


Nguyễn Phúc Khải

CHƯƠNG 7: *CÁC THÀNH PHẦN CƠ BẢN VÀ CÁC KIỂU DỮ LIỆU CỦA C*

Các nội dung:

- Danh hiệu
- Các kiểu dữ liệu chuẩn của C
- Hằng *(constant)*
- Biến *(variable)*
- Biểu thức
- Các phép toán của C
- Cấu trúc tổng quát của một chương trình C
- Bài tập cuối chương

DANH HIỆU

- Danh hiệu là tên của hằng, biến, hàm... hoặc các ký hiệu đã được quy định đặc trưng cho một thao tác nào đó.
- Danh hiệu có hai loại: ký hiệu và danh hiệu.

DANH HIỆU

- **Ký hiệu (symbol)** là các dấu đã được C quy định để biểu diễn cho một thao tác nào đó.
- Nếu dùng ***một dấu*** để biểu diễn cho một thao tác thì ta có ký hiệu đơn (single symbol).
  - Ví dụ: +, -, *, /, %, =, >, <
- Nếu dùng ***hai dấu*** trở lên biểu diễn cho một thao tác thì ta có ký hiệu kép (compound symbol).
  - Ví dụ: ==, >=, <=, /*, */, ++, --, &&, ||, ...

DANH HIỆU

- **Danh hiệu (Identifier)** là các từ khóa của ngôn ngữ hoặc tên của các hằng, biến, hàm trong C. Danh hiệu bao hàm ***từ khóa*** và ***danh hiệu***.
- ***Từ khóa (keyword)*** là các danh hiệu mà C đã định nghĩa sẵn cho lập trình viên sử dụng để thiết kế chương trình, tập các từ khóa của C sẽ được liệt kê trong phần phụ lục.
- Ví dụ: ***if***, ***for***, ***while***...

DANH HIỆU

- Danh hiệu là tên của các hằng, biến, hàm...
- Nếu các hằng, biến, hàm... này do C đã khai báo và thiết kế sẵn thì các danh hiệu có được gọi là các ***danh hiệu chuẩn***.
  - Ví dụ: main, scanf, printf...
- Nếu các hằng, biến, hàm... này do lập trình viên khai báo và định nghĩa trong quá trình thiết kế chương trình thì các danh hiệu đó được gọi là các ***danh hiệu không chuẩn***.
  - Ví dụ: a, b, delta

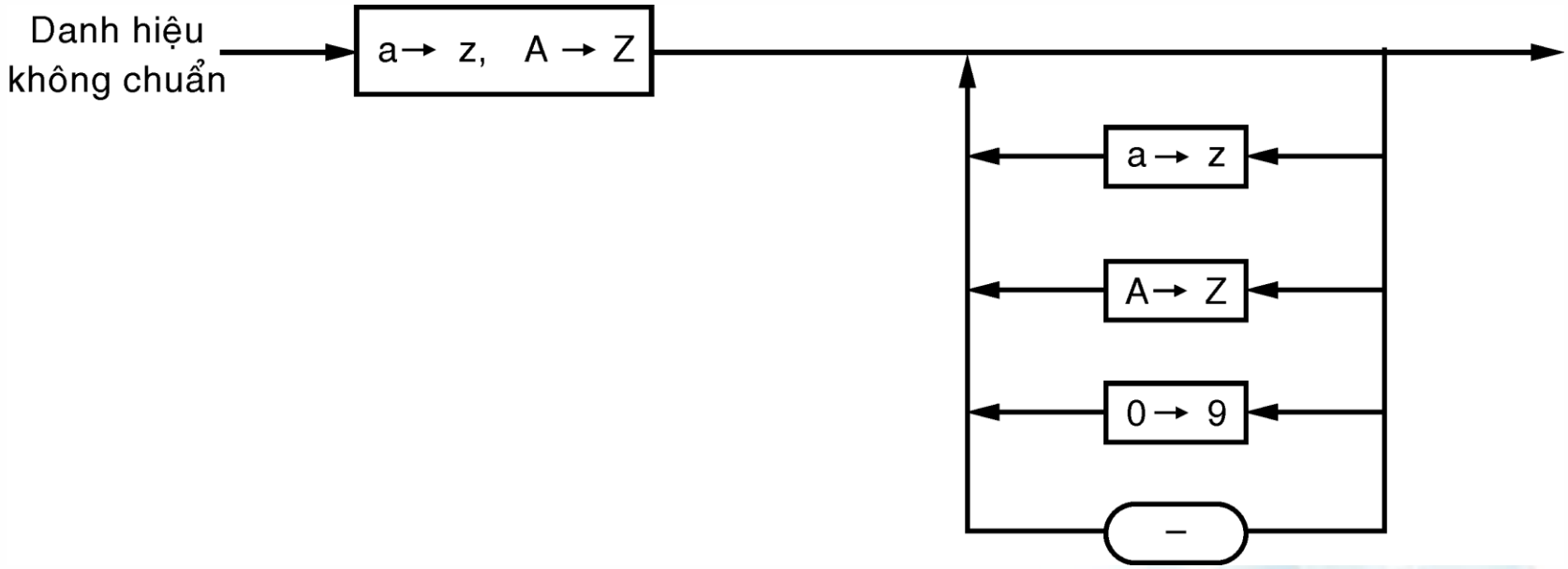
DANH HIỆU

- Chú ý rằng C là một ngôn ngữ nhạy cảm với sự phân biệt giữa ký tự hoa và ký tự thường, do đó khi viết ***While*** sẽ hoàn toàn phân biệt với ***while***.
- Các từ khóa của C đều ở dạng chữ thường.

