

MỤC LỤC

CHƯƠNG 1 CƠ BẢN VỀ C/C++	5
1.1 Giới thiệu C và C++	5
1.2 Cấu trúc một chương trình C/C++	5
1.2.1 Cấu trúc của một chương trình C/C++	5
1.2.2 Một chương trình C/C++ đơn giản.....	6
1.3 Bộ ký tự của ngôn ngữ C/C++	7
1.4 Tên, kiểu dữ liệu, hằng, biến.....	7
1.4.1 Tên	8
1.4.2 Kiểu dữ liệu	8
1.4.2.1 Kiểu số nguyên	8
1.4.2.2. Kiểu số thực (kiểu dấu phẩy động).....	10
1.4.2.3. Kiểu ký tự	10
1.4.3 Hằng, biến.....	11
1.4.3.1 Hằng.....	11
1.4.3.2. Biến	12
1.5 Các lệnh nhập/ xuất dữ liệu trong C/C++	14
1.5.1 Lệnh nhập/xuất dữ liệu trong C.....	14
1.5.2 Lệnh nhập/ xuất dữ liệu trong C++	16
1.6 Biểu thức và phép toán.....	18
1.6.1 Biểu thức.....	18
1.6.2 Phép toán (toán tử)	19
1.6.2.1 Toán tử gán	19
1.6.2.2. Toán tử số học.....	19
1.6.2.3.Các toán tử gán mở rộng.....	20
1.6.2.4. Phép toán tăng giảm.....	20
1.6.2.5.Các toán tử quan hệ	21
1.6.2.6. Các toán tử logic	21
1.6.2.7. Toán tử điều kiện	22
1.6.2.8. Các toán tử thao tác bit	22
1.6.2.9. Toán tử chuyển đổi kiểu	24
CHƯƠNG 2: CÁC CẤU TRÚC ĐIỀU KHIỂN CHƯƠNG TRÌNH.....	26
2.1 Cấu trúc rẽ nhánh if...else.....	26
2.2 Cấu trúc rẽ nhánh switch.....	28
2.3 Cấu trúc lặp while... ..	30
2.4 Cấu trúc lặp do...while	32
2.4 Cấu trúc lặp for.....	33

2.5 Lệnh break và continue trong thân vòng lặp	34
CHƯƠNG 3: HÀM.....	37
3.1 Hàm chuẩn và hàm tự tạo.....	37
3.2 Xây dựng hàm tự tạo	38
3.3 Sử dụng hàm (gọi hàm).....	39
3.4 Tham số và nguyên tắc truyền tham số	39
3.5 Hàm trùng tên (chồng hàm)	42
3.6 Hàm đệ quy	42
3.7 Bài tập	44
CHƯƠNG 4 MẢNG VÀ XÂU KÝ TỰ	45
4.1 Mảng.....	45
4.1.1 Mảng một chiều.....	45
4.1.1.1 Khai báo mảng một chiều	45
4.1.1.2 Truy xuất vào từng phần tử.....	45
4.1.1.3 Nhập/ xuất dữ liệu.....	45
4.1.2 Mảng hai chiều	47
4.1.2.1 Khai báo mảng hai chiều	47
4.1.2.2 Nhập/ xuất dữ liệu.....	47
4.1.3 Mảng là tham số của hàm.....	49
4.2 Chuỗi ký tự.....	49
4.2.1 Khai báo.....	50
4.2.2 Hằng dãy ký tự	50
4.2.3 Nhập chuỗi.....	51
4.2.4 Các hàm làm việc với chuỗi	53
4.2.5 Mảng dãy ký tự.....	55
4.3 Bài tập	56
CHƯƠNG 5 CON TRỎ TRONG C++.....	57
5.1 Khái niệm.....	57
5.2 Khai báo con trỏ	57
5.3 Các thao tác trên con trỏ.....	57
5.3.1 Thao tác lấy địa chỉ.....	57
5.3.2 Thao tác lấy nội dung	57
5.4 Con trỏ và biến động	58
5.5 Con trỏ và mảng một chiều	59
5.6 Con trỏ và mảng hai chiều	60
CHƯƠNG 6: DỮ LIỆU KIỂU CẤU TRÚC	62
6.1 Khái niệm dữ liệu kiểu cấu trúc	62

6.2 Khai báo dữ liệu kiểu cấu trúc	62
6.3 Truy nhập vào các trường của cấu trúc	63
6.4 Mảng cấu trúc	65
6.5 Con trỏ và địa chỉ cấu trúc	66
6.6 Cấu trúc tự trỏ và danh sách liên kết	66
6.6.1 Cấu trúc tự trỏ	66
6.6.2 Danh sách liên kết	67
6.7 Bài tập	69
CHƯƠNG 7 DỮ LIỆU KIỂU TẬP TRONG C++	70
7.1 Khái niệm về tập	70
7.2 Phân loại tập	70
7.2.1 Phân loại tập theo bản chất dữ liệu của tập	70
7.2.1 Phân loại tập theo cách truy nhập tập	71
7.3 Các bước xử lý tập	71
7.3.1 Khai báo luồng nhập/xuất	71
7.3.2 Mở tệp	71
7.3.3 Xử lý tệp	72
7.3.4 Đóng tệp	73
7.4 Làm việc với tệp văn bản	73
7.4.1 Xuất dữ liệu ra tệp văn bản (ghi tệp)	73
7.4.2 Nhập dữ liệu từ tệp văn bản (đọc tệp)	74
7.5 Làm việc với tệp nhị phân	76
7.5.1 Một số hàm làm việc trên tệp nhị phân	76
7.5.2 Đọc/ ghi các số nguyên vào tệp	76
CHƯƠNG 8 LỚP VÀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG	78
8.1 Giới thiệu về lập trình hướng đối tượng	78
8.1.1 Lập trình hướng đối tượng	78
8.1.2 Đặc điểm của lập trình hướng đối tượng	78
8.1.2.1 Tính bền vững	78
8.1.2.2 Tính đóng gói (encapsulation)	78
8.1.2.3 Tính kế thừa (Inheritance)	79
8.1.2.4 Tính đa kế thừa (Inheritance)	80
8.1.2.4 Tính đa hình	80
8.2 Lớp và đối tượng	81
8.2.1 Khái niệm lớp	81
8.2.2 Khai báo lớp	81
8.2.3 Khai báo biến lớp	82

8.2.4 Khai báo thuộc tính	82
8.2.5 Khai báo và xây dựng các phương thức	83
8.2.6 Sử dụng phương thức	85
8.3 Hàm bạn, hàm tạo, hàm hủy.....	86
8.3.1 Khái niệm phạm vi	86
8.3.2 Phạm vi truy nhập lớp	86
8.3.3 Hàm tạo	87
8.3.4 Hàm hủy	90
8.3.5 Hàm bạn.....	91
8.3.5.1 Hàm tự do bạn của một lớp	91
8.3.5.2 Phương thức của lớp là bạn của một lớp khác.....	92
8.3.5.3 Lớp bạn	93
8.4 Kế thừa	94
8.4.1 Khai báo kế thừa.....	95
8.4.2 Hàm tạo trong kế thừa	96
8.4.3 Hàm hủy trong kế thừa	98
8.5 Chồng toán tử trên lớp	99
8.6 Bài tập	100

CHƯƠNG 1 CƠ BẢN VỀ C/C++

1.1 Giới thiệu C và C++

Ngay từ khi mới ra đời vào đầu những năm 1980, ngôn ngữ C đã nhanh chóng được phổ biến rộng rãi. Các lập trình viên sử dụng ngôn ngữ C để viết chương trình cho nhiều loại máy tính khác nhau từ máy chủ cho đến máy tính cá nhân. Sau đó vào giữa những năm 1980 một phiên bản mới của C ra đời đó là C++ chứa đựng tất cả những đặc điểm của ngôn ngữ C, và được bổ sung nhiều đặc điểm mới rất mạnh. Hai trong số các đặc điểm mới đó là lớp và đối tượng. Nhờ hai đặc điểm mới này mà C++ có thể hỗ trợ lập trình hướng đối tượng.

Tập bài giảng này sẽ trình bày cách viết một chương trình bằng ngôn ngữ C/C++. Trong các chương từ chương 1 đến chương 7 là phần mà cả C và C++ đều hỗ trợ. Một số cách viết khác nhau giữa C và C++ sẽ được đề cập đến. Chương 8 là kỹ thuật lập trình chỉ có trong lập trình C++.

1.2 Cấu trúc một chương trình C/C++

1.2.1 Cấu trúc của một chương trình C/C++

Một chương trình viết trong ngôn ngữ C/C++ thường gồm các phần sau đây:

Các chỉ thị tiền xử lý
Khai báo hàm tự định nghĩa
Hàm main()

Trong đó phần các chỉ thị tiền xử lý và hàm main() bắt buộc phải có, phần khai báo hàm tự định nghĩa có thể có có thể không và nó có thể được viết sau hàm main() vì vậy chương trình C/C++ có thể có dạng:

Các chỉ thị tiền xử lý
Hàm main()
Khai báo hàm tự định nghĩa

+ Các chỉ thị tiền xử lý thường là các lệnh khai báo gọi tệp tin thư viện có dạng `#include <tên_tệp>`, hoặc các lệnh định nghĩa dạng `#define`.

+ Hàm main() là hàm chính trong chương trình C/C++, hàm này bắt buộc phải có. Hàm main() chứa toàn bộ các lệnh của chương trình. Hàm main có dạng như sau:

```
kieuhammain main()
```

```
{
```

```
    Lệnh;
```

```
    return(0); //menu ham main có kiểu nguyên
}
```

Trong đó *kiểu hammain* là kiểu trả về của hàm `main()` nó có thể là một trong số các kiểu dữ liệu (ví dụ kiểu nguyên `int`), hay không kiểu `void`. Các lệnh trong thân hàm được đặt trong cặp dấu ngoặc `{ }` trong đó có cả các câu lệnh khai báo dữ liệu và các câu lệnh tính toán, hay câu lệnh gọi hàm,...

+ Phần khai báo các hàm tự định nghĩa dùng để xây dựng các hàm do người dùng tự định nghĩa, các hàm này không phải là các hàm có sẵn trong hệ thống thư viện của ngôn ngữ lập trình. Phần này có thể không có khi chương trình không cần xây dựng hàm của người sử dụng.

1.2.2 Một chương trình C/C++ đơn giản

Một trong những cách dễ dàng nhất để tiếp cận với một ngôn ngữ lập trình là tìm hiểu một chương trình cụ thể. Ví dụ sau đây là đoạn mã nguồn của một chương trình C++

Dòng	Mã nguồn	Kết quả chạy chương trình
1. 2. 3. 4. 5. 6. 7.	<pre>//day lachuong trinh dau tien #include <iostream.h> void main () { cout <<"Hello World\n"; }</pre>	Hello World

Giải thích:

Dòng 1: Đây là một chú thích. Tất cả các dòng bắt đầu bằng dấu `//` đều được coi là dòng chú thích, nó chỉ có tác dụng đối với người đọc và theo dõi chương trình mà không có một ảnh hưởng nào đến hoạt động của chương trình. Ngoài ra có thể viết một khối chú thích bằng cặp dấu `/* Đây là khối chú thích*/`.

Dòng 2: Sử dụng chỉ thị tiên xử lý `#include` để khai báo việc sử dụng các tệp tin thư viện trong chương trình. Tệp tin `iostream.h` là tệp tin thư viện chứa các lệnh vào ra cơ bản của C++. Ngoài ra còn nhiều tệp tin thư viện khác, nếu muốn sử dụng các lệnh, **hay** các hàm có trong tệp nào thì ta phải include tệp đó.

Dòng 3: Định nghĩa một hàm được gọi là `main`. Hàm có thể không có hay có nhiều **tham số** (parameters); các tham số này luôn xuất hiện sau tên hàm, giữa một cặp dấu ngoặc. Việc xuất hiện của từ `void` ở giữa dấu ngoặc chỉ định rằng hàm `main` trong ví dụ này không có tham số. Hàm có thể có **kiểu trả về**; kiểu trả về luôn xuất hiện trước tên hàm. Kiểu trả về cho hàm `main` ở đây là kiểu nguyên `int`. Tất cả các chương trình C++ phải có một hàm `main` duy nhất. Việc thực thi chương trình luôn bắt đầu từ hàm `main`.

Dòng 4: Dấu ngoặc nhọn bắt đầu thân của hàm `main`.

Dòng 5: Là một **câu lệnh** (statement). Một lệnh là một sự tính toán để cho ra một giá trị. Câu lệnh luôn kết thúc bằng dấu chấm phẩy (;). Câu lệnh này xuất ra **chuỗi** "Hello World\n" để gửi đến dòng xuất cout. Chuỗi là một dãy các ký tự được đặt trong cặp nháy kép. Ký tự cuối cùng trong chuỗi này (\n) là một ký tự xuống hàng (newline). Dòng là một đối tượng được dùng để thực hiện các xuất hoặc nhập. cout là dòng xuất chuẩn trong C++ (xuất chuẩn thường được hiểu là màn hình máy tính). Ký tự << là toán tử xuất, nó xem dòng xuất như là toán hạng trái và xem biểu thức như là toán hạng phải, và tạo nên giá trị của biểu thức được gửi đến dòng xuất. Trong trường hợp này, kết quả là chuỗi "Hello World\n" được gửi đến dòng cout, làm cho nó được hiển thị trên màn hình máy tính.

Dòng 6: Lệnh **Return** kết thúc hàm main và trả về mã đi sau nó. Trong trường hợp này trả về 0, đây là một kết thúc trong trường hợp chương trình không có lỗi.

Dòng 7: Dấu ngoặc đóng kết thúc thân hàm main.

1.3 Bộ ký tự của ngôn ngữ C/C++

Mỗi ngôn ngữ kể cả ngôn ngữ tự nhiên lẫn ngôn ngữ lập trình đều được cấu thành từ một bảng chữ cái (trong các ngôn ngữ lập trình còn gọi là bộ ký tự), bộ ký tự của ngôn ngữ lập trình C/C++ gồm có:

+ Các chữ cái từ a đến z và từ A đến Z (các chữ cái in thường và in hoa trong bảng chữ cái tiếng Anh). Các chữ số: 0 đến 9

+ Các ký hiệu đặc biệt

Ký hiệu	Ý nghĩa	Ký hiệu	Ý nghĩa
—	Dấu cách	-	Dấu trừ
!	Dấu chấm than	/	Dấu gạch chéo
“	Dấu nháy kép	:	Dấu hai chấm
#	Dấu thăng	;	Dấu chấm phẩy
\$	Dấu đô la	<	Dấu nhỏ hơn
%	Dấu phần trăm	>	Dấu lớn hơn
&	Dấu và	=	Dấu bằng
‘	Dấu nháy đơn	?	Dấu chấm hỏi
(Dấu ngoặc mở	@	Dấu a còng
)	Dấu ngoặc đóng	[Dấu ngoặc vuông
*	Dấu sao]	Dấu ngoặc vuông
+	Dấu cộng	^	Dấu mũ
,	Dấu phẩy	_	Dấu gạch nối
{	Dấu ngoặc nhọn	}	Dấu ngoặc nhọn
	Dấu gạch dọc	.	Dấu chấm

1.4 Tên, kiểu dữ liệu, hằng, biến

1.4.1 Tên

Tên hay còn gọi là định danh (identifier) là một khái niệm rất quan trọng nó được dùng để xác định các đại lượng khác nhau trong chương trình. Ta có tên hằng, tên biến, tên chương trình con, tên kiểu dữ liệu,...

Tên có thể chia làm 3 loại: Tên chuẩn, từ khóa và tên do người sử dụng tự đặt. Tên chuẩn là tên của các đối tượng đã có sẵn của ngôn ngữ như tên các kiểu dữ liệu, tên các tệp tin thư viện,... Từ khóa là các từ được ngôn ngữ sử dụng vào những mục đích riêng, người sử dụng không thể đặt tên cho các đại lượng của mình trùng với từ khóa.

Tên được đặt theo quy tắc:

Tên là một dãy ký tự bao gồm chữ cái, chữ số, và dấu gạch nối (_). Ký tự đầu tiên phải là một chữ cái hoặc dấu gạch nối. Độ dài cực đại của tên theo mặc định là 32 ký tự. Tên không được trùng với từ khóa hay các toán tử.

Các từ khóa trong C/C++:

asm, car, bool, break, marry, catch, to char, class, const, const_cast, continue, default, delete, do, double, dynamic_cast, else, enum, explicit, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, operator, private, protected, public, to register, reinterpret_cast, return, short, signed, sizeof, static, static_cast, struct, switch, template, this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t

Các toán tử trong C/C++:

and, and_eq, bitand, bitor, compl, not, not_eq, or, or_eq, xor, xor_eq,...

Chú ý: Ngôn ngữ C/C++ là một ngôn ngữ có phân biệt chữ hoa chữ thường (Case Sensitive) do vậy ví dụ biến tong khác với TONG và cũng khác với Tong. Tên nên đặt ngắn gọn, gợi nhớ, điều đó sẽ làm cho việc theo dõi và kiểm soát chương trình dễ dàng hơn.

1.4.2 Kiểu dữ liệu

1.4.2.1 Kiểu số nguyên

Trong C/C++ có nhiều kiểu số nguyên với miền giá trị khác nhau. Kiểu số nguyên có bản nhất trong C/C++ là kiểu int, mỗi biến kiểu int chiếm 2 byte trong bộ nhớ. Miền giá trị của kiểu nguyên là từ -32768 đến +32767. Ngoài ra còn một số kiểu số nguyên khác như bảng dưới đây:

Bảng 1.1: Các kiểu số nguyên

Tên kiểu	Từ khóa	Kích thước	Miền giá trị
Character	signed char	1 byte	$-128 \div 127$
	unsigned char		$0 \div 255$
Integer	Int	2 byte	$-32768 \div 32767$
	unsigned int		$0 \div 65535$
Short integer	Short	2 byte	$-32768 \div 32767$
	unsigned short		$0 \div 65535$
Long integer	Long	4 byte	$-2147483648 \div 2147483647$
	unsigned long		$0 \div 4294967295$

Ghi chú: Kiểu *char* thực chất là kiểu ký tự, nhưng C/C++ cho phép sử dụng kiểu ký tự này như một kiểu số nguyên. Khi tham gia các biểu thức số học thì kiểu *char* được hiểu là kiểu số nguyên. Nhưng khi tham gia các phép toán trong biểu thức ký tự thì kiểu *char* lại được hiểu là kiểu ký tự. Thực chất một ký tự bất kỳ đều có một giá trị mã ASCII được cho trong bảng mã ASCII.

Các ký tự có mã từ 0 đến 31 là các ký tự điều khiển. Ví dụ ký tự có mã 13 là ký tự chuyển con trỏ về đầu dòng, ký tự có mã 10 chuyển con trỏ xuống dòng. Các ký tự điều khiển này không hiển thị ra màn hình.

Các ký tự văn bản có mã từ 32 đến 126. Các ký tự này có thể đưa ra màn hình hoặc máy in.

Các ký tự có mã từ 127 đến 255 là các ký tự đồ họa.

Các phép toán có thể thực hiện được trên kiểu số nguyên:

Bảng 1.2: Các phép toán số học trên kiểu số nguyên

Tên phép toán	Ký hiệu	Ví dụ	Ghi chú
Cộng	+	$a + b$	
Trừ	-	$a - b$	
Nhân	*	$a * b$	
Chia nguyên	/	a / b	Kết quả cho ra một số nguyên (VD: $7/2 = 3$)
Chia lấy phần dư	%	$a \% b$	$7 \% 2 = 1$

Chú ý: Khi thực hiện phép chia hai số nguyên cho nhau thì kết quả có thể ra một số không nguyên. Nếu kết quả này được chứa trong một biến kiểu nguyên thì phần lẻ trong kết quả bị cắt đi và lúc này phép chia trở thành phép chia lấy phần nguyên. Ngược lại nếu kết quả được chứa trong một biến kiểu số thực thì phần lẻ vẫn được giữ nguyên.

Cách viết số nguyên trong các hệ đếm khác:

Cách viết các số nguyên dưới dạng số hệ 16, và hệ 8:

Hệ 16: Viết 0x trước giá trị số

Ví dụ: Số 72 ở hệ 10 được viết là 0x48 (số 48 trong hệ 16)

Số 65 ở hệ 10 được viết là 0x41 (số 41 trong hệ 16)

Hệ 8: Viết số 0 đằng trước số ở hệ 8

Ví dụ: Số 65 ở hệ 10 được viết là 0101 (số 101 ở hệ 8)

Số 72 ở hệ 10 được viết là 0110 (số 110 ở hệ 8)

1.4.2.2. Kiểu số thực (kiểu dấu phẩy động)

Các kiểu số thực được ngôn ngữ C/C++ cung cấp đó là: kiểu float, double, và long double.

Bảng 1.3: Các kiểu số thực

Tên kiểu	Từ khóa	Kích thước	Miền giá trị
Kiểu số thực có độ chính xác đơn	Float	4 byte	$\pm 3.4 \cdot 10^{-38} \div \pm 3.4 \cdot 10^{+38}$ (Khoảng 7 chữ số có nghĩa)
Kiểu số thực có độ chính xác kép	Double	8 byte	$1.7 \cdot 10^{-308} \div 1.7 \cdot 10^{+308}$ (Khoảng 15 chữ số có nghĩa)
Kiểu số thực lớn có độ chính xác kép	Long double	10 byte	$3.4 \cdot 10^{-4932} \div 1.1 \cdot 10^{+4932}$ (Khoảng 17 chữ số có nghĩa)

Cách viết số thực:

Cách 1: Viết dưới dạng hệ 10, sử dụng dấu chấm (.) để ngăn cách phần nguyên và phần thập phân. Ví dụ:

5.14 3.12 123.1678 145.23

Cách 2: Mỗi số thực được viết thành 2 phần: phần định trị và phần bậc. Ví dụ:

Số 5.14 = $514 \cdot 10^{-2}$ và ta có thể viết 514E-2 (Phần định trị là 514 và phần bậc là E-2). Hoặc ta cũng có thể viết 51.4E-1 ($51.4 \cdot 10^{-1}$) hay 0.514E+1 ($0.514 \cdot 10^1$)

1.4.2.3. Kiểu ký tự

Ký tự là một chữ cái, chữ số, hay một ký tự đặc biệt. Trong ngôn ngữ C/C++ kiểu ký tự được khai báo bằng từ khóa char. Mỗi dữ liệu kiểu ký tự được máy tính giành 1 byte (=8 bit) bộ nhớ để lưu trữ. Do vậy số ký tự mà máy tính có thể nhận biết là $2^8 = 256$ ký tự, tuy nhiên chỉ có 128 ký tự đầu tiên (các ký tự có mã từ 0 đến 127) là hay được sử dụng và chúng thường được sử dụng để tạo ra bộ ký tự cho các ngôn ngữ lập trình.

Ngoài các kiểu dữ liệu cơ sở như đã nêu, trong C/C++ còn có một số kiểu dữ liệu sinh ra từ các kiểu cơ sở. Ví dụ kiểu liệt kê (enumeration), kiểu con trỏ (pointer), kiểu mảng (array), kiểu cấu trúc (struct). Riêng trong ngôn ngữ C++ còn có kiểu lớp (class) mà trong C không có

1.4.3 Hằng, biến

1.4.3.1 Hằng

- Khái niệm: Là đại lượng có giá trị xác định và không thay đổi trong toàn bộ chương trình. Hằng cũng có các kiểu như những kiểu dữ liệu đã nêu trên: hằng số nguyên, hằng số thực, hằng bool, hằng ký tự,...

Khi viết hằng ký tự thì ta thêm cặp dấu nháy bao quanh. Ví dụ: hằng ký tự 'a' hay hằng chuỗi ký tự "hello word". Biểu diễn ký tự đơn thì ta đặt nó trong cặp dấu nháy đơn, khi biểu diễn chuỗi ký tự ta đặt chuỗi đó trong cặp dấu nháy kép.

Có một số hằng ký tự đặc biệt phải được viết liền với dấu \ :

Bảng 1.4: Các kiểu số nguyên

Cách viết	Kí tự
"\"	'
"\""	"
"\\'	\
"\n'	xuống dòng
"\0'	Null
"\t'	Tab
"\b'	Backspace
"\r'	CR(về đầu dòng)
"\f'	LF(sang trang)

Chuỗi ký tự có thể được viết trên nhiều dòng mỗi dòng kết thúc bằng dấu \. Ví dụ:

“xau ky tu tren \
hai dong”

Hằng logic: Có thể nhận một trong hai giá trị là true và false.

- Định nghĩa hằng:

Định nghĩa hằng trong chương trình sử dụng chỉ thị tiền xử lý #define với cú pháp như sau:

#define tenhang giatri

Ví dụ: *#define pi 3.1416 // định nghĩa hằng pi bằng 3.1416*

Trong chương trình có thể sử dụng hằng pi tham gia vào biểu thức tính toán:

*chuvi=2*pi*r;*

- Khai báo hằng: Sử dụng từ khóa const ta có thể khai báo hằng với một kiểu xác định tương tự như việc khai báo biến, ví dụ:

const int songuyen=100;

const char kytu='a';

1.4.3.2. Biến

- **Khái niệm**: Biến là tên một vùng nhớ lưu trữ dữ liệu. Giá trị của biến chính là giá trị đã được lưu trữ trên vùng nhớ mang tên biến đó. Giá trị này có thể được thay thế bằng một giá trị khác trong quá trình thực hiện chương trình.

- **Khai báo**: Trước khi sử dụng biến ta phải khai báo biến theo qui ước như sau:

Ten_kieuDL tenbien1, tenbien2,...,tenbienk;

Trong đó:

Ten_kieuDL là tên kiểu dữ liệu.

tenbien1, tenbien2,...,tenbienk là danh sách các biến có cùng kiểu dữ liệu cần khai báo.

Ví dụ : *int a, b; float c;*

Dòng đầu tiên khai báo hai biến *a, b* có cùng kiểu số nguyên *int*.

Dòng thứ hai khai báo một biến *c* có kiểu dấu phẩy động *float*.

Sau khi khai báo một biến máy tính giành ra số ô nhớ bằng với dung lượng của kiểu dữ liệu của biến đó, và đặt tên ô nhớ đó là tên biến đã khai báo. Trong ví dụ trên sau khi gặp câu lệnh *float c;* máy tính giành ra vùng nhớ 4 byte (mỗi biến kiểu *float* có kích thước là 4 byte) cho biến *c*.

Ghi chú: Với các kiểu số nguyên *char, short, long, int* có thể là số có dấu hay không dấu tùy thuộc vào miền giá trị mà chúng ta biểu diễn. Vì vậy khi khai báo một kiểu số nguyên ta sử dụng từ khóa *signed*(có dấu) hay *unsigned*(không dấu) trước tên kiểu.

Ví dụ: *signed char mynumber;*

unsigned int a;

Trong trường hợp không chỉ rõ là *signed* hay *unsigned* thì mặc định ngôn ngữ hiểu là kiểu có dấu *signed*. Cho nên trong hầu hết các trường hợp người ta không cần sử dụng từ khóa *signed*, duy chỉ có trường hợp kiểu dữ liệu *char* là ngoại lệ vì kiểu *char* có thể được hiểu là kiểu ký tự khác hẳn với kiểu *signed char* và *unsigned char*.

- **Khởi tạo biến**: Khi khai báo một biến thì giá trị của biến đó chưa được xác định, cho nên nếu ta muốn xác định một giá trị cho biến ngay từ lúc khai báo thì ta sử dụng dấu bằng và giá trị khởi tạo trong câu lệnh khai báo biến như sau:

Ten_kieuDL tenbien=giatrikhoitao;

Hoặc:

Tên_kieuDL tenbien(giatrikhoitao);

Ví dụ 1.1:

```
// Khởi tạo giá trị cho biến
#include <iostream.h>
void main ()
{
    int a=5;    // Khởi tạo giá trị cho a = 5
    int b(2);  // Khởi tạo giá trị cho b = 2
    int ketqua;    a = a + 3; ketqua = a - b;
    cout << ketqua;
    //return 0;
}
```

6

- Vị trí khai báo: Tất cả các biến sử dụng trong chương trình đều phải được khai báo trước khi sử dụng. Có một điểm khác biệt giữa C và C++ đó là trong C++ ta có thể khai báo biến ở bất kỳ vị trí nào trong chương trình miễn là trước khi biến đó được sử dụng. Trong C thì vị trí khai báo biến bắt buộc phải ở đầu của thân hàm (sau dấu ngoặc mở { của thân hàm) : Ví dụ cách khai báo biến trong C:

```
void main()
{
    int a,b,c;
    a=2;
    int d; /* Vị trí khai báo sai trong C*/
}
```

Tuy trong C++ cho phép khai báo linh hoạt hơn nhưng chúng ta vẫn nên tuân theo cách của C là khai báo toàn bộ các biến trong phần bắt đầu các hàm (biến cục bộ) hay trực tiếp trong thân chương trình ngoài các hàm con (biến toàn cục). Điều này sẽ giúp ta dễ kiểm soát các biến của chương trình.

