Chương 2: Biểu thức

1. Biểu thức

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

2. Kiểu dữ liệu (data types)

- C/C++ có 5 kiểu dữ liệu cơ sở:
  - ký tự ***(char)***,
  - số nguyên ***(int)***,
  - số thực ***(float)***,
  - số thực có độ chính xác gấp đôi***(double)***,
  - kiểu vô định ***(void)***.

2. Kiểu dữ liệu (data types)

- Kích thước của kiểu **int** là 16 bits (2 bytes) trên môi trường 16-bits, và 32 bits (4 bytes) trên môi trường 32-bits. 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).

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

3. 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

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

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

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

4. 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

5. 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

```
type variableNames;
```

ng quát là:

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

```
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***.

6. Biến cục bộ *(local variables)*

- khai báo bên trong một hàm.
- Các biến cục bộ chỉ được tham chiếu đến một khối bắt đầu với dấu { và kết thúc với dấu }.
- Tồn tại trong khi khối chứa nó đang thực thi và bị hủy khi khối chứa nó thực thi xong.

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.

6. 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. Ví dụ:

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

from, to: 2 tham số hình thức

total: biến cục bộ của hàm ***sum.***

i: khai báo trong cấu trúc lặp for nên nó là biến cục bộ chỉ tồn tại trong cấu trúc ***for***

6. 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;}
```

Value of gVar= 100
After increased, gVar= 101;
After decreased, gVar= 100;

6. 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 truớc 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;
```

VI.6. 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 chuổi.
- Lưu ý phân biệt hằng chuổi 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

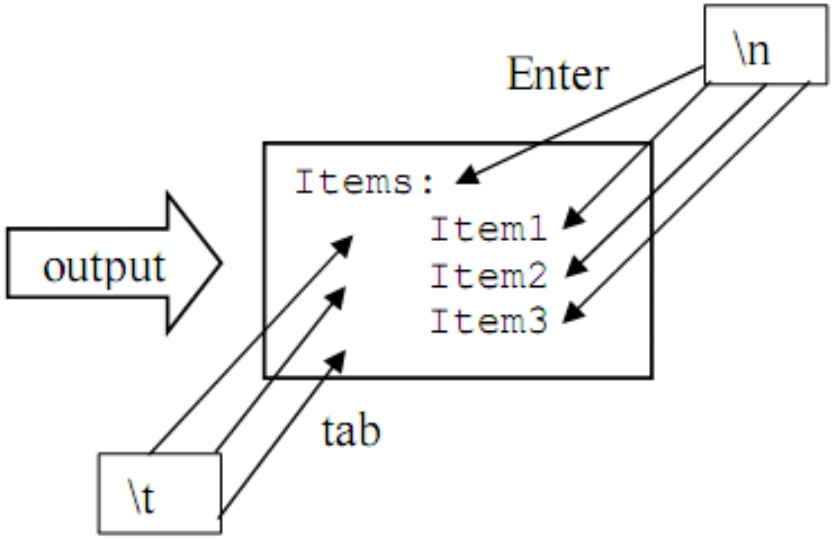
V.7. Hằng ký tự đặc biệt *(escape sequences)*

C/C++ có những hằng ký tự đặc biệt mà không thể biểu diễn nhu những hằng ký tự th là **es** ách cá

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)

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

}
```



VI Toán tử *(operators)*

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

VI.1. T

```
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;
```

VI.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. ðối với câu lệnh gán, giá trị của biểu thức bên phải dấu bằng được tự động chuyển thành kiểu dữ liệu của biến trái dấu bằng.

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

Ví dụ:

```
i = d;
```

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.

```
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)
```

