Bài 5

Lập trình C
Các biểu thức

Nội dung

- Kiểu dữ liệu cơ bản
- Bộ hiệu chỉnh kiểu dữ liệu (modifier)
- Định danh (identifier)
- Biến
- Phạm vi khả kiến của định danh (scope)
- Khởi tạo biến
- Bộ định tính biến (qualifier)
- Phép toán
- Biểu thức

Kiểu dữ liệu cơ bản

- Ngôn ngữ C hổ trợ **5 kiểu dữ liệu cơ bản**:
  - char – kí tự, kích thước 1 byte
  - int – số nguyên có dấu, kích thước 2 hoặc 4 byte
  - float – số thực có dấu, kích thước 4 byte
  - double – số thực có dấu, kích thước 8 byte
  - void – kiểu rỗng
- Kích thước và phạm vi của các kiểu dữ liệu có thể khác nhau giữa các CPU và trình biên dịch.
- Ngôn ngữ C chỉ qui định về phạm vi tối thiểu của kiểu dữ liệu.

Bộ hiệu chỉnh kiểu dữ liệu

- Modifiers (data type modifiers) là nhóm các từ dùng để báo cho trình biên dịch biết các đặc tính bổ sung của một kiểu dữ liệu cơ bản khi thực hiện một lệnh khai báo.
- Ngôn ngữ C cung cấp **4 từ hiệu chỉnh** cho các kiểu dữ liệu cơ bản:
  - signed: có dấu
  - unsigned: không dấu
  - short: kích thước lưu trữ nhỏ
  - long: kích thước lưu trữ lớn
- Các từ này có thể được kết hợp với nhau để bổ sung cho các kiểu dữ liệu cơ bản **ngoại trừ kiểu void**.
- Thường sử dụng kiểu **char** hoặc **unsigned char** để lưu giữ giá trị là các **kí tự chữ cái và chữ số**.

Tất cả kiểu dữ liệu theo C chuẩn (1)

Type	Typical Size in Bits	Minimal Range
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	Same as **int**
short int	16	–32,767 to 32,767
unsigned short int	16	0 to 65,535
signed short int	16	Same as **short int**

Tất cả kiểu dữ liệu theo C chuẩn (2)

long int	32	–2,147,483,647 to 2,147,483,647
long long int	64	$-(2^{63}-1)$ to $2^{63}-1$ (Added by C99)
signed long int	32	Same as **long int**
unsigned long int	32	0 to 4,294,967,295
unsigned long long int	64	$2^{64}-1$ (Added by C99)
float	32	1E –37 to 1E+37 with six digits of precision
double	64	1E –37 to 1E+37 with ten digits of precision
long double	80	1E –37 to 1E+37 with ten digits of precision

Bảng mã ASCII chuẩn

0	*NUL*	16	*DLE*	32	SPC	48	0	64	@	80	P	96	`	112	p
1	*SOH*	17	*DC1*	33	!	49	1	65	A	81	Q	97	a	113	q
2	*STX*	18	*DC2*	34	"	50	2	66	B	82	R	98	b	114	r
3	*ETX*	19	*DC3*	35	#	51	3	67	C	83	S	99	c	115	s
4	*EOT*	20	*DC4*	36	$	52	4	68	D	84	T	100	d	116	t
5	*ENQ*	21	*NAK*	37	%	53	5	69	E	85	U	101	e	117	u
6	*ACK*	22	*SYN*	38	&	54	6	70	F	86	V	102	f	118	v
7	*BEL*	23	*ETB*	39	'	55	7	71	G	87	W	103	g	119	w
8	*BS*	24	*CAN*	40	(	56	8	72	H	88	X	104	h	120	x
9	*HT*	25	*EM*	41	)	57	9	73	I	89	Y	105	i	121	y
10	*LF*	26	*SUB*	42	*	58	:	74	J	90	Z	106	j	122	z
11	*VT*	27	*ESC*	43	+	59	;	75	K	91	[	107	k	123	{
12	*FF*	28	*FS*	44	,	60	<	76	L	92	\	108	l	124	|
13	*CR*	29	*GS*	45	-	61	=	77	M	93	]	109	m	125	}
14	*SO*	30	*RS*	46	.	62	>	78	N	94	^	110	n	126	~
15	*SI*	31	*US*	47	/	63	?	79	O	95	_	111	o	127	DEL

