Chương 1

TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH C/C++

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

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

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

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

2. Kỹ thuật để giải quyết một bài toán

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

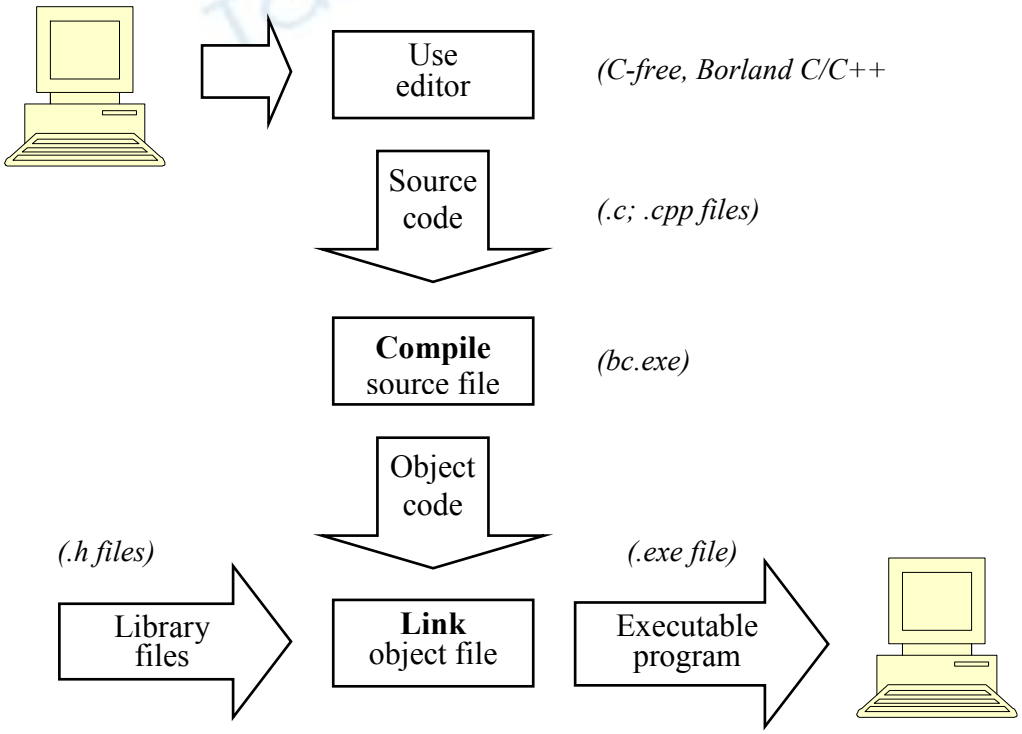
- Xác định yêu cầu của bài toán.
- Nghĩ ra một phương cách *(algorithm)* để tìm lời giải.
- Thực hiện phương cách đó.
- Kiểm tra kết quả để xem lời giải có đúng với yêu cầu của bài toán.

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

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

3. Các bước trong chu trình phát triển chương trình

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



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

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

3.2. Biên dịch (*compile*)

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

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

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

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

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

3.4. Thực thi chương trình

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

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

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

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

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

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

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

// my first program in C/C++

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

#include < iostream.h>

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

int main()

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

cout << "Hello World!";

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

getch();

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

return 0;

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

Chương trình trên có thể viết lại như sau:

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

cũng cho cùng một kết quả.

5. Các chú thích (*comments*)

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

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

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

Ví dụ:

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

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

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

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

Khảo sát chương trình sau:

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

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

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

Khai báo các tập tin thư viện

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

Khai báo prototype của các hàm tự tạo

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

Hàm chính của chương trình

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

Định nghĩa các hàm do người dùng tự xây dựng

7. Các tập tin thư viện thông dụng

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

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

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

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

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

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

Chương 2
BIỂU THỨC
(Expressions)

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

1. Kiểu dữ liệu (*data types*)

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

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

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

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

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

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

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

2. Các định danh (*Identifier names*)

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

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

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

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

3. Từ khóa (*keywords*)

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

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

4. Biến (*variables*)

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

```
type variableNames;
```

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

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

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

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

Ví dụ:

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

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

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

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

4.1. Biến cục bộ (*local variables*)

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

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

Ví dụ: Xem xét hai hàm sau:

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

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

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

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

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

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

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

4.3. Biến toàn cục (*global variables*)

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

Khảo sát chương trình sau:

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

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

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

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

Value of gVar= 100

After increased, gVar= 101;

After decreased, gVar= 100;

5. Từ khóa const

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

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

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

6. Hằng (*constants*)

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

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

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

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

Ví dụ về cách viết các loại hằng số:

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

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

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

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

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

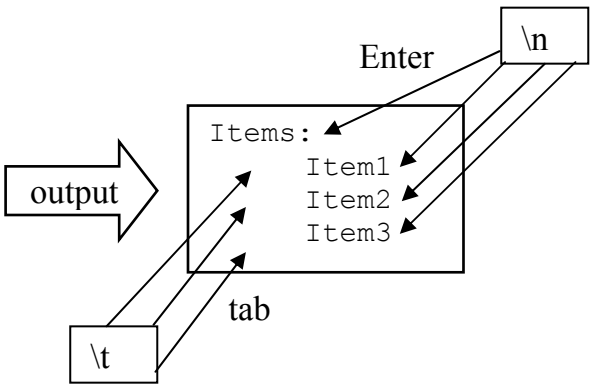
C/C++ có những hằng ký tự đặc biệt mà không thể biểu diễn như những hằng ký tự thông thường. Những hằng này còn gọi là escape sequences. Sau đây là danh sách các hằng đặc biệt đó:

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

Xem xét ví dụ sau:

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

}
```



9. Toán tử (*operators*)

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

9.1. Toán tử gán (*assignment operator*)

Dạng tổng quát của toán tử gán là

variableName = expression;

variableName: Tên biến

expression: Biểu thức

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

Ví dụ:

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

9.2. Chuyển đổi kiểu trong câu lệnh gán

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

Ví dụ:

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

Nếu thực thi lệnh

```
i = d;
```

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

Nếu thực thi lệnh