VII. Các toán tử số học (arithmetic operators)

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

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;
```

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

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

- Toán tử quan hệ liên quan đến sự quan hệ của hai giá trị. Toán tử luận lý liên quan đến sự nối kết của những quan hệ.
- Các biểu thức mà dùng toán tử quan hệ và toán tử luận lý được định trị là true (đúng) hoặc false(sai). Trong C/C++, giá trị 0 (zero) được xem là false và giá trị khác 0 (non-zero) được xem là true.

Các toán tử quan hệ gồm:

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

Ví dụ về toán tử quan hệ:

100 < 200 // true

200 == 300 // false

300 != 400 // true

400 >= 500 // false

Các toán tử về luận lý:

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

Bảng chân trị của các toán tử luận lý

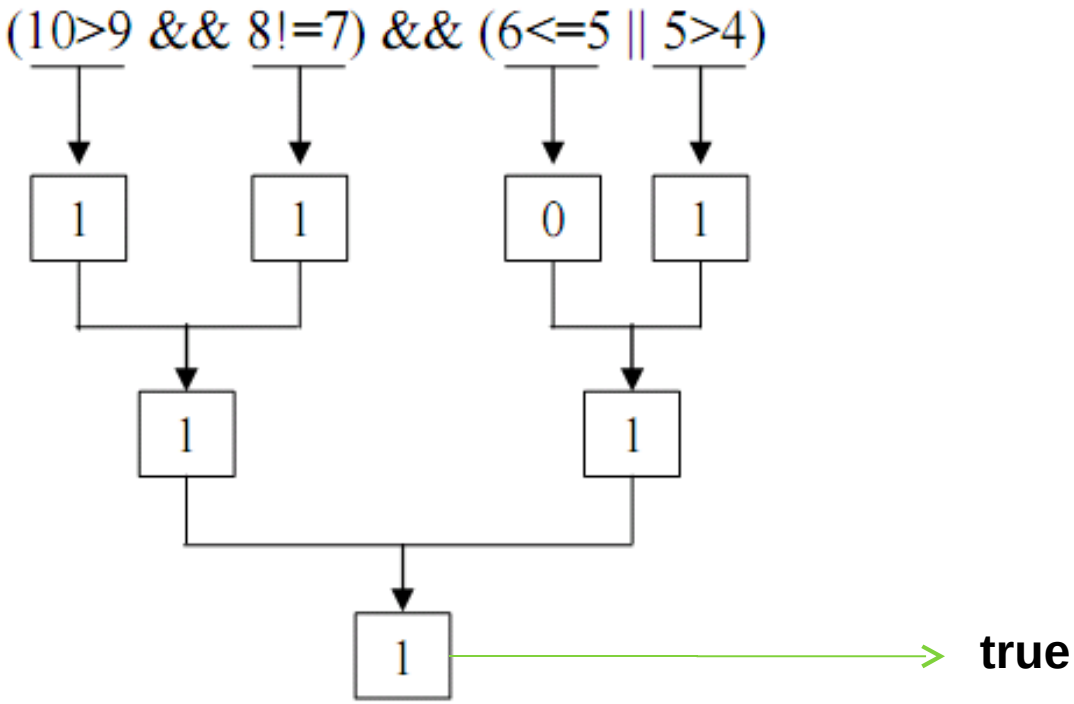
p	q	p && q	p || q	!p
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

- Cả hai toán tử quan hệ và luận lý có độ ưu tiên thấp hơn toán tử số học. Trong một biểu thức có thể có nhiều loại toán tử, thứ tự để định trị biểu thức dựa vào độ ưu tiên của toán tử. Để thay đổi thứ tự định trị ặp dấu ngoặc ức số học.

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

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

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



Toán tử ? (? operator)

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

```
Exp1 ? Exp2 : Exp3;
```

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

Ví dụ:

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

Toán tử sizeof

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

