Chương 1

TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH C/C++

1. Lịch sử của ngôn ngữ lập trình C/C++

C được tạo bởi Dennis Ritchie ở Bell Telephone Laboratories vào năm 1972 để cho phép lập trình viên phát triển các ứng dụng hiệu quả hơn các ngôn ngữ lập trình hiện có tại thời điểm đó.

Điểm mạnh và mềm dẻo của C cho phép các nhà phát triển ở Bell Labs tạo nên các ứng dụng phức tạp như hệ điều hành Unix. Vào năm 1983, học viện chuẩn quốc gia Mỹ (American National Standards Institute - ANSI) thành lập một tiểu ban để chuẩn hóa C được biết đến như ANSI Standard C. Ngày nay, tất cả trình biên dịch C/C++ đều tuân theo ANSI Standard C. C++ được xây dựng trên nền tảng của ANSI Standard C.

C++ là một ngôn ngữ lập trình hướng đối tượng mà bao hàm ngôn ngữ C ở trong nó. Trong giáo trình này chưa khảo sát phần lập trình hướng đối tượng của C++.

2. Kỹ thuật để giải quyết một bài toán

Một chương trình máy tính được thiết kế để giải quyết một bài toán nào đó. Vì vậy, những bước cần để tìm kiếm lời giải cho một bài toán cũng giống như những bước cần để viết một chương trình. Các bước gồm:

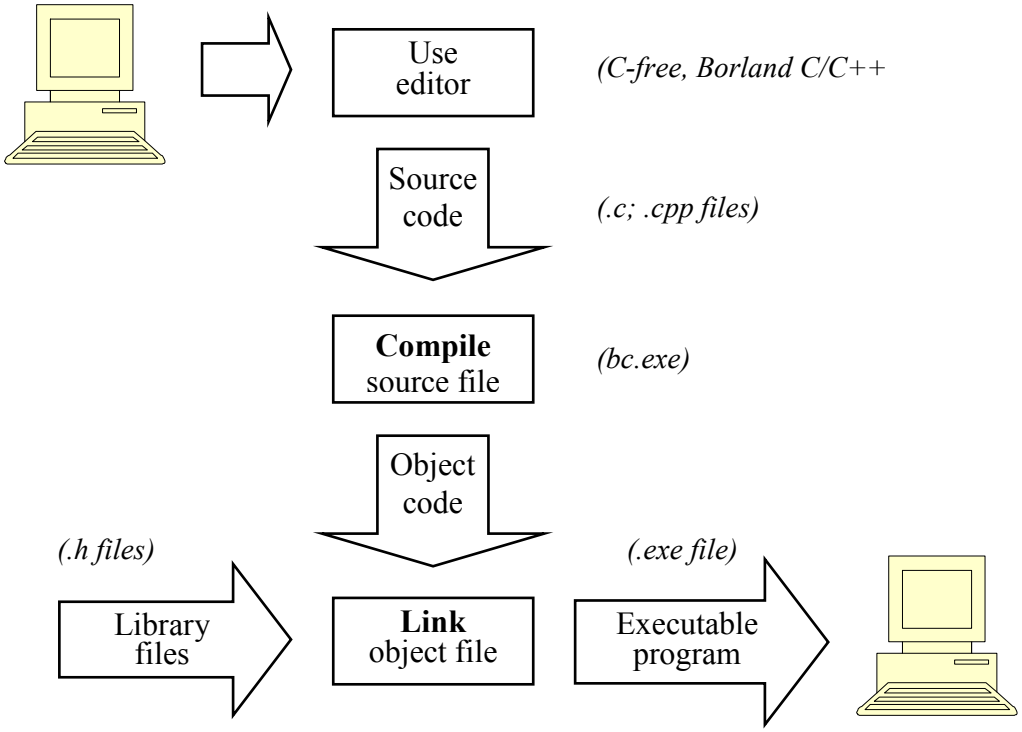
- Xác định yêu cầu của bài toán.
- Nghĩ ra một phương cách *(algorithm)* để tìm lời giải.
- Thực hiện phương cách đó.
- Kiểm tra kết quả để xem lời giải có đúng với yêu cầu của bài toán.

Khi viết một chương trình trong C/C++, đây là những bước được đề nghị:

1. Xác định mục đích của chương trình
2. Nghĩ ra thuật toán phù hợp để giải quyết bài toán (dùng mã giả, lưu đồ, ...)
3. Cài đặt (viết) thuật toán dùng ngôn ngữ lập trình C/C++
4. Thực thi chương trình và kiểm thử *(testing)*.

3. Các bước trong chu trình phát triển chương trình

Chu trình phát triển chương trình (*program development cycle*) có những bước sau đây. 1. Một trình soạn thảo văn bản được dùng để nhập mã nguồn (*source code*). 2. Mã nguồn được biên dịch (*compile*) để tạo nên tập tin đối tượng (*object file*). 3. Các tập tin đối tượng được liên kết (*link*) để tạo nên tập tin có thể thực thi (*executable file*). 4. Thực hiện (*run*) chương trình để chỉ ra chương trình có làm việc đúng như đã định không.



3.1. Soạn thảo mã nguồn (*source code editor*)

Mã nguồn là một tập các lệnh dùng để chỉ dẫn máy tính thực hiện công việc mong muốn. Tập tin mã nguồn được lưu trữ với phần phân loại .c (C) hoặc .cpp (C++).

3.2. Biên dịch (*compile*)

Tập tin mã nguồn được viết bằng những từ giống tiếng Anh nên dễ dàng để đọc và hiểu. Tuy nhiên, máy tính không thể hiểu những từ này. Máy tính yêu cầu các chỉ dẫn nhị phân (*binary*) trong dạng thức của ngôn ngữ máy (*machine language*). Trước khi một chương trình được viết bằng ngôn ngữ cấp cao như C/C++ có thể thực thi trên máy tính, nó phải được biên dịch từ mã nguồn sang mã máy. Việc dịch này được thực hiện bởi một chương trình gọi là trình biên dịch (*compiler*).

Các chỉ dẫn ngôn ngữ máy được tạo bởi trình biên dịch được gọi là mã đối tượng (*object code*) và tập tin chứa chúng gọi là tập tin đối tượng. Tập tin đối tượng có cùng tên như tập tin mã nguồn nhưng có phần phân loại .obj.

3.3. Tạo tập tin thực thi (*executable files*)

C/C++ có một thư viện hàm chứa mã đối tượng cho những hàm đã được tạo sẵn. Những hàm này thực hiện các tác vụ thường dùng như xóa màn hình (clrscr()), nhập một chuỗi ký tự từ bàn phím (gets()), tính căn bậc hai (sqrt()), ... mà chương trình được viết có thể sử dụng mà không phải viết lại.

Tập tin đối tượng được tạo ra bởi trình biên dịch sẽ kết hợp với mã đối tượng của các hàm thư viện để tạo nên tập tin thực thi. Quá trình này được gọi là liên kết (*linking*), được thực hiện bởi một chương trình gọi là bộ liên kết (*linker*).

3.4. Thực thi chương trình

Khi chương trình nguồn được biên dịch và liên kết để tạo nên tập tin thực thi (có phần phân loại .exe), nó có thể thực thi trên máy tính tại dấu nhắc hệ thống.

Nếu chương trình hoạt động không đúng như yêu cầu, vấn đề có thể là do lỗi lập trình. Trong trường hợp này, chỉnh sửa chương trình nguồn, biên dịch lại và liên kết lại để tạo nên phiên bản mới của tập tin chương trình.

Quá trình bốn bước này được lập đi lập lại cho đến khi tập tin thực thi thực hiện đúng yêu cầu của bài toán.

4. Khảo sát một chương trình C/C++ đơn giản

Khảo sát một chương trình đơn giản dùng để xuất ra màn hình dòng chữ Hello World!

```
// my first program in C/C++
#include <conio.h>
#include <iostream.h>
int main()
{
      cout << "Hello World!"; //Output "Hello World!"
      getch();
      return 0;
}
```

Đây là chương trình đơn giản nhưng nó đã bao hàm những thành phần cơ bản mà mọi chương trình C/C++ đều có. Với ý nghĩa của từng dòng như sau:

// my first program in C/C++

Đây là dòng chú thích, tất cả các dòng bắt đầu bằng hai dấu // được coi là các dòng chú thích, nó không ảnh hưởng đến hoạt động của chương trình, chỉ dùng để giải thích mã nguồn của chương trình.

#include < iostream.h>

Các lệnh bắt đầu bằng dấu # được dùng cho các chỉ thị tiền xử lý (*preprocessor*). Trong ví dụ này, câu lệnh **#include** báo cho trình biên dịch biết cần phải gộp thư viện **iostream.h** là tập tin header chuẩn của C/C++, chứa các định nghĩa về nhập và xuất.

int main()

Định nghĩa hàm **main().** Hàm main() là điểm mà tất cả các chương trình C/C++ bắt đầu thực hiện. Nó không phụ thuộc vào vị trí của hàm, nội dung của nó luôn được thực hiện đầu tiên khi chương trình thực thi. Một chương trình C/C++ đều phải tồn tại một hàm main(). Hàm main() có thể có hoặc không có tham số. Nội dung của hàm main() tiếp ngay sau phần khai báo chính thức được đặt trong cặp dấu ngoặc { }.

cout << "Hello World!";

Đây là một lệnh nằm trong phần thân của hàm main. **cout** là một dòng (*stream*) xuất chuẩn trong C/C++ được định nghĩa trong thư viện **iostream.h**. Khi dòng lệnh này được thực thi, kết quả là chuỗi "Hello World!" được xuất ra màn hình. Dòng lệnh được kết thúc bằng dấu chấm phẩy (;).

getch();

Đây là một hàm thư viện dùng để chờ nhập một ký tự từ bàn phím.

return 0;

Lệnh **return** kết thúc hàm main và trả về mã đi sau nó, trong trường hợp này là 0. Đây là một kết thúc bình thường của một chương trình không có lỗi trong quá trình thực hiện.

Chương trình trên có thể viết lại như sau:

```
int main() { cout << " Hello World! "; getch(); return 0;}
```

cũng cho cùng một kết quả.

5. Các chú thích (*comments*)

Các chú thích được các lập trình viên sử dụng để ghi chú hay mô tả trong các phần của chương trình. Trong C/C++ có hai cách để chú thích:

Chú thích dòng: dùng cặp dấu //. Từ vị trí // đến cuối dòng được xem là chú thích

Chú thích khối (chú thích trên nhiều dòng) dùng cặp /* ... */. Những gì nằm giữa cặp dấu này được xem là chú thích.

Ví dụ:

```
/*    My second program in C/C++ with more comments
      Author: Novice programmer
      Date: 01/01/2008
*/
#include <conio.h>
#include <iostream.h>
int main()
{
   cout << "Hello World! "; // output Hello World!
   cout << "I hate C/C++."; // output I hate C/C++.
   getch();
   return 0;
}
```

Kết *quả xuất của chương trình là:* **Hello World! I hate C/C++.**

6. Cấu trúc của một chương trình C/C++

Cấu trúc một chương trình C/C++ bao gồm các thành phần như: Các chỉ thị tiền xử lý, khai báo biến toàn cục, chương trình chính (hàm main), ...

Khảo sát chương trình sau:

```
/* fact.c
Purpose: prints the factorials of
the numbers from 0 through 10
Author: Mr.Beginner
Date: 01/01/2008
*/
```

Phần này thường dùng để mô tả mục đính chương trình, tác giả, ngày viết, ... (Phần không bắt buộc)

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

Khai báo các tập tin thư viện

```
int factorial(int n);
```

Khai báo prototype của các hàm tự tạo

```
int main()
{
   int i;
   for(i=0; i<=10; i++)
     cout<<i<<"!="<<factorial(i);
   return 0;
}
```

Hàm chính của chương trình

```
/* This function computes the
factorial of its parameter, returning it */
int factorial(int n)
{
    int i, product;
    product = 1;
    for (i=2;i<=n;i++) prod *= i;
    return product;
}
```

Định nghĩa các hàm do người dùng tự xây dựng

7. Các tập tin thư viện thông dụng

Đây là các tập tin chứa định nghĩa các hàm thông dụng khi lập trình C/C++. Muốn sử dụng các hàm trong các tập tin header này thì phải khai báo #include <FileName.h> ở phần đầu của chương trình, với FileName.h là tên tập tin thư viện. Các tập tin thư viện thông dụng gồm:

stdio.h(C), **iostream.h**(C++) Tập tin định nghĩa các hàm vào/ra chuẩn (standard input/output) gồm các hàm xuất dữ liệu (printf())/cout), nhập giá trị cho biến (scanf())/cin), nhận ký tự từ bàn phím (getc()), in ký tự ra màn hình (putc()), nhập một chuỗi ký tự từ bàm phím (gets()), xuất chuỗi ký tự ra màn hình (puts()), xóa vùng đệm bàn phím (fflush()), fopen(), fclose(), fread(), fwrite(), getchar(), putchar(), ...

conio.h : Tập tin định nghĩa các hàm vào ra trong chế độ DOS (DOS console) gồm các hàm clrscr(), getch(), getche(), getpass(), cgets(), cputs(), putch(), clreol(), …

math.h: Tập tin định nghĩa các hàm toán học gồm các hàm abs(), sqrt(), log(), log10(), sin(), cos(), tan(), acos(), asin(), atan(), pow(), exp(), …

alloc.h: Tập tin định nghĩa các hàm liên quan đến việc quản lý bộ nhớ gồm các hàm calloc(), realloc(), malloc(), free(), farmalloc(), farcalloc(), farfree(), …

io.h: Tập tin định nghĩa các hàm vào ra cấp thấp gồm các hàm open(), _open(), read(), _read(), close(), _close(), creat(), _creat(), creatnew(), eof(), filelength(), lock(), …

Chương 2
BIỂU THỨC
(Expressions)

Biểu thức được tạo thành từ những thành tố như dữ liệu và toán tử. Dữ liệu có thể chứa trong biến hoặc hằng. Toán tử trong các ngôn ngữ lập trình có cùng ý nghĩa như trong toán học. Có nhiều loại toán tử như toán tử gán (=), toán tử số học (+ - * / %), toán tử quan hệ (== < <= > >= !=), toán tử luận lý (&& || !), ...

1. Kiểu dữ liệu (*data types*)

C/C++ có năm kiểu dữ liệu cơ sở: ký tự (*char*), số nguyên (*int*), số thực (*float*), số thực có độ chính xác gấp đôi (*double*), và kiểu vô định (*void*). Tất cả những kiểu dữ liệu khác đều dựa trên 5 kiểu cơ sở này. Kích thước và phạm vi của những kiểu dữ liệu này có thể thay đổi tùy theo loại CPU và trình biên dịch.

Kiểu char được dùng để giữ các giá trị của bộ mã ASCII (*Amercican Standard Code for Information Interchange*). Kiểu char chiếm 1 byte bộ nhớ.

Kích thước của kiểu int là 16 bits (2 bytes) trên môi trường 16-bit như DOS, Windows 3.1. Môi trường 32-bit như Windows 95, kích thước kiểu int là 32 bits (4 bytes). Nói chung, tùy thuộc môi trường, kích thước của kiểu int có thể khác nhau.

Chuẩn C chỉ ra phạm vi tối thiểu của kiểu dữ liệu số thực (*float, double*) là 1E-37 đến 1E+37.

Kiểu void dùng để khai báo hàm không trả về giá trị hoặc tạo nên các con trỏ tổng quát (*generic pointers*).

Ngoại trừ kiểu void, các kiểu dữ liệu cơ sở có thể có các từ như signed, unsigned, long, short đi trước nó. Các từ này làm thay đổi phạm vi tối thiểu mỗi kiểu cơ sở có thể chứa. Bảng sau trình bày tất cả các kết hợp hợp lệ của kiểu dữ liệu với các từ trên.

Kiểu dữ liệu	Kích thước bằng bits	Phạm vi tối thiểu
char	8	-127 to 127
unsigned char	8	0 to 255
signed char	8	-127 to 127
int	16 or 32	-32,767 to 32,767
unsigned int	16 or 32	0 to 65,535
signed int	16 or 32	giống như int
short int hoặc short	16	-32,767 to 32,767
unsigned short int	16	0 to 65,535
signed short int	16	giống như short int
long int hoặc long	32	-2,147,483,647 to 2,147,483,647
signed long int	32	giống như long int
unsigned long int	32	0 to 4,294,967,295
float	32	Độ chính xác là 6 ký số
double	64	Độ chính xác là 10 ký số
long double	80	Độ chính xác là 10 ký số

2. Các định danh (*Identifier names*)

Trong C/C++, tên của các biến, hằng, hàm,... được gọi là định danh. Những định danh này có thể là 1 hoặc nhiều ký tự. Ký tự đầu tiên phải là một chữ cái hoặc dấu _ (*underscore*), những ký tự theo sau phải là chữ cái, chữ số, hoặc dấu _. Sau đây là những định danh đúng và sai:

Đúng	**Sai**
count	1count
test23	hi!there
high_balance	high...balance

C/C++ phân biệt ký tự HOA và thường. Vì vậy, count, Count, và COUNT là 3 danh định khác nhau.

Định danh không được trùng với các từ khóa (*keywords*) và không nên có cùng tên như các hàm thư viện của C/C++.

3. Từ khóa (*keywords*)

Là những từ đã được dành riêng bởi ngôn ngữ lập trình cho những mục đích riêng của nó. Không được dùng từ khóa để đặt tên cho những định danh như biến, hằng, hàm, ... Tất cả các từ khóa trong C/C++ đều là chữ thường (*lowercase*). Sau đây là danh sách các từ khóa của C/C++:

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

4. Biến (*variables*)

Một biến là định danh của một vùng trong bộ nhớ dùng để giữ một giá trị mà có thể bị thay đổi bởi chương trình. Tất cả biến phải được khai báo trước khi được sử dụng. Dạng khai báo biến tổng quát là:

```
type variableNames;
```

type: phải là một trong các kiểu dữ liệu hợp lệ.

variableNames: tên của một hay nhiều biến phân cách nhau bởi dấu phẩy.

Ngoài ra, ta có thể vừa khai báo vừa khởi tạo giá trị ban đầu cho các biến dùng cú pháp sau:

```
type varName1=value, ... , varName_n=value;
```

Ví dụ:

```
int i, j; // khai báo 2 biến i, j kiểu int
// khai báo ba biến day, month và year kiểu short int
short day, month, year;
```

```
char ch; // khai báo biến ch kiểu ký tự
// khai bao 4 bien kieu float, gán average giá trị 0
float mark1, mark2, mark3, average = 0;
```

Lưu ý: Khi khai báo biến nếu không cung cấp giá trị khởi tạo thì giá trị của biến là chưa xác định. Do đó, việc dùng những biến này trong các biểu thức là vô nghĩa.

Biến được khai báo tại ba nơi: bên trong hàm, trong định nghĩa tham số của hàm, và bên ngoài tất cả hàm. Những biến này được gọi lần lượt là biến cục bộ, các tham số hình thức, và biến toàn cục.

4.1. Biến cục bộ (*local variables*)

Những biến được khai báo bên trong một hàm gọi là biến cục bộ. Các biến cục bộ chỉ được tham chiếu đến chỉ bởi những lệnh nằm trong khối (*block*) có khai báo biến. Một khối bắt đầu với dấu { và kết thúc với dấu }.

Biến cục bộ chỉ tồn tại trong khi khối chứa nó đang thực thi và bị hủy khi khối chứa nó thực thi xong.

Ví dụ: Xem xét hai hàm sau:

```
void func1(void)
{
      int x;
      x = 10;
}

void func2(void)
{
      int x;
      x = -199;
}
```

Biến nguyên x được khai báo 2 lần, một trong hàm func1() và một trong hàm func2(). Biến x trong func1() không có quan hệ gì với biến x trong func2() bởi vì mỗi x chỉ tồn tại trong khối chứa nó.

4.2. Các tham số hình thức (*formal parameters*)

Nếu một hàm có nhận các đối số truyền vào hàm thì nó phải khai báo các biến để nhận giá trị của các đối số khi hàm được gọi. Những biến này gọi là các tham số hình thức. Những biến này được đối xử giống như các biến cục bộ khác được khai báo trong hàm. Xem xét ví dụ sau:

```
int sum(int from, int to)
{
      int total=0;
      for(int i=from ; i<=to ; i++) total +=i;
      return total;
}
```

Trong ví dụ này, from và to là 2 tham số hình thức và được đối xử như biến cục bộ total của hàm sum. Lưu ý biến i được khai báo trong cấu trúc lặp for nên nó là biến cục bộ chỉ tồn tại trong cấu trúc for mà thôi. Những lệnh nằm ngòai cấu trúc for sẽ không tham chiếu được biến i này.

4.3. Biến toàn cục (*global variables*)

Biến toàn cục có phạm vi là toàn bộ chương trình. Do đó, tất cả các lệnh có trong chương trình đều có thể tham chiếu đến biến toàn cục. Biến toàn cục được khai báo bên ngoài tất cả hàm.

Khảo sát chương trình sau:

```
#include <iostream.h>
void increase();
void decrease();
int gVar = 100;
void main()
{
      cout << "Value of gVar= " << gVar;
      increase();
      cout << "After increased, gVar= " << gVar;
      decrease();
      cout << "After decreased, gVar= " << gVar;
}

void increase()
{     gVar = gVar + 1;}

void decrease()
{     gVar = gVar -1;}
```

Sau khi thực thi chương trình trên, kết quả xuất trên màn hình là:

Value of gVar= 100

After increased, gVar= 101;

After decreased, gVar= 100;

5. Từ khóa const

Giá trị của biến thay đổi trong suốt quá trình thực thi chương trình. Để giá trị của biến không bị thay đổi, ta đặt trước khai báo biến từ khóa const. Từ khi biến đã có giá trị, giá trị này sẽ không bao giờ bị thay đổi bởi bất kỳ lệnh nào trong chương trình. Thông thường ta dùng chữ HOA để đặt tên cho những biến này.

Ví dụ: khai báo hằng nguyên MAX có giá trị 100

```
const int MAX = 200;
```

6. Hằng (*constants*)

Hằng là những giá trị cố định (*fixed values*) mà chương trình không thể thay đổi. Bất kỳ kiểu dữ liệu nào cũng có hằng tương ứng. Hằng còn được gọi là literals.

Hằng ký tự được rào quanh bởi cặp dấu nháy đơn. Ví dụ 'a' và '%' là những hằng ký tự.

Hằng nguyên là những số mà không có phần thập phân. Ví dụ 100 và -100 là những hằng nguyên.

Hằng số thực yêu cầu một dấu chấm thập phân phân cách phần nguyên với phần thập phân. Ví dụ 123.45 là một hằng số thực.

Ví dụ về cách viết các loại hằng số:

Kiểu dữ liệu	**Các ví dụ về hằng**	**Ghi chú**
int	1, 123, 21000, 234	
long int	35000L, 34l	Có ký tự l hoặc L ở cuối
unsigned int	10000U, 987u, 40000U	Có ký tự u hoặc U ở cuối
float	123.23f, 4.34e-3F	Có ký tự f hoặc F ở cuối
double	123.23, 1.0, 0.9876324	
long double	1001.2L	Có ký tự l hoặc L ở cuối

7. Hằng chuỗi ký tự (*string constants*)

C/C++ cung cấp một loại hằng khác gọi là chuỗi. Một chuỗi là một tập các ký tự được bao quanh bởi cặp dấu nháy đôi. Ví dụ, "This is a string" là một chuỗi.

Lưu ý phân biệt hằng chuỗi và hằng ký tự. Một hằng ký tự là một ký tự bao quanh bởi cặp dấu nháy đơn. Do đó, 'a' là hằng ký tự nhưng "a" là hằng chuỗi.

8. Hằng ký tự đặc biệt (*escape sequences*)

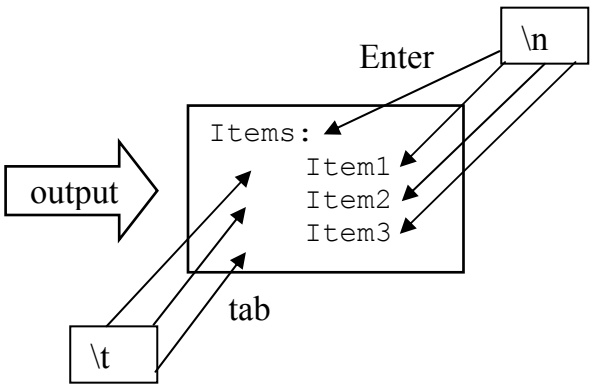
C/C++ có những hằng ký tự đặc biệt mà không thể biểu diễn như những hằng ký tự thông thường. Những hằng này còn gọi là escape sequences. Sau đây là danh sách các hằng đặc biệt đó:

Mã	**Ý nghĩa**
\b	Lùi sang trái 1 ký tự
\f	Về đầu dòng
\n	Sang dòng mới
\r	Xuống dòng
\t	Tab theo chiều ngang
\"	Dấu nháy đôi
\'	Dấu nháy đơn
\0	Null
\\	Dấu \
\v	Tab theo chiều đứng
\a	Cảnh báo
\?	Dấu hỏi
\N	Hằng bát phân (với N là một hằng bát phân)
\xN	Hằng thập lục phân (với N là một hằng thập lục phân)

Xem xét ví dụ sau:

```
#include <iostream.h>
void main(void)
{
  cout <<"Items:\n";
  cout <<"\tItem1\n";
  cout <<"\tItem2\n";
  cout <<"\tItem3\n";

}
```



9. Toán tử (*operators*)

C/C++ có bốn loại toán tử: số học (*arithmetic*), quan hệ (*relational*), luận lý (*logical*), và bitwise.

