

Ngôn ngữ lập trình C


Bài 2. Cấu trúc chương trình C, kiểu dữ liệu & biến

Soạn bởi: TS. Nguyễn Bá Ngọc

Nội dung

- Cấu trúc chương trình C
- Các hệ đếm
- Kiểu dữ liệu & biến
 - Biểu diễn & Kiểu số nguyên
 - Biểu diễn & Kiểu số thực
 - Khai báo biến thuần

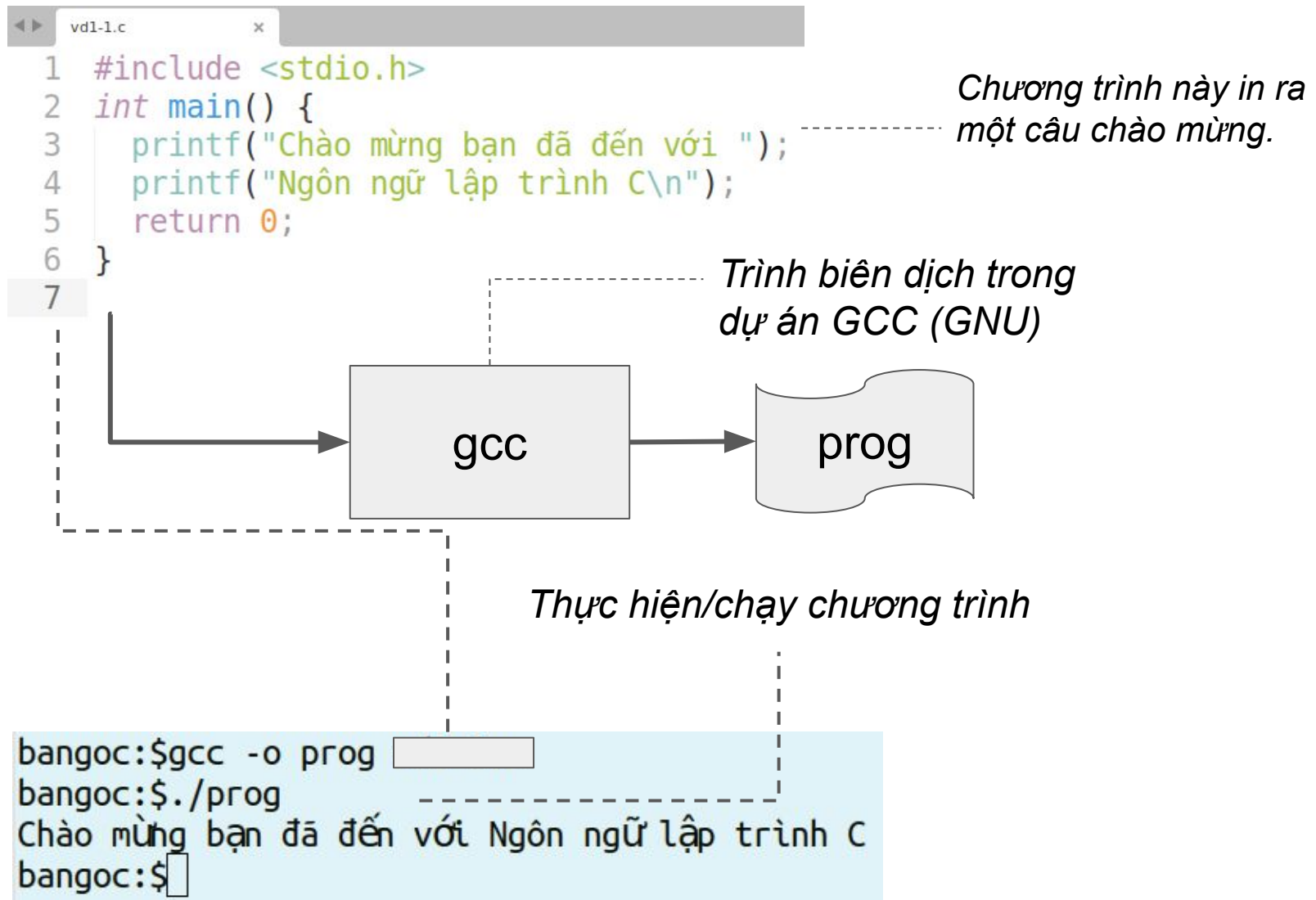
Nội dung

- 
- Cấu trúc chương trình C
 - Các hệ đếm
 - Kiểu dữ liệu & biến
 - Biểu diễn & Kiểu số nguyên
 - Biểu diễn & Kiểu số thực
 - Khai báo biến thuần