DANH HIỆU

- Nguyên tắc đặt tên của danh hiệu không chuẩn cũng cần phải được nêu cụ thể:
  - Danh hiệu không chuẩn không trùng với từ khóa
  - Danh hiệu không chuẩn không trùng với danh hiệu chuẩn

DANH HIỆU



Sơ đồ cú pháp cho danh hiệu không chuẩn

DANH HIỆU

- *Ví dụ:*
  - Main ?
  - -batdau ?
  - _batdau ?
  - 2thang9 ?
  - ket thuc ?

DANH HIỆU

- Mỗi bộ dịch C sẽ có quy định về chiều dài danh hiệu khác nhau:
  - Bộ dịch C/C++ thì danh hiệu có thể dài tùy ý
  - Tuy nhiên trong các bộ dịch Borland C/C++ có quy định ***một giá trị xác định số ký tự đầu có nghĩa để phân biệt*** sự giống nhau và khác nhau giữa hai danh hiệu.
    - Trong Turbo C 2.0, giá trị này là 31
    - Trong Borland C++ 5.02, giá trị này là 55.

DANH HIỆU

- Ví dụ:

ket_thuc_vong_lap_in_ra_ky_tu_khoang_trang

ket_thuc_vong_lap_in_ra_k

CÁC KIỂU DỮ LIỆU CỦA C

- Bốn kiểu dữ liệu chuẩn: char, int, float và double, mỗi kiểu sẽ có yêu cầu về bộ nhớ và tầm trị như sau:

KIỂU	KÍCH THƯỚC	TẦM TRỊ BIỂU DIỄN
char	8 bit	-128 .. + 127
int	16 bit	- 32768 .. + 32767
float	32 bit	- 3.4E37 .. 3.4E+38
double	64 bit	- 1.7E307.. 1.7E+308

CÁC KIỂU DỮ LIỆU CỦA C

- **Kiểu char**
- **char** là kiểu nguyên một byte, kiểu này có thể được sử dụng để khai báo biến và có thể giữ một ký tự hoặc một giá trị 8 bit.
- Mỗi bộ dịch C sẽ có quy định khác nhau về tầm trị của kiểu char, đối với bộ dịch TURBO C VERSION 2.0 kiểu char là kiểu có dấu.

CÁC KIỂU DỮ LIỆU CỦA C

Ví dụ: Biến kiểu char lưu trị hằng ký tự

```
#include <stdio.h>
main()
   {
        char d;
        d = 'a';
        printf (“Ky tu trong bien d la %c ", d);
         printf ("Tri trong bien d la %d ", d);
   }
```

CÁC KIỂU DỮ LIỆU CỦA C

- **Kiểu int** là một kiểu số nguyên, biến đó có kích thước trong bộ nhớ là kích thước của số nguyên mà máy quy định
- Đối với bộ dịch Borland C/C++ thì chiều dài của kiểu int là 16 bit có dấu, tầm trị biểu diễn từ –32768 đến 32767 (tức từ -2^{15} đến $2^{15} - 1$).

CÁC KIỂU DỮ LIỆU CỦA C

Ví dụ :

```
#include <stdio.h>
main()
   {    int i;
        i = 1234;
        i = i + 123;
        printf ("Tri trong bien i la %d ", i);
   }
```

CÁC KIỂU DỮ LIỆU CỦA C

- **Kiểu float và double**
- **float** là kiểu số thực dấu chấm động, có độ chính xác đơn (7 ký số sau dấu chấm thập phân), **double** là kiểu số thực, dấu chấm động, có độ chính xác kép (15 ký số sau dấu chấm thập phân).
- **double** còn có thể được khai báo là **long float**, do đó khi khai báo *double b*; thì cũng hoàn toàn tương đương với *long float b;*

CÁC KIỂU DỮ LIỆU CỦA C

- **Kiểu float và double**
- Để xuất nhập cho hằng, biến, biểu thức **float** chuỗi định dạng được sử dụng là **"%f"** đối với kiểu **double** thì chuỗi định dạng là **"%lf"** cho các hàm printf và scanf.

