXT = 2. Viết từ trên xuống.

Để chỉ định một trong các cách viết trên ta dùng lệnh

settextjustify(Theo hướng ngang, Theo hướng dọc);

5. Chuyển động

Nguyên tắc: xóa hình ở vị trí cũ rồi vẽ lại hình đó tại vị trí mới theo hướng chuyển động. Để xoá, ta vẽ lại hình ngay tại vị trí cũ nhưng với mầu vẽ trùng với màu nền (do đó hình vẽ bị chìm vào nền giống như đã bị xóa). Để biết màu nền hiện tại có thể dùng hàm setcolor(getbkcolor()). Tóm lại có thể đưa ra sơ đồ như sau:

– vẽ lại hình với màu nền tại vị trí cũ // xóa hình

– delay // tạm dừng

– vẽ lại hình (với màu của hình) tại vị trí mới // hình chuyển đến vị trí khác

Các bước trên nếu được lặp đi lặp lại ta sẽ thấy hình chuyển động từ vị trí này đến vị trí khác.

Đối với các hình vẽ phức tạp, để xóa nhanh ta có thể vẽ lại hình trong chế độ XOR_PUT như được trình bày trong phần sau.

Chúng ta hãy xem qua một số hàm phức tạp hơn để vẽ hình.

- **setviewport(x1, y1, x2, y2, clip)**: Tạo một cửa sổ mới trong chế độ đồ hoạ. Khi đó tọa độ của các điểm sẽ được tính lại theo cửa sổ mới này. Cụ thể điểm (x1, y1) của màn hình bây giờ sẽ lại được tính với tọa độ mới là (0,0). Nếu clip = 0 sẽ cho phép các hình vẽ được mở rộng khỏi khung cửa sổ, nếu clip = 1 các phần của hình vẽ nằm ngoài khung cửa sổ sẽ bị cắt.
- **getviewsettings(struct viewporttype *vp)**: Lấy toạ độ cửa sổ hiện tại vào biến con trỏ vp. Kiểu của cuả sổ là một cấu trúc như sau:

  struct viewporttype {int left, top, right, bottom, clip;};
- **imagesize(x1, y1, x2, y2)**: Cho lại kích thước (byte) của một ảnh bitmap trong khung chữ nhật được xác định bởi các tọa độ (x1, y1, x2, y2).
- **getimage(x1, y1, x2, y2, *pict)**: Lưu ảnh từ màn hình vào vùng bộ nhớ được trỏ bởi con trỏ pict.
- **putimage(x1, y1, *pict, op)**: Ghi ra màn hình ảnh đã được lưu tại vị trí con trỏ pict. op là chế độ qui định việc hiện ảnh lên màn hình, màu của các điểm sẽ được qui định thông qua màu của ảnh được lưu trong pict và màu hiện tại của điểm trên màn hình. Hai màu này sẽ "trộn" theo các phép toán qui định bởi op dưới đây để cho ra màu vẽ của ảnh:

COPY_PUT = 0	Săn cầu thủ
XOR_PUT = 1	Hoặc loại trừ (giống nhau thì bằng 0). Để xóa ảnh ta có thể vẽ lại chúng với chế độ này.
OR_PUT = 2	Hoặc
AND_PUT = 3	Và
NOT_PUT = 4	Not

Ví dụ 8 : Vẽ bánh xe xoay

```
void bx(int x, int y, int r, float phi, int xoa)        // xoá ảnh nếu xoa = 1
{
    int i, x1, x2, y1, y2;
    if (xoa) setcolor(BLACK);                           // đặt màu vẽ bằng màu nền
    circle(x, y, r);                                    // vẽ vành bánh xe
    for (i=0; i<6; i++) {
        x1 = x+int(r*cos(phi)); y1 = y-int(r*sin(phi));
        x2 = x-int(r*cos(phi)); y2 = y+int(r*sin(phi));
        line(x1, y1, x2, y2);                           // vẽ các nan hoa
```

```
            phi = phi + pi/3;                    // lệch nhau 60^0
        }
        setcolor(WHITE);
    }

    void xoay()
    {
        int i, x, y, r;
        static float phi = 0;
        x = midx; y = midy; r = 100;
        while (!kbhit()) {
            bx(x, y, r, phi, 0);                 // vẽ bánh xe
            delay(100);                          // tạm dừng
            bx(x, y, r, phi, 1);                 // xóa bánh xe
            phi = phi-pi/72;                     // xoay đi một góc phi
        }
    }
```

Ví dụ 8 : Vẽ bánh xe lăn trên đường nằm ngang

```
    void lan()
    {
        int i, x, y, r;
        float phi=0;
        x = 0; y = maxy-110; r = 60;
        setlinestyle(SOLID_LINE, 1, 3);
        line(0, maxy-50, maxx, maxy-50);
        setlinestyle(SOLID_LINE, 1, 1);
        while (x-r<=maxx) {
            bx(x, y, r, phi, 0);
            delay(20); bx(x, y, r, phi, 1); x += 1;    phi = phi-pi/72;
        }
```

```
}
```

6. Vẽ đồ thị của các hàm toán học

Để vẽ đồ thị của một hàm toán học, ta vẽ từng điểm một của đồ thị. Mỗi điểm được xác định bởi cặp tọa độ (x, y) trên màn hình. Do vậy cần tính các điểm này theo tọa độ trên màn hình. Các bước cần làm gồm có:

- Xác định hệ trục tọa độ. Thông thường ta sẽ lấy tâm màn hình làm tâm hệ trục bằng việc xác định lại cửa sổ màn hình bởi câu lệnh:

```
viewport(midx, midy, maxx, maxy, 0);
```

trong đó midx, midy là tọa độ tâm màn hình, maxx, maxy là tọa độ góc dưới bên phải của màn hình. Câu lệnh trên tạo một cửa sổ là phần tư bên phải, phía dưới của màn hình. Tham trị cuối (1) cho phép các hình vẽ sẽ được vẽ ra ngoài khung cửa sổ này. Như vậy tâm màn hình sẽ biến thành tâm của hệ trục tọa độ. Tọa độ của tâm màn hình bây giờ được tính là (0,0).

- Xác định tỉ lệ: Cần xác định một đơn vị của x và y của hàm cần vẽ sẽ tương ứng với bao nhiêu điểm trên trục x và y của màn hình. Do số điểm theo chiều rộng và chiều cao của màn hình khác nhau và do giá trị của hàm (y) có thể rất lớn so với giá trị của đối (x) (ví dụ hàm $y = x^4$) hoặc rất bé (ví dụ hàm y = sinx) nên các tỉ lệ này theo x và y có thể khác nhau để hình vẽ trên màn hình được cân đối. Việc xác định các tỉ lệ này phụ thuộc vào kinh nghiệm và thường được điều chỉnh sau khi chạy thử chương trình.
- Vẽ hệ trục : Có thể vẽ hệ trục tọa độ hay không. Hàm sau cho phép vẽ các trục tọa độ với tâm nằm giữa màn hình.

```
void vetruc()                                   // Ve truc toa do
{
        line(0, midy, maxx, midy);              // truc hoanh
        line(maxx-7, midy-3, maxx, midy);       // mui ten
        line(maxx-7, midy+3, maxx, midy);
        line(midx, 0, midx, maxy);              // truc tung
        line(midx-3, 7, midx, 0);               // mui ten
        line(midx+3, 7, midx, 0);
        outtextxy(midx+6, midy+6, "(0, 0)");    // in toa do (0,0)
}
```

Các ví dụ sau sẽ vẽ đồ thị của một số hàm quen thuộc.

```
void Sinx()                                  // Do thi ham Sinx
{
    int tileX = 20, tileY = 60;              // Tỉ lệ theo X và Y
    int x, y, i;
    setviewport(midx, midy, maxx, maxy, 0);
    for (i = -400; i<=400; i++) {
        x = 2*pi*i*tileX/200;
        y = sin(2*pi*i/200)*tileY;
        putpixel(x, y, 1);
    }
    setviewport(0, 0, maxx, maxy, 0);
}

void Sinoverx()                              // Ham Sinx/x
{
    float t;
    float tileX = 50/pi;
    float tileY = 80;
    int x, y;
    for (x = 30; x < maxx-30; x++) {
        t = ((x==midx)? 1 : (x-midx))/tileX;
        y = midy - int(sin(t)/t*tileY);
        putpixel(x, y, 2);
    }
}
```

Ve do thi theo tham so (x = x(t), y = y(t))

```
void Hypocycloide()                          // Ham x = cos^3t, y = sin^3t
{                                            // t ∈ [0, 2π]
```

```
        float t;
        int i, x, y;
        for (i = 0; i<1000; i++) {
                t = (pi/500)*i;
                x = int(120*pow(cos(t), 3)) + midx;
                y = int(120*pow(sin(t), 3)) + midy;
                putpixel(x, y, 3);
        }
}
void Trocoide()                          // Ham (2t-3sint, 2-3cost)
{                                        // t ∈ [-9, 9]
        float t;
        int i, x, y;
        for (i = -1000; i<=1000; i++) {
                t = 0.01*i;
                x = int(15*(2*t-3*sin(t))) + midx;
                y = -int(10*(2-3*cos(t))) + midy;
                putpixel(x, y, 4);
        }
}
void So3()                               // x = sintcos²t + sint
{                                        // y = sin²tcost, t ∈ [0, 2π]
        float t;
        int i, x, y;
        for (i = 0; i<=1000; i++) {
                t = (pi/500)*i;
                x = int(150*(sin(t)*(1+cos(t)*cos(t)))) + midx;
                y = int(200*sin(t)*sin(t)*cos(t)) + midy;
                putpixel(x, y, 5);
        }
}
```

Ve do thi theo toa do cuc r = φ(θ)

```
void Archimede()                          // Ham r = θ, θ ∈ [0, 40]
{
    int i, x, y;
    float r, t;
    for (i = 0;i<=2500;i++) {
        t = 0.02*i;
        x = int(3*t*cos(t))+midx;
        y = -int(3*t*sin(t))+midy;
        putpixel(x, y, 6);
    }
}

void Hoahong()                            // Ham r = sin2θ, θ ∈ [0, 2π]
{
    int i, x, y;
    float r, t;
    for (i = 0;i<=2000;i++) {
        t = (pi/500)*i;
        x = int(150*(sin(2*t)*cos(t)))+midx;
        y = int(150*sin(2*t)*sin(t))+midy;
        putpixel(x, y, 7);
    }
}
```

Chương trình dưới đây cho phép vẽ hai mặt trong không gian 3 chiều được cho bởi hai hàm f = sinx.siny và g = $\frac{sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}$

```
typedef struct TOADO {
    int OX, OY, UX, UY, UZ;          // truc hoanh, tung va don vi cac truc
    double Xx, Xy;  // goc (OX, ox), (OY, oy)
```

```
};
TOADO gr3 = { 320, 20, 20, 20, 20, 0.8*pi, 0.2*pi };

void Vetruc()                                   // Ve truc Ox, Oy
{
        setviewport(0, 0, maxx, maxy, 0);
        settextstyle(DEFAULT_FONT, HORIZ_HUONG, 0);
        setcolor(WHITE);
        line(0, midy, maxx, midy);
        line(maxx-7, midy-3, maxx, midy); line(maxx-7, midy+3, maxx, midy);
        line(midx, 0, midx, maxy);
        line(midx-3, 7, midx, 0); line(midx+3, 7, midx, 0);
        outtextxy(midx+6, midy+6, "(0, 0)");
        outtextxy(maxx-18, midy+6, "x"); outtextxy(midx+8, 6, "y");
        setbkcolor(CYAN); setcolor(RED);
        settextstyle(TRIPLEX_FONT, HORIZ_HUONG, 2);
        outtextxy(10, 0, "DO THI KHONG GIAN 3 CHIEU");
}

int X(double x, double y, double z)     // doi toa do xyz sang truc X
{
        return gr3.OX + x*gr3.UX*cos(gr3.Xx) + y*gr3.UY*cos(gr3.Xy);
}
int Y(double x, double y, double z)     // doi toa do xyz sang truc Y
{
        return gr3.OY + x*gr3.UX*sin(gr3.Xx) + y*gr3.UY*sin(gr3.Xy) – z*gr3.UZ;
}
double f(double x, double y)            // Ham f(x, y) can ve
{
        return 4*sin(x)*sin(y);
}
```

```
double g(double x, double y)            // Ham g(x, y) can ve
{
        return 5*sin(sqrt(x*x+y*y))/sqrt(x*x+y*y);
}
void Vehamf()
{
        double x, y, z;
        double xa = -6.28, xb = 6.28;
        double ya = -6.28, yb = 6.28;
        double xp = 0.2, yp = 0.2;
        int mat[8];
        settextstyle(TRIPLEX_FONT, HORIZ_HUONG, 1);
        outtextxy(10, 20, "Ham z = sinx.siny");
        setviewport(0, midy, maxx, maxy, 0);
        for (x = xa; x <= xb; x+=xp)                    // ve mat an
        for (y = ya; y <= yb; y+=yp)
        {
                if (kbhit()) return;
                z = f(x, y);                            // diem A
                mat[0] = X(x, y, z); mat[1] = Y(x, y, z);
                z = f(x, y+yp);                         // diem B
                mat[2] = X(x, y+yp, z); mat[3] = Y(x, y+yp, z);
                z = f(x+xp, y+yp);                      // diem C
                mat[4] = X(x+xp, y+yp, z); mat[5] = Y(x+xp, y+yp, z);
                z = f(x+xp, y);                         // diem D
                mat[6] = X(x+xp, y, z); mat[7] = Y(x+xp, y, z);
                if ((mat[3]-mat[1]) * (mat[6]-mat[0]) –
                        (mat[7]-mat[1]) * (mat[2]-mat[0]) < 0)
                setfillstyle(1, YELLOW); else setfillstyle(1, GREEN);
                fillpoly(4, mat);
                delay(10);
```

```
        }
        getch();
}

void Vehamg()
{
        double x, y, z;
        double xa = -10, xb = 10;
        double ya = -10, yb = 10;
        double xp = 0.1, yp = 0.1;
        settextstyle(TRIPLEX_FONT, HORIZ_DIR, 1);
        outtextxy(10, 20, "Ham z = sin(sqrt(x*x+y*y))");
        outtextxy(100, 30, "------------");
        outtextxy(115, 40, "sqrt(x*x+y*y)");
        setviewport(0, midy, maxx, maxy, 0);
        setcolor(BLUE);
        for (x = xa; x <= xb; x+=xp)
        for (y = ya; y <= yb; y+=yp)
        {
                if (kbhit()) return;
                z = g(x, y); lineto(X(x, y, z), Y(x, y, z));
                delay(10);
        }
        getch();
}

void main()
{
        Initgraph(); Vetruc(); Vehamf();
        cleardevice(); Vetruc(); Vehamg();
        closegraph();
}
```

II. ÂM THANH

Âm thanh được đặc trưng bởi cao độ (tần số) và trường độ (độ dài). Việc tạo ra một chuỗi âm (bài hát chẳng hạn), là sự kết hợp lặp đi lặp lại của các hàm sau với các tham số n và t được chọn thích hợp.

– sound(n): phát âm ra loa máy tính với tần số n.

– delay(t): kéo dài trong t miligiây.

– nosound(): tắt âm thanh đã phát.

Ví dụ 1 : Tiếng còi báo động

```
void coi(int cao; int thap)
{
    do {
        sound(cao); delay(400); sound(thap); delay(400);
    } while (!kbhit())
    nosound();
}
```

Ví dụ 2 : Tiếng bóng nảy

```
void bong(int cao; int thap)
{
    int sodem = 20;
    while (sodem > 1) {
        sound(thap-2*sodem); delay(sodem*500/20);
        nosound(); delay(100);
        sound(cao); delay(sodem*500/15); nosound(); delay(150);
        sodem --;
    }
}
```

Ví dụ 3 : Tiếng bom

```
void bong(int cao; int thap; int t)
{
    int sodem = thap;
    while (sodem <= cao) {
        sound(sodem); delay(t/sodem*75);
        sodem += 10;
    }
    for (sodem =1 to 3) {
        nosound();
        sound(40); delay(500); nosound(); delay(100);
    }
    sound(40); delay(3000); nosound();
}
```

Để tạo âm phát của một nốt nhạc có tần số (cao độ) n và dài trong t miligiây cần viết hàm :

```
void not(unsigned n, float t);
{
    sound(n);
    delay(t);
    nosound();
}
```

Ví dụ 4 : Chơi bài hát Tiến quân ca trên nền cờ đỏ sao vàng.

```
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <dos.h>
// cao độ của các nốt nhạc

#define  do1    66    #define  dod1   70    #define  re1    73
#define  red1   78    #define  mi1    82    #define  fa1    86
```

```
#define  fad1   91    #define  sol1   96    #define  sold1  102
#define  la1    108   #define  lad1   115   #define  si1    122
#define  do2    130   #define  dod2   139   #define  re2    148
#define  re2    148   #define  red2   156   #define  mi2    164
#define  fa2    176   #define  fad2   188   #define  sol2   196
#define  sold2  209   #define  la2    230   #define  lad2   233
#define  si2    247   #define  do3    264   #define  dod3   281
#define  re3    297   #define  red3   313   #define  mi3    330
#define  fa3    352   #define  fad3   374   #define  sol3   396
#define  sold3  415   #define  la3    440   #define  lad3   468
#define  si3    495   #define  do4    528   #define  dod4   565
#define  re4    594   #define  red4   625   #define  mi4    660
#define  fa4    704   #define  fad4   748   #define  sol4   792
#define  sold4  836   #define  la4    880   #define  lad4   935
#define  si4    990   #define  lang   30000

void not(unsigned caodo, float truongdo)
{ sound(caodo); delay(truongdo); nosound(); }

void main()
{
      int gdriver = DETECT, gmode;
      initgraph(&gdriver, &gmode, "c:\\borlandc\\bgi");
      int star[20] = {320, 150, 285, 225, 200, 225, 270, 270, 240, 350,
      320, 300, 390, 350, 360, 270, 430, 225, 345, 225};
      setbkcolor(RED); setcolor(YELLOW);       // Vẽ lá cờ đỏ sao vàng
      setfillstyle(SOLID_FILL, YELLOW);
      fillpoly(10, star);

      // Trường độ các nốt nhạc
      float d = 300;                  // đen
      float tr = 2*d;                 // trắng
```

```
        float tro = 4*d;            // tròn
        float md = d/2;             // móc đen
        float mk = d/4;             // móc kép
        float m3 = d/8;             // móc 3
        float m4 = d/16;            // móc 4
        float dc = 3*d/2;           // đen chấm
        float trc = 3*d;            // trắng chấm
        float troc = 6*d;           // tròn chấm

        // Choi bai TQC
        not(re2, d); not(mi2, d); not(re2, d); not(sol2, tr); not(sol2, tro);
        not(la2, d); not(sol2, d); not(si2, tr); not(si2, tro); not(la2, d);
        not(sol2, d); not(mi2, tr); not(sol2, tr); not(sol2, d); not(mi2, tro);
        not(re2, d); not(si2, d); not(re2, tro); not(sol2, d); not(la2, d);
        not(si2, tr); not(si2, tr); not(si2, tr); not(la2, d); not(sol2, d);
        not(re3, tro); not(si2, d); not(sol2, d); not(la2, tr); not(la2, tr);
        not(si2, tr); not(fad2, d); not(re2, d); not(sol2, tro); not(si2, d);
        not(do3, d); not(re3, tr); not(re3, tr); not(mi3, tro); not(re3, d);
        not(si2, tro); not(si2, tr); not(la2, d); not(sol2, tr); not(re2, tr);
        not(fad2, tr); not(fad2, d); not(la2, d); not(sol2, tr); not(si2, d);
        not(do3, d); not(re3, tr); not(re3, tr); not(mi3, tro); not(re3, d);
        not(si2, tro); not(si2, tr); not(la2, d); not(sol2, tr); not(sol2, tr);
        not(re2, tro); not(re3, tro); not(si2, tr); not(sol2, tr); not(mi3, tro);
        not(re3, tr); not(si2, d); not(la2, d); not(re2, d); not(la2, tr);
        not(la2, tro); not(si2, tr); not(sol2, tro);
        closegraph();
}
```

BÀI TẬP

1. Vẽ hai hình chữ nhật, lần lượt cho mất từng hình, rồi hiện lại cả hai.

2. Biểu diễn dãy 5 giá trị (được nhập từ bàn phím) bằng biểu đồ bar.

3. Biểu diễn dãy 5 giá trị (được nhập từ bàn phím) bằng biểu đồ hình quạt.

4. Vẽ một bàn cờ quốc tế với các ô đen trắng.

5. Viết chương trình vẽ đồ thị hàm số y = 100*sin(x/4.8) trong khoảng x $\in$ [0, 60] với giá trị mỗi bước $\Delta x = 0,1$. Yêu cầu :

- nền màn hình mầu đen.
- trục tọa độ màu xanh lá cây
- đồ thị màu trắng.

