CHƯƠNG 3

PHÉP TOÁN VÀ BIỂU THỨC

Nội Dung

- Dữ Liệu Và Kiểu Dữ Liệu
- Biểu thức
- Hằng
- Biến
- Khai báo
- Phép toán
  - Phép toán số học
  - Phép toán luận lý
  - Phép toán quan hệ
  - Phép toán tăng, giảm
  - Phép toán điều kiện

Nội Dung

- Dữ Liệu Và Kiểu Dữ Liệu
- Biểu thức
- Hằng
- Biến
- Khai báo
- Phép toán
  - Phép toán số học
  - Phép toán luận lý
  - Phép toán quan hệ
  - Phép toán tăng, giảm
  - Phép toán điều kiện

Nội Dung

- Phép toán gán
- Độ ưu tiên và thứ tự tính toán

Dữ Liệu và Kiểu Dữ Liệu

- *Thông tin* là tin tức (information) được thông báo, nó cho ta biết một điều gì đó.
- *Dữ liệu* (datum, data) là thông tin dưới dạng ký hiệu, chữ viết, hình ảnh, âm thanh…
- Một *kiểu dữ liệu* là một biểu diễn cụ thể một khái niệm trong thực tế.
- Một kiểu dữ liệu được đặc trưng bởi *kích thước* và *cách sử dụng*.

Dữ Liệu và Kiểu Dữ Liệu

- *Kiểu có sẵn* (built-in data types) được cung cấp sẵn trong ngôn ngữ lập trình, thường là các kiểu cơ bản.
- Kiểu cơ bản có sẵn trong C bao gồm các loại: kiểu số nguyên, số thực, ký tự,…
- Biến và hằng là các đối tượng dữ liệu cơ bản trong một chương trình.

Dữ Liệu và Kiểu Dữ Liệu

- Biến và hằng phải được đặt theo đúng quy ước đặt tên trong C.
- *Tên* hợp lệ trong C được tạo bởi chữ cái và chữ số, chữ số bao gồm các ký tự trong bảng mẫu tự và them dấu gạch dưới "_", tên phải bắt đầu bằng ký tự và không được phép trùng với từ khoá.
- *Từ khoá* (keywords) là các từ dành riêng của C.

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

Từ khoá trong C

Kiểu Dữ Liệu trong C

- Ngôn ngữ C chỉ có một số ít kiểu dữ liệu cơ bản:

`char`	Kích thước 1 byte, biểu diễn khái niệm ký tự.
`int`	Biểu diễn khái niệm số nguyên có dấu, kích thước bằng một từ máy.
`float`	số thực dấu chấm động đơn.
`double`	số thực dấu chấm động kép

- `char` cũng là kiểu nguyên và int cũng có thể dùng như ký tự.

Kiểu Dữ Liệu trong C

- Ngôn ngữ C còn có các từ khoá bổ nghĩa áp dụng cho các kiểu trên, bao gồm short, long, signed, unsigned đi kèm với các kiểu trên.

```
short int a, b;
long int number_of_students;
unsigned int u;
```

- `short, long` áp dụng cho kiểu `int`.
- `signed` áp dụng cho `char` và `int`.

Kiểu Dữ Liệu

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Kiểu nguyên trong C

- *Lưu ý:* Kiểu nguyên có thể bị tràn số.

Kiểu Dữ Liệu

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Kiểu thực trong C

Lưu ý:

- Kiểu thực có thể bị tràn số.
- Kiểu thực có sai số
- Phép toán sizeof cho biết kích thước kiểu dữ liệu

Câu hỏi

- Tại sao kiểu nguyên có thể bị tràn số.
- Tại sao số thực có sai số.
- Lấy một số thực (kiểu float) khởi động bằng 1 rồi chia đôi liên tiếp 100 lần, sau đó nhân đôi trở lại 100 lần. Kết quả thu được là bao nhiêu.

Khái niệm biểu thức

- Một *biểu thức* (expression) là sự kết hợp của các hằng, biến, phép toán, phép gọi hàm để cuối cùng rút ra được một kết quả duy nhất gọi là kết quả của biểu thức.
- Biểu thức có kiểu trả về, ta có thể có biểu thức nguyên, biểu thức thực, biểu thức chuỗi.