CÁC KIỂU DỮ LIỆU CỦA C

- **Ví dụ:**

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
main()
{
 double x, y, luy_thua;
 clrscr(); printf ("Moi nhap 2 so:");
 scanf ("%lf %lf", &x, &y);
 if (x < 0 && (y - (int)y != 0))  printf ("Ban da nhap sai tri");
 else     { luy_thua = pow (x, y);
        printf ("Luy thua cua %5.2lf voi %5. 2lf la %5.2lf", x, y, luy_thua); }
}
```

CÁC KIỂU DỮ LIỆU CỦA C

- Ngoài ra, ANSI (American National Standards Institute) còn đưa thêm một kiểu dữ liệu nữa là **void**.
- Đây là kiểu không trị, chỉ dùng để biểu diễn kết quả trả về của hàm và khai báo pointer không trỏ đến một kiểu dữ liệu xác định nào cả. Kiểu này sẽ được nói chi tiết hơn ở các phần sau.

CÁC KIỂU DỮ LIỆU CỦA C

- Để bổ sung cho bốn kiểu dữ liệu cơ bản C còn đưa ra các dạng bổ sung **signed, unsigned, short, long :**
  - **signed** xác định kiểu có dấu.
  - **unsigned** xác định kiểu không dấu.
  - **short** xác định kiểu ngắn của kiểu cơ bản.
  - **long** xác định kiểu dài của kiểu cơ bản.

CÁC KIỂU DỮ LIỆU CỦA C

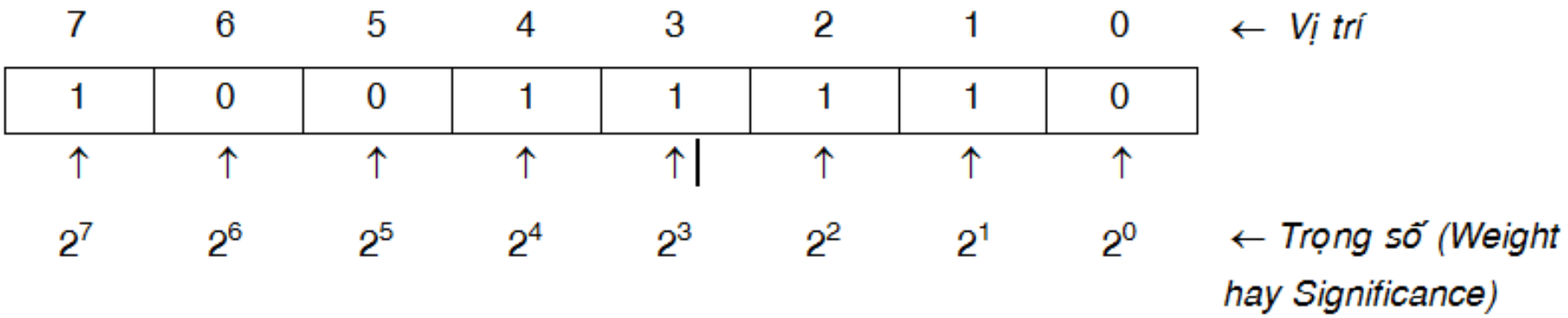
Dạng / Kiểu	signed	unsigned	short	long
char	signed char → char	unsigned char	x	x
int	signed int → int	unsigned int → unsigned	short int → int/short	long int → long
float	x	x	x	long float → double
double	x	x	x	long double

CÁC KIỂU DỮ LIỆU CỦA C

- Chú ý rằng trong bộ nhớ máy tính, các giá trị đều được lưu trữ dưới dạng mã nhị phân có nhiều bit (8, 16 hoặc 32 bit), trong đó số thứ tự của các bit được đánh số từ ***phải sang trái bắt đầu từ 0***, số hiệu này được gọi là vị trí của bit.
- Mỗi bit như vậy có một trọng số là 2 mũ n, với n là vị trí của bit đó.

CÁC KIỂU DỮ LIỆU CỦA C

- Do đó, một số mà nhị phân là 10011110 sẽ được biểu diễn như sau:



CÁC KIỂU DỮ LIỆU CỦA C

- Ví dụ:
  - Một giá trị kiểu ***integer***
  - 0000001010001010b
  - 1000001010001010b
  - Một giá trị kiểu ***unsigned***
  - 1000001010001010b

CÁC KIỂU DỮ LIỆU CỦA C

- Khi cần khai báo một biến n có kiểu là unsigned int ta chỉ cần viết: ***unsigned int*** n;
- hoặc gọn hơn ***unsigned*** n;
- hoặc chỉ cần viết: ***long*** p; là đủ để khai báo cho biến p có kiểu là ***signed long int***.

CÁC KIỂU DỮ LIỆU CỦA C

KIỂU	KÍCH THƯỚC	TẦM TRỊ BIỂU DIỄN
unsigned char	8 bit	0..255
char	8 bit	-128..+ 127
unsigned short	16 bit	0..65535
short	16 bit	-32768..+ 32767
unsigned	16 bit	0..65535
int	16 bit	-32768..+ 32767
unsigned long	32 bit	0.. 4294967295
long	32 bit	-2147483648..+ 2147483647
float	32 bit	-3.4 E 37..3.4 E + 38
double	64 bit	-1.7 E 307..1.7 E + 308
long double	80 bit	-3.4 E 4932..1.1 E + 4932

HẰNG (CONSTANT)

- Hằng là những giá trị cố định có trị hoàn toàn xác định và không thể thay đổi được chúng trong quá trình thực thi chương trình. Trong C, mỗi hằng đều có một kiểu dữ liệu riêng mà căn cứ vào kiểu dữ liệu ta có các loại hằng sau:
  - Hằng số
  - Hằng ký tự
  - Chuỗi ký tự
  - Biểu thức hằng

HẰNG (CONSTANT)

- **Hằng số**
- Hằng số là các trị số đã xác định, một hằng số có thể là số nguyên hoặc số thực.
- ***Hằng số nguyên***
  - Trong ngôn ngữ C, hằng số nguyên có thể thuộc một trong hai kiểu là ***integer*** hoặc ***long***. Ứng với mỗi kiểu, hằng số có thể được biểu diễn ở dạng thập phân, bát phân hay thập lục phân(0x6).

HẰNG (CONSTANT)

- ***Hằng số nguyên***
  - Hằng số nguyên được viết một cách bình thường và thường chiếm một từ (word) trong bộ nhớ, do đó nó có giá trị đi từ –32768 đến 32767, có nghĩa là MSB của dạng lưu trữ nhị phân của một số nguyên luôn là bit dấu.
  - Các hằng số nguyên: 10 –4 –23456
  - Ta cần lưu ý khi sử dụng hằng số nguyên vượt quá tầm quy định.

HẰNG (CONSTANT)

- **Ví dụ: Xét chương trình sau:**