- Biến cục bộ, biến toàn cục và phạm vi hoạt động:

+ Biến cục bộ (**local variables**): Là những biến được khai báo bên trong thân của mỗi hàm (kể cả hàm main()). Biến cục bộ chỉ có phạm vi ảnh hưởng trong hàm mà nó được khai báo. Do đó ta có thể khai báo hai biến trùng tên, trùng kiểu dữ liệu ở trong hai hàm khác nhau mà không gây xung đột hay hiểu lầm.

+ Biến toàn cục (**global variables**): Biến toàn cục là các biến được khai báo bên ngoài tất cả các hàm (sau các chỉ thị tiền xử lý). Biến toàn cục có thể được sử dụng ở trong hàm main hoặc bất kỳ hàm nào khác của chương trình. Do vậy không được đặt tên của một biến cục bộ trùng với tên của biến toàn cục.

Chúng ta có thể thấy rõ hơn về vị trí khai báo cục bộ và toàn cục qua ví dụ sau đây:

Ví dụ 1.2:

```
#include <iostream.h>
int tong;
void tinhTong(int a, int b)
{
    tong=a+b;
}
void main()
{
    int a,b;
    cout<<"Nhập vào hai số cần tính tổng";
    cin>>a; cin>>b;
    tinhTong(a,b);
    cout<<"Giá trị của biến tong="<<tong;
}
```

Biến toàn cục (Global variables)

Sử dụng biến toàn cục trong thân hàm

Biến cục bộ (Local Variables)

Sử dụng biến toàn cục trong thân hàm

1.5 Các lệnh nhập/ xuất dữ liệu trong C/C++

Khi nói đến nhập xuất dữ liệu mà không đề cập đến thiết bị vào/ra tức là ta nói đến nhập/xuất dữ liệu từ thiết bị vào ra chuẩn đó chính là bàn phím và màn hình. Nhập dữ liệu từ bàn phím và xuất dữ liệu ra màn hình.

1.5.1 Lệnh nhập/xuất dữ liệu trong C

- Nhập dữ liệu:

Cú pháp:

```
scanf ("chuỗi định dạng", danh sách biến);
```

Trong đó:

+ “*chuỗi định dạng*” là chuỗi chứa các mã định dạng, mỗi mã định dạng bắt đầu bằng ký tự %, “*chuỗi định dạng*” phải được đặt trong dấu nháy kép “”. Có bao nhiêu biến trong *danh sách biến* thì phải có bấy nhiêu mã định dạng trong *chuỗi định dạng*.

+ *danh sách biến*: Mỗi biến trong danh sách phải có ký tự (&) đặt trước tên biến để chỉ định địa chỉ biến. Các biến ngăn cách nhau bởi dấu phẩy (,)

Chú ý: Thứ tự của các mã định dạng phải phù hợp với thứ tự của biến.

Bảng 1.5: Bảng mã định dạng

Mã định dạng	Ý nghĩa
%d	Mã định dạng cho dữ liệu kiểu số nguyên
%f	Mã định dạng cho dữ liệu kiểu số thực
%o	Mã định dạng cho dữ liệu số nguyên ở hệ 8
%x	Mã định dạng cho dữ liệu số nguyên ở hệ 16
%c	Mã định dạng cho dữ liệu kiểu ký tự
%s	Mã định dạng cho dữ liệu kiểu chuỗi ký tự
%e, %E, %g, %G	Mã định dạng cho dữ liệu số thực dạng mũ
l,L	Thêm vào trước các kiểu để chuyển sang kiểu long hay double.

	Ví dụ: l đi với d,o,x để ra kiểu long L đi với e, E, g, G, f để ra kiểu double
--	---

Ví dụ: Nhập dữ liệu cho hai biến nguyên x và y

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

Nhập dữ liệu cho hai số thực a và b

```
scanf("%f%f", &a, &b);
```

Nhập dữ liệu cho biến kiểu nguyên x, y, biến kiểu thực a, b như sau:

```
scanf("%d%d%f%f", &x, &y, &a, &b);
```

- Xuất dữ liệu:

Cú pháp:

```
printf ("chuỗi định dạng", danh sách biến);
```

Trong đó:

+ "*chuỗi định dạng*" tuân theo các quy định như trong câu lệnh scanf. Ngoài ra chuỗi định dạng có thể được sử dụng để quy định thêm định dạng khi xuất dữ liệu:

Ví dụ: Sử dụng mã định dạng "%3d" để in ra giá trị của biến kiểu nguyên có tối đa 3 chữ số. "%5f" để in ra giá trị của biến kiểu số thực có tối đa 5 ký tự. "%.2f" in số thực có 2 số lẻ trong phần thập phân.

+ *danh sách biến*: mỗi biến ngăn cách nhau bởi dấu phẩy (.). Trước tên biến không đặt dấu &.

Lệnh scanf và printf nằm trong thư viện stdio.h. Nên trong chương trình sử dụng hai lệnh này thì ta phải có chỉ thị #include<stdio.h>

Câu hỏi:

1. Cho biết kết quả thực hiện đoạn chương trình sau, biết dòng dữ liệu vào là 15e10

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

```
main()
```

```
{
```

```
int a; char c; float x;
```

```
scanf("%d%c%f", &a, &c, &x);
```

```
printf("a=%d,c=%d,x=%f",a,c,x);
```

```
}
```

2. Không dùng đúng mã định dạng chương trình sẽ in sai:

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

```
main()
```

```
{
```

```
int x=10;
```

```
long y=3478925;
```

```
printf("\n x=%ld \n y=%d",x,y);
```

```
}
```

Kết quả sai: $x=361562122$

$y=53$

Sửa lại chương trình cho đúng

1.5.2 Lệnh nhập/ xuất dữ liệu trong C++

Trong thư viện *iostream.h* của C++, các thao tác vào ra cơ bản của chương trình được hỗ trợ bởi hai dòng điều khiển: *cin* để nhập dữ liệu và *cout* để xuất dữ liệu. Thông thường *cin* được gắn với thiết bị nhập chuẩn đó là bàn phím còn *cout* được gắn với thiết bị xuất chuẩn là màn hình. Có nghĩa là mặc định nhập dữ liệu từ bàn phím và xuất dữ liệu ra màn hình.

Một điều đặc biệt là với C++ ta không cần quan tâm tới các đặc tả của biến, vì khi dịch chương trình dịch đã tự động nhận ra đó là biến loại gì khi gặp câu lệnh khai báo biến.

- Nhập dữ liệu:

Cú pháp: *cin>> tenbien;*

Khi thực hiện câu lệnh trên chương trình cho phép người sử dụng gõ một giá trị từ bàn phím sau đó nhấn phím enter. Giá trị nhập vào này sẽ được gán cho biến *tenbien*.

Ví dụ:

```
int tuoi; //Khai bao bien tuoi kiểu nguyên
```

```
cin>> tuoi; //nhap dl cho bien tuoi
```

Ghi chú: Giá trị nhập vào cần phải nằm trong miền giá trị của kiểu biến tương ứng. Ví dụ nếu ta khai báo biến *tuoi* kiểu số nguyên (int) thì giá trị nhập vào cho biến *tuoi* phải nằm trong miền giá trị của kiểu nguyên int.

Ta có thể sử dụng một câu lệnh nhập để nhập đồng thời giá trị cho nhiều biến như sau: *cin>> bien1>> bien2>> ...;*

Nhập chuỗi: Sử dụng dòng nhập *cin* ta cũng có thể nhập vào một chuỗi ký tự: dòng lệnh *cin >> str;* cho phép nhập vào một chuỗi và gán chuỗi đó cho biến *str*. Tuy nhiên chuỗi nhập vào mà chứa khoảng trống thì dòng vào *cin* sẽ ngắt bỏ đoạn từ sau khoảng trống đó. Để khắc phục điều này thì ta sử dụng hàm **getline** theo cú pháp:

```
getline(cin, str); //str la bien kieu string
```

Ví dụ 1.3:

```
// Nhap string voi ham getline
#include <iostream.h>
int main ()
{
    char  mystr[30];
```

```
What's your name? Nguyen
Quynh
Hello Nguyen Quynh
What is your favorite team?
Manchester United
```


<pre>cout << "What's your name? "; getline (cin, mystr); cout << "Hello " << mystr << ".\n"; cout <<"What is your favorite team?"; getline (cin, mystr); cout << "I like " << mystr <<"too!\n"; return 0; }</pre>	I like Manchester United too!
---	--------------------------------------

- **Xuất dữ liệu:** Sử dụng dòng điều khiển *cout* và toán tử <<

Cú pháp: *cout << "chuỗi ký tự";* //hiển thị ra màn hình chuỗi ký tự trong dấu “ ”

hoặc *cout<<biến;* // hiển thị ra màn hình giá trị của *biến*

hoặc *cout<<giatri;* //hiển thị ra màn hình giá trị *giatri*

Ví dụ 1.4:

Mã nguồn	Kết quả
<pre># include <iostream.h> void main() { int x=5; //in mot chuỗi ra màn hình cout <<" gia tri cua bien x la"; //in gia tri cua bien ra màn hình cout<<x; }</pre>	gia tri cua bien x la 5

Chú ý rằng sau khi thực hiện xong lệnh *cout* không tự động xuống dòng, vì vậy mặc dù sử dụng hai dòng lệnh *cout*:

cout << " gia tri cua bien x la";

cout<<x;

Nhưng kết quả chỉ thể hiện trên một dòng. Như vậy muốn xuống dòng thì ta sử dụng ký tự \n. Đoạn chương trình trên có thể sửa lại như sau:

Mã nguồn	Kết quả
<pre># include <iostream.h> void main() { int x=5; cout <<" gia tri cua bien x la \n"; cout<<x; }</pre>	gia tri cua bien x la 5

Ngoài ra chúng ta cũng có thể sử dụng toán tử << nhiều lần trong một câu lệnh xuất ví dụ:

cout<< "gia tri cua bien x la " << x;

Ví dụ 1.5: Sử dụng dòng nhập và xuất *cin*, *cout*

Mã nguồn	Kết quả
<pre># include< iostream.h> void main() {int x, tong; cout <<"nhapgia tri cho bien x:"; cin>>x; tong=x+100; cout<<"\n gia tri cua tong la"<<tong; }</pre>	<p>Nhap gia tri cho bien x: 121 gia tri cua tong la 221</p>

Định dạng khi in ra màn hình

+ Để qui định số thực (float,double) được in ra đúng p chữ số sau dấu chấm thập phân, ta sử dụng đồng thời các hàm sau:

```
setiosflags(ios::showpoint); // Bật cờ hiệu showpoint
setprecision(p);
```

Các hàm này cần đặt trong toán tử xuất như sau:

cout<<setiosflags(ios::showpoint)<<setprecision(p) ;

Câu lệnh trên sẽ có hiệu lực đối với tất cả các toán tử xuất tiếp theo cho đến khi gặp một câu lệnh định dạng mới.

+ Để qui định độ rộng tối thiểu là v vị trí cho giá trị (nguyên,thực,chuỗi) được in trong các toán tử xuất,ta dùng hàm

setw()

Hàm này cần đặt trong toán tử xuất và nó chỉ có hiệu lực cho 1 giá trị được in gần nhất. Các giá trị in ra tiếp theo sẽ có độ rộng tối thiểu là 0. Như vậy câu lệnh:

```
cout<<setw(3)<<"AB"<<"CD"
```

Sẽ in ra 5 kí tự là: một dấu cách và 4 chữ cái A,B,C và D.

Muốn sử dụng các hàm trên cần #include <iomanip.h>

1.6 Biểu thức và phép toán

1.6.1 Biểu thức

Biểu thức là sự kết hợp giữa các phép toán (toán tử) và các toán hạng diễn đạt một công thức toán học nào đó, cho phép xác định một giá trị nhất định qua một quá trình tính toán.

Toán hạng là một hằng, biến hoặc một hàm. Toán tử được viết ra bằng dấu phép toán. Trong trường hợp đơn giản nhất biểu thức chỉ gồm một toán hạng và một toán tử.

Căn cứ vào đặc điểm của biểu thức mà người ta chia biểu thức thành các loại khác nhau. Trong C/C++ có biểu thức số học, biểu thức logic, và biểu thức quan hệ.

+ **Biểu thức số học** là biểu thức xác định một giá trị kiểu số nguyên hoặc một giá trị kiểu số thực.

+ **Biểu thức logic** là biểu thức xác định một giá trị logic *true* hoặc *false*. Tuy nhiên trong C++ không dùng chữ *true* hay *false* để biểu diễn giá trị logic mà sử dụng số 1 thay cho *true* và số 0 thay cho *false*.

+ **Biểu thức quan hệ** là các biểu thức chứa các toán tử quan hệ (>, <, >=, <=, =, !=). Các toán hạng trong biểu thức quan hệ có thể là số nguyên, số thực, các ký tự và chúng không nhất thiết phải tương thích nhau về kiểu.

1.6.2 Phép toán (toán tử)

Các phép toán hay còn gọi là các toán tử là các ký hiệu được sử dụng trong chương trình giúp cho máy tính thực hiện một phép tính hay một thao tác xử lý nào đó trên các toán hạng. Ngôn ngữ C/C++ cho phép sử dụng rất nhiều toán tử trên nhiều loại dữ liệu khác nhau. Để dễ nhớ người ta nhóm các toán tử thành các nhóm như sau:

1.6.2.1 Toán tử gán

Toán tử gán được sử dụng để gán một giá trị nào đó cho một biến.

Ví dụ: `x=5; //gán giá trị 5 cho biến x`

`tong=x+100; // gán giá trị x+100 cho biến tong`

Cú pháp: *variable=expression*

Khi thực hiện lệnh gán, biểu thức ở vế phải (*expression*) sẽ được tính giá trị và kết quả được gán cho biến ở vế trái (*variable*).

Chú ý rằng trong lệnh gán sử dụng dấu bằng nhưng không có nghĩa là sự so sánh ($a = b$) mà đây là gán giá trị của *b* cho *a* và sau câu lệnh đó *b* có thay đổi giá trị thì cũng không ảnh hưởng gì đến giá trị của *a*.

Một khả năng của phép gán trong C++ góp phần giúp nó vượt lên các ngôn ngữ lập trình khác đó là vế phải của một phép gán có thể chứa một phép gán khác.

Ví dụ:

`tong = a + (b = 5);`

Câu lệnh trên tương đương với hai câu lệnh sau:

`b = 5; tong = a + b;`

Vì vậy biểu thức sau cũng hợp lệ trong C++

`a = b = c = 5;`

gán giá trị 5 cho cả ba biến *a*, *b* và *c*

1.6.2.2. Toán tử số học

Ngôn ngữ C++ sử dụng các toán tử số học như cộng, trừ, nhân, chia (+, -, *, /). Đây là các toán tử có hai toán hạng (phép toán hai ngôi). Ý nghĩa của các phép toán này cũng giống như các phép toán trong số học. Thứ tự ưu tiên là nhân chia trước, cộng trừ

sau. Ngoài ra trong C++ còn có phép toán lấy phần dư %. Ví dụ viết $a \% b$ có nghĩa là lấy phần dư của phép chia a cho b .

Chúng ta có thể theo dõi các toán tử số học qua bảng sau:

Bảng 1.6 Các toán tử số học

Toán tử	Kí hiệu	Mô tả	Ví dụ
Đảo dấu	$+$, $-$	Phép toán một ngôi, đảo dấu	$-x$
Cộng	$+$	Cộng 2 toán hạng với nhau	$x + y$
Trừ	$-$	Toán hạng thứ nhất trừ toán hạng thứ hai	$x - y$
Nhân	$*$	Nhân 2 toán hạng với nhau	$x * y$
Chia	$/$	Chia toán hạng thứ nhất cho toán hạng thứ hai	x / y
Modulus	$\%$	Lấy phần dư của phép chia giữa 2 số nguyên	$x \% y$

1.6.2.3. Các toán tử gán mở rộng

Một đặc tính của ngôn ngữ C/C++ làm cho nó nổi tiếng là một ngôn ngữ súc tích chính là các toán tử gán mở rộng cho phép thay đổi giá trị của một biến với một trong những toán tử cơ bản sau:

$+=$, $-=$, $*=$, $/=$, $\%=$, $>>=$, $<<=$, $\&=$, $\^=$, $|=$

Bảng 1.7 Các toán tử gán mở rộng

Cách viết	Tác dụng	Ví dụ
$X += y$	$x = x + y$	$x += 5$ tương đương với $x = x + 5$
$X -= y$	$x = x - y$	$x -= 6$ tương đương với $x = x - 6$
$X *= y$	$x = x * y$	$x *= 6$ tương đương với $x = x * 6$
$X /= y$	$x = x / y$	$x /= 4$ tương đương với $x = x / y$
$X \% = y$	$x = x \% y$	Phép lấy phần dư
$X >> = y$	$x = x >> y$	Phép dịch bit dịch trái x đi y bit
$X << = y$	$x = x << y$	Phép dịch phải x đi y bit
$X \& = y$	$x = x \& y$	Phép và
$X = y$	$x = x y$	Phép hoặc
$X \^ = y$	$x = x \^ y$	Phép lũy thừa

1.6.2.4. Phép toán tăng giảm

Một ưu điểm nữa của C/C++ trong việc tiết kiệm khi viết mã lệnh là toán tử tăng ($++$) và giảm ($--$). Chúng tăng hoặc giảm giá trị chứa trong một biến đi 1 tương ứng với $+=1$ hoặc $-=1$. Vì vậy các cách viết sau là tương đương:

$a++$

$a = a + 1$

$a += 1$

Phép toán tăng hoặc giảm có thể được viết theo hai cách:

+ Cách 1: Phép toán được viết trước tên biến, ví dụ ++a. Có nghĩa là giá trị của biến được tăng trước khi biểu thức được tính và giá trị đã tăng được sử dụng trong biểu thức.

Cách 1	Cách 2
B=3; A=++B; // A = 4, B = 4	B=3; A=B++; // A = 3, B = 4

+ Cách 2: Phép toán được viết sau tên biến, ví dụ a++. Giá trị của biến được tăng sau khi tính toán xong giá trị của biểu thức. (Xem 2 ví dụ trên)

1.6.2.5. Các toán tử quan hệ

Bảng 1.8 Các toán tử quan hệ

Toán tử	Kí hiệu	Ví dụ	Ghi chú
Lớn hơn	>	a > b cho giá trị false (sai) b > a cho giá trị true (đúng)	Với a = 5 b = 7
Nhỏ hơn	<	a < b cho giá trị true (đúng) b < a cho giá trị false (sai)	
Lớn hơn hoặc bằng	>=	a >= b b >= a	
Nhỏ hơn hoặc bằng	<=	a <= b b <= a	
Bằng nhau	==	a == b	
Khác nhau	!=	a != b	

Trong ngôn ngữ C kết quả của phép toán quan hệ luôn luôn là một số nguyên bằng 1 nếu đúng, 0 nếu sai. Do đó nó hoàn toàn có thể tham gia vào biểu thức số học tiếp theo. Tuy nhiên trong C++ theo chuẩn ANSI-C++ thì giá trị của thao tác quan hệ chỉ có thể là giá trị **true** hoặc **false**, tùy theo biểu thức kết quả là đúng hay sai.

1.6.2.6. Các toán tử logic

Các toán tử logic (!, &&, ||).

Toán tử ! tương đương với toán tử logic NOT, nó chỉ có một đối số ở phía bên phải và việc duy nhất mà nó làm là đổi ngược giá trị của đối số từ **true** sang **false** hoặc ngược lại. Ví dụ:

!(5 == 5)	trả về false vì biểu thức bên phải (5 == 5) có giá trị true .
!(6 <= 4)	trả về true vì (6 <= 4) có giá trị false .
!true	trả về false .
!false	trả về true .

Toán tử logic **&&** và **||** được sử dụng khi tính toán hai biểu thức để lấy ra một kết quả duy nhất. Chúng tương ứng với các toán tử logic *AND* và *OR*. Kết quả của chúng phụ thuộc vào mối quan hệ của hai đối số:

Đối số thứ nhất a	Đối số thứ hai b	a && b	a b
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Ví dụ:

((5 == 5) && (3 > 6)) trả về **false** (*true && false*).

((5 == 5) || (3 > 6)) trả về **true** (*true || false*).

1.6.2.7. Toán tử điều kiện

Toán tử điều kiện (?)

Toán tử điều kiện tính toán một biểu thức và trả về một giá trị tùy thuộc vào biểu thức đó là đúng hay sai. Cấu trúc của nó như sau:

dieukien ? giatri1 : giatri2

Nếu *dieukien* là **true** thì giá trị trả về sẽ là *giatri1*, nếu không giá trị trả về là *giatri2*.

7==5 ? 4 : 3 trả về **3** vì **7** không bằng **5**.

7==5+2 ? 4 : 3 trả về **4** vì **7** bằng **5+2**.

5>3 ? a : b trả về **a**, vì **5** lớn hơn **3**.

a>b ? a : b trả về giá trị lớn hơn, **a** hoặc **b**.

1.6.2.8. Các toán tử thao tác bit

Trong ngôn ngữ C/ C++ cung cấp các phép toán thao tác trên bit như sau:

Bảng 1.9 Các toán tử thao tác bit

Tên phép toán	Ký hiệu của phép toán	Mô tả
AND	&	Logical AND
OR	 	Logical OR
XOR	^	Logical exclusive OR
NOT	~	Đảo ngược bit
SHL	<<	Dịch bit sang trái
SHR	>>	Dịch bit sang phải

Kết quả thực hiện các phép toán trên hai bit A và B được cho trong bảng sau:

Bit A	Bit B	Kết quả trên bit tương ứng của các phép toán
-------	-------	--

		!A	A & B	A B	A ^ B
0	0	1	0	1	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	0	0

Các phép toán trên bit chỉ thực hiện được với các dữ liệu có kiểu nguyên như char, int, long, kể cả signed và unsigned.

Ví dụ: Có hai số ở hệ 10 A=1234 B = 4321

Chuyển hai số này sang hệ 2:

A = 1234 → 0000 0100 1101 0010
 B = 4321 → 0001 0000 1110 0001
 A & B → 0000 0000 1100 0000
 A | B → 0001 0100 1111 0011
 A ^ B → 0001 0100 0011 0011

Phép toán dịch trái <<:

Phép toán dịch trái cho phép dịch toán hạng sang trái n bit. Khi dịch sang trái n bit thì n bit phải của toán hạng được điền vào các bit 0.

Ví dụ: số a_(hệ 10) = 201

Số a_(hệ 2) =

00000000	11001001
----------	----------

a<<2

00000011	00100100	00
----------	----------	----

 ← 2 bit bên phải bằng 0

Kết quả sau khi dịch

a_(hệ 2) = 0011 0010 0100

a_(hệ 10) = 804

Phép toán dịch phải >>:

+ Đối với các toán hạng có kiểu **unsigned** (int, long, char)

Phép toán dịch phải cho phép dịch toán hạng sang phải n bit. Khi dịch sang phải n bit thì n bit trái của toán hạng được điền vào các bit 0.

Ví dụ: số a_(hệ 10) = 39470

Số a_(hệ 2) =

10011010	00101110
----------	----------

a>>2

00100110	10001011
----------	----------

↑
2 bit bên phải bằng 0

Kết quả sau khi dịch

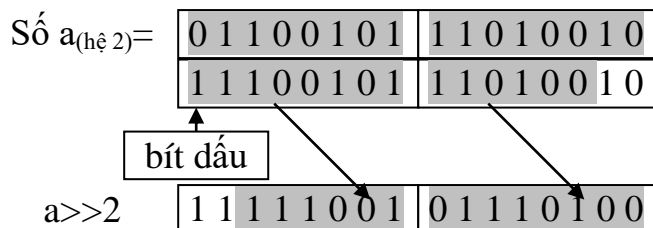
a_(hệ 2) = 0010 0110 1000 1011

a_(hệ 10) = 9867

+ Đối với toán hạng có kiểu **singed**

Phép toán dịch phải sẽ điền các bit 1 (bít dấu) vào bên trái

Ví dụ: số $a_{(hệ\ 10)} = 26066$



Kết quả của phép dịch:

$a_{(hệ\ 2)} = 0001\ 1001\ 0111\ 0100$

$a_{(hệ\ 10)} = 6516$

1.6.2.9. Toán tử chuyển đổi kiểu

Một biểu thức của C/C++ có thể gồm những toán hạng thuộc các kiểu khác nhau. Tuy nhiên trong quá trình tính toán giá trị của biểu thức xảy ra sự chuyển đổi dữ liệu để chúng có thể tham gia tính toán giá trị biểu thức.

Chuyển đổi kiểu tự động:

+ Chuyển đổi dữ liệu số học

Các toán hạng của biểu thức có thể thuộc các kiểu số học khác nhau như kiểu số nguyên hay kiểu số thực. Nguyên tắc chuyển đổi kiểu tự động là máy tính sẽ chuyển đổi từ kiểu dữ liệu đơn giản thành kiểu dữ liệu cao hơn (nếu cần) để quy đổi kiểu kết quả. Cụ thể chuyển kiểu dữ liệu tự động tuân theo thứ tự sau:

int → **long** → **float** → **double** → **long double**

Ví dụ: Ta có biến n kiểu số nguyên **int**, a kiểu số nguyên **long**, b kiểu số thực **float**

Ta có biểu thức gán như sau: $ketqua = n/a + b$ thì đầu tiên n được chuyển thành kiểu **long** để tham gia vào biểu thức tính n/a , sau đó kết quả của phép toán n/a được tự động chuyển thành kiểu **float** để tham gia vào phép tính $n/a + b$.

Phép chuyển đổi kiểu được tự động thực hiện trong phép gán. Khi này kiểu của vế phải sẽ được tự động chuyển thành kiểu của vế trái đó chính là kiểu của kết quả.

Trong ví dụ trên nếu biến $ketqua$ có kiểu **int** thì giá trị cuối cùng sẽ bị ngắt bỏ phần thập phân để thành một giá trị nguyên. Nếu biến $ketqua$ có kiểu **float** thì kết quả sẽ không bị mất phần thập phân.

+ Chuyển đổi dữ liệu kiểu **char** và **short**

Trong biểu thức nếu có các toán hạng có kiểu **char** hay kiểu **short** thì các toán hạng này được tự động chuyển sang kiểu **int** trước khi tham gia tính toán.

Tuy nhiên nếu gặp trường hợp biểu thức có dạng 'A' + 1 thì C/C++ sẽ lấy mã ASCII của ký tự 'A' là 65 cộng với 1 thành 66. Số 66 này được hiểu là số nguyên 66 hoặc ký tự 'B'.

Ví dụ 1.6:

<pre>#include <stdio.h> void main() { int x; char c; x='A'+1; printf("ket qua =%c",x); }</pre>	<pre>ketqua=B</pre>
--	---------------------

Nhưng nếu trong câu lệnh dòng 7 sửa định dạng in ra kết quả là %d `printf("ket qua =%d",x);` thì kết quả sẽ là `ketqua=66`.

Chuyển đổi kiểu ép buộc (ép kiểu):

Các toán tử chuyển đổi kiểu cho phép ta chuyển đổi dữ liệu từ kiểu này sang kiểu khác. Có vài cách để làm việc này trong C++, cách cơ bản nhất được thừa kế từ ngôn ngữ C theo cú pháp sau: (*tenkieu*) biểu thức;

Ví dụ 1.7:

```
int a=10; int b=3;
float ketqua1, ketqua2;
ketqua1=a/b;
ketqua2=(float)a/b;//ketqua2=3.333 (giá trị thực là
thương của phép chia a cho b)
cout<<"ketqua1="<<ketqua1;//ketqua1=3 (phần nguyên của
phép chia a cho b);
```

Ngoài ra trong C++ còn cho phép viết theo cú pháp: *tenkieu*(biểu thức)

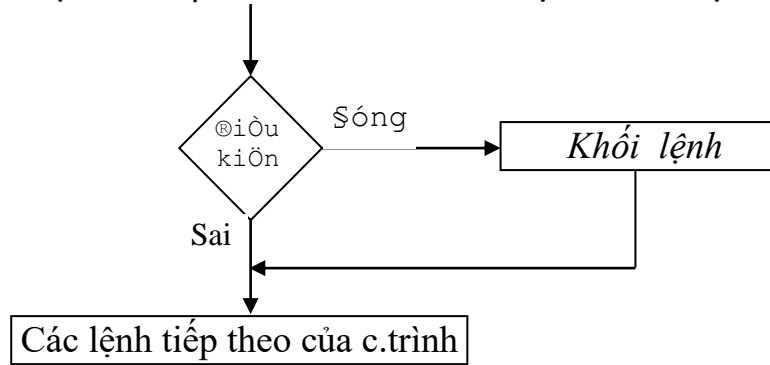
CHƯƠNG 2: CÁC CẤU TRÚC ĐIỀU KHIỂN CHƯƠNG TRÌNH

Một chương trình thường không chỉ bao gồm các lệnh tuần tự nối tiếp nhau. Trong quá trình chạy nó có thể rẽ nhánh hay lặp lại một đoạn mã nào đó. Để làm điều này chúng ta sử dụng các cấu trúc điều khiển.