9.1. Toán tử gán (*assignment operator*)

Dạng tổng quát của toán tử gán là

variableName = expression;

variableName: Tên biến

expression: Biểu thức

Lưu ý, phía bên trái dấu bằng phải là một biến hay con trỏ và không thể là hàm hay hằng.

Ví dụ:

```
total = a + b + c + d;
```

9.2. Chuyển đổi kiểu trong câu lệnh gán

Khi những biến của một kiểu kết hợp với những biến của một kiểu khác thì một sự chuyển đổi kiểu xảy ra. Đối với câu lệnh gán, giá trị của biểu thức bên phải dấu bằng được tự động chuyển thành kiểu dữ liệu của biến bên trái dấu bằng.

Ví dụ:

```
int i=100;
double d = 123.456;
```

Nếu thực thi lệnh

```
i = d;
```

thì i sẽ có giá trị là 123 vì 123.456 sẽ tự động chuyển thành số nguyên nên bị cắt bỏ phần thập phân. Sự chuyển đổi kiểu này gọi là chuyển đổi kiểu bị mất mát thông tin.

Nếu thực thi lệnh

```
d = i;
```

thì d sẽ có giá trị là 100.0. Sự chuyển đổi kiểu này gọi là chuyển đổi kiểu không mất mát thông tin.

Tóm lại, khi chuyển đổi kiểu từ kiểu dữ liệu có miền giá trị nhỏ sang kiểu dữ liệu có miền giá trị lớn hơn thì việc chuyển đổi kiểu này là an toàn vì không bị mất mát thông tin. Thứ tự tăng dần từ kiểu dữ liệu có miền giá trị nhỏ đến kiểu dữ liệu có miền giá trị lớn là **char → int → long → float → double.**

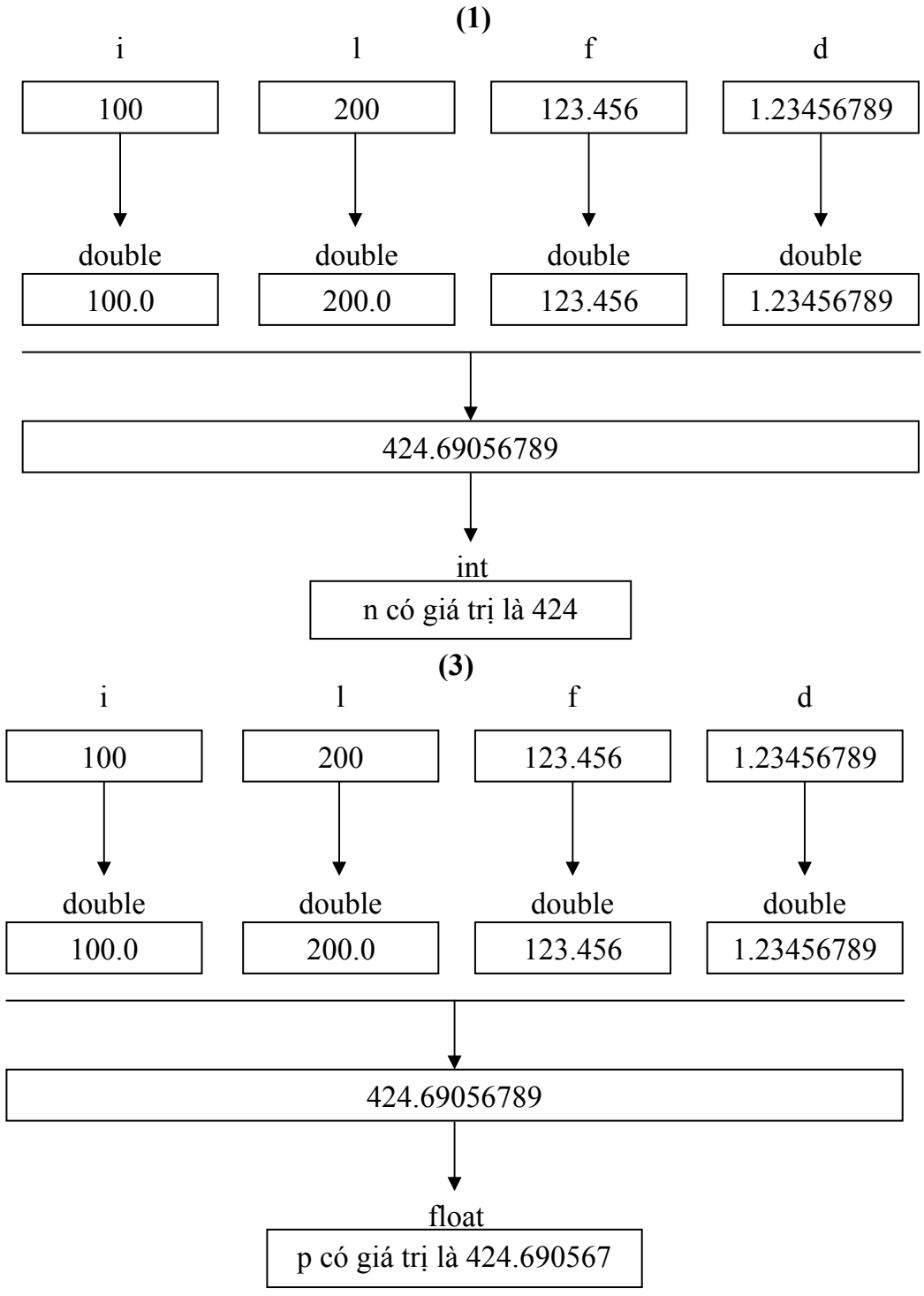
Khi chuyển đổi kiểu dữ liệu có miền giá trị lớn sang kiểu dữ liệu có miền giá trị nhỏ hơn thì việc chuyển đổi kiểu này là không an toàn vì có thể mất mát thông tin. Thứ tự giảm dần từ kiểu dữ liệu có miền giá trị lớn đến kiểu dữ liệu có miền giá trị nhỏ là **double → float → long → int → char.**

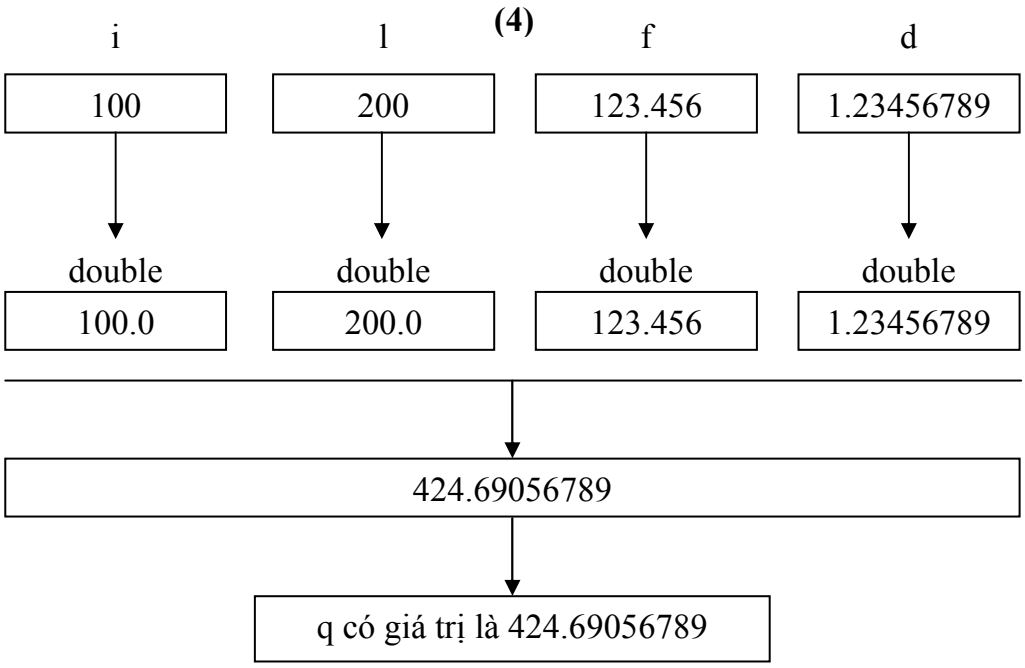
Ví dụ:

```
int i = 100;
long l = 200;
float f = 123.456f;
double d = 1.23456789;
```

Khảo sát các lệnh gán sau:

```
int n; long m; float p; double q;
n = i + l + f + d; // (1)
m = i + l + f + d; // (2)
p = i + l + f + d; // (3)
q = i + l + f + d; // (4)
```


(1)
i
l
f
d
100
200
123.456
1.23456789
double
double
double
double
100.0
200.0
123.456
1.23456789
424.69056789
int
n có giá trị là 424
(3)
i
l
f
d
100
200
123.456
1.23456789
double
double
double
double
100.0
200.0
123.456
1.23456789
424.69056789
float
p có giá trị là 424.690567



Kết luận: Trường hợp (1) và (3): mất mát thông tin. Trường hợp (4) không mất mát thông tin.

10. Các toán tử số học (*arithmetic operators*)

Các toán tử số học gồm: + (cộng), - (trừ), * (nhân), / (chia), và % (lấy phần dư của phép chia nguyên).

Khi tử số và mẫu số của phép chia là số nguyên thì đó là phép chia nguyên nên phần dư của phép chia nguyên bị cắt bỏ. Ví dụ, 5/2 thì kết quả là 2.

Toán tử lấy phần dư % (*modulus operator*) chỉ áp dụng với số nguyên và trả về phần dư. Ví dụ, 7%2 thì kết quả là 1.

11. Toán tử ++ và -- (*increment and decrement operators*)

C/C++ có hai toán tử rất thường dùng là ++ và --. Toán tử ++ cộng 1 đến toán hạng của nó và toán tử -- thì trừ 1 từ toán hạng của nó.

Ví dụ:

```
x++; // tuong duong x = x + 1;
x--; // tuong duong x = x - 1;
```

Toán tử ++ và -- có thể đặt phía trước (*prefix*) hoặc phía sau (*postfix*) toán hạng. Sự khác nhau của 2 trường hợp này là khi toán tử ++ và -- đứng trước toán hạng, hành động tăng và giảm trên toán hạng được thực hiện trước, sau đó giá trị mới của toán hạng sẽ dùng để tham gia vào việc định trị của biểu thức.

Ví dụ: Khảo sát đoạn lệnh sau

```
int x = 100;
int n,m;
```

Nếu thực hiện lệnh:

```
n = ++x + 1; // n sẽ có giá trị là 102 (1)
```

Nếu thực hiện lệnh:

```
n = x++ + 1; // n sẽ có giá trị là 101 (2)
```

Sau khi lệnh (1) hay (2) được thực thi thì x có giá trị là 101

Tương tự, nếu thực hiện lệnh:

```
m = --x + 1; // n sẽ có giá trị là 100 (3)
```

Nếu thực hiện lệnh:

```
m = x-- + 1; // n sẽ có giá trị 101 (4)
```

Sau khi lệnh (3) hay (4) được thực thi thì x có giá trị là 99

Khi các toán tử số học xuất hiện trong một biểu thức, thì độ ưu tiên thực hiện như sau:

Cao nhất ++ – –

– (dấu âm)

***** **/** **%**

Thấp nhất + –

Những toán tử trên cùng hàng thì có cùng độ ưu tiên. Khi biểu thức có nhiều toán tử có cùng độ ưu tiên thì thứ tự định trị biểu thức là từ trái sang phải.

12. Toán tử quan hệ & luận lý (*relational & logical operators*)

Toán tử quan hệ liên quan đến sự quan hệ của hai giá trị. Toán tử luận lý liên quan đến sự nối kết của những quan hệ.

Các biểu thức mà dùng toán tử quan hệ và toán tử luận lý được định trị là true (đúng) hoặc false(sai). Trong C/C++, giá trị 0 (*zero*) được xem là false và giá trị khác 0 (*non-zero*) được xem là true.

Các toán tử quan hệ gồm:

Toán tử	**Ý nghĩa**
>	Lớn hơn
>=	Lớn hơn hay bằng
<	Nhỏ hơn
<=	Nhỏ hơn hay bằng
==	Bằng (có 2 dấu =)
!=	Không bằng

Ví dụ về toán tử quan hệ:

100 < 200 // true

200 == 300 // false

300 != 400 // true

400 >= 500 // false

Các toán tử luận lý:

Operator	**Action**	**Ý nghĩa**
&&	AND	Và
||	OR	Hoặc (có 2 dấu |)
!	NOT	Phủ định

Bảng chân trị của các toán tử luận lý

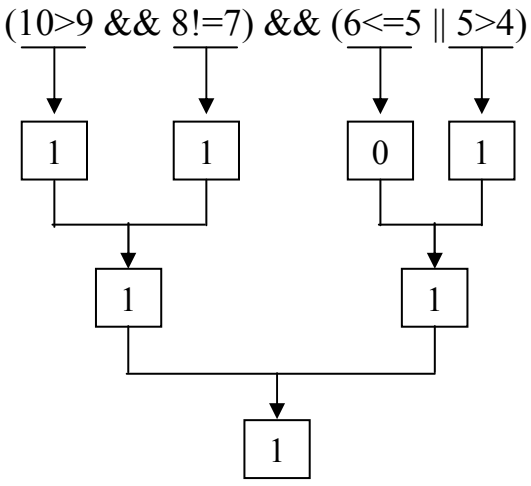
p	**q**	**p && q**	**p || q**	**!p**
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

Cả hai toán tử quan hệ và luận lý có độ ưu tiên thấp hơn toán tử số học. Trong một biểu thức có thể có nhiều loại toán tử, thứ tự để định trị biểu thức dựa vào độ ưu tiên của toán tử. Để thay đổi thứ tự định trị trong một biểu thức, dùng cặp dấu ngoặc đơn () như trong các biểu thức số học.

Độ ưu tiên của toán tử quan hệ và luận lý:

Cao nhất	**!**
	> >= < <=
	== !=
	&&
Thấp nhất	**||**

Ví dụ: biểu thức (10>9 && 8!=7) || (6<=5 || 5>4) được định trị như sau:



Kết quả định trị biểu thức trên là 1 (true).

13. Toán tử ? *(? operator)*

Toán tử ? là một toán tử ba ngôi do đó có ba toán hạng. Dạng tổng quát của toán tử ? là:

Exp1 ? Exp2 : Exp3;

Exp1, Exp2, và Exp3 là các biểu thức.

Toán tử ? làm việc như sau: Đầu tiên Exp1 được định trị. Nếu kết quả là true thì Exp2 được định trị và giá trị của nó trở thành giá trị của cả biểu thức. Nếu Exp1 là false thì Exp3 được định trị và giá trị của nó trở thành giá trị của cả biểu thức.

Ví dụ:

```
x = 10;
y = x>9 ? 100*x : 200*x;
```

Vì x>9 là true nên giá trị của biểu thức sẽ là 1000. Vậy y sẽ có giá trị là 1000

14. Toán tử sizeof

sizeof là toán tử một ngôi mà trả về số byte của kiểu dữ liệu chiếm trong bộ nhớ. Mỗi môi trường (hệ điều hành, loại CPU, ...) dùng số byte khác nhau cho mỗi kiểu dữ liệu. Dạng tổng quát của toán tử sizeof

sizeof(operand)

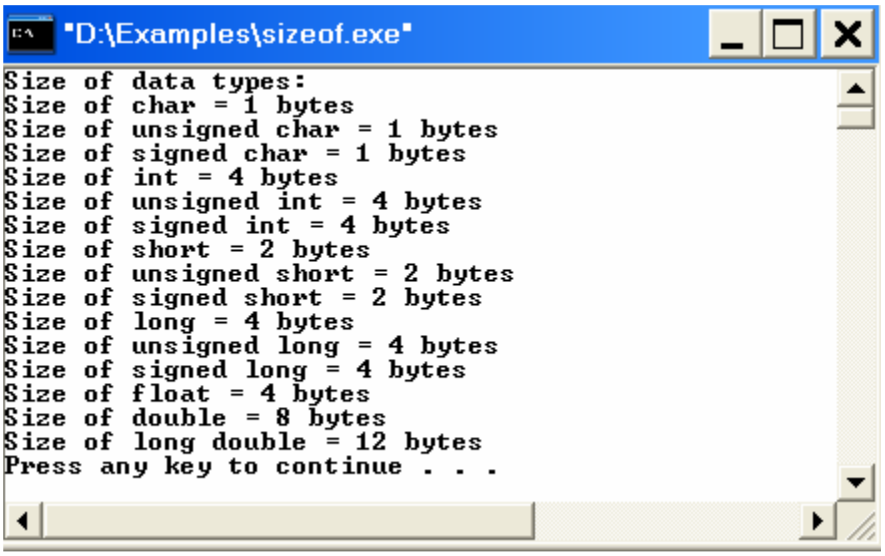
operand: có thể là tên kiểu dữ liệu, biến, biểu thức.

Ví dụ sau cho biết số byte của mỗi kiểu dữ liệu

```
#include <iostream.h>
void main(void)
{
cout <<"Size of data types:\n";
cout <<"Size of char = " << sizeof(char) << " bytes\n";
cout <<"Size of unsigned char = " << sizeof(unsigned char) << " bytes\n";
cout <<"Size of signed char = " << sizeof(signed char) << " bytes\n";
cout <<"Size of int = " << sizeof(int) << " bytes\n";
cout <<"Size of unsigned int = " << sizeof(unsigned int) << " bytes\n";
cout <<"Size of signed int = " << sizeof(signed int) << " bytes\n";
cout <<"Size of short = " << sizeof(short) << " bytes\n";
cout <<"Size of unsigned short = " <<sizeof(unsigned short)<< " bytes\n";
```

```
cout <<"Size of signed short = " << sizeof(signed short) << " bytes\n";
cout <<"Size of long = " << sizeof(long) << " bytes\n";
cout <<"Size of unsigned long = " << sizeof(unsigned long) << " bytes\n";
cout <<"Size of signed long = " << sizeof(signed long) << " bytes\n";
cout <<"Size of float = " << sizeof(float) << " bytes\n";
cout <<"Size of double = " << sizeof(double) << " bytes\n";
cout <<"Size of long double = " << sizeof(long double) << " bytes\n";
}
```

Kết quả khi thực hiện chương trình trên:



15. Toán tử dấu phẩy (*comma operator*)

Toán tử comma buộc các biểu thức cùng với nhau. Biểu thức bên trái của toán tử comma luôn luôn được định trị như void, biểu thức bên phải được định trị và trở thành giá trị của biểu thức. Dạng tổng quát của toán tử comma:

(exp_1, exp_2, ..., exp_n)

Các biểu thức được định trị từ trái sang phải, biểu thức cuối cùng (exp_n) được định trị và trở thành giá trị của toàn bộ biểu thức.

Ví dụ:

```
x = (y=3, y+1);
```

Đầu tiên y được gán giá trị 3 và rồi x được gán giá tri y+1 là 4.

Bảng tóm tắt độ ưu tiên của các toán tử

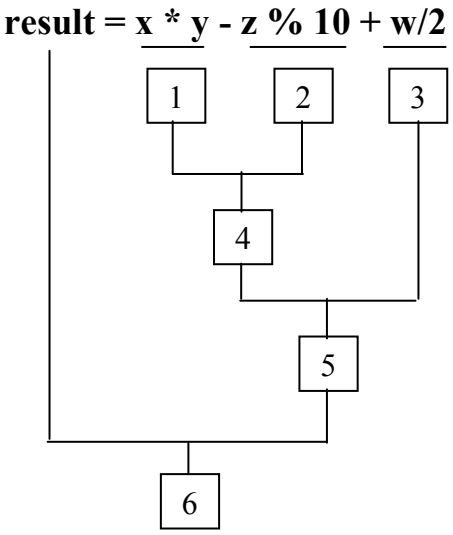
Cao nhất	**() [] -> .**
	! ~ ++ -- (type) * & sizeof
	*** / %**
	+ -
	<< >>
	< <= > >=
	== !=
	&
	^
	|
	&&
	||
	?:
	= += -= *= /= %=
Thấp nhất	**,**

16. Biểu thức (*expressions*)

Một biểu thức trong C/C++ là sự kết hợp của các thành phần như toán tử, hằng, biến, và hàm có trả về giá trị.

Thứ tự định trị của biểu thức tùy thuộc vào độ ưu tiên của các toán tử. Do đó, để viết biểu thức rõ ràng và thực hiện việc định trị đúng theo yêu cầu của lập trình viên ta nên dùng cặp dấu ngoặc tròn () để bao quanh các biểu thức con của biểu thức.

Ví dụ: Định trị biểu thức sau: result = x * y - z % 10 + w/2;



Để viết lại biểu thức trên rõ ràng dễ đọc và thực hiện như mong muốn, ta viết:

```
result = (x * y) - (z % 10) + (w/2);
```

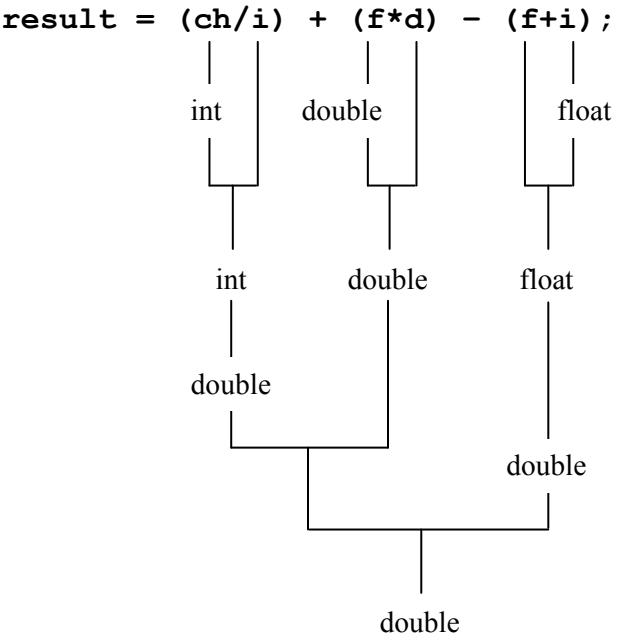
17. Chuyển đổi kiểu trong các biểu thức

Khi các hằng và biến của những kiểu khác nhau tồn tại trong một biểu thức, giá trị của chúng phải được chuyển thành cùng kiểu trước khi các phép toán giữa chúng được thực hiện. Trình biên dịch sẽ thực hiện việc chuyển kiểu (*convert*) tự động đến kiểu của toán hạng có kiểu lớn nhất. Việc chuyển kiểu này gọi là thăng cấp kiểu (*type promotion*).

Ví dụ:

```
char ch;
```

```
int i;
float f;
double d;
```



18. Ép kiểu (*casting*)

Casting dùng để ép một biểu thức thành một kiểu theo ý muốn của lập trình viên. Dạng tổng quát của casting là

(type)expression *hoặc* ***type(expression)***

type: là tên một kiểu dữ liệu hợp lệ.

Ví dụ:

```
float result;
result = 7/2;
```

Do 7/2 là phép chia nguyên nên kết quả không có phần thập phân. Sau lệnh trên result có giá trị là 3.

Để phép chia trên là phép chia số thực dù rằng toán hạng có kiểu nguyên ta thực hiện ép kiểu tử số hoặc mẫu số hoặc cả hai. Các cách viết sau sẽ cho cùng kết qủa:

```
result = (float)7/2;
result = 7/(float)2;
result = (float)7/(float)2;
result = float(7)/float(2);
```

Như vậy, trong các lệnh trên, ta đã ép một hoặc 2 toán hạng đến kiểu float. Do đó, để định trị biểu thức, toán hạng kia sẽ được thăng cấp (*type promotion*) thành kiểu float tương ứng để thực hiện phép chia sau đó. Kết quả của 4 lệnh trên là tương đương và result sẽ có giá trị là 3.5

19. Dạng viết tắt của câu lệnh gán (*shorthand assignments*)

Các dạng viết tắt của câu lệnh gán với các toán tử số học gồm +=, -=, *=, /=, và %=. Những lệnh có dạng:

variable = variable operator expression;

variable: Tên biến

operator: Toán tử số học (+, -, *, /, %)

expression: Biểu thức

có thể được viết dưới dạng ngắn gọn hơn như sau:

variable operator= expression;

Ví dụ:

```
x = x + 10; ⇔ x += 10;
x = x – 10; ⇔ x -= 10;
x = x * 10; ⇔ x *= 10;
x = x / 10; ⇔ x /= 10;
x = x % 10; ⇔ x %= 10;
```

Bài tập Chương 2

1. Nhập bán kính đường tròn r. Tính và xuất chu vi, diện tích đường tròn tương ứng.

HD: cv=2*π*r và dt=π*r^2

2. Nhập cạnh a. Tính và xuất chu vi, diện tích hình vuông.

HD: cv=4*a và dt=a^2

3. Nhập cạnh a,b. Tính và xuất chu vi, diện tích hình chữ nhật.

HD: cv=2*(a+b) và dt=a*b

4. Nhập cạnh a,h1,h2. Tính và xuất chu vi, diện tích hình thoi.

HD: cv=4*a và dt=1.0/2*h1*h2

5. Nhập cạnh a,b,c,d,h. Tính và xuất chu vi, diện tích hình thang.

HD: cv=(a+b+c+d) và dt=1.0/2*h*(a+b)

6. Nhập cạnh a,b,c. Tính và xuất chu vi, diện tích hình tam giác.

HD: cv=a+b+c và dt=$\sqrt{p*(p-a)*(p-b)*(p-c)}$ với p=cv/2

7. Nhập vào hai số nguyên dương a,b. Tính và xuất tổng, hiệu, tích, thương.

8. Nhập 2 số nguyên a,b. Tính và xuất a^b.

HD: dùng hàm **pow(x,y) → x^y**

9. Nhập 1 số n. Tính và xuất giá trị tuyết đối.

HD: dùng hàm **abs(a) → |a|**

10. Nhập 1 số n. Tính và xuất căn bậc hai của n.

HD: dùng hàm **sqrt(a) →** $\sqrt{a}$

11. Nhập 1 góc x. Tính và xuất sinx, cosx, tgx, cotgx.

HD: các hàm sin, cos, tan chỉ tính theo đơn vị radian nên chúng ta phải đổi từ độ x sang độ radian t như sau: t=x*π/180 => sinx=sin(t), cosx=cos(t), tgx=tan(t), cotgx=1/tgx

12. Nhập toạ độ 2 điểm A(x_A,y_A), B(x_B,y_B). Tính và xuất độ dài đoạn AB.

HD: |AB|=$d_{AB}=\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2)}$

13. Nhập toạ độ 2 điểm $A(x_A,y_A)$, $B(x_B,y_B)$. Tính hệ số góc của đường thẳng đi qua hai điểm đó theo công thức:

Hệ số góc = $(y_B - y_A) /(x_B - x_A)$

14. Viết chương trình nhập vào số nguyên n và số thực x. Tính và in ra biểu thức $A = (x^2 + x + 1)^n + (x^2 - x + 1)^n$

15. Nhập vào số giây bất kỳ **t**. Tính và xuất ra dạng **Giờ:Phút:Giây**

Ví dụ: Nhập **3750** thì xuất ra **1:2:30 AM**

16. Nhập **51100** thì xuất ra **2:11:40 PM**

HD: hour=(t/3600)%24

minute=(t%3600)/60

second=(t%3600)%60

17. Viết chương trình nhập vào ngày, tháng, năm. In ra ngày tháng năm theo dạng dd/mm/yy. (dd: ngày, mm: tháng, yy : năm. Ví dụ: 20/11/99)

18. Viết chương trình tính logax với a, x là các số thực nhập vào từ bàn phím, và x>0, a>0, a != 1.(dùng logax=lnx/lna)

19. Viết chương trình nhập vào một ký tự:

In ra mã ASCII của ký tự đó.
In ra ký tự kế tiếp của nó.