```
#include <stdio.h>
main()
        {               printf ("%d %d %d", 32767, 32767 + 1,
32767 + 2);
        }
```

Kết quả xuất màn hình:

32767 -32768 –32767

HẰNG (CONSTANT)

- ***Hằng số nguyên***
- Hằng số nguyên dạng ***long*** lại được lưu trữ trong bộ nhớ với chiều dài 32 bit, có nghĩa là nó có thể có trị nằm trong khoảng -2147483648 đến +2147483647, và khi viết các hằng số nguyên dạng này ta cần phải thêm l hay L ngay sau số cần làm việc.
- Hằng số nguyên có thể ở dạng ***unsigned***, khi đó ta sẽ thêm u hoặc U vào ngay sau số đang làm việc (số đó có thể đang ở kiểu ***integer*** hoặc ***long***).

HẰNG (CONSTANT)

Hằng nguyên	Dạng biểu diễn	Tương đương thập phân
12347	decimal	12347
–153	decimal	–153
034	octal	28 ($3.8^1 + 4.8^0$)
0xa5	hex	165 ($10.16^1 + 5.16^0$)
0Xa2	hex	162 ($10.16^1 + 2.16^0$)
123L	decimal (dạng long)	123
034L	octal (dạng long)	28
0xa2L	hex (dạng long)	162

Ví dụ: Các hằng số sau đây ở dạng *unsigned*

123U 234u 24UL

HẰNG (CONSTANT)

- ***Hằng số thực***
- Trong ngôn ngữ C, số thực có thể ở dạng dấu chấm tĩnh hoặc dấu chấm động.
- **Ví dụ:** các số thực sau ở dạng dấu chấm tĩnh

  1.4 –2.34 –10.0234

HẰNG (CONSTANT)

- ***Hằng số thực***
- Một hằng thực ở dạng số dấu chấm động có thể có các thành phần sau:
  - phần nguyên: phần này là tùy yêu cầu.
  - dấu chấm thập phân: bắt buộc phải có.
  - phần lẻ: tùy yêu cầu.
  - các ký tự "e" hoặc "E" và một số mũ. Số mũ bản thân nó có thể âm.

HẰNG (CONSTANT)

Hằng thực dấu chấm động	Hằng thực tương đương
2.1415e4	21415.0
0.2344e–4	0.00002344
.2344e3	234.4

HẰNG (CONSTANT)

- *Cần lưu ý:*
  - Các hằng số được viết không có dấu thập phân và số mũ, sẽ được hiểu là nguyên và được lưu trữ theo kiểu ***int***, ngược lại sẽ được lưu trữ theo kiểu ***double***.
  - Các hằng số nguyên lớn hơn khả năng một ***int*** được tự động lưu trữ theo kiểu ***long***.
  - Các hằng số nguyên lớn hơn một ***long*** được lưu trữ theo kiểu ***double***.

HẰNG (CONSTANT)

- **Hằng ký tự**
- Hằng ký tự biểu diễn một giá trị ký tự đơn được viết giữa cặp dấu nháy đơn (' '). Mỗi ký tự có một mã số tương ứng trong bảng mã ký tự của máy, bình thường là mã ASCII.
- **Ví dụ:**
  - 'a' '/' '9‘
  - 'A' có mã là 65 trong bảng mã ASCII.
  - '0' có mã là 48 (0x30) trong bảng mã ASCII.

HẰNG (CONSTANT)

- **Hằng ký tự**
- **Ví dụ 1.26:**

```
#include <stdio.h>
main()
    {
     printf ("Ky tu: %c %c %c \n", 'A', '$', '1' );
     printf ("Ma ASCII (Octal): %o %o %o \n", 'A', '$', '1' );
     printf ("Ma ASCII (Decimal): %d %d %d \n", 'A', '$', '1' );
    }
```

HẰNG (CONSTANT)

- Chương trình in ra màn hình kết quả sau:

Ky tu: A $ 1

Ma ASCII (Octal): 101 44 61

Ma ASCII (Decimal): 65 36 49

- Lưu ý:
  - '\n' là ký tự xuống dòng (line feed)
  - '\0' là ký tự NUL

HẰNG (CONSTANT)

- **Chuỗi ký tự:**
- Chuỗi ký tự là một loạt các ký tự nằm trong cặp dấu nháy kép (“ ”); các ký tự này có thể là ký tự được biểu diễn bằng chuỗi thoát.
- **Ví dụ:**
  - “Mot chuoi ky tu"
  - "Chuoi ky tu co chuoi thoat: i can\'t go to school \n\a"

HẰNG (CONSTANT)

- **Chuỗi ký tự:**
- Ví dụ: “String”

s	t	r	i	n	g	\0

HẰNG (CONSTANT)

- **Biểu thức hằng**
- Một biểu thức được xem là một biểu thức hằng nếu giá trị của biểu thức hoàn toàn xác định, như vậy một biểu thức toán học là một biểu thức hằng khi trong biểu thức đó các toán hạng đều là những hằng số hoặc hằng ký tự.
- **Ví dụ: Xét các biểu thức hằng sau**
  - 10 – 13 % 3 sẽ được ghi là 9
  - 'a' – 'A' sẽ được ghi là 32
  - 1 < 8 sẽ được ghi là 1 (true)

BIẾN (VARIABLE)

- **Khai báo biến**
- Tất cả các biến được sử dụng trong một chương trình C đều phải được khai báo trước.
- Khi muốn sử dụng biến ta chỉ cần gọi tên biến, dĩ nhiên tên biến phải là một danh hiệu không chuẩn hợp lệ

BIẾN (VARIABLE)

- **Khai báo biến**
- Cú pháp khai báo biến

kiểu dsach_tenbien;

- Giải thích:
  - *kiểu* : kiểu của các biến cần khai báo
  - *dsach_tenbien* : danh sách liệt kê các tên biến cần khai báo, các biến cách nhau bằng dấu ","
- **Ví dụ:**