2.1 Cấu trúc rẽ nhánh if...else

- **Dạng 1:** Thực hiện *Khối Lệnh* khi biểu thức điều kiện đưa ra được thỏa mãn.

Sơ đồ mô tả:



Hình 2.1

Cú pháp: if (*biểu thức điều kiện*)

Khối lệnh;

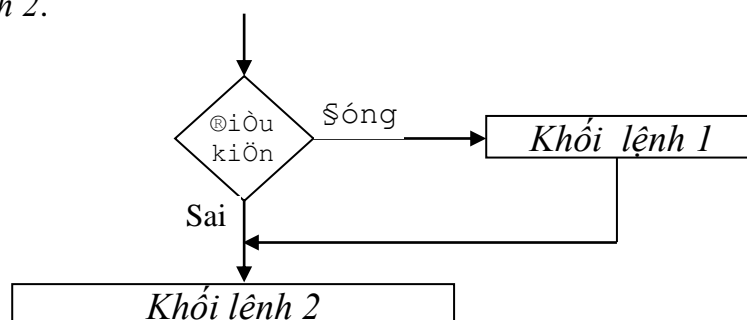
Ghi chú: *Khối lệnh* có thể là một câu lệnh đơn hoặc khối lệnh (lệnh gộp- là nhóm lệnh được đặt trong cặp dấu ngoặc nhọn {các lệnh}).

Ví dụ 2.1: Nhập vào 2 số, sau đó tìm số lớn nhất trong hai số đã nhập

<pre>#include <iostream.h> void main() { int a, b, max; cout<<"Nhập vào số a="; cin>>a; cout<<"Nhập vào số b="; cin>>b; max=a; if (b>max) max=b; cout <<" số lớn nhất trong hai số là max="<<max; } //ket thuc ham main()</pre>	<p>Nhập vào số a=4 Nhập vào số b=5 số lớn nhất trong hai số là max=5</p>
--	--

- **Dạng 2:** Thực hiện *Khối lệnh 1* nếu biểu thức điều kiện được thỏa mãn, ngược lại thực hiện *Khối lệnh 2*.

Sơ đồ mô tả:



Hình 2.2

Chương 2: Các cấu trúc điều khiển chương trình

Cú pháp: if (*biểu thức điều kiện*)

Khởi lệnh 1;

else

Khởi lệnh 2;

Ví dụ 2.2:

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

```
void main()
```

```
{
```

```
    int a, b;
```

```
    cout<<"Nhập vào số a="; cin>>a;
```

```
    cout<<"Nhập vào số b="; cin>>b;
```

```
    if (a > b)
```

```
        max=a;
```

```
    else
```

```
        max=b;
```

```
    cout<<"số lớn nhất trong hai số là max=";
```

```
    cout <<max;
```

```
}//kết thúc hàm main()
```

Nhập vào số a=4

Nhập vào số b=5

số lớn nhất trong hai số là max=5

- **Dạng 3:** Nếu biểu thức điều kiện 1 thỏa mãn thì thực hiện *Khởi lệnh 1* ngược lại nếu biểu thức điều kiện 1 không thỏa mãn thì xét tiếp *biểu thức điều kiện 2*, nếu biểu thức điều kiện 2 được thỏa mãn thì thực hiện *Khởi lệnh 2*, và cứ như vậy ta có thể sử dụng nhiều lần lệnh else if để xét nhiều trường hợp:

Cú pháp: if (*biểu thức điều kiện 1*)

Khởi lệnh 1;

else if (*biểu thức điều kiện 2*)

Khởi lệnh 2;

.....

else if (*biểu thức điều kiện n*)

Khởi lệnh n;

else *Khởi lệnh n+1;*

Ví dụ 2.3: Viết chương trình nhập vào điểm trung bình học tập của một sinh viên. Và cho biết sinh viên này có học lực Giỏi, Khá, TBK, TB hay Yếu. Với điều kiện như sau: Nếu điểm trung bình ≥ 8.0 thì học lực giỏi, điểm trung bình từ 7 đến dưới 8 thì học lực khá, điểm trung bình từ 6.5 đến dưới 7 thì học lực TBK, điểm trung bình từ 5 đến dưới 6.5 thì học lực TB, còn lại là học lực yếu.

<pre>#include <iostream.h> #include <stdio.h> void main() { float dtb; cout<<"Nhap vao diem trung binh:\n"; cin>>dtb; if (dtb >= 8) cout<<"Hoc sinh nay xep loai gioi"; else if (dtb>=7) cout<<"Hoc sinh nay xep loai kha"; else if (dtb>=6.5) cout<<"Hoc sinh nay xep loai TBK"; else if (dtb>=5) cout<<"Hoc sinh nay xep loai Trung binh"; else cout<<"Hoc sinh nay xep loai Yeu"; }</pre>	<pre>Nhap vao diem trung binh: 7.5 Hoc sinh nay xep loai kha</pre>
--	--

2.2 Cấu trúc rẽ nhánh switch...

Để viết chương trình phân thành nhiều nhánh (tức là chọn một trong nhiều khả năng) ta có thể dùng các câu lệnh if...else lồng nhau (Dạng 3). Tuy nhiên sử dụng câu lệnh **switch** trong nhiều trường hợp rẽ nhiều nhánh sẽ làm cho chương trình rõ ràng, ngắn gọn hơn.

Compiling NONAME00.CPP:

Cú pháp:

switch (*biểu thức nguyên*)

```
{
    case hằng 1:
        câu lệnh 1; break;
    case hằng 2:
        câu lệnh 2; break;
    .....
    case hằng n;
        câu lệnh n; break;
    default: câu lệnh n+1;
}
```

Giải thích:

Chương 2: Các cấu trúc điều khiển chương trình

Biểu thức nguyên là biểu thức có giá trị nguyên hoặc một kiểu nguyên (char, int, long,...). Khi giá trị này bằng *hằng i* thì máy sẽ thực hiện các câu lệnh có trong nhánh **case hằng i**. Sau khi thực hiện lệnh ở nhánh này nếu gặp câu lệnh **break**; thì sẽ nhảy ra khỏi cấu trúc **switch** và thực hiện các lệnh tiếp theo của chương trình (bỏ qua các nhánh khác trong thân **switch**). Nếu không có câu lệnh **break** ở mỗi nhánh thì sau khi thực hiện lệnh ở nhánh đó xong thực hiện tiếp các lệnh trong các nhánh khác của **switch**.

Nếu biểu thức nguyên có giá trị không bằng một trong các *hằng i* thì máy sẽ thực hiện *câu lệnh n+1* ở nhánh **default**.

Ví dụ 2.4:

<pre>#include <iostream.h> #include <stdio.h> void main() { int stt; cout<<"Nhan STT cua mon an ma ban thich\n"; cout<<"1. Thit bo ham\n"; cout<<"2. Thit ga nuong\n"; cout<<"3. Thit tho\n"; cin>>stt; switch (stt) { case 1: cout<<"Moi ban den voi nha hang Bo Tung Xeo";break; case 2: cout <<"Moi ban den voi khach san Dan toc"; break; case 3: cout<<"Moi ban den quan Thit thu rung"; break; default: cout<<"ban khong thich mon gi</pre>	<pre>Nhan so thu tu cua mon an ma ban thich 1. Thit bo ham 2. Thit ga nuong 3. Thit tho 2 Moi ban den voi khach san Dan toc</pre>
---	---

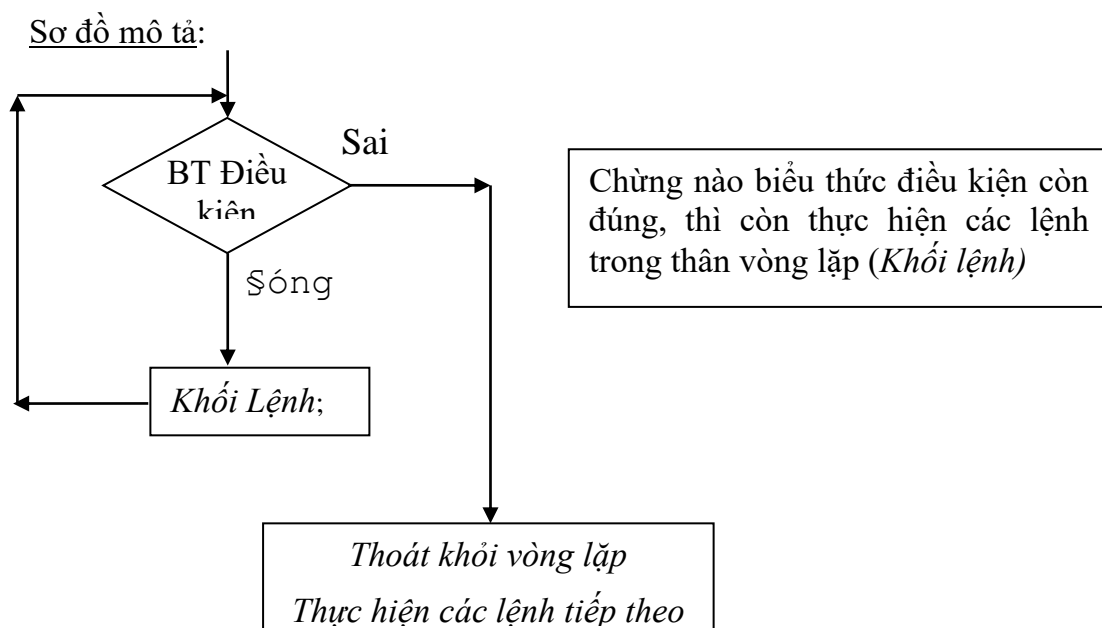
Ví dụ 2.5: Viết đoạn chương trình tính tiền điện phải trả = điện tiêu thụ * đơn giá. Trong đó đơn giá được tính theo mức như sau: điện tiêu thụ ≤ 50 số thì đơn giá 230, từ trên 50 đến 100 số thì đơn giá 480, từ trên 100 đến 150 số thì đơn giá 700, từ 150 trở lên thì đơn giá là 900.

```
#define MUC00 50
#define MUC01 MUC00+50
#define MUC02 MUC01+50
#include<iostream.h>
#include<stdio.h>
void main()
{
    int csdau, cscuoi;
    unsigned int tienDM=0;
    unsigned long tieuthu, tienVDM=0;
    cout<<"Nhập chỉ số đầu=";<<cin>>csdau;
    cout<<"Nhập chỉ số cuối=";<<cin>>cscuoi;
    tieuthu=cscuoi-csdau;
    cout<<"Diện tích thu "<<tieuthu<<"\n";
    switch ((tieuthu>150)?3:(tieuthu>100)?2:(tieuthu>50)?1:0)
    {
        case 3: tienVDM = 900*(tieuthu-150); cout<<tienVDM<<"\n";
                tieuthu=150;
        case 2: tienVDM+=700*(tieuthu-100); cout<<tienVDM<<"\n";
                tieuthu=100;
        case 1: tienVDM+=480*(tieuthu-50); cout<<tienVDM<<"\n";
                tieuthu=50;
        case 0: tienDM=230*tieuthu;
                default: break;
    }
    cout<<" Tiền trong DM "<<tienDM;
    cout<<"\n Tiền vượt DM "<<tienVDM;
}
```

2.3 Cấu trúc lặp while...

Cú pháp:

while (Biểu thức điều kiện)
Khối lệnh;



Hình 2.3

Chương 2: Các cấu trúc điều khiển chương trình

Nguyên tắc hoạt động: Vòng lặp **while** thực hiện kiểm tra điều kiện trước sau đó nếu điều kiện thỏa mãn (*biểu thức điều kiện* nhận giá trị true) thì thực hiện các lệnh trong thân vòng lặp (*Khối lệnh*), và lại quay lên kiểm tra *điều kiện*, chừng nào điều kiện còn đúng thì còn thực hiện *Khối lệnh*, ngược lại nếu *biểu thức điều kiện* nhận giá trị false thì thoát khỏi vòng lặp và thực hiện các lệnh tiếp theo của chương trình.

Chú ý: Trong thân vòng lặp **while** phải chứa câu lệnh làm thay đổi giá trị của *biểu thức điều kiện* theo chiều hướng tiến dần đến việc làm cho *biểu thức điều kiện* không còn đúng nữa. Thì vòng lặp while mới có điểm dừng, ngược lại vòng lặp sẽ trở thành lặp vô hạn.

Ví dụ 2.6: Viết chương trình in ra màn hình dòng các số từ 0 đến n với n nhập vào từ bàn phím:

```
#include <iostream.h>
void main()
{ int n,i;
  cout<<"Nhập vào số n=";
  cin>>n; i=0; //gan gia tri ban dau cho i
  while (i<=n)
  {
    cout<<i<<" ";
    i=i+1; //cau lenh lam thay doi gia tri bien i
  } }
```

Ví dụ 2.7: Viết chương trình yêu cầu nhập vào mật khẩu, chừng nào mật khẩu chưa đúng thì còn phải nhập lại. Nếu đúng mật khẩu thì hiện ra dòng thông báo “chúc mừng bạn, mật khẩu đã đúng”. (mật khẩu ở đây là chuỗi “happy”).

<pre>#include <iostream.h> #include <string.h> #include <stdio.h> void main() { char mk[7]; cout<<"Nhập vào mật khẩu "; cin.get(mk,7); while (strcmp(mk,"happy")!=0) { cout<<"\nNhập lại mật khẩu "; scanf("%s",&mk); } cout<<"\nChúc mừng bạn, mật khẩu đã đúng";</pre>	<pre>Nhập vào mật khẩu 123 Nhập lại mật khẩu happy Nhập lại mật khẩu haapy Nhập lại mật khẩu happy Nhập lại mật khẩu happyy Chúc mừng bạn, mật khẩu đã đúng</pre>
--	---

2.4 Cấu trúc lặp do...while

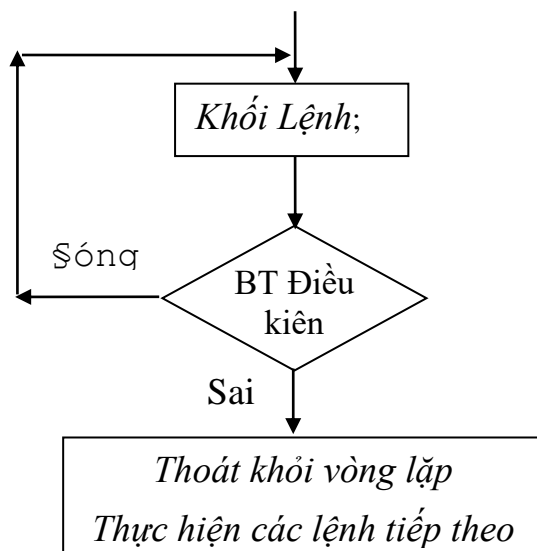
Cú pháp:

do

Khởi lệnh;

While (*biểu thức điều kiện*);

Sơ đồ mô tả:



Hoạt động: Thực hiện *Khởi lệnh* sau đó mới tiến hành kiểm tra điều kiện nếu điều kiện còn thỏa mãn, thì còn thực hiện *Khởi lệnh*; Nếu điều kiện sai thì thoát ra khỏi vòng lặp. Thực hiện các lệnh tiếp theo của chương trình.

Hình 2.4

Vòng lặp **do...while** có tác dụng giống như vòng lặp **while** chỉ khác nhau ở một điểm là vòng lặp **while** kiểm tra điều kiện trước khi thực hiện khối lệnh trong thân vòng lặp, còn vòng lặp **do...while** lại thực hiện *Khởi lệnh* rồi mới thực hiện kiểm tra biểu thức điều kiện. Như vậy ở vòng lặp **do...while** thì *Khởi lệnh* luôn được thực hiện ít nhất một lần ngay cả trong trường hợp *biểu thức điều kiện* sai

Ví dụ 2.8: Viết lại chương trình kiểm tra mật khẩu trên đây sử dụng vòng lặp **do...while**

<pre>#include <iostream.h> #include <string.h> #include <stdio.h> void main() { char mk[7]; do { cout<<"\nNhap vao mat khau "; scanf("%s",&mk); } while (strcmp(mk,"happy") !=0); cout<<"\nChuc mung ban,matkhau da dung";}</pre>	<p>Nhap vao mat khau 1233</p> <p>Nhap vao mat khau hap</p> <p>Nhap vao mat khau happy</p> <p>Nhap vao mat khau happyy</p> <p>Chuc mung ban, mat khau da dung</p>
---	--

Ví dụ 2.9: Viết chương trình cho phép nhập các số từ bàn phím, việc nhập số dừng lại khi gặp số 0. Tính tổng của các số đã nhập vào và in ra màn hình kết quả.

Chương 2: Các cấu trúc điều khiển chương trình

<pre>#include<iostream.h> void main() { int so, tong=0; do{ cout<<"Nhap mot so,nhap so 0 de dung "; cin>>so; tong = tong+so; } while (so!=0); cout<<"Tong cua cac so vua nhap La:"<<tong;}</pre>	<pre>Nhap vao mot so, nhap so 0 de dung 4 Nhap vao mot so, nhap so 0 de dung 5 Nhap vao mot so, nhap so 0 de dung 6 Nhap vao mot so, nhap so 0 de dung 0 Tong cua cac so vua nhap la:15</pre>
---	---

2.4 Cấu trúc lặp for...

Cú pháp:

for (*[biểu thức khởi tạo]; [biểu thức kiểm tra]; [biểu thức tăng giảm]*)

Khởi lệnh;

Trong đó:

[biểu thức khởi tạo]: Thông thường là một phép gán để khởi tạo giá trị ban đầu cho biến điều khiển. Lệnh này chỉ được thực hiện 1 lần.

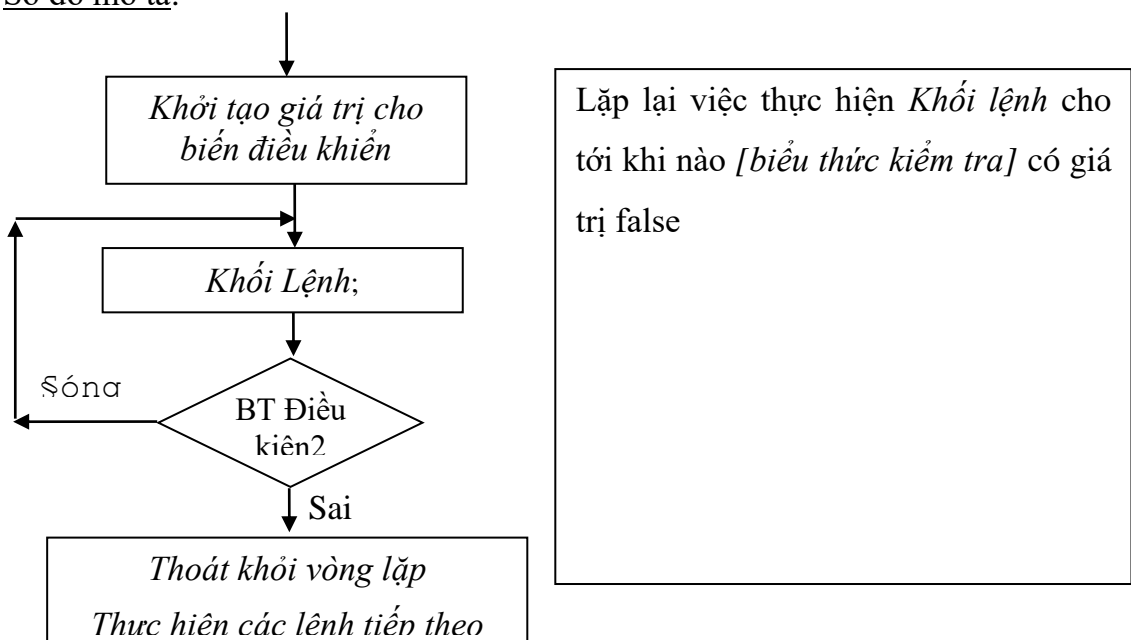
[biểu thức kiểm tra]: Là một biểu thức logic thể hiện điều kiện để tiếp tục vòng lặp

[biểu thức tăng giảm]: Là một phép gán dùng để thay đổi giá trị của biến điều khiển.

Một trong 3 biểu thức này đều có thể vắng mặt trong câu lệnh for hoặc cả 3 đều vắng mặt tuy nhiên biểu thức nào vắng mặt vẫn phải để chỗ trống của biểu thức đó và vẫn ngăn cách nó với biểu thức khác bằng một dấu chấm phẩy.

Ví dụ: for (i=1; ;i++) // câu lệnh for vắng mặt biểu thức kiểm tra

Sơ đồ mô tả:



Hình 2.5

Chương 2: Các cấu trúc điều khiển chương trình

Lưu ý:

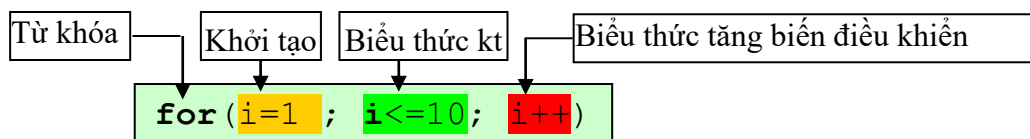
- [Biểu thức khởi tạo] bao giờ cũng chỉ được tính một lần
- Nếu [Biểu thức kiểm tra] vắng mặt thì nó luôn được xem là đúng. Trong trường hợp này để thoát khỏi vòng lặp for phải sử dụng câu lệnh break, goto, hoặc return viết bên trong thân for.

- Tại vị trí của mỗi biểu thức ta có thể đặt một vài biểu thức ngăn cách nhau bởi dấu phẩy. Khi đó các biểu thức này được xác định từ trái qua phải.

Ví dụ 2.10: Viết chương trình sử dụng vòng lặp for để in ra các số từ 0 đến 10

<pre>#include <iostream.h> void main () { int i; for (i=1;i<=10;i++) { cout<<i<<" "; } }</pre>	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td></tr></table>	1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10		

Trong đó các thành phần của câu lệnh for



Chú ý: Trong vòng lặp for không cần có câu lệnh làm thay đổi giá trị của biến điều khiển như trong vòng lặp **while** và **do...while**. Mà biến điều khiển sẽ tự động thay đổi giá trị sau mỗi lần lặp, giá trị của biến điều khiển có thể tăng hay giảm, và mỗi lần tăng hay giảm bao nhiêu là tùy thuộc [biểu thức tăng giảm]

Ví dụ 2.11:

```
#include <iostream.h>
void main ()
{
    int i;
    for (i=1;i<=10;i=i+2)/*Tăng i lên 2 đơn vị sau
                           mỗi lần lặp*/
    {
        cout<<i<<" "; //in ra màn hình 1 3 5 7 9
    }
}
```

2.5 Lệnh break và continue trong thân vòng lặp

Chương 2: Các cấu trúc điều khiển chương trình

Sử dụng lệnh *break* để thoát khỏi vòng lặp ngay cả khi điều kiện dừng chưa được thoả mãn. Lệnh này thường được dùng để kết thúc một vòng lặp không xác định, hoặc buộc một vòng lặp phải kết thúc giữa chừng thay vì kết thúc một cách bình thường.

Ví dụ 2.12: Chúng ta sẽ dừng việc đếm ngược trước khi vòng lặp for kết thúc:

<pre>// break loop example #include <iostream.h> void main () { int n; for (n=10; n>0; n--) { cout << n << ", "; if (n==3) { cout<<"Dem lui tam dung!"; break; } }//end for }//end main</pre>	10, 9, 8, 7, 6, 5, 4, Dem lui tam dung!
--	--

Lệnh *continue*.

Lệnh *continue* làm cho chương trình bỏ qua phần còn lại của vòng lặp và nhảy sang lần lặp tiếp theo. Ví dụ chúng ta sẽ bỏ qua số 5 trong phần đếm ngược:

<pre>// break loop example #include <iostream.h> int main () { for (int n=10; n>0; n--) { if (n==5) continue; cout << n << ", "; } cout << "Good bye!"; return 0; }</pre>	10, 9, 8, 7, 6, 4, 3, 2, 1, Good bye!
--	--

2.6 Bài tập

- 1) Viết chương trình nhập vào một số nguyên và cho biết tính chẵn lẻ của nó
- 2) Viết chương trình giải phương trình bậc 2 với 3 hệ số a, b, c nhập vào từ bàn phím.
- 3) Viết chương trình nhập vào tuổi của một người. Và cho biết người này còn trẻ, trung tuổi hay đã già. Nếu tuổi <40 thì in ra *còn trẻ*, tuổi từ 40 đến dưới 60 thì *trung tuổi*, tuổi từ 60 trở lên thì in ra là *đã già*

Chương 2: Các cấu trúc điều khiển chương trình

4) Viết chương trình nhập vào hai số x, y . Nếu $x > y$ thì cho phép hoán đổi giá trị của hai biến này. Rồi in kết quả sau khi hoán đổi.

5) Viết chương trình thi trắc nghiệm với nội dung sau:

Câu hỏi: “Ai là người đã đề xuất ra ngôn ngữ turbo C”

Trả lời:

Ken Thompson

Niklaus Wirth

.Dennis Ritchie

Nếu chọn sai máy thông báo “bạn đã trả lời sai”. Nếu chọn đúng máy thông báo “bạn đã trả lời đúng”

6) Viết chương trình nhập vào một số nguyên và kiểm tra xem một số có phải là một số nguyên tố hay không.

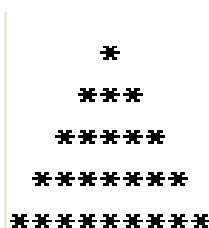
Gợi ý thuật giải:

Lần lượt kiểm tra các số tự nhiên từ 1 đến $n/2$ nếu gặp một số i nào đó mà n chia hết cho i thì lập tức dừng lại và kết luận n không phải là số nguyên tố. Ngược lại kết thúc vòng lặp mà không gặp i nào mà n chia hết thì kết luận n là số nguyên tố.

7)Viết các chương trình in ra màn hình các hình tam giác như sau:



Hình a



Hình b



Hình c

Gợi ý thuật giải:

Dựa vào số dấu * trên mỗi dòng, để tìm ra quy luật.

Ví dụ: Trên hình a, trên dòng 1 có một dấu *, dòng 2 có 2 dấu *, ..., dòng n có n dấu *. Hình b dòng 1 có 1 dấu *, dòng 2 có 3 dấu *, dòng 3 có 5 dấu *,... Những vị trí trống được viết bằng dấu cách “ ”.

8) Viết chương trình tính tổng của n số nguyên đầu tiên với n nhập vào từ bàn phím . ($S=1+2+3+...+n$)

9) Viết chương trình tính tổng sau:

$S=1^2+2^2+3^2+...+n^2$ với n nhập vào từ bàn phím.

10)Viết chương trình tính $n!=1*2*3*...*(n-1)*n$

11)Viết chương trình tính các tổng sau (với n nhập vào từ bàn phím):

a. $S=1+3+5+7+...+(2*n-1)$. b. $S=1*3+2*4+3*5+...+n*(n+2)$

c. $S=1/2+2/3+...+n/(n+1)$

CHƯƠNG 3: HÀM

Chương trình con là những đoạn chương trình thực hiện một công việc nào đó, được người lập trình hay nhà thiết kế ngôn ngữ viết sẵn để khi cần thực hiện công việc tương tự thì người sử dụng chỉ cần gọi chương trình con này ra, như vậy sẽ tiết kiệm được thời gian và công sức viết lại và làm cho chương trình trở lên sáng sủa hơn, dễ sửa lỗi hơn.

Trong đa số các ngôn ngữ lập trình bậc cao, có hai loại chương trình con đó là thủ tục và hàm.

Hàm là một chương trình con dùng để tính toán một giá trị và trả lại giá trị đó thông qua tên hàm.

Thủ tục là một chương trình con thực hiện một công việc nào đó, thủ tục cũng có thể trả về giá trị nhưng thông qua tham biến hoặc biến toàn cục chứ không trả về giá trị thông qua tên thủ tục.

Sự khác nhau cơ bản giữa hàm và thủ tục là hàm trả về giá trị thông qua tên hàm nên lời gọi hàm có thể tham gia vào một biểu thức tính toán như là một toán hạng. Còn thủ tục không trả về giá trị thông qua tên thủ tục nên lời gọi thủ tục không thể tham gia vào một biểu thức tính toán như là một toán hạng. Lời gọi thủ tục đứng độc lập như một câu lệnh hoàn chỉnh.

Trong ngôn ngữ lập trình C/C++ chỉ có một loại chương trình con là chương trình con dạng HÀM.