Hằng

- Hằng là đại lượng có *giá trị không thay đổi* trong suốt quá trình thực hiện chương trình.
- Trong C có thể có hằng thực sự và hằng ký hiệu.
- Hằng nguyên:

```
1235, 0x1E2, 045
#define MAXLINE 100
const int zero = 0;
421L, 0x12AFL, 50UL
```

Hằng

- Hằng thực:

```
1235.21f, 3218.F, 1e-7f
213.12, 0.002 .07
1.23e-5, 1.22e8
#define PI 3.1415926
```

- Hằng ký tự:

```
'x', 'A', '\n', '\x1b', '\t'
#define ESC '\x1b'
```

Hằng

\a	alert (bell) character	\\	backslash
\b	backspace	\?	question mark
\f	formfeed	\'	single quote
\n	newline	\"	double quote
\r	carriage return	*ooo*	octal number
\t	horizontal tab	\x*hh*	hexadecimal number
\v	vertical tab		

Escape Sequences trong C

Biến

- *Biến* là đại lượng *có thể thay đổi giá trị* trong quá trình thực hiện chương trình.
- Biến trong C phải được khai báo trước khi dùng.

Khai báo

- *Khai báo* là giới thiệu ra một tên và các thông tin đi kèm theo tên đó.
- *Khai báo biến* giới thiệu tên biến và kiểu của biến.
- Biến phải được khai báo trước khi dùng.

```
int lower, upper, step;
char c;
```

Khai báo

- Có thể khởi động biến khi khai báo:

```
char esc = '\x1b';
int i = 0;
int limit = MAXLINE+1;
float eps = 1.0e-5;
```

- Có thể qui định giá trị của biến không được phép thay đổi:

```
const double e = 2.71828182845905;
```

Phép toán

- Một *phép toán* là một ký hiệu mà nó thao tác trên dữ liệu.
- Dữ liệu được thao tác gọi là toán hạng.
- Phép toán có thể hai ngôi (có hai toán hạng) hoặc một ngôi (một toán hạng).
- Trong C có một phép toán đặc biệt `? :` là phép toán 3 ngôi.

Phép toán số học

- *Phép toán số học* là một phép toán cho kết quả thuộc kiểu số (thực hoặc nguyên). Các phép toán số học trong C bao gồm: `+, -, *, /` và `%`.
- Phép toán *, / có cùng độ ưu tiên, cao hơn các phép toán +, - cũng có cùng độ ưu tiên.
- Ví dụ về phép toán %:

```
if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)
        printf("%d is a leap year\n", year);
else
        printf("%d is not a leap year\n", year);
```

Phép toán số học

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division(modulo division)

Ví dụ minh hoạ phép toán số học

```c
// C Program to demonstrate the working of arithmetic operators
#include <stdio.h>
int main()
{
    int a = 9,b = 4, c;

    c = a+b;
    printf("a+b = %d \n",c);

    c = a-b;
    printf("a-b = %d \n",c);

    c = a*b;
    printf("a*b = %d \n",c);

    c=a/b;
    printf("a/b = %d \n",c);

    c=a%b;
    printf("Remainder when a divided by b = %d \n",c);

    return 0;
}
```

Ví dụ minh hoạ phép toán số học

Output

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1
```

Phép toán quan hệ

- *Phép toán quan hệ* cho kết quả so sánh hai biểu thức (số) .
- Các phép toán quan hệ trong C là: `<, >, <=, >=, ==, !=`
- Các phép toán: `<, >, <=, >=` có cùng độ ưu tiên và cao hơn hai phép toán `==, !=`
- Các phép toán: `<, >, <=, >=` có cùng độ ưu tiên và cao hơn hai phép toán `==, !=`
- C không có kiểu boolean như pascal, một biểu thức có kết quả khác 0 được xem là đúng, bằng 0 được xem là sai.

Phép toán quan hệ

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 returns 0
>	Greater than	5 > 3 returns 1
<	Less than	5 < 3 returns 0
!=	Not equal to	5 != 3 returns 1
>=	Greater than or equal to	5 >= 3 returns 1
<=	Less than or equal to	5 <= 3 return 0

Ví dụ minh hoạ phép toán quan hệ