Cấu trúc chương trình C

- Mã nguồn của 1 chương trình C có thể được chia thành nhiều phần được lưu trong nhiều tệp khác nhau. Các thành phần có thể được biên dịch độc lập rồi sau đó được ghép lại thành chương trình thực thi.
 - Chúng ta gọi kết quả thu được sau tiền xử lý **tệp mã nguồn (.c)** cùng với các **tệp tiêu đề (.h)** và các tệp mã nguồn khác được chèn vào bằng các lệnh tiền biên dịch (`#include`) là **đơn vị biên dịch**.
 - Mỗi đơn vị biên dịch có thể sử dụng các thành phần có trong các đơn vị biên dịch khác, tạo thành các liên kết phụ thuộc giữa các đơn vị biên dịch.

Ví dụ 2.1. Biên dịch 1 tệp mã nguồn



Ví dụ 2.2. Biên dịch nhiều tệp mã nguồn

```
1 #include <stdio.h>
2
3 double f(double x);
4
5 int main() {
6     double x, y;
7     printf("Nhập x và y: ");
8     scanf("%lf%lf", &x, &y);
9     printf("%g + %g = %g\n", x, y, x + y);
10    printf("f(%g) = %g\n", x, f(x));
11    return 0;
12 }
```

Chương trình này được tạo thành từ 2 tệp. Khi được thực hiện, chương trình hỏi người dùng nhập vào 2 số thực x và y, sau đó in ra tổng x + y và giá trị hàm $f(x) = x^2 + 2 * x + 1$

```
bangoc:$gcc -o prog
bangoc:$./prog
Nhập x và y: 3 5.5
3 + 5.5 = 8.5
f(3) = 16
bangoc:$
```

Ý nghĩa của các câu lệnh ?

`gcc -o prog a.c` và `gcc -o prog b.c f.c`

Một số yêu cầu tổ chức mã nguồn

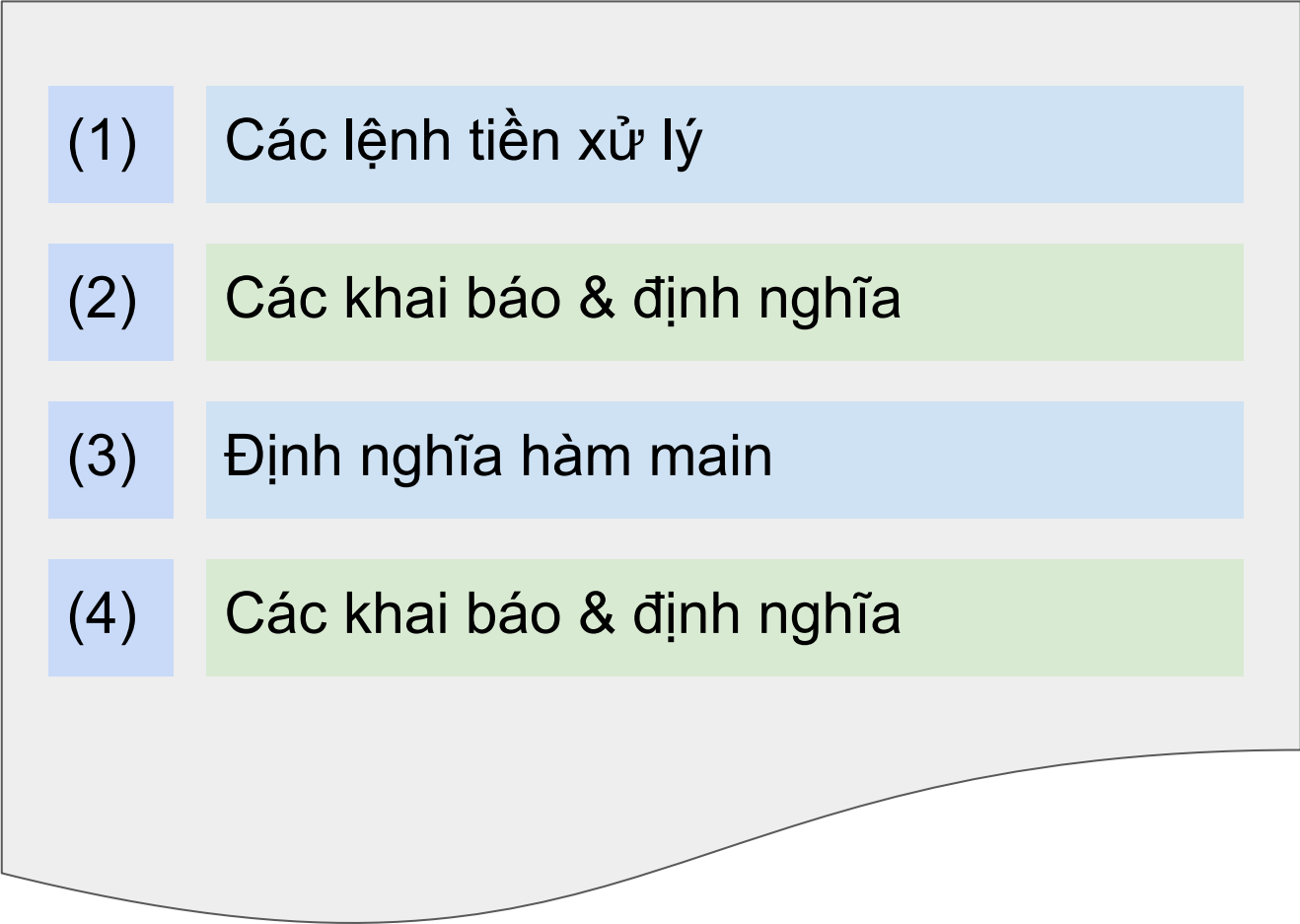
- *Định danh phải được khai báo trước khi sử dụng*
 - Thứ tự được quy ước từ trái sang phải, từ trên xuống dưới.
- *Mỗi chương trình có đúng 1 hàm main*
 - Hàm main được quy ước là hàm được gọi đầu tiên khi chương trình được thực hiện, là điểm bắt đầu thực hiện chương trình.
- *Mã nguồn được lưu trong tệp văn bản với phần mở rộng .c*
 - Trình biên dịch sử dụng phần mở rộng của tệp để xác định tiến trình biên dịch.
 - GCC quy ước tệp **.c** là tệp chứa mã nguồn C:
 - Trình biên dịch áp dụng tiến trình biên dịch chương trình C cho tệp .c .
 - Tệp có phần mở rộng .cpp, .cc, .cxx, ... được nhận diện là tệp chứa mã nguồn C++, và được biên dịch như chương trình C++.
 - Không biên dịch được tệp với phần mở rộng .txt, ...
 - => Chúng ta lưu mã nguồn trong tệp văn bản với phần mở rộng .c