3.1 Hàm chuẩn và hàm tự tạo

Hàm chuẩn là các hàm đã được định nghĩa sẵn trong thư viện hàm của ngôn ngữ lập trình. Lập trình viên có thể sử dụng hàm chuẩn mà không cần định nghĩa lại các hàm đó.

Bảng 3.1 Một số hàm toán học thông dụng

Tên hàm	Ý nghĩa	Tập thư viện chứa hàm
abs(int x)	Trả về giá trị tuyệt đối của x	math.h
cos(double x)	Trả về giá trị cosin của x	math.h
sin (double x)	Trả về giá trị sin của x	math.h
sqrt(double x)	Trả về giá trị căn bậc 2 của x	math.h
pow(double x, double y)	Tính x^y	math.h
exp (double x)	Tính e^x	math.h
random(n)	Cho số ngẫu nhiên trong khoảng $0 \div n-1$	stdlib.h

3.2 Xây dựng hàm tự tạo

Như trong phần cấu trúc một chương trình C/C++ đã đề cập, việc xây dựng các hàm tự tạo có thể nằm trước hàm main hoặc sau hàm main. Trong phần này chỉ đề cập đến cách xây dựng hàm tự tạo ở vị trí trước hàm main.

Cú pháp: *kieuham* *tenham*(danh sách tham số hình thức)

```
{
    Các lệnh trong thân hàm;
    return giatri_tra_ve;
}
```

Trong đó:

- *kieuham* là kiểu dữ liệu được trả về của hàm
- *tenham* là tên đặt cho hàm theo nguyên tắc đặt tên (định danh) đã nêu.
- danh sách tham số hình thức: là danh sách các tham số hình thức, mỗi tham số được ngăn cách nhau bởi dấu phẩy “,”. Số lượng tham số hình thức tùy thuộc vào lập trình viên, có thể không có tham số hình thức.

Danh sách tham số hình thức được viết dưới dạng:

(*kieu thamso1, kieu thamso2, ..., kieu thamso_n*);

Trong đó *kieu* là kiểu dữ liệu của từng tham số.

- Câu lệnh *return giatri_tra_ve*; là câu lệnh gán một giá trị trả về cho tên hàm.

Đối với hàm không kiểu (void) thì *giatri_tra_ve* là 0.

Ví dụ 3.1: Viết chương trình cho phép nhập vào 3 số nguyên từ bàn phím và tìm số lớn nhất trong 3 số đó. Yêu cầu chương trình có sử dụng hàm *timmax*.

<pre>#include <iostream.h> int timmax(int x, int y, int z)//xd hàm { int max; if (x>y) max=x; else max=y; if (z>max)max=z; return max;// trả về gtri max cho hàm } //kết thúc xây dựng hàm void main() { int a,b,c; cout<<"Nhập vào 3 số a, b, c"; cin>>a>>b>>c; cout<<" Số lớn nhất trong 3 số là :"; cout<<timmax(a,b,c); //lời gọi hàm }</pre>	<p>Nhập vào 3 số a, b, c 31 15 67 Số lớn nhất trong 3 số là :67</p>
---	---

3.3 Sử dụng hàm (gọi hàm)

Trong ngôn ngữ C/C++ lời gọi hàm được sử dụng như một toán hạng tham gia vào một biểu thức trong thân chương trình chính. Hoặc có thể đứng độc lập như một câu lệnh hoàn chỉnh.

Cú pháp: **tenham**(*danh sách tham số thực sự*)

Trong đó danh sách tham số thực sự phải tương đương với danh sách tham số hình thức về kiểu, số lượng, và về vị trí.

Trong ví dụ 3.1 hàm timmax(x, y, z) có 3 tham số hình thức là x, y, z có kiểu int. Như vậy trong lời gọi hàm timmax (a,b,c) cũng phải có 3 tham số thực tương ứng a, b, c có kiểu nguyên.

Ví dụ 3.2: Viết chương trình in ra n lần một chuỗi nhập vào từ bàn phím.

<pre>#include <iostream.h> #include <stdio.h> #include <string.h> void inchuoi(char xau[15], int n) { int i; for (i=0;i<=n-1;i++) { cout<<xau; cout<<"\n"; } } void main() { int n; char xau[15]; cout<<"Nhập vào số lần in xau n="; cin>>n; cout<<" Nhập vào xau"; gets(xau); inchuoi(xau,n); //lời gọi hàm }</pre>	<pre>Nhập vào số lần in xau n=5 Nhập vào xau xin chào bạn xin chào bạn xin chào bạn xin chào bạn xin chào bạn xin chào bạn</pre>
---	--

3.4 Tham số và nguyên tắc truyền tham số

Tham số có hai loại:

Tham số hình thức: Là các tham số có trong phần xây dựng hàm. Tham số hình thức có hai loại:

+ Tham số giá trị: Trong các ví dụ trên chúng ta đều sử dụng tham số giá trị trong phần xây dựng hàm. Các tham số này đóng vai trò là đầu vào cho hàm, và giá trị của nó không bị thay đổi sau khi ra khỏi hàm.

+ Tham số biến: Được sử dụng trong trường hợp chúng ta muốn nhận lại sự thay đổi của chúng sau khi thực hiện hàm. Tham số biến phải được đặt thêm dấu & trước tên biến.

Chương 3 Hàm

Đây chính là khái niệm tham chiếu trong C++

Ngôn ngữ C++ giới thiệu một khái niệm mới đó là “reference” tạm dịch là “tham chiếu”. Về mặt bản chất tham chiếu chính là bí danh của một vùng nhớ được cấp phát cho một biến nào đó. Hay ta có thể hiểu tham chiếu chứa địa chỉ của vùng nhớ đã được cấp phát cho biến.

Ví dụ: ta khai báo `int n=3;` và `int &p=n;` // *p là tham chiếu đến biến n.*

`cout<<"n="<<p;` // *in ra giá trị của vùng nhớ mà p tham chiếu đến*

Một tham chiếu phải được gắn với một biến để nó chứa địa chỉ của biến đó, và tham chiếu này khi đã gắn với một biến rồi thì không được tham chiếu tới biến khác nữa. Cũng như một bí danh phải được gắn với một tên người cụ thể, và ta có thể sử dụng tên của anh ta hay bí danh đều được.

Ví dụ: `int x,y,&p=x;` // *p tham chiếu đến x*

`&p=y;` // *Không hợp lệ*

Một tham chiếu có thể là một biến hoặc một tham số hình thức của hàm.

Quay trở lại mục tham số hình thức của hàm là dạng tham biến. Đây chính là cách truyền tham số cho hàm bằng tham chiếu.

Ví dụ 3.3: Ví dụ sau sử dụng hai hàm hoán vị đó là **hoanvi1** và **hoanvi2** để thực hiện hoán đổi giá trị của hai số.

```
#include <iostream.h>
void hoanvi1(int ,int );//Dùng tham trị
void hoanvi2(int &,int &);//Dùng tham chiếu
void main()
{
    int a=5,b=6;
    cout<<"trước khi hoanvi1 a="<<a<<"    b="<<b; hoanvi1(a,b);
    cout<<"\n Sau khi hoan vi 1:a="<<a<<"    b="<<b; a=5;b=6;
    cout<<"\n trước khi hoanvi2 a="<<a<<"    b="<<b;hoanvi2(a,b);
    cout<<"\n Sau khi hoan vi 2:a="<<a<<"    b="<<b;

}
void hoanvi1(int x, int y)
{
    int tg;
    tg=x;x=y;y=tg;
}
void hoanvi2(int &x,int &y)
{
    int tg;
    tg=x;x=y;y=tg;
}
```

Kết quả thực hiện chương trình


```

trước khi hoanvi1 a= 5   b=6
Sau khi hoan vi 1:a=5   b=6
trước khi hoanvi2 a= 5   b=6
Sau khi hoan vi 2:a=6   b=5

```

Khi gặp lời gọi hàm **hoanvi1(a,b)** thì máy tính sẽ truyền **giá trị** của a, b cho hai tham số hình thức tương ứng là x, và y. Lúc này việc hoán vị xảy ra tại vùng nhớ của x và y, còn vùng nhớ của tham số thực a và b không hề thay đổi, do vậy sau khi gọi hàm hoanvi1 giá trị của a, b vẫn giữ nguyên.

Khi gặp lời gọi hàm hoanvi2(a,b) thì **bản thân** hai tham số a, b được truyền cho tham chiếu x,y tương ứng. Lúc này việc hoán vị xảy ra trực tiếp tại vùng nhớ của a,b nên kết quả in ra a,b đã đảo giá trị cho nhau.

Ví dụ 3.4: Viết chương trình nhập giá trị cho hai biến x và y. Chương trình có sử dụng hàm nhapdl.

```

#include <iostream.h>
#include <stdio.h>
void nhapdl(int &x, int &y)
{
    cin>>x; cin>>y;
}
void main()
{
    int x,y;
    cout<<"Nhập vào số x, y";
    nhapdl(x,y);
    cout<<" x="<<x;
    cout<<" y="<<y;
}

```

```

Nhập vào số x, y  135
43
x=135 y=43

```

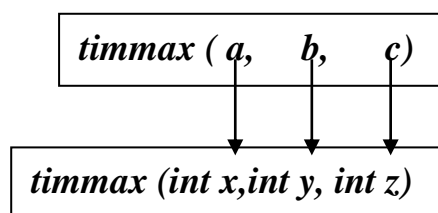
Truyền tham số: Khi biên dịch chương trình C/C++ máy tính luôn xuất phát từ hàm main(). Khi gặp câu lệnh gọi hàm trong thân hàm main, máy tính mới chuyển điểm thực hiện đến phần xây dựng chương trình con. Lúc này giá trị của các tham số thực sự mới được truyền cho các tham số hình thức tương ứng. Sau đó các giá trị này tham gia vào quá trình tính toán trong các câu lệnh bên trong hàm. Khi kết thúc các câu lệnh trong thân hàm thì máy tính lại thực hiện các lệnh tiếp theo trong thân chương trình chính.

Trong chương trình ví dụ 3.1.

Lời gọi hàm: **timmax(a,b,c)**

Trong đó a, b, c đã có giá trị nhập vào bởi câu lệnh **cin>>a>>b>>c;**

Truyền tham số



3.5 Hàm trùng tên (chồng hàm)

Trong C++ cho phép sử dụng cùng một tên cho nhiều hàm khác nhau và ta gọi đó là sự “chồng hàm”.

Hai hay nhiều hàm có thể có cùng tên nếu khai báo tham số của chúng khác nhau (khác nhau về số lượng tham số, về kiểu tham số), hay kiểu trả về của hàm khác nhau. Điều này có nghĩa là bạn có thể đặt cùng một tên cho nhiều hàm nếu chúng có số tham số khác nhau hay kiểu dữ liệu của các tham số khác nhau, hay là kiểu dữ liệu trả về khác nhau.

Ví dụ 3.5:

<pre>// overloaded function #include <iostream.h> int chia (int a, int b) { return (a/b); } float chia (float a, float b) { return (a/b); } int main () { int x=5,y=2; float n=5.0,m=2.0; cout << chia (x,y); cout << "\n"; cout << chia (n,m); return 0; }</pre>	<p>2</p> <p>2.5</p>
---	-----------------------------------

Trong ví dụ này chúng ta định nghĩa hai hàm có cùng tên **chia** nhưng một hàm dùng hai tham số kiểu **int** và hàm còn lại dùng kiểu **float**. Trình biên dịch sẽ biết cần phải gọi hàm nào bằng cách phân tích kiểu tham số khi hàm được gọi.

3.6 Hàm đệ quy

Định nghĩa: Hàm được gọi là đệ quy nếu bên trong thân hàm có lời gọi đến chính nó.

Ví dụ: Định nghĩa giai thừa của một số nguyên dương như sau:

$$n! = 1 * 2 * 3 * \dots * (n-1) * n = (n-1)! * n \text{ (Với } 0! = 1)$$

Như vậy để tính $n!$ ta kiểm tra nếu $n=0$ thì $n!=1$. Ngược lại $n! = (n-1)! * n$

Hàm đệ quy gồm hai phần:

- **Phần neo** : Trong đó định nghĩa giá trị của hàm hoặc thủ tục tại một giá trị của tham số.

- **Phần đệ quy**: Trong đó tác động cần được thực hiện cho giá trị hiện thời của các tham số được định nghĩa bằng các giá trị được định nghĩa trước đây.

Ở ví dụ tính giai thừa của một số nguyên không âm thì phần neo là câu lệnh đầu tiên.

$0!=1$; nó định nghĩa giá trị của hàm tại điểm 0.

Phần đệ quy là phần định nghĩa giá trị của $n!$ thông qua các giá trị của n và $(n-1)!$

Ví dụ 3.6: Viết chương trình tính $n!$. Sử dụng hàm đệ quy, và hàm không đệ quy

```
#include <stdio.h>
#include <conio.h>
/*Hàm tính n! bang de quy*/
unsigned int gt_dequy(int n)
{
    if (n==0) return 1;
    else return n*gt_dequy(n-1);
}
/*Hàm tính n! không de quy*/
unsigned int gt_khongdequy(int n)
{ unsigned int kq,i;
  kq=1;
  for (i=2;i<=n;i++) kq=kq*i;
  return kq;
}
void main()
{ int n;
  printf("\n Nhap so n can tinh giai thua ");
  scanf("%d",&n);
  printf("\nGoi ham de quy: %d !=%u",n,gt_dequy(n));
  printf("\nGoi ham khong de quy: %d != %u",
    n,gt_khongdequy(n));
}
```

Ví dụ 3.7: Viết hàm tìm USCLN của hai số nguyên a,b

```
int ucln(int a,int b)
{
    if(a==b) return a;
    else { if(a>b)      a=a-b;
          else        b=b-a; }
    return ucln(a,b);
}
```

Sử dụng hàm đệ quy trong chương trình sẽ làm chương trình dễ đọc, dễ hiểu và rõ ràng. Tuy nhiên trong đa số trường hợp thì hàm đệ quy tốn bộ nhớ nhiều hơn, và tốc độ thực hiện chương trình chậm hơn.

Như vậy tùy theo bài toán cụ thể mà người lập trình quyết định có nên dùng đệ quy hay không. Có trường hợp bắt buộc phải dùng đệ quy mới giải quyết được bài toán

3.7 Bài tập

Bài 1: Xây dựng chương trình tính tổng sau:

$$S = \max(a,b,c) + \min(x,y)$$

Trong đó: a, b, c, x, y là các giá trị thực bất kỳ nhập vào từ bàn phím. Yêu cầu chương trình có sử dụng hai chương trình con max và min.

Bài 2: Viết chương trình tính giai thừa của một số n nhập vào từ bàn phím. Yêu cầu trong chương trình có sử dụng hàm tính giai thừa.

Bài 3: Xây dựng chương trình tính tổng $S = a^n + b^m$ với a,b,m,n là các số nguyên nhập vào từ bàn phím, yêu cầu chương trình có sử dụng hàm tính lũy thừa.

Bài 4: Xây dựng hàm kiểm tra một số có phải nguyên tố không. Sử dụng hàm này để kiểm tra các số nguyên tố từ 1 đến n. Với n nhập vào từ bàn phím.

Bài 5: Xây dựng hàm kiểm tra một số có phải số hoàn hảo không. Sử dụng hàm này để kiểm tra các số hoàn hảo từ 1 đến n. Với n nhập vào từ bàn phím.

Bài 6: Xây dựng hàm đệ quy tìm USCLN của hai số a, b. Áp dụng kiểm tra hai số nguyên bất kỳ có nguyên tố cùng nhau không.

Bài 7: Viết hàm đệ quy tìm chữ số đầu tiên của số nguyên n.

Bài 8: Viết hàm đệ quy tính số Fibonacci F(n) với n nguyên dương nhập vào từ bàn phím. Biết rằng nếu $n \leq 2$ thì $F(n)=1$. Ngược lại $F(n)=F(n-2)+F(n-1)$

CHƯƠNG 4 MẢNG VÀ XÂU KÝ TỰ

4.1 Mảng

Mảng là tập hợp các phần tử có cùng kiểu dữ liệu và cùng tên chung, chỉ khác nhau ở chỉ số.

4.1.1 Mảng một chiều

4.1.1.1 Khai báo mảng một chiều

Cú pháp: *kieudl* *tenbienmang* [*sophantu*];

Trong đó: *kieudl* là kiểu dữ liệu chung cho các phần tử trong mảng. *kieudl* có thể là một trong các kiểu thông thường hoặc một kiểu dữ liệu do người lập trình tự định nghĩa.

tenbienmang: là tên của biến mảng (tên chung cho các phần tử trong mảng).

[*sophantu*]: là số phần tử tối đa mà mảng có thể chứa. Trong câu lệnh khai báo này bắt buộc phải chỉ rõ số phần tử.

Ví dụ: `int x[10];` // Khai báo mảng x có tối đa 10 phần tử kiểu số nguyên int

`float a[15];` // Khai báo mảng a có tối đa 15 phần tử kiểu số thực

Sau câu lệnh khai báo mảng thì máy tính giành ra vùng nhớ có kích thước bằng tổng kích thước của *kieudl* nhân với *sophantu* cho mảng. Ví dụ như sau câu lệnh khai báo `int x[10]` thì máy tính sẽ giành ra $2 \text{ byte} \times 10 = 20 \text{ byte}$ (2 byte là kích thước của kiểu int, 10 là số phần tử tối đa của mảng).

4.1.1.2 Truy xuất vào từng phần tử

Để truy xuất đến các phần tử trong mảng ta sử dụng tên biến mảng[chỉ số phần tử]. Ví dụ: `a[0]`, `a[1]`, ..., `a[n]`. Các phần tử của mảng được đánh số từ 0 đến $n-1$ (với n là số phần tử của mảng). Chúng được lưu trữ trong bộ nhớ như sau:

a[1]	a[2]	a[n]
------	------	------	------

4.1.1.3 Nhập/ xuất dữ liệu

- **Nhập dữ liệu cho mảng:**

Mỗi phần tử của mảng cũng đóng vai trò như một biến nên ta sử dụng cách nhập dữ liệu cho một biến để nhập dữ liệu cho từng phần tử trong mảng. Để đỡ “tốn” công viết chương trình ta sử dụng vòng lặp để làm việc này. Vòng lặp này sẽ thực hiện n lần câu lệnh nhập dữ liệu tương ứng với n phần tử trong mảng.

- Sử dụng câu lệnh của C: `scanf(“chuoidinh dang”, &tenbienmang[chiso])`

Ví dụ 4.1:

`n=5;` //mảng có 5 phần tử

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

```
{    printf(“Nhap phan tu thu %d=”,i);
    /*Nhap dl cho phan tu thu i trong mang*/
    scanf(“%d",&x[i]);    }
```

- Sử dụng câu lệnh cin trong C++

```
for (i=0; i<n; i++)
{
    cout<<"x["<<i<<"]=";
    cin>>x[i]; //Nhập dữ liệu cho phần tử x[i]
}
```

- Xuất dữ liệu mảng

Sử dụng vòng lặp để lần lượt in giá trị của từng phần tử mảng ra màn hình

Ví dụ 4.2:

```
for (i = 0; i<n ; i++)
    printf("%d ", x[i]); // in giá trị của phần tử thứ i;
hoặc ta có thể sử dụng lệnh cout
for (i = 0; i<n ; i++)
    cout<<x[i]<<" "; // in giá trị của phần tử thứ i;
```

Chú ý: Chương trình dịch C++ không kiểm tra giới hạn của chỉ số mảng. Ví dụ ta nhập vào một mảng x gồm 5 phần tử (chỉ số được đánh từ 0 đến 4), nếu ta lại sử dụng phần tử có chỉ số lớn hơn hoặc bằng 5 như x[5] hay x[6],...thì máy sẽ không báo lỗi khi biên dịch nhưng khi chạy chương trình thì phần tử x[5] và x[6] này có giá trị không xác định.

Ví dụ 4.3: Viết chương trình nhập vào một dãy số nguyên rồi in chúng theo chiều ngược lại.

<pre>#include <iostream.h> #include <stdio.h> void main() { int n,i, x[10]; cout<<"Nhập vào số phần tử của mảng"; cin>>n; for (i=0;i<n;i++) { cout<<"Nhập phần tử thứ:"<<i<<" "; cin>>x[i]; } cout<<" Các phần tử theo thứ tự ngược lại là:\n"; for (i=n-1;i>=0;i--) { cout<<x[i]<<" "; } }</pre>	<p>Nhập vào số phần tử của mảng 3 Nhập phần tử thứ:0 12 Nhập phần tử thứ:1 10 Nhập phần tử thứ:2 45 Các phần tử theo thứ tự ngược lại là: 45 10 12</p>
---	---

4.1.1.4 Khởi tạo giá trị ban đầu

Ta có thể kết hợp khởi gán giá trị cho các phần tử trong mảng trong câu lệnh khai báo biến mảng theo cú pháp như sau:

Cú pháp: *kieudl* tenbienmang []={Dữ liệu gán};

Trong đó: Dữ liệu gán là danh sách các giá trị được gán lần lượt cho các phần tử của mảng. Các giá trị này được viết ngăn cách nhau bởi dấu phẩy “,”.

Ví dụ: `int a[] = {10, 14, 12, 24, 56} ;`

Khai báo một mảng số nguyên a và gán 5 giá trị cho 5 phần tử của mảng. Sau câu lệnh này `a[0]` được gán giá trị 10, `a[1]` được gán giá trị 14,..., `a[4]` được gán giá trị 56.

Ví dụ 4.4: Nhập và chạy thử đoạn chương trình sau:

<pre>#include <iostream.h> void main() { int i, a[]={10, 14, 12, 24, 56} ; cout<<" Cac phan tu cua mang\n"; for (i=0;i<5;i++) { cout<<"a ["<<i<<"]="<<a[i]; cout<<"\n"; } }</pre>	<p>Cac phan tu cua mang</p> <p>a[0]=10 a[1]=14 a[2]=12 a[3]=24 a[4]=56</p>
--	--

4.1.2 Mảng hai chiều

Ở phần trên ta đã tìm hiểu về mảng một chiều. Mảng một chiều được sử dụng khi ta cần làm việc với một dãy số có cùng kiểu dữ liệu. Tuy nhiên trong trường hợp ta cần quản lý một bảng dữ liệu gồm các hàng và các cột thì cần đến mảng hai chiều. Một chiều biểu diễn cột, một chiều biểu diễn hàng. Ngoài ra hầu hết các ngôn ngữ lập trình bậc cao còn cho phép sử dụng mảng nhiều hơn hai chiều để quản lý dữ liệu.

4.1.2.1 Khai báo mảng hai chiều

Cú pháp: *kieudl* tenbienmang [sophatu1][sophantu2]...[sophantun];

Trong đó:

kieudl là kiểu của các phần tử trong mảng

sophantu1: là kích thước của chiều thứ nhất của mảng

sophantu2: là kích thước của chiều thứ hai của mảng

tương tự như vậy sophantun là kích thước của chiều thứ n.

Chú ý: Số phần tử phải là một hằng số, không được để trống

4.1.2.2 Nhập/ xuất dữ liệu

Mảng hai chiều được lưu trữ trong bộ nhớ như sau. Ví dụ ở đây là mảng `x[10][15]`:

<code>x[0][0]</code>	...	<code>x[0][15]</code>	<code>x[1][0]</code>	...	<code>x[1][15]</code>	...	<code>x[10][0]</code>	...	<code>x[10][15]</code>
----------------------	-----	-----------------------	----------------------	-----	-----------------------	-----	-----------------------	-----	------------------------

Nhập bằng lệnh cin của C++

Việc nhập dữ liệu cho các phần tử của mảng 2 chiều bằng lệnh **cin** của C++ cũng hoàn toàn tương tự như nhập dữ liệu cho mảng một chiều. Chỉ khác ở chỗ mảng hai chiều ta phải làm việc với hai chỉ số: chỉ số hàng và chỉ số cột.

Sử dụng hai vòng lặp for lồng nhau. Một vòng chạy theo chỉ số hàng, một vòng chạy theo chỉ số cột.

Ví dụ 4.5: Viết chương trình nhập vào một ma trận số nguyên có n hàng, m cột (với n, m nhập vào từ bàn phím). Sau đó tính tổng các phần tử trên hàng 2.

<pre>#include<iostream.h> void main() { int i,j,n,m,a[10][10],tong; cout<<"nhap vao so hang,cot n=,m="; cin>>n>>m; for (i=0;i<n;i++) for (j=0;j<m;j++) { cout<<"a["<<i<<"", "<<j<<""]="; cin>>a[i][j]; } cout<<"Mang vua nhap la\n"; for (i=0;i<n;i++) { cout<<"\t"; for (j=0;j<m;j++) cout<<a[i][j]<<" "; cout<<"\n"; } tong =0; for (j=0;j<m;j++) tong = tong + a[1][j]; cout<<"tong cua hang 2 la"<<tong; }</pre>	<pre>nhap vao so hang, so cot n=,m=3 3 a[0,0]=1 a[0,1]=2 a[0,2]=3 a[1,0]=4 a[1,1]=5 a[1,2]=6 a[2,0]=7 a[2,1]=8 a[2,2]=9 Mang vua nhap la 1 2 3 4 5 6 7 8 9 tong cua hang 2 la15</pre>
---	---

Nhập bằng lệnh scanf của C:

Trong môi trường của trình biên dịch C ta không thể dùng phép lấy địa chỉ để nhập dữ liệu cho từng phần tử trong mảng hai chiều.

Ví dụ: Trong C không cho phép câu lệnh sau: `scanf("%d",&x[i][j])`

Nhưng nếu biên dịch trong C++ thì lại cho phép việc này.

Tuy nhiên theo kinh nghiệm ta nên sử dụng một biến trung gian để nhập dữ liệu, sau đó gán giá trị của biến trung gian này cho từng phần tử của mảng. Cách này có tính ổn định hơn, bởi vì nếu sử dụng cách nhập trực tiếp dữ liệu vào biến mảng thì ta phải áp dụng phép toán lấy địa chỉ lên các phần tử của mảng, trong một số trường hợp khi kiểu dữ liệu thành phần không phải là kiểu nguyên thì hay xảy ra tình trạng treo máy.

Vậy để nhập dữ liệu cho mảng hai chiều ta nên làm theo cách sau, sử dụng biến tạm **tam**:

`scanf("%f", tam);//nhập dữ liệu kiểu số thực`

`a[i][j]=tam;`

4.1.3 Mảng là tham số của hàm

Trong phần xây dựng hàm:

Mảng một chiều:

kieuham *tenham* (*kieumang* *tenmang*[*sophantu*],...)

trong đó *sophantu* có thể để trống [] (đây là kiểu khai báo không tường minh)

Mảng hai chiều:

kieuham *tenham* (*kieumang* *tenmang*[*sophantu*][*sophantu*],...)

Ghi chú: Mặc dù trong một số trường hợp C/C++ cho phép kiểu khai báo không tường minh nhưng ta nên khai báo tường minh.

Trong lời gọi hàm:

Như ta đã biết cú pháp của lời gọi hàm là **tenham**(*danh sách tham số thực*). Nếu tham số hình thức (tham số trong phần xây dựng hàm) có kiểu mảng thì tham số thực sự cũng phải có kiểu mảng và lời gọi hàm theo cú pháp:

`tenham(tenbienmang,...);`

Lúc gọi hàm ta chỉ đưa tên biến mảng vào danh sách tham số thực mà không cần đưa thêm [số phần tử].

Ví dụ 4.6: Viết chương trình nhập vào một mảng gồm 5 số thực, sau đó in ra màn hình mảng vừa nhập. Có sử dụng hai hàm nhập dữ liệu và xuất dữ liệu:

```
#include <iostream.h>
int n;//Khai bao bien toan cuc n
void nhapdl(float x[])
{
    for (int i=0;i<n;i++)
        { cout<<"x["<<i<<"]="; cin>>x[i];}
}
void xuatdl(float x[])
{
    for (int i=0; i<n; i++)
        cout<<x[i]<<" ";
}
void main()
{
    float a[10];
    cout<<"Nhập vào số phần tử của mảng n="; cin>>n;
    cout<<" Nhập vào từng phần tử"; nhapdl(a);//Lời gọi hàm
    cout<<"Day vua nhap\n"; xuatdl(a);
}
```

4.2 Chuỗi ký tự

Trong C/C++ không có kiểu dữ liệu riêng cho dãy ký tự. Dãy ký tự trong C/C++ là một mảng ký tự một chiều, mỗi phần tử của mảng là một ký tự. Dãy ký tự còn được gọi là chuỗi hoặc xâu.

4.2.1 Khai báo

`char tenbien_chuoi[dodai];`

Trong đó: *char* là kiểu dữ liệu, *tenbien_chuoi* là tên của biến kiểu chuỗi, *dodai* là một hằng số quy định độ dài tối đa của chuỗi.