6. Viết chương trình vẽ tam giác với các tọa độ đỉnh lần lượt là A(300, 20), B(100, 220), C(500, 220) và đường tròn ngoại tiếp của nó. Yêu cầu :

- nền màn hình mầu đen.
- các cạnh tam giác màu xanh lá cây
- đường tròn ngoại tiếp màu đỏ tươi.

7. Viết chương trình vẽ hình chữ nhật có tọa độ đỉnh góc trên bên trái là (100,150), chiều ngang 120, chiều dọc 90 và đường tròn ngoại tiếp nó. Yêu cầu :

- nền màn hình mầu đen.
- các cạnh hình chữ nhật màu xanh da trời.
- đường tròn ngoại tiếp màu đỏ tươi.

8. Vẽ tam giác nội tiếp trong hình tròn, hình tròn nội tiếp trong elip. Tô các màu khác nhau cho các miền tạo bởi các đường trên.

9. Vẽ một đài phát sóng. Các vòng sóng phát từ đỉnh của tháp ở tâm màn hình lan tỏa ra chung quanh. Quá trình lặp đến khi ấn phím bất kỳ thì dừng.

10. Vẽ hai hình người đi vào từ 2 phía màn hình với tốc độ khác nhau. Gặp nhau hai hình người xoay lại và đi ngược về 2 phía màn hình. Chương trình dừng khi cả hai đã đi khuất vào hai phía của màn hình.

CHƯƠNG 7

LỚP VÀ ĐỐI TƯỢNG

Lập trình có cấu trúc và lập trình hướng đối tượng
Lớp và đối tượng
Đối của phương thức - Con trỏ this
Hàm tạo (contructor)
Hàm hủy (destructor)
Các hàm trực tuyến (inline)

I. LẬP TRÌNH CÓ CẤU TRÚC VÀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

1. Phương pháp lập trình cấu trúc

- Lập trình cấu trúc là tổ chức chương trình thành các chương trình con. Trong một số ngôn ngữ như PASCAL có 2 kiểu chương trình con là thủ tục và hàm, còn trong C++ chỉ có một loại chương trình con là hàm.
- Hàm là một đơn vị chương trình độc lập dùng để thực hiện một phần việc nào đó như: Nhập số liệu, in kết quả hay thực hiện một số công việc tính toán. Hàm cần có đối và các biến, mảng cục bộ dùng riêng cho hàm.
- Việc trao đổi dữ liệu giữa các hàm thực hiện thông qua các đối và các biến toàn cục.
- Một chương trình cấu trúc gồm các cấu trúc dữ liệu (như biến, mảng, bản ghi) và các hàm, thủ tục.
- Nhiệm vụ chính của việc tổ chức thiết kế chương trình cấu trúc là tổ chức chương trình thành các hàm, thủ tục.

Ví dụ, ta xét yêu cầu sau: Viết chương trình nhập toạ độ (x,y) của một dãy điểm, sau đó tìm một cặp điểm cách xa nhau nhất.

Trên tư tưởng của lập trình cấu trúc có thể tổ chức chương trình như sau:

- Sử dụng 2 mảng thực toàn bộ x và y để chứa toạ độ dãy điểm.
- Xây dựng 2 hàm:

Hàm nhapsl dùng để nhập toạ độ n điểm, hàm này có một đối là biến nguyên n và được khai báo như sau:

```
void nhapsl(int n);
```

Hàm do_dai dùng để tính độ dài đoạn thẳng đi qua 2 điểm có chỉ số là i và j,

nó được khai báo như sau:

```
float do_dai(int i, int j);
```

Chương trình C của ví dụ trên được viết như sau:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
float x[100],y[100];
float do_dai(int i, int j)
{
    return sqrt(pow(x[i]-x[j],2)+pow(y[i]-y[j],2));
}
void nhapsl(int n)
{
    int i;
    for (i=1; i<=n; ++i)
    {
        printf("\n Nhap toa do x, y cua diem thu %d : ",i);
    scanf("%f%f",&x[i],&y[i]);
    }
}
void main()
{
    int n, i, j, imax,jmax;
    float d, dmax;
    printf("\n So diem N= ");
    scanf("%d", &n);
    nhapsl(n);
    dmax=do_dai(1,2);imax=1; jmax=2;
    for(i=1; i<=n-1; ++i)
    for (j=i+1; j<=n; ++j)
    {
        d=do_dai(i,j);
        if (d>dmax)
```

```
            {
                dmax=d;
                imax=i; jmax=j;
            }
        }
        printf("\nDoan thang lon nhat co do dai bang: %0.2f",dmax);
        printf("\n Di qua 2 diem co chi so la %d va %d",imax,jmax);
        getch();
}
```

2. Phương pháp lập trình hướng đối tượng

Là lập trình có cấu trúc + trừu tượng hóa dữ liệu. Có nghĩa chương trình tổ chức dưới dạng cấu trúc. Tuy nhiên việc thiết kế chương trình sẽ xoay quanh dữ liệu, lấy dữ liệu làm trung tâm. Nghĩa là trả lời câu hỏi: Chương trình làm việc với những đối tượng dữ liệu nào, trên các đối tượng dữ liệu này cần thao tác, thực hiện những gì. Từ đó gắn với mỗi đối tượng dữ liệu một số thao tác thực hiên cố định riêng của đối tượng dữ liệu đó, điều này sẽ qui định chặt chẽ hơn những thao tác nào được thực hiện trên đối tượng dữ liệu nào. Khác với lập trình cấu trúc thuần túy, trong đó dữ liệu được khai báo riêng rẽ, tách rời với thao tác xử lý, do đó việc xử lý dữ liệu thường không thống nhất khi chương trình được xây dựng từ nhiều lập trình viên khác nhau.

Từ đó lập trình hướng đối tượng được xây dựng dựa trên đặc trưng chính là khái niệm đóng gói. Đóng gói là khái niệm trung tâm của phương pháp lập trình hướng đối tượng, trong đó dữ liệu và các thao tác xử lý nó sẽ được qui định trước và "đóng" thành một "gói" thống nhất, riêng biệt với các dữ liệu khác tạo thành kiểu dữ liệu với tên gọi là các lớp. Như vậy một lớp không chỉ chứa dữ liệu bình thường như các kiểu dữ liệu khác mà còn chứa các thao tác để xử lý dữ liệu này. Các thao tác được khai báo trong gói dữ liệu nào chỉ xử lý dữ liệu trong gói đó và ngược lại dữ liệu trong một gói chỉ bị tác động, xử lý bởi thao tác đã khai báo trong gói đó. Điều này tạo tính tập trung cao khi lập trình, mọi đối tượng trong một lớp sẽ chứa cùng loại dữ liệu được chỉ định và cùng được xử lý bởi các thao tác như nhau. Mọi lập trình viên khi làm việc với dữ liệu trong một gói đều sử dụng các thao tác như nhau để xử lý dữ liệu trong gói đó. C++ cung cấp cách thức để tạo một cấu trúc dữ liệu mới thể hiện các gói nói trên, cấu trúc dữ liệu này được gọi là **lớp**.

Để minh hoạ các khái niệm vừa nêu về kiêu dữ liệu lớp ta trở lại xét bài toán tìm độ dài lớn nhất đi qua 2 điểm. Trong bài toán này ta gặp một thực thể là dãy điểm. Các thành phần dữ liệu của lớp dãy điểm gồm:

- Biến nguyên n là số điểm của dãy
- Con trỏ x kiểu thực trỏ đến vùng nhớ chứa dãy hoành độ

- Con trỏ y kiểu thực trỏ đến vùng nhớ chứa dãy tung độ

Các phương thức cần đưa vào theo yêu cầu bài toán gồm:

- Nhập toạ độ một điểm
- Tính độ dài đoạn thẳng đi qua 2 điểm

Dưới đây là chương trình viết theo thiết kế hướng đối tượng.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <alloc.h>
class daydiem
{
        int n;
        float *x,*y;
public:
        float do_dai(int i, int j)
        {
                return sqrt(pow(x[i]-x[j],2)+pow(y[i]-y[j],2));
        }
        void nhapsl(void);
};
void daydiem::nhapsl(void)
{
        int i;
        printf("\n So diem N= ");
        scanf("%d",&n);
        x = (float*)malloc((n+1)*sizeof(float));
        y = (float*)malloc((n+1)*sizeof(float));
        for (i=1; i<=n; ++i)
        {
                printf("\n Nhap toa do x, y cua diem thu %d : ",i);
                scanf("%f%f",&x[i],&y[i]);
        }
}
```

```
void main()
{
	clrscr();
	daydiem p;
	p.nhapsl();
	int n,i,j,imax,jmax;
	float d,dmax;
	n = p.n;
	dmax=p.do_dai(1,2);imax=1; jmax=2;
	for (i=1;i<=n-1;++i)
	for (j=i+1;j<=n;++j)
	{
		d=p.do_dai(i,j);
		if (d>dmax)
		{
			dmax=d;
			imax=i; jmax=j;
		}
	}
	printf("\n Doan thang lon nhat co do dai bang: %0.2f",dmax);
	printf("\n Di qua 2 diem co chi so la %d va %d" , imax,jmax);
	getch();
}
```

II. LỚP VÀ ĐỐI TƯỢNG

Trong lập trình hướng đối tượng, lớp (class) là một khái niệm rất quan trọng, nó cho phép giải quyết các vấn đề phức tạp của việc lập trình. Một lớp đơn (được định nghĩa như struct, union, hoặc class) bao gồm các hàm và dữ liệu có liên quan. Các hàm này là các hàm thành phần (member functon) hay còn là phương thức (method), thể hiện tác động của lớp có thể được thực hiện trên dữ liệu của chính lớp đó (data member).

Cũng giống như cấu trúc, lớp có thể xem như một kiểu dữ liệu. Vì vậy lớp còn gọi là kiểu đối tượng và lớp được dùng để khai báo các biến, mảng đối tượng (như thể dùng kiểu int để khai báo các biến mảng nguyên).

Như vậy từ một lớp có thể tạo ra (bằng cách khai báo) nhiều đối tượng (biến,

mảng) khác nhau. Mỗi đối tượng có vùng nhớ riêng của mình và vì vậy ta cũng có thể quan niệm lớp chính là tập hợp các đối tượng cùng kiểu.

1. Khai báo lớp

Để khai báo một lớp, ta sử dụng từ khoá class như sau:

```
class tên_lớp
{
    // Khai báo các thành phần dữ liệu (thuộc tính)
    // Khai báo các phương thức (hàm)
};
```

Chú ý: Việc khai báo một lớp không chiếm giữ bộ nhớ, chỉcác đối tượng của lớp mới thực sự chiếm giữ bộ nhớ.

Thuộc tính của lớp có thể là các biến, mảng, con trỏ có kiểu chuẩn (int, float, char, char*, long,...) hoặc kiểu ngoài chuẩn đã định nghĩa trước (cấu trúc, hợp, lớp,...). Thuộc tính của lớp không thể có kiểu của chính lớp đó, nhưng có thể là con trỏ của lớp này, ví dụ:

```
class A
{
    A x;          //Không cho phép, vì x có kiểu lớp A
    A* p ;        //Cho phép , vì p là con trỏ kiểu lớp A
} ;
```

2. Khai báo các thành phần của lớp (thuộc tính và phương thức)

a. Các từ khóa private và public

Khi khai báo các thành phần dữ liệu và phương thức có thể dùng các từ khoá private và public để quy định phạm vi sử dụng của các thành phần này. Trong đó từ khóa private qui định các thành phần (được khai báo với từ khóa này) chỉ được sử dụng bên trong lớp (trong thân các phương thức của lớp). Các hàm bên ngoài lớp (không phải là phương thức của lớp) không được phép sử dụng các thành phần này. Đặc trưng này thể hiện tính che giấu thông tin trong nội bộ của lớp, để đến được các thông tin này cần phải thông qua chính các thành phần hàm của lớp đó. Do vậy thông tin có tính toàn vẹn cao và việc xử lý thông tin (dữ liệu) này mang tính thống nhất hơn và hầu như dữ liệu trong các lớp đều được khai báo với từ khóa này. Ngược lại với private, các thành phần được khai báo với từ khóa public được phép sử dụng ở cả bên trong và bên ngoài lớp, điều này cho phép trong chương trình có thể sử dụng các hàm này để truy nhập đến dữ liệu của lớp. Hiển nhiên nếu các thành phần dữ liệu đã khai báo là privte thì các thành phần hàm phải có ít nhất một vài hàm được khai báo dạng public để chương trình có thể truy cập được, nếu không

toàn bộ lớp sẽ bị đóng kín và điều này không giúp gì cho chương trình. Do vậy cách khai báo lớp tương đối phổ biến là các thành phần dữ liệu được ở dạng private và thành phần hàm dưới dạng public. Nếu không quy định cụ thể (không dùng các từ khoá private và public) thì C++ hiểu đó là private.

b. Các thành phần dữ liệu (thuộc tính)

Được khai báo như khai báo các thành phần trong kiểu cấu trúc hay hợp. Bình thường các thành phần này được khai báo là private để bảo đảm tính giấu kín, bảo vệ an toàn dữ liệu của lớp không cho phép các hàm bên ngoài xâm nhập vào các dữ liệu này.

c. Các phương thức (hàm thành viên)

Thường khai báo là public để chúng có thể được gọi tới (sử dụng) từ các hàm khác trong chương trình.

Các phương thức có thể được khai báo và định nghĩa bên trong lớp hoặc chỉ khai báo bên trong còn định nghĩa cụ thể của phương thức có thể được viết bên ngoài. Thông thường, các phương thức ngắn được viết (định nghĩa) bên trong lớp, còn các phương thức dài thì viết bên ngoài lớp.

Một phương thức bất kỳ của một lớp, có thể sử dụng bất kỳ thành phần (thuộc tính và phương thức) nào của lớp đó và bất kỳ hàm nào khác trong chương trình (vì phạm vi sử dụng của hàm là toàn chương trình).

Giá trị trả về của phương thức có thể có kiểu bất kỳ (chuẩn và ngoài chuẩn)

Ví dụ sau sẽ minh hoạ các điều nói trên. Chúng ta sẽ định nghĩa lớp để mô tả và xử lý các điểm trên màn hình đồ hoạ. Lớp được đặt tên là DIEM.

- Các thuộc tính của lớp gồm:

```
int x ;          // Hoành độ (cột)
int y ;          // Tung độ (hàng)
int m ;          // Mầu
```

- Các phương thức:

  Nhập dữ liệu một điểm

  Hiển thị một điểm

  Ẩn một điểm

Lớp điểm được xây dựng như sau:

```
#include <iostream.h>
#include <graphics.h>
class DIEM
{
```

```
private:
        int x, y, m ;
public:
        void nhapsl() ;
        void hien() ;
        void an() { putpixel(x, y, getbkcolor());}
};

void DIEM::nhapsl()
{
        cout <<"\n Nhap hoanh do (cot) va tung do (hang) cua diem: ";
        cin >> x >> y ;
        cout << "\n Nhap ma mau cua diem: ";
        cin >> m ;
}

void DIEM::hien()
{
        int mau_ht ;
        mau_ht = getcolor();
        putpixel(x, y, m);
        setcolor(mau_ht);
}
```

Qua ví dụ trên có thể rút ra một số chú ý sau:

+ Trong cả 3 phương thức (dù viết trong hay viết ngoài định nghĩa lớp) đều được phép truy nhập đến các thuộc tính x, y và m của lớp.
+ Các phương thức viết bên trong định nghĩa lớp (như phương thức an()) được viết như một hàm thông thường.
+ Khi xây dựng các phương thức bên ngoài lớp, cần dùng thêm tên lớp và toán tử phạm vi :: đặt ngay trước tên phương phức để quy định rõ đây là phương thức của lớp nào.

3. Biến, mảng và con trỏ đối tượng

Như đã nói ở trên, một lớp (sau khi định nghĩa) có thể xem như một kiểu đối tượng và có thể dùng để khai báo các biến, mảng đối tượng. Cách khai báo biến, mảng đối tượng cũng giống như khai báo biến, mảng các kiểu khác (như int, float,

cấu trúc, hợp,...), theo mẫu sau:

Tên_lớp danh sách đối ;

Tên_lớp danh sách mảng ;

Ví dụ sử dụng DIEM ở trên, ta có thể khai báo các biến, mảng DIEM như sau:

```
DIEM d1, d2, d3 ;      // Khai báo 3 biến đối tượng d1, d2, d3
DIEM d[20] ;           // Khai báo mảng đối tượng d gồm 20 phần tử
```

Mỗi đối tượng sau khi khai báo sẽ được cấp phát một vùng nhớ riêng để chứa các thuộc tính của nó. Chú ý rằng sẽ không có vùng nhớ riêng để chứa các phương thức cho mỗi đối tượng, các phương thức sẽ được sử dụng chung cho tất cả các đối tượng cùng lớp. Như vậy về bộ nhớ được cấp phát thì đối tượng giống cấu trúc.

Trong trường hợp này:

```
sizeof(d1) = sizeof(d2) = sizeof(d3) = 3*sizeof(int) = 6
sizeof(d) = 20*6 = 120
```

a. Thuộc tính của đối tượng

Trong ví dụ trên, mỗi đối tượng d1, d2, d3 và mỗi phần tử d[i] đều có 3 thuộc tính là x, y, m. Chú ý là mỗi thuộc tính đều thuộc về một đối tượng, vì vậy không thể viết tên thuộc tính một cách riêng rẽ mà bao giờ cũng phải có tên đối tượng đi kèm, giống như cách viết trong cấu trúc của C. Nói cách khác, cách viết thuộc tính của đối tượng như sau:

tên_đối_tượng.Tên_thuộc_tính

Với các đối tượng d1, d2, d3 và mảng d, có thể viết như sau:

```
d1.x;          // Thuộc tính x của đối tượng d1
d2.x;          // Thuộc tính x của đối tượng d2
d3.y;          // Thuộc tính y của đối tượng d3
d[2].m;        // Thuộc tính m của phần tử d[2]
d1.x = 100;    // Gán 100 cho d1.x
d2.y =d1.x;    // Gán d1.x cho d2.y
```

b. Sử dụng các phương thức

Cũng giống như hàm, một phương thức được sử dụng thông qua lời gọi. Tuy nhiên trong lời gọi phương thức bao giờ cũng phải có tên đối tượng để chỉ rõ phương thức thực hiện trên các thuộc tính của đối tượng nào.