20. Viết chương trình nhập vào điểm ba môn Toán, Lý, Hóa của một học sinh. In ra điểm trung bình của học sinh đó với hai số lẻ thập phân.

21. Viết chương trình đảo ngược một số nguyên dương có đúng 3 ký số.

Chương 3

CÁC CẤU TRÚC ĐIỀU KHIỂN (Control structures)

1. GIỚI THIỆU

Tất cả các chương trình máy tính dù đơn giản hay phức tạp đều được viết bằng cách sử dụng các cấu trúc điều khiển. Có 3 loại cấu trúc điều khiển cơ bản là cấu trúc tuần tự (*sequence*), cấu trúc lựa chọn (*selection*), và cấu trúc lặp (*repetition or loop*). Các cấu trúc này điều khiển thứ tự thực thi các lệnh của chương trình.

Cấu trúc tuần tự: thực hiện các lệnh theo thứ tự từ trên xuống dưới.
Cấu trúc lựa chọn: dựa vào kết quả của biểu thức điều kiện. Tùy theo sự định trị của biểu thức này mà những lệnh tương ứng sẽ được thực hiện. Các cấu trúc lựa chọn gồm cấu trúc if, switch.
Cấu trúc lặp: lặp lại 1 hay nhiều lệnh cho đến khi biểu thức điều kiện là sai. Các cấu trúc lặp gồm for, while, do ... while.

Tuy nhiên, thứ tự thực hiện các lệnh của chương trình còn bị chi phối bởi các lệnh nhảy như continue, break, goto.

Lệnh (*statement*): một biểu thức kết thúc bởi 1 dấu chấm phẩy gọi là 1 lệnh.
Ví dụ:

```
int a,b,c;
a=10;
a++;
```

Khối lệnh (*block*): một hay nhiều lệnh được bao quanh bởi cặp dấu {} được gọi là một khối lệnh. Về mặt cú pháp, khối lệnh tương đương 1 câu lệnh đơn. Do đó nơi đâu xuất hiện 1 lệnh thì nơi đó có thể xuất hiện 1 khối lệnh.
Ví dụ:

```
if (a<b)
{
      temp=a;
      a=b;
      b=temp;
}
```

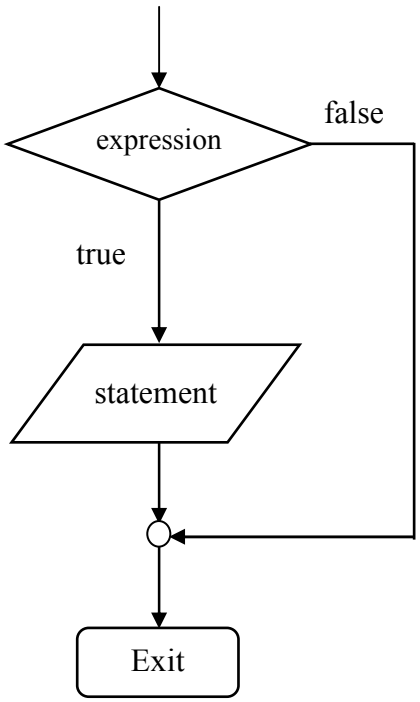
2. CẤU TRÚC LỰA CHỌN IF

Cấu trúc if có các dạng tổng quát sau:

Dạng 1: `if (expression)`
`statement;`

expression: Biểu thức luận lý (có giá trị là true hay false)
statement: Câu lệnh

Lưu đồ cú pháp:



Ý nghĩa: Đầu tiên ***expression*** được định trị. Nếu kết quả là true (<>0) thì ***statement*** được thực thi. Ngược lại, không làm gì cả.

Ví dụ 1: Viết chương trình nhập vào một số nguyên a. In ra màn hình kết quả kiểm tra a có phải là 1 số dương không.

```
#include <iostream.h>
#include <conio.h>
int main()
{     int a;
      cout << "Input a = ";
      cin>>a;
      if(a>=0)
            cout << a << " is a positive.”;
      getch();
      return 0;
}
```

Giải thích:

- Nếu nhập một số a >=0 thì câu lệnh

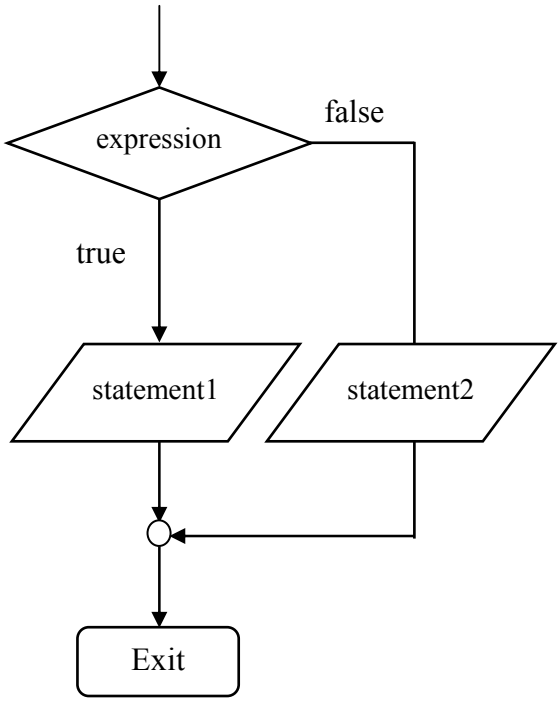
  cout << a << " is a positive.";

  sẽ được thực hiện, ngược lại câu lệnh này không được thực hiện.
- Lệnh getch() luôn luôn được thực hiện vì nó không bị ảnh hưởng bởi câu lệnh if.

Dạng 2:

```
if (expression)
     statement1;
else
     statement2;
```

Lưu đồ cú pháp:



Ý nghĩa: Đầu tiên ***expression*** được định trị. Nếu kết quả là true (<>0) thì ***statement1*** được thực thi. Ngược lại, thì ***statement2*** được thực thi.

Ví dụ 1: Viết chương trình nhập vào một số nguyên a. In ra màn hình kết quả kiểm tra a là số âm hay dương.

```
#include <iostream.h>
#include <conio.h>
int main()
{
      int a;
      cout << "Input a = ";
      cin >> a;
      if(a>=0)
            cout << a << " is a positive.”;
      else
            cout << a << " is a negative.”;
      getch();
      return 0;
}
```

Ví dụ 2:

Viết chương trình nhập vào một số nguyên dương là tháng trong năm và in ra số ngày của tháng đó. Biết rằng:

- Tháng có 31 ngày: 1, 3, 5, 7, 8, 10, 12
- Tháng có 30 ngày: 4, 6, 9, 11
- Tháng có 28 hoặc 29 ngày : 2

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int m;
  cout << "Input month: ";
  cin >> m;
  if(m==1 || m==3 || m==5 || m==7 || m==8 || m==10 || m==12)
    cout << "Month ‟ << m << ” has 31 days.";
  else
    if(m==4 || m==6 || m==9 || m==11)
      cout << "Month ‟ << m << ” has 30 days.";
    else
      if(m==2)
        cout << "Month ‟ << m << ” has 28 or 29 days.";
      else
        cout << "This is not a valid month: ” << m;
  getch();
}
```

Lưu ý:

- Ta có thể sử dụng các câu lệnh if…else lồng nhau. Trong trường hợp if…else lồng nhau thì *else sẽ kết hợp với if gần nhất chưa có else*.
- Trong trường hợp câu lệnh if "bên trong" không có else thì phải viết nó trong cặp dấu {} (coi như là khối lệnh) để tránh sự kết hợp else if sai.

Ví dụ:

```
if (ch >= '0' && ch <= '9')
    kind = digit;
else
{
    if (ch >= 'A' && ch <= 'Z')
          kind = upperLetter;
    else
    {
          if (ch >= 'a' && ch <= 'z')
                kind = lowerLetter;
          else
                kind = special;
    }
}
```

Để cho dễ đọc có thể sử dụng hình thức sau:

```
if (ch >= '0' && ch <= '9')
    kind = digit;
else if (ch >= 'A' && ch <= 'Z')
    kind = capitalLetter;
else if (ch >= 'a' && ch <= 'z')
    kind = smallLetter;
else
    kind = specialLetter;
```

3. CẤU TRÚC LỰA CHỌN switch

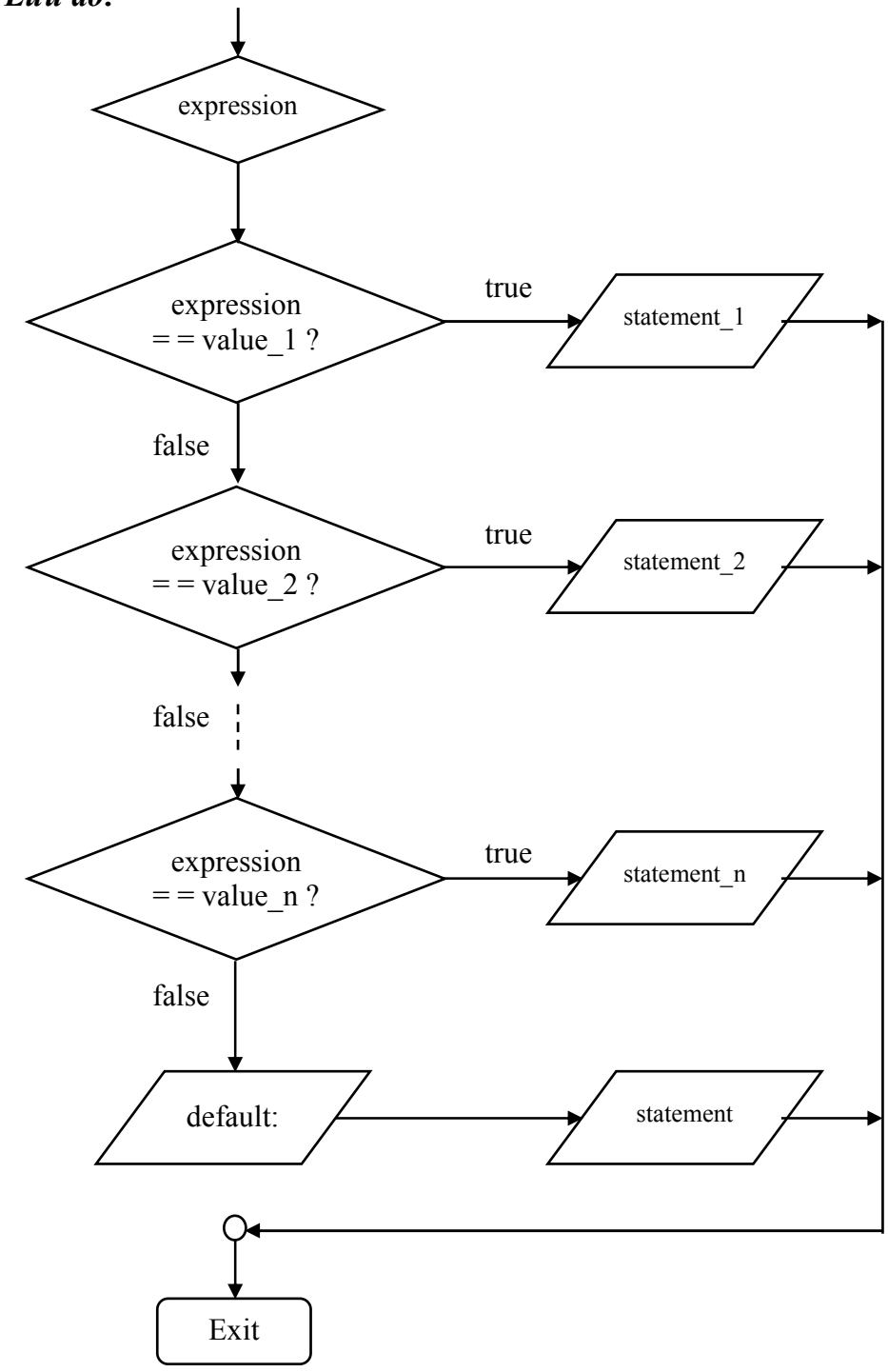
Cấu trúc switch là một cấu trúc lựa chọn có nhiều nhánh. Khi có nhiều sự lựa chọn thi đây là cấu trúc phù hợp thay vì phải dùng một chuỗi lệnh if..else lồng nhau.

Dạng tổng quát của cấu trúc switch:

Cú pháp:

```
switch(expression)
{
      case value_1:
      statement_1;
      [break;]
      …
      case value_n:
      statement_n;
      [break;]
      [default :
      statement;]
}
```

Lưu đồ:


expression
expression
= = value_1 ?
true
statement_1
false
expression
= = value_2 ?
true
statement_2
false
expression
= = value_n ?
true
statement_n
false
default:
statement
Exit

***Giải thích*:**

- Trước hết chương trình sẽ định trị của ***expression***.
- Nếu giá trị của ***expression*** bằng ***value_1*** thì thực hiện ***statement_1*** rồi thoát.
- Nếu giá trị của ***expression*** khác ***value _1*** thì so sánh với ***value _2***, nếu bằng ***value_2*** thì thực hiện ***statement_2*** rồi thoát…., so sánh tới ***value_n***.
- Nếu tất cả các phép so sánh trên đều sai thì thực hiện ***statement*** của trường hợp ***default***.

Lưu ý:

- ***expression*** trong switch() phải có kết quả là giá trị kiểu số nguyên (int, char, long).
- Các giá trị sau case phải là hằng nguyên.
- Không bắt buộc phải có default.
- Thông thường mỗi case có 1 câu lệnh break. Khi thực hiện lệnh tương ứng của case có giá trị bằng ***expression***, chương trình thục hiện lệnh break để thoát khỏi cấu trúc switch.

***Ví dụ 1*:** Nhập vào một số nguyên, chia số nguyên này cho 2 lấy phần dư. Kiểm tra nếu phần dư bằng 0 thì in ra thông báo "là số chẵn", nếu số dư bằng 1 thì in thông báo "là số lẻ".

```
#include <iostream.h>
#include <conio.h>
void main ()
{
      clrscr();
      int n, remainder;
      cout<<"Input an number: "; cin>>n;
      remainder = (n % 2);
      switch(remainder)
      {
            case 0: cout << n << " is an even.";
            break;
            case 1: cout << n << " is an odd."; break;
      }
      getch();
}
```

***Ví dụ 2*:**

Nhập vào 2 số nguyên và 1 phép toán.

- Nếu phép toán là '+', '-', '*' thì in ra kết qua là tổng, hiệu, tích của 2 số.
- Nếu phép toán là '/' thì kiểm tra xem số thứ 2 có khác không hay không? Nếu khác không thì in ra thương của chúng, ngược lại thì in ra thông báo "Cannot divide by zero!".
-

```
#include <iostream.h>
#include <conio.h>
int main()
{
   int a, b;
   char operation;
   clrscr();
   cout << "Input two numbers: ";
   cin >> a >> b;
   fflush(stdin);// Clear buffer
   cout << "Input operation(+-*/): ";
   cin >> operation;
   switch(operation)
   {
      case '+':
         cout << a << " + " << b << " =" << a+b;
         break;
      case '-':
         cout << a << " - " << b << " =" << a-b;
         break;
      case '*':
         cout << a << " * " << b << " =" << a*b;
         break;
      case '/':
         if(b!=0)
               cout << a << " / " << b << " =" << (float)a/b;
         else
               cout << "Cannot divide by zero!";
         break;
   }
   getch();
}
```

***Ví dụ 3*:**
Yêu cầu người thực hiện chương trình nhập vào một số nguyên dương là tháng trong năm và in ra số ngày của tháng đó.
- Tháng có 31 ngày: 1, 3, 5, 7, 8, 10, 12
- Tháng có 30 ngày: 4, 6, 9, 11
- Tháng có 28 hoặc 29 ngày : 2
- Nếu nhập vào số <1 hoặc >12 thì in ra câu thông báo “There is no month like this.“.

```
#include <iostream.h>
#include <conio.h>
int main ()
{
  int month;
  clrscr();
  cout << "Input month: ";
  cin >> month;
  switch(month)
  {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
      cout<<"Month ” << month << ” has 31 days.";
      break;
    case 4:
    case 6:
    case 9:
    case 11:
      cout << "Month ” << month << ” has 30 days.";
      break;
    case 2:
      cout << "Month ‟ << month << ‟ has 28 or 29 days.";
      break;
    default :
      cout<<”There is no month like this.”;
  }
  getch();
}
```

4. CÁC CẤU TRÚC LẶP (Loop structures)

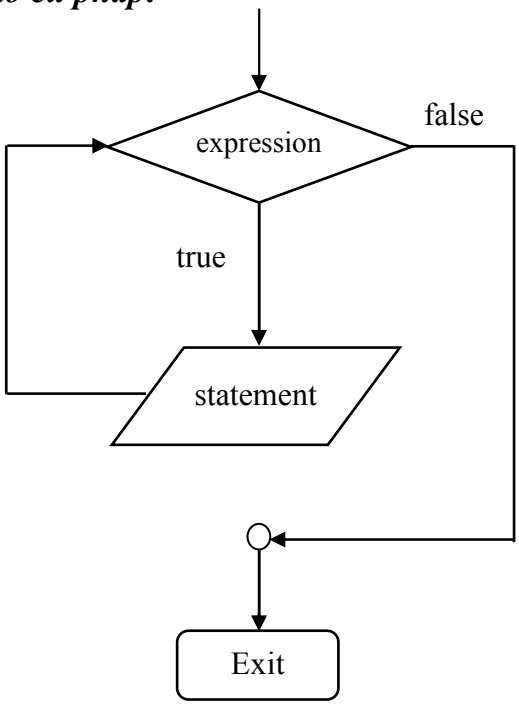
Cấu trúc lặp cho phép lặp lại nhiều lần 1 câu lệnh hay 1 khối lệnh nào đó cho đến khi biểu thức điều kiện còn thỏa.

4.1. Cấu trúc while

Cấu trúc ***while*** cho phép thực hiện ***statement*** trong khi ***expression*** vẫn còn đúng.

***Cú pháp*:**
```
while(expression)
    statement;
```

Lưu đồ cú pháp:



Bước 1: ***expression*** được định trị.
Bước 2: Nếu kết quả là true thì thực hiện ***statement*** rồi quay lại bước 1.
Bước 3: Nếu kết quả là false thì thoát khỏi vòng lặp while.

Ví dụ 1: Viết chương trình tính tổng các số nguyên từ 1 tới n.

```
#include<iostream.h>
#include<conio.h>
void main ()
{
      clrscr();
      int i, n, sum;
      cout<<”Input n= ”;
      cin >> n;
      i = 1;
      sum = 0;
      while(i<=n)
      {
            sum += i;
            i++;
      }
      getch();
}
```

Ví dụ 2: Viết đoạn chương trình in dãy số nguyên từ 1 đến 10.

```
#include <iostream.h>
#include <conio.h>
void main()
{
      int i;
      clrscr();
      cout<<”Displat one to ten: ";
      i=1;
      while (i<=10)
      {
            cout<< setw(3) << i;
            i++;
      }
      getch();
}
```

Lưu ý: vòng lặp phải kết thúc ở một điểm nào đó, vì vậy bên trong vòng lặp phải cung cấp một cách thức nào đó để buộc ***expression*** trở thành false nếu không thì sẽ lặp vô tận. Trong ví dụ trên i++; là cách thức tăng biến i để đến khi i=11 thì vòng lặp kết thúc.

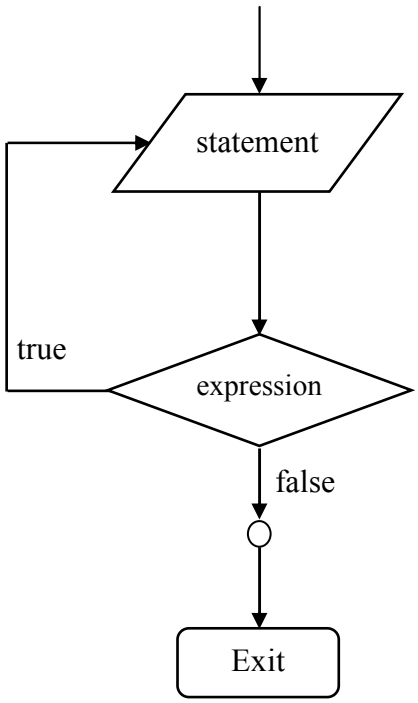
4.2. Cấu trúc lặp do … while

Cấu trúc lặp **do … while** giống như vòng lặp while, dùng để lặp lại một ***statement*** trong khi ***expression*** là true. ***statement*** luôn luôn được thực hiện ít nhất 1 lần.

Cú pháp:

```
do
{
   statement;
}while(expression);
```

Lưu đồ:



Bước 1: ***statement*** được thực hiện
Bước 2: ***expression*** được định trị.
Bước 3: Nếu ***expression*** là true thì quay lại bước 1
Bước 4: Nếu ***expression*** là false thì thoát khỏi do ... while.

Ví dụ 1: Viết đoạn chương trình in dãy số nguyên từ 1 đến 10.

```
#include <iostream.h>
#include <conio.h>
void main ()
{
      int i;
      clrscr();
      cout<<"Display one to ten: ";
      i=1;
      do
      {
            cout << setw(3) << i;
            i+=1;
      } while(i<=10);
      getch();
}
```

Ví dụ 2: Viết chương trình nhập vào một số nguyên n. Tính tổng của các số nguyên từ 1 đến n.