Định danh

- Định danh (identifier) là cách gọi chung tên của các biến, các hàm và các thành phần do người lập trình định nghĩa.
  - Độ dài của định danh có thể là một kí tự hay nhiều kí tự.
  - Kí tự đầu tiên trong một định danh phải là chữ cái hoặc dấu _. Các kí tự tiếp theo phải là chữ cái, chữ số hoặc dấu _.
- Trong một định danh, có sự phân biệt chữ hoa, chữ thường.
- Định danh không được trùng với các từ dành riêng của ngôn ngữ.

Biến

- Lệnh khai báo biến
  <kiểu dữ liệu> <tên biến>;

```
char c;
int i, j, k;
long double balance, profit, loss;
```

- Biến có thể được khai báo ở ba nơi:
  - Bên trong các hàm (*biến cục bộ*).
  - Nơi định nghĩa các tham số của hàm (*tham số hình thức*).
  - Bên ngoài tất cả các hàm (*biến toàn cục*).

Biến cục bộ (1)

- Là biến được khai báo bên trong một hàm.
- Biến cục bộ
  - Chỉ được sử dụng bởi các câu lệnh trong khối mã nơi đó được khai báo.
  - Chỉ tồn tại trong khi khối mã đang thực thi.

```
void func1(void)
{
  int x;

  x = 10;
}

void func2(void)
{
  int x;

  x = -199;
}
```

Biến cục bộ (2)

```
void f(void)
{
  int t;

  scanf("%d%*c", &t);

  if(t==1) {
    char s[80]; /* this is created only upon
                   entry into this block */
    printf(''Enter name:");
    gets(s);
    /* do something . . . */
  }

  /* s not known here */
}
```


Biến cục bộ (3)

```
#include <stdio.h>

int main(void)
{
  int x;

  x = 10;

  if(x == 10) {
    int x; /* this x hides the outer x */

    x = 99;
    printf("Inner x: %d\n", x);
  }
  printf("Outer x: %d\n", x);

  return 0;
}
```

Tham số hình thức

- Nếu một hàm có sử dụng các đối số, ta phải khai báo các biến để nhận giá trị của các đối số.
- Các biến này được gọi là các tham số hình thức của hàm.
- Tham số hình thức hoạt động như biến cục bộ bên trong hàm.

```
/* Return 1 if c is part of string s; 0 otherwise */
int is_in(char *s, char c)
{
  while(*s)
    if(*s==c) return 1;
    else s++;

  return 0;
}
```


Biến toàn cục (1)

- Là biến được khai báo bên ngoài tất cả các hàm.
- Biến toàn cục
  - Được sử dụng bởi bất kỳ đoạn mã nào của chương trình.
  - Tồn tại trong suốt quá trình chương trình thực thi.
- Các biến toàn cục được lưu giữ trong một vùng bộ nhớ cố định được trình biên dịch dành riêng.

Biến toàn cục (2)

```
#include <stdio.h>
int count;  /* count is global  */

void func1(void);
void func2(void);

int main(void)
{
  count = 100;
  func1();

  return 0;
}

void func1(void)
{
  int temp;

  temp = count;
  func2();
  printf("count is %
d", count); /* will print 100 */
}
```


Phạm vi khả kiến của định danh (1)

- Mô tả phạm vi trong chương trình mà một định danh được biết đến và có thể được tham chiếu.
- Ngôn ngữ C qui định 4 phạm vi khả kiến của định danh:
  - Phạm vi tập tin (file scope).
  - Phạm vi khối mã (block scope).
  - Phạm vi nguyên mẫu hàm (function prototype scope).
  - Phạm vi hàm (function scope).