Ví dụ lời gọi sau sẽ thực hiện nhập số liệu vào các thành phần d1.x, d1.y và d1.m: **d1.nhapsl();** Câu lệnh sau sẽ thực hiện nhập số liệu vào các thành phần d[3].x, d[3].y và d[3].m: **d[3].nhapsl() ;**

Chúng ta sẽ minh họa các điều nói trên bằng một chương trình đơn giản sử

dụng lớp DIEM để nhập 3 điểm, hiện rồi ẩn các điểm vừa nhập. Trong chương trình đưa vào hàm kd_do_hoa() dùng để khởi động hệ đồ hoạ.

```
#include <conio.h>
#include <iostream.h>
#include <graphics.h>

class DIEM
{
private:
        int x, y, m ;
public:
        void nhapsl();
        void an() { putpixel(x,y,getbkcolor());}
        void hien();
};

void DIEM::nhapsl()
{
        cout << "\n Nhap hoanh do (cot) va tung do (hang) cua DIEM: " ;
        cin>> x >> y ;
        cout << " \n Nhap ma tran cua diem: " ;
        cin >> m ;
}

void DIEM::hien()
{
        int mau_ht;
        mau_ht = getcolor() ;
        putpixel(x,y,m);
        setcolor(mau_ht);
}

void kd_do_hoa()
{
        int mh, mode ;
```

```
    mh=mode=0;
    initgraph(&mh, &mode, "C:\\TC\BGI");
}

void main()
{
    DIEMd1, d2, d3 ;
    d1.nhapsl(); d2.nhapsl(); d3.nhapsl();
    kd_do_hoa();
    setbkcolor(BLACK);
    d1.hien(); d2.hien(); d3.hien();
    getch();
    d1.an(); d2.an(); d3.an();
    getch();
    closegraph();
}
```

c. Con trỏ đối tượng

Con trỏ đối tượng dùng để chứa địa chỉ của biến, mảng đối tượng. Nó được khai báo như sau:

```
Tên_lớp *con trỏ;
```

Ví dụ dùng lớp DIEM có thể khai báo:

```
DIEM *p1, *p2, *p3 ;        // Khai báo 3 con trỏ p1, p2, p3
DIEM d1, d2 ;               // Khai báo 2 đối tượng d1, d2
DIEM d[20] ;                // Khai báo mảng đối tượng
```

và có thể thực hiện các câu lệnh:

```
p1= &d2 ;                   // p1 chứa địa chỉ của d2 , hay p1 trỏ tới d2
p2 = d ;                    // p2 trỏ tới đầu mảng d
p3 = new DIEM               // Tạo một đt và chứa địa chỉ của nó vào p3
```

Để sử dụng thuộc tính của đối tượng thông qua con trỏ, ta viết như sau:

Tên_con_trỏ → Tên_thuộc_tính

Chú ý: Nếu con trỏ chứa địa chỉ đầu của mảng, có thể dùng con trỏ như tên mảng.

Như vậy sau khi thực hiện các câu lệnh trên thì:

p1 → x và d2.x là như nhau

p2[i].y và d[i].y là như nhau

Từ đó ta có quy tắc sử dụng thuộc tính: Để sử dụng một thuộc tính của đối tượng ta phải dùng phép . hoặc phép →. Trong chương trình, không cho phép viết tên thuộc tính một cách đơn độc mà phải đi kèm tên đối tượng hoặc tên con trỏ theo các mẫu sau:

Tên_đối_tượng.Tên_thuộc_tính

Tên_con_trỏ → Tên_thuộc_tính

Tên_mảng_đối_ tượng[chỉ_số].Tên_thuộc_tính

Tên_con_trỏ[chỉ_số].Tên_thuộc_tính

Chương trình dưới đây cũng sử dụng lớp DIEM để nhập một dãy điểm, hiển thị và ẩn các điểm vừa nhập. Chương trình dùng một con trỏ kiểu DIEM và dùng toán tử new để tạo ta một dãy đối tượng

```
#include <conio.h>
#include <iostream.h>
#include <graphics.h>
class DIEM
{
private:
      int x, y, m ;
public:
      void nhapsl();
      void an() { putpixel(x,y,getbkcolor());}
      void hien();
};

void DIEM::nhapsl()
{
      cout <<"\n Nhap hoanh do (cot) va tung do (hang) cua diem:" ;
      cin >> x >> y ;
      cout << " \n Nhap ma mau cua DIEM: " ;
      cin >> m ;
}

void DIEM::hien()
{
```

```
        int mau_ht;
        mau_ht = getcolor() ;
        putpixel(x,y,m);
        setcolor(mau_ht);
}

void kd_do_hoa()
{
        int mh, mode ;
        mh=mode=0;
        initgraph(&mh, &mode, "C:\\TC\BGI");
}

void main()
{
        DIEM *p;
        int i, n;
        cout << "So diem: " ;
        cin >> n;
        p = new DIEM[n+1];
        for (i=1;i<=n;++i)
        p[i].nhapsl();
        kd_do_hoa();
        for (i=1;i<=n;++i) p[i].hien();
        getch();
        for (i=1;i<=n;++i) p[i].an();
        getch();
        closegraph();
}
```

III. ĐỐI CỦA PHƯƠNG THỨC, CON TRỎ THIS

1. Con trỏ this là đối thứ nhất của phương thức

Chúng ta hãy xem lại phương thức nhapsl của lớp DIEM

```
void DIEM::nhapsl()
```

```
{
    cout <<"\n Nhap hoanh do (cot) va tung do (hang) cua diem:" ;
    cin >> x >> y ;
    cout <<" \n Nhap ma mau cua diem: " ;
    cin >> m ;
}
```

Trong phương thức này chúng ta sử dụng tên các thuộc tính x, y và m một cách đơn độc. Điều này có vẻ như mâu thuẫn với quy tắc sử dụng thuộc tính nêu trong mục trước. Thực tế C++ đã ngầm định sử dụng một con trỏ đặc biệt với tên gọi this trong các phương thức trên. Các thuộc tính viết trong phương thức được hiểu là thuộc một đối tượng do con trỏ this trỏ tới. Do đó, nếu tường minh hơn, phương thức nhapsl() có thể được viết dưới dạng tương đương như sau:

```
void DIEM::nhapsl()
{
    cout << "\n Nhap hoanh do (cot) va tung do (hang) cua diem:" ;
    cin >> this → x >> this → y ;
    cout << "\n Nhap ma mau cua diem: " ;
    cin >>this → m;
}
```

Như vậy có thể kết luận rằng: Phương thức bao giờ cũng có ít nhất một đối là con trỏ this và nó luôn luôn là đối đầu tiên của phương thức.

2. Tham số ứng với đối con trỏ this

Xét một lời gọi tới phương thức nhapsl() :

```
DIEM d1;
d1.nhapsl() ;
```

Trong trường hợp này tham số truyền cho con trỏ this chính là địa chỉ của d1:

```
this = &d1
```

Do đó:

this → x chính là d1.x

this → y chính là d1.y

this → m chính là d1.m

Như vậy câu lệnh:**d1.nhapsl() ;**sẽ nhập dữ liệu cho các thuộc tính của đối tượng d1. Từ đó có thể rút ra kết luận sau:

Tham số truyền cho đối con trỏ this chính là địa chỉ của đối tượng đi kèm với

phương thức trong lời gọi phương thức.

3. Các đối khác của phương thức

Ngoài đối đặc biệt this (đối này không xuất hiện một cách tường minh), phương thức còn có các đối khác được khai báo thư trong các hàm. Đối của phương thức có thể cókiểu bất kỳ (chuẩn và ngoài chuẩn).

Ví dụ để xây dựng phương thức vẽ đường thẳng qua 2 điểm ta cần đưa vào 3 đối: Hai đối là 2 biến kiểu DIEM, đối thứ ba kiểu nguyên xác định mã mầu. Vì đã có đối ngầm định this là đối thứ nhất, nên chỉ cần khai báo thêm 2 đối. Phương thức có thể viết như sau:

```
void DIEM::doan_thang(DIEM d2, int mau)
{
    int mau_ht;
    mau_ht = getcolor();
    setcolor(mau);
    line(this → x, this → y,d2.x,d2.y);
    setcolor(mau_ht);
}
```

Chương trình sau minh hoạ các phương thức có nhiều đối. Ta vẫn dùng lớp DIEM nhưng có một số thay đổi:

- Bỏ thuộc tính m (mầu)
- Bỏ các phương thức ***hien*** và ***an***
- Đưa vào 4 phương thức mới:

  ve_doan_thang (Vẽ đoạn thẳng qua 2 điểm)

  ve_tam_giac (Vẽ tam giác qua 3 điểm)

  do_dai (Tính độ dài của đoạn thẳng qua 2 điểm)

  chu_vi (Tính chu vi tam giác qua 3 điểm)

Chương trình còn minh hoạ:

- Việc phương thức này sử dụng phương thức khác (phương thức ve_tam_giac sử dụng phương thức ve_doan_thang, phương thức chu_vi sử dụng phương thức do_dai)
- Sử dụng con trỏ this trong thân các phương thức ve_tam_giac và chu_vi

Nội dung chương trình là nhập 3 điểm, vẽ tam giác có đỉnh là 3 điểm vừa nhập sau đó tính chu vi tam giác.

```
#include <conio.h>
```

```
#include <iostream.h>
#include <graphics.h>
#include <math.h>
#include <stdio.h>
class DIEM
{
private:
	int x, y ;
public:
	void nhapsl();
	void ve_doan_thang(DIEM d2, int mau) ;
	void ve_tam_giac(DIEM d2, DIEM d3,int mau) ;
	double do_dai(DIEM d2)
	{
		DIEM d1 = *this ;
		return sqrt(pow(d1.x - d2.x,2) + pow(d1.y - d2.y,2) ) ;
	}
	double chu_vi(DIEM d2, DIEM d3);
};

void DIEM::nhapsl()
{
	cout <<" \n Nhap hoanh do (cot) va tung do (hang) cua diem:" ;
	cin >> x >> y;
}

void kd_do_hoa()
{
	int mh, mode ;
	mh=mode=0;
	initgraph(&mh, &mode, "");
}

void DIEM::ve_doan_thang(DIEM d2, int mau)
{
```

```
        setcolor(mau);
        line(this → x,this → y,d2.x,d2.y);
}
void DIEM:: ve_tam_giac(DIEM d2, DIEM d3,int mau)
{
        (*this).ve_doan_thang(d2,mau);
        d2.ve_doan_thang(d3,mau);
        d3.ve_doan_thang(*this,mau);
}
double DIEM:: chu_vi(DIEM d2, DIEM d3)
{
        double s;
        s=(*this).do_dai(d2)+ d2.do_dai(d3) + d3.do_dai(*this) ;
        return s;
}
void main()
{
        DIEM d1, d2, d3;
        char tb_cv[20] ;
        d1.nhapsl();
        d2.nhapsl();
        d3.nhapsl();
        kd_do_hoa();
        d1.ve_tam_giac(d2,d3,15);
        double s = d1.chu_vi(d2,d3);
        sprintf(tb_cv, "chu_vi = %0.2f", s);
        outtextxy(10,10,tb_cv);
        getch();
        closegraph();
}
```

Một số nhận xét về đối của phương thức và lời gọi phương thức:

+ Quan sát nguyên mẫu phương thức:

```
void ve_doan_thang(DIEM d2, int mau) ;
```

sẽ thấy phương thức có 3 đối:

Đối thứ nhất là một đối tượng DIEM do this trỏ tới

Đối thứ hai là đối tượng DIEM d2

Đối thứ ba là biến nguyên mẫu

Nội dung phương thức là vẽ một đoạn thẳng đi qua các điểm *this và d2 theo mã mầu mau. Xem thân của phương sẽ thấy được nội dung này:

```
void DIEM::ve_doan_thang(DIEM d2, int mau) ;
{
    setcolor(mau);
    line(this → x,this → y,d2.x,d2.y);
}
```

Tuy nhiên trong trường hợp này, vai trò của this không cao lắm, vì nó được đưa vào chỉ cốt làm rõ đối thứ nhất. Trong thân phương thức có thể bỏ từ khóa this vẫn được.

+ Vai trò của this trở nên quan trọng trong phương thức ve_tam_giac:

```
voidve_tam_giac(DIEM d2, DIEM d3, int mau)
```

Phương thức này có 4 đối là:

this : trỏ tới một đối tượng kiểu DIEM

d2 : một đối tượng kiểu DIEM

d3 : một đối tượng kiểu DIEM

mau : một biến nguyên

Nội dung phương thức là vẽ 3 cạnh:

cạnh 1 đi qua *this và d2

cạnh 2 đi qua d2 và d3

cạnh 3 đi qua d3 và *this

Các cạnh trên đuợc vẽ nhờ sử dụng phương thức ve_doan_thang:

Vẽ cạnh 1 dùng lệnh: (*this).ve_doan_thang(d2,mau) ;

Vẽ cạnh 2 dùng lệnh: d2.ve_doan_thang(d3,mau);

Vẽ cạnh 3 dùng lệnh: d3.ve_doan_thang(*this,mau);

Trong trường này rõ ràng vai trò của this rất quan trọng. Nếu không dùng nó thì công việc trở nên khó khăn, dài dòng và khó hiểu hơn. Chúng ta hãy so sánh 2 phương án:

Phương án dùng this trong phương thức ve_tam_giac:

```
void DIEM::ve_tam_giac(DIEM d2, DIEM d3, int mau)
{
    (*this).ve_doan_thang(d2,mau);
    d2.ve_doan_thang(d3,mau); d3.ve_doan_thang(*this,mau);
}
```

phương án không dùng this trong phương thức ve_tam_giac:

```
void DIEM::ve_tam_giac(DIEM d2, DIEM d3, int mau)
{
    DIEM d1;
    d1.x = x; d1.y = y;
    d1.ve_doan_thang(d2,mau);
    d2.ve_doan_thang(d3,mau);
    d3.ve_doan_thang(d1,mau);
}
```

IV. HÀM TẠO (CONSTRUCTOR)

1. Hàm tạo (hàm thiết lập)

Hàm tạo cũng là một phương thức của lớp (nhưng là hàm đặc biệt) dùng để tạo dựng một đối tượng mới. Chương trình dịch sẽ cấp phát bộ nhớ cho đối tượng sau đó sẽ gọi đến hàm tạo. Hàm tạo sẽ khởi gán giá trị cho các thuộc tính của đối tượng và có thể thực hiện một số công việc khác nhằm chuẩn bị cho đối tượng mới.

a. Cách viết hàm tạo

i. Điểm khác của hàm tạo và các phương thức thông thường:

Khi viết hàm tạo cần để ý 3 sự khác biệt của hàm tạo so với các phương thức khác như sau:

- Tên của hàm tạo: Tên của hàm tạo bắt buộc phải trùng với tên của lớp.
- Không khai báo kiểu cho hàm tạo.
- Hàm tạo không có kết quả trả về.

ii. Sự giống nhau của hàm tạo và các phương thức thông thường

Ngoài 3 điểm khác biệt trên, hàm tạo được viết như các phương thức khác:

- Hàm tạo có thể được xây dựng bên trong hoặc bên ngoài định nghĩa lớp.

- Hàm tạo có thể có đối hoặc không có đối.
- Trong một lớp có thể có nhiều hàm tạo (cùng tên nhưng khác bộ đối).

Ví dụ sau định nghĩa lớp DIEM_DH (Điểm đồ họa) có 3 thuộc tính:

```
int x;          // hoành độ (cột) của điểm
int y;          // tung độ (hàng) của điểm
int m;          // mầu của điểm
```

và đưa vào 2 hàm tạo để khởi gán cho các thuộc tính của lớp:

```
// Hàm tạo không đối: Dùng các giá trị cố định để khởi gán cho x, y, m
DIEM_DH() ;
// Hàm tạo có đối: Dùng các đối x1, y1, m1 để khởi gán cho x, y, m
DIEM_DH(int x1, int y1, int m1 = 15) ;      // Đối m1 có giá trị mặc định 15
class DIEM_DH                               // (mầu trắng)
{
private:
    int x, y, m ;
public:
    // Hàm tạo không đối: khởi gán cho x = 0, y = 0, m = 1
    // Hàm này viết bên trong định nghĩa lớp
    DIEM_DH()
    {
        x = y = 0;
        m = 1;
    }
    // Hàm tạo này xây dựng bên ngoài định nghĩa lớp
    DIEM_DH(int x1, int y1, int m1 = 15) ;
    // Các phương thức khác
};

// Xây dựng hàm tạo bên ngoài định nghĩa lớp
DIEM_DH:: DIEM_DH(int x1, int y1, int m1) ;
{
    x = x1; y = y1; m = m1;
}
```

b. Dùng hàm tạo trong khai báo

+ Khi đã xây dựng các hàm tạo, ta có thể dùng chúng trong khai báo để tạo ra một đối tượng đồng thời khởi gán cho các thuộc tính của đối tượng được tạo. Dựa vào các tham số trong khai báo mà trình biên dịch sẽ biết cần gọi đến hàm tạo nào.
+ Khi khai báo một biến đối tượng có thể sử dụng các tham số để khởi gán cho các thuộc tính của biến đối tượng.
+ Khi khai báo mảng đối tượng không cho phép dùng các tham số để khởi gán.
+ Câu lệnh khai báo một biến đối tượng sẽ gọi tới hàm tạo 1 lần.
+ Câu lệnh khai báo một mảng n đối tượng sẽ gọi tới hàm tạo n lần.

Ví dụ:

```
DIEM_DH d;                      // Gọi tới hàm tạo không đối.
// Kết quả d.x = 0, d.y = 0, d.m = 1
DIEM_DH u(300, 100, 5);         // Gọi tới hàm tạo có đối.
// Kết quả u.x = 300, u.y = 100, d.m = 5
DIEM_DH v(400, 200);            // Gọi tới hàm tạo có đối.
// Kết quả v.x = 400, v.y = 200, d.m = 15
DIEM_DH p[20] ;                 // Gọi tới hàm tạo không đối 20 lần
```

Chú ý: Với các hàm có đối kiểu lớp, thì đối chỉ xem là các tham số hình thức, vì vậy khai báo đối (trong dòng đầu của hàm) sẽ không tạo ra đối tượng mới và do đó không gọi tới các hàm tạo.

c. Dùng hàm tạo trong cấp phát bộ nhớ

+ Khi cấp phát bộ nhớ cho một đối tượng có thể dùng các tham số để khởi gán cho các thuộc tính của đối tượng, ví dụ

```
DIEM_DH *q = new DIEM_DH(40, 20, 4);        // Gọi tới hàm tạo có đối
// Kết quả q → x = 40, q → y = 20, q → m = 4
DIEM_DH *r = new DIEM_DH ;                  //Gọi tới hàm tạo không đối
// Kết quả r → x = 0, r → y = 0, r → m = 1
```

+ Khi cấp phát bộ nhớ cho một dãy đối tượng không cho phép dùng tham số để khởi gán, ví dụ:

```
int n = 30;
DIEM_DH *s = new DIEM_DH[n] ;     // Gọi tới hàm tạo không đối 30 lần.
```

d. Dùng hàm tạo để biểu diễn các đối tượng hằng

+ Như đã biết, sau khi định nghĩa lớp DIEM_DH thì có thể xem lớp này như một kiểu dữ liệu như int, double, char, ...