Nguyên mẫu hàm main

- Để tương thích với quy chuẩn, chương trình cần sử dụng 1 trong các nguyên mẫu hàm main sau:
 - `int main(void);` // 1 - Không có tham số, hoặc
 - `int main(int argc, char *argv[]);` // 2 - Có đúng 2 tham số
 - Hoặc nguyên mẫu khác tương đương với 1 hoặc 2
 - `int main();` // Tương đương với 1
 - `int main(int argc, char **argv);` // Tương đương với 2
 - có thể thay đổi tên các tham số ...
- Các nguyên mẫu hàm main sau (tuy vẫn có thể biên dịch được nhưng) không tương thích với quy chuẩn:
 - `void main();` // - Không trả về giá trị
 - `main();` // - Mặc định kiểu trả về là int
 - `int main(char *argv[], int n);` // - Sai kiểu tham số
 - V.V...

Chương trình 1 đơn vị biên dịch

Tập mã nguồn (.c)

- 
- The diagram illustrates the structure of a C source code file. It is represented as a light gray rectangular block with a wavy bottom edge. Inside this block, there are four horizontal bars, each preceded by a blue square containing a number in parentheses. The bars alternate in color: light blue for (1) and (3), and light green for (2) and (4). To the right of the gray block, a vertical gray arrow points downwards, indicating the flow of the program execution.
- (1) Các lệnh tiền xử lý
 - (2) Các khai báo & định nghĩa
 - (3) Định nghĩa hàm main
 - (4) Các khai báo & định nghĩa

Ví dụ 2.2. Phong cách & Khả năng đọc

```
vd2-1b.c
1 #include <time.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <ctype.h>
5
6 #define N 64
7
8 int main(void) {
9     int a, i = 1, j = 0, t[N];
10    for (srand(time(0));) {
11        a = getchar();
12        if (isalpha(a) && i < N) {
13            t[i++] = a;
14        } else {
15            j = 1;
16            while (i > 1) {
17                putchar(t[j]);
18                t[j] = t[--i];
19                j = i > 2 ? rand() % (i - 2) + 2 : 1;
20            }
21            if (a == EOF) {
22                break;
23            }
24            putchar(a);
25        }
26    }
27    return 0;
28 }
```

```
vd2-1a.c
1 #include <time.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 int main(){unsigned
5 a,i=1,j=0,t[64];for
6 (srand(time(0));i;)
7 j=j?--i<1&&a>8?0:(
8 putchar(i?t[j]:a),i
9 ++)?t[j]=t[--i],i>2
10 ?rand()%(i-2)+2:1:0
11 :(26>((a=getchar())
12 |32)-97||1==a>7)&&
13 i<64?t[i++]=a,0:1;}
14 }
```