Phạm vi khả kiến của định danh (2)

- Phạm vi tập tin
  - Bắt đầu ở đầu tập tin và kết thúc ở cuối tập tin.
  - Chỉ đề cập đến các định danh được khai báo bên ngoài tất cả các hàm.
  - Định danh phạm vi tập tin được thấy trong toàn bộ tập tin.
  - Các biến có phạm vi tập tin là toàn cục.
- Phạm vi khối mã
  - Bắt đầu ở dấu mở khối { và kết thúc ở dấu đóng khối } đó.
  - Các tham số của hàm được bao gồm trong phạm vi khối của hàm.
  - Các biến có phạm vi khối là cục bộ trong khối của chúng.

Phạm vi khả kiến của định danh (3)

- Phạm vi nguyên mẫu hàm
  - Định danh được khai báo trong nguyên mẫu hàm; có thể nhìn thấy trong nguyên mẫu.
- Phạm vi hàm
  - Bắt đầu ở dấu mở khối { của hàm và kết thúc ở dấu đóng khối } của hàm đó.
  - Phạm vi hàm chỉ áp dụng cho nhãn (label). Một nhãn được sử dụng làm đích cho câu lệnh ***goto*** và nhãn đó phải trong cùng hàm với ***goto***.

Khởi tạo biến

- Lệnh khởi tạo biến
  <kiểu dữ liệu> <tên biến> = <hằng>;

```
char ch = 'a';
int first = 0;
double balance = 123.23;
```

  - Biến toàn cục chỉ được khởi tạo khi bắt đầu chương trình.
  - Biến cục bộ được khởi tạo khi vào khối mã nơi nó được khai báo.
  - Biến cục bộ không được khởi tạo sẽ nhận giá trị rác trước khi thực hiện phép gán đầu tiên cho nó.
  - Biến toàn cục chưa được khởi tạo sẽ tự động nhận giá trị 0.

Bộ định tính biến

- Qualifier là nhóm các từ dùng để báo cho trình biên dịch biết cách thức các biến có thể được truy cập hay thay đổi giá trị.
- Ngôn ngữ C cung cấp **2 từ định tính**:
  - const – nội dung của biến không thể thay đổi

```
const int a=10;
```

  - volatile – ngăn chặn sự thay đổi nội dung của biến một cách không báo trước, thường được sử dụng trong lập trình nhúng, lập trình đa luồng.

Hằng (1)

- Hằng là các giá trị cố định mà chương trình không thể thay đổi. Hằng có thể thuộc bất kỳ loại dữ liệu cơ bản nào.
- Hằng kiểu số
  - Mặc định, trình biên dịch chọn cho hằng một kiểu dữ liệu tương thích có phạm vi nhỏ nhất để chứa nó.
  - Ta có thể chỉ định kiểu dữ liệu cho hằng bằng hậu tố F (float), L (long), U (unsigned).

Kiểu dữ liệu	Hằng
int	1 123 21000 -123
long int	35000L -34L
unsigned int	10000U 987u 40000U
float	123.23F 4.34e-3f
double	123.23 1.0 -0.9876324
long double	1001.2L

Hằng (2)

- Hằng kiểu kí tự
  - Hằng kí tự được đặt giữa cặp dấu nháy đơn: ***'A'***
- Hằng chuỗi kí tự
  - Một dãy các kí tự được đặt giữa cặp dấu nháy kép: ***"C Programming"***

Phép toán

- Ngôn ngữ C cung cấp 4 nhóm phép toán chính:
  - Phép toán số học.
  - Phép toán so sánh.
  - Phép toán logic.
  - Phép toán xử lý bit.
- Ngoài ra còn có một số phép toán đặc biệt
  - Phép gán.
  - Các phép toán khác.

Phép gán (1)

- Sử dụng trong các biểu thức. Dạng chung của phép gán là
  <tên biến> = <biểu thức>
- Chuyển kiểu trong phép gán
  - Giá trị của vế phải (biểu thức) của phép gán được chuyển đổi thành kiểu của vế trái (biến đích).

```
int x;
char ch;
float f;

void func(void)
{
   ch = x;     // line 1
   x = f;      // line 2
   f = ch;     // line 3
   f = x;      // line 4
}
```

Phép gán (2)