Với kiểu int chúng ta có các hằng int, như 253.

Với kiểu double chúng ta có các hằng double, như 75.42

Khái niệm hằng kiểu int, hằng kiểu double có thể mở rộng cho hằng kiểu DIEM_DH

+ Để biểu diễn một hằng đối tượng (hay còn gọi: Đối tượng hằng) chúng ta phải dùng tới hàm tạo. Mẫu viết như sau:

Tên_lớp(danh sách tham số) ;

Ví dụ đối với lớp DIEM_DH nói trên, có thể viết như sau:

```
DIEM_DH(234, l 23, 4)      // Biểu thị một đối tượng kiểu DIEM_DH
                           // có các thuộc tính x = 234, y = 123, m = 4
```

Chú ý: Có thể sử dụng một hằng đối tượng như một đối tượng. Nói cách khác, có thể dùng hằng đối tượng để thực hiện một phương thức, ví dụ nếu viết:

```
DIEM_DH(234, l 23, 4).in();
```

thì có nghĩa là thực hiện phương thức in() đối với hằng đối tượng.

e. Ví dụ minh họa

Chương trình sau đây minh họa cách xây dựng hàm tạo và cách sử dùng hàm tạo trong khai báo, trong cấp phát bộ nhớ và trong việc biểu diễn các hằng đối tượng.

```
#include <conio.h>
#include <iostream.h>
#include <iomanip.h>
class DIEM_DH
{
private:
    int x, y, m;
public:
    // Hàm bạn dùng để in đối tượng DIEM_DH
    friend void in(DIEM_DH d)
    {
        cout <<"\n " << d.x << " "<< d.y<<" " << d.m ;
    }
    // Phương thức dùng để in đối tượng DIEM_DH
```

```
        void in()
        {
                cout <<"\n " << x << " "<< y<<" " << m ;
        }
        // Hàm tạo không đối
        DIEM_DH()
        {
                x = y = 0;
                m = 1;
        }
        // Hàm tạo có đối, đối m1 có giá trí mặc định là 15 (mầu trắng)
        DIEM_DH(int x1, int y1, int m1 = 15);
};

// Xây dựng hàm tạo
DIEM_DH::DIEM_DH(int x1, int y1, int m1)
{
        x = x1; y = y1; m = m1;
}

void main()
{
        DIEM_DH d1;                     // Gọi tới hàm tạo không đối
        DIEM_DH d2(200, 200, 10);       // Gọi tới hàm tạo có đối
        DIEM_DH*d;
        d = new DIEM_DH(300, 300);      // Gọi tới hàm tạo có đối
        clrscr();
        in(d1);                         //Gọi hàm bạn in()
        d2.in();                        //Gọi phương thức in()
        in(*d);                         // Gọi hàm bạn in()
        DIEM_DH(2, 2, 2).in();          // Gọi phương thức in()
        DIEM_DH t[3];                   // 3 lần gọi hàm tạo không đối
        DIEM_DH*q;                      // Gọi hàm tạo không đối
        int n;
```

```
        cout << "\n N = "; cin >> n;
        q = new DIEM_DH[n+1];          // (n+1) lần gọi hàm tạo không đối
        for (int i = 0; i< = n; ++i)
            q[i] = DIEM_DH(300+i, 200+i, 8); //(n+1) lần gọi hàm tạo có đối
        for (i = 0; i< = n; ++i)
            q[i].in();                          // Gọi phương thức in()
        for (i = 0; i< = n; ++i)
            DIEM_DH(300+i, 200+i, 8).in();      // Gọi phương thức in()
        getch();
}
```

2. Lớp không có hàm tạo và hàm tạo mặc định

a. Nếu lớp không có hàm tạo

Chương trình dịch sẽ cung cấp một hàm tạo mặc định không đối (default), hàm này thực chất không làm gì cả. Như vậy một đối tượng tạo ra chỉ được cấp phát bộ nhớ, còn các thuộc tính của nó chưa được xác định. Chúng ta có thể kiểm chứng điều này, bằng cách chạy chương trình sau:

```
// Hàm tạo mặc định
#include <conio.h>
#include <iostream.h>
class DIEM_DH
{
private:
    int x, y, m;
public:
    // Phương thức
    void in() { cout <<"\n " << x << " "<< y<<" " << m ; }
};

void main()
{
    DIEM_DH d;
    d.in();
    DIEM_DH *p;
```

```
    p = new DIEM_DH[10];
    clrscr();
    d.in();
    for (int i = 0; i<10; ++i) (p+i)->in();
    getch();
}
```

b. Nếu trong lớp đã có ít nhất một hàm tạo

Khi đó hàm tạo mặc định sẽ không được phát sinh nữa và mọi câu lệnh xây dựng đối tượng mới đều sẽ gọi đến một hàm tạo của lớp. Nếu không tìm thấy hàm tạo cần gọi thì chương trình dịch sẽ báo lỗi. Điều này thường xẩy ra khi chúng ta không xây dựng hàm tạo không đối, nhưng lại sử dụng các khai báo không tham số như ví dụ sau:

```
#include <conio.h>
#include <iostream.h>
class DIEM_DH
{
private:
    int x, y, m;
public:
    // Phương thức dùng để in đối tượng DIEM_DH
    void in()
    {
        cout <<"\n" << x << " "<< y<<" " << m ;
    }
    //Hàm tạo có đối
    DIEM_DH::DIEM_DH(int x1, int y1 , int m1)
    {
        x = x1; y = y1 ; m = m1;
    }
};

void main()
{
    DIEM_DH d1(200, 200, 10); // Gọi tới hàm tạo có đối
```

```
    DIEM_DH d2; // Gọi tới hàm tạo không đối
    d2 = DIEM_DH(300, 300, 8); // Gọi tới hàm tạo có đối
    d1.in();
    d2.in();
    getch();
}
```

Trong các câu lệnh trên, chỉ có câu lệnh thứ 2 trong hàm main() là bị báo lỗi. Câu lệnh này sẽ gọi tới hàm tạo không đối, mà hàm này chưa được xây dựng.

Giải pháp: có thể chọn một trong 2 giải pháp sau:

- Xây dựng thêm hàm tạo không đối.
- Gán giá trị mặc định cho tất cả các đối x1, y1 và m1 của hàm tạo đã xây dựng ở trên.

Theo phương án 2, chương trình có thể sửa như sau:

```
#include <conio.h>
#include <iostream.h>
class DIEM_DH
{
private:
    int x, y, m;
public:
    // Phương thức dùng để in đối tượng DIEM_DH
    void in() { cout <<"\n " << x << " "<< y<<" " << m ; }
    // Hàm tạo có đối , tất cả các đối đều có giá trị mặc định
    DIEM_DH::DIEM_DH(int x1 = 0, int y1 = 0, int m1 = 15)
    {
        x = x1; y = y1; m = m1;
    }
};

void main()
{
    // Gọi tới hàm tạo, không dùng tham số mặc định
    DIEM_DH d1(200, 200, 10);
    // Gọi tới hàm tạo, dùng 3 tham số mặc định
```

```
    DIEM_DH d2;
    // Gọi tới hàm tạo, dùng 1 tham số mặc định
    d2 = DIEM_DH(300, 300);
    d1.in();
    d2.in();
    getch();
}
```

3. Hàm tạo sao chép (Copy Constructor)

a. Hàm tạo sao chép mặc định

Giả sử đã định nghĩa một lớp nào đó, ví dụ lớp PS (phân số). Khi đó:

+ Ta có thể dùng câu lệnh khai báo hoặc cấp phát bộ nhớ để tạo các đối tượng mới, ví dụ:

```
PS p1, p2 ;
PS *p = new PS ;
```

+ Ta cũng có thể dùng lệnh khai báo để tạo một đối tượng mới từ một đối tượng đã tồn tại, ví dụ:

```
PS u;
PS v(u) ; // Tạo v theo u
```

ý nghĩa của câu lệnh này như sau:

– Nếu trong lớp PS chưa xây dựng hàm tạo sao chép, thì câu lệnh này sẽ gọi tới một hàm tạo sao chép mặc định (của C++). Hàm này sẽ sao chép nội dung từng bit của u vào các bit tương ứng của v. Như vậy các vùng nhớ của u và v sẽ có nội dung như nhau. Rõ ràng trong đa số các trường hợp, nếu lớp không có các thuộc tính kiểu con trỏ hay tham chiếu, thì việc dùng các hàm tạo sao chép mặc định (để tạo ra một đối tượng mới có nội dung như một đối tượng cho trước) là đủ và không cần xây dựng một hàm tạo sao chép mới.

– Nếu trong lớp PS đã có hàm tạo sao chép (cách viết sẽ nói sau) thì câu lệnh: PS v(u); sẽ tạo ra đối tượng mới v, sau đó gọi tới hàm tạo sao chép để khởi gán v theo u.

Ví dụ sau sẽ minh họa cách dùng hàm tạo sao chép mặc định:

Trong chương trình đưa vào lớp PS (phân số):

+ Các thuộc tính gồm: t (tử số) và m (mẫu).

+ Trong lớp không có phương thức nào cả mà chỉ có 2 hàm bạn là các hàm toán tử nhập (>>) và xuất (<<).

+ Nội dung chương trình là: Dùng lệnh khai báo để tạo một đối tượng u (kiểu PS) có nội dung như đối tượng đã có d.

```
// Ham tao sao chep mac dinh
#include <conio.h>
#include <iostream.h>
class PS
{
private:
	int t, m ;
public:
	friend ostream& operator<< (ostream&os, const PS &p)
	{
		os << " = " << p.t << "/" << p.m;
		return os;
	}
	friend istream& operator>> (istream& is, PS &p)
	{
		cout << "\n Nhap tu va mau: " ;
		is >> p.t >> p.m ;
		return is;
	}
} ;

void main()
{
	PS d;
	cout << "\n Nhap PS d "; cin >> d;
	cout << "\n PS d " << d;
	PS u(d);
	cout << "\n PS u "<< u;
	getch();
}
```

b. Cách xây dựng hàm tạo sao chép

+ Hàm tạo sao chép sử dụng một đối kiểu tham chiếu đối tượng để khởi gán

cho đối tượng mới. Hàm tạo sao chép được viết theo mẫu:

```
Tên_lớp (const Tên_lớp & dt)
{
        // Các câu lệnh dùng các thuộc tính của đối tượng dt
        // để khởi gán cho các thuộc tính của đối tượng mới
}
```

+ Ví dụ có thể xây dựng hàm tạo sao chép cho lớp PS như sau:

```
class PS
{
private:
        int t, m ;
public:
        PS (const PS &p)
        {
                this->t = p.t ;
                this->m = p.m ;
        }
        ...
} ;
```

c. Khi nào cần xây dựng hàm tạo sao chép

+ Nhận xét: Hàm tạo sao chép trong ví dụ trên không khác gì hàm tạo sao chép mặc định.
+ Khi lớp không có các thuộc tính kiểu con trỏ hoặc tham chiếu, thì dùng hàm tạo sao chép mặc định là đủ.
+ Khi lớp có các thuộc tính con trỏ hoặc tham chiếu, thì hàm tạo sao chép mặc định chưa đáp ứng được yêu cầu.

Ví dụ:

```
class DT
{
private:
        int n; // Bac da thuc
        double *a; // Tro toi vung nho chua cac he so da thuc a0, a1, ...
public:
```

```
        DT() { this->n0; this->a = NULL; }
        DT(int n1)
        {
            this->n = n1;
            this->a = new double[n1+1];
        }
        friend ostream& operator << (ostream& os, const DT &d);
        friend istream& operator>> (istream& is, DT &d);
        ...
    } ;
```

Bây giờ chúng ta hãy theo dõi xem việc dùng hàm tạo mặc định trong đoạn chương trình sau sẽ dẫn đến sai lầm như thế nào:

```
        DT d ; // Tạo đối tượng d kiểu DT
        cin >> d ;
```

/* Nhập đối tượng d, gồm: nhập một số nguyên dương và gán cho d.n, cấp phát vùng nhớ cho d.a, nhập các hệ số của đa thức và chứa vào vùng nhớ được cấp phát */

```
        DT u(d);
```

/* Dùng hàm tạo mặc định để xây dựng đối tượng u theo d. Kết quả: u.n = d.n và u.a = d.a. Như vậy 2 con trỏ u.a và d.a cùng trỏ đến một vùng nhớ */

Nhận xét: Mục đích là tạo ra một đối tượng u giống như d, nhưng độc lập với d. Nghĩa là khi d thay đổi thì u không bị ảnh hưởng gì. Thế nhưng mục tiêu này không đạt được, vì u và d có chung một vùng nhớ chứa hệ số của đa thức, nên khi sửa đổi các hệ số của đa thức trong d thì các hệ số của đa thức trong u cũng thay đổi theo. Còn một trường hợp nữa cũng dẫn đến lỗi là khi một trong 2 đối tượng u và d bị giải phóng (thu hồi vùng nhớ chứa đa thức) thì đối tượng còn lại cũng sẽ không còn vùng nhớ nữa.

Ví dụ sau sẽ minh họa nhận xét trên: Khi d thay đổi thì u cũng thay đổi và ngược lại khi u thay đổi thì d cũng thay đổi theo.

```
#include <conio.h>
#include <iostream.h>
#include <math.h>
class DT
{
private:
    int n; // Bac da thuc
```

```
        double *a; // Tro tơi vung nho chua cac he so da thuc a0, a1 , ...
public:
        DT() { this->n = 0; this->a = NULL; }
        DT(int n1)
        {
                this->n = n1 ;
                this->a = new double[n1+1];
        }
        friend ostream& operator<< (ostream& os, const DT &d);
        friend istream& operator>> (istream& is, DT &d);
} ;

ostream& operator<< (ostream& os, const DT &d)
{
        os << " Cac he so (tu ao): ";
        for (int i = 0 ; i< = d.n ; ++i)
        os << d.a[i] <<" " ;
        return os;
}

istream& operator >> (istream& is, DT &d)
{
        if (d.a! = NULL) delete d.a;
        cout << " \n Bac da thuc: " ;
        cin >> d.n;
        d.a = new double[d.n+1];
        cout << "Nhap cac he so da thuc:\n" ;
        for (int i = 0 ; i< = d.n ; ++i)
        {
                cout << "He so bac "<< i << " = " ;
                is >> d.a[i] ;
        }
        return is;
}
```

```
void main()
{
        DT d;
        clrscr();
        cout <<"\n Nhap da thuc d " ; cin >> d;
        DT u(d);
        cout << "\n Da thuc d "<< d ;
        cout << "\n Da thuc u " << u ;
        cout <<"\n Nhap da thuc d " ; cin >> d;
        cout << "\nDa thuc d " << d;
        cout <<"\n Da thuc u " << u ;
        cout <<"\n Nhap da thuc u " ; cin >> u;
        cout << "\n Da thuc d "<< d ;
        cout << "\n Da thuc u " << u ;
        getch();
}
```

d. Ví dụ về hàm tạo sao chép

Trong chương trình trên đã chỉ rõ: Hàm tạo sao chép mặc định là chưa thoả mãn đối với lớp DT. Vì vậy cần viết hàm tạo sao chép để xây dựng đối tượng mới (ví dụ u) từ một đối tượng đang tồn tại (ví dụ d) theo các yêu cầu sau:

+ Gán d.n cho u.n
+ Cấp phát một vùng nhớ cho u.a để có thể chứa được (d.n + 1) hệ số.
+ Gán các hệ số chứa trong vùng nhớ của d.a sang vùng nhớ của u.a

Như vậy chúng ta sẽ tạo được đối tượng u có nội dung ban đầu giống như d, nhưng độc lập với d.

Để đáp ứng các yêu cầu nêu trên, hàm tạo sao chép cần được xây dựng như sau:

```
DT::DT(const DT &d)
{
        this → n = d.n ;
        this → a = new double[d.n+1];
        for (int i = 0; i< = d.n; ++i)
        this → a[i] = d.a[i];
}
```

Chương trình sau sẽ minh họa điều này: Sự thay đổi của d không làm ảnh hưởng đến u và ngược lại sự thay đổi của u không làm ảnh hưởng đến d.

```
// Viết hàm tạo sao chép cho lớp DT
#include <conio.h>
#include <iostream.h>
#include <math.h>
class DT
{
private:
    int n; // Bac da thuc
    double *a; // Tro toi vung nho chua cac he so da thuc a0, a1 , ...
public:
    DT() { this → n = 0; this → a = NULL; }
    DT(int n1)
    {
        this → n = n1 ;
        this → a = new double[n1+1];
    }
    DT(const DT &d);
    friend ostream& operator<< (ostream& os, const DT&d);
    friend istream& operator>> (istream& is, DT&d);
};

DT::DT(const DT&d)
{
    this → n = d.n;
    this → a = new double[d.n+1];
    for (int i = 0; i< = d.n; ++i)
    this → a[i] = d.a[i];
}

ostream& operator<< (ostream& os, const DT &d)
{
    os << " Cac he so (tu ao): " ;
```

```
        for (int i = 0 ; i< = d.n ; ++i) os << d.a[ i] <<" " ;
        return os;
}

istream& operator>> (istream& is, DT &d)
{
        if (d.a! = NULL) delete d.a;
        cout << "\n Bac da thuc: " ;
        cin >> d.n;
        d.a = new double[d.n+1];
        cout << "Nhap cac he so da thuc:\n" ;
        for (int i = 0 ; i< = d.n ; ++i)
        {
                cout << "He so bac " << i << " = " ;
                is >> d.a[i] ;
        }
        return is;
}

void main()
{
        DT d;
        clrscr();
        cout <<"\n Nhap da thuc d " ; cin >> d;
        DT u(d);
        cout <<"\n Da thuc d " << d ;
        cout << "\n Da thuc u " << u ;
        cout <<"\n Nhap da thuc d " ; cin >> d;
        cout << "\n Da thuc d "<< d ;
        cout <<"\n Da thuc u " << u ;
        cout <<"\n Nhap da thuc u " ; cin >> u;
        cout << "\n Da thuc d " << d ;
        cout <<"\n Da thuc u " << u ;
        getch();
```

```
}
```

V. HÀM HỦY (DESTRUCTOR)

Hàm hủy là một hàm thành viên của lớp (phương thức) có chức năng ngược với hàm tạo. Hàm hủy được gọi trước khi giải phóng (xoá bỏ) một đối tượng để thực hiện một số công việc có tính "dọn dẹp" trước khi đối tượng được hủy bỏ, ví dụ như giải phóng một vùng nhớ mà đối tượng đang quản lý, xoá đối tượng khỏi màn hình nếu như nó đang hiển thị, ...

Việc hủy bỏ một đối tượng thường xẩy ra trong 2 trường hợp sau:

+ Trong các toán tử và các hàm giải phóng bộ nhớ, như delete, free, ...
+ Giải phóng các biến, mảng cục bộ khi thoát khỏi hàm, phương thức.

1. Hàm hủy mặc định

Nếu trong lớp không định nghĩa hàm hủy, thì một hàm hủy mặc định không làm gì cả được phát sinh. Đối với nhiều lớp thì hàm hủy mặc định là đủ, và không cần đưa vào một hàm hủy mới.