```
int lap, count, max;
double he_so_1, he_so_2, delta;
```

BIẾN (VARIABLE)

- **Khai báo biến**
- Biến của một chương trình C có thể được khai báo ở một trong ba vị trí sau:
  - Ngoài tất cả các hàm khi đó ta có biến toàn cục.
  - Đầu phần thân của một hàm hoặc một khối lệnh khi đó ta có biến cục bộ.
  - Trong phần định nghĩa đối số của hàm (gọi là đối số hàm hoặc tham số hàm).

BIẾN (VARIABLE)

- **Ví dụ:**

```
#include <stdio.h>
int a, b;                    ← khai báo biến ngoài
main()
   { …
        a = 10;
        b = a + 24;
    … }
```

BIẾN (VARIABLE)

- **Ví dụ:**

```
int tong()
   {          int i, tam;   ← khai báo biến trong
              ...
              for (i = 10;...
              ...
   }
```

- **Ví dụ:**

```
int luy_thua (int n, char ket_qua)  ← đối số hàm
   {     …
         for (i =1; …
         …
   }
```

BIẾN (VARIABLE)

- **Khai báo biến**
- Một biến có thể được khởi động trị ngay sau khi khai báo bằng phép gán với giá trị tương ứng, khi đó cú pháp khai báo biến như sau:

kiểu bien1 = tri1, bien2 = tri2;

- với các **tri1** và **tri2** phải là những giá trị đã xác định.
- **Ví dụ:**

```
double he_so_1 = 20.37;
char ky_tu = 'A', ky_tu_moi = ky_tu:
```

BIẾN (VARIABLE)

- **Các bổ túc kiểu const và volatile**
- Từ khóa **const**: khi được khai báo cho biến thì nó xác định rằng biến sẽ không bị thay đổi trị trong suốt quá trình thực thi chương trình, mọi sự thay đổi trị đều gây ra lỗi, biến đó ta gọi là biến hằng.
- Cú pháp:

  const kiểu tên_biến [= trị được thay thế];
- **Ví dụ:** const double bat_dau = 3.1415:
- const int max = 100;

BIẾN (VARIABLE)

- Nếu kiểu của biến hằng không nêu cụ thể thì biến hằng đó sẽ thuộc loại **int**, ngay cả nếu trị được thay thế là một trị khác **int** thì chỉ phần nguyên được sử dụng và lưu vào biến hằng mà thôi.

const max = 100;

const pi = 3.14; khi đó biến **pi chỉ là 3 mà thôi**

BIẾN (VARIABLE)

- Từ khóa **volatile**: chỉ ra rằng một biến có thể bị thay đổi trị từ một tác nhân không nằm trong chương trình. Từ khóa này làm cho biến của C có một tính linh động rất cao, ví dụ như biến có thể thay đổi theo đồng hồ hệ thống hay theo một chương trình nền nào đó.
- Cú pháp:

volatile <tên_biến>;

BIỂU THỨC

- Biểu thức là một sự kết hợp của các toán hạng là các biến, hằng hoặc phép gọi hàm bằng các toán tử để tạo ra được một trị, trị này có thể được sử dụng hoặc không được sử dụng tùy nhu cầu của lập trình viên.
- **Ví dụ:** Đối với C, một biểu thức như sau là hợp lệ:

```
a = (x = 10) – (y = a + 1) * ( (b += 1) > 12);
```

CÁC PHÉP TOÁN CỦA C

- **Toán tử số học:**
  - Toán tử cộng (+) : thực hiện phép toán cộng
  - Toán tử trừ (–) : thực hiện phép toán trừ
  - Toán tử nhân (*) : thực hiện phép nhân
  - Toán tử chia (/) : thực hiện phép chia
  - Toán tử modulo (%): thực hiện phép toán lấy số dư của phép chia nguyên

CÁC PHÉP TOÁN CỦA C

- Các phép toán này đều thực hiện được trên tất cả các toán hạng là hằng, biến hoặc biểu thức có kiểu dữ liệu **char**, **int**, **long**, **float**, **double**, trừ toán tử modulo chỉ thực hiện phép toán trên các dữ liệu thuộc các kiểu nguyên.
- Thứ tự kết hợp theo nguyên tắc toán học: nhân, chia, modulo trước; cộng, trừ sau, nếu các toán hạng đều ngang cấp thì kết hợp từ trái sang phải.

CÁC PHÉP TOÁN CỦA C

- Dấu cộng, trừ cũng có thể được dùng như phép toán đơn hạng (lấy dương và âm của một toán hạng).
- Phép nhân và chia các số nguyên sẽ chỉ cho **kết quả nguyên**.

CÁC PHÉP TOÁN CỦA C

Ví dụ

```
int a = 10, b = 3;
double c;
c = a/b; /*   c =3 */
c = a % b; /* c =1 */
```

```
double x = 10., y = 3, z;
z = x/y; /*z =3.3333 */
```

CÁC PHÉP TOÁN CỦA C

- Nếu có nhiều toán hạng khác kiểu nhau thì C sẽ thực hiện việc chuyển kiểu tự động theo quy luật: toán hạng thuộc kiểu có trị nhỏ hơn sẽ được chuyển sang kiểu có trị lớn hơn.
- Ví dụ:

```
double a = 10, c; int b = 3;
c=a/b;
```

- Trong quá trình tính b kiểu **double**, sau khi tính *b* thuộc kiểu **int**

CÁC PHÉP TOÁN CỦA C

- **Toán tử quan hệ:**
  - Toán tử bằng (==)
  - Toán tử khác (!=)
  - Toán tử lớn hơn (>)
  - Toán tử nhỏ hơn (<)
  - Toán tử lớn hơn hoặc bằng (>=)
  - Toán tử nhỏ hơn hoặc bằng (<=)

CÁC PHÉP TOÁN CỦA C

- Khi mối quan hệ giữa hai toán hạng theo toán tử quan hệ trong biểu thức là ĐÚNG thì biểu thức đó sẽ trả về một trị nguyên là 1, còn ngược lại mối quan hệ đó là SAI thì biểu thức đó sẽ trả về một trị nguyên là 0, đây là các trị nguyên bình thường mà ta có thể dùng nó để tính toán.

CÁC PHÉP TOÁN CỦA C

- Ví dụ:

```
int a, b, c;
char kt;
a = 1;
b = 2;
c = –3;
kt = (a <= 4) * 2 + (b < 3) – c;
```

CÁC PHÉP TOÁN CỦA C

Ví dụ

a = (b == 4); /*toán tử quan hệ*/	a = (b = 4); /*toán tử gán*/

CÁC PHÉP TOÁN CỦA C

- **Toán tử logic:**
- C có các toán tử logic **not** (!), **and** (&&), **or** (||). Các toán tử này cho phép liên kết các biểu thức quan hệ đơn lại với nhau để tạo các biểu thức phức tạp hơn.
- Các toán tử này có độ ưu tiên là not, and, or; nếu trong biểu thức các toán tử đều ngang cấp nhau thì thứ tự tính toán từ trái sang phải.

CÁC PHÉP TOÁN CỦA C

- **Toán tử logic**

Toán hạng 1	Toán hạng 2	Kết quả		
A	B	! A	A && B	A ||B
bằng 0	bằng 0	1	0	0
bằng 0	khác 0	1	0	1
khác 0	bằng 0	0	0	1
khác 0	khác 0	0	1	1

CÁC PHÉP TOÁN CỦA C

- **Ví dụ:**

```
int a, b;
a = 4;
b = 5;
```

(a > 5) && (b < 3) có trị là FALSE (0)

CÁC PHÉP TOÁN CỦA C

- **Ví dụ:**
- Trong mệnh đề **if** sau đây thay vì viết

```
if (delta == 0) { ...   }
```

- Ta có thể viết gọn hơn như sau:

```
if (!delta)       { ...   }
```

CÁC PHÉP TOÁN CỦA C

- **Toán tử logic:**
- Điều cần lưu ý là các phép toán **and** (&&) và **or** (||) được thực hiện từ trái sang phải và việc tính toán sẽ **dừng lại ngay khi giá trị của cả biểu thức logic đã xác định** trị mà không cần tính tiếp các phần còn lại của biểu thức nữa.

CÁC PHÉP TOÁN CỦA C

- Ví dụ:

```
y = 10;
scanf ("%d", &x);
if ( (x >= 10) && (x <= (y = 100)) )
   {
        ...
   }
```

CÁC PHÉP TOÁN CỦA C

- **Toán tử trên bit:**
- C có sáu toán tử đặc biệt cho phép lập trình viên xử lý dữ liệu kiểu nguyên ở cấp độ bit, các toán tử này được gọi là toán tử trên bit.
  - NOT: ~, thực hiện việc đảo bit
  - AND: &, thực hiện việc and bit
  - OR: | , thực hiện việc or bit
  - XOR: ^ , thực hiện việc việc xor bit
  - Dịch trái: <<, dịch các bit sang trái
  - Dịch phải: >>, dịch các bit sang phải

CÁC PHÉP TOÁN CỦA C

Bit	Bit	Phép toán			
A	B	~ A	A & B	A | B	A ^ B
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

CÁC PHÉP TOÁN CỦA C

- Ví dụ:

a = 1030 có mã nhị phân là 0000 0100 0000 0110

b = 224 có mã nhị phân là 0000 0000 1110 0000

- Thực hiện các phép toán a&b, a|b, a^b

CÁC PHÉP TOÁN CỦA C

- **Toán tử trên bit:**
- Toán tử dịch trái (<<) và dịch phải (>>) cho phép thực hiện việc dời các bit của toán hạng sang bên trái hoặc sang phải. Cú pháp như sau:

  biểu_thức_nguyên << số_bit_dời

  biểu_thức_nguyên >> số_bit_dời

- Với *biểu_thức_nguyên* và *số_bit_dời* có thể là hằng, biến hoặc biểu thức kiểu nguyên.

CÁC PHÉP TOÁN CỦA C

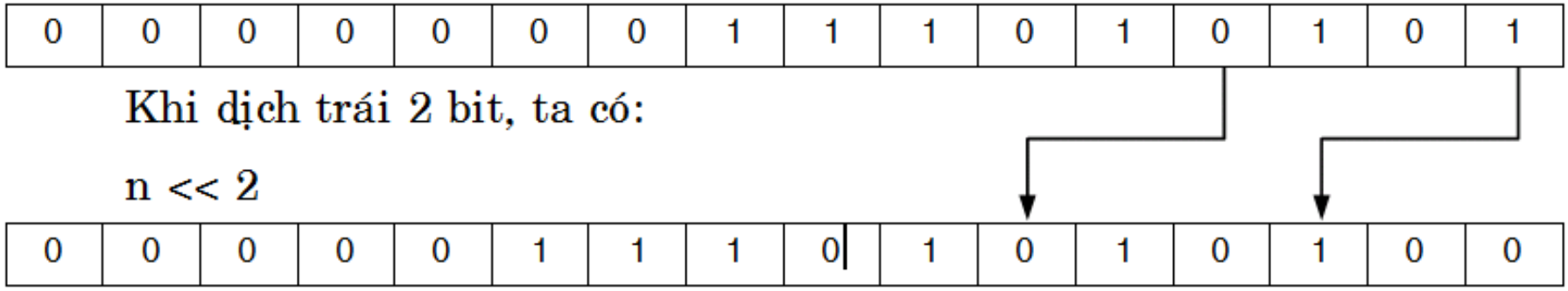
- Trong phép dịch trái, các bit ở bên phải của toán hạng sẽ được ghi vào các giá trị là 0
- Trong phép dịch phải ta có hai trường hợp:
  - Nếu toán hạng bên trái có dữ liệu thuộc dạng **unsigned** (unsigned, unsigned long, unsigned char) thì phép dịch phải sẽ ghi 0 vào các bit bên trái của kết quả.
  - Còn nếu toán hạng bên trái có dữ liệu thuộc dạng **signed** (int, long, char) thì phép dịch phải sẽ ghi bit dấu vào các bit bên trái của kết quả.

CÁC PHÉP TOÁN CỦA C

Xét biến n được khai báo