```
#include <iostream.h>
#include <conio.h>
void main ()
{
   unsigned int n,i,sum;
   clrscr();
   cout<<”Input a positive number: ";
   cin >> n;
   sum=0;
   i=1;
   do
   {
         sum+=i;
         i++;
   } while(i<=n);
   cout << ”Sum from 1 to ” << n << ”=” << sum;
   getch();
}
```

4.3. Cấu trúc lặp for

Chức năng chính của cấu trúc lặp for là lặp lại một đoạn lệnh nào đó trong khi ***Exp2*** còn là true. Cấu trúc lặp for thường sử dụng trong những chương trình mà số lần lặp lại một đoạn lệnh nào đó được biết trước.

Cú pháp:

for (Exp1*; Exp2; Exp3*)
statement;

Cách hoạt động của vòng lặp for như sau:

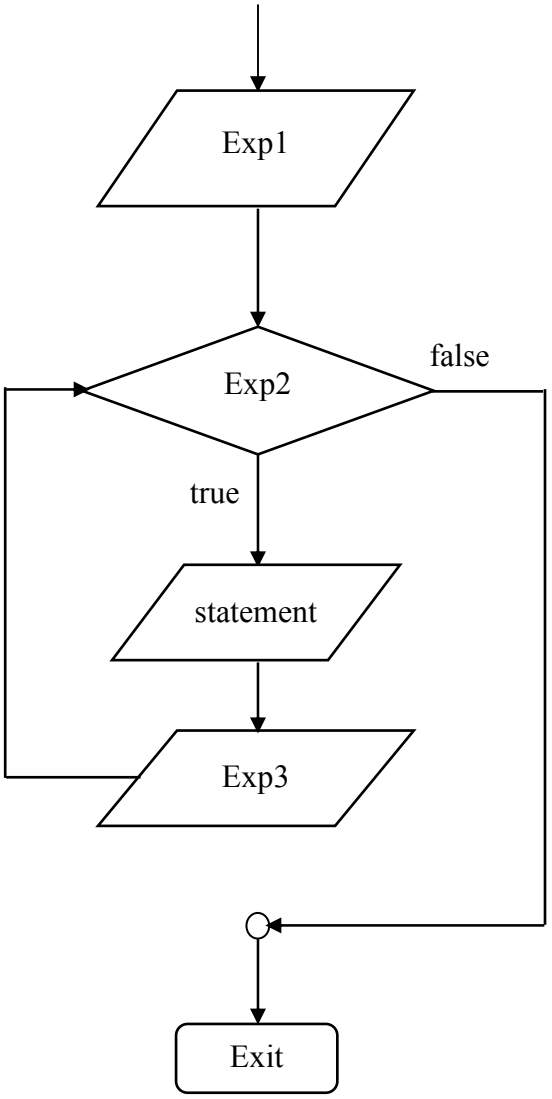
Bước 1: ***Exp1*** *là* biểu thức khởi tạo được thực hiện. Thông thường nó gán giá trị khởi tạo cho biến điều khiển cấu trúc for. Biểu thức này chỉ được thực hiện chỉ 1 lần.

Bước 2: ***Exp2*** *là* biểu thức điều kiện được định trị.

Bước 3: Nếu giá trị là true thì ***statement*** sẽ được thực thi, ***Exp3*** được thực thi. Quay lại bước 2

Bước 4: Nếu giá trị là false thì thoát khỏi cấu trúc for.

Lưu đồ cú pháp:



Ví dụ 1: Viết chương trình tính tổng của các số nguyên từ 1 tới n.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int i, n, sum;
  cout<<"Input a number:";
  cin >> n;
  sum = 0;
  for (i=1 ; i<=n ; i++)
    sum += i;
  cout<<"Sum from 1 to " << n << " is: " << sum;
  getch();
}
```

– C/C++ cho phép biểu thức đầu tiên trong vòng lặp for là một định nghĩa biến. Ví dụ trong vòng lặp trên thì i có thể được định nghĩa bên trong vòng lặp:

```
for(int i=1; i<=n; ++i)
      sum += i;
```

– Bất kỳ biểu thức nào trong 3 biểu thức của vòng lặp for đều có thể rỗng. Ví dụ, xóa biểu thức đầu và biểu thức cuối cho chúng ta dạng giống như vòng lặp while:

```
for(; i != 0;) statement;
//tương đương với while (i != 0) statement;
```

– Xóa tất cả các biểu thức cho chúng ta một vòng lặp vô tận.

```
for (;;)        // vòng lặp vô hạn
      statement;
```

5. CÁC LỆNH RẼ NHÁNH VÀ LỆNH NHẢY

5.1. Lệnh break

Lệnh break thường dùng trong phần case của cấu trúc switch để thoát khỏi cấu trúc switch sau khi các lệnh tương ứng của case đã được thực hiện. Ngoài ra, trong các cấu trúc lặp, nếu muốn thoát khỏi 1 vòng lặp tức thì mà không chờ cho đến khi biểu thức điều kiện (*conditional expression)* của cấu trúc được định trị là false, ta dùng lệnh break. Khi break được thực hiện bên trong 1 cấu trúc lặp, điều khiển *(control flow)* tự động nhảy đến lệnh đầu tiên ngay sau cấu trúc lặp đó. Lệnh break thường liên đới với một câu lệnh if trong những trường hợp này. Không sử dụng lệnh break bên ngoài các cấu trúc lặp như while, do...while, for hay cấu trúc switch.

Ví dụ 1:

Đọc vào một mật khẩu người dùng tối đa attempts lần

```
for (i=0; i<attempts ; ++i)
{
   cout << "Input a password: ";
   cin >> passWord;
   if (check(passWord)) // kiểm tra mật khẩu đúng hay sai
         break;         // thoát khỏi vòng lặp
   cout << "Password is wrong!\n";
}
```

Ở đây giả sử rằng có một hàm ***check*** để kiểm tra một mật khẩu và trả về 1 (true) nếu như mật khẩu đúng và ngược lại là 0 (false).

Ví dụ 2:

Viết chương trình tính tổng các số nguyên được nhập từ bàn phím, chương trình được kết thúc khi nhập số âm.

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

```
void main()
{
      int  num,sum=0;
      clrscr();
      while(1)
      {
            cout <<"Input a number (negative for exit): ";
            cin >> num;
            if (num < 0)  break;
            sum += num;
      }
      cout<<"\nSum all inputs is " << sum;
      getch();
}
```

Chương trình trên nếu không dùng break có thể được viết lại như sau:

```
do
{
      cout <<"Input a number (negative for exit): ”;
      cin >> num;
      if (num>=0)
               sum += num;
}while (num>=0);
```

5.2. Lệnh continue

Lệnh continue chỉ được dùng trong thân các cấu trúc lặp như for, while, do...while. Trong mỗi lần lặp của các cấu trúc trên, các lệnh trong thân vòng lặp được thực hiện rồi điều khiển sẽ quay về đầu vòng lặp chuẩn bị cho lần lặp kế tiếp. Tuy nhiên, nếu muốn điều khiển quay về đầu vòng lặp ngay lập tức mà không thực hiện các lệnh còn lại của lần lặp hiện hành thì ta dùng câu lệnh continue. Câu lệnh continue thường đi kèm với 1 câu lệnh if.

Ví dụ: một vòng lặp thực hiện đọc một số, xử lý nó nhưng bỏ qua những số âm, và dừng khi số là 0, có thể diễn giải như sau:

```
do
{
      cin >> num;
      if (num < 0) continue;
      // process num here
} while(num != 0);
```

Điều này tương đương với:

```
do
{
      cin >> num;
      if(num >= 0)
      {
            // process num here
      }
} while(num != 0);
```

– Một biến thể của vòng lặp này là để đọc chính xác một số *n* lần có thể được diễn giải như sau:

```
for(i=0; i<n; i++)
{
   cin >> num;
   if(num<0) continue; // jump to i++
         // process num here
}
```

Khi lệnh ***continue*** xuất hiện bên trong các cấu trúc lặp lồng nhau thì nó chỉ liên đới đến cấu trúc lặp trực tiếp chứa nó.

Ví dụ: trong một tập các cấu trúc lặp được lồng nhau sau đây, lệnh **continue** liên đới với cấu trúc lặp for và không liên đới với cấu trúc lặp while:

```
while(more)
{
      for (i = 0; i<n; i++)
      {
            cin >> num;
            if (num < 0)
                  continue; // jump to i++
            // process num here...
      }
      //etc...
}
```

BÀI TẬP CHƯƠNG 3

1. Nhập 1 số n>=0. Tính và xuất căn bậc hai của n.

   HD: dùng hàm **sqrt(a)=** $\sqrt{a}$

2. Nhập vào số giây bất kỳ **t>=0**. Tính và xuất ra dạng **Giờ:Phút:Giây**

   Ví dụ: Nhập **3750** thì xuất ra **1:2:30 AM**

   Nhập **51100** thì xuất ra **2:11:40 PM**

   HD: hour=(t/3600)%24

   minute=(t%3600)/60

   second=(t%3600)%60

3. Nhập 3 số thực a, b, c. Tìm số lớn nhất.
4. Nhập n. Kiểm tra n là số chẵn hay số lẻ.
5. Nhập 2 số a, b. Kiểm tra xem chúng có cùng dấu hay không.
6. Nhập vào hai số nguyên dương a, b. So sánh giá trị của chúng (lớn hơn, nhỏ hơn, bằng).
7. Giải và biện luận phương trình bậc 1: ax+b=0.
8. Giải và biện luận phương trình bậc 2: $ax^2+bx+c=0$.
9. Nhập vào tháng t (với 1<=t<=12). Cho biết t thuộc qúy mấy trong năm.
10. Nhập vào tháng t (với 1<=t<=12). Cho biết tháng t có bao nhiêu ngày. Riêng tháng 2 thì phải kiểm tra năm nhuận (Năm nhuận là năm chia hết cho 4 mà không chia hết cho 100, hoặc chia hết cho 400).
11. Nhập vào một ngày (*ngày, tháng, năm*). Tìm ngày kế sau ngày vừa nhập (*ngày/tháng/năm*).
12. Nhập vào một ngày (*ngày, tháng, năm*). Tìm ngày kế trước ngày vừa nhập (*ngày/tháng/năm*).

13. Nhập vào một ngày (*ngày, tháng, năm*). Cho biết ngày đó là ngày thứ bao nhiêu trong năm.
14. Nhập vào một năm dương lịch. Hãy cho biết năm âm lịch. (vd: n=2007 => Đinh Hợi)
15. Nhập một số n có tối đa 2 chữ số. Hãy cho biết cách đọc ra dạng chữ.

    (vd: n=35 => Ba mươi lăm, n=5 => năm).
16. Nhập một số n có tối đa 3 chữ số. Hãy cho biết cách đọc ra dạng chữ.

    (vd: n=235 => Hai trăm ba mươi lăm, n=305 => Ba trăm lẻ năm)
17. Nhập một số n bất kỳ. Hãy cho biết cách đọc ra dạng chữ.
18. Nhập vào điểm Toán, Lý, Hoá. Hãy tính ĐTB và Cho biết sinh viên đó xếp loại gì (Xuất sắc, Giỏi, Khá, Trung bình, Yếu)..
19. Kiểm tra số nguyên dương n có phải là *số chính phương* hay không?
20. Viết chương trình nhập vào một số nguyên dương n với 1<=n<=7. Tùy theo n=1,2,...,7 hãy in tương ứng các từ (Sunday, Monsday, Tuesday,.... , Saturday) ra màn hình.
21. Nhập vào số Kwh tiêu thụ điện. Tính tiền điện phải trả biết rằng cách thức tính tiền theo qui định như sau:
    a. 100 kwh định mức đầu tiên có đơn giá trung bình là 600đ/kwh
    b. Các kwh từ 101 đến 150 có đơn giá là 700đ/kwh
    c. Các kwh từ 151 đến 200 có đơn giá là 900đ/kwh
    d. Các kwh từ 201 trở đi có đơn giá là 1100đ/kwh
22. Nhập cạnh a>=0, b>=0, c>=0. Nếu a, b, c tạo thành tam giác thì hãy tính và xuất chu vi, diện tích hình tam giác. Ngược lại, thông báo "Không tạo thành tam giác"

    HD: cv=a+b+c, p=cv/2 và dt=$\sqrt{p*(p-a)*(p-b)*(p-c)}$
23. Viết chương trình nhập từ bàn phím 2 số a, b và một ký tự ch.

    Nếu:

- ch là "+" thì thực hiện phép tính a + b và in kết quả lên màn hình.
- ch là "–" thì thực hiện phép tính a - b và in kết quả lên màn hình.
- ch là "*" thì thực hiện phép tính a * b và in kết quả lên màn hình.
- ch là "/" thì thực hiện phép tính a / b và in kết quả lên màn hình.

24. Một số nguyên dương chia hết cho 3 nếu tổng các chữ số của nó chia hết cho 3. Viết chương trình nhập vào một số có 3 chữ số, kiểm tra số đó có chia hết cho 3 dùng tính chất trên.(if)

25. Viết chương trình nhận vào giờ, phút, giây dạng (hh:mm:ss), từ bàn phím. Cộng thêm một số giây vào và in ra kết quả dưới dạng (hh:mm:ss).

26. Kiểm tra một ký tự nhập vào thuộc tập hợp nào trong các tập ký tự sau:

    - Các ký tự chữ hoa: 'A' ...'Z'
    - Các ký tự chữ thường: 'a' ... 'z'
    - Các ký tự chữ số : '0' ... '9'
    - Các ký tự khác.

27. Hệ thập lục phân dùng 16 ký số bao gồm các ký tự

    0 .. 9 và A, B, C, D, E ,F.

    Các ký số A, B, C, D, E, F có giá trị tương ứng trong hệ thập phân như sau:

    A 10
    B 11
    C 12
    D 13
    E 14
    F 15

    Hãy viết chương trình cho nhập vào ký tự biểu diễn một ký số của hệ thập lục phân và cho biết giá trị thập phân tương ứng. Trường hợp ký tự nhập vào không thuộc các ký số trên, đưa ra thông báo lỗi: "Hệ thập lục phân không dùng ký số này"

28. Nhập n>=0. Tính S(n)=1+2+3+ … + n.
29. Nhập n>=0. Tính S(n)=2+4+ … + n.
30. Nhập n>=0. Tính S(n)=1+3+ … + n.
31. Nhập n>=0. Tính $S(n)=1^2+2^2+3^2+ \ldots + n^2$.
32. Nhập n>=0. Tính $S(n)=1^2+2^2+3^2+ \ldots + n^2$.
33. Nhập n>=0. Tính $S(n) = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}$
34. Nhập n>=0. Tính $S(n) = \frac{1}{2} + \frac{1}{4} + ... + \frac{1}{2n}$
35. Nhập n>=0. Tính $S(n) = 1 + \frac{1}{3} + \frac{1}{5} + ... + \frac{1}{2n+1}$
36. Nhập n>=0. Tính $S(n) = \frac{1}{1 \times 2} + \frac{1}{2 \times 3} + ... + \frac{1}{n \times (n+1)}$
37. Nhập n>=0. Tính $S(n) = \frac{1}{2} + \frac{2}{3} + ... + \frac{n}{n+1}$
38. Nhập n>=0. Tính $S(n) = \frac{1}{2} + \frac{3}{4} + ... + \frac{2n+1}{2n+2}$
39. Nhập n>=0. Tính T(n)=1×2×3× … ×n
40. Nhập n>=0. Tính S(n)=1+1×2+1×2×3+ … +1×2×3×…×n
41. Nhập x,n. Tính $T(x,n)=x^n$
42. Nhập x,n. Tính $S(n) = \sqrt[n]{x}$
43. Nhập x,n. Tính $S(x,n)=x+x^2+x^3+ \ldots +x^n$
44. Nhập x,n. Tính $S(x,n)=x^2+x^4+x^6+ \ldots +x^{2n}$
45. Nhập x,n. Tính $S(x,n)=x+x^3+x^5+ \ldots +x^{2n+1}$
46. Nhập n. Tính

$$S(n) = 1 + \frac{1}{1+2} + \frac{1}{1+2+3} + ... + \frac{1}{1+2+...+n}$$

47. Nhập x,n. Tính

$$S(x,n) = x + \frac{x^2}{1+2} + \frac{x^3}{1+2+3} + ... + \frac{x^n}{1+2+...+n}$$

48. Nhập x,n. Tính $S(x,n) = x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... + \frac{x^n}{n!}$
49. Nhập x,n. Tính $S(x,n) = x + \frac{x^2}{2!} + \frac{x^4}{4!} + ... + \frac{x^{2n}}{2n!}$
50. Nhập x,n. Tính $S(x,n) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + ... + \frac{x^{2n+1}}{(2n+1)!}$
51. Nhập n. Tính $S(n) = \sqrt{2 + \sqrt{2 + \sqrt{2 + ... + \sqrt{2}}}}$, có n dấu căn lồng nhau.
52. Nhập n. Tính $S(n) = \sqrt{n + \sqrt{(n-1) + \sqrt{(n-2) + ... + \sqrt{2 + \sqrt{1}}}}}$, có n dấu căn lồng nhau.
53. Nhập n. Tính $S(n) = \sqrt{1 + \sqrt{2 + \sqrt{3 + ... + \sqrt{(n-1) + \sqrt{n}}}}}$, có n dấu căn lồng nhau.
54. Liệt kê tất cả các *ước số* của số nguyên dương n.
55. Liệt kê tất cả các *ước số lẻ* của số nguyên dương n.
56. Liệt kê tất cả các *ước số chẵn* của số nguyên dương n.
57. Tính tổng tất cả các *ước số* của số nguyên dương n.
58. Tính tích tất cả các *ước số* của số nguyên dương n.
59. Đếm số lượng các *ước số* của số nguyên dương n.
60. Tìm *ước số lớn nhất* của số nguyên dương n.
61. Kiểm tra n có phải là *số nguyên tố* hay không?
62. Liệt kê các *số nguyên tố* nhỏ hơn hay bằng số nguyên dương n.
63. Liệt kê các chữ số là *số nguyên tố* của số nguyên dương n.
64. Tính tổng các chữ số là *số nguyên tố* của số nguyên dương n.
65. Tính tích các chữ số là *số nguyên tố* của số nguyên dương n.
66. Đếm số lượng các chữ số *chẵn* của số nguyên dương n.
67. Tính tổng các chữ số *chẵn* của số nguyên dương n.
68. Tính tích các chữ số *chẵn* của số nguyên dương n.

69. Đếm số lượng các chữ số *lẻ* của số nguyên dương n.
70. Tính tổng các chữ số *lẻ* của số nguyên dương n.
71. Tính tích các chữ số *lẻ* của số nguyên dương n.
72. Tìm *ước số chung lớn nhất* của 2 số nguyên dương a, b.
73. Tìm *Bội số chung lớn nhất* của 2 số nguyên dương a, b.
74. Kiểm tra số nguyên dương n có phải là *số đối xứng* hay không?
75. Kiểm tra số nguyên dương n có phải là *số hoàn thiện (Pefect number)* hay không? (Số hoàn thiện là số có tổng các ước số của nó (không kể nó) thì bằng chính nó. Vd: 6 có các ước số là 1,2,3 và 6=1+2+3 ➔ 6 là số hoàn thiện)
76. Kiểm tra số nguyên dương n có phải là *số thịnh vượng (Abundant number)* hay không? (Số thịnh vượng là số có tổng các ước số của nó (không kể nó) thì lớn hơn nó. Vd: 12 có các ước số là 1,2,3,4,6 và 12<1+2+3+4+6 ➔ 12 là số thịnh vượng)
77. Kiểm tra số nguyên dương n có phải là *số không trọn vẹn (Deficient number)* hay không? (Số không trọn vẹn là số có tổng các ước số của nó (không kể nó) thì nhỏ hơn nó. Vd: 9 có các ước số là 1,3 và 9>1+3 ➔ 9 là số không trọn vẹn)
78. Kiểm tra số nguyên dương n có các chữ số toàn là *chữ số chẵn* hay không?
79. Kiểm tra số nguyên dương n có các chữ số toàn là *chữ số lẻ* hay không?
80. Kiểm tra số nguyên dương n có *các chữ số tăng dần từ trái qua phải* hay không?
81. Kiểm tra số nguyên dương n có *các chữ số giảm dần từ trái qua phải* hay không?
82. Nhập n>0. Tìm số nguyên dương m lớn nhất sao cho 1+2+3+…+m<n.
83. Nhập n>0. Tìm số nguyên dương m nhỏ nhất sao cho 1+2+3+…+m>n.
84. Xuất số đảo của số nguyên dương n.
85. Xuất ra các ký tự từ A->Z, Z->A, a->z, z->a.

86. Xuất ra các số lẻ nhỏ hơn 50 trừ các số 11, 25, 37.
87. Nhập n>0. Xuất ra bảng cửu chương n.
88. Hãy tìm số gà và số chó? biết:

    Vừa gà vừa chó

    bó lại cho tròn

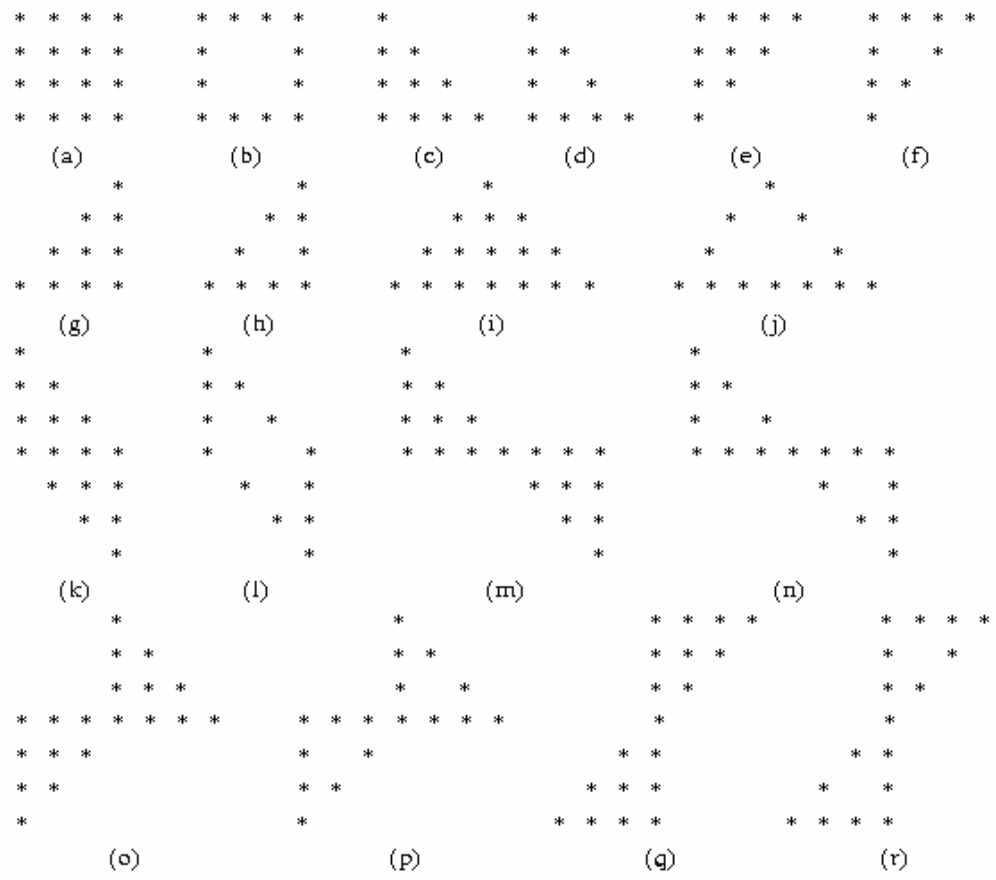
    ba mươi sáu con

    một trăm chân chẵn.

89. Hãy tìm số trâu mỗi loại? biết:

    Trăm trâu tăm cỏ
    Trâu đứng ăn năm
    Trâu nằm ăn ba
    Trâu già ba con một bó

90. Xuất ra màn hình các hình có chiều cao h>0. ví dụ h=4 ta có các hình như sau:



91. Viết chương trình thực hiện trò chơi đoán số như sau:

    Máy lấy ra một số ngẫu nhiên n∈[1,100] là số của máy: **Sốmáy** (sử dụng hàm random).

    - Người nhập vào một số (**Sốnhập**)
      + Nếu **Sốnhập** lớn hơn **Sốmáy** thì thông báo "Số bạn lớn hơn số máy".
      + Nếu **Sốnhập** nhỏ hơn **Sốmáy** thì thông báo "Số bạn nhỏ hơn số máy".
    - Trò chơi kết thúc khi:
      + Hoặc Bạn đã đoán trúng: thông báo "Ha ha bạn tài thật".
      + Hoặc Bạn đã đoán sai 7 lần: thông báo "Bạn đã thua rồi" và hiển thị **Sốmáy**.

92. Trò chơi lấy bì:

"*Có M viên bi, hai người chơi lần lượt lấy đi các viên bi sao cho số viên bi lấy ít nhất là 1 và nhiều nhất là 3, người nào mà lấy được viên bi cuối cùng thì người đó bị thua*"

- Giả sử bạn chơi với máy. Hãy viết chương trình mô phỏng trò chơi này sao cho máy có cơ hội thắng nhiều nhất.
- Người chơi cần nhập vào số viên bi M và chọn lượt lấy bi trước (*máy lấy trước hay bạn lấy trước*) sau đó cứ thay phiên nhau lấy. Cuối cùng thì thông báo kết quả của ván chơi.

Chương 4

MẢNG
(Array)

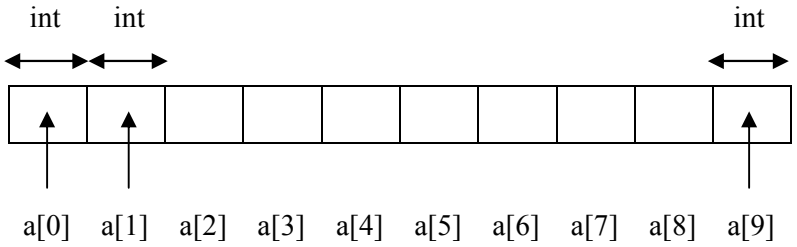
1. Mảng

Mảng là một tập hợp các biến có cùng kiểu dữ liệu nằm liên tiếp nhau trong bộ nhớ và được tham chiếu bởi một tên chung (tên mảng). Mỗi phần tử của mảng được tham chiếu thông qua chỉ mục (*index*). Nếu mảng có n phần tử thì phần tử đầu tiên có chỉ mục là 0 và phần tử cuối có chỉ mục là n-1. Để tham chiếu đến một phần tử ta dùng tên mảng và chỉ mục của phần tử được đặt trong cặp dấu [].

Số lượng phần tử trong mảng được gọi là **kích thước** của mảng. Kích thước của mảng là cố định và phải được xác định trước; nó không thể thay đổi trong suốt quá trình thực hiện chương trình.

Ví dụ: Khai báo mảng a có 10 phần tử. Mỗi phần tử có kiểu int

int a[10];



Có 2 loại mảng thông dụng là mảng 1 chiều và mảng nhiều chiều.

2. Mảng 1 chiều

2.1. Khai báo một mảng một chiều

Dạng tổng quát để khai báo một mảng một chiều là:

```
type arrayName[elements];
```

type: kiểu dữ liệu của mỗi phần tử mảng.

elements: số phần tử có trong mảng

arrayName: tên mảng

Giống như những biến khác, mảng phải được khai báo tường minh để cho trình biên dịch có thể cấp phát bộ nhớ cho nó.