2. Quy tắc viết hàm hủy

Mỗi lớp chỉ có một hàm hủy viết theo các quy tắc sau:

+ Kiểu của hàm: Hàm hủy cũng giống như hàm tạo là hàm không có kiểu, không có giá trị trả về.
+ Tên hàm: Tên của hàm hủy gồm một dấu ngã (đứng trước) và tên lớp:

  ~Tên_lớp
+ Đối: Hàm hủy không có đối

Ví dụ có thể xây dựng hàm hủy cho lớp DT (đa thức) như sau:

```
class DT
{
private:
    int n; // Bac da thua
    double *a; // Tro toi vung nho chua cac he so da thuc a0, a1 , ...
public:
    ~DT()
    {
        this → n = 0;
        delete this → a;
```

```
        }
    ...
};
```

3. Vai trò của hàm hủy trong lớp DT

Trong phần trước định nghĩa lớp DT (đa thức) khá đầy đủ gồm:

+ Các hàm tạo
+ Các toán tử nhập >>, xuất <<
+ Các hàm toán tử thực hiện các phép tính +, -, *, /

Tuy nhiên vẫn còn thiếu hàm hủy để giải phóng vùng nhớ mà đối tượng kiểu DT (cần hủy) đang quản lý.

Chúng ta hãy phân tích các khiếm khuyết của chương trình này:

+ Khi chương trình gọi tới một phương thức toán tử để thực hiện các phép tính cộng, trừ, nhân đa thức, thì một đối tượng trung gian được tạo ra. Một vùng nhớ được cấp phát và giao cho nó (đối tượng trung gian) quản lý.
+ Khi thực hiện xong phép tính sẽ ra khỏi phương thức. Đối tượng trung gian bị xoá, tuy nhiên chỉ vùng nhớ của các thuộc tính của đối tượng này được giải phóng. Còn vùng nhớ (chứa các hệ số của đa thức) mà đối tượng trung gian đang quản lý thì không hề bị giải phóng. Như vậy số vùng nhớ bị chiếm dụng vô ích sẽ tăng lên.

Nhược điểm trên dễ dàng khắc phục bằng cách đưa vào lớp DT hàm hủy trong mục 3 ở trên.

4. Ví dụ

Phần này chúng tôi trình bày một ví dụ tương đối hoàn chỉnh về lớp các hình tròn trong chế độ đồ họa. Chương trình gồm:

i. Lớp HT (hình tròn) với các thuộc tính:

```
int r;                  // Bán kính
int m;                  // Mầu hình tròn
int xhien, yhien;       // Vị trí hiển thị hình tròn trên màn hình
char *pht;              // Con trỏ trỏ tới vùng nhớ chứa ảnh hình tròn
int hienmh;             // Trạng thái hiện (hienmh = 1), ẩn (hienmh = 0)
```

ii. Các phương thức

+ Hàm tạo không đối thực hiện việc gán giá trị bằng 0 cho các thuộc tính của lớp. HT();
+ Hàm tạo có đối. HT(int n, int m1 = 15);

Thực hiện các việc:

– Gán r1 cho r, m1 cho m

– Cấp phát bộ nhớ cho pht

– Vẽ hình tròn và lưu ảnh hình tròn vào vùng nhớ của pht

+ Hàm hủy: ~HT();

Thực hiện các việc:

– Xoá hình tròn khỏi màn hình (nếu đang hiển thị)

– Giải phóng bộ nhớ đã cấp cho pht

+ Phương thức: void hien(int x, int y);

Có nhiệm vụ hiển thị hình tròn tại (x, y)

+ Phương thức : void an()

Có nhiệm vụ làm ẩn hình tròn

iii. Các hàm độc lập:

```
void ktdh();                    // Khởi tạo đồ họa
void ve_bau_troi();             // Vẽ bầu trời sao
void ht_di_dong_xuong();        // Vẽ một cặp 2 hình tròn di chuyển xuống
void ht_di_dong_len();          // Vẽ một cặp 2 hình tròn di chuyển lên trên
```

Nội dung chương trình là tạo ra các chuyển động xuống và lên của các hình tròn.

```
// Lop do hoa
// Ham huy
// Trong ham huy co the goi PT khac
#include <conio.h>
#include <iostream.h>
#include <math.h>
#include <stdlib.h>
#include <graphics.h>
#include <dos.h>
void ktdh();                    // Khởi tạo đồ họa
void ve_bau_troi();             // Vẽ bầu trời sao
void ht_di_dong_xuong();        // Vẽ một cặp 2 hình tròn di chuyển xuống
void ht_di_dong_len();          // Vẽ một cặp 2 hình tròn di chuyển lên trên
```

```
int xmax, ymax;
class HT
{
private:
        int r, m ;
        int xhien, yhien;
        char *pht;
        int hienmh;
public:
        HT();
        HT(int n, int m1 = 15);
        ~HT();
        void hien(int x, int y);
        void an();
};

HT:: HT()
{
        r = m = hienmh = 0;
        xhien = yhien = 0;
        pht = NULL;
}

HT::HT(int n, int m1)
{
        r = n; m = m1; hienmh = 0;
        xhien = yhien = 0;
        if (r<0) r = 0;
        if (r = = 0) pht = NULL;
        else
        {
                int size; char *pmh;
                size = imagesize(0, 0, r+r, r+r);
                pmh = new char[size];
```

```
            getimage(0, 0, r+r, r+r, pmh);
            setcolor(m);
            circle(r, r, r );
            setfillstyle(1, m);
            floodfill(r, r, m);
            pht = new char[size];
            getimage(0, 0, r+r, r+r, pht);
            putimage(0, 0, pmh, COPY_PUT);
            delete pmh;
            pmh = NULL;
        }
    }

    void HT::hien(int x, int y)
    {
        if (pmh! = NULL && !hienmh) // Chua hien
        {
            hienmh = 1;
            xhien = x; yhien = y;
            putimage(x, y, pht, XOR_PUT);
        }
    }

    void HT::an()
    {
        if (hienmh) // Dang hien
        {
            hienmh = 0;
            putimage(xhien, yhien, pht, XOR_PUT);
        }
    }

    HT::~HT()
    {
        an();
```

```
        if (pht! = NULL)
        {
                delete pht;
                pht = NULL;
        }
}

void ktdh()
{
        int mh = 0, mode = 0;
        initgraph(&mh, &mode, " ");
        xmax = getmaxx();
        ymax = getmaxy();
}

void ve_bau_troi()
{
        for (int i = 0; i<2000; ++i)
        putpixel(random(xmax), random(ymax), 1+random( 15));
}

void ht_di_dong_xuong()
{
        HT h(50, 4);
        HT u(60, 15);
        h.hien(0, 0);
        u.hien(40, 0);
        for (int x = 0; x< = 340; x+ = 10)
        {
                h.an();
                u.an();
                h.hien(x, x);
                delay(200);
                u.hien(x+40, x);
                delay(200);
```

```
        }
}

void ht_di_dong_len()
{
        HT h(50, 4);
        HT u(60, 15);
        h.hien(340, 340);
        u.hien(380, 340);
        for (int x = 340; x> = 0; x- = 10)
        {
                h.an();
                u.an();
                u.hien(x, x);
                delay(200);
                u.hien(x+40, x);
                delay(200);
        }
};

void main()
{
        ktdh();
        ve_bau_troi();
        ht_di_dong_xuong();
        ht_di_dong_len();
        getch();
        closegraph();
}
```

Các nhận xét:

+ Trong thân hàm hủy gọi tới phương thức an().
+ Điều gì xẩy ra khi bỏ đi hàm hủy:
  - Khi gọi hàm ht_di_dong_xuong() thì có 2 đối tượng kiểu HT được tạo ra. Trong thân hàm sử dụng các đối tượng này để vẽ các hình tròn di

chuyển xuống. Khi thoát khỏi hàm thì 2 đối tượng (tạo ra ở trên) được giải phóng. Vùng nhớ của các thuộc tính của chúng bị thu hồi, nhưng vùng nhớ cấp phát cho thuộc tính pht chưa được giải phóng và ảnh của 2 hình tròn (ở phía dưới màn hình) vẫn không được cất đi.

- Điều tương tự xẩy ra sau khi ra khỏi hàm ht_di_dong_len(): vùng nhớ cấp phát cho thuộc tính pht chưa được giải phóng và ảnh của 2 hình tròn (ở phía trên màn hình) vẫn không được thu dọn.

VI. CÁC HÀM TRỰC TUYẾN (INLINE)

Một số mở rộng của C++ đối với C đã được trình bày trong các chương trước như biến tham chiếu, định nghĩa chồng hàm, hàm với đối mặc định … Phần này ta xem một đặc trưng khác của C++ được gọi là hàm trực tuyến (inline).

1. Ưu nhược điểm của hàm

Việc tổ chức chương trình thành các hàm có 2 ưu điểm rõ rệt:

Thứ nhất là chia chương trình thành các đơn vị độc lập, làm cho chương trình được tổ chức một cách khoa học dễ kiểm soát, dễ phát hiện lỗi, dễ phát triển và mở rộng.

Thứ hai là giảm được kích thước chương trình, vì mỗi đoạn chương trình thực hiện nhiệm vụ của hàm được thay bằng một lời gọi hàm.

Tuy nhiên hàm cũng có nhược điểm là làm chậm tốc độ chương trình do phải thực hiện một số thao tác có tính thủ tục mỗi khi gọi hàm như: cấp phát vùng nhớ cho các đốivà biến cục bộ, truyền dữ liệu của các tham số cho các đối, giải phóng vùng nhớ trước khi thoát khỏi hàm.

Các hàm trực tuyến trong C++ có khả năng khắc phục được các nhược điểm nói trên.

2. Các hàm trực tuyến

Để biến một hàm thành trực tuyến ta viết thêm từ khoá **inline** vào trước khai báo nguyên mẫu hàm. Nếu không dùng nguyên mẫu thì viết từ khoá này trước dòng đầu tiên của định nghĩa hàm.

Ví dụ 1 :

```
inline float f(int n, float x);
float f(int n, float x)
{
      // Các câu lệnh trong thân hàm
}
```

hoặc

```
inline float f(int n, float x)
{
    // Các câu lệnh trong thân hàm
}
```

Chú ý: Trong mọi trường hợp, từ khoá inline phải xuất hiện trước các lời gọi hàm thì trình biên dịch mới biết cần xử lý hàm theo kiểu inline.

Ví dụ hàm f trong chương trình sau sẽ không phải là hàm trực tuyến vì từ khoá inline viết sau lời gọi hàm:

```
#include <conio.h>
#include <iostream.h>
void main()
{
    int s ;
    s = f(5,6);
    cout << s ;
    getch();
}
inline int f(int a, int b)
{
    return a*b;
}
```

Chú ý: Trong C^{++}, nếu hàm được xây dựng sau lời gọi hàm thì bắt buộc phải khai báo nguyên mẫu hàm trước lời gọi. Trong ví dụ trên, trình biên dịch C^{++} sẽ bắt lỗi vì thiếu khai báo nguyên ngẫu hàm f .

3. Cách biên dịch và dùng hàm trực tuyến

Chương trình dịch xử lý các hàm inline như các macro (được định nghĩa trong lệnh #define), nghĩa là nó sẽ thay mỗi lời gọi hàm bằng một đoạn chương trình thực hiện nhiệm vụ của hàm. Cách này làm cho chương trình dài ra, nhưng tốc độ chương trình tăng lên do không phải thực hiện các thao tác có tính thủ tục khi gọi hàm.

Phương án dùng hàm trực tuyến rút ngắn được thời gian chạy máy nhưng lại làm tăng khối lượng bộ nhớ chương trình (nhất là đối với các hàm trực tuyến có nhiều câu lệnh). Vì vậy chỉ nên dùng phương án trực tuyến đối với các hàm nhỏ.

4. Sự hạn chế của trình biên dịch

Không phải khi gặp từ khoá inline là trình biên dịch nhất thiết phải xử lý hàm theo kiểu trực tuyến.

Có một số hàm mà các trình biên dịch thường không xử lý theo cách inline như các hàm chứa biến static, hàm chứa các lệnh chu trình hoặc lệnh goto hoặc lệnh switch, hàm đệ quy. Trong trường hợp này từ khoá inline lẽ dĩ nhiên bị bỏ qua.

Thậm chí từ khoá inline vẫn bị bỏ qua ngay cả đối với các hàm không có những hạn chế nêu trên nếu như trình biên dịch thấy cần thiết (ví dụ đã có quá nhiều hàm inline làm cho bộ nhớ chương trình quá lớn)

Ví dụ 2 : Chương trình sau sử dụng hàm inline tính chu vi và diện tích của hình chữ nhật:

Cách 1: Không khai báo nguyên mẫu. Khi đó hàm dtcvhcn phải đặt trước hàm main.

```
#include <conio.h>
#include <iostream.h>
inline void dtcvhcn(int a, int b, int &dt, int &cv)
{
	dt=a*b;
	cv=2*(a+b);
}
void main()
{
	int a[20],b[20],cv[20],dt[20],n;
	cout << "\n So hinh chu nhat: " ;
	cin >> n;
	for (int i=1; i<=n; ++i)
	{
		cout <<"\n Nhap 2 canh cua hinh chu nhat thu " << i << ": ";
		cin >> a[i] >> b[i];
		dtcvhcn(a[i],b[i],dt[i], cv[i]);
	}
	clrscr();
	for (i=1; i<=n; ++i)
	{
		cout << "\n Hinh chu nhat thu "<< i << " : ";
```

```
            cout << "\n Do dai 2 canh= " << a[i] << " va " << b[i] ;
            cout <<"\n Dien tich= " << dt[i] ;
            cout << "\n Chu vi= " << cv[i] ;
        }
        getch();
    }
```

Cách 2:Sử dụng khai báo nguyên mẫu. Khi đó từ khoá inline đặt trước nguyên mẫu.

Chú ý: Không được đặt inline trước định nghĩa hàm. Trong chương trình dưới đây nếu đặt inline trước định nghĩa hàm thì hậu quả như sau: Chương trình vẫn dịch thông, nhưng khi chạy thì chương trình bị quẩn và không thoát đi được.

```
#include <conio.h>
#include <iostream.h>
inline void dtcvhcn(int a, int b, int &dt, int &cv);
void main()
{
    int a[20],b[20],cv[20],dt[20],n;
    cout << "\n So hinh chu nhat: " ;
    cin >> n;
    for (int i=1; i<=n; ++i)
    {
        cout <<"\n Nhap 2 canh cua hinh chu nhat thu " << i << ": ";
        cin >> a[i] >> b[i];
        dtcvhcn(a[i],b[i],dt[i], cv[i]);
    }
    clrscr();
    for (i=1; i<=n; ++i)
    {
        cout << "\n Hinh chu nhat thu "<< i << " : ";
        cout << "\n Do dai 2 canh= " << a[i] << " va " << b[i] ;
        cout <<"\n Dien tich= " << dt[i] ;
        cout << "\n Chu vi= " << cv[i] ;
    }
    getch();
```

```
}
void dtcvhcn(int a, int b, int&dt, int &cv)
{
	dt=a*b;
	cv=2*(a+b);
}
```

CHƯƠNG 8

HÀM BẠN, ĐỊNH NGHĨA PHÉP TOÁN CHO LỚP

Hàm bạn
Định nghĩa phép toán cho lớp

I. HÀM BẠN (FRIEND FUNCTION)

1. Hàm bạn

Để một hàm trở thành bạn của một lớp, có 2 cách viết:

Cách 1: Dùng từ khóa friend để khai báo hàm trong lớp và xây dựng hàm bên ngoài như các hàm thông thường (không dùng từ khóa friend). Mẫu viết như sau:

```
class A
{
private:
        // Khai báo các thuộc tính
public:
        ...
        // Khai báo các hàm bạn của lớp A
        friend void f1(...);
        friend double f2(...);
        friend A f3(...) ;
        ...
} ;
// Xây dựng các hàm f1, f2, f3
void f1(...)
{
        ...
}
double f2(...)
{
```

```
        ...
}
A f3(...)
{
        ...
}
```

Cách 2: Dùng từ khóa friend để xây dựng hàm trong định nghĩa lớp. Mẫu viết như sau:

```
class A
{
private:
        // Khai báo các thuộc tính
public:
        // Xây dựng các hàm bạn của lớp A
        void f1(...)
        {
                ...
        }
        double f2(...)
        {
                ...
        }
        A f3(...)
        {
                ...
        }
        ...
} ;
```

2. Tính chất của hàm bạn

Trong thân hàm bạn của một lớp có thể truy nhập tới các thuộc tính của các đối tượng thuộc lớp này. Đây là sự khác nhau duy nhất giữa hàm bạn và hàm thông thường.

Chú ý rằng hàm bạn không phải là phương thức của lớp. Phương thức có một

đối ẩn (ứng với con trỏ this) và lời gọi của phương thức phải gắn với một đối tượng nào đó (địa chỉ đối tượng này được truyền cho con trỏ this). Lời gọi của hàm bạn giống như lời gọi của hàm thông thường.

Ví dụ sau sẽ so sánh phương thức, hàm bạn và hàm thông thường.

Xét lớp SP (số phức), hãy so sánh 3 phương án để thực hiện việc cộng 2 số phức:

Phương án 1: Dùng phương thức

```
class SP
{
private:
    double a; // phần thực
    double b; // Phần ảo
public:
    SP cong(SP u2)
    {
        SP u:
        u.a = this → a + u2.a ;
        u.b = this → b + u2.b ;
        return u;
    }
};
```

Cách dùng:

```
SP u, u1, u2;
u = u1.cong(u2);
```

Phương án 2: Dùng hàm bạn

```
class SP
{
private:
    double a; // Phần thực
    double b; // Phần ảo
public:
    friend SP cong(SP u1 , SP u2)
```

```
    {
        SP u:
        u.a = u1.a + u2.a ;
        u.b = u1.b + u2.b ;
        return u;
    }
};
```

Cách dùng

```
SP u, u1, u2;
u = cong(u1, u2);
```

Phương án 3: Dùng hàm thông thường

```
class SP
{
private:
    double a; // phần thực
    double b; // Phần ảo
public:
    ...
};

SP cong(SP u1, SP u2)
{
    SP u:
    u.a = u1.a + u2.a ;
    u.b = u1.b + u2.b ;
    return u;
}
```

Phương án này không được chấp nhận, trình biên dịch sẽ báo lỗi trong thân hàm không được quyền truy xuất đến các thuộc tính riêng (private) a, b của các đối tượng u, u1 và u2 thuộc lớp SP.

3. Hàm bạn của nhiều lớp

Khi một hàm là bạn của nhiều lớp, thì nó có quyền truy nhập tới tất cả các thuộc tính của các đối tượng trong các lớp này.