Target Type	Expression Type	Possible Info Loss
signed char	char	If value > 127, target is negative
char	short int	High-order 8 bits
char	int (16 bits)	High-order 8 bits
char	int (32 bits)	High-order 24 bits
char	long int	High-order 24 bits
short int	int (16 bits)	None
short int	int (32 bits)	High-order 16 bits
int (16 bits)	long int	High-order 16 bits
int (32 bits)	long int	None
long int (32 bits)	long long int (64 bits)	High-order 32 bits (applies to C99 only)
int	float	Fractional part and possibly more
float	double	Precision, result rounded
double	long double	Precision, result rounded

Phép gán (3)

- **Dùng nhiều phép gán trong một câu lệnh**
  - Để gán nhiều biến cùng một giá trị.

```
x = y = z = 0;
```

- **Phép gán phức hợp**
  - Là một biến thể của phép gán giúp đơn giản hóa viết mã chương trình cho một kiểu phép gán nhất định.
  - Áp dụng cho tất cả các phép toán hai ngôi. Các câu lệnh gán có dạng thức
    var = var <phép toán hai ngôi> <biểu thức>
    có thể viết lại như
    var <phép toán hai ngôi>= <biểu thức>

```
x = x + 20;  có thể viết  x += 20;
```

Phép toán số học (1)

- Ngôn ngữ C có các phép toán số học sau:

Phép toán	Thao tác
-	Trừ, hoặc trừ một ngôi (đổi dấu)
+	Cộng
*	Nhân
/	Chia
%	Lấy số dư trong phép chia nguyên (modulo)
--	Giảm đơn vị
++	Tăng đơn vị

- Phép / áp dụng cho các kiểu int hoặc char thực hiện lấy thương trong phép chia nguyên.

Phép toán số học (2)

- Phép % không áp dụng cho các kiểu float, double.

```
int x, y;

x = 5;
y = 2;

printf("%d ", x/y);  /* will display 2 */
printf(''%d ", x%y);  /* will display 1, the remainder of
                         the integer division */
```

- Phép - một ngôi thực hiện nhân toán hạng của nó với -1.

```
int a = 10, x;
x = -a;       // multiply a by -1,
              // x receives the value -10
```

Phép toán số học (3)

- Phép toán ++ và -- thực hiện cộng 1 vào và trừ 1 từ toán hạng của nó.

```
x = x + 1;  có thể viết  ++x;
x = x – 1;  có thể viết  x--;
```

- Có thể sử dụng dạng tiền tố hoặc hậu tố với toán hạng.

```
x = x + 1;  có thể viết  ++x;  hoặc  x++;
```


Phép toán số học (4)

- Chú ý sự khác biệt giữa dạng tiền tố và hậu tố khi sử dụng phép toán ++ và -- trong biểu thức lớn.

```
x = 10;   // increment x by 1, then assign to y
y = ++x;  // the value of x is 11, and y is 11
```

```
x = 10;   // assign x to y, then increment x by 1
y = x++;  // the value of x is 11, and y is 10
```

- Độ ưu tiên của các phép toán số học

Độ ưu tiên ↑	Tính liên kết
++ --	trái sang phải
- (trừ một ngôi)	trái sang phải
* / %	trái sang phải
+ -	trái sang phải

Phép toán quan hệ và logic (1)

- **Phép toán quan hệ** thực hiện so sánh các giá trị với nhau.
- Ngôn ngữ C cung cấp các phép toán quan hệ sau

Phép toán	Thao tác
>	So sánh lớn hơn
>=	So sánh lớn hơn hoặc bằng
<	So sánh nhỏ hơn
<=	So sánh nhỏ hơn hoặc bằng
==	So sánh bằng
!=	So sánh không bằng

- Kết quả của một phép toán quan hệ là một trong hai giá trị ĐÚNG và SAI.
  - ***Trong C, ĐÚNG là một giá trị khác 0, SAI là bằng 0.***

Phép toán quan hệ và logic (2)

- **Phép toán logic** thực hiện kết nối (logic) các phép toán quan hệ.
- Ngôn ngữ C cung cấp các phép toán logic sau

Phép toán	Thao tác
&&	AND logic
||	OR logic
!	NOT logic

- Trong C **không có phép toán XOR logic**. Tuy nhiên, ta có thể thực hiện thao tác XOR logic theo cách sau

```
a XOR b = (a || b) && !(a && b)
```

Phép toán quan hệ và logic (3)

- Kết quả của một phép toán logic cũng là một trong hai giá trị ĐÚNG và SAI.
- Bảng chân trị của các phép toán logic

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

Phép toán quan hệ và logic (4)

- Độ ưu tiên của các phép toán quan hệ và logic.