Kích thước (tính bằng byte) của mảng được tính theo công thức:

Total_size = sizeof(type) * elements

Ví dụ, để khai báo một mảng có 100 phần tử tên num có kiểu int, ta dùng lệnh:

```
int num[100];
```

Vậy mảng trên có kích thước là 2bytes * 100 = 200bytes (giả sử int chiếm 2 bytes)

Mỗi phần tử mảng là một biến thông thường. Đoạn lệnh dưới đây minh họa việc sử dụng các phần tử mảng.

```
num[0] = 2; //gán phần tử có chỉ mục 0 giá trị 2
num[1] = num[0] + 3 //num[1] có giá trị 5
num[2] = num[0] + num[1]; //num[2] có giá trị 7
cout << num[1]; //In ra giá trị 5
```

2.2. Khai báo và khởi tạo mảng một chiều

Ngoài ra, ta còn có thể vừa khai báo vừa khởi tạo các phần tử của mảng một chiều. Dạng tổng quát như sau:

```
type arrayName[] = {value1, value2, ..., valuen};
```

Lưu ý: kích thước mảng không khai báo. Số lượng phần tử trong mảng là số số giá trị được cung cấp trong cặp dấu ngoặc {}. Mỗi giá trị phân cách nhau dùng dấu phẩy.

Ví dụ: Xem xét khai báo sau:

```
int soChan[] = {2,4,6,8,10};
```

Mảng soChan có 5 phần tử lần lượt là:
soChan[0] có giá trị là 2
soChan[1] có giá trị là 4
...
soChan[4] có giá trị là 10

2.3. Một số ví dụ

Ví dụ 1: Tạo một mảng nguyên a có N phần tử. Mỗi phần tử có giá trị là chỉ mục của nó. In mảng ra màn hình.

```
#include <iostream.h>
#include <conio.h>
#define N 10
void main()
{
      int a[];
      for(int i=0 ; i < N ; i++)
            a[i] = i ;
      cout<< "In mang:\n";
      for(int i=0 ; i < N ; i++)
            cout << "a[" << i <<"] = " << a[i] <<
      endl;
}
```

Ví dụ 2: Đổi một số nguyên dương thành số nhị phân. Việc chuyển đổi này được thực hiện bằng cách lấy số đó chia liên tiếp cho 2 cho tới khi bằng 0 và lấy các số dư theo chiều ngược lại để tạo thành số nhị phân. Ta sẽ dùng mảng một chiều để lưu lại các số dư đó. Chương trình cụ thể như sau:

```
#include <iostream.h>
#include <conio.h>
void main()
{
      unsigned int n;
      unsigned int remainder;
      unsigned int binary[20],k=0,i;
      cout << "Input an integer n= ";
      cin >> n;
      do
      {
            remainder = n % 2;
            binary[k]= remainder;
            k++;
            n = n/2;
      } while(n>0);
      cout << "Binary form: ";
      for(i=k-1 ; i>=0 ; i--)
            cout << setw(3) << binary[i];
      getch();
}
```

3. Mảng nhiều chiều

C/C++ hổ trợ mảng nhiều chiều. Dạng đơn giản nhất của mảng nhiều chiều là mảng 2 chiều. Mảng hai chiều thực chất là mảng của những mảng một chiều. Ta có thể xem mảng hai chiều là một ma trận gồm các hàng và các cột.

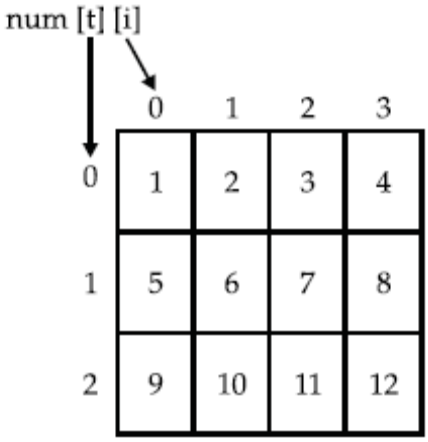
3.1. Khai báo mảng hai chiều

```
type arrayName[rows][columns];
```

rows: số hàng
columns: số cột

Giả sử ta khai báo một mảng num có 3 hàng và 4 cột, kiểu int và gán giá trị cho các phần tử như hình minh họa.

```
int num[3][4];
num[0][0] = 1;
num[0][1] = 2;
num[0][2] = 3;
num[0][3] = 4;
num[1][0] = 5;
...
num[2][3] = 12;
```



3.2. Khai báo và khởi tạo mảng hai chiều

Dạng tổng quát khai báo và khởi tạo mảng hai chiều:

```
type arrayName[][columns] = { {value1,value2,...,valuen},
                              {value1,value2,...,valuen},
                              {...},
                              {value1,value2,...,valuen}};
```

Lưu ý:

- Số phần tử của mỗi hàng phải bằng số cột (columns)
- Số hàng (rows) của khai báo mảng hai chiều để trống.
- Số hàng của mảng được xác định dựa vào số hàng trong phần khởi tạo. Giá trị các phần tử trong mỗi hàng được đặt trong cặp {}, các hàng phân cách nhau bằng một dấu phẩy.

Ví dụ, để khai báo và khởi tạo mảng hai chiều của hình minh họa trên, ta khai báo như sau:

```
int num[][4] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
```

3.3. Một số ví dụ

Ví dụ 1: Tạo 1 mảng hai chiều có ROWS hàng, COLUMNS cột. Giá trị của phần tử trong mảng được xác định bằng tích của chỉ mục hàng và chỉ mục cột của chúng.

```
#include<iostream.h>
#include<conio.h>
#define ROWS 4
#define COLUMNS 3

void main()
{
        int a[ROWS][COLUMNS];
        //Initialization
        for(int i=0 ; i<ROWS ; i++)
                for(int j=0 ; j<COLUMNS ; j++)
                        a[i][j] = i*j;
        //Display array contents
        cout << "Contents in array:\n";
        for(int i=0 ; i<ROWS ; i++)
        {
                for(int j=0 ; j<COLUMNS ; j++)
                        cout << setw(4) << a[i][j];
                cout << endl;
        }

}
```

0	0	0
0	1	2
0	2	4
0	3	6

Ví dụ 2: Tạo một ma trận vuông 4x4. Tính tổng các phần tử trên đường chéo chính

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

void main()
{
	int a[][4] = {{1,2,3,4},
		      {5,6,7,8},
		      {9,10,11,12},
		      {13,14,15,16}};
	int sum=0;
	//Tinh tong duong cheo chinh
	for(int i=0 ; i<4 ; i++)
		for(int j=0 ; j<4 ; j++)
			if(i==j)
				sum += a[i][j];
	cout << "Tong duong cheo chinh la: " << sum;
}
```

BÀI TẬP CHƯƠNG 4

1. Viết chương trình nhập vào một dãy n số thực a[0], a[1],..., a[n-1], sắp xếp dãy số theo thứ tự giảm dần. Xuất ra dãy số sau khi sắp xếp.
2. Viết chương trình sắp xếp một mảng theo thứ tự tăng dần sau khi đã loại bỏ các phần tử trùng nhau.
3. Viết chương trình nhập vào một mảng, hãy xuất ra màn hình:
   - Phần tử lớn nhất của mảng.
   - Phần tử nhỏ nhất của mảng.
   - Tính tổng của các phần tử trong mảng .
4. Viết chương trình nhập vào một dãy các số theo thứ tự tăng, nếu nhập sai quy cách thì yêu cầu nhập lại. In dãy số sau khi đã nhập xong.
5. Viết chương trình nhập vào một ma trận (mảng hai chiều) các số nguyên, gồm m hàng, n cột.In ma trận đó lên màn hình.
6. Viết chương trình để chuyển đổi vị trí từ dòng thành cột của một ma trận (ma trận chuyển vị) vuông 4 hàng 4 cột. Sau đó viết cho ma trận tổng quát cấp m*n.

   Ví dụ:

   1 2 3 4 1 2 9 1
   2 5 5 8 2 5 4 5
   9 4 2 0 3 5 2 8
   1 5 8 6 4 8 0 6
7. Viết chương trình nhập vào một mảng số tự nhiên. Hãy xuất ra màn hình:
   - Dòng 1 : gồm các số lẻ, tổng cộng có bao nhiêu số lẻ.
   - Dòng 2 : gồm các số chẵn, tổng cộng có bao nhiêu số chẵn.
   - Dòng 3 : gồm các số nguyên tố.
   - Dòng 4 : gồm các số không phải là số nguyên tố.
8. Viết chương trình tính tổng bình phương của các số âm trong một mảng các số nguyên.
9. Viết chương trình thực hiện việc đảo một mảng một chiều.

   Ví dụ : 1 2 3 4 5 7 9 10 đảo thành 10 9 7 5 4 3 2 1 .

10. Viết chương trình nhập vào hai ma trận A và B có cấp m, n. In hai ma trận lên màn hình. Tổng hai ma trận A và B là ma trận C được tính bởi công thức:

    $c_{ij} = a_{ij} + b_{ij}$ (i=0,1,2,...m-1; j=0,1,2...n-1)

    Tính ma trận tổng C và in kết quả lên màn hình.

11. Viết chương trình nhập vào hai ma trận A có cấp m, k và B có cấp k, n. In hai ma trận lên màn hình. Tích hai ma trận A và B là ma trận C được tính bởi công thức:

    $c_{ij} = a_{i1} * b_{1j} + a_{i2} * b_{2j} + a_{i3} * b_{3j} + ... + a_{ik} * b_{kj}$

    (i=0,1,2,...m-1;j=0,1,2...n-1)

    Tính ma trận tích C và in kết quả lên màn hình.

12. Nhập số phần tử và các phần tử nguyên dương của mảng a.
    a) In các số nguyên tố có trong mảng a.
    b) Sắp xếp các số chẵn trong mảng theo thứ tự tăng dần.
13. Viết chương trình nhập vào mảng a
    a) Viết hàm kiểm tra mảng đối xứng không? Nếu có trả về 1 ngược lại trả về 0.
    b) Nhập mảng b, kiểm tra mảng b có phải là mảng con của mảng a không? Nếu có trả về số lần mảng b xuất hiện trong mảng a.
14. Viết chương trình theo dạng hàm: nhập vào mảng nguyên a có n phần tử với :
    a) Các số nguyên tố (nếu có) trong mảng phải < 100.
    b) Không có phần tử trùng nhau trong mảng.
    c) Tính tổng các số nguyên tố trong mảng.
15. Viết chương trình thực hiện các bước sau:
    a) Nhập mảng thực.
    b) Sắp xếp mảng thực theo thứ tự tăng dần.
    c) In phần tử có số lần xuất hiện nhiều nhất trong mảng.
16. Nhập vào mảng a, b theo kiểu cấp phát động. Với:
    a) Các phần tử của a và b không trùng nhau.
    b) Xếp theo thứ tự tăng dần hai mảng a, b.
    c) Nối hai mảng này lại thành một mảng duy nhất sao cho mảng vẫn tăng.
17. Nhập vào một mảng a. Thực hiện sắp xếp sau:

a) Tất cả các số lẻ nằm phía trước dãy số, các số chẵn nằm phía sau dãy số, các số 0 nằm giữa.
b) Nhập vào một số x, hãy tìm số nguyên tố trong a bé hơn và gần với x nhất.

18. Viết chương trình nhập vào mảng một chiều có n số nguyên dương. Hãy cho biết số nào trong mảng có giá trị gần với trung bình cộng của toàn mảng.
19. Nhập vào một mảng có n số nguyên dương khác nhau. Hãy in ra tất cả các phần tử trong mảng có giá trị nhỏ hơn giá trị lớn nhất và lớn hơn giá trị nhỏ nhất của mảng.
20. Viết chương trình nhập ngẫu nhiên một mảng có n số nguyên dương. Nhập vào một số nguyên dương k. Hãy tính trung bình cộng của các phần tử trong mảng có giá trị lớn hơn hay bằng k.
21. Nhập vào một dãy số nguyên dương ngẫu nhiên (random) có n phần tử. Viết chương trình in ra số lớn hơn số nhỏ nhất của dãy và nhỏ hơn hay bằng với mọi số còn lại (nghĩa là tìm số nhỏ thứ hai trong dãy). Nếu n phần tử đều bằng nhau thì thông báo: không tồn tại số cần tìm.
22. Viết chương trình nhập vào mảng số nguyên có n phần tử. Hãy tìm số chẵn lớn nhất và số lẻ nhỏ nhất.
23. Hãy nhập dãy n số nguyên dương có giá trị trong khoảng từ 1->100. Sắp xếp lại dãy số trên theo chiều tăng dần và loại bỏ các phần tử trùng nhau (chỉ giữ lại một giá trị trong số đó)
24. Hãy nhập dãy n số nguyên dương có giá trị trong khoảng từ 1->100. Sắp xếp lại dãy số trên theo chiều tăng dần. Nhập vào một số x nguyên dương. Chèn x vào dãy sao cho thứ tự của dãy không thay đổi.
25. Hãy nhập dãy n số nguyên dương có giá trị trong khoảng từ 1 -> 100. In ra màn hình các số chẵn xuất hiện trong dãy theo thứ tự tăng dần.
26. Hãy nhập dãy n số nguyên dương có giá trị trong khoảng từ 1->100. In ra giá trị trung bình cộng của các số chẵn xuất hiện trong dãy.
27. Viết chương trình thực hiện các công việc sau:
    a) Nhập vào một ma trận các giá trị thực kích thước mxn, với n và m được nhập từ bàn phím.
    b) Tính tổng các số dương có trong mảng.
28. Viết chương trình thực hiện các công việc sau:

a) Nhập vào một ma trận các giá trị thực kích thước nxn, với n được nhập từ bàn phím.
b) Tìm tất cả các vị trí trong ma trận thỏa yêu cầu sau: giá trị của ma trận tại vị trí đó là giá trị lớn nhất của ma trận.

29. Viết chương trình thực hiện công việc sau:
    a) Nhập vào số nguyên dương N. Cấp phát động một mảng nguyên A có N phần tử. Thực hiện việc nhập giá trị cho mảng này.
    b) Tìm số nguyên tố lớn nhất có trong mảng. Nếu không có phải có thông báo.
30. Viết chương trình nhập vào ma trận vuông A(NxN), với N nhập vào từ bàn phím.
    a) In ra tổng các giá trị trong tam giác vuông trên của ma trận A (kể cả các phần tử trên đường chéo của ma trận A)

In ma trận tích AxA ra màn hình.

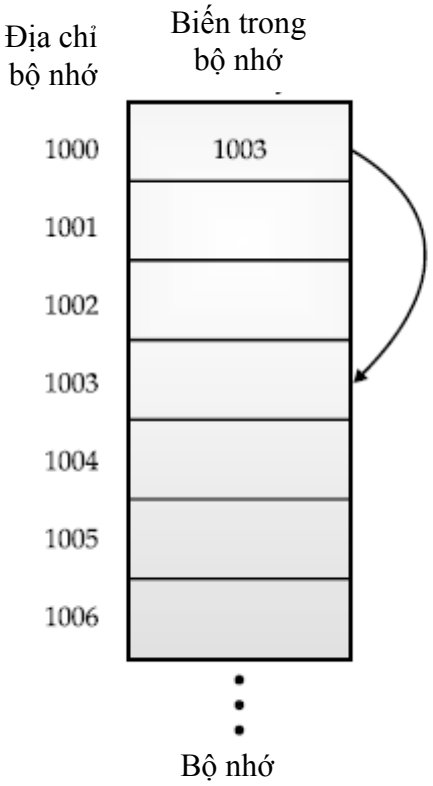
Chương 5

CON TRỎ (Pointers)

1. Con trỏ

Một con trỏ là 1 biến chứa một địa chỉ bộ nhó. Địa chỉ này là vị trí của một đối tượng khác (thường là một biến) trong bộ nhớ. Nếu một biến chứa địa chỉ của một biến khác, biến thứ nhất được gọi là trỏ đến biến thứ hai.

Ví dụ:



Một biến được cấp phát ô nhớ tại địa chỉ 1000 có giá trị là địa chỉ (1003) của 1 biến khác. Biến thứ nhất được gọi là con trỏ.

2. Biến con trỏ (pointer variables)

Nếu một biến sẽ chứa địa chỉ của một biến khác thì nó phải được khai báo là một con trỏ. Khai báo 1 biến là con trỏ gồm kiểu dữ liệu cơ sở, một dấu *, và tên biến. Dạng tổng quát để khai báo một biến con trỏ là

```
type *pointerVariable;
```

type: xác định kiểu dữ liệu của biến mà con trỏ có thể trỏ đến. Ví dụ con trỏ có kiểu int sẽ trỏ đến biến có kiểu int. Do các phép toán số học trên con trỏ (tăng, giảm) liên quan đến type của nó nên cần phải khai báo type của con trỏ đúng đắn.

2.1. Các toán tử con trỏ (*pointer operators*)

Có 2 toán tử con trỏ là * và &.

Toán tử & là toán tử 1 ngôi mà trả về địa chỉ bộ nhớ của toán hạng của nó. (toán tử 1 ngôi chỉ yêu cầu 1 toán hạng).

Ví dụ:

```
int count;

int *m;

m = &count;
```

Lệnh m=&count; đặt địa chỉ bộ nhớ của biến count vào con trỏ m. Lệnh trên có thể phát biểu: "con trỏ m nhận địa chỉ của biến count.".

Giả sử biến count được cấp phát tại địa chỉ bộ nhớ 2000 để lưu trữ giá trị của nó. Giả sử rằng count có giá trị 100. Như vậy, tại địa chỉ bộ nhớ 2000 có chứa giá trị 100. Sau khi lệnh m = &count; được thực hiện thì m sẽ có giá trị là 2000.

Toán tử con trỏ * là toán tử một ngôi trả về giá trị tại địa chỉ con trỏ trỏ đến.

Ví dụ: q = *m;

Lấy giá trị tại địa chỉ mà m trỏ đến và đặt vào biến q. Như vậy q sẽ có giá trị là 100 (là giá trị của biến count).

2.2. Các thao tác trên con trỏ

2.2.1. Lệnh gán con trỏ

Ta có thể dùng một con trỏ ở bên phải của câu lệnh gán (=) để gán giá trị của 1 con trỏ cho một con trỏ khác. Ví dụ:

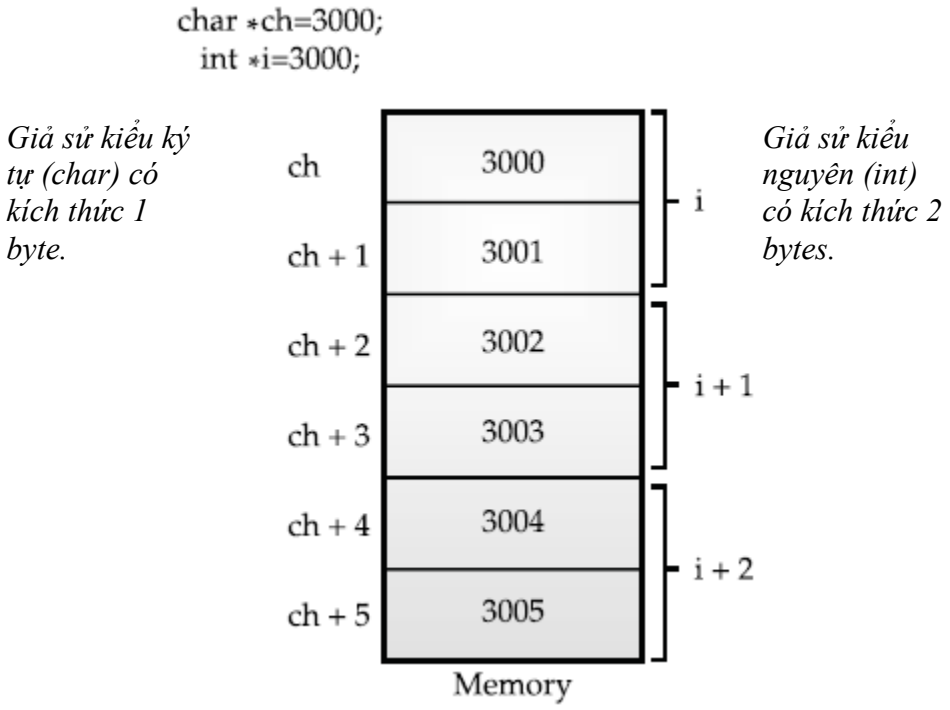
```
int x;
int *p1, *p2;
p1 = &x;
p2 = p1;
```

Sau khi đọan lệnh trên được thực hiện, cả hai p1 và p2 cùng trỏ đến biến x.

2.2.2. Phép toán số học trên con trỏ

Chỉ có 2 phép toán số học ta có thể dùng trên con trỏ đó là cộng và trừ. Giả sử p1 là một con trỏ nguyên với giá trị hiện tại là 2000. Cũng giả sử rằng số nguyên chiếm 2 bytes bộ nhớ. Như vậy, sau khi thực hiện lệnh `p1++;` thì p1 có giá trị là 2002 chứ không phải 2001. Tương tự, Giả sử p1 là một con trỏ nguyên với giá trị hiện tại là 2000. Cũng giả sử rằng số nguyên chiếm 2 bytes bộ nhớ. Như vậy, sau khi thực hiện lệnh `p1--;` thì p1 có giá trị là 1998 chứ không phải 1999.

Tổng quát từ 2 ví dụ trên: Tất cả con trỏ sẽ tăng hay giảm với đơn vị là kích thước của kiểu dữ liệu của nó.



Ngoài toán tử tăng (++) và giảm (--), ta có thể cộng hay trừ số nguyên với con trỏ. Ví dụ: theo hình minh họa trên.

Con trỏ char ch chứa địa chỉ 3000, vậy lệnh

```
ch = ch + 3;
```

ch sẽ chứa địa chỉ 3003

Con trỏ nguyên i chứa địa chỉ 3000, vậy lệnh

```
i = i + 2;
```

i sẽ chứa địa chỉ 3004

Lưu ý: đơn vị tăng của con trỏ char là 1 byte, con trỏ int là 2 bytes.

Tương tự, giả sử con trỏ char ch chứa địa chỉ 3003, vậy lệnh

```
ch = ch - 3;
```

ch sẽ chứa địa chỉ 3000

Giả sử con trỏ nguyên i chứa địa chỉ 3004, vậy lệnh

```
i = i – 2;
```

i sẽ chứa địa chỉ 3000

3. Một số ví dụ về con trỏ

Ví dụ 1: Viết chương trình hoán đổi giá trị của 2 biến dùng con trỏ

```
#include <iostream.h>
#include<conio.h>
void main ()
{
  int a = 20, b = 15;
  int *pa, *pb, temp;
  pa = &a; // con trỏ pa chứa địa chỉ của a
  pb = &b; // con trỏ pb chứa địa chỉ của b
  temp = *pa;
  *pa = *pb;
  *pb = temp;
  cout << "a = " << a << endl;
  cout << "b = " << b;
  getch();
}
```

// kết quả xuất ra màn hình

a = 15

b = 20

4. Cấp phát bộ nhớ động

Con trỏ cung cấp sự hổ trợ cho cấp phát bộ nhớ động trong C/C++. Cấp phát động là phương tiện nhờ đó một chương trình có thể dành được thêm bộ nhớ trong khi đang thực thi.

Biến toàn cục (*global variables*) được cấp phát bộ nhớ vào lúc biên dịch. Biến cục bộ (*local variables*) dùng stack. Tuy nhiên, biến toàn cục hay cục bộ không thể được tạo thêm trong khi thực thi

chương trình. Một số chương trình cần thêm bộ nhớ khi thực thi, giải pháp cho vấn đề này là cấp phát động.

C/C++ hổ trợ hai hệ thống cấp phát động: một cái được định nghĩa bởi C và một cái bởi C++.

4.1. Cấp phát động được định nghĩa bởi C

Bộ nhớ cấp phát động bởi những hàm cấp phát động của C là từ heap (heap là vùng nhở rỗi nằm giữa chương trình của bạn và vùng lưu trữ thường trực và stack). Mặc dầu kích thước vùng nhớ heap là không biết trước, nhưng nói chung là khá lớn.

Hai hàm cấp phát động quan trọng nhất của C là malloc() và free(). Những hàm này làm việc cùng nhau để dùng vùng nhớ rỗi để cấp phát và thu hồi bộ nhớ. Hàm malloc() dùng để cấp phát bộ nhớ động và hàm free() dùng để thu hồi. Bất kỳ chương trình nào dùng những hàm này phải include tập tin header stdlib.h.

Hàm malloc() có nguyên mẫu (prototype) sau:

```
void *malloc(length)
```

length: là số byte muốn cấp phát bộ nhớ. Hàm malloc() trả về một con trỏ có kiểu void, do đó có thể gán nó cho con trỏ có kiểu bất kỳ. Sau khi cấp phát thành công, hàm malloc() trả về địa chỉ của byte đầu tiên của vùng nhớ được cấp phát từ heap. Nếu không thành công (không có đủ vùng nhớ rỗi yêu cầu), hàm malloc() trả về null.

Đọan mã dưới đây cấp phát 1000 bytes vùng nhớ liên tục:

```
char *p;

p = (char *) malloc(1000); //cấp phát 1000 bytes
```

Vì hàm malloc() trả về con trỏ kiểu void, trong trường hợp này ta phải ép kiểu (casting) nó thành con trỏ char cho phù hợp với biến con trỏ p.

Đọan mã dưới đây cấp phát vùng nhớ cho 50 số nguyên.