Để làm cho hàm f trở thành bạn của các lớp A, B và C ta sử dụng mẫu viết như sau:

```
class A;          // Khai báo trước lớp A
class B;          // Khai báo trước lớp B
class C;          // Khai báo trước lớp C
// Định nghĩa lớp A
class A
{
      // Khai báo f là bạn của A
      friend void f(...) ;
} ;
// Định nghĩa lớp B
class B
{
      // Khai báo f là bạn của B
      friend void f(...) ;
} ;
// Định nghĩa lớp C
class C
{
      // Khai báo f là bạn của C
      friend void f(...) ;
} ;
// Xây dựng hàm f
void f(...)
{
      ...
}
```

Chương trình sau đây minh họa cách dùng hàm bạn (bạn của một lớp và bạn của nhiều lớp). Chương trình đưa vào 2 lớp VT (véc tơ), MT (ma trận) và 3 hàm bạn để thực hiện các thao tác trên 2 lớp này:

```
// Hàm bạn với lớp VT dùng để in một véc tơ
      friend void in(const VT &x);
// Hàm bạn với lớp MT dùng để in một ma trận
      friend void in(const MT &a);
```

```
// Hàm bạn với cả 2 lớp MT và VT dùng để nhân ma trận với véc tơ
    friend VT tich(const MT &a, const VT &x);
```

Nội dung chương trình là nhập một ma trận vuông cấp n và một véc tơ cấp n, sau đó thực hiện phép nhân ma trận với véc tơ vừa nhập.

```
#include <conio.h>
#include <iostream.h>
#include <math.h>
class VT;
class MT;
class VT
{
private:
    int n;
    double x[20]; // Toa do cua diem
public:
    void nhapsl();
    friend void in(const VT &x);
    friend VT tich(const MT &a, const VT &x) ;
};

class MT
{
private:
    int n;
    double a[20][20];
public:
    friend VT tich(const MT &a, const VT &x);
    friend void in(const MT &a);
    void nhapsl();
} ;

void VT::nhapsl()
{
    cout << "\n Cap vec to = ";
```

```
        cin >> n ;
        for (int i = 1; i< = n ; ++i)
        {
                cout << "\n Phan tu thu " << i <<" = " ;
                cin >> x[i];
        }
}

void MT::nhapsl()
{
        cout <<"\n Cap ma tran = ";
        cin >> n ;
        for (int i = 1; i< = n ; ++i)
        for (int j = 1; j< = n; ++j)
        {
                cout << "\n Phan tu thu: "<<i<< " hang "<< i << " cot " << j << " = ";
                cin >> a[i][j];
        }
}

VT tich(const MT &a, const VT &x)
{
        VT y;
        int n = a.n;
        if (n! = x.n)
        return x;
        y.n = n;
        for (int i = 1; i< = n; ++i)
        {
                y.x[i] = 0;
                for (int j = 1; j< = n; ++j)
                y.x[i] = a.a[i][j]*x.x[j];
        }
        return y;
```

```
}
void in(const VT &x)
{
	cout << "\n";
	for (int i = 1; i< = x.n; ++i)
	cout << x.x[i] << " ";
}
void in(const MT &a)
{
	for (int i = 1; i< = a.n; ++i)
	{
		cout << "\n " ;
		for (int j = 1; j< = a.n; ++j)
		cout << a.a[i][j] << " ";
	}
}
void main()
{
	MT a; VT x, y;
	clrscr();
	a.nhapsl();
	x.nhapsl();
	y = tich(a, x);
	clrscr();
	cout << "\n Ma tran A:";
	in(a);
	cout << "\n Vec to x: " ;
	in(x);
	cout << "\n Vec to y = Ax: " ;
	in(y);
	getch();
}
```

II. ĐỊNH NGHĨA PHÉP TOÁN CHO LỚP

Đối với mỗi lớp ta có thể sử dụng lại các kí hiệu phép toán thông dụng (+, -, *, …) để định nghĩa cho các phép toán của lớp. Sau khi được định nghĩa các kí hiệu này sẽ được dùng như các phép toán của lớp theo cách viết thông thường. Cách định nghĩa này được gọi là phép chồng toán tử (như khái niệm chồng hàm trong các chương trước).

1. Tên hàm toán tử

Gồm từ khoá operator và tên phép toán.

Ví dụ:

operator+(định nghĩa chồng phép +)

operator- (định nghĩa chồng phép -)

2. Các đối của hàm toán tử

– Với các phép toán có 2 toán hạng thì hàm toán tử cần có 2 đối. Đối thứ nhất ứng với toán hạng thứ nhất, đối thứ hai ứng với toán hạng thứ hai. Do vậy, với các phép toán không giao hoán (phép -) thì thứ tự đối là rất quan trọng.

Ví dụ: Các hàm toán tử cộng, trừ phân số được khai báo như sau:

```
struct PS
{
    int a;      //Tử số
    int b;      // Mẫu số
};
PS operator+(PS p1, PS p2);          // p1 + p2
PS operator-(PS p1 , PS p2);         // p1 - p2
PS operator*(PS p1, PS p2);          // p1 *p2
PS operator/(PS p1, PS p2);          // p1/p2
```

– Với các phép toán có một toán hạng, thì hàm toán tử có một đối. Ví dụ hàm toán tử đổi dấu ma trận (đổi dấu tất cả các phần tử của ma trận) được khai báo như sau:

```
struct MT
{
    double a[20][20] ;    // Mảng chứa các phần tử ma trận
    int m ;               // Số hàng ma trận
```

```
    int n ;                    // Số cột ma trận
};
MT operator-(MT x) ;
```

3. Thân của hàm toán tử

Viết như thân của hàm thông thường. Ví dụ hàm đổi dấu ma trận có thể được định nghĩa như sau:

```
struct MT
{
    double a[20][20] ;         // Mảng chứa các phần tử ma trận
    int m ;                    // Số hàng ma trận
    int n ;                    // Số cột ma trận
};
MT operator-(MT x)
{
    MT y;
    for (int i=1 ;i<= y.m ; ++i)
    for (int j =1 ;j<= y.n ; ++j)y.a[i][j] =- x.a[i][j];
    return y;
}
```

a. Cách dùng hàm toán tử

Có 2 cách dùng:

Cách 1: Dùng như một hàm thông thường bằng cách viết lời gọi

Ví dụ:

```
PS p, q, u, v ;
u = operator+(p, q) ;          // u = p + q
v = operator-(p, q) ;          // v= p - q
```

Cách 2: Dùng như phép toán của C++

Ví dụ:

```
PS p, q, u, v ;
u = p + q ;                    // u = p + q
v = p - q ;                    //v = p - q
```

Chú ý: Khi dùng các hàm toán tử như phép toán của C++ ta có thể kết hợp nhiều

phép toán để viết các công thức phức tạp. Cũng cho phép dùng dấu ngoặc tròn để quy định thứ tự thực hiện các phép tính. Thứ tự ưu tiên của các phép tính vẫn tuân theo các quy tắc ban đầu của C++. Chẳng hạn các phép * và / có thứ tự ưu tiên cao hơn so với các phép + và -

b. Các ví dụ về định nghĩa chồng toán tử

Ví dụ 1 : Trong ví dụ này ngoài việc sử dụng các hàm toán tử để thực hiện 4 phép tính trên phân số, còn định nghĩa chồng các phép toán << và >> để xuất và nhập phân số.

Hàm operator<< có 2 đối kiểu ostream& và PS (Phân số). Hàm trả về giá trị kiểu ostream& và được khai báo như sau:

```
ostream& operator<< (ostream& os, PS p);
```

Tượng tự hàm operator>> được khai báo như sau:

```
istream& operator>> (istream& is,PS &p);
```

Dưới đây sẽ chỉ ra cách xây dựng và sử dụng các hàm toán tử.

Chúng ta cũng sẽ thấy việc sử dụng các hàm toán tử rất tự nhiên, ngắn gọn và tiện lợi.

```
#include <conio.h>
#include <iostream.h>
#include <math.h>
typedef struct
{
      int a,b;
} PS;
ostream& operator<< (ostream& os, PS p);
istream& operator>> (istream& is,PS &p);
int uscln(int x, int y);
PS rutgon(PS p);
PS operator+(PS p1, PS p2);
PS operator-(PS p1, PS p2);
PS operator*(PS p1, PS p2);
PS operator/(PS p1, PS p2);
ostream& operator<< (ostream& os, PS p)
{
      os << p.a << '/' << p.b ;
```

```
        return os;
}
istream& operator>> (istream& is,PS &p)
{
        cout << "\n Nhap tu va mau: " ;
        is >> p.a >> p.b ;
        return is;
}
int uscln(int x, int y)
{
        x=abs(x);y=abs(y);
        if (x*y==0) return 1;
        while (x!=y)
        {
                if (x>y) x-=y;
                else y-=x;
        }
        return x;
}
PS rutgon(PS p)
{
        PS q;
        int x;
        x=uscln(p.a,p.b);
        q.a = p.a / x ;
        q.b = p.b/ x ;
        return q;
}
PS operator+(PS p1, PS p2)
{
        PS q;
        q.a = p1.a*p2.b + p2.a*p1.b;
        q.b = p1 .b * p2.b ;
```

```
        return rutgon(q);
}
PS operator-(PS p1, PS p2)
{
        PS q;
        q.a = p1.a*p2.b - p2.a*p1 .b;
        q.b = p1.b * p2.b ;
        return rutgon(q);
}
PS operator*(PS p1, PS p2)
{
        PS q;
        q.a = p1.a * p2.a ;
        q.b = p1.b * p2.b ;
        return rutgon(q);
}
PS operator/(PS p1 , PS p2)
{
        PS q;
        q.a = p1.a * p2.b ;
        q.b = p1.b * p2.a ;
        return rutgon(q);
}
void main()
{
        PS p, q, z, u, v ;
        PS s;
        cout <<"\nNhap cac PS p, q, z, u, v: " ;
        cin >> p >> q >> z >> u >> v ;
        s = (p - q*z) / (u + v) ;
        cout << "\n Phan so s = " << s;
        getch();
}
```

Ví dụ 2 : Chương trình đưa vào các hàm toán tử:

operator- có một đối dùng để đảo dấu một đa thức

operator+ có 2 đối dùng để cộng 2 đa thức

operator- có 2 đối dùng để trừ 2 đa thức

operator* có 2 đối dùng để nhân 2 đa thức

operator^có 2 đối dùng để tính giá đa thức tại x

operator<< có 2 đối dùng để in đa thức

operator>> có 2 đối dùng để nhập đa thức

Chương trình sẽ nhập 4 đa thức: p, q, r, s. Sau đó tính đa thức: f = -(p+q)*(r-s)
Cuối cùng tính giá trị f(x), với x là một số thực nhập từ bàn phím.

```
#include <conio.h>
#include <iostream.h>
#include <math.h>
struct DT
{
	double a[20];// Mang chua cac he so da thuc a0, a1,...
	int n ;// Bac da thuc
};
ostream& operator<< (ostream& os, DT d);
istream& operator>> (istream& is, DT &d);
DT operator-(const DT& d);
DT operator+(DT d1, DT d2);
DT operator-(DT d1, DT d2);
DT operator*(DT d1, DT d2);
double operator^(DT d, double x);// Tinh gia tri da thuc
ostream& operator<< (ostream& os, DT d)
{
	os << " Cac he so (tu ao): " ;
	for (int i=0 ;i<= d.n ;++i)
	os << d.a[i] <<" " ;
```

```
        return os;
}
istream& operator>> (istream& is, DT &d)
{
        cout << " Bac da thuc: " ;
        cin >> d.n;
        cout << "Nhap cac he so da thuc:" ;
        for (int i=0 ;i<=d.n ;++i)
        {
                cout << "\n He so bac " << i <<" = " ;
                is >> d.a[i] ;
        }
        return is;
}
DT operator-(const DT& d)
{
        DT p;
        p.n = d.n;
        for (int i=0 ;i<=d.n ;++i)
        p.a[i] = -d.a[i];
        return p;
}
DT operator+(DT d1, DT d2)
{
        DT d;
        int k,i;
        k = d1.n > d2.n ? d1.n : d2.n ;
        for (i=0;i<=k ;++i)
                if (i<=d1.n && i<=d2.n) d.a[i] = d1.a[i] + d2.a[i];
                else  if (i<=d1.n) d.a[i] = d1.a[i];
                        else d.a[i] = d2.a[i];
        i = k;
        while (i>0 && d.a[i]==0.0) --i;
```

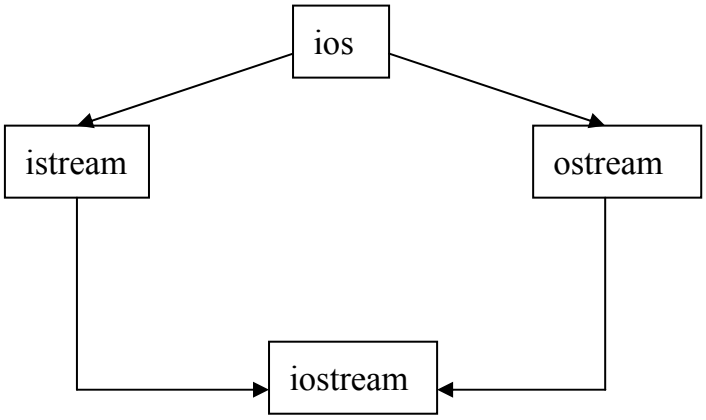
```
        d.n=i;
        return d ;
}
DT operator-(DT d1, DT d2)
{
        return (d1 + (-d2));
}
DT operator*(DT d1 , DT d2)
{
        DT d;
        int k, i, j;
        k = d.n = d1.n + d2.n ;
        for (i=0;i<=k;++i) d.a[i] = 0;
        for (i=0 ;i<= d1 .n ;++i)
        for (j=0 ;j<= d2.n ;++j)
        d.a[i+j] += d1 .a[i]*d2.a[j];
        return d;
}
double operator^(DT d, double x)
{
        double s=0.0 , t=1.0;
        for (int i=0 ;i<= d.n ;++i)
        {
                s += d.a[i]*t;
                t *= x;
        }
        return s;
}
void main()
{
        DT p,q,r,s,f;
        double x,g;
        clrscr();
```

```
        cout <<"\n Nhap da thuc P " ;cin >> p;
        cout <<"\n Nhap da thuc Q " ;cin >> q;
        cout <<"\n Nhap da thuc R " ;cin >> r;
        cout <<"\n Nhap da thuc S " ;cin >> s;
        cout << "\n Nhap so thuc x: " ;cin >> x;
        f = -(p+q)*(r-s);
        g = f^x;
        cout << "\n Da thuc f "<< f ;
        cout << "\n x = " << x;
        cout << "\n f(x) = " << g;
        getch();
}
```

CHƯƠNG 9
CÁC DÒNG NHẬP/XUẤT VÀ FILE

Nhập/xuất với cin/cout
Định dạng
In ra máy in
Làm việc với File
Nhập/xuất nhị phân

Trong C++ có sẵn một số lớp chuẩn chứa dữ liệu và các phương thức phục vụ cho các thao tác nhập/xuất dữ liệu của NSD, thường được gọi chung là ***stream*** (dòng). Trong số các lớp này, lớp có tên **ios** là lớp cơ sở, chứa các thuộc tính để định dạng việc nhập/xuất và kiểm tra lỗi. Mở rộng (kế thừa) lớp này có các lớp **istream, ostream** cung cấp thêm các toán tử nhập/xuất như >>, << và các hàm get, getline, read, ignore, put, write, flush … Một lớp rộng hơn có tên **iostream** là tổng hợp của 2 lớp trên. Bốn lớp nhập/xuất cơ bản này được khai báo trong các file tiêu đề có tên tương ứng (với đuôi *.h). Sơ đồ thừa kế của 4 lớp trên được thể hiện qua hình vẽ dưới đây.



Đối tượng của các lớp trên được gọi là các *dòng* dữ liệu. Một số đối tượng thuộc lớp iostream đã được khai báo sẵn (*chuẩn*) và được gắn với những thiết bị nhập/xuất cố định như các đối tượng **cin, cout, cerr, clog** gắn với bàn phím (cin) và màn hình (cout, cerr, clog). Điều này có nghĩa các toán tử >>, << và các hàm kể trên khi làm việc với các đối tượng này sẽ cho phép NSD nhập dữ liệu thông qua bàn phím hoặc xuất kết quả thông qua màn hình.

Để nhập/xuất thông qua các thiết bị khác (như máy in, file trên đĩa …), C++

cung cấp thêm các lớp **ifstream, ofstream, fstream** cho phép NSD khai báo các đối tượng mới gắn với thiết bị và từ đó nhập/xuất thông qua các thiết bị này.

Trong chương này, chúng ta sẽ xét các đối tượng chuẩn **cin, cout** và một số toán tử, hàm nhập xuất đặc trưng của lớp **iostream** cũng như cách tạo và sử dụng các đối tượng thuộc các lớp **ifstream, ofstream, fstream** để làm việc với các thiết bị như máy in và file trên đĩa.

I. NHẬP/XUẤT VỚI CIN/COUT

Như đã nhắc ở trên, **cin** là dòng dữ liệu nhập (đối tượng) thuộc lớp *istream.* Các thao tác trên đối tượng này gồm có các toán tử và hàm phục vụ nhập dữ liệu vào cho biến từ bàn phím.

1. Toán tử nhập >>

Toán tử này cho phép nhập dữ liệu từ một dòng Input_stream nào đó vào cho một danh sách các biến. Cú pháp chung như sau:

Input_stream >> biến1 >> biến2 >> …

trong đó Input_stream là đối tượng thuộc lớp istream. Trường hợp Input_stream là cin, câu lệnh nhập sẽ được viết:

cin >> biến1 >> biến2 >> …

câu lệnh này cho phép nhập dữ liệu từ bàn phím cho các biến. Các biến này có thể thuộc các kiểu chuẩn như : kiểu nguyên, thực, ký tự, xâu kí tự. Chú ý 2 đặc điểm quan trọng của câu lệnh trên.

- Lệnh sẽ bỏ qua không gán các dấu trắng (dấu cách <>, dấu Tab, dấu xuống dòng ↵) vào cho các biến (kể cả biến xâu kí tự).
- Khi NSD nhập vào dãy byte nhiều hơn cần thiết để gán cho các biến thì số byte còn lại và kể cả dấu xuống dòng ↵ sẽ nằm lại trong cin. Các byte này sẽ tự động gán cho các biến trong lần nhập sau mà không chờ NSD gõ thêm dữ liệu vào từ bàn phím. Do vậy câu lệnh

```
cin >> a >> b >> c;
```

cũng có thể được viết thành

```
cin >> a;
cin >> b;
cin >> c;
```

và chỉ cần nhập dữ liệu vào từ bàn phím một lần chung cho cả 3 lệnh (mỗi dữ liệu nhập cho mỗi biến phải cách nhau ít nhất một dấu trắng)

Ví dụ 1 : Nhập dữ liệu cho các biến

```
int a;
float b;
char c;
char *s;
cin >> a >> b >> c >> s;
```

giả sử NSD nhập vào dãy dữ liệu : <><>12<>34.517ABC<>12E<>D ↵

khi đó các biến sẽ được nhận những giá trị cụ thể sau:

a = 12

b = 34.517

c = 'A'

s = "BC"

trong cin sẽ còn lại dãy dữ liệu : <>12E<>D ↵.

Nếu trong đoạn chương trình tiếp theo có câu lệnh cin >> s; thì s sẽ được tự động gán giá trị "12E" mà không cần NSD nhập thêm dữ liệu vào cho cin.

Qua ví dụ trên một lần nữa ta nhắc lại đặc điểm của toán tử nhập >> là các biến chỉ lấy dữ liệu vừa đủ cho kiểu của biến (ví dụ biến c chỉ lấy một kí tự 'A', b lấy giá trị 34.517) hoặc cho đến khi gặp dấu trắng đầu tiên (ví dụ a lấy giá trị 12, s lấy giá trị "BC" dù trong cin vẫn còn dữ liệu). Từ đó ta thấy toán tử >> là không phù hợp khi nhập dữ liệu cho các xâu kí tự có chứa dấu cách. C++ giải quyết trường hợp này bằng một số hàm (phương thức) nhập khác thay cho toán tử >>.