*Ngôn ngữ C rất linh hoạt, có thể diễn đạt 1 ý tưởng theo nhiều cách nhưng **chúng ta hướng tới viết mã nguồn dễ đọc và dễ hiểu.***

Ví dụ 2.3. Quy tắc khai báo định danh

Chương trình này có lỗi biên dịch

```
vd2-2a.c
1 #include <stdio.h>
2 #define M 10
3 float f1(float x) {
4     return x + M;
5 }
6
7 int main() {
8     scanf("%f", &x);
9     float x;
10    f1(x);
11    f2(x);
12    return 0;
13 }
14 #define N 2.0
15 float f2(float x) {
16     return x * N;
17 }
```

Sẽ học sau (tạm chấp nhận ở thời điểm này)

Bổ xung các khai báo

Các định danh x và f2 chưa được khai báo trước khi sử dụng.

vd2-2a.c:8:16: error: 'x' undeclared (first use in this function)

scanf("%f", &x);

vd2-2a.c:8:16: note: each undeclared identifier is reported only once for each function it appears in

vd2-2a.c: At top level:

vd2-2a.c:15:7: error: conflicting types for 'f2'

float f2(float x) {

vd2-2a.c:11:3: note: previous implicit declaration of 'f2' was here

f2(x);

1 cách sửa lỗi

```
vd2-2b.c
1 #include <stdio.h>
2 #define M 10
3 float f1(float x) {
4     return x + M;
5 }
6 float f2(float x);
7 int main() {
8     float x;
9     scanf("%f", &x);
10    f1(x);
11    f2(x);
12    return 0;
13 }
14 #define N 2.0
15 float f2(float x) {
16     return x * N;
17 }
```

(1)

(2)

(3)

(4)

OK

Từ khóa

Từ chức năng, có ý nghĩa xác định, được nhận diện và xử lý bởi trình biên dịch trong tiến trình biên dịch, không được sử dụng làm định danh.

auto

break

case

char

const

continue

default

do

double

else

enum

extern

float

for

goto

if

inline

int

long

register

restrict

return

short

signed

sizeof

static

struct

switch

typedef

union

unsigned

void

volatile

while

_Bool

_Complex

_Imaginary

Những từ khóa mà chúng ta sẽ học & sử dụng trong học phần này.

- C có phân biệt chữ hoa và chữ thường
 - char ≠ Char, char là từ khóa còn Char thì không

Định danh

Người lập trình có thể đặt tên (định danh) cho các thứ có trong chương trình để thuận tiện diễn đạt các ý tưởng.

- Có thể đặt tên cho:
 - biến / variables;
 - hàm / functions;
 - kiểu và thành phần của
 - cấu trúc (struct),
 - nhóm (union),
 - hoặc danh mục (enum);
 - kiểu typedef;
 - nhãn / label (đích đến của lệnh goto);
 - Macro;
 - tham số Macro.
 - v.v...

Định danh₍₂₎

- Các quy định cơ bản đối với định danh:
 - Không được trùng với từ khóa.
 - Ví dụ: `int if; //` => Lỗi biên dịch
 - Phải được bắt đầu bằng chữ cái tiếng Anh hoặc dấu gạch nối (`'_'`), và chỉ chứa dấu gạch nối, chữ cái, hoặc chữ số.
 - `_a1, ab100` => Hợp lệ
 - `0ab2` => Không hợp lệ (bắt đầu bằng chữ số => nhập nhầm với số)
 - C có phân biệt chữ hoa và chữ thường
 - `Abc` và `abc` là các định danh khác nhau.
 - Một định danh có thể là tên của nhiều thực thể khác nhau, phân giải được thực hiện dựa trên ngữ cảnh sử dụng:
 - => cho phép tái sử dụng tên, đặt tên dễ dàng hơn.
 - trình biên dịch thường đưa ra thông báo lỗi nếu không thể phân giải định danh dựa trên các quy định ngôn ngữ.
 - Ví dụ: `struct student student; //` Ok - kiểu & biến cấu trúc
 - `int x; {float x;} //` => Ok - khác ngữ cảnh
 - `int x; float x; //` => Lỗi biên dịch