Độ ưu tiên ↑	Tính liên kết
!	trái sang phải
> >= < <=	trái sang phải
== !=	trái sang phải
&&	trái sang phải
||	trái sang phải

- Có thể kết hợp các phép toán quan hệ và logic trong cùng một biểu thức logic.
  - ***Trong C, kết quả của biểu thức logic là một trong hai giá trị 1 và 0.***

```
10 > 5 && !(10 < 9) || 3 <= 4  // the result
                               // is 1 (TRUE)
```

Phép toán xử lý bit (1)

- **Phép toán bit thực hiện thao tác** kiểm tra, thiết lập hoặc dịch chuyển các bit thật của một vùng nhớ có kích thước 1 byte hoặc 1 word (2 bytes) tương ứng **với các kiểu cơ bản char, int hoặc các biến thể của chúng**.
- ***Không thể áp dụng phép toán bit trên các kiểu float, double.***
- Ngôn ngữ C cung cấp các phép toán bit sau

Phép toán	Thao tác
&	AND bit
|	OR bit
^	XOR bit
~	Đảo bit
>>	Dịch phải bit
<<	Dịch trái bit

Phép toán xử lý bit (2)

- Bảng chân trị của các phép toán &, |, ~, ^

p	q	p & q	p | q	~p	p ^ q
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

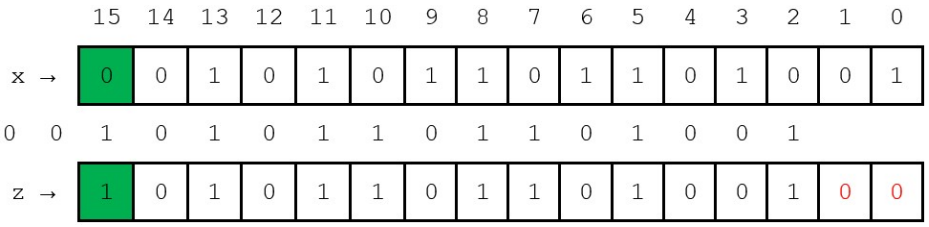
```
int x = 11113, y = 5550, z;
z = x & y;    // the result of z is 296
```

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x →	0	0	1	0	1	0	1	1	0	1	1	0	1	0	0	1
&																
y →	0	0	0	1	0	1	0	1	1	0	1	0	1	1	1	0
z →	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0

Phép toán xử lý bit (3)

- Phép toán <<

```
int x = 11113, z;
z = x << 2;   // the result of z is -21048
```

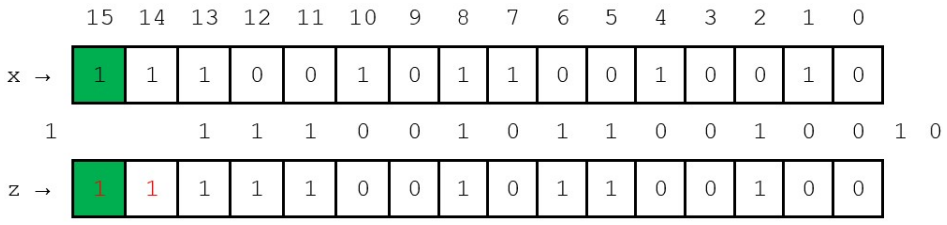


- n << k tương đương n * 2^k
  - Trường hợp trên z = -21048 là do 11113 * 2^2 = 44452 (vượt phạm vi kiểu int của biến z).

Phép toán xử lý bit (4)

- Phép toán >>

```
int x = -6766, z;
z = x >> 2;   // the result of z is -1692
```



- n >> k tương đương n / 2^k.