2. Các hàm nhập kí tự và xâu kí tự

a. Nhập kí tự

- **cin.get()** : Hàm trả lại một kí tự (kể cả dấu cách, dấu ↵).. Ví dụ:

  ```
  char ch;
  ch = cin.get();
  ```

  - nếu nhập AB↵, ch nhận giá trị 'A', trong cin còn B↵.
  - nếu nhập A↵, ch nhận giá trị 'A', trong cin còn ↵.
  - nếu nhập ↵, ch nhận giá trị '↵', trong cin rỗng.

- **cin.get(ch)** : Hàm nhập kí tự cho ch và trả lại một tham chiếu tới cin. Do hàm trả lại tham chiếu tới cin nên có thể viết các phương thức nhập này liên tiếp trên một đối tượng cin. Ví dụ:

  ```
  char c, d;
  cin.get(c).get(d);
  ```

nếu nhập AB↵ thì c nhận giá trị 'A' và d nhận giá trị 'B'. Trong cin còn 'C↵'.

b. Nhập xâu kí tự

- **cin.get(s, n, fchar)** : Hàm nhập cho s dãy kí tự từ cin. Dãy được tính từ kí tự đầu tiên trong cin cho đến khi đã đủ n – 1 kí tự hoặc gặp kí tự kết thúc fchar. Kí tự kết thúc này được ngầm định là dấu xuống dòng nếu bị bỏ qua trong danh sách đối. Tức có thể viết câu lệnh trên dưới dạng **cin.get(s, n)** khi đó xâu s sẽ nhận dãy kí tự nhập cho đến khi đủ n-1 kí tự hoặc đến khi NSD kết thúc nhập (bằng dấu ↵).

Chú ý :

– Lệnh sẽ tự động gán dấu kết thúc xâu ('\0') vào cho xâu s sau khi nhập xong.

– Các lệnh có thể viết nối nhau, ví dụ: **cin.get(s1, n1).get(s2,n2);**

– Kí tự kết thúc fchar (hoặc ↵) vẫn nằm lại trong cin. Điều này có thể làm trôi các lệnh get() tiếp theo. Ví dụ:

```
struct Sinhvien {
        char *ht;                   // họ tên
        char *qq;                   // quê quán
};
void main()
{
        int i;
        for (i=1; i<=3; i++) {
                cout << "Nhap ho ten sv thu " << i; cin.get(sv[i].ht, 25);
                cout << "Nhap que quan sv thu "<< i; cin.get(sv[i].qq, 30);
        }
        …
}
```

Trong đoạn lệnh trên sau khi nhập họ tên của sinh viên thứ 1, do kí tự ↵ vẫn nằm trong bộ đệm nên khi nhập quê quán chương trình sẽ lấy kí tự ↵ này gán cho qq, do đó quê quán của sinh viên sẽ là xâu rỗng.

Để khắc phục tình trạng này chúng ta có thể sử dụng một trong các câu lệnh nhập kí tự để "nhấc" dấu enter còn "rơi vãi" ra khỏi bộ đệm. Có thể sử dụng các câu lệnh sau :

```
cin.get();                  // đọc một kí tự trong bộ đệm
cin.ignore(n);              //đọc n kí tự trong bộ đệm (với n=1)
```

như vậy để đoạn chương trình trên hoạt động tốt ta có thể tổ chức lại như sau:

```
void main()
{
    int i;
    for (i=1; i<=3; i++) {
        cout << "Nhap ho ten sv thu " << i; cin.get(sv[i].ht, 25);
    cin.get();                    // nhấc 1 kí tự (enter)
    cout << "Nhap que quan sv thu "<< i; cin.get(sv[i].qq, 30);
    cin.get()                     // hoặc cin.ignore(1);
}
…
}
```

- **cin.getline(s, n, fchar):** Phương thức này hoạt động hoàn toàn tương tự phương thức **cin.get(s, n, fchar),** tuy nhiên nó có thể khắc phục "lỗi enter" của câu lệnh trên. Cụ thể hàm sau khi gán nội dung nhập cho biến s sẽ xóa kí tự enter khỏi bộ đệm và do vậy NSD không cần phải sử dụng thêm các câu lệnh phụ trợ (cin.get(), cin.ignore(1)) để loại enter ra khỏi bộ đệm.
- **cin.ignore(n):** Phương thức này của đối tượng cin dùng để đọc và loại bỏ n kí tự còn trong bộ đệm (dòng nhập cin).

Chú ý: Toán tử nhập >> cũng giống các phương thức nhập kí tự và xâu kí tự ở chỗ cũng để lại kí tự enter trong cin. Do vậy, chúng ta nên sử dụng các phương thức cin.get(), cin.ignore(n) để loại bỏ kí tự enter trước khi thực hiện lệnh nhập kí tự và xâu kí tự khác.

Tương tự dòng nhập cin, **cout** là dòng dữ liệu xuất thuộc lớp *ostream.* Điều này có nghĩa dữ liệu làm việc với các thao tác xuất (in) sẽ đưa kết quả ra cout mà đã được mặc định là màn hình. Do đó ta có thể sử dụng toán tử xuất << và các phương thức xuất trong các lớp ios (lớp cơ sở) và ostream.

3. Toán tử xuất <<

Toán tử này cho phép xuất giá trị của dãy các biểu thức đến một dòng Output_stream nào đó với cú pháp chung như sau:

Output_stream << bt_1 << bt_2 << …

ở đây Output_stream là đối tượng thuộc lớp ostream. Trường hợp Output_stream là cout, câu lệnh xuất sẽ được viết:

cout << bt_1 << bt_2 << …

câu lệnh này cho phép in kết quả của các biểu thức ra màn hình. Kiểu dữ liệu của

các biểu thức có thể là số nguyên, thực, kí tự hoặc xâu kí tự.

II. ĐỊNH DẠNG

Các giá trị in ra màn hình có thể được trình bày dưới nhiều dạng khác nhau thông qua các công cụ định dạng như các phương thức, các cờ và các bộ phận khác được khai báo sẵn trong các lớp ios và ostream.

1. Các phương thức định dạng

a. Chỉ định độ rộng cần in

cout.width(n) ;

Số cột trên màn hình để in một giá trị được ngầm định bằng với độ rộng thực (số chữ số, chữ cái và kí tự khác trong giá tị được in). Để đặt lại độ rộng màn hình dành cho giá trị cần in (thông thường lớn hơn độ rộng thực) ta có thể sử dụng phương thức trên.

Phương thức này cho phép các giá trị in ra màn hình với độ rộng n. Nếu n bé hơn độ rộng thực sự của giá trị thì máy sẽ in giá trị với số cột màn hình bằng với độ rộng thực. Nếu n lớn hơn độ rộng thực, máy sẽ in giá trị căn theo lề phải, và để trống các cột thừa phía trước giá trị được in. Phương thức này chỉ có tác dụng với giá trị cần in ngay sau nó. Ví dụ:

```
int a = 12; b = 345;          // độ rộng thực của a là 2, của b là 3
cout << a;                    // chiếm 2 cột màn hình
cout.width(7);                // đặt độ rộng giá trị in tiếp theo là 7
cout  << b;                   // b in trong 7 cột với 4 dấu cách đứng trước
```

Kết quả in ra sẽ là: 12<><><><>345

b. Chỉ định kí tự chèn vào khoảng trống trước giá trị cần in

cout.fill(ch) ;

Kí tự độn ngầm định là dấu cách, có nghĩa khi độ rộng của giá trị cần in bé hơn độ rộng chỉ định thì máy sẽ độn các dấu cách vào trước giá trị cần in cho đủ với độ rộng chỉ định. Có thể yêu cầu độn một kí tự ch bất kỳ thay cho dấu cách bằng phương thức trên. Ví dụ trong dãy lệnh trên, nếu ta thêm dòng lệnh cout.fill('*') trước khi in b chẳng hạn thì kết quả in ra sẽ là: 12****345.

Phương thức này có tác dụng với mọi câu lệnh in sau nó cho đến khi gặp một chỉ định mới.

c. Chỉ định độ chính xác (số số lẻ thập phân) cần in

cout.precision(n) ;

Phương thức này yêu cầu các số thực in ra sau đó sẽ có n chữ số lẻ. Các số

thực trước khi in ra sẽ được làm tròn đến chữ số lẻ thứ n. Chỉ định này có tác dụng cho đến khi gặp một chỉ định mới. Ví dụ:

```
int a = 12.3; b = 345.678;          // độ rộng thực của a là 4, của b là 7
cout << a;                          // chiếm 4 cột màn hình
cout.width(10);                     // đặt độ rộng giá trị in tiếp theo là 10
cout.precision(2);                  // đặt độ chính xác đến 2 số lẻ
cout  << b;                  // b in trong 10 cột với 4 dấu cách đứng trước
```

Kết quả in ra sẽ là: 12.3<><><><>345.68

2. Các cờ định dạng

Một số các qui định về định dạng thường được gắn liền với các "cờ". Thông thường nếu định dạng này được sử dụng trong suốt quá trình chạy chương trình hoặc trong một khoảng thời gian dài trước khi gỡ bỏ thì ta "bật" các cờ tương ứng với nó. Các cờ được bật sẽ có tác dụng cho đến khi cờ với định dạng khác được bật. Các cờ được cho trong file tiêu đề iostream.h.

Để bật/tắt các cờ ta sử dụng các phương thức sau:

```
cout.setf(danh sách cờ);      // Bật các cờ trong danh sách
cout.unsetf(danh sách cờ);    // Tắt các cờ trong danh sách
```

Các cờ trong danh sách được viết cách nhau bởi phép toán hợp bit (|). Ví dụ lệnh cout.setf(ios::left | ios::scientific) sẽ bật các cờ ios::left và ios::scientific. Phương thức cout.unsetf(ios::right | ios::fixed) sẽ tắt các cờ ios::right | ios::fixed.

Dưới đây là danh sách các cờ cho trong iostream.h.

a. Nhóm căn lề

- **ios::left** : nếu bật thì giá trị in nằm bên trái vùng in ra (kí tự độn nằm sau).
- **ios::right** : giá trị in nằm bên phái vùng in ra (kí tự độn nằm trước), đây là trường hợp ngầm định nếu ta không sử dụng cờ cụ thể.
- **ios::internal** : giống cờ ios::right tuy nhiên dấu của giá trị in ra sẽ được in đầu tiên, sau đó mới đến kí tự độn và giá trị số.

Ví dụ:

```
int a = 12.3; b = –345.678;     // độ rộng thực của a là 4, của b là 8
cout << a;                      // chiếm 4 cột màn hình
cout.width(10);                 // đặt độ rộng giá trị in tiếp theo là 10
cout.fill('*') ;                // dấu * làm kí tự độn
cout.precision(2);              // đặt độ chính xác đến 2 số lẻ
cout.setf(ios::left) ;          // bật cờ ios::left
```

```
cout << b;                    // kết qủa: 12.3–345.68***
cout.setf(ios::right) ;       // bật cờ ios::right
cout << b;                    // kết qủa: 12.3***–345.68
cout.setf(ios::internal) ;    // bật cờ ios::internal
cout << b;                    // kết qủa: 12.3–***345.68
```

b. Nhóm định dạng số nguyên

- **ios::dec** : in số nguyên dưới dạng thập phân (ngầm định)
- **ios::oct** : in số nguyên dưới dạng cơ số 8
- **ios::hex** : in số nguyên dưới dạng cơ số 16

c. Nhóm định dạng số thực

- **ios::fixed** : in số thực dạng dấu phảy tĩnh (ngầm định)
- **ios::scientific** : in số thực dạng dấu phảy động
- **ios::showpoint** : in đủ n chữ số lẻ của phần thập phân, nếu tắt (ngầm định) thì không in các số 0 cuối của phần thập phân.

Ví dụ: giả sử độ chính xác được đặt với 3 số lẻ (bởi câu lệnh cout.precision(3))

- nếu fixed bật + showpoint bật :

  123.2500 được in thành 123.250
  123.2599 được in thành 123.260
  123.2 được in thành 123.200

- nếu fixed bật + showpoint tắt :

  123.2500 được in thành 123.25
  123.2599 được in thành 123.26
  123.2 được in thành 123.2

- nếu scientific bật + showpoint bật :

  12.3 được in thành 1.230e+01
  2.32599 được in thành 2.326e+00
  324 được in thành 3.240e+02

- nếu scientific bật + showpoint tắt :

  12.3 được in thành 1.23e+01
  2.32599 được in thành 2.326e+00
  324 được in thành 3.24e+02

d. Nhóm định dạng hiển thị

- **ios::showpos** : nếu tắt (ngầm định) thì không in dấu cộng (+) trước số dương. Nếu bật trước mỗi số dương sẽ in thêm dấu cộng.
- **ios::showbase** : nếu bật sẽ in số 0 trước các số nguyên hệ 8 và in 0x trước số hệ 16. Nếu tắt (ngầm định) sẽ không in 0 và 0x.
- **ios::uppercase** : nếu bật thì các kí tự biểu diễn số trong hệ 16 (A..F) sẽ viết hoa, nếu tắt (ngầm định) sẽ viết thường.

3. Các bộ và hàm định dạng

iostream.h cũng cung cấp một số bộ và hàm định dạng cho phép sử dụng tiện lợi hơn so với các cờ và các phương thức vì nó có thể được viết liên tiếp trên dòng lệnh xuất.

a. Các bộ định dạng

```
dec                 // tương tự ios::dec
oct                 // tương tự ios::dec
hex                 // tương tự ios::hex
endl                // xuất kí tự xuống dòng ('\n')
flush               // đẩy toàn bộ dữ liệu ra dòng xuất
```

Ví dụ :

```
cout.setf(ios::showbase) ;          // cho phép in các kí tự biểu thị cơ số
cout.setf(ios::uppercase) ;         // dưới dạng chữ viết hoa
int a = 171; int b = 32 ;
cout << hex << a << endl << b ;     // in 0xAB và 0x20
```

b. Các hàm định dạng (#include <iomanip.h>)

```
setw(n)             // tương tự cout.width(n)
setprecision(n)     // tương tự cout.precision(n)
setfill(c)          // tương tự cout.fill(c)
setiosflags(l)      // tương tự cout.setf(l)
resetiosflags(l)    // tương tự cout.unsetf(l)
```

III. IN RA MÁY IN

Như trong phần đầu chương đã trình bày, để làm việc với các thiết bị khác với màn hình và đĩa … chúng ta cần tạo ra các đối tượng (thuộc các lớp ifstream, ofstream và fstream) tức các dòng tin bằng các hàm tạo của lớp và gắn chúng với

thiết bị bằng câu lệnh:

ofstream Tên_dòng(thiết bị) ;

Ví dụ để tạo một đối tượng mang tên Mayin và gắn với máy in, chúng ta dùng lệnh:

ofstream Mayin(4) ;

trong đó 4 là số hiệu của máy in.

Khi đó mọi câu lệnh dùng toán tử xuất << và cho ra Mayin sẽ đưa dữ liệu cần in vào một bộ đệm mặc định trong bộ nhớ. Nếu bộ đệm đầy, một số thông tin đưa vào trước sẽ tự động chuyển ra máy in. Để chủ động đưa tất cả dữ liệu còn lại trong bộ đệm ra máy in chúng ta cần sử dụng bộ định dạng flush (Mayin << flush << …) hoặc phương thức flush (Mayin.flush();). Ví dụ:

Sau khi đã khai báo một đối tượng mang tên Mayin bằng câu lệnh như trên Để in chu vi và diện tích hình chữ nhật có cạnh cd và cr ta có thể viết:

```
Mayin << "Diện tích HCN = " << cd * cr << endl;
Mayin << "Chu vi HCN = " << 2*(cd + cr) << endl;
Mayin.flush();
```

hoặc :

```
Mayin << "Diện tích HCN = " << cd * cr << endl;
Mayin << "Chu vi HCN = " << 2*(cd + cr) << endl << flush;
```

khi chương trình kết thúc mọi dữ liệu còn lại trong các đối tượng sẽ được tự động chuyển ra thiết bị gắn với nó. Ví dụ máy in sẽ in tất cả mọi dữ liệu còn sót lại trong Mayin khi chương trình kết thúc.

IV. LÀM VIỆC VỚI FILE

Làm việc với một file trên đĩa cũng được quan niệm như làm việc với các thiết bị khác của máy tính (ví dụ như làm việc với máy in với đối tượng Mayin trong phần trên hoặc làm việc với màn hình với đối tượng chuẩn cout). Các đối tượng này được khai báo thuộc lớp ifstream hay ofstream tùy thuộc ta muốn sử dụng file để đọc hay ghi.

Như vậy, để sử dụng một file dữ liệu đầu tiên chúng ta cần tạo đối tượng và gắn cho file này. Để tạo đối tượng có thể sử dụng các hàm tạo có sẵn trong hai lớp ifstream và ofstream. Đối tượng sẽ được gắn với tên file cụ thể trên đĩa ngay trong quá trình tạo đối tượng (tạo đối tượng với tham số là tên file) hoặc cũng có thể được gắn với tên file sau này bằng câu lệnh mở file. Sau khi đã gắn một đối tượng với file trên đĩa, có thể sử dụng đối tượng như đối với Mayin hoặc cin, cout. Điều này có nghĩa trong các câu lệnh in ra màn hình chỉ cần thay từ khóa cout bởi tên đối tượng mọi dữ liệu cần in trong câu lệnh sẽ được ghi lên file mà đối tượng đại diện. Cũng tương tự nếu thay cin bởi tên đối tượng, dữ liệu sẽ được đọc vào từ file thay cho từ

bàn phím. Để tạo đối tượng dùng cho việc ghi ta khai báo chúng với lớp *ofstream* còn để dùng cho việc đọc ta khai báo chúng với lớp *ifstream*.

1. Tạo đối tượng gắn với file

Mỗi lớp ifstream và ofstream cung cấp 4 phương thức để tạo file. Ở đây chúng tôi chỉ trình bày 2 cách (2 phương thức) hay dùng.

+ Cách 1: **<Lớp> đối_tượng;**

  đối_tượng.open(tên_file, chế_độ);

Lớp là một trong hai lớp ifstream và ofstream. Đối tượng là tên do NSD tự đặt. Chế độ là cách thức làm việc với file (xem dưới). Cách này cho phép tạo trước một đối tượng chưa gắn với file cụ thể nào. Sau đó dùng tiếp phương thức open để đồng thời mở file và gắn với đối tượng vừa tạo.

Ví dụ:

```
ifstream f;              // tạo đối tượng có tên f để đọc hoặc
ofstream f;              // tạo đối tượng có tên f để ghi
f.open("Baitap");        // mở file Baitap và gắn với f
```

+ Cách 2: **<Lớp> đối_tượng(tên_file, chế_độ)**

Cách này cho phép đồng thời mở file cụ thể và gắn file với tên đối tượng trong câu lệnh.

Ví dụ:

```
ifstream f("Baitap");    // mở file Baitap gắn với đối tượng f để
ofstream f("Baitap);     // đọc hoặc ghi.
```

Sau khi mở file và gắn với đối tượng f, mọi thao tác trên f cũng chính là làm việc với file Baitap.