Ví dụ 2.4. Định danh và từ khóa

```
vd2-2b.c
1  #include <stdio.h>
2  #define M 10
3  float f1(float x) {
4      return x + M;
5  }
6  float f2(float x);
7  int main() {
8      float x;
9      scanf("%f", &x);
10     f1(x);
11     f2(x);
12     return 0;
13 }
14 #define N 2.0
15 float f2(float x) {
16     return x * N;
17 }
```

Trong mã nguồn này:

- Các định danh: M, f1, x, f2, main, scanf, N.
- Các từ khóa: float, return, int.

Các lệnh tiền xử lý

- Được trình biên dịch thực hiện ở pha tiền xử lý
 - (*Trước pha Dịch*)
- Chúng ta thường sử dụng:
 - Macro đối tượng
 - Lệnh chèn tệp

Macro đối tượng

Macro đối tượng về hình thức là định danh, không có tham số:

#define định-danh nội-dung-thay-thế

Ví dụ:

```
#define N 100
```

Thường kết thúc trong phạm vi 1 dòng.

```
#define M 200
```

!Lưu ý: Lệnh tiền xử lý không có dấu ; ở cuối

```
#define MN (M * N)
```

Trình biên dịch thay *định-danh* Macro bằng nội-dung-thay-thế

```
#define N 100
```

```
int a[N]; => int a[100]; // N được thay bằng 100
```

Với

```
#define N 100;
```

*Chạy lệnh với khóa -E để xem kết quả tiền xử lý:
gcc -E tệp.c*

```
int a[N]; => int a[100;]; // Lỗi
```

Đặt tên cho hằng giá trị giúp mã nguồn dễ đọc & dễ quản lý hơn

Chèn tệp

Lệnh chèn tệp (#include) có hai định dạng thường gặp:

`#include "đường-dẫn-tệp"`

`#include <đường-dẫn-tệp>`

- Phạm vi tìm kiếm tệp được chèn vào theo định dạng <> là tệp con của phạm vi tìm kiếm theo định dạng "".
 - Định dạng `#include "đường-dẫn-tệp"` được gọi là định dạng tìm kiếm mở rộng, còn `#include <đường-dẫn-tệp>` được gọi là định dạng tìm kiếm cơ bản. Nếu không tìm thấy trong phần mở rộng thì tiếp tục xử lý như `#include <đường-dẫn-tệp>`.
- Định dạng "" thường được sử dụng cho các tệp nằm trong thư mục dự án, còn định dạng <> thường được sử dụng cho tệp nằm trong thư mục hệ thống.

Triển khai trình biên dịch tự thiết lập quy tắc tìm kiếm cụ thể.

Quy tắc tìm tệp được chèn vào với GCC

Mặc định:

- `#include "đường-dẫn-tệp"`: Trước tiên tìm trong thư mục chứa tệp đang được xử lý, sau đó tìm trong các *đường dẫn của hệ thống* (như định dạng `<>`).
- `#include <đường-dẫn-tệp>`: Chỉ tìm trong các *đường dẫn của hệ thống* (biến môi trường PATH).

Tùy chỉnh đường dẫn tìm kiếm với khóa `-I`:

- `-I thư-mục`: Các đường dẫn được bổ xung vào trong câu lệnh biên dịch được tìm sau thư mục hiện hành.
- Có thể xem chi tiết các thư mục tìm kiếm bằng cách bổ xung tham số `-v` (verbose) vào câu lệnh biên dịch, ví dụ:
 - `gcc -v -o prog main.c`

[<https://gcc.gnu.org/onlinedocs/cpp/Search-Path.html>]

Nội dung

- Cấu trúc chương trình C
- Các hệ đếm
- Kiểu dữ liệu & biến
 - Biểu diễn & Kiểu số nguyên
 - Biểu diễn & Kiểu số thực
 - Khai báo biến thuần

(TĐC bỏ qua phần này)

Hệ cơ số 10

- Decimal Number System
 - Decem (Latin) => 10
- Các đặc điểm:
 - Chúng ta (và nhiều nơi khác, nhưng không phải mọi nơi đều) sử dụng các chữ số 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
 - Có trọng số vị trí
 - $123 \neq 321$
 - $123 = 1 * 10^2 + 2 * 10^1 + 3 * 10^0$

Hệ cơ số 10 là hệ đếm được sử dụng phổ biến nhất trong đời sống (*bởi người*)

Hệ nhị phân

- Binary Number System

- Nghĩa của từ binary: (Nhận 1 trong) 2 trạng thái loại trừ: đúng hoặc sai, bật hoặc tắt, có hoặc không v.v.