```
int *p;
```

```
p = (int *) malloc(50*sizeof(int));
```

Lưu ý: trong ví dụ trên ta dùng toán tử sizeof để xác định kích thước kiểu dữ liệu int.

Từ đó, do kích thước của heap thì không xác định nên khi cấp phát bộ nhớ ta phải kiểm tra giá trị trả về của hàm malloc() để biết là bộ nhớ có được cấp phát thành công hay không. Đoạn mã dưới đây dùng để kiểm tra:

```
p = (int *)malloc(100);
if(p == NULL)
{     cout << "Khong du bo nho";
      exit(1);
}
```

Hàm free() thì ngược lại với hàm malloc(). free() trả về vùng nhớ được cấp trước đó cho hệ thống. Hàm free() co khuôn mẫu sau:

```
void free(void *p);
```

Ở đây, p là con trỏ đến vùng nhớ đã được cấp phát trước đó bởi hàm malloc().

4.2. Cấp phát động được định nghĩa bởi C++

C++ cung cấp hai toán tử cấp phát bộ nhớ động: new và delete. Những toán tử này dùng để cấp phát và thu hồi bộ nhớ trong khi chương trình thực thi.

Toán tử new cấp phát bộ nhớ và trả về một con trỏ đến byte đầu tiên của vùng nhớ được cấp phát. Toán tử delete thu hồi vùng nhớ được cấp phát trước đó bởi toán tử new. Dạng tổng quát của new và delete là:

```
p = new type;
delete p;
```

Ở đây, p là một biến con trỏ mà nhận địa chỉ của vùng nhớ được cấp phát đủ lớn để chứa 1 đối tượng có kiểu là type.

Ví dụ:

```
#include <iostream>
#include <new>
int main()
{
   int *p;
   p = new int; // allocate space for an int
   *p = 100;
   cout << "At " << p << " ";
   cout << "is the value " << *p << "\n";
   delete p;
   return 0;
}
```

5. Con trỏ void (*void pointers*)

Kiểu dữ liệu khi khai báo biến con trỏ chính là kiểu dữ liệu mà con trỏ có thể trỏ đến. Địa chỉ đặt vào biến con trỏ phải cùng kiểu với kiểu của con trỏ. Xem xét đoạn mã sau:

```
int a;

float f;

int *pa;

float *pf;
```

Những lệnh sau là hợp lệ:

```
pa = &a;

pf = &f;
```

Những lệnh sau là không hợp lệ:

```
pa = &f;   //pa là con trỏ int do đó chỉ chứa địa chỉ
           //của biến kiểu int
pf = &a;   //pf là con trỏ float do đó chỉ chứa
           // địa chỉ của biến kiểu float
```

Con trỏ void là một loại con trỏ đặc biệt mà có thể trỏ đến bất kỳ kiểu dữ liệu nào. Cú pháp khai báo con trỏ void như sau:

```
void *pointerVariable;
```

Nếu ta khai báo con trỏ void sau:

```
void *p;
```

thì các lệnh sau đây là hợp lệ

```
p = &a; //sau lệnh này p trỏ đến biến nguyên a
p = &f; //sau lệnh này p trỏ đến biến thực f
```

Tuy nhiên, tùy thuộc con trỏ void đang trỏ đến kiểu dữ liệu nào, ta phải ép về đúng kiểu tương ứng khi dùng trong các biểu thức.

Ví dụ p đang trỏ đến biến nguyên a, để tăng giá trị của biến a lên 10 ta phải dùng lệnh sau: `(int*)*p + 10;`

Nếu p đang trỏ đến biến thực f, để tăng giá trị của biến f lên 10 ta phải dủng lệnh sau: `(float*)*p + 10;`

6. Con trỏ null (*Null pointers*)

Một con trỏ hiện hành không trỏ đến một địa chỉ bộ nhớ hợp lệ thì được gán giá trị NULL (mà là zero). Bởi qui ước, con trỏ NULL là con trỏ không trỏ đến đâu cả và không nên dùng. NULL được định nghĩa trong <cstdlib>

Nếu chương trình vô tình dùng con trỏ null như dưới đây thì sẽ nhận lỗi khi thực thi chương trình (*run-time error*).

```
#include <iostream.h>
void main()
{       int *p;
        cout << "Gia tri con tro p tro den la: " << *p;
}
```

Kết quả của chương trình trên là:

NULL POINTER ASSIGNMENT

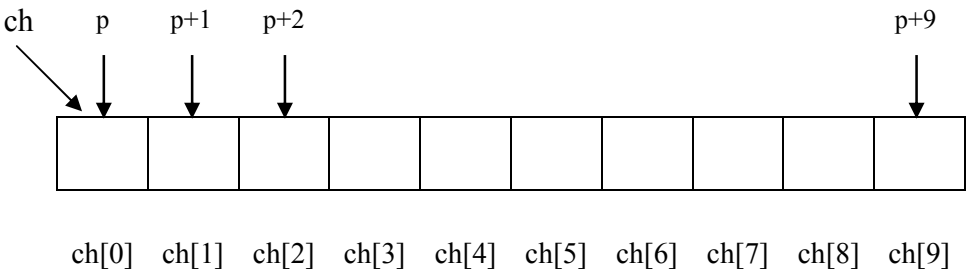
Chương trình trên sẽ gây ra lỗi vào lúc thực thi vì biến con trỏ p không được gán địa chỉ bộ nhớ hợp lệ. Nó là con trỏ null, vì vậy, truy cập đến địa chỉ 0 gây ra thông báo lỗi trên.

7. Con trỏ và mảng

Giữa con trỏ và mảng có một sự quan hệ gần. Xem xét đoạn mã dưới đây:

```
char ch[80], *p;

p = ch;
```



Ở đây, p được gán địa chỉ của phần tử đầu tiên của mảng ch. Để tham chiếu phần tử thứ 3 trong mảng ch, ta dùng một trong 2 cách sau: ch[2] hoặc *(p+2). Cả hai cách đều tham chiếu đến phần tử thứ ba trong mảng ch (lưu ý chỉ mục của mảng bắt đầu là 0).

Tổng quát: C/C++ cung cấp hai phương thức để truy cập các phần tử mảng: dùng chỉ mục thông qua tên mảng và dùng số học con trỏ.

Tên mảng được xem là "con trỏ hằng" nghĩa là nó chứa địa chỉ của phần tử đầu tiên trong mảng và giá trị này không bao giờ thay đổi. Do đó ta không thể áp dụng các phép toán số học (cộng, trừ) cho tên mảng. Như vậy, trong ví dụ trên ch là tên mảng đồng thời cũng là con trỏ hằng và lệnh p = ch; sao chép địa chỉ của ch vào biến con trỏ p.

Ví dụ: In ra giá trị các phần tử của mảng sau khi đã nhân cho 10 dùng con trỏ

```
#include <iostream>
void main()
{
   int a[] = {0,1,2,3,4,5,6,7,8,9};
   int *p;
   p = a;
   for(int i=0 ; i<10 ; i++)
   {
      *(p+i) *= 10; //tuong duong a[i] = a[i]*10
      cout << "a[" << i << "] = " << *(p+i);
   }
}
```

8. Mảng con trỏ

Mỗi biến con trỏ là một biến đơn. Ta có thể tạo mảng của các con trỏ. Mỗi phần tử của mảng là một con trỏ thông thường. Cú pháp để khai báo mảng con trỏ là:

```
type *pointerArray[elements];
```

type: kiểu dữ liệu mà các con trỏ phần tử trỏ đến.

pointerArray: tên mảng con trỏ.

elements: số phần tử của mảng con trỏ.

Ví dụ: int *p[5]; lệnh này khai báo mảng con trỏ p có 5 phần tử. Các lệnh dưới đây minh họa cách sử dụng mảng này:

p[0] = &a; //gán địa chỉ của biến nguyên a cho con trỏ p[0]

p[2] = p[0]; //sao chép địa chỉ có trong p[0] vào p[2]

b = *p[0]; //gán giá trị tại địa chỉ p[0] trỏ đến vào biến b. Trong trường hợp này, lệnh tương đương với b=a; vì hiện tại p[0] chứa địa chỉ của biến a.

BÀI TẬP CHƯƠNG 5

1. Viết chương trình nhập vào một mảng a gồm n phần tử nguyên. Sắp xếp mảng theo chiều giảm dần (lưu ý sử dụng tên mảng như con trỏ và sử dụng con trỏ).
2. Hãy dùng một vòng for để nhập vào một ma trận vuông cấp n với các phần tử thực và tìm phần tử Max của ma trận này.
3. Viết hàm hoán vị hai biến thực a, b bằng cách sử dụng con trỏ (đối vào là hai con trỏ). Viết chương trình chính nhập hai số thực a, b. Sử dụng hàm trên để đổi chỗ a và b.
4. Viết hàm giải hệ phương trình bậc nhất với sáu đối vào là a, b, c, d, e, f và 2 đối ra là x và y.
5. Viết hàm tính giá trị đa thức:

   $f(x) = a_0x^n + \ldots + a_{n-1}x + a_n$. với đối vào là biến nguyên n và mảng thực a.
6. Viết hàm cộng hai ma trận vuông a và b cấp n (sử dụng con trỏ).
7. Viết chương trình tính tích phân của f(x) trên đoạn [a, b] bằng công thức hình thang. Theo đó, tích phân của f(x) trên [a, b] bằng: h * s. Trong đó:

   h là độ dài khoảng phân hoạch đoạn [a, b] thành n khoảng.

   s là tổng tất cả các f(a+i*h) với i từ 1 tới n.

   Sử dụng hàm trên để tính tích phân trong đoạn [-1, 4] của:

   $f(x) = (e^x-2\sin(x^2))/ (1+x^4)$. (nghiên cứu cách đưa con trỏ vào giải quyết bài toán).

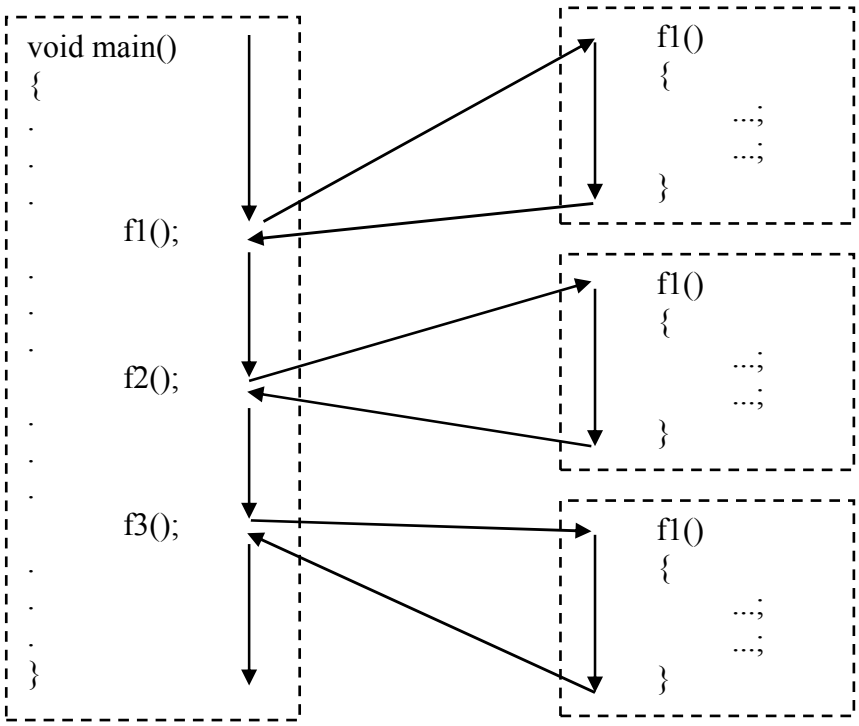
Chương 6

HÀM
(Functions)

1. Khái niệm hàm

Một hàm là một khối lệnh được đặt tên và có một tính chất là nó có thể được thực thi từ nhiều điểm khác nhau trong chương trình khi được gọi.

Hàm nhóm một số lệnh thành một khối và được đặt một tên. Khối này còn gọi là unit hay module. Hàm có thể được gọi từ nhiều chỗ khác nhau trong chương trình. Khi hàm được gọi, khối lệnh tương ứng của hàm được thực thi. Sau khi thực hiện xong, quyền điều khiển được trả về cho chương trình gọi. Xem hình minh họa dòng thực thi của chương trình có gọi đến nhiều hàm.



Hàm còn được gọi là chương trình con *(subroutine)*. Hàm có thể trả về giá trị cho chương trình gọi hoặc không. Nếu hàm không trả về giá trị cho chương trình gọi thì nó được gọi là thủ tục *(procedure)*. Hàm có thể được gọi từ chương trình chính (hàm main) hoặc từ 1 hàm khác. Hàm có thể được gọi nhiều lần.

Có hai loại hàm: hàm thư viện và hàm do người dùng định nghĩa. Hàm thư viện là những hàm đã được xây dựng sẵn. Muốn sử dụng các hàm thư viện trong chương trình ta phải khai báo thư viện chứa nó trong phần khai báo #include.

Ví dụ: Tìm min của 4 giá trị a,b,c,d

```
#include <iostream>
int min(int a, int b); //prototype
void main()
{
   int a=40, b=30, c=10, d=20, min4;
   min4 = min(a,b);
   min4 = min(min4,c);
   min4 = min(min4,d)
   cout << "Min = " << min4;
}
int min(int a, int b)
{
      if(a<b) return a;
      else return b;
}
```

2. Dạng tổng quát của hàm

Hàm có dạng tổng quát như sau:

```
returnType functionName(parameterList)
{
     body of the function
}
```

returnType: Kiểu dữ liệu của giá trị trả về bởi hàm. Nếu hàm không trả về giá trị thì returnType là void
functionName: Tên hàm.
parameterList: Danh sách các tham số hình thức được đặt trong cặp dấu (). Là danh sách các tên biến và kiểu dữ liệu tương ứng của chúng phân cách nhau bởi dấu phẩy. Nếu hàm không có tham số thì danh sách này là rỗng.

Ví dụ: xem khai báo hàm sau:

```
int max(int a, int b)
{
      if(a<b)    return b;
      else  return a;
}
```

Trong ví dụ trên, max là tên hàm, hàm cần 2 tham số để hoạt động là 2 biến nguyên, dữ liệu trả về của hàm có kiểu int. Phần mã nằm giữa {} là thân hàm.

3. Các qui tắc về phạm vi của hàm

Hàm giống như một hộp đen, bên trong hàm (thân hàm) là không biết đối với những lệnh nằm ngoài hàm. Phần thân của hàm chỉ được thực thi khi hàm được gọi. Do đó, không có lệnh nào bên ngoài hàm có thể nhảy trực tiếp vào thân hàm. Tóm lại, mã và dữ liệu bên trong một hàm không thể tương tác với mã và dữ liệu nằm trong một hàm khác.

Những biến được khai báo bên trong hàm (kể cả các tham số hình thức) là những biến cục bộ. Những biến này được tạo ra khi hàm được gọi và biến mất sau khi hàm thực thi xong.

4. Tham số hình thức và tham số thực

Khi hàm cần nhận đối số (*arguments*) để thực thi thì khi khai báo hàm cần khai báo danh sách các tham số để nhận giá trị từ chương trình gọi. Các tham số này được gọi là tham số hình thức. Khi gọi hàm, ta cung cấp các giá trị thật, các giá trị này sẽ được sao chép vào các tham số hình thức và khi đó ta gọi chúng là tham số thực.

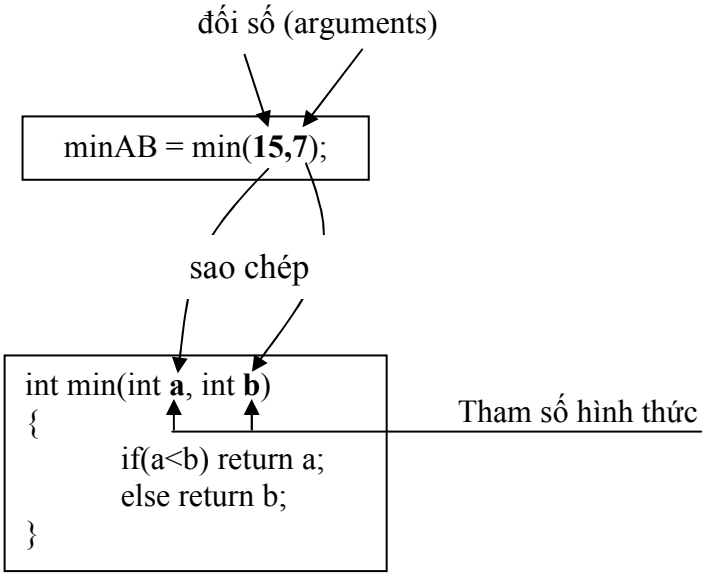
Ví dụ: Xem xét định nghĩa hàm sau

```
int min(int a, int b)
{
      if(a<b) return a;
      else return b;
}
```

Trong định nghĩa hàm min ở trên thì a và b là 2 tham số hình thức. Khi gọi hàm như câu lệnh sau:

minAB = min(15,7);

thì 15 được sao chép vào biến a, 7 được sao chép vào biến b. 15 và 7 được gọi là đối số. Khi a và b nhận giá trị do lời gọi hàm thì chúng được gọi là tham số thực.



Có hai cách truyền đối số vào tham số hình thức: Truyền tham trị và truyền tham biến.

4.1. Truyền tham trị (*call by value*)

Cách này sao chép giá trị của đối số vào tham số hình thức của hàm. Trong trường hợp này, những thay đổi của tham số không ảnh hưởng đến đối số. Như vậy, nếu không muốn hàm làm thay đổi giá trị của đối số truyền vào thì ta khai báo tham số của hàm là biến thông thường (nghĩa là không phải biến con trỏ).

Ví dụ: Khảo sát chương trình sau

```
#include <iostream>
void doubleNum(int a); //prototype
void main()
{
   int a=40;
   doubleNum(a);
   cout << "Inside main function:" << endl;
   cout << "a = " << a << endl;
}
void doubleNum(int a)
{
      a = a*2;
      cout << "Inside doubleNum function. a = " << a;
}
```

Khi đối số a của hàm main được truyền vào tham số a của hàm doubleNum thì cách truyền này là truyền tham trị, nghĩa là giá trị của a là 40 được truyền vào tham số a. Trong thân hàm doubleNum, tham số a được nhân đôi và có giá trị là 80. Khi hàm doubleNum kết thúc, biến a trong hàm main vẫn không thay đổi.

4.2. Truyền tham chiếu *(call by reference)*

Trong cách này, địa chỉ của đối số được sao chép vào tham số hình thức. Do đó, những thay đổi làm đối với tham số sẽ có tác dụng trên đối số.

Ví dụ 1: Xem xét ví dụ trên được viết lại như sau:

```
#include <iostream.h>
void doubleNum(int *b); //prototype
void main()
{
  int a=40;
  doubleNum(&a);
  cout << "Inside main function:" << endl;
  cout << "a = " << a << endl;
}
void doubleNum(int *b)
{
  *b = *b + *b;
  cout << "Inside doubleNum function. a = " << *b;
}
```

Trong chương trình trên, khi gọi hàm doubleNum ta truyền địa chỉ của đối số a vào biến con trỏ b của hàm. Như vậy, con trỏ b sẽ chứa địa chỉ của biến a của hàm main. Trong thân hàm doubleNum, lệnh làm gấp đôi giá trị của vùng nhớ do con trỏ b trỏ đến thực chất đã làm tăng gấp đôi giá trị của biến a trong hàm main.

Ví dụ 2: Hoán đổi giá trị 2 biến dùng con trỏ (truyền tham chiếu)

```
#include <iostream.h>
void swap(int *a, int *b); //prototype
void main()
{
   int a = 20, b = 40;
   int *pa, *pb;
   pa = &a;
   pb = &b;
   swap(pa,pb);
   cout << "After call swap, Values:" << endl;
   cout << "a = " << a << endl;
   cout << "b = " << b << endl;

}
void swap(int *a, int *b)
{
      int temp;
      temp = *a;
      *a = *b;
      *b = temp;
}
```

5. Truyền mảng vào hàm

Khi một mảng được dùng như một đối số để truyền cho hàm, địa chỉ của mảng được truyền đến hàm vào tham số hình thức. Như vậy, truyền mảng vào hàm mặc định là truyền tham chiếu. Do đó, những thay đổi đến giá trị của các phần tử mảng trong thân hàm sẽ ảnh hưởng đến mảng gốc.

Ví dụ: Viết chương trình thay đổi giá trị các phần tử mảng theo yêu cầu: nếu giá trị >=0 thì thay bằng 1, ngược lại thay bằng 0.

```
#include <iostream.h>
void change(int a[], int elements); //prototype
void main()
{
   int arr[] = {5, -5, -3, 3, 7, -7};
   change(arr,6);
   cout << "After call change, value of array:\n";
   for(int i=0 ; i<6 ; i++)
      cout << "arr[" <<i<< "] = " << arr[i] << endl;
}
void change(int a[], int elements)
{
      for(int i=0 ; i<elements ; i++)
            if(a[i] < 0) a[i]=0;
            else a[i] = 1;
}
```

Như vậy, sau khi gọi hàm change, giá trị các phần tử trong mảng bị thay đổi thành 0 hay 1.

6. Đối số của hàm main

Hàm main là điểm bắt đầu của mọi chương trình C/C++.
Thỉnh thoảng, ta cần truyền thông tin vào hàm main khi nó thực thi. Những thông tin này ta gọi là đối số dòng lệnh *(command line arguments)*.

Hàm main có 2 tham số là argv và argc dùng để nhận các đối số dòng lệnh. Tham số argc là một biến nguyên giữ số đối số có trong dòng lệnh. Tham số argv là một mảng con trỏ char. Mỗi phần tử của mảng này trỏ đến một đối số dòng lệnh. Tất cả đối số dòng lệnh là chuỗi (*string*).

Khi hàm main có nhận đối số dòng lệnh, nó được khai báo như sau:

```
int main(int argc, char *argv[])
```

Xem xét chương trình sau:

```
#include <iostream.h>
int main(int argc, char *argv[])
{
      if(argc!=2)
      {
            count << "Hello, " << argv[1];
            exit(1);
      }
      return 0;
}
```

Giả sử sau khi biên dịch chương trình trên, ta được tập tin thực thi là greeting.exe

Tại dấu nhắc hệ điều hành DOS, ta nhập lệnh sau:
greeting Mr.IT

Thì trên màn hình xuất hiện:
Hello, Mr.IT

Lệnh geeting Mr.IT gồm 2 đối số dòng lệnh là getting và Mr.IT, các đối số này được trỏ đến bởi 2 con trỏ là argv[0] và argv[1]. Do đó, khi thực thi chương trình trên, chuỗi Mr.IT sẽ được in ra.

Khi một chương trình không yêu cầu cung cấp đối số dòng lệnh, thông thường nó được khai báo là main() mà không có tham số. Đây là mẫu được dùng cho hầu hết các ví dụ trong tài liệu này.

7. Lệnh return

Lệnh return có 2 cách dùng quan trọng. Thứ nhất, nó kết thúc ngay lập tức hàm chứa nó khiến sự thực thi chương trình được trả về cho chương trình gọi hàm. Thứ hai, nó dùng để trả về một giá trị cho chương trình gọi.

7.1. Cách dùng thứ nhất (kết thúc hàm)

Có hai cách thức để hàm kết thúc sự thực thi của nó và trả điều khiển về chương trình gọi nó.
Một là (thông thường) là khi lệnh cuối cùng có trong hàm được thực thi.

Ví dụ: Xem xét chương trình sau:

```
#include <iostream.h>

void printMessage();

void main()
{
  cout << "Calling printMessage: " <<endl;
  printMessage();
  cout << "Calling printMessage again: " << endl;
  printMessage();
}

void printMessage()
{
  cout << "This message comes from the printMessage function." << endl;
}
```

Trong ví dụ trên, vì hàm main khai báo kiểu trả về là void nên sau khi thực thi tất cả các lệnh có trong hàm main, chương trình kết thúc và trả quyền điều khiển về cho hệ điều hành. Trong thân hàm main, khi gọi hàm printMessage, điều khiển được trao cho hàm printMessage, các lệnh trong thân hàm printMessage thực thi và rồi điều khiển được trả về cho chương trình gọi (trong trường hợp này là hàm main). Lưu ý là cả hai hàm trên đều không dùng lệnh return.

Hai là khi hàm thực hiện câu lệnh return.
Ví dụ: In các phần tử của mảng đến khi gặp phần tử có giá trị âm

```
#include <iostream.h>

void main()
{     int a[] = {3,2,1,0,-1,-2,-3];
      for(int i=0 ; i<7 ; i++)
      {     if(a[i] < 0) return;
            count << setw(5) << a[i];
      }
}
```

Thực hiện chương trình trên, khi duyệt đến phần tử a[4] có giá trị là -1 nên biểu thức điều kiện của câu lệnh if là true nên lệnh return được thực thi và chương trình kết thúc.

7.2. Cách dùng thứ hai (trả về một giá trị)

Xem xét ví dụ sau: Tính tổng các phần tử có trong mảng

```
#include <iostream.h>
int total(int a[], int size);
void main()
{
      int a1[] = {1,2,3];
      int a2[] = {1,2,3,4,5,6];
      int total1, total2;
      total1 = total(a1,3);
      total2 = total(a2,6);
      cout << "Tong mang 1 la = " << total1 << endl;
      cout << "Tong mang 2 la = " << total2 << endl;
}

int total(int a[], int size)
{
      int sum=0;
      for(int i=0 ; i<size ; i++)
            sum += a[i];
      return sum;
}
```

Như vậy, mỗi khi hàm total được gọi, mảng tương ứng được tính tổng và lệnh return trả về giá trị này cho chương trình gọi.

8. Đệ qui

Một hàm có thể gọi đến chính nó. Một hàm được gọi là đệ qui nếu một lệnh trong thân hàm gọi đến chính hàm đó.

Ví dụ, xem xét chương trình tính giai thừa của n.
n! = 1*2*..*n