Ví dụ: `char hovaten[30];` //Khai báo biến chuỗi *hovaten* có độ dài tối đa 30 ký tự

Sau câu lệnh khai báo trên máy tính giành ra vùng nhớ 30 byte cho chuỗi *hovaten*. Và nếu ta nhập vào giá trị *Nguyen Thi Huong* thì nội dung của biến *hovaten* được chứa trong bộ nhớ như sau:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...	29
N	g	u	Y	e	n		T	h	i		H	u	o	n	g	\0	*	.	*

Trong đó: Độ dài của chuỗi nhập vào là 16 (các ký tự được đánh số từ 0). Ký tự thứ 17 là ký tự kết thúc chuỗi được tự động thêm vào cuối phần dữ liệu. Từ ký tự 18 đến ký tự thứ 30 là các ký tự không xác định.

Chú ý: Sau khi kết thúc nhập chuỗi ký tự thì một ký tự kết thúc chuỗi `\0` luôn được tự động thêm vào cuối. Cho nên nếu độ dài tối đa của chuỗi là *sophantu* thì chuỗi chỉ được phép nhận *sophantu-1* ký tự còn 1 ký tự giành cho ký tự kết thúc chuỗi.

Vừa khai báo vừa gán giá trị:

Ta có thể vừa khai báo vừa gán giá trị cho biến chuỗi như sau:

dạng 1: `char tenbien_chuoi[]="chuỗi khởi tạo";`

Ví dụ: `char hovaten[]="Nguyen Thi Van Anh";`

Sau câu lệnh khai báo dạng này máy tính sẽ giành ra số byte bằng số ký tự của “chuỗi khởi tạo” cộng thêm 1 cho ký tự kết thúc chuỗi.

dạng 2: `char tenbien_chuoi [dodai]="chuỗi khởi tạo";`

Ví dụ: `char hovaten[15]="Nguyen Van Anh";`

Chú ý: Không được khởi tạo giá trị cho biến chuỗi bằng một chuỗi có độ dài lớn hơn độ dài tối đa của biến chuỗi.

Ví dụ: Câu lệnh sau đây không hợp lệ:

`char hovaten[10]="Nguyen Van Anh";` //chuoi Nguyen Van Anh dài 14 ký tự

4.2.2 Hằng dãy ký tự

Trong C/C++ hằng dãy ký tự là một hay một nhóm ký tự được đặt trong cặp dấu ngoặc kép “ ”. Ví dụ “ Ngon ngu C”.

Hằng dãy ký tự được lưu trữ trong bộ nhớ bằng một dãy byte liền nhau, số byte bằng số ký tự có trong dãy cộng thêm 1 (1 ký tự kết thúc dãy).

Chú ý: Phân biệt hằng ký tự và hằng dãy ký tự. Hằng ký tự thì được đặt trong cặp dấu nháy đơn ‘ ’, còn hằng dãy ký tự thì được đặt trong cặp dấu nháy kép “ ”. Ví dụ cách viết “A” và ‘A’ là hoàn toàn khác nhau. Sự khác nhau thể hiện trên hai điểm sau:

Điểm thứ nhất:

“A” là hằng dãy ký tự, chiếm hai byte bộ nhớ:

A	\0
---	----

‘A’ là hằng ký tự, chiếm một byte bộ nhớ

Và điểm thứ 2: ‘A’ là hằng ký tự nên có thể tham gia vào biểu thức tính toán. Ví dụ ‘A’+1 (lấy mã ASCII của ký tự A là 65 cộng với 1). Còn đối hằng dãy ký “A” thì không làm được việc này.

4.2.3 Nhập chuỗi

Trong C:

Cách 1: Sử dụng lệnh `scanf("%s",&bienchuoi);`

Ví dụ: `scanf("%s",&hovaten); //nhap chuỗi hovaten`

Cách này chỉ nhập được chuỗi không có khoảng cách. Kết thúc chuỗi cần nhập bằng khoảng cách hoặc gõ Enter.

Chú ý: Nên sử dụng riêng câu lệnh nhập chuỗi cho mỗi biến chuỗi cần nhập. Và trước câu lệnh nhập chuỗi nên sử dụng lệnh `fflush(stdin)`. Lệnh này có tác dụng đẩy hết các ký tự còn lại trong dòng vào ra khỏi dòng vào `stdin`, để tránh việc nhận ký tự không mong muốn có trong dòng vào `stdin` vào chuỗi sẽ nhập.

Cách 2: Sử dụng lệnh `gets(bienchuoi);`

Hàm `gets` đọc các ký tự được gõ từ bàn phím (kể cả ký tự trống) và điền vào chuỗi `bienchuoi`. Việc nhập ký tự kết thúc nếu gặp ký tự xuống dòng ‘\n’ (khi ta gõ phím enter), hoặc số ký tự nhập vào đã vượt quá kích thước tối đa của chuỗi. Sau khi kết thúc việc nhập chuỗi thì một ký tự kết thúc chuỗi ‘\0’ được tự động thêm vào cuối chuỗi.

Cũng như `scanf` nếu `gets` đi sau các lệnh nhập dữ liệu khác thì nên sử dụng lệnh `fflush(stdin)` trước khi sử dụng `scanf` để đảm bảo độ chính xác của dữ liệu nhập vào.

Trong C++:

Cách 1: Sử dụng lệnh `cin>>bienchuoi;`

Nhập vào một chuỗi không có dấu cách.

Cách 2: Sử dụng lệnh `cin.get(bienchuoi,n);`

Câu lệnh này cho phép nhập vào n- 1 ký tự cho chuỗi `bienchuoi`. Kết thúc nhập bằng cách gõ phím Enter. Ký tự xuống dòng enter ‘\n’ không được đưa vào chuỗi, mà nằm ở dòng vào `stdin`.

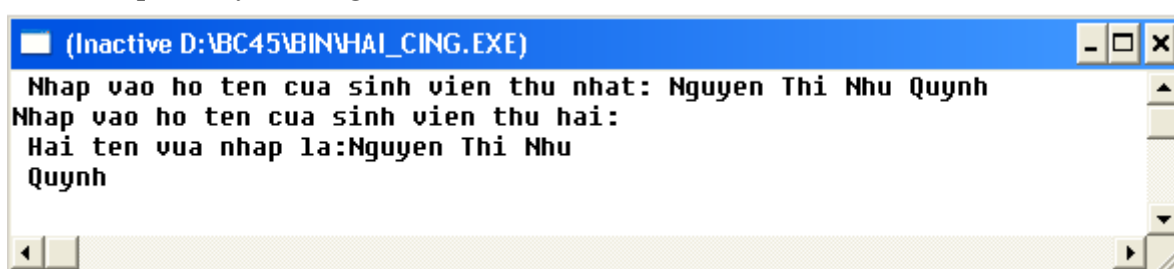
Nếu số ký tự gõ vào vượt quá n-1 ký tự thì máy tính chỉ lấy n-1 ký tự đầu tiên để gán cho `bienchuoi`, còn số ký tự thừa cũng nằm ở dòng vào `stdin`.

Điều này dẫn đến việc nếu chúng ta sử dụng nhiều câu lệnh nhập chuỗi trong một chương trình thì câu lệnh nhập chuỗi sau sẽ nhận ký tự enter từ dòng vào `stdin` dẫn đến hiện tượng “trôi lệnh nhập dãy ký tự”.

Ví dụ 4.7: Ta có đoạn chương trình sau đây:

```
#include <iostream.h>
void main()
{
    char hoten1[15], hoten2[15];
    cout<<"Nhập vào họ tên của sinh viên thứ nhất: ";
    cin.get(hoten1,15); // câu lệnh nhập chuỗi thứ nhất
    cout<<"Nhập vào họ tên của sinh viên thứ hai:";
    cin.get(hoten2,15); // câu lệnh nhập chuỗi thứ hai
    cout <<" Hai tên vừa nhập là:";
    cout<<hoten1<<"\n"<<hoten2;
}
```

Kết quả chạy chương trình



Khi chạy chương trình trên ta nhập vào chuỗi thứ nhất là Nguyen Thi Nhu Quynh, chuỗi này dài 20 ký tự, và máy tính cắt 14 ký tự đầu tiên đưa vào biến hoten1 (do câu lệnh `cin.get(hoten1,15)` chỉ cho phép nhập tối đa 15-1 ký tự cho biến hoten1). Còn lại 6 ký tự “Quynh” và ký tự khi ta gõ enter vẫn nằm ở dòng vào stdin nên các ký tự này được đẩy hết vào biến chuỗi hoten2. Và dẫn đến hiện tượng “trôi lệnh nhập chuỗi hoten2” tức là không cho phép nhập dữ liệu cho biến hoten2 nữa. Rồi thực hiện lệnh in ra màn hình nội dung hai biến hoten1, hoten2 như trên.

Để khắc phục hiện tượng trên ta có thể dùng một trong các cách sau:

Cách 1: Sử dụng lệnh `cin.get()` sau câu lệnh nhập chuỗi thứ nhất để lấy ký tự enter ra khỏi dòng vào stdin. Hoặc sử dụng lệnh `cin.ignore(1)` để bỏ qua ký tự enter. Và khi chạy chương trình phải nhập chuỗi có số ký tự không vượt quá số ký tự giới hạn. Ví dụ lệnh `cin.get(hoten1,15)` thì chỉ được nhập chuỗi có tối đa 14 ký tự.

Ví dụ 4.7: Có thể sửa lại như sau:

```
#include <iostream.h>
void main()
{
    char hoten1[15], hoten2[15];
    cout<<" Nhap vao ho ten cua sinh vien thu nhat: ";
    cin.get(hoten1,15);//câu lệnh nhập chuỗi thứ nhất
    cout<<"Nhap vao ho ten cua sinh vien thu hai:";
    cin.get();//hoặc cin.ignore
    cin.get(hoten2,15);// câu lệnh nhập chuỗi thứ hai
    cout <<" Hai ten vua nhap la:\n";
    cout<<hoten1<<"\n"<<hoten2;
}
```

Cách 2: Sử dụng lệnh `cin.getline(bienchuoi,n)`. Lệnh này chỉ khác lệnh `cin.get()` ở chỗ nó lấy cả ký tự enter vào chuỗi chứ không để ở dòng vào `stdin`. Như vậy sẽ không có hiện tượng trôi lệnh nhập do ký tự enter còn sót lại ở dòng vào `stdin`. Tuy nhiên khi sử dụng câu lệnh `cin.getline(bienchuoi,n)` thì chỉ được phép nhập vào `n-2` ký tự cho chuỗi còn lại 2 ký tự thì một là ký tự kết thúc chuỗi, và một là ký tự enter sẽ được thêm vào cuối chuỗi.

4.2.4 Các hàm làm việc với chuỗi

Các hàm làm việc với chuỗi nằm trong tệp tin thư viện `<string.h>`

- **Hàm `strcpy(s1, s2)`:** Sao chép chuỗi `s2` vào `s1`. Kích thước tối đa của `s1` phải đủ lớn để chứa `s2`. Hàm này dùng để gán một chuỗi cho một chuỗi khác. Thông thường `s1` là một biến chuỗi, `s2` là một hằng chuỗi, hoặc một biến chuỗi đã có giá trị. Sau câu lệnh này toàn bộ nội dung của `s1` nếu có đều được thay thế bởi nội dung của `s2`.

Để gán giá trị chuỗi cho một biến chuỗi ta không sử dụng được lệnh gán thông thường như đối với các kiểu dữ liệu khác mà ta phải sử dụng hàm `strcpy(bienchuoi, "chuỗi cần gán")`.

Ví dụ: Để gán chuỗi "Nguyen Thi Nhu Quynh" cho biến `hovaten` ta phải làm như sau: `strcpy(hovaten,"Nguyen Thi Nhu Quynh");`

- **Hàm `strcmp(s1,s2)`:** So sánh hai chuỗi `s1` và `s2`. Hàm này trả về giá trị:

0 nếu `s1==s2`

>0 nếu `s1>s2`

<0 nếu `s1<s2`

Hai dãy ký tự được so sánh với nhau theo cách sau:

Lấy trên mỗi chuỗi một ký tự từ bên trái sang tạo thành một cặp, và so sánh mã ASCII của cặp ký tự đó. Quá trình so sánh kết thúc khi :

- Kết thúc khi gặp một cặp ký tự có mã ASCII khác nhau: Kết quả là dãy nào có ký tự có mã ASCII lớn hơn thì dãy đó lớn hơn, và ngược lại.

- Hoặc Kết thúc khi một trong hai chuỗi đã kết thúc: Chuỗi nào có số ký tự nhiều hơn thì chuỗi đó lớn hơn.

Ví dụ: `strcmp("ABCD","ABCE")>0` cho kết quả là chuỗi "ABCE" lớn hơn chuỗi "ABCD" vì mã ASCII của ký tự **E** lớn hơn của ký tự **D**.

Ví dụ: `strcmp("ABCD","ABCDEF")<0` phần đầu giống hệt nhau, dãy 1 kết thúc trước-> dãy 2 lớn hơn.

Hai chuỗi bằng nhau khi các ký tự của hai chuỗi giống hệt nhau (các ký tự giống nhau, số ký tự bằng nhau, thứ tự các ký tự như nhau). Ví dụ `strcmp("ABCD","ABCD")=0`.

Chú ý: `strcmp("ABCD","Abcd")` cho kết quả chuỗi 2 lớn hơn chuỗi 1 vì mã ASCII của ký tự **c** lớn hơn của ký tự **C**.

- **Hàm `strcmpi(s1,s2)`:** So sánh hai chuỗi ký tự nhưng không phân biệt chữ hoa chữ thường. Ví dụ: `strcmpi("ABCD","Abcd")=0`, chuỗi 1 bằng chuỗi 2.

- **Hàm `strlen(s)`:** Cho độ dài của chuỗi **s** không tính ký tự kết thúc chuỗi. Ví dụ: `strlen("Nguyen Van B")` cho giá trị là 12.

- **Hàm `strcat(s1,s2)`:** Ghép chuỗi **s2** vào cuối của chuỗi **s1**. Kích thước tối đa của chuỗi **s1** phải đủ lớn để chứa cả chuỗi **s2**.

Ví dụ 4.8: Viết chương trình nhập vào họ tên và năm sinh của một người. Máy tính sẽ cho biết người đó tuổi gì.

Gợi ý thuật giải: Có 10 can là Canh Tân Nhâm Quý Giáp Ất Bính Đinh Mậu Kỷ. Và có 12 chi là Tý Sửu Dần Mão Thìn Tỵ Ngọ Mùi Thân Dậu Tuất Hợi. Lấy năm sinh trừ 4 rồi chia cho 10 dư 0 thì là can Giáp, dư 1 là can Ất,..., dư 9 là can Quý. Lấy năm sinh trừ đi 4 rồi chia cho 12 dư 0 thì là chi Tý, dư 1 là chi Sửu,..., dư 11 là chi Hợi. Ví dụ một người sinh năm 1989 là tuổi Kỷ Tỵ (1989-4=1985 chia cho 10 dư 5 ứng với can Kỷ, 1985 chia cho 12 dư 5 ứng với chi Tỵ)

```
#include<iostream.h>
void main()
{
    int namsinh, sodu, i;
    char hovaten[25];
    char can[]={"GiapAt BinhDinhMau ky CanhTan NhamQuy "};
    char chi[]={"Ty  Suu Dan Mao ThìnTy  Ngo  Mùi  ThânDau
    TuấtHoi "};

    cout<<"Nhập vào họ và tên"; cin.get(hovaten,25);
    cout<<"Nhập vào năm sinh"; cin>>namsinh;
```

```
cout<<hovaten<<" sinh nam:"<<namsinh<<" tuoi:";
namsinh=namsinh-4;
sodu=(namsinh)%10;//Lay so du tinh can
i=sodu*4;
cout<<can[i]<<can[i+1]<<can[i+2]<<can[i+3];
sodu=(namsinh)%12;
i=sodu*4;
cout<<chi[i]<<chi[i+1]<<chi[i+2]<<chi[i+3];
}
```

Ví dụ 4.9: Xem Ví dụ 2.8 chương 2

4.2.5 Mảng dãy ký tự

Giả sử có họ tên của 50 sinh viên, mỗi họ tên có không quá 25 ký tự. Như vậy ta sẽ phải dùng mảng dãy ký tự như sau:

`char dayhoten[50][25];`

Cú pháp chung khai báo mảng dãy ký tự:

`char tenbienmang [sophantu][sokytu];`

Trong đó: sophantu là số phần tử tối đa có trong mảng. sokytu là số ký tự tối đa của một phần tử mảng.

Ví dụ 4.10: Viết chương trình nhập vào một danh sách họ tên của n sinh viên (n nhập vào từ bàn phím). Sau đó sắp xếp dãy này theo chiều tăng dần của họ tên (theo vần alphabet).

```
#include <iostream.h>
#include <string.h>
void main()
{ char dayhoten[50][25];
  char tg[25]; int i,j,n;
  cout<<"Nhập vào số sinh viên"; cin>>n; cin.get();
  for (i=0;i<n;i++)
  {
    cout<<"Nhập họ tên sinh viên "<<i<<" ";
    cin.get(dayhoten[i],25); cin.get();
  }
  for (i=0;i<n-1;i++)
  for (j=i+1;j<n;j++)
    if (strcmp(dayhoten[i],dayhoten[j])>0 )
    { strcpy(tg,dayhoten[i]);
      strcpy(dayhoten[i],dayhoten[j]);
      strcpy(dayhoten[j],tg);
    }
}
```

```
cout<<"\n Danh sach sinh vien da sap xep:\n";  
for (i=0;i<n; i++)  
    cout<<dayhoten[i]<<"\n";  
}
```

Kết quả chạy chương trình:

4.3 Bài tập

Bài 1: Viết chương trình nhập vào n số nguyên (với n nhập vào từ bàn phím). Tính trung bình cộng của các phần tử của mảng.

Bài 2:Viết chương trình nhập vào n số nguyên (với n nhập vào từ bàn phím). Tính trung bình cộng của các số lẻ, trung bình cộng của các số chẵn, rồi in kết quả ra màn hình.

Bài 3:Viết chương trình nhập vào một dãy các số nguyên. Đưa ra màn hình phần tử chẵn lớn nhất và phần tử lẻ nhỏ nhất của dãy.

Bài 4: Viết chương trình nhập vào n số nguyên. Nhập một giá trị cần tìm từ bàn phím. Tìm xem trong dãy đã nhập có phần tử này không nếu có thì cho biết vị trí của phần tử cần tìm.

Bài 5: Một cửa hàng bán n loại mặt hàng khác nhau. Biết đơn giá bán ra và đơn giá mua vào của mặt hàng thứ i là Ban[i], Mua[i]. Viết chương trình tính lợi nhuận thu được của cửa hàng.

Bài 6: Có n công nhân với hệ số lương của người thứ i là hs[i], lương cơ bản là 250.000đ. Viết chương trình tính thu nhập của người thứ i, và tính tổng số tiền mà công ty phải trả cho n công nhân.

Bài 7: Viết hàm thực hiện xóa phần tử thứ k của mảng n số nguyên.

Bài 8: Viết chương trình nhập vào mảng n số nguyên, sắp xếp mảng này theo chiều tăng dần. In mảng trước và sau khi sắp xếp. Yêu cầu: có xây dựng và sử dụng hàm nhapmang, inmang, nhapdl, sapxep, hoanvi.

CHƯƠNG 5 CON TRỎ TRONG C++

5.1 Khái niệm

Khi ta khai báo một biến, chương trình dịch sẽ dành cho biến đó một ô nhớ để lưu trữ giá trị của biến. Ô nhớ này được đánh một địa chỉ để máy tính quản lý. Khi ta sử dụng biến thì chương trình dịch sẽ truy cập tự động đến địa chỉ của ô nhớ chứa biến.

Một con trỏ là một biến mà nội dung của nó là địa chỉ của một biến khác hay một hàm. Nhờ có con trỏ mà ta có thể sử dụng các biến thông qua địa chỉ của biến, thay vì thông qua tên biến.

Trong chương này chúng ta đề cập đến các thao tác với con trỏ trong C++.

5.2 Khai báo con trỏ

Cú pháp: *KiểuDL* **tencontro*;

Trong đó *KiểuDL* là kiểu dữ liệu của biến mà con trỏ chứa địa chỉ

* là ký tự định kiểu con trỏ

tencontro là một tên mà người sử dụng đặt cho con trỏ

Sau câu lệnh khai báo máy tính tạo ra một con trỏ và gán giá trị NULL cho con trỏ này. NULL là một hằng địa chỉ thể hiện rằng con trỏ không trỏ vào đâu cả. Và máy tính giành ra ô nhớ có số byte bằng số byte của *KieuDL* cho biến con trỏ.

Ví dụ:

char *contro1; // Khai báo một con trỏ kiểu char, máy giành ra 1 byte bộ nhớ

float *contro2; // Khai báo một con trỏ kiểu float

int *contro3; // Khai báo một con trỏ kiểu int

5.3 Các thao tác trên con trỏ

5.3.1 Thao tác lấy địa chỉ

Như đã biết khi chúng ta khai báo một biến máy tính sẽ giành ra một ô nhớ cho biến đó. Ô nhớ này có một địa chỉ nhất định trong bộ nhớ, và đó cũng chính là địa chỉ của biến. Muốn lấy địa chỉ này ta sử dụng toán tử &.

Ví dụ: int x,y,*p; // Khai báo 2 biến nguyên x, y và một biến con trỏ p

p=&x; // gán địa chỉ của biến x cho con trỏ p (hay con trỏ p trỏ tới biến x)

Sau câu lệnh gán trên con trỏ p chứa địa chỉ của biến x.

5.3.2 Thao tác lấy nội dung

Muốn lấy nội dung của biến được trỏ bởi con trỏ ta sử dụng toán tử *

Ví dụ: int x=10, y, *p;

p=&x; // gán địa chỉ của biến x cho con trỏ p (hay con trỏ p trỏ tới biến x)

y=*p; // lấy nội dung của ô nhớ được con trỏ p trỏ tới và gán cho biến y

Mà ô nhớ được con trỏ p trỏ tới là ô nhớ giành cho biến x. Như vậy y nhận giá trị

Các cách viết sau là tương đương nhau: `x`, `*p`, `*&x` đều có tác dụng lấy nội dung của biến `x`.

5.4 Con trỏ và biến động

Các biến thuộc các kiểu dữ liệu mà chúng ta đã nghiên cứu là các biến tĩnh. Chúng được khai báo trong các câu lệnh khai báo biến và chúng tồn tại (chiếm vùng nhớ) trong toàn bộ quá trình hoạt động của chương trình chứa biến. Điều này có thể dẫn đến việc lãng phí bộ nhớ khi chúng ta chưa sử dụng đến biến hoặc đã sử dụng xong biến này rồi.

Để khắc phục nhược điểm trên C/C++ cung cấp một loại biến là *biến động*. Các biến loại này có thể được tạo ra và xóa đi trong quá trình thực hiện chương trình, tùy theo yêu cầu của người lập trình.

Biến động không có tên, nó được tạo ra bằng câu lệnh **new**, xóa đi bằng câu lệnh **delete** của C++. Và nó được truy cập thông qua con trỏ.

Cú pháp: Tạo biến động `tencontro=new kiểu;`

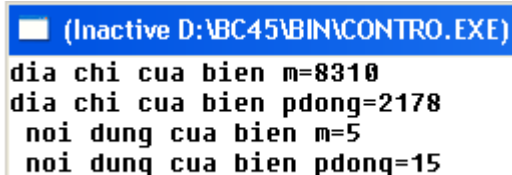
Xóa biến động **delete** `tencontro`

Theo câu lệnh 1 máy tính tạo ra một vùng nhớ có kích thước bằng kích thước của `kiểu`. Địa chỉ của vùng nhớ này được chứa trong con trỏ `tencontro`.

Ví dụ 5.1:

```
#include <iostream.h>
#include <stdio.h>
void main()
{
    int m=5, *pm, *pdong; // Khai bao hai con tro pm va pdong
    pm = &m; // con tro pm tro toi vung nho cua bien m
    pdong = new int; // tạo ra một vùng nhớ (một biến động)
    do con tro pdong tro toi
        *pdong = 15; // gán giá trị 15 cho biến động được tro
    tới bởi con tro pdong
    printf("địa chỉ của biến m=%d\n", pm);
    printf("địa chỉ của biến pdong=%d\n", pdong);
    printf("nội dung của biến m=%d\n", *pm);
    printf("nội dung của biến pdong=%d\n", *pdong);
    delete pdong;
}
```

Kết quả chạy chương trình:



```
(Inactive D:\BC45\BIN\CONTRO.EXE)
địa chỉ của biến m=8310
địa chỉ của biến pdong=2178
nội dung của biến m=5
nội dung của biến pdong=15
```

5.5 Con trỏ và mảng một chiều

Giữa con trỏ và mảng có quan hệ chặt chẽ với nhau. Các thao tác mà ta thực hiện trên mảng nhờ sử dụng tên biến mảng và chỉ số đều có thể được thực hiện thông qua con trỏ.

Trước tiên ta xem xét việc sử dụng con trỏ và mảng 1 chiều kiểu nguyên.

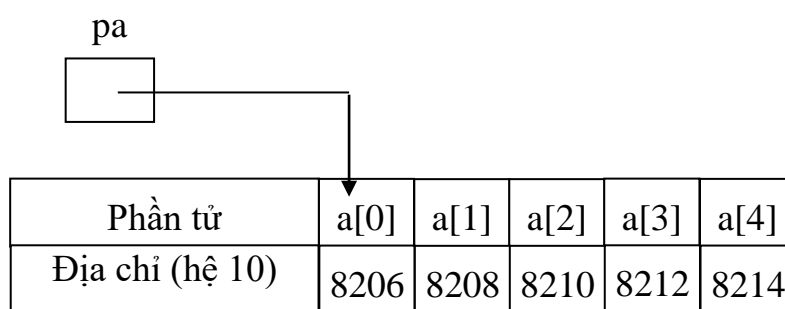
Ví dụ: Ta có các câu lệnh như sau:

```
int a[5]; // Khai báo mảng a có 5 phần tử
int *pa; // Khai báo con trỏ pa trỏ tới một biến kiểu int
pa=a; // gán địa chỉ của mảng a cho biến con trỏ pa
```

Gặp câu lệnh thứ nhất, máy tính sẽ giành ra một vùng nhớ gồm 5 ô nhớ liên tiếp nhau. Và tên mảng a chính là địa chỉ của ô nhớ đầu tiên trong vùng nhớ này (địa chỉ của mỗi phần tử hơn kém nhau 2 đơn vị vì mỗi phần tử kiểu int có kích thước 2 byte).

Câu lệnh thứ 2: Khai báo một biến con trỏ pa kiểu nguyên.

Câu lệnh thứ 3: Gán địa chỉ của mảng a cho biến pa. Lúc này pa trỏ vào phần tử đầu tiên trong mảng a.



Lúc này ta có thể truy cập và thao tác trên các phần tử của mảng a thông qua con trỏ pa. Cụ thể :

Viết pa+i: Cho địa chỉ của phần tử a[i]

*(pa+i): Cho giá trị của phần tử a[i]

Ví dụ 5.2: Chương trình minh họa:

```
#include <iostream.h>
#include <stdio.h>
void main()
{ int i,a[5]={1,12,5,7,11};
  int *pa; pa=a;
  for(i=0;i<5;i++) printf("a[%d]=%d\n ",i,a[i]);
  for(i=0;i<5;i++) printf("(pa+%d)=%d\n ",i,*(pa+i));
  for(i=0;i<5;i++) printf("địa chỉ của a[%d]=%d\n ",i,(pa+i));
}
```

Kết quả thực hiện chương trình:

```
a[0]=1
a[1]=12
a[2]=5
a[3]=7
a[4]=11
*(pa+0)=1
*(pa+1)=12
*(pa+2)=5
*(pa+3)=7
*(pa+4)=11
địa chỉ của a[0]=8256
địa chỉ của a[1]=8258
địa chỉ của a[2]=8260
địa chỉ của a[3]=8262
địa chỉ của a[4]=8264
```

Trường hợp sử dụng con trỏ với mảng một chiều kiểu thực cũng hoàn toàn tương tự như mảng một chiều kiểu nguyên.