```
int n;
```

đang có trị n = 469 (tức 0x01d5), biểu diễn trong bộ nhớ dưới dạng nhị phân là



CÁC PHÉP TOÁN CỦA C

unsigned n;

Giả sử n đang lưu giá trị n = 42569 (tức 0xA649), tức trong bộ nhớ ta có n

1	0	1	0	0	1	1	0	0	1	0	0	1	0	0	1

Khi phép dịch phải 2 bit xảy ra (n >> 2), kết quả như sau:

0	0	1	0	1	0	0	1	1	0	0	1	0	0	1	0

0 →

```
int n;
```

Giả sử n đang lưu giá trị n = –32766 (tức 0x8002), tức trong bộ nhớ ta có n

1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

bit dấu

Khi phép dịch phải 2 bit xảy ra (n >> 2), kết quả như sau:

1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

1 →

CÁC PHÉP TOÁN CỦA C

- **Toán tử tăng giảm**
- Trong C có hai toán tử đặc biệt gọi là toán tử tăng (++) và toán tử giảm (--) dùng để tăng hoặc giảm một biến nào đó đi 1.Cú pháp sử dụng:

++ biến

biến ++

– – biến

biến – –

- với ***biến*** có thể thuộc loại bất kỳ hoặc pointer.

CÁC PHÉP TOÁN CỦA C

- Cần lưu ý rằng cặp dấu ++ hoặc phải được viết liền nhau và cần phải rõ ràng, trong những trường hợp có thể lầm lẫn, ví dụ như 2 biểu thức:

a++ + b

- và

a + ++b

- là khác nhau.

CÁC PHÉP TOÁN CỦA C

- Các phép toán tăng giảm này nhanh hơn nhiều so với thao tác bình thường là cộng hoặc trừ biến đó với 1 rồi gán trở lại cho biến đó vì chúng rất gần với các phép toán tăng giảm của ngôn ngữ máy

CÁC PHÉP TOÁN CỦA C

- Ví dụ:
- Trong thân vòng while của chương trình tính tổng từ 1 tới n, ta có thể viết như sau:

```
s = 0;
so = 1;
while (so <= n)
   {
        s = s + so;
        so++;
   }
```

CÁC PHÉP TOÁN CỦA C

- **Toán tử gán:**
- Phép toán gán là phép toán cơ bản trong mỗi ngôn ngữ lập trình. Phép gán có hai dạng theo cú pháp sau đây:
  - Gán đơn giản: **biến = trị**
  - Gán phức tạp: **biến op= trị**
  - Với **trị** có thể là hằng, biến hoặc là biểu thức
  - **op** có thể là * / % + – hoặc << >> & ^ |
- Phép gán trị phức tạp:

"biến op= trị" chính là "biến = biến op trị"

CÁC PHÉP TOÁN CỦA C

- **Ví dụ:**

```
int a, b = 2;
a = 4;
b *= a * 3;
```

CÁC PHÉP TOÁN CỦA C

- Trong phép gán đơn,
  - Nếu hai toán hạng có cùng kiểu thì toán hạng bên phải sẽ được gán vào toán hạng bên trái.
  - Nếu không, trước khi gán toán hạng bên phải sẽ được chuyển theo kiểu của toán hạng bên trái, điều này có thể sẽ gây ra kết quả sai hoặc không chính xác, nếu như kiểu của toán hạng bên trái thấp hơn kiểu của toán hạng bên phải.

CÁC PHÉP TOÁN CỦA C

Kiểu toán hạng trái	Kiểu toán hạng phải	Trị có thể mất sau khi gán
signed char	unsigned char	Giá trị > 127, thành số âm
char	short int	Mất trị từ bit 8 trở đi
char	int	Mất trị từ bit 8 trở đi
char	long	Mất trị từ bit 8 trở đi
short int	long int	Mất 16 bit cao (một int)
int	float	Mất phần thập phân và phần trị lớn hơn một int
float	double	Độ chính xác do làm tròn

CÁC PHÉP TOÁN CỦA C

- Đối với phép gán phức, việc chuyển kiểu được thực hiện theo việc chuyển kiểu tự động trong khi thực hiện việc tính toán biểu thức và việc chuyển kiểu của phép gán đơn giản.
- Phép gán phức hợp rất hiệu quả khi các toán hạng bên trái là những biến khá dài.

CÁC PHÉP TOÁN CỦA C

- Ví dụ:
- Thay vì viết:

```
n = n * (x + 5) + n * (a + 8);
```

- ta chỉ cần viết:

```
n *= x + 5 + a + 8;
```

- Hoặc phức tạp hơn

```
a[i][j] –= b[i][j];
```

- thay vì phải viết dài dòng

```
a[i][j] = a[i][j] – b[i][j];
```

CÁC PHÉP TOÁN CỦA C

- Nếu một biểu thức gán được kết thúc bằng một dấu ";" thì ta có một lệnh gán, còn nếu biểu thức gán này được sử dụng trong một biểu thức phức hợp khác thì biểu thức gán sẽ có trị là trị của biến sau khi gán.
- **Ví dụ:**

```
int a = 4, b = 3;
b + = (a = 2 * b) + (a *=  b);
```

CÁC PHÉP TOÁN CỦA C

- **Toán tử phẩy - Biểu thức phẩy**
- Cú pháp:

  biểu_thức_1, biểu_thức_kết_quả

- Với *biểu_thức_1* và *biểu_thức_kết_quả* là hai biểu thức bất kỳ.
- **Ví dụ:** m = (a = 2, t = a + 3);
- sẽ cho a = 2, t = 5 và m = t = 5
- Hoặc x = (t = 1, t + 4);
- sẽ cho t = 1 và x = 5

CÁC PHÉP TOÁN CỦA C

- **Toán tử điều kiện - biểu thức điều kiện**
- Trong ngôn ngữ C có một toán tử khá đặc biệt gọi là toán tử điều kiện, ký hiệu của toán tử điều kiện là hai dấu "?" và ":" theo cú pháp sau:

dieu-kien ? bieu-thuc1 : bieu-thuc2

- với *dieu-kien* là một biểu thức bất kỳ có kết quả thuộc kiểu chuẩn (scalar type)
- *bieu-thuc1*, *bieu-thuc2* là hai biểu thức bất kỳ và dĩ nhiên có thể là một biểu thức điều kiện khác.

CÁC PHÉP TOÁN CỦA C

- **Ví dụ:**
- Thay vì phải viết dài dòng

```
if ( i >0 )
n = 1;
else n  = 0;
```

- ta chỉ cần dùng biểu thức điều kiện

```
n =(i > 0)  ? 1 : 0;
```

CÁC PHÉP TOÁN CỦA C

- **Toán tử sizeof**
- Đây là một toán tử cho ta kích thước của một biến hoặc một kiểu dữ liệu nào đó. Do phạm vi sử dụng của **sizeof** rất rộng và thường được dùng để lấy kích thước các kiểu dữ liệu phức hợp như struct, union... Việc sử dụng toán tử này cho phép ta không phải quan tâm đến chiều dài cụ thể của các biến.

CÁC PHÉP TOÁN CỦA C

- Toán hạng của sizeof là một biến hoặc một kiểu dữ liệu bất kỳ nào đó đã định nghĩa.

  sizeof *(biến)*

  sizeof *biến*

  sizeof *(kiểu)*

- Kết quả của toán tử này là một giá trị nguyên chỉ kích thước (tính bằng **byte** hoặc char) của kiểu dữ liệu hoặc của biến đó. Biến hoặc kiểu này có thể là một biến hoặc một kiểu đơn giản hay phức hợp.

Độ ưu tiên	Phép toán	Thứ tự kết hợp
1	() [ ] ->.	Trái qua phải
2	! ~ ++ – + (type) * & sizeof	Phải qua trái *
3	* / %	Trái qua phải
4	+ –	Trái qua phải
5	<< >>	Trái qua phải
6	< <= > >=	Trái qua phải
7	== !=	Trái qua phải
8	&	Trái qua phải
9	^	Trái qua phải
10	|	Trái qua phải
11	&&	Trái qua phải
12	||	Trái qua phải
13	?:	Trái qua phải*
14	= += –= *= /= %= <<= >>= &= |= ^=	Phải qua trái *
15	,	Trái qua phải

CẤU TRÚC TỔNG QUÁT

- Một chương trình C tổng quát gồm hai phần: phần khai báo đầu (header) và phần hàm (function).
- Phần khai báo đầu bao gồm:
  - Các lệnh tiền xử lý: include, define ...
  - Các khai báo hằng, biến ngoài ...
  - Các prototype của các hàm được sử dụng trong chương trình
- Phần hàm là phần định nghĩa các hàm sử dụng trong chương trình, trong các hàm này phải có hàm **main**().

CẤU TRÚC TỔNG QUÁT

- **Ví dụ:** Nhập một số kiểm tra số đó chẵn hay lẻ.

```
#include <stdio.h>
#include <conio.h>
int kiem_tra (int so);
/* ham kiem_tra nhan vao doi so
la mot so nguyen, tra ve tri
- 0 la so chan
- 1 la so le
*/
```

CẤU TRÚC TỔNG QUÁT

```
main()
   {
        int n;
        clrscr();
        printf ("Nhap mot so: ");
        scanf ("%d", &n);
        if (kiem_tra(n))
                printf ("So da nhap la so le \n");
        else
                printf ("So da nhap la so chan \n");
```

CẤU TRÚC TỔNG QUÁT

```
        getch();
    }
int kiem_tra (int so)
    {
return (so % 2 == 0)? 0:1;
    }
```

Kết thúc chương 7