```
#include <iostream.h>

int giaiThua(int n);

void main()
{
      int gt4, gt7;
      gt4 = giaiThua(4);
      gt7 = giaiThua(7);
      cout << "4! =" << gt4 << endl;
      cout << "7! =" << gt7 << endl;
}

int giaiThua(int n)
{
      int gt;
      if(n==1) return(1);
      gt = giaiThua(n-1)*n; // goi de qui
      return gt;
}
```

9. Nguyên mẫu hàm (*function prototypes*)

Trong C/C++, tất cả các hàm phải được khai báo trước khi chúng được sử dụng. Việc này thực hiện bằng cách khai báo nguyên mẫu của hàm. Nguyên mẫu hàm cho phép C/C++ cung cấp chức năng kiểm tra sự hợp lệ của tham số khi định nghĩa cũng như khi gọi hàm. Khi biên dịch, trình biên dịch sẽ dựa vào nguyên mẫu hàm để kiểm tra xem có sự không hợp lệ nào của các đối số khi gọi hàm và kiểu của các tham số hình thức trong định nghĩa hàm. Nó cũng kiểm tra xem số đối số cung cấp khi gọi hàm có phù hợp với số tham số hình thức của hàm.

Dạng tổng quát của một nguyên mẫu hàm:

```
type functionName(type parameter1, type parameter2, ...);
```

type: kiểu dữ liệu trả về bởi hàm
functionName: tên hàm
parameter1, parameter2,... : danh sách các tham số hình thức và kiểu của chúng.

Lưu ý: Khai báo nguyên mẫu hàm phải có dấu chấm phẩy ở cuối. Nhưng khi định nghĩa hàm thì không có.

10. Cấu trúc của một chương trình viết dưới dạng hàm

- Phần khai báo các thư viện
- Phần khai báo các hằng toàn cục *(nếu có)*
- Phần khai báo các biến toàn cục *(nếu có)*
- **Phần khai báo các nguyên mẫu hàm (prototype)**
- Phần hàm main (sẽ gọi các hàm thực hiện)
- **Phần định nghĩa các hàm đã được khai báo prototype**

Ví dụ **:** Viết chương trình nhập vào 2 số nguyên a,b và xuất ra màn hình số lớn nhất trong 2 số (sử dụng hàm)

```
#include <iostream.h>// Khai báo thư viện iostream.h
#include <conio.h>// Khai báo thư viện conio.h
int max(int x, int y);// khai báo nguyên mẫu hàm max
(prototype)

void main()//hàm main (sẽ gọi các hàm thực hiện)
{
      int a, b;// khai báo biến
      cout<<" Nhap vao 2 so a, b ";
      cin>>a>>b;
      cout<<"so lon nhat la:"<< max(a,b);
      getch();
      return;
}
int max(int x, int y)// Định nghĩa hàm max(a,b) đã được
khai báo prototype
{
      return (x>y) ? x:y;
}
```

BÀI TẬP CHƯƠNG 6

Viết lại tất cả bài tập chương 3 và 4 dưới dạng hàm.

Chương 7
CHUỖI KÝ TỰ
(Strings)

1. Chuỗi

Trong C/C++, một chuỗi là một mảng ký tự với ký tự null ở cuối chuỗi. Ký tự null (‘\0’) là ký tự dùng để kết thúc chuỗi. Như vậy, một chuỗi bao gồm các ký tự tạo nên chuỗi và theo sau là ký tự null. Khi khai báo một mảng ký tự dùng để chứa chuỗi, ta cần khai báo nó dài hơn 1 byte để chứa ký tự null.

Ví dụ: để khai báo một mảng str để chứa chuỗi có độ dài 10 ký tự, ta phải khai báo như sau: char str[11];

Hằng chuỗi là chuỗi được bao quanh bởi cặp dấu nháy đôi. Ví dụ: "Hello" là một hằng chuỗi. Ta không cần thêm ký tự null vào sau chuỗi vì trình biên dịch sẽ làm điều này tự động.

2. Khai báo và khởi tạo chuỗi

Có 2 cách khai báo và khởi tạo chuỗi. Giả sử khai báo và khởi tạo chuỗi “Hello”.

Cách 1: Dùng mảng một chiều

char str[] = {‘H’,’e’,’l’,’l’,’o’,’\0’};

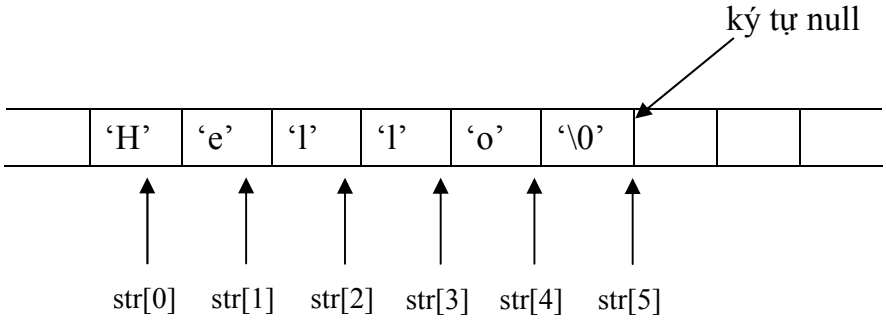
Lưu ý: trong trường hợp này, ta phái thêm ký tự null vào cuối.

hoặc

char str[] = “Hello”;

Lưu ý không cung cấp ký tự null.

Chuỗi trên được lưu trữ trong bộ nhớ như sau:



Cách 2: Dùng con trỏ

char *str = “Hello”;

3. Nhập chuỗi

Để nhập dữ liệu cho biến chuỗi, ta dùng hàm gets() của thư viện stdio.h. Hàm này có cú pháp sau:

```
char *gets(char *s);
```

Hàm gets() đọc các ký tự từ bàn phím (stdin) vào trong mảng trỏ đến bởi s cho đến khi nhấn Enter. Ký tự null sẽ được đặt sau sau ký tự cuối cùng của chuỗi nhập vào trong mảng.

Hoặc ta có thể dùng cin (Console INput). Cú pháp như sau:

```
cin >> s;
```

4. Xuất chuỗi

Để xuất chuỗi ra màn hình, ta dùng hàm puts() của thư viện stdio.h. Hàm này có cú pháp sau:

```
int puts(const char *s);
```

Hoặc ta có thể dùng cout (Console OUTput). Cú pháp như sau:

```
cout << s;
```

5. Một số hàm thư viện thao tác trên chuỗi

Để sử dụng các hàm này, ta phải khai báo dòng lệnh sau: #include <string.h>

strcpy(s1, s2)	Sao chép chuỗi s2 vào s1
strcat(s1, s2)	Nối chuỗi s2 vào cuối chuỗi s1
strlen(s1)	Trả về độ dài của chuỗi
strcmp(s1, s2)	Trả về 0 nếu s1 và s2 giống nhau, giá trị nhỏ hơn 0 nếu s1<s2 và giá trị lớn hơn 0 nếu s1>s2

strchr(s1, ch)	Trả về con trỏ đến vị trí xuất hiện đầu tiên của ký tự ch trong chuỗi s1
strstr(s1, s2)	Trả về con trỏ đến vị trí xuất hiện đầu tiên của chuỗi s2 trong s1

6. Một số ví dụ về chuỗi

Ví dụ 1:

```
#include <stdio.h>
#include <iostream.h>
#include <string.h>

void main()
{
  char s1[80], s2[80];
  cout << "Input the first string: ";
  gets(s1);
  cout << "Input the second string: ";
  gets(s2);

  cout << "Length of s1= " << strlen(s1);
  cout << "Length of s2= " << strlen(s2);

  if(!strcmp(s1, s2))
     cout << "These strings are equal\n";

  strcat(s1, s2);
  cout << "s1 + s2: " << s1 << endl;;

  strcpy(s1, "This is a test.\n");
  cout << s1;

  if(strchr("hello", 'e')) cout << "e is in hello\n";

  if(strstr("hi there", "hi")) cout << "found hi";
}
```

Ví dụ 2: Nhập một chuỗi str, nhập một ký tự ch. Cho biết ch xuất hiện bao nhiêu lần trong chuỗi str.

```
#include <stdio.h>
#include <iostream.h>
#include <string.h>
void main()
{
  char str[80], ch;
  int num=0;
  cout << "Input str: ";
  gets(str);
  cout << "Input ch: ";
  cin >> ch;

  for(int i=0 ; i<strlen(str) ; i++)
     if(str[i] == ch) num++;
  cout << ch << " is appeared " << num << " times.";
}
```

7. Mảng các chuỗi

Để tạo một mảng các chuỗi, dùng một mảng ký tự hai chiều. Kích thước của chỉ mục thứ nhất là số chuỗi và kích thước của chỉ mục thứ hai xác định chiều dài lớn nhất của mỗi chuỗi.

Đọan mã dưới đây khai báo một mảng của 5 chuỗi, mỗi chuỗi có chiều dài tối đa là 79 ký tự.

char str[5][80];

Để nhập dữ liệu cho chuỗi thứ nhất từ bàn phím, ta dùng lệnh:

gets(str[0]);

cin >> str[0] //Tuong duong voi lenh tren

Để xuất chuỗi thứ hai ra màn hình, ta dùng lệnh:

puts(str[1]);

cout << str[1]; //Tuong duong voi lenh tren

Khai báo và khởi tạo mảng các chuỗi

```
char arrayList[][length] = {    constantString_1,
                                constantString_2,
                                ...
                                constantString_n};
```

arrayList: Tên của mảng chuỗi

constantString_1, ..., constantString_n : Các hằng chuỗi

Ví dụ: Để khai báo một mảng danh sách các ngôn ngữ lập trình thông dụng, ta khai báo như sau:

char listOfPL[][10] = {“Pascal”, “C/C++”, “CSharp”, “Java”, “VB”};

Câu lệnh trên sẽ khai báo mảng listOfPL gồm 5 chuỗi.

Mảng chuỗi trên được lưu trữ trong bộ nhớ như sau:

P	a	s	c	a	l	‘\0’			
C	/	C	+	+	‘\0’				
C	S	h	a	r	p	‘\0’			
J	a	v	a	‘\0’					
V	B	‘\0’							

Lưu ý vị trí của các ký tự null

Ví dụ:

Nhập tên của 5 người dùng mảng char hai chiều, in chúng ra màn hình.

```
#include <stdio.h>
#include <iostream.h>
#include <string.h>
```

```
void main()
{
      char name[5][20];
      for(int i=0 ; i<5 ; i++)
      {
            cout << "Input name " << i+1 <<": ";
            cin >> name[i];
      }

      cout << "List of names: ";
      for(int i=0 ; i<5 ; i++)
            cout << name[i] << ", ";
}
```

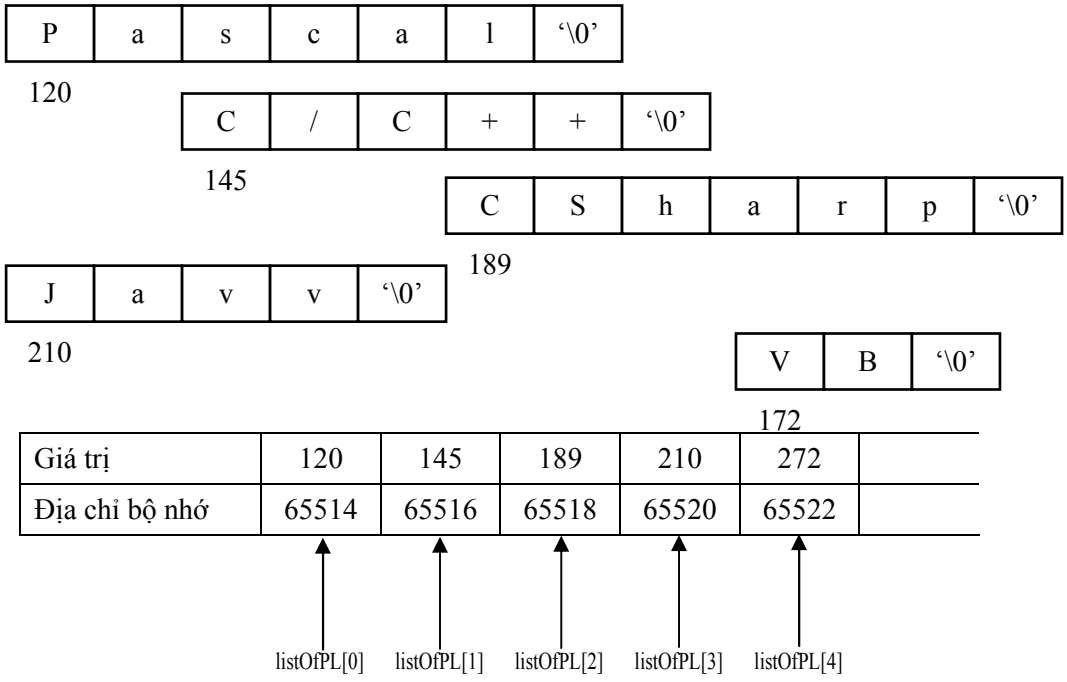
8. Mảng con trỏ đến các chuỗi

Ngoài cách dùng mảng ký tự hai chiều để lưu trữ mảng các chuỗi, ta có thể dùng mảng của các con trỏ. Mỗi con trỏ sẽ chứa địa chỉ của chuỗi.

Cũng ví dụ như phần trên, ta dùng mảng con trỏ

char *listOfPL[] = {"Pascal", "C/C++", "CSharp", "Java", "VB"};

Mảng con trỏ trên có thể được lưu trữ trong bộ nhớ như sau:



Ví dụ: Nhập tên của 5 người dùng mảng con trỏ, in chúng ra màn hình.

```
#include <stdio.h>
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
void main()
{
      char *name[5];
      for(int i=0 ; i<5 ; i++)
            name[i] = (char *)malloc(20);

      for(int i=0 ; i<5 ; i++)
      {
      cout << "Input name " << i+1 <<": ";
      gets(name[i]);
            }

      cout << "List of names: ";
      for(int i=0 ; i<5 ; i++)
            cout << name[i] << ", ";
}
```

BÀI TẬP CHƯƠNG 7

1. Viết chương trình nhập một chuỗi ký tự từ bàn phím, xuất ra màn hình mã ASCII của từng ký tự vừa nhập vào (gợi ý mỗi ký tự trên một dòng).

2. Viết chương trình nhập một chuỗi ký tự từ bàn phím, xuất ra màn hình đảo ngược của chuỗi đó. Ví dụ đảo của "abcd egh" là "hge dcba".

3. Viết chương trình nhập một chuỗi ký tự và kiểm tra xem chuỗi đó có đối xứng không.

Ví dụ : ABCDEDCBA là đối xứng.

4. Nhập vào một chuỗi ký tự bất kỳ, hãy đếm số lần xuất hiện của mỗi loại ký tự.

5. Viết chương trình nhập vào một chuỗi ký tự.

a) In ra màn hình từ bên trái nhất và phần còn lại của chuỗi. Ví dụ: "Nguyễn Văn Minh" in ra thành:

Nguyễn

Văn Minh

b) In ra màn hình từ bên phải nhất và phần còn lại của chuỗi. Ví dụ: "Nguyễn Văn Minh" in ra thành:

Minh

Nguyễn Văn

6. Viết chương trình nhập vào một chuỗi rồi xuất chuỗi đó ra màn hình dưới dạng mỗi từ một dòng.

Ví dụ: "Nguyễn Văn Minh"

In ra :

Nguyễn

Văn

Minh

7. Viết chương trình nhập vào một chuỗi, in ra đảo ngược của chuỗi đó theo từng từ.

Ví dụ : "Nguyễn Văn Minh" đảo thành "Minh Văn Nguyễn"

8. Viết chương trình đổi số tiền từ số thành chữ.

9. Viết chương trình nhập vào họ và tên của một người, cắt bỏ các khoảng trống không cần thiết (nếu có), tách tên ra khỏi họ và tên, in tên lên màn hình. Chú ý đến trường hợp cả họ và tên chỉ có một từ.

10. Viết chương trình nhập vào họ và tên của một người, cắt bỏ các khoảng trắng bên phải, trái và các khoảng trắng không có nghĩa trong . In ra màn hình toàn bộ họ tên người đó dưới dạng chữ hoa, chữ thường.

11. Viết chương trình nhập vào một danh sách họ và tên của n người theo kiểu chữ thường, đổi các chữ cái đầu của họ, tên và chữ lót của mỗi người thành chữ hoa. In kết quả lên màn hình.

12. Viết chương trình nhập vào một danh sách họ và tên của n người, tách tên từng người ra khỏi họ và tên rồi sắp xếp danh sách tên theo thứ tự từ điển. In danh sách họ và tên sau khi đã sắp xếp.

Chương 8

STRUCTURES – ENUM - typedef

Ngôn ngữ C/C++ đưa ra 5 cách để tạo nên một kiểu dữ liệu tùy biến (*custom data types*).

1. Kiểu cấu trúc (*structure*): Là một nhóm của các biến được định nghĩa dưới một tên. Kiểu này còn gọi là kiểu dữ liệu phức hợp (*compound data types*).

2. bit-field: là một biến thể của kiểu structure và cho phép dễ dàng truy cập đến từng bit riêng rẽ.

3. Union: cho phép cùng một mẫu bộ nhớ được định nghĩa như hai hay nhiều kiểu biến khác nhau.

4. Enumeration: là danh sách của của các tên hằng nguyên.

5. Từ khóa typedef: định nghĩa một tên khác cho một kiểu dữ liệu đã có.

Trong phần này chỉ thảo luận structures, enumerations, và typedef.

1. Structures

Một cấu trúc là một tập các biến được tham chiếu thông qua một tên chung. Một khai báo cấu trúc hình thành một khuôn mẫu (*template*) mà có thể dùng để tạo nên các biến cấu trúc có cùng kiểu. Những biến mà tạo nên cấu trúc được gọi là các thành viên (*members*).

Nói chung, tất cả các thành viên của một cấu trúc về mặt logic là có liên quan với nhau. Ví dụ sau đây khai báo một cấu trúc address gồm các thông tin về một địa chỉ. Từ khóa struct dùng để khai báo một cấu trúc. Xem xét khai báo sau:

```
struct addr
{
      char name[30];
      char street[40];
      char city[20];
      char state[3];
```

```
    unsigned long int zip;
};
```

Tại thời điểm này, ta mới chỉ có khai báo một cấu trúc. Để khai báo một hoặc nhiều biến có kiểu address, ta dùng tên cấu trúc như bất kỳ kiểu dữ liệu nào. Ví dụ, để khai báo 2 biến kiểu address, ta khai báo như sau:

```
address addr1, addr2;
```

Khi một biến cấu trúc được khai báo, trình biên dịch tự động cấp phát đủ bộ nhớ cho tất cả thành viên của cấu trúc.

1.1. Dạng tổng quát của một khai báo cấu trúc

```
struct structureName
{
   type member_1;
   type member_2;
   ...
   type member_n;
   .. .
} varNames;
```

structureName: Tên của cấu trúc

type: Kiểu dữ liệu của thành viên tương ứng

member_1, member_2, ..., member_n: Tên các biến thành viên của cấu trúc

varNames: Tên các biến cấu trúc phân cách nhau bằng dấu phẩy.

1.2. Truy cập các thành viên của biến cấu trúc

Toán tử dấu chấm (dot operator) dùng để truy cập (access) các thành viên của một biến cấu trúc. Dạng tổng quát để truy cập một thành viên của một biến cấu trúc là:

```
structureName.memberName
```

Ví dụ: Xem xét khai báo cấu trúc sau

```
struct coordXY
{     int x;
      int y;
} diemA, diemB;
```

Để gán giá trị tọa độ cho diemA, ta dùng các lệnh sau:

```
diemA.x = 100;
diemA.y = 200;
```

Để in tọa độ điểm A, ta dùng lệnh sau:

```
cout < "A(" << diemA.x << "," << diemA.y << ")";
```

1.3. Lệnh gán cấu trúc

Nội dung trong 1 biến cấu trúc có thể gán cho một biến cấu trúc khác có cùng kiểu dùng một câu lệnh gán. Ví dụ, để gán nội dung biến cấu trúc pointA cho biến pointB, ta thực hiện lệnh sau:

```
pointB = pointA;
```

Sau câu lệnh này, biến pointB có cùng nội dung như biến pointA. Tuy nhiên, ta cũng có thể sao chép từng thành viên như sau:

```
pointB.x = pointA.x;
pointB.y = pointA.y;
```

Ví dụ: Khai báo, nhập và xuất cấu trúc book gồm các thông tin title, author, pages, price.

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

void main()
{     struct book { char title[30];
                    char author[20];
                    int pages;
                    float price;
```

```
            };
      book b;
      cout << "Input book information:" << endl;
      cout << "Title: "; gets(b.title);
      cout << "Author: "; gets(b.author);
      cout << "Number of pages: "; cin >> b.pages;
      cout << "Price: "; cin >> b.price;

      cout << "Information of this book is:" << endl;
      cout << "Title: " << b.title << endl;
      cout << "Author: " << b.author << endl;
      cout << "Pages: " << b.pages << endl;
      cout << "Price: " << b.price << endl;
}
```

1.4. Mảng các cấu trúc

Để khai báo một mảng các cấu trúc, đầu tiên ta khai báo cấu trúc, sau đó khai báo một mảng của cấu trúc đó. Ví dụ, để khai báo mảng points có 100 phần tử, ta khai báo như sau:

```
coordXY points[100];
```

Để truy cập (*access*) đến từng thành viên của từng phần tử của mảng, ta dùng chỉ mục của phần tử và toán tử thành viên (.). Ví dụ, để gán tọa độ x,y cho phần tử thứ 10, ta dùng các lệnh:

```
points[9].x = 100;
points[9].y = 200;
```

1.5. Truyền các cấu trúc vào hàm

a. Truyền các thành viên của biến cấu trúc vào hàm

Khi ta truyền một thành viên của một cấu trúc vào một hàm, ta thật sự truyền giá trị của thành viên đó cho tham số hình thức của hàm

(truyền tham trị). Nếu muốn truyền địa chỉ của thành viên cho hàm (truyền tham chiếu) ta đặt phía trước dấu &.

Ví dụ: Giả sử ta có hàm int distanceAB(int x1, int y1, int x2, int y2) để tính khoảng cách giữa 2 điểm. Để tính khoảng cách giữa 2 điểm nào đó, ta truyền tọa độ x,y của 2 điểm tương ứng. Ta dùng lệnh sau:

```
length1 = distance(pointA.x, pointA.y, pointB.x, pointB.y);
```

Lệnh trên gọi hàm distance và truyền tọa độ x,y của hai điểm A,B. Kết quả thực hiện hàm trả về gán cho biến length.

Để truyền địa chỉ của thành viên của cấu trúc vào hàm dùng toán tử & đặt phía trước tên biến cấu trúc chứ không đặt trước tên của thành viên của biến cấu trúc.

Ví dụ: Ta có hàm void move(int *x, int *y, int delta_x, int delta_y); dùng để thay đổi tọa độ x,y của một điểm; delta_x lượng di chuyển theo chiều ngang; delta_y lượng di chuyển theo chiều dọc. Vậy, để thay đổi tọa độ biến cấu trúc pointA, ta dùng lệnh sau:

move(&pointA.x, &pointA.y, 10, 20);

b. Truyền toàn bộ biến cấu trúc đến hàm

Khi một cấu trúc được dùng như một đối số của một hàm, tòan bộ cấu trúc được truyền dùng cách truyền tham trị. Với cách này, hàm không thể làm thay đổi nội dung của đối số. Tuy nhiên, nếu muốn hàm có thể làm thay đổi nội dung của đối số, ta truyền tham chiếu (thêm dấu & vào trước đối số).

Ví dụ ta có hàm distance2 có khai báo nguyên mẫu như sau:

int distance2(point p1, point p2);

Để gọi hàm trên tính khỏang cách của 2 điểm pointA và pointB, ta dùng lệnh sau:

length2 = distance2(pointA, pointB);

Trong trường hợp này, ta sao chép nội dung của 2 biến cấu trúc pointA, pointB vào 2 tham số hình thức p1 và p2 của hàm distance2.

1.6. Con trỏ đến cấu trúc

C/C++ cho phép các con trỏ đến các cấu trúc như đến bất kỳ kiểu dữ liệu nào của biến.

Khai báo một con trỏ cấu trúc

Cú pháp khai báo con trỏ cấu trúc giống như các loại con trỏ khác. Dạng tổng quát để khai báo con trỏ cấu trúc:

```
structureName *structurePointers;
```

1.7. Sử dụng con trỏ cấu trúc

Để tham chiếu đến thành viên của một cấu trúc được trỏ đến bởi một con trỏ, ta dùng toán tử -> (toán tử tham chiếu gồm một dấu trừ và một dấu lớn hơn).

Xem xét ví dụ sau:

```
points *p; //khai báo con trỏ p có kiểu cấu trúc points
p = &pointA; //gán địa chỉ của biến cấu trúc pointA cho con trỏ p
p->x = 100; //gán giá trị 100 cho thành viên x của biến cấu trúc pointA
```

Lưu ý: Để truy cập đến thành viên của một cấu trúc, nếu dùng biến cấu trúc thì dùng toán tử chấm (dot operator), nếu dùng biến con trỏ thì dùng toán tử -> (arrow operator).

Khi con trỏ cấu trúc được truyền vào một hàm thì hàm có thể thay đổi nội dung của biến cấu trúc đó vì cách truyền này là truyền tham chiếu.