5.6 Con trỏ và mảng hai chiều

Ví dụ: Ta có khai báo sau:

```
int a[3][5], *pa;
```

Để gán địa chỉ của mảng `a` cho con trỏ `pa` ta viết câu lệnh gán như sau:

```
pa=&a[0][0];
```

Để truy nhập vào phần tử `a[i][j]` (hàng `i` cột `j`) ta sử dụng công thức sau:

```
pa=&a[0][0]+(i*m)+j;
```

Trong đó `m` là số cột của ma trận (tức là số phần tử trên mỗi dòng)

Ví dụ 5.3: Chương trình minh họa:

```
#include<iostream.h>
void main()
{
    const n=4,m=3;
    int i,j, a[n][m], *pa;
    cout<<"Nhập các phần tử của mảng:\n";
    for (i=0;i<n;i++)
        for (j=0;j<m;j++)
        {
            cout<<"a["<<i<<","<<j<<"]="; cin>>a[i][j];
        }
    cout<<"Mảng vừa nhập là\n";
    for (i=0;i<n;++i)
    { for (j=0;j<m;++j)
        {
            pa=&a[0][0] +i*m+j;
            cout<<*pa<<" ";
        }
        cout<<"\n";
    }
}
```

Kết quả thực hiện chương trình

Nhap cac phan tu cua mang:

a[0,0]=1

a[0,1]=2

a[0,2]=4

a[1,0]=5

a[1,1]=6

a[1,2]=7

a[2,0]=8

a[2,1]=9

a[2,2]=1

a[3,0]=2

a[3,1]=3

a[3,2]=4

Mang vua nhap la

1 2 4

5 6 7

8 9 1

2 3 4

CHƯƠNG 6: DỮ LIỆU KIỂU CẤU TRÚC

6.1 Khái niệm dữ liệu kiểu cấu trúc

Trong chương 4 chúng ta đã nghiên cứu dữ liệu kiểu mảng. Mảng là tập hợp các phần tử có cùng kiểu. Dữ liệu kiểu cấu trúc cũng là tập hợp các phần tử. Tuy nhiên mỗi phần tử có thể có kiểu dữ liệu khác nhau. Mỗi phần tử của kiểu cấu trúc được gọi là một trường (field). Dữ liệu kiểu cấu trúc trong C/C++ cũng tương tự như dữ liệu kiểu bản ghi (RECORD) trong Pascal.

6.2 Khai báo dữ liệu kiểu cấu trúc

- Định nghĩa kiểu dữ liệu cấu trúc với các thành phần dữ liệu

```
struct ten_cau_truc
{
    Kieu1 ten_truong1;
    Kieu2 ten_truong2;
    .....
    Kieun ten_truongn;
};
```

Trong đó:

struct là từ khóa để khai báo kiểu cấu trúc

ten_cau_truc là tên của kiểu cấu trúc do người lập trình tự đặt theo quy tắc đặt tên biến.

Kieu1, Kieu2, ..., Kieun là các kiểu dữ liệu của các trường tương ứng. Các kiểu này có thể là các kiểu dữ liệu cơ bản như char, int, float, ... hay những kiểu dữ liệu có cấu trúc khác như kiểu mảng, chuỗi, hay chính là kiểu cấu trúc.

Ví dụ: Khai báo một kiểu cấu trúc **ngaythang**, và một cấu trúc **sinhvien** như sau:

```
struct ngaythang
{
    int ngay;//ngay kieu nguyen
    char thang[10];//thang kieu xau ky tu co toi da 10 ky tu
    int nam;//nam kieu nguyen
};

struct sinhvien
{
    char hodem[20],ten[10];
    ngaythang ngaysinh;//ngaysinh kieu ngaythang
    float diemtb; //diemtb kieu so thuc
};
```

- Khai báo kiểu cấu trúc:

Sau khi đã định nghĩa được kiểu cấu trúc thì ta cần khai báo các biến có kiểu cấu trúc đã xây dựng:

Ví dụ: **sinhvien** sv1,sv2;//Khai báo hai biến sv1,sv2 thuộc kiểu cấu trúc **sinhvien**

Ngoài ra chúng ta có thể kết hợp vừa xây dựng kiểu cấu trúc vừa khai báo biến như sau:

```
struct ten_cau_truc
{
    Kieu1 ten_truong1;
    Kieu2 ten_truong2;
    .....
    Kieun ten_truongn;
}bien1,bien2,...,bien_n;
```

Ví dụ:

```
struct sinhvien
{
    char hodem[20],ten[10];
    ngaythang ngaysinh;//ngaysinh kieu ngaythang
    float diemtb; //diemtb kieu so thuc
} sv1,sv2;
```

6.3 Truy nhập vào các trường của cấu trúc

Cú pháp:

tenbiencautruc.tentruong

Ví dụ: cin.get(sv1.hodem,20);
sv1.diemtb=6.5;

Nếu một trường trong cấu trúc có kiểu cấu trúc thì ta phải truy nhập đến từng trường con của trường đó.

Ví dụ: sv1.ngaysinh.ngay=18; sv1.ngaysinh.nam=1978;

Ví dụ 6.1: Chương trình minh họa: Nhập vào thông tin của hai sinh viên và in ra màn hình danh sách hai sinh viên vừa nhập:

```
#include<iostream.h>
void main()
{
    struct ngaythang
    {
        int ngay,thang,nam;
    };
    struct sinhvien
```

Chương 6 Dữ liệu cấu trúc

```
{  char hodem[20];
    char ten[10];
    float diemtb;
    ngaythang ngaysinh;
}sv1,sv2;
cout<<"Nhap TT sinh vien thu nhat\n";
cout<<"Ho dem: ";cin.get(sv1.hodem,20);cin.get();
cout<<"Ten: " ;cin.get(sv1.ten,10);
cout<<"diem trung binh:";cin>>sv1.diemtb;
cout<<"nhap ngay sinh:";cin>>sv1.ngaysinh.ngay;
cout<<"nhap thang sinh:";cin>>sv1.ngaysinh.thang;
cout<<"nhap nam sinh:";cin>>sv1.ngaysinh.nam;
cout<<"\nNhap TT sinh vien thu hai\n"; cin.get();
cout<<"Ho dem: ";cin.get(sv2.hodem,20);cin.get();
cout<<"Ten: " ;cin.get(sv2.ten,10);
cout<<"diem trung binh:";cin>>sv2.diemtb;
cout<<"nhap ngay sinh:";cin>>sv2.ngaysinh.ngay;
cout<<"nhap thang sinh:";cin>>sv2.ngaysinh.thang;
cout<<"nhap nam sinh:";cin>>sv2.ngaysinh.nam;
cout<<"\n\n Danh sach sinh vien vua nhap \n\n";
cout<<"Ho va ten \t\t"<<"Ngay sinh\t"<<"Diem TB  \n";
cout<<sv1.hodem<<"\t"<<sv1.ten<<"\t"<<sv1.ngaysinh.ngay<<"/";
cout<<sv1.ngaysinh.thang<<"/"<<sv1.ngaysinh.nam<<"\t";
cout<<sv1.diemtb<<endl;
cout<<sv2.hodem<<"\t"<<sv2.ten<<"\t"<<sv2.ngaysinh.ngay<<"/";
cout<<sv2.ngaysinh.thang<<"/"<<sv2.ngaysinh.nam<<"\t"<<sv2.diemtb
}
```

Kết quả thực hiện chương trình:

```
Nhap TT sinh vien thu nhat
Ho dem: Nguyen Thi Nhu
Ten: Quynh
diem trung binh:8.5
nhap ngay sinh:18
nhap thang sinh:12
nhap nam sinh:1978
```

```
Nhap TT sinh vien thu hai
Ho dem: Nguyen Thi Hai
Ten: Yen
diem trung binh:8.8
nhap ngay sinh:9
nhap thang sinh:9
nhap nam sinh:1978
```

Danh sach sinh vien vua nhap

Ho va ten	Ngay sinh	Diem TB
Nguyen Thi Nhu Quynh	18/12/1978	8.5
Nguyen Thi Hai Yen	9/9/1978	8.8

6.4 Mảng cấu trúc

Theo dõi chương trình minh họa ở ví dụ 6.1 ta thấy đoạn chương trình nhập dữ liệu và in dữ liệu cho hai sinh viên phải viết lặp lại nên rất tốn công sức viết chương trình. Để khắc phục điều này ta sử dụng mảng cấu trúc.

Khai báo mảng cấu trúc như sau:

ten_kieu_cau_truc tenmang[sophantu];

Ví dụ: `sinhvien sv[2];`//Khai báo mảng sv gồm 2 phần tử

Ví dụ 6.2: Sửa lại ví dụ 6.1 như sau:

```
#include<iostream.h>
void main()
{ int i;
  struct ngaythang
  {    int ngay,thang,nam;
  };
  struct sinhvien
  {    char hodem[20];
char ten[10];
float diemtb;
ngaythang ngaysinh;
  };
  sinhvien sv[10];//Khai báo mảng sinh viên tối đa 10 sv
  int n=2;// số sv = 2
  for (i=0;i<n;i++)
  { cout<<"Nhập TT sinh viên thứ "<<i+1<<" \n";cin.get();
    cout<<"Họ tên: ";cin.get(sv[i].hodem,20);cin.get();
    cout<<"Tên: " ;cin.get(sv[i].ten,10);
    cout<<"điểm trung bình:";cin>>sv[i].diemtb;
    cout<<"nhập ngày sinh:";cin>>sv[i].ngaysinh.ngay;
    cout<<"nhập tháng sinh:";cin>>sv[i].ngaysinh.thang;
    cout<<"nhập năm sinh:";cin>>sv[i].ngaysinh.nam;
  }

  cout<<"\n\n Danh sách sinh viên vừa nhập \n\n";
  cout<<"Họ và tên \t\t"<<"Ngày sinh\t"<<"Điểm TB  \n";
  for (i=0;i<n;i++)
  {cout<<sv[i].hodem<<"\t"<<sv[i].ten<<"\t";
  cout<<sv[i].ngaysinh.ngay<<"/"<<cout<<sv[i].ngaysinh.thang;
  cout<<"/"<<sv[i].ngaysinh.nam<<"\t"<<sv[i].diemtb<<endl;
  }
```

6.5 Con trỏ và địa chỉ cấu trúc

Một con trỏ cấu trúc cũng giống như các con trỏ của các kiểu dữ liệu khác. Con trỏ cấu trúc chứa địa chỉ của một biến cấu trúc hoặc một vùng nhớ có kiểu cấu trúc nào đó.

Khai báo con trỏ cấu trúc

tenkieucautruc *tencontro;

Ví dụ 6.3:

```
struct sinhvien
{
    char maso[10], hodem[20], ten[10];
    float dtb;
};
sinhvien sv, sv1, *psv;
cout<<"Nhap m.so sinh vien sv"; cin.get();
cin.get(sv.maso, 10);
cout<<"Nhap ho dem cua sinh vien sv";
cin.get(); cin.get(sv.hodem, 10);
cout<<"Nhap ten cua sinh vien sv";
cin.get(); cin.get(sv.ten, 10);
psv=&sv; //con tro psv tro toi bien cau truc sv
psv->dtb=5.5; //gan gia tri 5.5 cho thuoc tinh dtb cua
sv
sv1=*psv; //gan gia tri cua sv cho sv1
(*psv).dtb=6.0;
```

6.6 Cấu trúc tự trỏ và danh sách liên kết

6.6.1 Cấu trúc tự trỏ

Trong phần trước ta đã học cách sử dụng biến mảng để lưu trữ một danh sách các phần tử thuộc cùng một kiểu, ví dụ danh sách các sinh viên, danh sách các cán bộ,... Tuy nhiên nhược điểm của kiểu dữ liệu này là ta phải chỉ rõ số phần tử tối đa của mảng trong phần khai báo, điều này dẫn đến hai khả năng. Thứ nhất là lãng phí bộ nhớ do số phần tử thực có trong mảng nhỏ hơn số phần tử tối đa đã khai báo. Thứ hai là thiếu bộ nhớ do số phần tử thực có lớn hơn số phần tử tối đa đã khai báo. Để khắc phục nhược điểm này C++ cho phép sử dụng cơ chế cấp phát bộ nhớ động. Khi này ta không cần khai báo trước số phần tử tối đa nữa mà sử dụng đến đâu ta xin cấp phát bộ nhớ đến đó.

Tuy nhiên mỗi khi gặp lệnh xin cấp phát bộ nhớ cho phần tử mới trong danh sách thì máy tính sẽ tìm ra một vùng nhớ còn trống bất kỳ để cấp phát. Điều này dẫn đến tình trạng các phần tử trong cùng một danh sách nằm rải rác trong bộ nhớ (không nằm kề nhau như ở dữ liệu kiểu mảng). Và để có thể “liên lạc” được với nhau thì mỗi phần tử trong danh sách phải chứa địa chỉ của phần tử tiếp theo hoặc chứa cả địa chỉ của phần tử

Chương 6 Dữ liệu cấu trúc

trước nó nữa. Như vậy mỗi phần tử trong danh sách phải có hai phần: phần dữ liệu và phần con trỏ. Mỗi phần tử trong danh sách được gọi là một cấu trúc tự trỏ, nghĩa là trong mỗi phần tử đều chứa con trỏ trỏ đến chính phần tử thuộc kiểu của phần tử đó.

Khai báo cấu trúc tự trỏ:

```
struct tencautruc
{
    Các thành phần dữ liệu;
    tencautruc *contro;
};
```

Ví dụ:

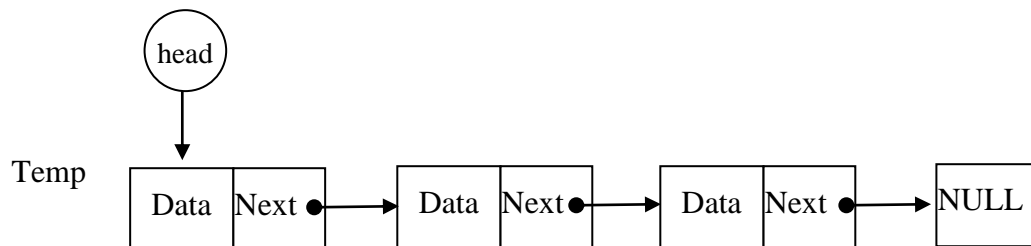
```
struct sinhvien
{
    //cac thanh phan du lieu
    char hodem[20],ten[10];
    float dtb;
    //khai bao con tro
    sinhvien *next;
};
```

6.6.2 Danh sách liên kết

Danh sách liên kết là một cấu trúc dữ liệu cho phép thể hiện và quản lý danh sách bằng các cấu trúc liên kết với nhau thông qua con trỏ.

Danh sách liên kết đơn: Mỗi phần tử trong danh sách chứa một con trỏ trỏ tới phần tử tiếp theo. Phần tử đầu tiên trong danh sách được trỏ bởi con trỏ head, phần tử cuối cùng trong danh sách trỏ vào hằng con trỏ NULL.

Hình vẽ sau sẽ minh họa một danh sách liên kết đơn:



Các phép toán trên danh sách liên kết:

- Tạo phần tử mới và thêm nó vào danh sách:

- + Xin cấp phát bộ nhớ cho phần tử mới
- + Đưa giá trị vào các trường dữ liệu của phần tử mới
- + Tạm thời gán con trỏ next của phần tử mới là NULL (phần tử mới chưa trỏ vào đâu cả).

+ Gắn phần tử mới vào danh sách (ở đây ta chọn cách gắn phần tử mới vào cuối danh sách).

Ví dụ 6.4: Chương trình con thực hiện việc tạo một phần tử mới:

```
sinhvien *taomoi()
{
    sinhvien *tamsv=new sinhvien;
    cout<<"nhap ma so";cin.get();cin.get(tamsv->maso,10);
    cout<<"Nhap hodem ";cin.get();cin.get(tamsv->hodem,20);
    cout<<"Nhap ten ";cin.get();cin.get(tamsv->ten,10);
    cout<<"Nhap diem trung binh ";cin>>tamsv->dtb;
    tamsv->next=NULL;
    return tamsv;
}
```

//chương trình con thực hiện việc thêm một phần tử mới vào cuối danh sách:

```
void themcuoi ()
{
    sinhvien *tam;
    tam=taomoi();
    if (sosv==0) head=tail=tam; //neu ds chua co sv nao
    else tail->next=tam; tail=tam;
}
```

- Xóa một phần tử thứ k khỏi danh sách

+ Nếu danh sách có một phần tử, và k=1, sau khi xóa đưa danh sách về trạng thái rỗng (nghĩa là con trỏ đầu và cuối đều trở vào NULL head=tail=NULL).

+ Nếu phần tử cần xóa đứng đầu danh sách (n=1)

```
void xoa(int k)
{
    sinhvien *tam;
    if (sosv==1 && k==1)
        {tam=head; head=tail=NULL;delete tam;sosv--;}
    else if (k==1)// xoa phan tu dau
        {tam=head; head=head->next;delete tam;sosv--;}
    else { sinhvien *i,*j; int t;
        i=head;
        for (t=0;t<k-1;t++) i=i->next;
        tam=i->next; j=tam->next;i->next=j;
        delete tam;
    }
}
```

6.7 Bài tập

Bài 1: Viết chương trình nhập vào một danh sách sinh viên, mỗi sinh viên gồm các thông tin masv, hodem, ten, dt,dv. Yêu cầu lập danh sách sinh viên thi lại toán, danh sách sinh viên thi lại văn ($dt < 4$, $dv < 4$). Yêu cầu chương trình có sử dụng chương trình con nhapdl, indl.

Bài 2: Viết chương trình nhập vào một danh sách cán bộ, mỗi cán bộ gồm các thông tin macb, hodem, ten, hsluong, dantoc, socon. Yêu cầu đưa ra màn hình danh sách cán bộ người dân tộc thiểu số mà có $socon < 2$. Tính lương cho mỗi cán bộ theo $ct\ luong = hsluong * 512$. Tìm giá trị lương cao nhất và lập danh sách các cán bộ có lương cao nhất. Chương trình có sử dụng chương trình con nhapdl, indl.

Bài 3: Nhập danh sách sinh viên, mỗi sinh viên gồm các thông tin masv, hodem, ten, dtb. Yêu cầu sắp xếp danh sách sinh viên theo dtb giảm dần. In danh sách trước và sau khi sắp. Chương trình có sử dụng chương trình con nhapdl, sapxep, indl.

Bài 4: Nhập danh sách n mặt hàng, mỗi mặt hàng gồm các thông tin mahang, tenhang, sl, dongia. Tính thành tiền cho mỗi mặt hàng và tổng thành tiền của các mặt hàng.

CHƯƠNG 7 DỮ LIỆU KIỂU TỆP TRONG C++

7.1 Khái niệm về tệp

Tệp là một tập hợp các dữ liệu có cùng kiểu được nhóm lại với nhau, được ghi lên đĩa từ hoặc các thiết bị nhớ ngoài của máy tính để sử dụng lại sau này.

Dữ liệu kiểu tệp giống dữ liệu kiểu mảng ở chỗ các phần tử của tệp có cùng kiểu dữ liệu. Tuy nhiên tệp và mảng có những điểm khác nhau như sau:

- Tệp thì được lưu trữ trên bộ nhớ ngoài, và tồn tại ở đó đến khi ta xóa tệp, mất điện tệp cũng không bị mất. Còn mảng được lưu trữ tạm thời trong bộ nhớ RAM. Khi tắt máy hoặc mất điện thì mảng sẽ bị xóa.
- Số phần tử của mảng được xác định khi khai báo mảng, còn số phần tử của tệp không được xác định trước.
- Việc truy nhập vào các phần tử của mảng được thực hiện bằng việc sử dụng tên mảng kèm theo chỉ số, hoặc con trỏ. Còn việc truy cập vào các phần tử của tệp thì được thực hiện thông qua con trỏ tệp.

7.2 Phân loại tệp

7.2.1 Phân loại tệp theo bản chất dữ liệu của tệp

- **Tệp văn bản (text file):** Là loại tệp tin mà các phần tử của nó là các ký tự (chữ cái, chữ số, hoặc ký tự đặc biệt không kể các ký tự điều khiển). Các phần tử này được tổ chức thành từng dòng, mỗi dòng có dấu kết thúc dòng là CR ('\r') và LF ('\n').

Khi đọc dữ liệu từ tệp thì hai ký tự CR và LF sẽ được chuyển thành một ký tự LF. Khi ghi dữ liệu lên tệp thì một ký tự LF được chuyển thành hai ký tự CR và LF.

Ký tự kết thúc tệp (EOF-End Of File), được xác định bởi tổ hợp phím Ctrl+Z, ký hiệu là ^Z có mã ASCII là 26. Hàm eof() sẽ cho giá trị là 1 nếu gặp mã kết thúc tệp.

Ví dụ: Ta có văn bản như sau:

Day la van ban cua toi

Day so 12345678

Ket thuc

Văn bản trên được tổ chức trong tệp văn bản dưới dạng như sau:

Day la van ban cua toi	CR LF	Day so 12345678	CR LF	Ket thuc	EOF
------------------------	-------	-----------------	-------	----------	-----

Vì tệp tin văn bản được lưu trữ dưới dạng mã ASCII nên ta có thể đọc được nội dung của tệp bằng lệnh Type của DOS hay bằng các chương trình xử lý văn bản khác.

- **Tệp nhị phân (binary file):** Tệp tin này dùng để ghi lên đĩa những cấu trúc dưới dạng nhị phân. Mỗi lần đọc, hay ghi là đọc hay ghi một cấu trúc. Mã kết thúc tệp nhị phân là -1.

7.2.1 Phân loại tệp theo cách truy nhập tệp

- **Tệp truy nhập tuần tự:** Tệp truy nhập tuần tự có đặc điểm là để truy nhập một phần tử của tệp phải lần lượt duyệt qua các phần tử đứng trước phần tử này.

- **Tệp truy nhập ngẫu nhiên:** Tệp truy nhập ngẫu nhiên cho phép di chuyển con trỏ vị trí tệp đến bất kỳ phần tử nào thuộc tệp, để truy nhập phần tử này.

7.3 Các bước xử lý tệp

Quá trình xử lý tệp thường gồm các bước sau đây:

- Khai báo một luồng nhập-xuất (khai báo biến tệp)
- Mở tệp
- Xử lý tệp: Gồm các thao tác như đọc-ghi dữ liệu và các thao tác khác đối với tệp.
- Đóng tệp

7.3.1 Khai báo luồng nhập/xuất

Từ phần này về sau chúng ta quy ước sử dụng từ “nhập” để chỉ thao tác đọc dữ liệu từ tệp vào bộ nhớ. Và từ “xuất” để chỉ thao tác ghi dữ liệu vào tệp.

- Khai báo một luồng dùng để xuất dữ liệu:

ofstream tenluong;

- Khai báo một luồng dùng để nhập dữ liệu:

ifstream tenluong; - Khai báo một luồng dùng để cả nhập và xuất dữ liệu:

fstream tenluong;

tenluong: Là tên của luồng dữ liệu do người lập trình tự đặt, tên này sẽ gắn với tên của một tệp trên đĩa khi mở tệp, và nó được sử dụng trong các lệnh làm việc với tệp.

7.3.2 Mở tệp

Cú pháp:

tenluong.open(“tentep”, kieuuxuly, capbaove);

Trong đó:

tenluong là tên luồng nhập dữ liệu hay luồng xuất dữ liệu đã được khai báo. Tên này sẽ gắn với tên tệp (**tentep**) trên đĩa từ.

open(): Là hàm dùng để mở tệp, hàm này được dùng cho cả tệp văn bản lẫn tệp nhị phân.

tentep: Là tên của tệp cần mở. Tên này phải phù hợp với nguyên tắc đặt tên của hệ điều hành DOS. Thay vì đưa vào tên của tệp ta có thể đưa vào cả đường dẫn và cả tên tệp nếu cần.

kieuuxuly: Kiểu xử lý là một dãy ký tự xác định kiểu tệp (tệp văn bản hay tệp nhị phân), và những thao tác như đọc, ghi,... cần thực hiện với tệp này. **Kieuuxuly** có thể được kết hợp từ một số giá trị sau đây:

Kiểu xử lý	Ý nghĩa
ios::binary	Mở tệp nhị phân. Nếu không có tham số này thì mặc định là mở tệp văn bản
ios::in	Mở tệp đã có trên đĩa để đọc dữ liệu từ tệp này vào bộ nhớ
ios::out	Mở tệp để ghi dữ liệu vào tệp. Nếu tệp đã tồn tại trên đĩa thì nó bị xóa đi và thay thế bởi một tệp mới.
ios::trunc	Mở tệp đã có trên đĩa và xóa các dữ liệu đã có trong tệp để ghi DL mới
ios::app	Mở tệp đã có trên đĩa và ghi dữ liệu bổ xung vào cuối tệp. Nếu tệp chưa tồn tại thì một tệp mới được tạo ra.
ios::ate	Mở một tệp và chuyển con trỏ xuống cuối tệp
ios::nocreat	Mở một tệp đã có trên đĩa, nếu không có tệp thì thông báo lỗi, không tạo ra tệp mới.
ios::noreplace	Nếu đã có tệp thì thông báo lỗi, không thay tệp đó bằng tệp mới

Tham số này có thể vắng mặt trong câu lệnh open, khi đó máy tính sẽ hiểu theo chế độ mặc định.

capbaove Tham số này quy định cấp bảo vệ của luồng nhập/xuất. Tham số này có thể bỏ qua vì nó đã được gán với một giá trị mặc định.

7.3.3 Xử lý tệp

Xử lý tệp thường gồm các công việc đọc, ghi dữ liệu và một số công việc khác. Tùy theo yêu cầu công việc cụ thể, và loại tệp được xử lý, không có mẫu cố định cho công việc này.

Trong quá trình xử lý tệp thường phải kiểm tra trạng thái kết thúc tệp, hay các lỗi có thể xảy ra trong quá trình mở/ đóng hay đọc ghi dữ liệu tệp. Sau đây là một số khả năng lỗi có thể xảy ra

- Mở tệp đọc nhưng tệp không tồn tại
- Đọc dữ liệu nhưng con trỏ tệp đã ở cuối tệp
- Ghi dữ liệu lên tệp nhưng đĩa đã đầy
- Tạo tệp mới nhưng đĩa hỏng, đĩa đầy hoặc đĩa có bảo vệ cấm ghi
- Tên tệp không hợp lệ
- Thao tác định thực hiện không hợp lệ với tệp

Một số hàm dùng để kiểm tra quá trình xử lý tệp:

Hàm eof(): Trả về giá trị 1 (true) nếu máy đọc được mã kết thúc tệp. Ngược lại hàm trả về giá trị 0 (false).

Hàm fail(): Trả về giá trị 1 (true) nếu lần nhập/xuất cuối cùng có lỗi, ngược lại trả về giá trị 0 (false).

Hàm bad(): Trả về giá trị 1 (true) nếu thao tác đọc/ghi không hợp lệ hoặc có lỗi mà chưa phát hiện ra, ngược lại hàm trả về giá trị (false).

Hàm good(): Trả về giá trị 1 (true) nếu không có lỗi xảy ra, ngược lại trả về giá trị false.

Hàm clear(): Là hàm không kiểu. Hàm này dùng để xóa tất cả các bit lỗi.

7.3.4 Đóng tệp

Sau khi thực hiện các công việc trên tệp cần phải đóng tệp.

Cú pháp:

```
tenluong.close();
```

Trong đó **tenluong** là tên luồng nhập/xuất dữ liệu được gán với tệp. Hàm được dùng cho cả tệp văn bản lẫn tệp nhị phân.

7.4 Làm việc với tệp văn bản

Trong phần trước chúng ta đã sử dụng toán tử nhập >>, và toán tử xuất << để nhập và xuất dữ liệu từ các luồng nhập xuất chuẩn (là bàn phím và màn hình).

Ngoài ra toán tử xuất << cũng được dùng để xuất các kiểu dữ liệu khác nhau ra các tệp văn bản. Toán tử nhập >> có thể được dùng để nhập dữ liệu thuộc các kiểu dữ liệu khác nhau từ tệp văn bản vào các biến. Bên cạnh đó C++ còn cho phép sử dụng các hàm để nhập/ xuất dữ liệu đối với các tệp văn bản.

7.4.1 Xuất dữ liệu ra tệp văn bản (ghi tệp)

- Sử dụng toán tử << theo cú pháp:

```
tenluong<<tenbien;
```

Trong đó **tenluong** là tên của luồng xuất đã được khai báo và gán với tệp cần làm việc.

Ví dụ 7.1: Chương trình sau đây làm việc nhập dữ liệu từ bàn phím vào các biến và ghi dữ liệu từ các biến đó vào tệp văn bản.

```
#include <iostream.h>
#include <fstream.h>
int main()
{
    int namsinh; char ten[15];
    ofstream tep1;//Khai bao luong de xuat du lieu
    tep1.open("c:\\tep1.txt"); //mo tep text1.txt de ghi dl
    if (!tep1)
    {
        cout << "khong mo dc tep";return 1; }
        cout<<"Nhap du lieu tu ban phim\n";
        cout<<"nhap mot so nguyen chi nam sinh ";
        cin>>namsinh;
        cout<<"\n nhap Ten";
```

```
cin>>ten;
//Ghi du lieu vao tep
tep1<<ten; tep1<<' ';
tep1<<namsinh; tep1.close();
cout<<"qua trinh ghi tep ket thuc:";
}
```

Chương trình này chỉ cho phép nhập chuỗi tên (không chứa dấu cách), và một số nguyên chỉ năm sinh. Muốn nhập chuỗi tên có dấu cách thì ta có thể thay lệnh `cin>>ten` bằng hai câu lệnh **`cin.ignore(); cin.get(ten, 15);`** (theo cách nhập vào một chuỗi ký tự có chứa dấu cách ở chương 4 đã đề cập).