- Các đặc điểm:

- Sử dụng 2 chữ số: 0, 1
- Có trọng số vị trí
 - $101_2 \neq 110_2$
 - 101_2 có giá trị = $1 * 2^2 + 0 * 2^1 + 1 * 2^0$ ở HCS 10

- Một số khái niệm:

- bit = 1 chữ số nhị phân (binary digit)
- byte = 8 bits
- nybble = 4 bits

Hầu hết các hệ thống máy tính sử dụng hệ nhị phân

Chuyển đổi từ HCS 2 => 10

- Sử dụng trọng số vị trí của các chữ số:

$$110101_2 = 1 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$$
$$= 32 + 16 + 0 + 4 + 0 + 1$$
$$= 53$$

Bit có vị trí lớn
nhất - MSB (**M**ost
Significant **B**it)

Bit có vị trí nhỏ
nhất - LSB (**L**east
Significant **B**it)

Chuyển đổi từ HCS 10 \Rightarrow 2

- **Cách 1.** Lặp trừ đi lũy thừa của 2 gần nhất

$$\begin{array}{rcll} 53 & = & 110101_2 & \\ - & 32 & (2^5) & \leq 1 \\ & 21 & & \\ - & 16 & (2^4) & \leq 1 \\ & 5 & & \leq 0 \\ - & 4 & (2^2) & \leq 1 \\ & 1 & & \leq 0 \\ - & 1 & (2^0) & \leq 1 \\ & 0 & & \end{array}$$

Đọc kết quả từ trên xuống và lấp đầy các khoảng trống giữa các lũy thừa bằng giá trị 0.

Chuyển đổi từ HCS 10 \Rightarrow $2_{(2)}$

- **Cách 2.** Lặp chia 2 và lấy phần dư

$$\begin{array}{rcll} 53 & / & 2 & = 26 \text{ Dư } 1 \\ 26 & / & 2 & = 13 \text{ Dư } 0 \\ 13 & / & 2 & = 6 \text{ Dư } 1 \\ 6 & / & 2 & = 3 \text{ Dư } 0 \\ 3 & / & 2 & = 1 \text{ Dư } 1 \\ 1 & / & 2 & = 0 \text{ Dư } 1 \end{array}$$

Đọc kết quả từ dưới lên
 110101_2

Hệ cơ số 16

- Hexadecimal Number System
 - Hexa (Hy Lạp cổ) \Rightarrow 6
 - Decem (Latin) \Rightarrow 10
- Các đặc điểm
 - Sử dụng các ký hiệu: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
 - Có trọng số vị trí:
 - $ABC_{16} \neq BCA_{16}$
 - ABC_{16} có giá trị bằng $10 * 16^2 + 11 * 16^1 + 12 * 16^0 = 2748$ (HCS 10)
- Ví dụ trong C:
 - Sử dụng tiền tố 0x
 - Ví dụ số nguyên ở HCS 16: 0xA1BC

Người lập trình thường sử dụng HCS 16 (Hex, vì sao?)

Chuyển đổi HCS 10<=>16

- Chuyển đổi từ HCS 16 sang 10: Sử dụng trọng số vị trí của chữ số:

$$\begin{aligned}A2_{16} &= 10 * 16^1 + 2 * 16^0 \\ &= 160 + 2 \\ &= 162\end{aligned}$$

- Chuyển đổi từ HCS 10 sang 16: Phương pháp chia

$$162 / 16 = 10 \text{ Dư } 2 \quad \text{Độc kết quả từ dưới lên}$$

$$10 / 16 = 0 \text{ Dư } 10 \text{ (A)} \quad A2_{16}$$

Các giải thuật tương tự trường hợp chuyển đổi HCS 10<=>2.

Chuyển đổi HCS $2 \Leftrightarrow 16$

- **Quan sát:** 1 chữ số trong HCS 16 tương đương 4 chữ số trong HCS 2 ($16^1 = 2^4$).
- Chuyển đổi HCS 2 \Rightarrow 16: Nếu số lượng bit không chia hết cho 4 thì bổ sung bit 0 vào đầu (không thay đổi giá trị) cho tới khi số lượng bit chia hết cho 4. Sau đó chuyển đổi từng cụm 4 bits thành các chữ số HCS 16.

$$10\mathbf{1101}1011\mathbf{1001}_2 \Rightarrow 0010\mathbf{1101}1011\mathbf{1001}_2$$

$\quad\quad\quad 2 \quad D \quad B \quad 9_{16}$

- Chuyển đổi HCS 16 \Rightarrow 2: Chuyển đổi từng chữ số thành nhóm 4 bits, sau đó xóa các bits 0 ở đầu (nếu cần).