```
d = i;
```

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

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

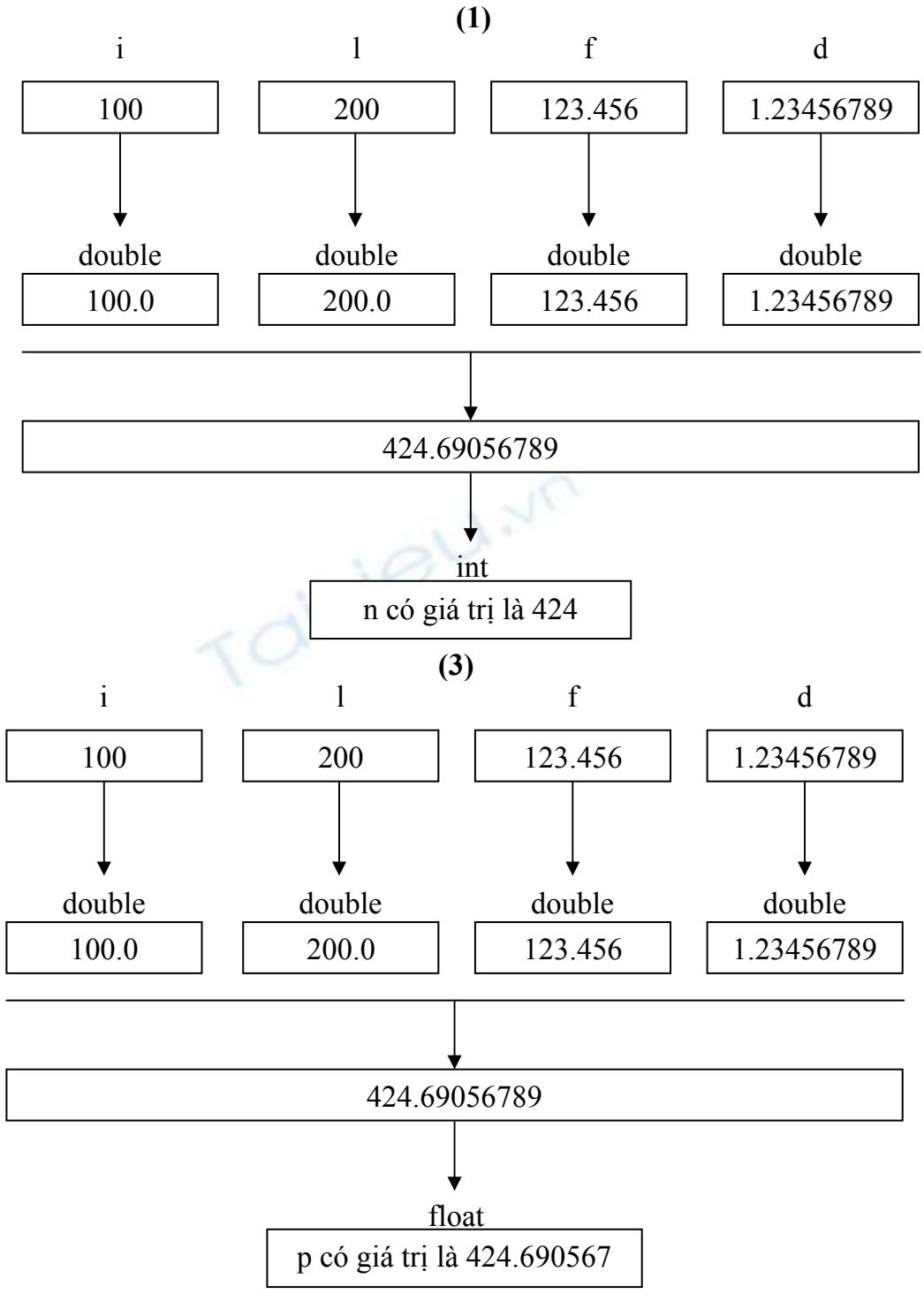
Khi chuyển đổi kiểu dữ liệu có miền giá trị lớn sang kiểu dữ liệu có miền giá trị nhỏ hơn thì việc chuyển đổi kiểu này là không an toàn vì có thể mất mát thông tin. Thứ tự giảm dần từ kiểu dữ liệu có miền giá trị lớn đến kiểu dữ liệu có miền giá trị nhỏ là **double → float → long → int → char.**

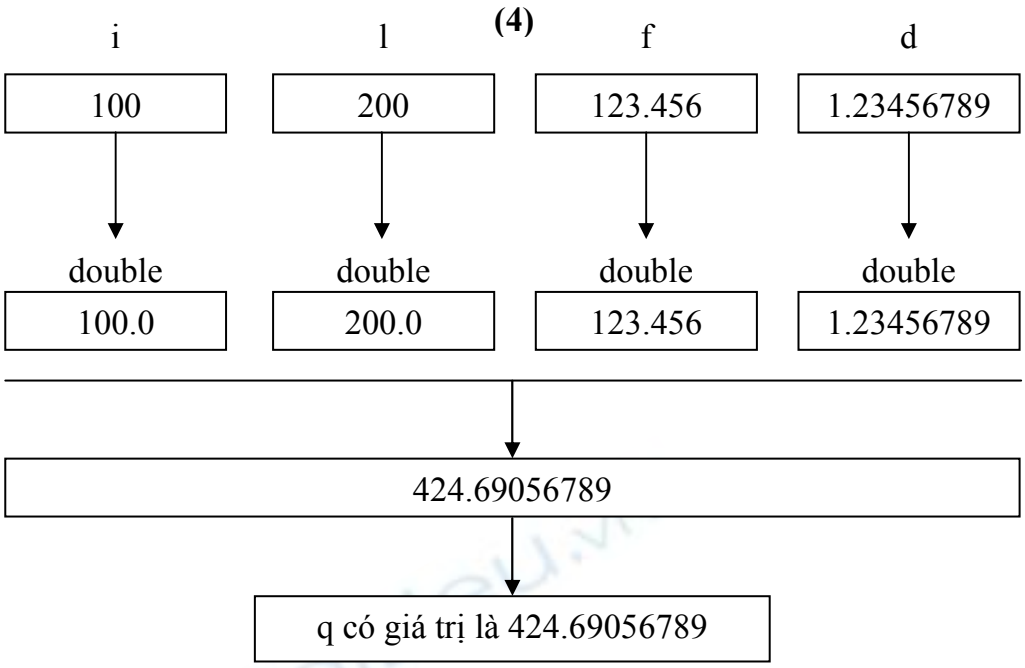
Ví dụ:

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

Khảo sát các lệnh gán sau:

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


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



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

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

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

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

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

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

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

Ví dụ:

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

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

Ví dụ: Khảo sát đoạn lệnh sau

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

Nếu thực hiện lệnh:

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

Nếu thực hiện lệnh:

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

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

Tương tự, nếu thực hiện lệnh:

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

Nếu thực hiện lệnh:

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

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

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