Ví dụ 7.2 sau đây thực hiện nhập từng dòng dữ liệu từ bàn phím và ghi vào tệp theo dòng. Sử dụng hàm `getline()` để nhập từng dòng từ bàn phím.

```
#include <iostream.h>
#include <fstream.h>
int main()
{
    char traloi, dong[255];
    ofstream tep1;//Khai bao luong de xuat du lieu
    tep1.open("c:\\tep2.txt"); //mo tep text1.txt de ghi dl
    if (!tep1)
    {
        cout << "khong mo dc tep";
        return 1;
    }
    cout<<"Nhap du lieu tu ban phim";
    do {
        cin.getline(dong, 255); tep1<<dong;
        cout<<"Ban co muon tt khong (C/K)";
        cin>>traloi;cin.ignore();
    } while (traloi=='c');
    tep1.close();
    cout<<"qua trinh ghi tep ket thuc:";
}
```

7.4.2 Nhập dữ liệu từ tệp văn bản (đọc tệp)

Ta thường sử dụng dòng vào `cin` và toán tử `>>` để lấy dữ liệu từ dòng vào chuẩn (bàn phím) đưa vào một biến.

Để lấy dữ liệu từ tệp văn bản đưa vào một biến ta sử dụng toán tử `>>` theo cú pháp **`tenluong>>tenbien;`** hoặc sử dụng hàm **`tenluong.get()`** hay **`tenluong.getline()`**. Trong đó **`tenluong`** là tên luồng nhập đã được khai báo và được gắn với tệp cần mở.

Ví dụ 7.3: Chương trình đọc dữ liệu từ `tep1.txt` đã được ghi ở ví dụ 7.1 như sau:

```
#include <iostream.h>
#include <fstream.h>
int main()
{
    int namsinh;  char ten[15];
    ifstream tep1;//Khai bao luong de nhap du lieu
    tep1.open("c:\\tep1.txt"); //mo tep text1.txt de doc dl
    if (!tep1)
    { cout << "khong mo dc tep"; return 1;}

    cout<<" Nhap du lieu tu tep";
    tep1>>ten;//doc du lieu tu tep vao bien ten
    tep1>>namsinh;//doc du lieu tu tep vao bien namsinh

    cout<<"Ten " <<ten<<"\n";//in nd bien ten ra man hinh
    cout<<"nam sinh:"<<namsinh;//in nd bien namsinh ra man hinh
    tep1.close();
    cout<<"\nqua trinh doc tep ket thuc:";
}
```

Nếu chương trình trong ví dụ 7.1 sử dụng lệnh **cin.ignore(); cin.get(ten, 15);** để nhập dữ liệu cho biến **ten** để ghi vào tệp thì chương trình đọc tệp trong ví dụ 7.3 này cần thay lệnh **tep1>>ten;** bằng lệnh **tep1.get(ten,15);**

Ví dụ 7.4: Chương trình đọc tệp văn bản được ghi theo từng dòng ở ví dụ 7.2, sử dụng lệnh **tenluong.getline()**

```
#include <iostream.h>
#include <fstream.h>
#include<string.h>
int main()
{
    char dong[255];
    ifstream tep1;//Khai bao luong de nhap du lieu
    tep1.open("c:\\tep2.txt"); //mo tep text1.txt de doc dl
    if (!tep1) { cout << "khong mo dc tep";    return 1; }
    cout<<"Nhap du lieu tu tep\n";
    do {tep1.getline(dong, 255);
        cout<<dong<<"\n";
    }
    while (!tep1.eof());tep1.close();
    cout<<"qua trinh doc tep ket thuc:";}
```

7.5 Làm việc với tệp nhị phân

7.5.1 Một số hàm làm việc trên tệp nhị phân

- Hàm ghi dữ liệu ra tệp nhị phân có cú pháp như sau:

tenluong.write(char *vungnho, int n);

Trong đó **tenluong** là tên luồng xuất dữ liệu đã được khai báo và được gán với tệp nhị phân cần ghi.

Hàm này có tác dụng ghi n byte dữ liệu chứa trong vùng nhớ **vungnho** vào tệp đã được gán với **tenluong**.

- Hàm đọc dữ liệu từ tệp nhị phân:

Cú pháp: **tenluong.read**(char *vungnho, int n);

Trong đó **tenluong** là tên luồng nhập đã được khai báo và được gán với tên tệp nhị phân cần đọc.

Hàm này đọc n byte dữ liệu từ tệp vào vùng nhớ xác định bởi **vungnho**.

- Hàm đếm số ký tự thực đọc được bởi hàm read: **gcount()**

7.5.2 Đọc/ghi các số nguyên vào tệp

Ví dụ 7.5: Chương trình sau đây sẽ thực hiện ghi dãy số nguyên vào tệp nhị phân:

```
#include <iostream.h>
#include <fstream.h>
int main()
{
    int a, i=1;    char traloi;
    ofstream tep1;
    //mo tep np de ghi du lieu
    tep1.open("c:\\tepnpl.dat",ios::out|ios::binary);
    if (!tep1)
        { cout<<"khong mo dc tep";return 1;}
    cout <<"Nhap cac so can ghi vao tep:\n";
    do {
        cout<<"Nhap so thu "<<i; cin>>a;
        tep1.write((char *)&a, sizeof(a));
        cout<<" Tiep tục? (c/k) ";cin>>traloi;
        i++;
    } while (traloi=='c');
    tep1.close();
}
```

Ví dụ 7.6: Chương trình đọc tệp nhị phân vừa ghi:

```
#include <iostream.h>
#include <fstream.h>
int main()
```

```
{
    int a, i=1; ifstream tep1;
    //mo tep np de doc du lieu
    tep1.open("c:\\tepnpl.dat", ios::in| ios::binary);
    if (!tep1)
        { cout<<"khong mo dc tep";return 1;}
    cout <<"Nhap cac so doc duoc tu tep:\n";
    while (tep1.read((char *)&a, sizeof(a))&& !tep1.eof())
    {
        cout<<"so thu "<<i<<" "<<a<<"\n";
        i++;
    }
    tep1.close();
}
```

\

CHƯƠNG 8 LỚP VÀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

8.1 Giới thiệu về lập trình hướng đối tượng

8.1.1 Lập trình hướng đối tượng

Lập trình hướng đối tượng (Object Oriented Programming- gọi tắt là OOP) là phương pháp lấy đối tượng làm nền tảng để xây dựng thuật giải, xây dựng chương trình. Thực chất đây không phải là một phương pháp mới mà là một cách nhìn mới trong việc lập trình.

So sánh lập trình hướng đối tượng với lập trình cấu trúc

Ở phương pháp lập trình theo kiểu cấu trúc mà ta đã quen thuộc thì người lập trình phân tích bài toán lớn thành bài toán nhỏ hơn sau đó dần dần chi tiết, cụ thể hóa thành bài toán đơn giản, có giải thuật rõ ràng. Mỗi bài toán nhỏ sẽ được giải quyết trong các chương trình con. Đây là phương pháp lập trình từ trên xuống Top-down.

Ở phương pháp lập trình hướng đối tượng người ta tiếp cận bài toán theo hướng từ dưới lên (Bottom- up). Nghĩa là ta xem xét các đối tượng liên quan đến bài toán, và quan tâm tới các thuộc tính của đối tượng và các hành động có thể thực hiện của đối tượng. Với cách nhìn này thì đối tượng đóng vai trò trung tâm. Người ta luôn cố gắng sắp xếp các đối tượng lại thành từng nhóm có chung các đặc điểm nổi bật để có thể phân biệt loại này với loại khác. Ví dụ như người ta xếp chó, mèo, lợn,... thành nhóm động vật, còn bàn, ghế, tivi, tủ lạnh được xếp vào loại đồ vật,...

Phát triển hướng đối tượng thường xoay quanh định nghĩa các đối tượng thích hợp, sắp xếp chúng, thêm vào các thuộc tính cho đối tượng bằng cách mô tả các đặc tính thích hợp trong một ngữ cảnh phức tạp. Và thêm vào các hàm (phương thức) cho đối tượng để thể hiện các nhiệm vụ yêu cầu mà đối tượng đó cần thực hiện.

8.1.2 Đặc điểm của lập trình hướng đối tượng

8.1.2.1 Tính bền vững (persistence)

Hãy khảo sát trường hợp bán xe hơi. Những chi tiết của khách hàng được lưu trữ ngay khi xe hơi đã được phân phối. Việc duy trì dữ liệu vẫn cần thiết cho đến khi dữ liệu được chỉnh sửa hoặc hủy bỏ chính thức.

Tính Bền vững là khả năng lưu trữ dữ liệu của một đối tượng ngay cả khi đối tượng ấy không còn tồn tại.

Cửa hàng bán xe lưu trữ chi tiết khách hàng vào một file. Những chi tiết này sẽ tồn tại trong file cho đến khi chúng bị hủy, hoặc bản thân file bị hủy.

8.1.2.2 Tính đóng gói (encapsulation)

Tiến trình trừu tượng hóa dữ liệu hỗ trợ cho việc xác định những thuộc tính và những phương thức cần thiết. Thông thường, các đối tượng sử dụng những thuộc tính và những phương thức mà người sử dụng đối tượng không đòi hỏi. Chẳng hạn như trong trường hợp lớp ‘Hàng hóa’. Khi phương thức tính lợi nhuận thực thi thì người sử dụng

Chương 8 Lập trình hướng đối tượng

chỉ cần quan tâm đến tổng lợi nhuận thu được của từng mặt hàng, mà không cần quan tâm là lợi nhuận đó được tính toán như thế nào, tính toán dựa trên thuộc tính nào.

Đối tượng sử dụng những thuộc tính và những phương thức mang tính nội bộ. Các đối tượng khác và những người sử dụng không nhận thức được các thuộc tính và / hoặc các phương thức như thế có tồn tại hay không.

Đóng gói là tiến trình che giấu việc thực thi những chi tiết của một đối tượng đối với người sử dụng đối tượng ấy.

8.1.2.3 Tính kế thừa (Inheritance)

Hãy khảo sát các lớp sau:

Lớp Sinh viên	Lớp Nhân viên	Lớp Khách hàng
Tên	Tên	Tên
Địa chỉ	Địa chỉ	Địa chỉ
Nhập tên	Nhập tên	Nhập địa chỉ
Nhập địa chỉ	Nhập địa chỉ	Nhập tên
Điểm môn 1	Lương	Nhập kiểu xe
Điểm môn 2	Chức vụ	Kiểu xe đã bán
Nhập điểm	Nhập chức vụ	Xuất hóa đơn
Tính tổng điểm	Tính lương	

Trong tất cả ba lớp, chúng ta thấy có một vài thuộc tính và hoạt động chung. Chúng ta muốn nhóm những thuộc tính và những hoạt động ấy lại, và định nghĩa chúng trong một lớp ‘Người’.

Lớp Người
Tên
Địa chỉ
Nhập tên
Nhập địa chỉ

Ba lớp “Sinh viên”, “Nhân viên” và “Khách hàng” đều có tất cả các thuộc tính và các phương thức của lớp ‘Người’, ngoài ra chúng còn có những thuộc tính và những phương thức riêng.

Lớp Sinh viên	Lớp Nhân viên	Lớp Khách hàng
Điểm môn 1	Lương	Kiểu xe bán được
Điểm môn 2	Chức vụ	Nhập kiểu xe
Nhập điểm	Nhập chức vụ	Xuất hóa đơn
tính tổng điểm	Tính lương	

Theo ngôn ngữ hướng đối tượng, lớp ‘Khách hàng’, ‘Nhân viên’, ‘Sinh viên’ được gọi là thừa kế lớp ‘Người’.

Chương 8 Lập trình hướng đối tượng

Tính thừa kế cho phép một lớp chia sẻ các thuộc tính và phương thức được định nghĩa trong một hoặc nhiều lớp khác.

Trong tính kế thừa có hai khái niệm liên quan:

+ **Lớp con** (Subclass): Là lớp được thừa hưởng từ một lớp khác.

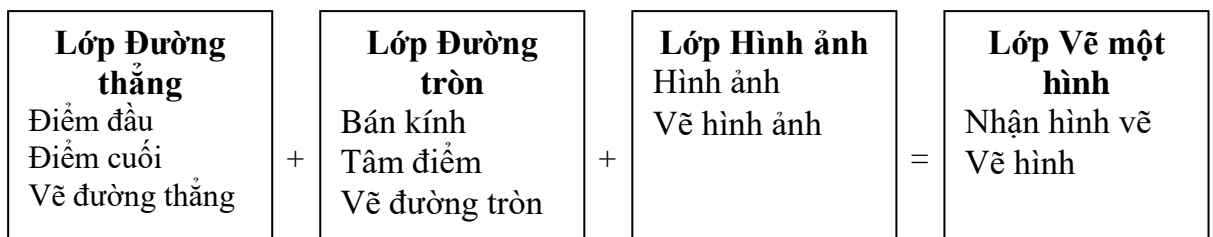
+ **Lớp cha** (Superclass): Là một lớp mà các đặc tính của nó được lớp khác kế thừa.

8.1.2.4 Tính đa kế thừa (Inheritance)

Trong các ví dụ trên, một lớp thừa kế chỉ từ một lớp. Trường hợp như thế gọi là ‘thừa kế đơn’ (single inheritance).

Trong ‘đa thừa kế’, một lớp con thừa kế từ hai hay nhiều lớp cha.

Hãy khảo sát ví dụ sau:



Trong ví dụ trên, chúng ta đã xây dựng một lớp ‘Vẽ một hình’, lớp này thừa hưởng ba lớp: ‘Đường thẳng’, ‘Đường tròn’, ‘Hình ảnh’. Như thế lớp ‘Vẽ một hình’ kết hợp chức năng của ba lớp trên thêm vào chức năng được định nghĩa bên trong nó. Lớp ‘Vẽ một hình’ là một ví dụ về tính đa thừa kế.

Sự thuận lợi quan trọng nhất của tính thừa kế là nó thúc đẩy việc tái sử dụng mã nguồn của chương trình.

8.1.2.4 Tính đa hình

Trong một chương trình có cấu trúc (a structured program), một phương thức chỉ ứng dụng cho một loại đối tượng. Chẳng hạn xét toán tử ‘Cộng’. Toán tử này chỉ tính tổng của hai số nguyên. Khi truyền hai giá trị 2 và 3 thì nó hiển thị 5. Chúng ta không thể có một loại toán tử ‘Cộng’ để tính tổng của hai giá trị văn bản (text) ‘Hello!’ và ‘How are you?’ để có được chuỗi văn bản kết quả ‘Hello! How are you?’

Trong hệ thống hướng đối tượng thì tình huống mô tả trên là khả thể.

Tính đa hình cho phép một phương thức có các tác động khác nhau trên nhiều loại đối tượng khác nhau.

Với tính đa hình, nếu cùng một phương thức ứng dụng cho các đối tượng thuộc các lớp khác nhau thì nó đưa đến những kết quả khác nhau.

Tính đa hình là một trong những đặc tính quan trọng nhất của hệ thống hướng đối tượng.

Chương 8 Lập trình hướng đối tượng

Một ví dụ khác là phương thức hiển thị. Tùy thuộc vào đối tượng tác động, phương thức ấy có thể hiển thị một chuỗi, hoặc vẽ một đường thẳng, hoặc hiển thị một hình ảnh.

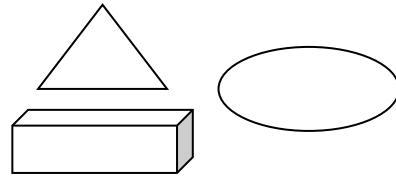
Lớp: Hình thể

Các phương thức:

Vẽ

Di chuyển

Các lớp con



Hình trên cho thấy rằng ‘Vẽ’ là một phương thức được chia sẻ giữa các lớp con của lớp ‘Hình thể’. Tuy nhiên, phương thức Vẽ được ứng dụng cho hình hộp sẽ khác với hình êlip.

Tính đa hình hỗ trợ tính đóng gói.

Xét trên mức độ người sử dụng, họ chỉ cần một phương thức ‘Vẽ’ của lớp ‘Hình thể’. Còn cách thức mà phương thức ‘Vẽ’ được thực thi cho các trường hợp khác nhau thì họ không cần biết.

8.2 Lớp và đối tượng

8.2.1 Khái niệm lớp

Khi có nhiều đối tượng giống nhau về mặt dữ liệu và phương thức chúng được nhóm lại với nhau thành một lớp:

- Lớp là sự trừu tượng hóa của đối tượng
- Đối tượng là một thể hiện của lớp

8.2.2 Khai báo lớp

```
class ten_lop
{
    // Khai báo các thành phần dữ liệu (các thuộc tính)
    // Khai báo các phương thức (hàm)
};
```

Trong đó:

- **class** là từ khóa để khai báo lớp, từ khóa này phải được viết bằng chữ thường.
- `ten_lop` là tên của lớp do người lập trình tự đặt, và phải tuân theo nguyên tắc đặt tên biến trong C++
- Các thành phần dữ liệu (các thuộc tính) và các phương thức (các hàm) gọi chung là các thành phần của lớp.

Việc khai báo các thành phần này cụ thể như sau:

```
class ten_lop
{
    private:
```

<Khai báo các thành phần riêng>

protected:

<Khai báo các thành phần được bảo vệ>

public:

<Khai báo các thành phần công cộng>

};

+ Các thành phần được khai báo trong từ khóa **private** chỉ được truy cập trong phạm vi của chính lớp này (trong các phương thức của lớp, hay trong các hàm bạn của lớp)

+ Các thành phần được khai báo trong từ khóa **protected** chỉ cho phép một số đối tượng nhất định được truy cập.

+ Các thành phần được khai báo trong từ khóa **public** cho phép truy cập từ bất cứ hàm nào trong chương trình.

Ví dụ 8.1: Khai báo lớp Oto có hai thuộc tính tốc độ và nhãn hiệu như sau:

```
class Oto
{
    private:
        int tocdo;
    public:
        char nhan_hieu[20];
};
```

8.2.3 Khai báo biến lớp

Khi khai báo một lớp là ta định nghĩa một kiểu dữ liệu mới ví dụ như kiểu cấu trúc. Muốn sử dụng các thành phần của lớp chúng ta phải khai báo biến lớp:

Cú pháp khai báo biến lớp:

Ten_lop ten_bien_lop;

Ví dụ: *Oto xe1, xe2; //Khai báo biến xe1, xe2 có kiểu lớp Oto vừa khai báo ở trên*

8.2.4 Khai báo thuộc tính

Cú pháp: *Kiểu dữ liệu ten_thuoc_tinh;*

Trong đó:

Kiểu dữ liệu: Có thể là các kiểu dữ liệu cơ bản của C++, hoặc có thể là các kiểu dữ liệu phức tạp do người dùng tự định nghĩa như kiểu cấu trúc struct, hoặc kiểu một lớp đã được định nghĩa trước đó.

ten_thuoc_tinh: Là tên thuộc tính của lớp, tên thuộc tính đặt theo quy tắc đặt tên biến của C++.

Chú ý: Không được vừa khai báo thuộc tính vừa khởi tạo giá trị ban đầu cho nó, vì thuộc tính chỉ có giá trị khi nó được gắn với một đối tượng cụ thể.

Sử dụng các thuộc tính của lớp

Chương 8 Lập trình hướng đối tượng

Thuộc tính của lớp có thể được sử dụng trong chương trình (bên ngoài định nghĩa lớp) thông qua biến lớp hoặc sử dụng ngay trong lớp bởi các phương thức của lớp.

- Nếu thuộc tính được dùng bên ngoài phạm vi lớp và có thuộc tính **public**, thì phải truy cập thông qua tên biến lớp như sau:

ten_bien_lop. ten_thuoc_tinh

Ví dụ: xe1.tocdo=50;

- Nếu thuộc tính được dùng bên trong phần định nghĩa các phương thức của lớp thì sử dụng như biến thông thường mà không cần truy cập thông qua tên biến lớp. Tuy nhiên nếu tên thuộc tính trong trường hợp này trùng với tên của một biến toàn cục trong chương trình thì phải viết theo cú pháp sau: *ten_lop::ten_thuoc_tinh*

Ví dụ: Với định nghĩa lớp:

```
class Oto
```

```
    private:
```

```
        int tocdo;
```

```
    public:
```

```
        char nhan_hieu[20];
```

```
};
```

Ta khai báo biến lớp

Oto xe1, xe2;

In ra màn hình nhãn hiệu của xe 1 như sau:

```
cout<<xe1.nhan_hieu;
```

8.2.5 Khai báo và xây dựng các phương thức

Phương thức được khai báo tương tự như các hàm trong C++ theo cú pháp như sau:

<KiểuDL> <Ten_phuong_thuc> ([<DS tham số>]);

Trong đó:

- *KiểuDL*: Là kiểu trả về của phương thức, có thể là một trong các kiểu dữ liệu cơ bản của C++ hoặc là một kiểu dữ liệu tự định nghĩa, hoặc kiểu lớp đã được định nghĩa từ trước.

- *Ten_phuong_thuc*: Là tên của phương thức do người lập trình tự đặt, tên này tuân thủ theo nguyên tắc đặt tên biến, tên hàm trong C++.

- *DS tham số*: Là danh sách các tham số của phương thức, mỗi tham số ngăn cách nhau bởi dấu phẩy (.). Có thể xây dựng phương thức không có tham số, khi này ta để dấu ngoặc trống sau tên phương thức ().

Trường hợp 1: Khai báo đồng thời xây dựng phương thức ngay trong phần định nghĩa lớp, cú pháp như sau:

<KiểuDL> <Ten_phuong_thuc> ([<DS tham số>])

```
{  
    Các lệnh bên trong thân phương thức;  
}
```

Ví dụ 8.2:

```
class Oto  
{  
    private:  
        int tocdo;  
        char nhan_hieu[20];  
    public:  
        void show()  
        {  
            cout<< "Day la chiec xe mang nhan hieu  
"<<nhan_hieu;  
            cout<< "Chay voi toc do "<<tocdo  
<<"km/h";  
        }  
};
```

Trường hợp 2: Khai báo phương thức trong thân class, xây dựng phương thức bên ngoài thân class thì ta phải sử dụng chỉ thị phạm vi "::" để chỉ rõ đây là phương thức của lớp chứ không phải là một hàm tự do trong chương trình. Cú pháp như sau:

```
<Kiểu DL><Ten_lop>:: <Ten_phuong_thuc>([DS thamso])  
{  
    Các lệnh bên trong thân lớp  
}
```

Ví dụ 8.3: Xây dựng phương thức show() bên ngoài thân lớp

```
class Oto  
{  
    private:  
        int tocdo;  
        char nhan_hieu[20];  
    public:  
        void show();//Khai báo phương thức  
}; //kết thúc class  
  
//Xây dựng phương thức show() bên ngoài lớp  
void Oto:: show()  
{  
    cout<< "Day la chiec xe mang nhan hieu "<<nhan_hieu;
```

```
        cout<< "Chay voi toc do " << tocdo << "km/h";  
    }
```

Ghi chú: Chỉ các phương thức đơn giản gồm một vài câu lệnh thì mới nên xây dựng ngay trong thân lớp, còn lại ta nên xây dựng phương thức bên ngoài thân lớp làm cho chương trình sáng sủa hơn.

8.2.6 Sử dụng phương thức

Cũng như các thuộc tính các phương thức có thể được sử dụng bên ngoài lớp, hoặc ngay bên trong lớp trong các phương thức khác.

- Nếu phương thức được sử dụng bên ngoài lớp thì tuân theo cú pháp sau:

`<ten_bien_lop>.<ten_phuong_thuc>([<DS tham số>])`

Tuy nhiên cách này chỉ sử dụng đối với các phương thức có tính chất **public**:

- Nếu phương thức được sử dụng ngay trong thân lớp bởi các phương thức khác thì cú pháp như sau:

`<ten_phuong_thuc>([<DS tham số>])`

Ví dụ 8.4: Chương trình dưới đây xây dựng một lớp xe Oto có các thuộc tính mang tính chất private:

- tocdo: chỉ tốc độ xe

- nhan_hieu: chỉ nhãn hiệu của xe

Các phương thức

- init (**int** tocdo_in, **char** nhan_hieu_in[]): có chức năng khởi tạo giá trị ban đầu cho hai thuộc tính tocdo và nhan_hieu.

- show(): có chức năng giới thiệu các thông tin về chiếc xe.

<pre> d:\bc45\bin\class1.cpp #include<iostream.h> #include<string.h> class Oto { public: int tocd; char nhan_hieu[20]; public: void show(); void init(int , char[]); }; void Oto::show() { cout<<"Chiec xe mang nhan hieu "<<nhan_hieu; cout<<"\n chay voi van toc"<<tocdo<<" km/h"; } void Oto::init(int tocd_in, char nhan_hieu_in[]) { tocd= tocd_in; strcpy(nhan_hieu,nhan_hieu_in); } void main() { Oto xe1; xe1.init(500,"Ford"); xe1.show(); } </pre>	<p>Kết quả chạy chương trình</p> <pre> Chiec xe mang nhan hieu Ford chay voi van toc500 km/h </pre>
--	---

8.3 Hàm bạn, hàm tạo, hàm hủy

8.3.1 Khái niệm phạm vi

Trong một chương trình C++ được quy định một số phạm vi như sau:

- Phạm vi khối lệnh: Các lệnh bên trong phạm vi của một cặp dấu {} được gọi là cùng phạm vi khối lệnh. Ví dụ:

```

if (delta>0)
{
    x1=(-b+sqrt(delta))/2*a;
    x2=(-b+sqrt(delta))/2*a;
}

```

- Phạm vi hàm: Các lệnh trong thân một hàm được gọi là cùng phạm vi hàm
 - Phạm vi lớp: Các thành phần của một lớp (các thuộc tính và các phương thức) được coi là cùng phạm vi lớp.

- Phạm vi chương trình: Các thành phần trong cùng một chương trình (như các biến, các hàm, các lớp) được gọi là có cùng phạm vi chương trình.

8.3.2 Phạm vi truy nhập lớp

Khi nói về phạm vi truy nhập lớp ta chỉ đề cập đến phạm vi lớp và phạm vi chương trình. Các từ khóa thể hiện phạm vi trong lớp:

Chương 8 Lập trình hướng đối tượng

- **private**: Các thuộc tính và phương thức được khai báo với từ khóa private thì chỉ có thể được truy nhập bởi các phương thức của lớp này.

- **protected**: Các thành phần của lớp được khai báo với từ khóa protected thì cũng chỉ được truy nhập trong phạm vi lớp, ngoài ra còn được truy nhập trong các lớp con khi có sự kế thừa.

- **public**: Các thành phần của lớp được khai báo với từ khóa public thì có thể được truy cập trong phạm vi chương trình.

8.3.3 Hàm tạo

Hàm tạo cũng là một phương thức của lớp (nhưng là hàm đặc biệt) dùng để khởi tạo giá trị ban đầu cho các thuộc tính. Hàm tạo sẽ được gọi mỗi khi một đối tượng mới được tạo ra. Ngay cả khi không khai báo tường minh trong C++ cũng gọi hàm tạo ngầm định khi đối tượng được khai báo.

Khai báo hàm tạo

Hàm khởi tạo của một lớp được khai báo tường minh theo cú pháp sau:

```
class <Tên lớp>{  
    public:  
        <Tên lớp>([<Các tham số>]); // Khai báo hàm khởi tạo  
};
```

Ví dụ :

```
class Oto{  
    int  tocdo;  
    char nhan_hieu[20];  
    float gia;  
    public:  
        Oto (int tocdo_in, char nhan_hieu_in[], float gia_in)  
        {  
            tocdo = tocdo_in;  
            strcpy(nhan_hieu, nhan_hieu_in);  
            gia = gia_in;  
        }  
};
```

Chú ý:

- Hàm tạo phải có tên trùng với tên của lớp
- Hàm khởi tạo không có giá trị trả về cũng không có từ khóa **void**
- Hàm khởi tạo có tính chất public
- Có thể có nhiều hàm khởi tạo của cùng một lớp (chồng hàm)

Sử dụng hàm tạo của lớp

Chương 8 Lập trình hướng đối tượng

Hàm khởi tạo được sử dụng khi khai báo một đối tượng mới. Khi đó, ta có thể vừa khai báo, vừa khởi tạo giá trị các thuộc tính của đối tượng theo cú pháp sau:

`<Ten_lop> <Ten_doi_tuong>([<Các đối số khởi tạo>]);`

Trong đó:

- *Ten_lop*: Là tên kiểu lớp đã được định nghĩa
- *Ten_doi_tuong*: Là tên biến có kiểu lớp, tuân thủ theo quy tắc đặt tên biến của

C++

- *Các đối số khởi tạo*: Là các đối số tương ứng với hàm khởi tạo của lớp tương ứng. Tương tự như việc truyền đối số khi dùng các lời gọi hàm thông thường.