```c
int main()
{
   int a = 5, b = 5, c = 10;
   printf("%d == %d = %d \n", a, b, a == b); // true
   printf("%d == %d = %d \n", a, c, a == c); // false
   printf("%d > %d = %d \n", a, b, a > b); //false
   printf("%d > %d = %d \n", a, c, a > c); //false
   printf("%d < %d = %d \n", a, b, a < b); //false
   printf("%d < %d = %d \n", a, c, a < c); //true
   printf("%d != %d = %d \n", a, b, a != b); //false
   printf("%d != %d = %d \n", a, c, a != c); //true
   printf("%d >= %d = %d \n", a, b, a >= b); //true
   printf("%d >= %d = %d \n", a, c, a >= c); //false
   printf("%d <= %d = %d \n", a, b, a <= b); //true
   printf("%d <= %d = %d \n", a, c, a <= c); //true
   return 0;
}
```

Ví dụ minh hoạ phép toán quan hệ

Output

```
5 == 5 = 1
5 == 10 = 0
5 > 5 = 0
5 > 10 = 0
5 < 5 = 0
5 < 10 = 1
5 != 5 = 0
5 != 10 = 1
5 >= 5 = 1
5 >= 10 = 0
5 <= 5 = 1
5 <= 10 = 1
```

Phép toán luận lý

- *Phép toán luận lý* cho kết quả 1 (đúng) hoặc 0 (sai). Phép toán luận lý thường được dùng trong các phát biểu rẽ nhánh hoặc lặp. Các phép toán luận lý trong C là: &&, ||, !

Operator	Meaning of Operator	Example
&&	Logial AND. True only if all operands are true	If c = 5 and d = 2 then, expression `((c == 5) && (d > 5))` equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression `((c == 5) || (d > 5))` equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression `! (c == 5)` equals to 0.

Ví dụ minh hoạ phép toán luận lý

```
int main()
{
  int a = 5, b = 5, c = 10, result;
  result = (a == b) && (c > b);
  printf("(a == b) && (c > b) equals to %d \n", result);
  result = (a == b) && (c < b);
  printf("(a == b) && (c < b) equals to %d \n", result);
  result = (a == b) || (c < b);
  printf("(a == b) || (c < b) equals to %d \n", result);
  result = (a != b) || (c < b);
  printf("(a != b) || (c < b) equals to %d \n", result);
  result = !(a != b);
  printf("!(a != b) equals to %d \n", result);
  result = !(a == b);
  printf("!(a == b) equals to %d \n", result);
  return 0;
}
```

Phép toán tăng và giảm

- *Phép toán* ++ trả về kết quả bằng với toán hạng cộng thêm 1 và phép toán -- trả kết quả bằng với toán hạng trừ 1.
- ++ và -- có thể dùng như tiền tố hay hậu tố.

```
++a, --b
a++, b--
```

- *Lưu ý:* Khác biệt ++, -- khi sử dụng như tiền tố và khi sử dụng như hậu tố?

Ví dụ minh hoạ phép tăng và giảm

```
int main()
{
    int a = 10, b = 100;
    float c = 10.5, d = 100.5;
    printf("++a = %d \n", ++a);
    printf("--b = %d \n", --b);
    printf("++c = %f \n", ++c);
    printf("--d = %f \n", --d);
    return 0;
}
```

Output

```
++a = 11
--b = 99
++c = 11.500000
++d = 99.500000
```

Ví dụ minh hoạ phép tăng và giảm

```
int main()
{
    int a = 10, x = a;
    printf("a = %d, x = %d \n", x, a);
    x = ++a;
    printf("a = %d, x = %d \n", x, a);
    a = 10; x = a;
    printf("a = %d, x = %d \n", x, a);
    x = a++;
    printf("a = %d, x = %d \n", x, a);
    return 0;
}
```

Phép gán

- *Phép gán* dùng để gán giá trị cho một biến.

```
x = a+b*5;
```

- *Phép gán kết hợp* kết hợp một phép toán nhị phân với phép gán:

```
x += 5;          /* x = x+5; */
a *= expr;       /* a = a * (expr)
a *= b + 5;      /* a = a * (b+5);
```

- Phép gán và phép gán kết hợp đều có *giá trị trả về*.