```
sizeof(operand)
```

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

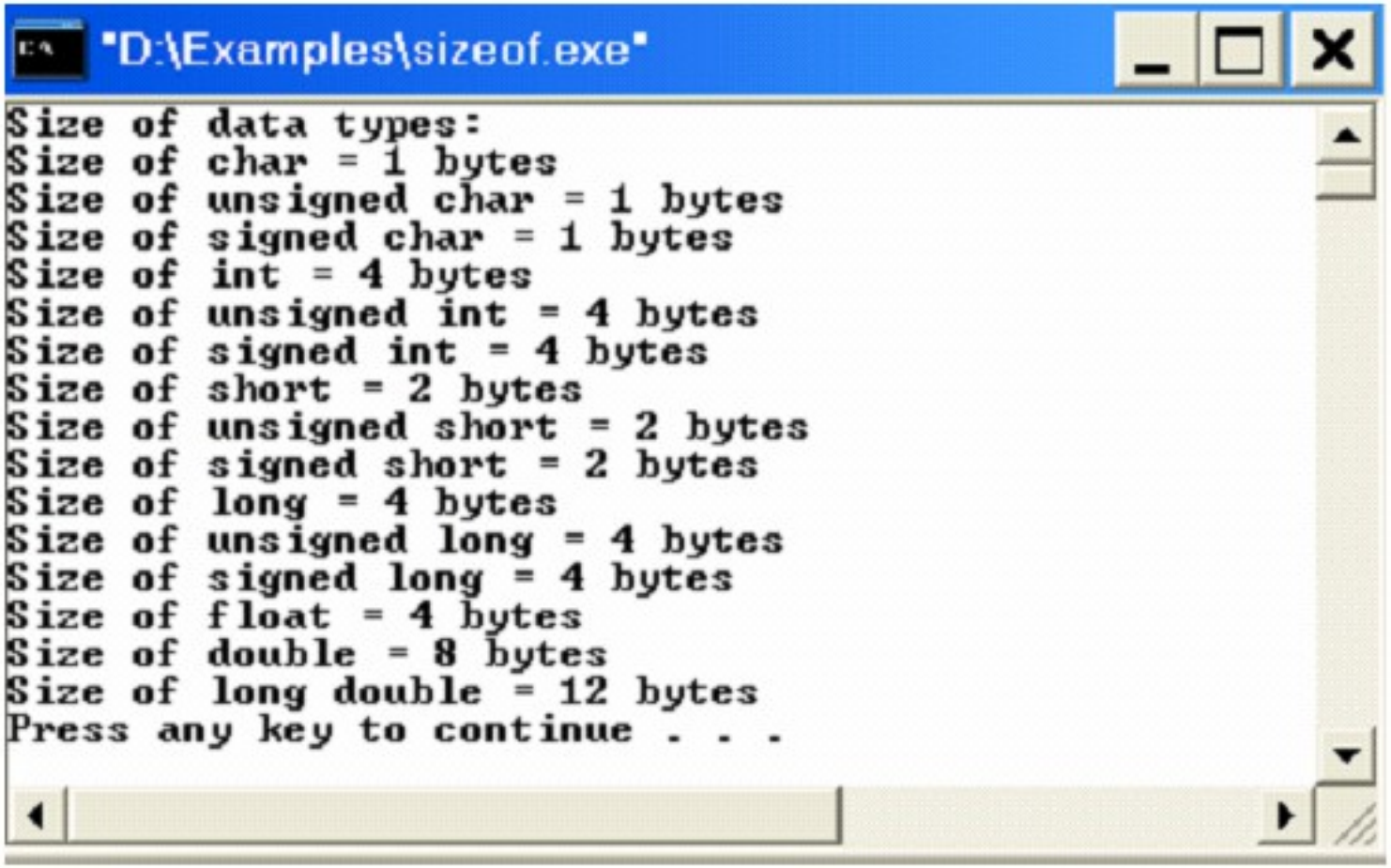
Ví dụ sau cho biết số byte của mỗi kiểu dữ liệu

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

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

}
```

Kết quả khi thực hiện chương trình trên:



Toán tử dấu phẩy (comma operator)

- Toán tử comma buộc các biểu thức cùng với nhau. Biểu thức bên trái của toán tử comma luôn luôn được định trị như **void**, biểu thức bên phải được định trị và trở thành giá trị của biểu thức. Dạng tổng quát của tc `(exp_1, exp_2, ..., exp_n)`

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

Ví dụ:

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

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

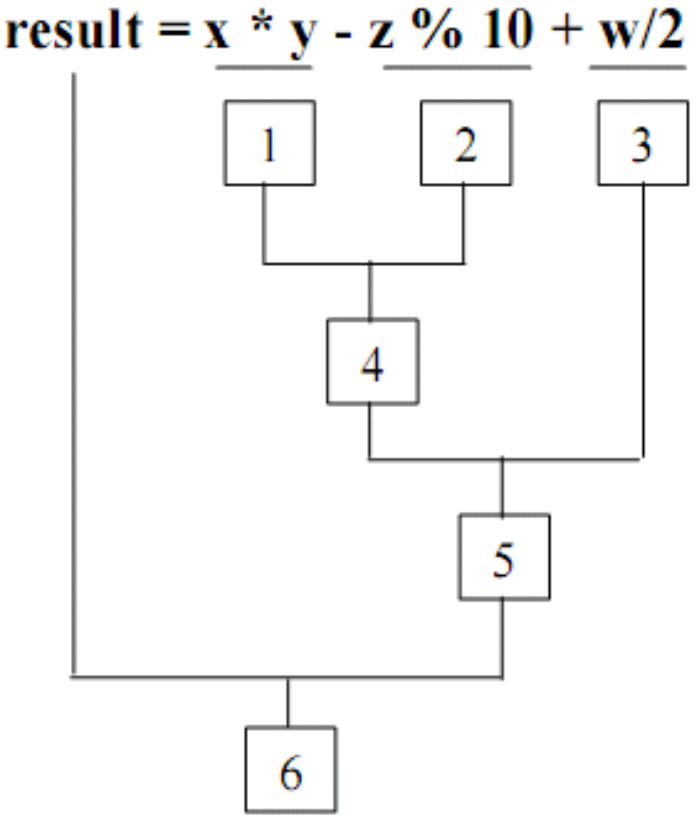
Bảng tóm tắt độ ưu tiên của các toán tử

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

Biểu thức (expressions)

- Một biểu thức trong C/C++ là sự kết hợp của các thành phần nhu toán tử, hằng, biến, và hàm có trả về giá trị.
- Thứ tự định trị của biểu thức tùy thuộc vào độ ưu tiên của các toán tử. Do đó, để viết biểu thức rõ ràng và thực hiện việc định trị đúng theo yêu cầu của lập trình viên ta nên dùng cặp dấu ngoặc tròn () để bao quanh các biểu thức con của biểu thức.

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



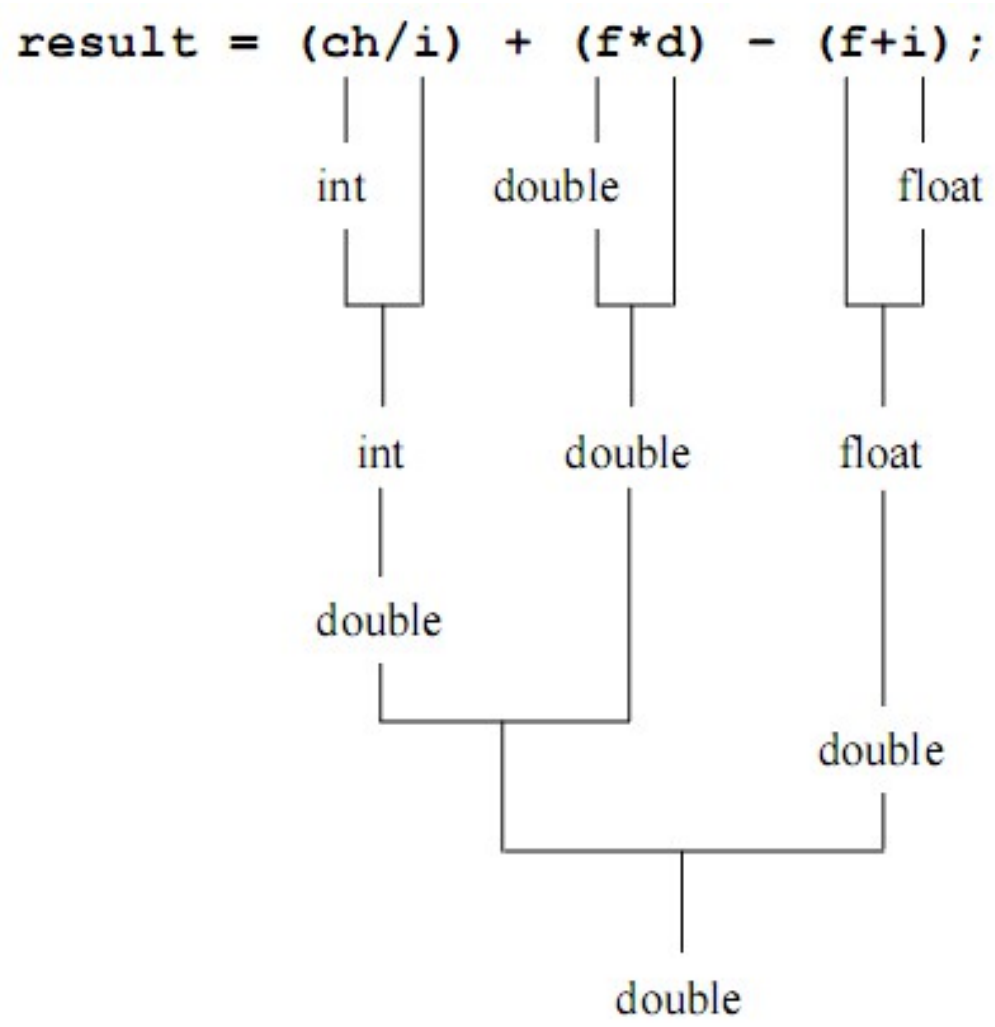

```
result = (x * y) – (z % 10)
+ (w/2);
```

Chuyển đổi kiểu trong các biểu thức

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

Ví dụ:

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



Ép kiểu (casting)

- Casting dùng để ép một biểu thức thành một kiểu theo ý muốn của lập trình viên.