$\quad\quad\quad 2 \quad D \quad B \quad 9_{16}$

$$0010 \ 1101 \ 1011 \ 1001_2 \Rightarrow 10 \ 1101 \ 1011 \ 1001_2$$

Hệ cơ số 8

- Octal Number System
 - Octo (Latin) => 8
- Các đặc điểm:
 - Sử dụng 8 chữ số: 0, 1, 2, 3, 4, 5, 6, 7
 - Có trọng số vị trí:
 - $1367_8 \neq 3671_8$
 - 25_8 có giá trị bằng $2 * 8^1 + 5 * 8^0 = 21$ (HCS 10)
- Ví dụ trong C:
 - Sử dụng tiền tố 0
 - Ví dụ số nguyên HCS 8: 01367

Người lập trình thường sử dụng HCS 8 (Octo, vì sao?)

Một số giá trị tương đương trong các HCS

HCS 10	HCS 2	HCS 8	HCS 16
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Nội dung

- Cấu trúc chương trình C
- Các hệ đếm
- Kiểu dữ liệu & biến
 - Biểu diễn & Kiểu số nguyên
 - Biểu diễn & Kiểu số thực
 - Khai báo biến thuần

Kiểu dữ liệu

- Kiểu dữ liệu là tính năng của ngôn ngữ lập trình để làm việc với biểu diễn dữ liệu
 - Mô tả cách biểu diễn giá trị như dãy bits và ngược lại
 - Có thể được hiểu như 1 tập hợp các giá trị có cùng định dạng biểu diễn
 - Ví dụ: Với một vùng nhớ nào đó đang lưu một dãy **8-bit** **10101001**:
Chi tiết về các kiểu dữ liệu trong C sẽ được cung cấp sau.
 - Nếu kiểu của nó là mã bù-2 (**số nguyên có dấu**), thì có giá trị = **-87**.
 - Nếu kiểu của nó là **số nguyên không dấu**, thì có giá trị = **169**
 - Nếu kiểu của nó là **ký tự**, thì có thể được đọc như ký tự © (mã số 169) trong 1 bảng mã ASCII mở rộng.

Kiểu dữ liệu₍₂₎

- Kiểu dữ liệu còn là cơ chế an toàn đơn giản để hạn chế lỗi:
 - Trong NNLT C biến, hằng giá trị, kết quả thực hiện biểu thức, v.v.. thậm chí cả hàm đều được định kiểu.
 - Kiểm tra kiểu có thể giúp phát hiện sớm những diễn đạt sai
 - Ví dụ ngăn cản việc áp dụng hàm tính tổng cho 2 chuỗi
 - gán giá trị số thực cho biến con trỏ v.v...
 - Tuy nhiên cơ chế an toàn kiểu cũng làm phát sinh một số vấn đề như ép kiểu, kiểm tra tương thích kiểu.
- C không hỗ trợ cơ chế suy diễn kiểu
 - Người lập trình tự lựa chọn kiểu dữ liệu tối ưu cho từng trường hợp sử dụng.

Biến

- **Biến** là **vùng nhớ** được **đặt tên** và **định kiểu**, là tính năng của NNLT để làm việc với các vùng nhớ
 - Mỗi biến ở 1 thời điểm có thể lưu 1 giá trị bất kỳ thuộc kiểu của nó.
- Các đặc tính cơ bản của biến bao gồm
 - Các đặc điểm lưu trữ
 - Phân cấp bộ nhớ: vùng nhớ RAM hay cao hơn (register)?
 - Thời gian sử dụng: vùng nhớ được duy trì trong suốt thời gian chương trình hoạt động hay trong 1 phạm vi (hàm, lệnh, v.v..) ?
 - Các tính chất gắn với định danh (tên biến)
 - Giới hạn truy cập & Liên kết: Tên biến là riêng tư trong phạm vi 1 khối lệnh, 1 hàm, 1 đơn vị biên dịch? Hay công khai trong toàn chương trình?
 - Các tính chất gắn với kiểu của biến
 - Các đặc điểm đọc/ghi

Nội dung chi tiết sẽ được cung cấp trong các bài sau.