Trong các câu lệnh trên có các chế độ để qui định cách thức làm việc của file. Các chế độ này gồm có:

- ios::binary : quan niệm file theo kiểu nhị phân. Ngầm định là kiểu văn bản.
- ios::in : file để đọc (ngầm định với đối tượng trong ifstream).
- ios::out : file để ghi (ngầm định với đối tượng trong ofstream), nếu file đã có trên đĩa thì nội dung của nó sẽ bị ghi đè (bị xóa).ios::app : bổ sung vào cuối file
- ios::trunc : xóa nội dung file đã có
- ios::ate : chuyển con trỏ đến cuối file
- ios::nocreate : không làm gì nếu file chưa có

- ios::replace : không làm gì nếu file đã có

có thể chỉ định cùng lúc nhiều chế độ bằng cách ghi chúng liên tiếp nhau với toán tử hợp bit |. Ví dụ để mở file bài tập như một file nhị phân và ghi tiếp theo vào cuối file ta dùng câu lệnh:

```
ofstream f("Baitap", ios::binary | ios::app);
```

2. Đóng file và giải phóng đối tượng

Để đóng file được đại diện bởi f, sử dụng phương thức close như sau:

đối_tượng.close();

Sau khi đóng file (và giải phóng mối liên kết giữa đối tượng và file) có thể dùng đối tượng để gắn và làm việc với file khác bằng phương thức open như trên.

Ví dụ 2 : Đọc một dãy số từ bàn phím và ghi lên file. File được xem như file văn bản (ngầm định), các số được ghi cách nhau 1 dấu cách.

```
#include <iostream.h>
#include <fstream.h>
#include <conio.h>
void main()
{
    ofstream f;                    // khai báo (tạo) đối tượng f
    int x;
    f.open("DAYSO");               // mở file DAYSO và gắn với f
    for (int i = 1; i<=10; i++) {
        cin >> x;
        f << x << ' ';
    }
    f.close();
}
```

Ví dụ 3 : Chương trình sau nhập danh sách sinh viên, ghi vào file 1, đọc ra mảng, sắp xếp theo tuổi và in ra file 2. Dòng đầu tiên trong file ghi số sinh viên, các dòng tiếp theo ghi thông tin của sinh viên gồm họ tên với độ rộng 24 kí tự, tuổi với độ rộng 4 kí tự và điểm với độ rộng 8 kí tự.

```
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdlib.h>
```

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
struct Sv {
	char *hoten;
	int tuoi;
	double diem;
};
class Sinhvien {
	int sosv ;
	Sv *sv;
public:
	Sinhvien() {
		sosv = 0;
		sv = NULL;
	}
	void nhap();
	void sapxep();
	void ghifile(char *fname);
};

void Sinhvien::nhap()
{
	cout << "\nSố sinh viên: "; cin >> sosv;
	int n = sosv;
	sv = new Sinhvien[n+1];                    // Bỏ phần tử thứ 0
	for (int i = 1; i <= n; i++) {
		cout << "\nNhập sinh viên thứ: " << i << endl;
		cout << "\nHọ tên: "; cin.ignore(); cin.getline(sv[i].hoten);
		cout << "\nTuổi: "; cin >> sv[i].tuoi;
		cout << "\nĐiểm: "; cin >> sv[i].diem;
	}
}
```

```
void Sinhvien::ghi(char fname)
{
	ofstream f(fname) ;
	f << sosv;
	f << setprecision(1) << setiosflags(ios::showpoint) ;
	for (int i=1; i<=sosv; i++) {
		f << endl << setw(24) << sv[i].hoten << setw(4) << tuoi;
		f << setw(8) << sv[i].diem;
	}
	f.close();
}

void Sinhvien::doc(char fname)
{
	ifstream f(fname) ;
	f >> sosv;
	for (int i=1; i<=sosv; i++) {
		f.getline(sv[i].hoten, 25);
		f >> sv[i].tuoi >> sv[i].diem;
	}
	f.close();
}

void Sinhvien::sapxep()
{
	int n = sosv;
	for (int i = 1; i < n; i++) {
	for (int j = j+1; j <= n; j++) {
	if (sv[i].tuoi > sv[j].tuoi) {
		Sinhvien t = sv[i]; sv[i] = sv[j]; sv[j] = t;
	}
}

void main() {
	clrscr();
```

```
    Sinhvien x ;
    x.nhap(); x.ghi("DSSV1");
    x.doc("DSSV1"); x.sapxep(); x.ghi("DSSV2");
    cout << "Đã xong";
    getch();
}
```

3. Kiểm tra sự tồn tại của file, kiểm tra hết file

Việc mở một file chưa có để đọc sẽ gây nên lỗi và làm dừng chương trình. Khi xảy ra lỗi mở file, giá trị trả lại của phương thức bad là một số khác 0. Do vậy có thể sử dụng phương thức này để kiểm tra một file đã có trên đĩa hay chưa. Ví dụ:

```
ifstream f("Bai tap");
if (f.bad()) {
    cout << "file Baitap chưa có";
    exit(1);
}
```

Khi đọc hoặc ghi, con trỏ file sẽ chuyển dần về cuối file. Khi con trỏ ở cuối file, phương thức eof() sẽ trả lại giá trị khác không. Do đó có thể sử dụng phương thức này để kiểm tra đã hết file hay chưa.

Chương trình sau cho phép tính độ dài của file Baitap. File cần được mở theo kiểu nhị phân.

```
 #include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <conio.h>
void main()
{
    clrscr();
    long dodai = 0;
    char ch;
    ifstream f("Baitap", ios::in | ios::binary) ;
    if (f.bad()) {
        cout << "File Baitap không có";
        exit(1);
    }
```

```
        while (!f.eof()) {
            f.get(ch));
            dodai++;
        }
        cout << "Độ dài của file = " << dodai;
        getch();
}
```

4. Đọc ghi đồng thời trên file

Để đọc ghi đồng thời, file phải được gắn với đối tượng của lớp fstream là lớp thừa kế của 2 lớp ifstream và ofstream. Khi đó chế độ phải được bao gồm chỉ định ios::in | ios::out. Ví dụ:

```
fstream f("Data", ios::in | ios::out) ;
```

hoặc

```
fstream f ;
f.open("Data", ios::in | ios::out) ;
```

5. Di chuyển con trỏ file

Các phương thức sau cho phép làm việc trên đối tượng của dòng xuất (ofstream).

- **đối_tượng.seekp(n)** ; Di chuyển con trỏ đến byte thứ n (các byte được tính từ 0)
- **đối_tượng.seekp(n, vị trí xuất phát) ;** Di chuyển đi n byte (có thể âm hoặc dương) từ vị trí xuất phát. Vị trí xuất phát gồm:
  - ios::beg : từ đầu file
  - ios::end : từ cuối file
  - ios::cur : từ vị trí hiện tại của con trỏ.
- **đối_tượng.tellp(n) ;** Cho biết vị trí hiện tại của con trỏ.

Để làm việc với dòng nhập tên các phương thức trên được thay tương ứng bởi các tên : **seekg** và **tellg**. Đối với các dòng nhập lẫn xuất có thể sử dụng được cả 6 phương thức trên.

Ví dụ sau tính độ dài tệp đơn giản hơn ví dụ ở trên.

```
fstream f("Baitap");
f.seekg(0, ios::end);
cout << "Độ dài bằng = " << f.tellg();
```

Ví dụ 4 : Chương trình nhập và in danh sách sinh viên trên ghi/đọc đồng thời.

```
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
void main() {
        int stt ;
        char *hoten, *fname, traloi;
        int tuoi;
        float diem;
        fstream f;
        cout << "Nhập tên file: "; cin >> fname;
        f.open(fname, ios::in | ios::out | ios::noreplace) ;
        if (f.bad()) {
                cout << "Tệp đã có. Ghi đè (C/K)?" ;
                cin.get(traloi) ;
                if (toupper(traloi) == 'C') {
                        f.close() ;
                        f.open(fname, ios::in | ios::out | ios::trunc) ;
                } else exit(1);
        }
        stt = 0;
        f << setprecision(1) << setiosflags(ios::showpoint) ;
        // nhập danh sách
        while (1) {
                stt++;
                cout << "\nNhập sinh viên thứ " << stt ;
                cout << "\nHọ tên: "; cin.ignore() ; cin.getline(hoten, 25);
                if (hoten[0] = 0) break;
                cout << "\nTuổi: "; cin >> tuoi;
```

```
            cout << "\nĐiểm: "; cin >> diem;
            f << setw(24) << hoten << endl;
            f << setw(4) << tuoi << set(8) << diem ;
        }
        // in danh sách
        f.seekg(0) ;                        // quay về đầu danh sách
        stt = 0;
        clrscr();
        cout << "Danh sách sinh viên đã nhập\n" ;
        cout << setprecision(1) << setiosflags(ios::showpoint) ;
        while (1) {
            f.getline(hoten,25);
            if (f.eof()) break;
            stt++;
            f >> tuoi >> diem;
            f.ignore();
            cout << "\nSinh viên thứ " << stt ;
            cout << "\nHọ tên: " << hoten;
            cout << "\nTuổi: " << setw(4) << tuoi;
            cout << "\nĐiểm: " << setw(8) << diem;
        }
        f.close();
        getch();
    }
```

V. NHẬP/XUẤT NHỊ PHÂN

1. Khái niệm về 2 loại file: văn bản và nhị phân

a. File văn bản

Trong file văn bản mỗi byte được xem là một kí tự. Tuy nhiên nếu 2 byte 10 (LF), 13 (CR) đi liền nhau thì được xem là một kí tự và nó là kí tự xuống dòng. Như vậy file văn bản là một tập hợp các dòng kí tự với kí tự xuống dòng có mã là 10. Kí tự có mã 26 được xem là kí tự kết thúc file.

b. File nhị phân

Thông tin lưu trong file được xem như dãy byte bình thường. Mã kết thúc file được chọn là -1, được định nghĩa là EOF trong stdio.h. Các thao tác trên file nhị phân thường đọc ghi từng byte một, không quan tâm ý nghĩa của byte.

Một số các thao tác nhập/xuất sẽ có hiệu quả khác nhau khi mở file dưới các dạng khác nhau.

Ví dụ 1 : giả sử ch = 10, khi đó f << ch sẽ ghi 2 byte 10,13 lên file văn bản f, trong khi đó lệnh này chỉ khi 1 byte 10 lên file nhị phân.

Ngược lại, nếu f la file văn bản thì f.getc(ch) sẽ trả về chỉ 1 byte 10 khi đọc được 2 byte 10, 13 liên tiếp nhau.

Một file luôn ngầm định dưới dạng văn bản, do vậy để chỉ định file là nhị phân ta cần sử dụng cờ ios::binary.

2. Đọc, ghi kí tự

- **put(c);** // ghi kí tự ra file
- **get(c);** // đọc kí tự từ file

Ví dụ 2 : Sao chép file 1 sang file 2. Cần sao chép và ghi từng byte một do vậy để chính xác ta sẽ mở các file dưới dạng nhị phân.

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <conio.h>
void main()
{
        clrscr();
        fstream fnguon("DATA1", ios::in | ios::binary);
        fstream fdich("DATA2", ios::out | ios::binary);
        char ch;
        while (!fnguon.eof()) {
                fnguon.get(ch);
                fdich.put(ch);
        }
        fnguon.close();
        fdich.close();
}
```

3. Đọc, ghi dãy kí tự

- **write(char *buf, int n);** // ghi n kí tự trong buf ra dòng xuất
- **read(char *buf, int n);** // nhập n kí tự từ buf vào dòng nhập
- **gcount();** // cho biết số kí tự read đọc được

Ví dụ 3 : Chương trình sao chép file ở trên có thể sử dụng các phương thức mới này như sau:

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <conio.h>
void main()
{
        clrscr();
        fstream fnguon("DATA1", ios::in | ios::binary);
        fstream fdich("DATA2", ios::out | ios::binary);
        char buf[2000] ;
        int n = 2000;
        while (n) {
                fnguon.read(buf, 2000);
                n = fnguon.gcount();
                fdich.write(buf, n);
        }
        fnguon.close();
        fdich.close();
}
```

4. Đọc ghi đồng thời

```
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
```

```
#include <conio.h>
#include <string.h>
#include <ctype.h>

struct Sv {
	char *hoten;
	int tuoi;
	double diem;
};
class Sinhvien {
	int sosv;
	Sv x;
	char fname[30];
	static int size;
public:
	Sinhvien(char *fn);
	void tao();
	void bosung();
	void xemsua();
};

int Sinhvien::size = sizeof(Sv);
Sinhvien::Sinhvien(char *fn)
{
	strcpy(fname, fn) ;
	fstream f;
	f.open(fname, ios::in | ios::ate | ios::binary);
	if (!f.good) sosv = 0;
	else {
		sosv = f.tellg() / size;
	}
}

void Sinhvien::tao()
```

```
{
        fstream f;
        f.open(fname, ios::out | ios::noreplace | ios::binary);
        if (!f.good()) {
                cout << "danh sach da co. Co tao lai (C/K) ?";
                char traloi = getch();
                if (toupper(traloi) == 'C') return;
                else {
                        f.close() ;
                        f.open(fname, ios::out | ios::trunc | ios::binary);
                }
        }
        sosv = 0
        while (1) {
                cout << "\nSinh viên thứ: " << sosv+1;
                cout << "\nHọ tên: "; cin.ignore(); cin.getline(x.hoten);
                if (x.hoten[0] == 0) break;
                cout << "\nTuổi: "; cin >> x.tuoi;
                cout << "\nĐiểm: "; cin >> x.diem;
                f.write((char*)(&x), size);
                sosv++;
        }
        f.close();
}

void Sinhvien::bosung()
{
        fstream f;
        f.open(fname, ios::out | ios::app | ios::binary);
        if (!f.good()) {
                cout << "danh sach chua co. Tao moi (C/K) ?";
                char traloi = getch();
                if (toupper(traloi) == 'C') return;
                else {
```

```
                f.close() ;
                f.open(fname, ios::out | ios::binary);
            }
        }
        int stt = 0
        while (1) {
            cout << "\nBổ sung sinh viên thứ: " << stt+1;
            cout << "\nHọ tên: "; cin.ignore(); cin.getline(x.hoten);
            if (x.hoten[0] == 0) break;
            cout << "\nTuổi: "; cin >> x.tuoi;
            cout << "\nĐiểm: "; cin >> x.diem;
            f.write((char*)(&x), size);
            stt++;
        }
        sosv += stt;
        f.close();
}

void Sinhvien::xemsua()
{
        fstream f;
        int ch;
        f.open(fname, ios::out | ios::app | ios::binary);
        if (!f.good()) {
            cout << "danh sach chua co";
            getch(); return;
        }
        cout << "\nDanh sách sinh viên" << endl;
        int stt ;
        while (1) {
            cout << "\nCần xem (sua) sinh viên thứ (0: dừng): " ;
            cin >> stt;
            if (stt < 1 || stt > sosv) break;
            f.seekg((stt-1) * size, ios::beg);
```

```
            f.read((char*)(&x), size);
            cout << "\nHọ tên: " << x.hoten;
            cout << "\nTuổi: " << x.tuoi;
            cout << "\nĐiểm: " << x.diem;
            cout << "Có sửa không (C/K) ?";
            cin >> traloi;
            if (toupper(traloi) == 'C') {
                f.seekg(-size, ios::cur);
                cout << "\nHọ tên: "; cin.ignore(); cin.getline(x.hoten);
                cout << "\nTuổi: "; cin >> x.tuoi;
                cout << "\nĐiểm: "; cin >> x.diem;
                f.write((char*)(&x), size);
            }
        }
        f.close();
}

void main()
{
        int chon;
        Sinhvien SV("DSSV") ;
        while (1) {
            clrscr();
            cout << "\n1: Tạo danh sách sinh viên";
            cout << "\n2: Bổ sung danh sách";
            cout << "\n3: Xem – sửa danh sách";
            cout << "\n0: Kết thúc";
            chon = getch();
            chon = chon – 48;
            clrscr();
            if (chon == 1) SV.tao();
            else if (chon == 2) SV.bosung();
            else if (chon == 3) SV.xemsua();
            else break;
```

```
        }
    }
```

BÀI TẬP

1. Viết chương trình đếm số dòng của một file văn bản.

2. Viết chương trình đọc in từng kí tự của file văn bản ra màn hình, mỗi màn hình 20 dòng.

3. Viết chương trình tìm xâu dài nhất trong một file văn bản.

4. Viết chương trình ghép một file văn bản thứ hai vào file văn bản thứ nhất, trong đó tất cả chữ cái của file văn bản thứ nhất phải đổi thành chữ in hoa.

5. Viết chương trình in nội dung file ra màn hình và cho biết tổng số chữ cái, tổng số chữ số đã xuất hiện trong file.

6. Cho 2 file số thực (đã được sắp tăng dần). In ra màn hình dãy số xếp tăng dần của cả 2 file. (Cần tạo cả 2 file dữ liệu này bằng Editor của C++).

7. Viết hàm nhập 10 số thực từ bàn phím vào file INPUT.DAT. Viết hàm đọc các số thực từ file trên và in tổng bình phương của chúng ra màn hình.

8. Viết hàm nhập 10 số nguyên từ bàn phím vào file văn bản tên INPUT.DAT. Viết hàm đọc các số nguyên từ file trên và ghi những số chẵn vào file EVEN.DAT còn các số lẻ vào file ODD.DAT.

9. Nhập bằng chương trình 2 ma trận số nguyên vào 2 file văn bản. Hãy tạo file văn bản thứ 3 chứa nội dung của ma trận tích của 2 ma trận trên.

10. Tổ chức quản lý file sinh viên (Họ tên, ngày sinh, giới tính, điểm) với các chức năng : Nhập, xem, xóa, sửa, tính điểm trung chung.

11. Thông tin về một nhân viên trong cơ quan bao gồm : họ và tên, nghề nghiệp, số điện thoại, địa chỉ nhà riêng. Viết hàm nhập từ bàn phím thông tin của 7 nhân viên và ghi vào file INPUT.DAT. Viết hàm tìm trong file INPUT.DAT và in ra thông tin của 1 nhân viên theo số điện thoại được nhập từ bàn phím.

MỤC LỤC

Chương 3. CẤU TRÚC ĐIỀU KHIỂN VÀ DỮ LIỆU KIỂU MẢNG

Chương 4. HÀM VÀ CHƯƠNG TRÌNH

Chương 5. DỮ LIỆU KIỂU CẤU TRÚC VÀ HỢP

TÀI LIỆU THAM KHẢO

1. B.W. Kerninghan and D.M. Ritchie. *The C Programming Language*. Prentice-Hall, 1978. Ngô Trung Việt (dịch). *Ngôn ngữ lập trình C*. Viện Tin học, Hà Nội 1990.
2. Peter Norton. *Advanced C Programming*. Nguyễn Việt Hải (dịch). *Lập trình C nâng cao*. Nhà xuất bản Giao thông vận tải. Hà Nội, 1995.
3. Phạm Văn Ất. *Kỹ thuật lập trình C. Cơ sở và nâng cao*. Nhà xuất bản Khoa học và kỹ thuật. Hà Nội, 1996.
4. Phạm Văn Ất. *C++ và lập trình hướng đối tượng*. Nhà xuất bản Khoa học và kỹ thuật. Hà Nội, 2000.
5. Scott Robert Ladd. Nguyễn Hùng (dịch). *C++ Kỹ thuật và ứng dụng*. Công ty cổ phần tư vấn và dịch vụ KHKT - SCITEC, 1992.
6. Jan Skansholm. C++ From the Beginning. Addison-Wesley, 1997.