Dạ*ng* ***(type)expression*** *hoặc* ***type(expression)***

type: là tên một kiểu dữ liệu hợp lệ.

Ví dụ:

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

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

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

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

Dạng viết tắt của câu lệnh gán (shorthand assignments)

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

```
variable = variable operator expression;
```

variable: Tên biến

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

expression: Biểu thức

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

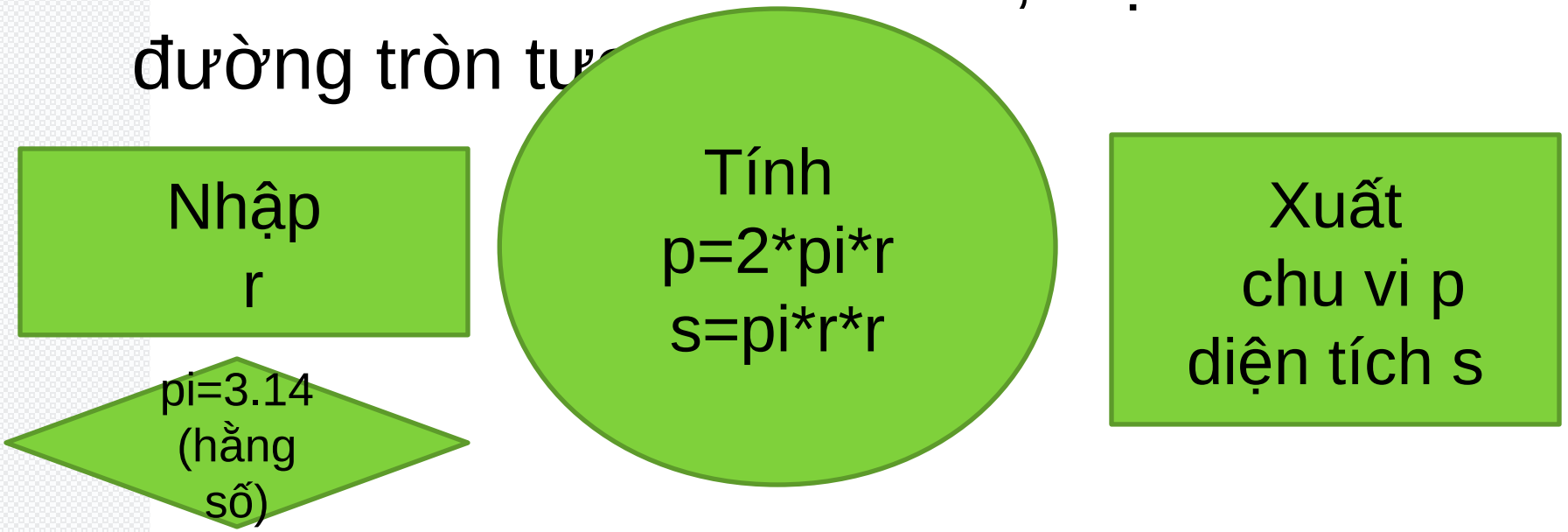
```
variable operator= expression;
```

Ví dụ:

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

Bài tập:

- Bài 1: Nhập vào bán kính r của đường tròn. Tính và xuất ra chu vi, diện tích của đường tròn tư



Cấu trúc 1 chương trình

- r , p, s biến.
- Kiểu dữ liệu: int, float, double?

```
#include <iostream.h>
#include <conio.h>
const float pi=3.14
void main()
{
    float r,p,s;
    cout<<”Nhap vao ban kinh hinh tron”;   cin>>r; // nhap r
    p=r*2*pi; // công thức tinh hinh tron
    s=pi*r*r; // cong thuc tinh dien tich
    cout << “Chu vi hinh tron " << r ; // Xuat chu vi hinh tron
    cout << “Dien tich hinh tron " << s; // Xuat dien tich hinh tron
}
```