Ví dụ: Nếu lớp *Oto* có hàm khởi tạo *Oto(int, char[], float)* thì khi khai báo biến có kiểu lớp *Oto*, ta sẽ sử dụng hàm khởi tạo này như sau:

`Oto xe1(100, "Ford", 3000);`

Chú ý:

- Khi sử dụng hàm khởi tạo, phải truyền đúng số lượng và đúng kiểu của các tham số của các hàm khởi tạo đã được định nghĩa của lớp.

- Khi một lớp đã có ít nhất một hàm khởi tạo tường minh, thì không được sử dụng hàm khởi tạo ngầm định của C++. Do đó, khi đã khai báo ít nhất một hàm khởi tạo, nếu muốn khai báo biến mà không cần tham số, lớp tương ứng phải có ít nhất một hàm khởi tạo không có tham số.

Ví dụ: Nếu lớp *Oto* chỉ có duy nhất một hàm khởi tạo như sau:

`class Oto{`

public:

`Oto(int, char[], float);`

`};`

Thì không thể khai báo một biến như sau:

`Oto xe1; // Khai báo lỗi`

Trong trường hợp dùng hàm khởi tạo không có tham số, ta không cần phải sử dụng cặp dấu ngoặc đơn “()” sau tên biến đối tượng. Việc khai báo trở thành cách khai báo thông thường.

Ví dụ 8.5: Chương trình minh họa việc khai báo và sử dụng lớp *Oto* với 2 hàm khởi tạo khác nhau:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
/* Định nghĩa lớp */
class Oto{
private:
    int  tocdo;    // Tốc độ
    char  nhan_hieu[20];    // Nhân hiệu
    float gia;    // Giá xe
```



```
public:
    Oto();    // Khởi tạo không tham số
    Oto(int, char[], float); // Khởi tạo đầy đủ tham số
    void show();    // Giới thiệu xe
};
/* Khai báo phương thức bên ngoài lớp */
Oto::Oto() // Hàm khởi tạo không tham số
{
    tocdo = 0; strcpy(nhanhieu, ""); price = 0;
}
// Hàm khởi tạo có tham số
Oto::Oto(int tocdo_in, char nhan_hieu_in[], float gia_in)
{
    tocdo =tocdo_in;
    strcpy(nhan_hieu, nhan_hieu_in);
    gia =gia_in;
}
void Oto::show()// Phương thức giới thiệu xe
{
    cout << "Day la xe mang nhan hieu " <<nhan_hieu<<endl ;
    cout << "toc do cho phep "<<tocdo << "km/h"<<endl;
    cout <<" Gia cua xe la $"<<gia << endl;
    return;
}
// Hàm main, chương trình chính
void main()
{
    clrscr();
    Oto xe1;    // Sử dụng hàm khởi tạo không tham số
    Oto xe2(150, "Mercedes", 5000); // Dùng hàm khởi tạo có
    tham số
    // Giới thiệu xe thứ nhất
    cout << "Xe thu nhât: " << endl; xe1.show();
    // Giới thiệu xe thứ hai
    cout << "Xe thu hai: " << endl; xe2.show();
    return;
}
```

Kết quả chạy chương trình

Xe thu nhât:

Day la xe mang nhan hieu

Toc do cho phep 0km/h

Gia cua xe la \$0

Xe thu hai:

Day la xe mang nhan hieu Mercedes

Toc do cho phep 150km/h

Gia cua xe la \$5000

Chương 8 Lập trình hướng đối tượng

Trong ví dụ trên sử dụng 2 hàm khởi tạo để khởi tạo 2 bộ giá trị khác nhau cho các thuộc tính. Về nguyên tắc ta hoàn toàn có thể chỉ sử dụng một hàm khởi tạo, và truyền vào các tham số khác nhau:

Ví dụ: `Oto xe1(0,"",0);`

`Oto xe2(150, "Mercedes",5000);`

Hoặc ta sử dụng giá trị ngầm định như sau:

Sửa lại câu lệnh khai báo hàm khởi tạo Oto:

`Car(int tocd0_in=0, char nhan_hieu_in[]="", float gia_in=0);`

Sau đó khai báo các đối tượng xe1, xe2 như sau:

`Oto xe1; //Sử dụng các giá trị ngầm định`

`Oto xe2 (150, "Mercedes",5000); //Khởi tạo giá trị cho các`

`thuộc tính`

8.3.4 Hàm hủy

Hàm hủy là hàm được tự động gọi đến khi mà đối tượng được giải phóng khỏi bộ nhớ. Nhiệm vụ của hàm hủy là dọn dẹp bộ nhớ trước khi đối tượng bị giải phóng.

Khai báo hàm hủy:

`class <Ten_lop>{`

public:

`~<Ten_lop>([<Các tham số>]); // Khai báo hàm khởi tạo`

`};`

Ví dụ:

`class Oto{`

`int tocd0; char *nhan_hieu; float gia;`

`}`

public:

`~Oto() { delete []nhan_hieu;}`

`}`

Chú ý:

- Hàm hủy bỏ phải có tên bắt đầu bằng dấu "~", theo sau là tên của lớp tương ứng.
- Hàm hủy bỏ không có giá trị trả về.
- Hàm hủy bỏ phải có tính chất public
- Mỗi lớp chỉ có nhiều nhất một hàm hủy bỏ. Trong trường hợp không khai báo tường minh hàm hủy bỏ, C++ sẽ sử dụng hàm hủy bỏ ngầm định.

Nói chung, khi có ít nhất một trong các thuộc tính của lớp là con trỏ, thì nên dùng hàm hủy bỏ tường minh để giải phóng triệt để các vùng nhớ của các thuộc tính, trước khi đối tượng bị giải phóng khỏi bộ nhớ.

8.3.5 Hàm bạn

Có hai kiểu hàm bạn cơ bản trong C++:

- Một hàm tự do là hàm bạn của một lớp
- Một hàm thành phần (phương thức) của một lớp là bạn của một lớp khác. Ngoài ra còn có một số kiểu hàm bạn mở rộng từ hai kiểu này:

- Một hàm là bạn của nhiều lớp
- Tất cả các phương thức của một lớp là bạn của lớp khác (lớp bạn)

8.3.5.1 Hàm tự do bạn của một lớp

Khai báo:

- + Khai báo khuôn mẫu hàm tự do trong khai báo lớp mà hàm này sẽ làm bạn theo cú pháp:

```
class <Ten_lop>{  
    // Khai báo các thành phần lớp như thông thường  
    // Khai báo hàm bạn  
    friend <Kiểu trả về> <Tên hàm bạn> ([<Các tham số>]);  
};
```

- + Sau đó, định nghĩa chi tiết hàm bạn như định nghĩa một hàm tự do thông thường:

```
<Kiểu trả về> <Tên hàm bạn> ([<Các tham số>]){  
    ... // Có thể truy nhập trực tiếp các thành phần private của lớp đã khai báo  
}
```

Lưu ý:

- Mặc dù hàm bạn được khai báo khuôn mẫu trong phạm vi lớp, nhưng hàm bạn tự do lại không phải là một phương thức của lớp. Nó là hàm tự do, việc định nghĩa và sử dụng hàm này hoàn toàn tương tự như các hàm tự do khác.

1-Việc khai báo khuôn mẫu hàm bạn trong phạm vi lớp ở vị trí nào cũng được: hàm bạn không bị ảnh hưởng bởi các từ khóa private, protected hay public trong lớp.

-Trong hàm bạn, có thể truy nhập trực tiếp đến các thành phần private và protected của đối tượng có kiểu lớp mà nó làm bạn (truy nhập thông qua đối tượng cụ thể).

Ví dụ 8.6: Chương trình sau đây minh họa việc khai báo và sử dụng hàm tự do là bạn của một lớp. Chương trình thực hiện việc so sánh giá của hai chiếc xe.

```
#include<stdio.h>  
#include<string.h>  
/* Định nghĩa lớp */  
class Oto{  
private:  
    int tocdto;  
    char nhan_hieu[20];  
    float gia;
```

```

public:
    void khoitao(int, char[], float); // Khởi tạo thông tin về xe
    // Khai báo hàm bạn của lớp
    friend void so_sanh(Oto, Oto);
}; // End of class
/* Khai báo phương thức bên ngoài lớp */
void Oto::khoitao(int tocdo_in, char nhan_hieu_in[], float gia_in)
{   tocdo=tocdo_in;   strcpy(nhan_hieu, nhan_hieu_in); gia = gia_in;
    return;
}
/* Định nghĩa hàm bạn tự do */
void so_sanh(Oto xe1, Oto xe2)
{
    if (xe1.gia > xe2.gia ) // Truy nhập đến các thuộc tính private
        cout << "xe thu nhat dat hon" << endl;
    else if (xe1.gia < xe2.gia)
        cout << "xe thu nhat dat hon" << endl;
    else   cout << "hai xe dat nhu nhau" << endl;
    return;
}
// Hàm main, chương trình chính
void main() { clrscr();
    Oto xe1, xe2;           // Khai báo 2 biến lớp
    // Khởi tạo xe thứ nhất, thứ hai
    xe1.khoitao(100, "Ford", 3000);
    xe2.khoitao(150, "Mercedes", 3500);
    // So sánh giá hai xe
    so_sanh(xe1, xe2);      // Sử dụng hàm bạn tự do
    return;
}

```

Kết quả chạy chương trình:

xe thu hai dat hon

8.3.5.2 Phương thức của lớp là bạn của một lớp khác

Trong C++, một phương thức của lớp này cũng có thể làm bạn của một lớp kia. Để khai báo một phương thức f của lớp B là bạn của lớp A và f nhận một tham số có kiểu lớp A, ta phải khai báo tuần tự như sau (trong cùng một chương trình):

- Khai báo khuôn mẫu lớp A, để làm tham số cho phương thức f của lớp B:

class A;

- Khai báo lớp B có dòng nguyên mẫu của phương thức f :

class B

{

// Khai báo các thành phần khác của lớp B

void f(A);

};

- Khai báo chi tiết lớp A với phương thức f của lớp B là bạn

class A{

```
// Khai báo các thành phần khác của lớp A
friend void B::f(A);
};
```

- Định nghĩa chi tiết phương thức f của lớp B:

```
void B::f(A){
// Định nghĩa chi tiết phương thức f
}
```

Lưu ý:

- Trong trường hợp này, phương thức f chỉ được định nghĩa chi tiết một khi lớp A đã được định nghĩa chi tiết. Do vậy, chỉ có thể định nghĩa chi tiết phương thức f ngay trong lớp A hoặc sau khi định nghĩa lớp A, mà không thể định nghĩa chi tiết hàm f ngay trong lớp B.

- Hàm f có thể truy nhập đến các thành phần private và protected của cả hai lớp A và B. Tuy nhiên, muốn f truy nhập đến các thành phần của lớp A thì phải thông qua một đối tượng cụ thể có kiểu lớp A.

Ví dụ 8.7: Chương trình dưới đây minh họa việc xây dựng và sử dụng một phương thức duocphep() của lớp Congdan, là hàm bạn của lớp Oto. Hàm này thực hiện việc kiểm tra xem một công dân có đủ quyền điều khiển xe hay không, theo qui định sau:

- + Với các loại xe thông thường, người điều khiển phải đủ 18 tuổi.
- + Với các loại xe có tốc độ cao hơn 150km/h, người điều khiển phải đủ 21 tuổi.

8.3.5.3 Lớp bạn

Khi tất cả các phương thức của lớp B đều là bạn của lớp A thì ta nói lớp B là lớp bạn của lớp A. Lúc này các phương thức của lớp B đều có thể truy nhập đến các dữ liệu thành phần của lớp A thông qua đối tượng lớp A.

Ví dụ 8.8:

```
#include<iostream.h>
#include<conio.h>
//xd lop A
class A
{
    int x;
    friend class B;//lop B la ban cua lop A
    void showx();
public:
    void getx(int x1)
    { x=x1;}
};//end of class A
void A::showx()
{
    cout<<"Class A: x="<<x<<endl;
}
//xd lop B
class B
{
    int y;
public:
    void gety(int y1)
    { y=y1;}
    void showy(A a);
};//end of class B

void B::showy(A a)
{
    cout<<"class B: y="<<y<<endl;
    cout<<"class B: x="<<a.x<<endl;//sd thanh phan du lieu rieng cua lop A
    a.showx();//trong lop B goi phuong thuc cua lop A
}

void main()
{
    A a; B b;
    a.getx(100);
    b.gety(50);
    b.showy(a);
    getch();
}
```

8.4 Kế thừa

Thừa kế là một trong bốn nguyên tắc cơ sở của phương pháp lập trình hướng đối tượng. Đặc biệt đây là cơ sở cho việc nâng cao khả năng sử dụng lại các bộ phận của chương trình. Thừa kế cho phép ta định nghĩa một lớp mới, gọi là lớp dẫn xuất, từ một lớp đã có, gọi là lớp cơ sở. Lớp dẫn xuất sẽ kế thừa các thành phần (dữ liệu, hàm) của lớp cơ sở, đồng thời thêm vào các thành phần mới, bao hàm cả việc “làm tốt hơn” hoặc “làm lại” những công việc mà trong lớp cơ sở chưa làm tốt hoặc không còn phù hợp với lớp dẫn xuất. Chẳng hạn có thể định nghĩa lớp “mặt hàng nhập khẩu” dựa trên lớp “mặt hàng”, bằng cách bổ xung thêm thuộc tính thuế. Khi đó cách tính chênh lệch giá bán, mua cũ trong lớp “mặt hàng” sẽ không còn phù hợp nữa nên cần sửa lại cho phù hợp. Lớp “điểm có màu” được định nghĩa dựa trên lớp “điểm không màu”, bằng cách bổ xung thêm thuộc tính màu. Lúc này hàm display() ngoài việc phải hiển thị hai thành phần tọa độ, sẽ còn phải hiển thị thêm thuộc tính màu.

Thừa kế cho phép không cần biên dịch lại các thành phần chương trình vốn đã có trong các lớp cơ sở và hơn thế nữa không cần phải có chương trình nguồn tương ứng.

8.4.1 Khai báo kế thừa

```
class <Tên lớp dẫn xuất>: <Từ khóa dẫn xuất> <Tên lớp cơ sở>{  
... // Khai báo các thành phần lớp  
};
```

Trong đó:

- *Tên lớp dẫn xuất*: Là tên lớp được cho kế thừa từ lớp khác. Tên lớp này tuân thủ theo quy tắc đặt tên biến trong C++.

- *Tên lớp cơ sở*: là tên lớp đã được định nghĩa trước đó để cho lớp khác kế thừa. Tên lớp này cũng tuân thủ theo quy tắc đặt tên biến của C++.

- *Từ khóa dẫn xuất*: là từ khóa quy định tính chất của sự kế thừa. Có ba từ khóa dẫn xuất private, protected và public.

Ví dụ 8.9: Xây dựng một lớp *point* với hai toạ độ của một điểm x,y. Và một hàm tạo để khởi tạo giá trị ban đầu cho x và y, một phương thức *display()* để hiển thị giá trị của x, y. Một lớp *coloredpoint* kế thừa hai thành phần dữ liệu x,y từ lớp *point*, và thêm thành phần *color* (màu) của điểm, và một phương thức *display()* hiển thị thêm thông tin về màu so với phương thức *display()* của lớp *point*.

```
//Xây dựng lớp cơ sở point  
class point  
{  
    float x,y;  
    public:  
        point (float ox,float oy) {x=ox;y=oy;}  
        void display()  
        { cout<<"Toa do : "<<x<<" "<<y<<endl;}  
};  
  
//Xây dựng lớp dẫn xuất colored point  
class coloredpoint:public point  
{  
    unsigned int color;//thêm thuộc tính color  
    public:  
        //xây dựng hàm tạo kế thừa từ hàm tạo của class point  
        coloredpoint (float ox, float oy, unsigned int c):  
        point (ox,oy)  
        { color=c;}  
        void display()  
        {point::display();//gọi hàm trùng tên ở lớp cơ sở  
        cout<<"Mau "<<color<<endl; }  
};  
}
```

Truy nhập các thành phần của lớp cơ sở từ lớp dẫn xuất:

Lớp dẫn xuất không truy cập được vào các thành phần **private** của lớp cơ sở, kể cả sử dụng từ khoá dẫn xuất **public**.

8.4.2 Hàm tạo trong kế thừa

Khi khai báo một đối tượng có kiểu lớp được dẫn xuất từ một lớp cơ sở khác. Chương trình sẽ tự động gọi tới hàm khởi tạo của lớp dẫn xuất. Tuy nhiên, trước khi thực hiện các lệnh trong hàm khởi tạo của lớp dẫn xuất thì hàm khởi tạo của lớp cơ sở đã được thực thi. Do đó, thông thường, trong hàm khởi tạo của lớp dẫn xuất phải có hàm khởi tạo của lớp cơ sở.

Cú pháp khai báo hàm khởi tạo của lớp dẫn xuất như sau:

```
<Tên hàm khởi tạo dẫn xuất>([<Các tham số>]):  
<Tên hàm khởi tạo cơ sở>([<Các đối số>]){  
... // Khởi tạo các thuộc tính mới bổ sung của lớp dẫn xuất  
};
```

Vì tên hàm khởi tạo là trùng với tên lớp, nên có thể viết lại thành:

```
<Tên lớp dẫn xuất>([<Các tham số>]):  
<Tên lớp cơ sở>([<Các đối số>]){  
... // Khởi tạo các thuộc tính mới bổ sung của lớp dẫn xuất  
};
```

Ví dụ

```
coloredpoint (float ox, float oy, unsigned int c) : point (ox,oy)  
{ color=c;}
```

Trong đó *point(ox,oy)* là hàm khởi tạo của lớp *point*, hàm *coloredpoint* là hàm khởi tạo của lớp *coloredpoint*, hàm này sẽ kế thừa việc khởi tạo hai thành phần x,y từ lớp *point*, và nó thực hiện thêm việc khởi tạo giá trị cho thành phần *color*.

Chú ý:

- Nếu định nghĩa hàm khởi tạo bên ngoài phạm vi lớp thì phải thêm tên lớp dẫn xuất và toán tử phạm vi “::” trước tên hàm khởi tạo.

- Giữa tên hàm khởi tạo của lớp dẫn xuất và hàm khởi tạo của lớp cơ sở, chỉ có một dấu hai chấm “:”, nếu là hai dấu “::” thì trở thành toán tử phạm vi lớp.

- Nếu không chỉ rõ hàm khởi tạo của lớp cơ sở sau dấu hai chấm “:” chương trình sẽ tự động gọi hàm khởi tạo ngầm định hoặc hàm khởi tạo không có tham số của lớp cơ sở nếu hàm đó được định nghĩa tường minh trong lớp cơ sở.

Ví dụ: Nếu ta viết

```
coloredpoint (float ox, float oy, unsigned int c)  
{ color=c;}
```


Chương 8 Lập trình hướng đối tượng

Thì chương trình sẽ tự động gọi đến hàm khởi tạo không có tham số trong lớp cơ sở point. Với điều kiện trong lớp point phải xây dựng hàm tạo này

`point() {x=0;y=0;} //hàm tạo không có tham số trong lớp point`

Ví dụ 8.10: Xây dựng chương trình định nghĩa lớp **oto** có 3 thuộc tính với hai hàm khởi tạo, sau đó định nghĩa lớp **xebuyt** có thêm thuộc tính *sohieu* là số hiệu của tuyến xe buýt. Lớp **xebuyt** sẽ được cài đặt hai hàm khởi tạo tương minh, gọi đến hai hàm khởi tạo tương ứng của lớp **oto**

```
#include<iostream.h>
#include <string.h>
class oto
{
    int tocdo;
    char nhanhieu[20];
    float gia;
public:
    //ham khoi tao khong co tham so
    oto()
    {
        tocdo=0;
        strcpy(nhanhieu,"");gia=0;
    }
    //ham khoi tao co tham so
    oto(int , char [], float ) ;
    void display()
    {
        cout<<"Ham display trong lop oto \n ";
        cout<<" Chiec xe mang nhan hieu:" <<nhanhieu<<endl;
        cout<<" Chay voi toc do "<<tocdo<<endl;
        cout<<"co gia "<<gia<<endl;
    }
};
//xay dung ham khoi tao ben ngoai dinh nghia lop
oto::oto(int tocdo_in, char nhanhieu_in[], float gia_in)
{
    tocdo=tocdo_in;  strcpy(nhanhieu,nhanhieu_in);
    gia=gia_in;
}
class xebuyt:public oto
{
    int sohieu;//them thanh phan so hieu xe buyt
public:
    xebuyt()
    {sohieu=0;}//Ham ktạo ko thso, tu dong gọi ham ktạo oto()
    //Ham khoi tao cua lop xebuyt
    xebuyt(int tocdo_in, char nhanhieu_in[], float gia_in,
int sohieu_in):oto(tocdo_in,nhanhieu_in,gia_in)
    {  sohieu=sohieu_in;}
    void display()
    {
        oto::display(); //gọi ham display trong lop co so
```

```
        cout<<" Thêm thông tin về số hiệu xe:"<<sohieu<<endl;
    }
};
void main()
{
    oto xe1(150,"mercedes",123.5);
    xebuyt xb1(200,"Ford",1234,55);
    cout<<"Hien thi thông tin về xe oto 1"; xe1.display();
    cout<<"\n Thông tin về chiếc xe buýt ";xb1.display();
}
```

8.4.3 Hàm hủy trong kế thừa

Khi một đối tượng lớp dẫn xuất bị giải phóng khỏi bộ nhớ, thứ tự gọi các hàm hủy bỏ ngược với thứ tự gọi hàm thiết lập: gọi hàm hủy bỏ của lớp dẫn xuất trước khi gọi hàm hủy bỏ của lớp cơ sở.

Vì mỗi lớp chỉ có nhiều nhất là một hàm hủy bỏ, nên ta không cần phải chỉ ra hàm hủy bỏ nào của lớp cơ sở sẽ được gọi sau khi hủy bỏ lớp dẫn xuất. Do vậy, hàm hủy bỏ trong lớp dẫn xuất được khai báo và định nghĩa hoàn toàn giống với các lớp thông thường:

```
<Tên lớp>::~~<Tên lớp>([<Các tham số>]){
    ... // giải phóng phần bộ nhớ cấp phát cho các thuộc tính bổ sung
}
```

Lưu ý:

- Hàm hủy bỏ của lớp dẫn xuất chỉ giải phóng phần bộ nhớ được cấp phát động cho các thuộc tính mới bổ sung trong lớp dẫn xuất (nếu có), mà không được giải phóng bộ nhớ được cấp cho các thuộc tính trong lớp cơ sở (phần này là do hàm hủy bỏ của lớp cơ sở đảm nhiệm).

- Không phải gọi tường minh hàm hủy bỏ của lớp cơ sở trong hàm hủy bỏ của lớp dẫn xuất.

- Ngay cả khi lớp dẫn xuất không định nghĩa tường minh hàm hủy bỏ (do không cần thiết) mà lớp cơ sở lại có định nghĩa tường minh. Chương trình vẫn gọi hàm hủy bỏ ngầm định của lớp dẫn xuất, sau đó vẫn gọi hàm hủy bỏ tường minh của lớp cơ sở.

Ví dụ 8.11: Chương trình sau xây dựng lớp xebuyt kế thừa từ lớp oto: lớp oto có một thuộc tính có dạng con trỏ nên cần giải phóng bằng hàm hủy bỏ tường minh. Lớp xebuyt có thêm một thuộc tính có dạng con trỏ là danh sách các đường phố mà xe buýt đi qua (mảng động các chuỗi kí tự *char[]) nên cũng cần giải phóng bằng hàm hủy bỏ tường minh.

```
#include<string.h>
/* Định nghĩa lớp oto */
class oto
{   char   *nhanhieu;    // Nhãn hiệu xe
```

```
public:
    ~oto();    // Hủy bỏ tường minh
};
oto::~~oto() // Hủy bỏ tường minh
{ delete [] nhanhieus; }
/* Định nghĩa lớp xebuyt kế thừa từ lớp oto */
class xebuyt: public oto
{ char *hanhtrinh[];    // Hành trình tuyến xe
public:
    ~xebuyt();    // Hủy bỏ tường minh
};
xebuyt::~~xebuyt()    // Hủy bỏ tường minh
{delete [] hanhtrinh; }
```

8.5 Chồng toán tử trên lớp

Định nghĩa chồng toán tử là việc ta định nghĩa lại một phép toán của C++ cho phù hợp với mục đích sử dụng của mình.,

Toán tử được định nghĩa chồng bằng cách định nghĩa một hàm toán tử. Tên hàm toán tử bao gồm từ khóa operator theo sau là ký hiệu của toán tử được định nghĩa chồng.

Hầu hết các toán tử của C++ đều có thể định nghĩa chồng. Không thể tạo ra các ký hiệu phép toán mới.

Phải đảm bảo các đặc tính nguyên thủy của toán tử được định nghĩa chồng. Chẳng hạn như : độ ưu tiên, trật tự kết hợp, số ngôi.

Không sử dụng tham số có giá trị ngầm định để định nghĩa chồng toán tử. Các toán tử (), [], ->, = yêu cầu Hàm toán tử phải là hàm thành phần của lớp. Các toán tử khác thì hàm toán tử có thể là hàm thành phần hay hàm bạn của lớp.

Khi hàm toán tử là hàm thành phần, toán hạng bên trái luôn là đối tượng thuộc lớp. Nếu toán hạng bên trái là đối tượng thuộc lớp khác thì hàm toán tử tương ứng phải là hàm bạn.

Chương trình dịch không tự biết cách chuyển kiểu giữa kiểu dữ liệu chuẩn và kiểu dữ liệu tự định nghĩa, vì vậy người lập trình phải mô tả tường minh các chuyển đổi kiểu này dưới dạng hàm tạo chuyển kiểu hay hàm toán tử chuyển kiểu.

Ví dụ 8.12: Định nghĩa chồng toán tử + để thực hiện phép cộng hai phân số:

```
class phanso
{ private:
    int tu, mau;
public:
    phanso (int , int);
    //định nghĩa chong toan tu
    phanso operator +(phanso ps);
    void hienthi()
    {
```

```
        cout<<" Tu so "<<this->tu<<"\n";
        cout<<" Mau so "<<this->mau<<"\n";
    }

};

//Xây dựng hàm tạo
phanso::phanso(int ts=1, int ms=1)
{
    tu=ts; mau=ms;
}

//Xây dựng hàm toán tử
phanso phanso::operator+( phanso ps)
{
    phanso p;
    int bc=bscnn(mau,ps.mau);
    p.tu=(bc/mau)* tu+(bc/ps.mau)*ps.tu;
    p.mau=bc;
    return p;
}

void main()
{
    phanso ps1(5,6),ps2(6,7),kq;
    ps1.hienthi();
    ps2.hienthi();
    // Sử dụng phép toán + đã được định nghĩa chồng
    kq=ps1+ps2;
    kq.hienthi();
}
```

8.6 Bài tập

Bài 1: Xây dựng lớp Congdan gồm các thuộc tính họ đệm, tên, tuổi, hàm tạo, phương thức hiển thị. Xây dựng lớp Canbo kế thừa từ lớp Congdan, thêm thuộc tính mã cán bộ, hệ số lương. Hàm tạo của lớp Canbo kế thừa từ hàm tạo của lớp Congdan, phương thức hiển thị của lớp Canbo gọi đến phương thức hienthi của lớp Congdan. Thêm phương thức tính lương cho cán bộ lương= hệ số lương* 512;

Bài 2: Xây dựng lớp Diem gồm các thuộc tính hoành độ x, tung độ y, hàm tạo, phương thức hienthi. Lớp Hinhtron kế thừa từ lớp Diem, thêm thuộc tính bán kính r. (Các thuộc tính có tính chất private). Hàm tạo của lớp Hinhtron kế thừa từ hàm tạo của lớp Diem, phương thức hienthi của lớp Hinhtron gọi phương thức hienthi của lớp Diem. Xây dựng thêm phương thức tính chu vi hình tròn.

Bài 3: Viết chương trình có sử dụng lớp Diem gồm các thuộc tính hoành độ x và tung độ y (các thuộc tính có tính chất private). Hàm tạo, phương thức hiển thị. Viết hàm bạn tính độ dài đoạn thẳng nối hai điểm bất kỳ. So sánh độ dài hai đoạn thẳng.

Bài 4:Viết chương trình có sử dụng lớp Congdan gồm các thuộc tính Họ đệm, tên, tuổi. Lớp Oto gồm các thuộc tính nhãn hiệu, tốc độ. (Các thuộc tính đều có tính chất

Chương 8 Lập trình hướng đối tượng

private). Mỗi lớp đều có hàm tạo. Phương thức hiển thị. Viết hàm bạn kiểm tra xem một công dân bất kỳ có đủ điều kiện lái một chiếc xe hay không. Điều kiện: nếu công dân nhỏ hơn 18 tuổi thì không được lái xe, công dân ≥ 18 tuổi và < 50 tuổi thì được lái mọi loại xe, công dân ≥ 50 tuổi thì chỉ được lái xe tốc độ < 150 km/h.