Một số phép toán khác

- Ngôn ngữ C còn cung cấp một số phép toán tiện dụng khác

Phép toán	Thao tác
()	Tăng mức độ ưu tiên cho các phép toán bên trong
& *	Thao tác trên các con trỏ
[]	Lập chỉ mục cho mảng
?	Thay thế cho một dạng của lệnh lựa chọn
,	Xâu chuỗi một số biểu thức lại với nhau (thường sử dụng trong lệnh lặp for)
(*type*)	Chuyển đổi kiểu dữ liệu

- Phép toán dấu phẩy

```
int x, y;
x = (y = 3, y + 1);     // first assignes y the value 3,
                        // and then assigns x the value 4
```


Thứ tự ưu tiên của tất cả các phép toán

Độ ưu tiên ↑	Tính liên kết
() [] -> .	- Tất cả các phép toán, ngoại trừ các phép toán một ngôi, các phép toán gán và phép toán ?, liên kết từ trái sang phải.
! ~ ++ -- - (*type*) * & sizeof	
* / %	
+ -	
<< >>	
< <= > >=	
== !=	- Các phép toán một ngôi (* & -), các phép toán gán và phép toán ? liên kết từ phải sang trái
&	
^	
|	
&&	
||	
?	
= += -= *= /=	
,	

Biểu thức

- **Biểu thức trong ngôn ngữ C** là sự kết hợp hợp lệ của các phép toán, các hằng, các biến và các hàm.
  - Hầu hết các biểu thức tuân theo các qui tắc chung của đại số.
- Ước tính biểu thức
  - Ta có thể ước tính giá trị của một biểu thức dựa trên độ ưu tiên và tính liên kết của các phép toán trong biểu thức.
  - Thực tế, ngôn ngữ C không chỉ định thứ tự ước tính các biểu thức con trong một biểu thức. Điều này giúp trình biên dịch có thể tự do sắp xếp lại biểu thức để tạo ra mã đối tượng tối ưu hơn.

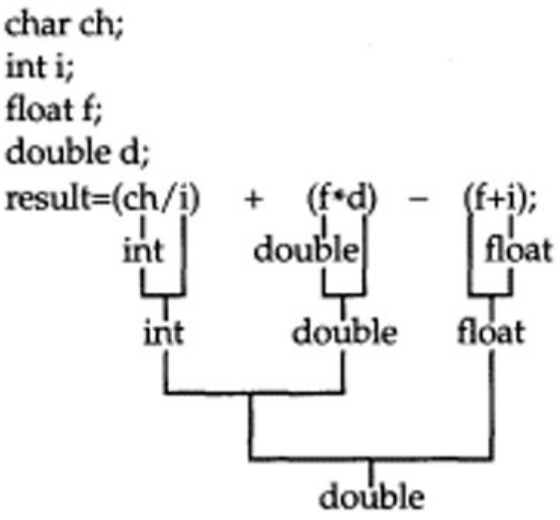
```
x = f1() + f2();
```


Chuyển kiểu trong biểu thức (1)

- Nếu biểu thức chứa các toán hạng (biến, hằng, hàm) có kiểu dữ liệu khác nhau thì trình biên dịch sẽ tự động chuyển đổi chúng về cùng một kiểu.
- Quá trình chuyển đổi kiểu như sau
  - Trước tiên, các giá trị kiểu char và short int được chuyển thành int.
  - Sau đó thực hiện chuyển đổi cho từng phép toán theo logic sau

IF an operand is a **long double**
THEN the second is converted to **long double**
ELSE IF an operand is a **double**
THEN the second is converted to **double**
ELSE IF an operand is a **float**
THEN the second is converted to **float**
ELSE IF an operand is an **unsigned long**
THEN the second is converted to **unsigned long**
ELSE IF an operand is **long**
THEN the second is converted to **long**
ELSE IF an operand is **unsigned int**
THEN the second is converted to **unsigned int**

Chuyển kiểu trong biểu thức (2)



Ép kiểu trong biểu thức

- Có thể chuyển đổi kiểu của một biểu thức bằng phép toán (*type*), cách chuyển đổi này còn được gọi là ép kiểu (type casting).
- Dạng chung của ép kiểu như sau
  (<tên kiểu>) <biểu thức>

```
(float) x/2
```