```
x = (a = 5);    y = (a += 7);
```

Phép toán điều kiện

- *Phép toán 3 ngôi* `? :`

```
expr1 ? expr2 : expr3
```

Cho kết quả expr2 nếu expr1 đúng, cho kết quả expr3 nếu expr1 sai.

```
a = x > y ? x : y;
```

Độ ưu tiên và thứ tự tính toán

Thứ tự tính toán trong một biểu thức:

1. Độ ưu tiên của phép toán
2. Trái sang phải hoặc phải sang trái tuỳ phép toán.
3. Có thể qui định lại thứ tự tính toán bằng dấu (), các biểu thức trong 2 dấu () sẽ được tính trước.

Operators	Associativity
`() [] -> .`	left to right
`! ~ ++ -- + - *` (*type*) `sizeof`	right to left
`* / %`	left to right
`+ -`	left to right
`<< >>`	left to right
`< <= > >=`	left to right
`== !=`	left to right
`&`	left to right
`^`	left to right
`|`	left to right
`&&`	left to right
`||`	left to right
`?:`	right to left
`= += -= *= /= %= &= ^= |= <<= >>=`	right to left
`,`	left to right

Độ ưu tiên và thứ tự tính toán

- Các phép toán một ngôi là phép toán có độ ưu tiên cao, có thứ tự tính toán từ *phải sang trái*.
- Phép toán truy xuất thành phần cấu trúc (dấu . và dấu ->) là các phép toán 2 ngôi có độ ưu tiên cao hơn phép toán 1 ngôi.
- Các phép toán hai ngôi còn lại có độ ưu tiên thấp hơn phép toán 1 ngôi, hầu hết có thứ tự tính toán từ trái sang phải, ngoại trừ các phép gán.

Độ ưu tiên và thứ tự tính toán

- Các phép toán số học có độ ưu tiên hơn các phép toán 2 ngôi khác, trong đó *, / có độ ưu tiên cao hơn +, -.
- Các phép toán so sánh có độ ưu tiên thấp hơn phép toán số học. Nghĩa là các biểu thức số học được tính trước khi được so sánh.
- Phép toán gán có độ ưu tiên gần như thấp nhất (chỉ đứng trên phép toán dấu ,). Nghĩa là biểu ở bên phải phép toán gán được tính xong xuôi mới đem gán vào biến ở bên tay trái.

Ví dụ minh hoạ thứ tự tính toán

```c
#include <stdio.h>

main() {

   int a = 20;
   int b = 10;
   int c = 15;
   int d = 5;
   int e;

   e = (a + b) * c / d;       // ( 30 * 15 ) / 5
   printf("Value of (a + b) * c / d is : %d\n",  e );

   e = ((a + b) * c) / d;     // (30 * 15 ) / 5
   printf("Value of ((a + b) * c) / d is  : %d\n" ,  e );

   e = (a + b) * (c / d);    // (30) * (15/5)
   printf("Value of (a + b) * (c / d) is  : %d\n",  e );

   e = a + (b * c) / d;      //  20 + (150/5)
   printf("Value of a + (b * c) / d is  : %d\n" ,  e );

   return 0;
}
```

Tóm tắt

- Một *kiểu dữ liệu* là một biểu diễn cụ thể một khái niệm trong thực tế. Kiểu dữ liệu cho ta biết cách sử dụng.
- C cung cấp một số kiểu cơ bản: số nguyên, số thực, ký tự, …
- Một *biểu thức* (expression) là sự kết hợp của các hằng, biến, phép toán, phép gọi hàm để cuối cùng rút ra được một *kết quả duy nhất* gọi là kết quả của biểu thức.

Tóm tắt

- *Hằng* là đại lượng có *giá trị không thay đổi* trong suốt quá trình thực hiện chương trình.
- *Biến* là đại lượng *có thể thay đổi giá trị* trong quá trình thực hiện chương trình. Biến phải được khai báo trước khi dùng.
- Một *phép toán* là một ký hiệu mà nó thao tác trên dữ liệu. Dữ liệu được thao tác gọi là *toán hạng*. Phép toán có thể hai ngôi hoặc một ngôi.

Tóm tắt

- *Thứ tự tính toán* trong một biểu thức được quyết định bằng dấu mở đóng ngoặc (), độ ưu tiên phép toán và tính từ trái sang phải hoặc từ phải sang trái tuỳ theo phép toán.

Q&A