2. Kiểu liệt kê (Enumerations, enum)

Một enum là một tập của các tên hằng nguyên mà xác định tất cả các giá trị hợp lệ mà một biến của kiểu đó có thể có.

Ví dụ, ta có một enum là danh sách giá trị tiền tệ sau:
one$, two$, five$, ten$, twenty$, fifty$, hundred$

Dạng tổng quát để khai báo một enum là

```
enum enumName {enumList} enumVars;
```

enum: từ khóa để khai báo enum

enumName: Tên của enum

enumList: Danh sách các tên hằng nguyên phân cách nhau bởi dấu phẩy

enumVars: Tên các biến kiểu enum.

Ví dụ, khai báo enum trên

```
enum money {one$,two$,five$,ten$,twenty$,fifty$,hundred$} m1, m2;
```

Khai báo hai biến m1, m2 có kiểu money

Khảo sát các lệnh sau:

```
m1 = one$;
m2 = ten$;
if(m1 == m2) cout << "They are same.";
if(m1 == one$) cout << "m1 is one dollar.";
if(m2 != five$) cout << "m2 is not five-dollar.";
```

Điểm quan trọng để hiểu về enum là mỗi một tên trong danh sách enum tượng trưng cho một giá trị nguyên. Giá trị của tên thứ nhất trong enum là 0, kế tiếp là 1, ... Trong khai báo trên giá trị của các tên lần lượt là:

one$	0
two$	1
five$	2

ten$ 3
twenty$ 4
fifty$ 5
hundred$ 6

Ta có thể gán giá trị khác cho mỗi tên hằng nguyên như trong câu lệnh sau:

enum money {one$=1, two$=2, five$=5, ten$=10, twenty$=20, fifty$=50, hundred$=100};

Lệnh này sẽ gán mỗi tên hằng nguyên một giá trị đứng sau dấu bằng.

one$ 1
two$ 2
five$ 5
ten$ 10
twenty$ 20
fifty$ 50
hundred$ 100

3. typedef

Từ khóa typedef dùng để định nghĩa một tên mới cho một kiểu dữ liệu đã có. Dạng tổng quát của dùng typedef là

```
typedef existingType newType;
```

existingType: là bất kỳ kiểu dữ liệu nào đã tồn tại

newType: tên mới của kiểu dữ liệu

Ví dụ: để tạo một tên mới cho kiểu dữ liệu nguyên

```
typedef int int2bytes; //Kiểu int có thêm một tên mới là int2bytes
typedef long int4bytes; //Kiểu long có thêm một tên mới là int4bytes
```

Sau khi các lệnh trên thực hiện thì lệnh

```
long n1, n2; //Khai báo 2 biến long
```

tương đương

```
int4bytes n1, n2;
```

BÀI TẬP CHƯƠNG 8

1. Cho cấu trúc NHANVIEN như sau:

- MaNV: kiểu số nguyên có giá trị trong khoảng 0…65535
- Họtên: kiểu chuỗi.
- Địachỉ: kiểu chuỗi.
- CBQL: có giá trị 1 nếu nhân viên này là cán bộ quản lý.

Yêu cầu chương trình thực hiện:
(a) Viết hàm nhập vào thông tin của một nhân viên.
(b) Viết hàm xuất thông tin của một nhân viên.
(c) Viết hàm main có yêu cầu nhập vào n nhân viên với n được nhập từ bàn phím. In ra họ tên của các nhân viên là cán bộ quản lý.

2. Cho cấu trúc NHANVIEN như bài 1:

Nhập viết hàm Main có yêu cầu nhập vào n nhân viên với n được nhập từ bàn phím. Xóa các nhân viên không là cán bộ quản lý ra khỏi danh sách.

3. Cho cấu trúc NHANVIEN như bài 1

Viết hàm main có yêu cầu nhập vào n nhân viên với n được nhập từ bàn. Nhập thêm thông tin của một nhân viên và nhập một số nguyên k. Thực hiện việc chèn nhân viên mới vào danh sách tại vị trí k.

Chương 10

TẬP TIN
(Files)

C/C++ hổ trợ 2 hệ thống nhập xuất. Một hệ thống thừa kế từ ngôn ngữ C và một hệ thống nhập xuất hướng đối tượng của C++. Trong chương này, ta chỉ khảo sát hệ thống thứ nhất.

1. Streams và Files

Hệ thống nhập xuất của C cung cấp một giao diện (*interface*) nhất quán cho lập trình viên mà độc lập với thiết bị thật sự mà chương trình tương tác. Nghĩa rằng hệ thống nhập xuất của C cung cấp một mức độ trừu tượng giữa lập trình viên và thiết bị nhập xuất. Sự trừu tượng này được gọi là stream và thiết bị thật sự được gọi là file.

2. Streams (dòng nhập xuất)

Hệ thống file của C được thiết kế để làm việc với nhiều loại thiết bị khác nhau như terminals (*thiết bị đầu cuối*), các loại ổ đĩa, băng từ, ... Mặc dầu mỗi thiết bị là rất khác nhau, hệ thống file chuyển đổi mỗi loại thành một thiết bị logic gọi là stream. Tất cả stream có cùng hành vi. Bởi vì stream thì độc lập với thiết bị nên cùng một hoạt động trên stream như viết vào một tập tin trên đĩa cũng có thể dùng để viết đến loại thiết bị khác như console (*màn hình*). Có hai loại stream: văn bản (*text*) và nhị phân (*binary*).

2.1. Text Streams

Một text stream là một chuỗi các ký tự. Trong một text stream, một số ký tự có thể bị chuyển đổi (được hiểu như là một ký tự khác) tùy thuộc môi trường. Ví dụ, ký tự newline ('\n') có thể bị đổi thành cặp ký tự carriage return/linefeed (*ký tự xuống dòng và về đầu dòng*). Vì vậy, không có sự quan hệ một-một giữa các ký tự được viết (hay đọc) và những ký tự trên các thiết bị ngoài. Do đó, bởi vì có khả năng xảy ra sự chuyển đổi, nên số số ký tự được viết (hay đọc) có thể khác số số ký tự trên thiết bị ngoài.

2.2. Binary Streams

Một binary stream là một chuỗi bytes mà có sự tương ứng một-một với chuỗi bytes trên thiết bị ngoài. Nghĩa là không có sự chuyển đổi xảy ra. Do đó, số bytes được viết (hay đọc) thì bằng với số bytes trên thiết bị ngoài.

3. Files

Một file có thể là một tập tin trên đĩa, một terminal, hay máy in. Để tạo kết nối (*associate*) giữa một stream với một file ta dùng hoạt động mở (*open*). Một khi một file được mở, thông tin có thể được trao đổi giữa nó và chương trình.

Không phải tất cả file đều có cùng khả năng như nhau. Ví dụ, một tập tin trên đĩa (*file*) có thể hỗ trợ truy xuất ngẫu nhiên trong khi đó máy in (cũng là file) thì không thể. Việc này đưa đến một kết luận là: ***"Tất cả stream là như nhau nhưng file thì không".***

Để ngắt kết nối giữa một stream với một file ta dùng hoạt động đóng (*close*). Nếu đóng một file đang mở cho xuất (*output*) thì nội dung (nếu có) của stream tương ứng được viết ra thiết bị ngoài. Qúa trình này được gọi là flushing và đảm bảo là không có thông tin bị để lại trong vùng đệm (*buffer*). Tất cả file được tự động đóng khi chương trình mở chúng kết thúc bình thường. Files không được đóng khi chương trình mở chúng bị kết thúc bất thường như bị treo (*halt*) hay khi chương trình thực hiện hàm abort().

Mỗi stream liên đới với một file có một cấu trúc kiểu FILE.

3.1. Cơ bản về hệ thống file

C/C++ có nhiều hàm liên quan nhau hoạt động trên file. Những hàm này yêu cầu tập tin header stdio.h. Sau đây là danh sách các hàm:

Tên hàm	**Chức năng**
fopen( )	Mở một file
fclose( )	Đóng một file.
putc( )	Viết một ký tự đến một file.
fputc( )	Giống như **putc()** .
getc( )	Đọc một ký tự từ một file.

fgetc( )	Giống như **getc()** .
fgets( )	Đọc một chuỗi từ một file.
fputs( )	Viết một chuỗi đến một file.
fseek( )	Tìm một byte trong một file.
ftell( )	Trả về vị trí hiện hành của của file indicator.
feof( )	Trả về true nếu duyệt đến cuối file (end-of-file).
ferror( )	Trả về true nếu một lỗi xảy ra.
rewind( )	Đưa indicator về đầu.
remove( )	Xóa một file.
fflush( )	Xả hết vùng đệm của file.

Tập tin header **stdio.h** cung cấp các nguyên mẫu cho các hàm nhập xuất file. Ngoài ra còn có các macro như NULL, EOF, FOPEN_MAX, SEEK_SET, SEEK_CUR và SEEK_END. Macro NULL định nghĩa một con trỏ null. Macro EOF được định nghĩa là -1, là giá trị trả về khi hàm đọc file đến vị trí cuối của file. FOPEN_MAX định nghĩa một giá trị nguyên chỉ ra số file có thể mở đồng thời. Các macro còn lại hoạt động với hàm fseek() để thi hành hoạt động truy cập file ngẫu nhiên.

3.2. Con trỏ file (*File pointer*)

Một con trỏ file là một cấu trúc kiểu FILE. Nó trỏ đến thông tin mà định nghĩa nhiều thứ về file như tên file, trạng thái, và vị trí hiện hành của file. Con trỏ file được dùng bởi stream tương ứng để thực hiện các hoạt động nhập xuất trên file. Để đọc hay viết file, chương trình phải dùng con trỏ file. Để khai báo một biến con trỏ file, dùng lệnh:

```
FILE *fp;
```

3.3. Mở file

Hàm fopen() mở một stream để dùng và liên kết một file với stream đó. Hàm trả về một con trỏ file liên đới với tập tin được mở. Hàm fopen() có nguyên mẫu sau:

```
FILE *fopen(const char *filename, const char *mode);
```

filename: Là một hằng chuỗi chứa tên (và đường dẫn) của file
mode: Là một hằng chuỗi cho biết mở file theo mode nào.

Các mode để mở tập tin

Dưới đây là danh sách các mode để mở một tập tin.

"r" Nếu tập tin được mở thành công, hàm fopen() nạp nó vào trong bộ nhớ và trả về một con trỏ trỏ đến ký tự đầu tiên của tập tin. Nếu không thể mở tập tin, hàm fopen() trả về NULL

Các hoạt động có thể làm trên tập tin: đọc (read)

"w" Nếu tập tin tồn tại, nội dung của nó sẽ bị viết đè. Nếu tập tin không tồn tại, một tập tin mới được tạo. Trả về NULL nếu không thể mở tập tin.

Các hoạt động có thể làm trên tập tin: viết (write)

"a" Nếu tập tin được mở thành công, hàm fopen() nạp nó vào trong bộ nhớ và trả về một con trỏ trỏ đến ký tự cuối cùng của tập tin. Nếu tập tin không tồn tại, một tập tin mới được tạo. Trả về NULL nếu không thể mở tập tin.

Các hoạt động có thể làm trên tập tin: thêm (append) nội dung mới vào cuối tập tin.

"r+" Nếu tập tin được mở thành công, hàm fopen() nạp nó vào trong bộ nhớ. Trả về NULL nếu không thể mở tập tin.

Các hoạt động có thể làm trên tập tin: viết nội dung mới vào tập tin, đọc nội dung tập tin, và sửa đổi nội dung của tập tin.

"a+" Nếu tập tin được mở thành công, hàm fopen() nạp nó vào trong bộ nhớ và trả về một con trỏ trỏ đến ký tự đầu tiên của tập tin. Nếu tập tin không tồn tại, một tập tin mới được tạo. Trả về NULL nếu không thể mở tập tin.

Các hoạt động có thể làm trên tập tin: đọc nội dung tập tin, viết thêm nội dung mới vào cuối tập tin. Không thể sửa đổi nội dung đang có trong tập tin.

Ví dụ sau minh họa mở một file test.txt để viết

```
FILE *fp;
fp = fopen("test.txt", "w");
```

Đoạn mã trên là đúng nhưng thực tế đoạn mã trên được viết như sau:

```
FILE *fp;
if((fp = fopen("test.txt","w")) == NULL)
{    cout << "Cannot open file";
     exit(1);
}
```

Phương pháp này sẽ dò tìm bất kỳ lỗi nào khi mở file để viết như file được bảo vệ chống ghi hay đĩa bị đầy không thể tạo thêm file. Nói chung, ta phải luôn kiểm tra xem việc mở file có thành công hay không trước khi làm bất kỳ hành động nào trên file.

Một file có thể mở ở mode text hay binary. Mặc định là mở file ở text mode. Nếu muốn mở file ở binary mode thì thêm ký tự b vào chuỗi mode ở trên như "wb", "rb", ...
Số file được phép mở đồng thời được xác định bởi macro FOPEN_MAX. Giá trị này thường nhỏ nhất là 8.

3.4. Đóng file

Hàm fclose() đóng stream được mở bởi hàm fopen(). Khi hàm được gọi, nó sẽ viết bất kỳ dữ liệu nào vẫn còn trong buffer đến file rồi đóng file. Hàm fclose() có nguyên mẫu sau:

```
int fclose(FILE *fp);
```

fp: là con trỏ file trả về bởi hàm fopen().
Nếu đóng file thành công, hàm trả về giá trị zero. Nếu một lỗi xảy ra khi đóng file, hàm trả về EOF.

3.5. Viết một ký tự đến một file

Có hai hàm xuất ký tự đến file là putc() và fputc(). Hai hàm này là tương đương nhau. Hàm putc() viết một ký tự đến một file đã được mở bởi hàm fopen(). Nguyên mẫu của hàm như sau:

```
int putc(int ch, FILE *fp);
```

fp là con trỏ file trả về bởi hàm fopen() và ch là ký tự được viết đến file.
Nếu hoạt động putc() thành công, nó trả về ký tự được viết vào file. Ngược lại, nó trả về EOF.

3.6. Đọc một ký tự từ một file

Có hai hàm tương đương để đọc một ký tự từ file là getc() và fgetc(). Hàm getc() đọc mỗi lần một ký tự từ file được mở bởi hàm fopen() ở chế độ đọc (read). Nguyên mẫu của fopen là:

```
int getc(FILE *fp);
```

fp là con trỏ file kiểu FILE trả về bởi hàm fopen(). Hàm getc() trả về một số nguyên là giá trị của ký tự được đọc. Hàm getc() trả về EOF nếu một lỗi xảy ra.

3.7. Ví dụ minh họa fopen(), getc(), putc(), và fclose()

Ví dụ 1: Minh họa việc đọc từ bàn phím và ghi chúng vào một tập tin cho đến khi người dùng nhấn nhập ký tự $.

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
     FILE *fp;
     char ch;
     if((fp=fopen("test.txt", "w")==NULL)
     {    cout << "Cannot open file.\n";
          exit(1);
     }
     do {
          ch = getchar();
          putc(ch, fp);
     }while (ch != '$');
     fclose(fp);
}
```

Ví dụ 2: Minh họa việc đọc từ một file văn bản và xuất chúng ra màn hình.

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
     FILE *fp;
```

```
      char ch;
      if((fp=”test.txt”, "r")==NULL)
      {
            cout << "Cannot open file.\n";
            exit(1);
      }
      ch = getc(fp); // read one character
      while (ch!=EOF)
      {
            putchar(ch); // print on screen
            ch = getc(fp);
      }
      fclose(fp);
}
```

3.8. Đọc và viết chuỗi trên file

C/C++ hỗ trợ hai hàm fgets() và fputs() để đọc và viết chuỗi ký tự trên file. Những hàm này tương tự như getc() và putc() nhưng thay vì đọc hay viết từng ký tự, chúng đọc hay viết một chuỗi.
Nguyên mẫu của các hàm trên như sau:

```
int fputs(const char *str, FILE *fp);
char *fgets(char *str, int length, FILE *fp);
```

Hàm fputs() viết một chuỗi trỏ đến bởi str đến stream trỏ đến bởi con trỏ file fp. Hàm trả về EOF nếu một lỗi xảy ra.
Hàm fgets() đọc một chuỗi từ stream tương ứng cho đến khi gặp ký tự newline hay đã đọc được length-1 ký tự. Hàm trả về str nếu đọc thành công và một con trỏ null nếu không.

Chương trình sau minh họa hàm fputs(). Nó đọc các chuỗi từ bàn phím và viết chúng đến file tên teststr.txt. Để kết thúc chương trình, nhập một dòng trống

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void main(void)
{
      char str[80];
      FILE *fp;
```

```
      if((fp = fopen("teststr.txt", "w"))==NULL)
      {
            cout << "Cannot open file.\n";
            exit(1);
      }
      do
      {
            cout << "Enter a string (CR to quit):\n";
            gets(str);
            strcat(str, "\n"); /* add a newline */
            fputs(str, fp);
      } while(*str!='\n');
}
```

3.9. Hàm fread() và fwrite()

Để đọc và viết các kiểu dữ liệu có kích thước lớn hơn 1 byte, C/C++ cung cấp hai hàm fread() và fwrite(). Những hàm này cho phép đọc và viết một khối của bất kỳ dữ liệu nào. Nguyên mẫu của các hàm này như sau:

```
size_t fread(void *buffer, size_t numbytes, size_t count, FILE *fp);
size_t fwrite(const void *buffer, size_t numbytes, size_t count, FILE *fp);
```

Đối với fread(), buffer là một con trỏ đến một vùng bộ nhớ mà sẽ nhận dữ liệu đọc từ file. Đối với fwrite(), buffer là một con trỏ đến thông tin mà sẽ viết đến file. Giá trị của count cho biết bao nhiêu phần tử được đọc hay viết với mỗi phần tử có độ dài num_bytes. fp là con trỏ đến file đã được mở bởi fopen().

/* Write some non-character data to a disk file and read it back. */

```
#include <stdio.h>
#include <stdlib.h>
void main(void)
{
      FILE *fp;
      double d = 12.23;
      int i = 101;
      long l = 123023L;
      if((fp=fopen("testNchar.txt", "wb+"))==NULL)
      {
            cout << "Cannot open file.\n";
            exit(1);
      }
```

```
      fwrite(&d, sizeof(double), 1, fp);
      fwrite(&i, sizeof(int), 1, fp);
      fwrite(&l, sizeof(long), 1, fp);
      rewind(fp);
      fread(&d, sizeof(double), 1, fp);
      fread(&i, sizeof(int), 1, fp);
      fread(&l, sizeof(long), 1, fp);
      cout << "double= " << d << endl;
      cout << "integer= " << i << endl;
      cout << "long= " << l;
      fclose(fp);
}
```

Một trong những ứng dụng hữu dụng của fread() và fwrite() là đọc và viết kiểu dữ liệu người dùng định nghĩa như các cấu trúc.

Ví dụ 1: Nhập các mẫu tin nhân viên vào tập tin employee.dat

```
#include <stdio.h>
void main()
{
      FILE *fp;
      struct employee
      {
            char name[30];
            int age;
      };
      struct employee e;
      char more = 'Y';
      fp = fopen("employee.dat", "wb");
      if(fp == NULL)
      {     cout << "Cannot open file");
            exit(1);
      }
      while(more == 'Y'|| more == 'y')
      {
            cout << "\nEnter name and age: ";
            cin >> e.name >> e.age;
            fwrite(&e, sizeof(e), 1, fp);
            cout << "Input more (y/n)?: ";
            fflush(stdin);
            more = getche();
      }
      fclose(fp);
```

```
}
```

Ví dụ 2: Xuất các mẫu tin nhân viên có trong tập tin employee.dat

```
#include <stdio.h>
void main()
{
      FILE *fp;
      struct employee
      {
            char name[30];
            int age;
      };
      struct employee e;
      fp = fopen("employee.dat", "rb");
      if(fp == NULL)
      {     cout << "Cannot open file");
            exit(1);
      }
      while(fread(&e, sizeof(e), 1, fp) == 1)
      {
            cout << "\nName= " << e.name;
            cout << "\tAge= " << e.age;
      }
      fclose(fp);
}
```

Hàm rewind()

Hàm rewind() di chuyển indicator đến điểm bắt đầu của file. Hàm có prototype như sau:

```
void rewind(FILE *fp);
```

Hàm ferror()

Hàm ferror() cho biết một hoạt động trên file đã gây ra lỗi. Nguyên mẫu của hàm như sau:

```
int ferror(FILE *fp);
```

Hàm trả về true nếu một lỗi đã xảy ra với hoạt động trên file trước khi gọi hàm ferror(), ngược lại trả về false.

Xóa file

Hàm remove() dùng để xóa tập tin. Nguyên mẫu như sau:

```
int remove(const char *filename);
```

Hàm trả về zero nếu xóa thành công, ngược lại trả về nonzero

Flushing a stream

Hàm fflush() dùng để xuất tất cả nội dung còn lại trong buffer của stream. Hàm có nguyên mẫu sau:

```
int fflush(FILE *fp);
```

Hàm viết nội dung còn trong buffer đến file liên đới với fp. Nếu ta gọi hàm fflush() không có đối số thì tất cả flush tất cả file đang mở. Hàm trả về 0 nếu thành công, ngược lại trả về EOF

Truy xuất file ngẫu nhiên

Để đọc hay viết từ hay đến một vị trí bất kỳ trong file ta cần sự giúp đỡ của hàm fseek(). Hàm này dùng để di chuyển chỉ báo file. Hàm có nguyên mẫu sau:

```
int fseek(FILE *fp, long numbytes, int origin);
```

fp là con trỏ trả về bởi hàm fopen()
origin là một trong các giá trị sau: SEEK_SET (từ đầu file), SEEK_CUR (từ vị trí hiện hành), và SEEK_END (từ cuối file).
numbytes: số byte mà indicator di chuyển tùy thuộc origin cung cấp.

Các stream chuẩn

Khi một chương trình thực thi, ba stream được mở tự động. Đó là stdin (standard input), stdout (standard output), và stderr (standard error). stdin dùng để đọc từ bàn phím, stdout và stderr dùng để viết đến màn hình.
Bởi vì standard streams là các con trỏ file nên có thể dùng các hàm nhập xuất trên chúng.

BÀI TẬP CHƯƠNG 10

Viết hàm tạo một tập chứa 10000 số nguyên ngẫu nhiên khác nhau đôi một trong phạm vi từ 1 đến 32767 đặt tên là "SONGUYEN.INP"

1) Viết hàm đọc tập "SONGUYEN.INP", sau đó sắp xếp theo thứ tự tăng dần và lưu kết quả vào tập "SONGUYEN.OUT"
2) Viết hàm tạo tập văn bản có tên là "INPUT.TXT" có cấu trúc như sau
   - Dòng đầu tiên ghi N (N là số nguyên dương nhập từ bàn phím)
   - Trong các dòng tiếp theo ghi N số nguyên ngẫu nhiên trong phạm vi từ 0 đến 100, *mỗi dòng 10 số (các số cách nhau ít nhất một dấu cách)*
   - Hãy đọc dữ liệu của file "INPUT.TXT" và lưu vào mảng một chiều A

   Hãy thực hiện các công việc sau :
   a) Tìm giá trị lớn lớn nhất của mảng A
   b) Đếm số lượng số chẵn, số lượng số lẽ của mảng A
   c) Hãy sắp xếp các phần tử theo thứ tự tăng dần

   Hãy ghi các kết quả trên vào filetext có tên là "OUTPUT.TXT" theo mẫu sau:

INPUT.TXT

```
18
87 39 78 19 89  4 40 98 29 65
20  43   1  99  38  34  58   4
```

OUTPUT.TXT

```
Cau a: 99
cau b: 9  9
cau c: 1  4  4  19  20  29  34  38  39  40
       43 58 65 78 87 89 98 99
```

4) Viết hàm tạo tập văn bản có tên là "INPUT.TXT" có cấu trúc như sau

- Dòng đầu tiên ghi hai số M và N (M,N là các số nguyên dương nhập từ bàn phím)

- Trong M dòng tiếp theo mỗi dòng ghi N số nguyên ngẫu nhiên trong phạm vi từ 0 đến 50 (các số cách nhau ít nhất một dấu cách)

Hãy đọc dữ liệu của tập "INPUT.TXT" và lưu vào mảng hai chiều A. Hãy thực hiện các công việc sau:

a) Tìm giá trị lớn lớn nhất của mảng A
b) Đếm số lượng số chẵn, số lượng số lẽ của mảng A
c) Hãy tính tổng các phần tử trên mỗi dòng của mảng A

Hãy ghi các kết quả trên vào tập tin văn bản có tên là "OUTPUT.TXT" theo như mẫu trong ví dụ sau

INPUT.TXT

```
6  6
 41  17  33  23  12   1
 44  24  23  49   5  24
 33  20  17  25  33  19
  0  48  45  48  41  32
 10  24  36  19  19  24
 30   4  23  26  27  36
```

OUTPUT.TXT

```
Cau a:  49
Cau b:  17  19
Cau c:  127 169 147 214 132 146
```

5) Viết hàm đọc một file "SONGUYEN.INP" được tạo ở bài tập 1. Sau đó ghi các số chẵn vào file SOCHAN.OUT và các số lẻ vào file SOLE.OUT.

6) Viết một hàm gộp nội dung của hai tập tin có sẵn vào một tập tin thứ ba. Tên các tập tin được nhập vào từ bàn phím.

7) Viết một hàm tìm kiếm trên tập nhị phân có cấu trúc employee gồm tên và tuổi. Hiển thị kết quả là tất cả các thông tin về các nhân viên được tìm thấy khi cho biết :
   a) Tên nhân viên
   b) Có tuổi cao hơn một giá trị X nào đó.