Các kiểu dữ liệu cơ sở trong C

- Còn được gọi là các kiểu dữ liệu định sẵn, các kiểu dữ liệu cơ bản
- Về bản chất **chỉ có 2 nhóm**:
 - **Số nguyên và**
 - **Số thực (dấu chấm động)**
 - Ký tự được quy đổi (2 chiều) thành mã số duy nhất là số nguyên dựa trên bảng mã.
 - Giá trị chân lý (lô-gic) cũng được quy đổi (2 chiều) thành số nguyên: true => 1, false => 0;
 - Các giá trị số cũng có thể được quy đổi thành các giá trị chân lý trong các biểu thức lô-gic: Nếu $x \neq 0$ thì $x \Rightarrow \text{true}$, nếu ngược lại thì $x \Rightarrow \text{false}$
 - *(Nội dung chi tiết được cung cấp sau)*

Các kiểu suy diễn trong C

Người lập trình có thể tự định nghĩa các kiểu mới dựa trên những kiểu dữ liệu đang có. Chúng ta gọi các kiểu được xây dựng dựa trên các kiểu cơ sở là các kiểu suy diễn.

- Mảng (arrays)
- Con trỏ (pointers)
- Cấu trúc (structs)
- Nhóm (unions)
- Kiểu liệt kê (enum)
- Hàm (functions)
- Kiểu số phức (complex)

Học chi tiết sau

Được coi là kiểu suy diễn từ kiểu của giá trị mà nó trả về

Được xây dựng dựa trên kiểu số thực, phần thực và phần ảo có kiểu số thực

Nội dung

- Cấu trúc chương trình C
- Các hệ đếm
- **Kiểu dữ liệu & biến**
 - Biểu diễn & Kiểu số nguyên
 - Biểu diễn & Kiểu số thực
 - Khai báo biến

Biểu diễn số nguyên không dấu

- Số nguyên không dấu:
 - Trong toán học: Số nguyên không âm từ 0 tới $+\infty$
 - Trong hệ thống máy tính:
 - Miền giá trị phụ thuộc vào số bits được sử dụng
 - Nếu sử dụng n bits \Rightarrow miền giá trị là $[0, 2^n - 1]$
 - Vượt ra ngoài khoảng \Rightarrow tràn số
- Dãy bits trong biểu diễn số nguyên không dấu với n bits = dãy bits thu được sau khi thêm bit 0 vào bên trái giá trị số trong HCS 2 cho tới khi đủ n bits
- Ví dụ với $n = 8$: Miền giá trị là $[0, 255]$

0	-	00000000 ₂
5	-	00000101 ₂
18	-	00010010 ₂

Cộng các số nguyên không dấu

$$\begin{array}{r} 13 \\ + 25 \\ \hline 38 \end{array} \quad \begin{array}{r} 00001101_2 \\ + 00011001_2 \\ \hline 00100110_2 \end{array}$$

Thực hiện từ phải qua trái và nhớ 1 khi cần.

$$\begin{array}{r} 130 \\ + 250 \\ \hline 124 \end{array} \quad \begin{array}{r} 10000010_2 \\ + 11111010_2 \\ \hline 01111100_2 \end{array}$$

!Lưu ý: Tràn số

Chỉ lưu 8 bits = phần dư khi chia 256 hoặc 2^8

Trừ các số nguyên không dấu

11

$$\begin{array}{r} 25 \qquad 00011001_2 \\ - 13 \quad - 00001101_2 \\ \hline 12 \qquad 00001100_2 \end{array}$$

Thực hiện từ phải qua trái và mượn 1 khi cần.

11111

$$\begin{array}{r} 130 \qquad 10000010_2 \\ - 250 \quad - 11111010_2 \\ \hline 136 \qquad 10001000_2 \end{array}$$

!Lưu ý: Tràn số

Dịch chuyển số nguyên không dấu

- Dịch chuyển dãy bits sang phải (toán tử \gg trong C): Lấp đầy khoảng trống bên trái với các bits 0

```
100          >> 3 => 12  
011001002 >> 3 => 000011002
```

```
100          >> 5 => 3  
011001002 >> 5 => 000000112
```

*Tương đương
với phép toán
đại số nào?*

- Dịch chuyển dãy bits sang trái (toán tử \ll trong C): Lấp đầy khoảng trống bên phải với các bits 0

```
25          << 3 => 200  
000110012 << 3 => 110010002
```

```
25          << 5 => 32  
000110012 << 5 => 001000002
```

*Tương đương
với phép toán
đại số nào?*

Các phép toán theo bits khác với số nguyên không dấu

- Phủ định theo bits (toán tử \sim trong C) - đảo 0-1

$$\begin{aligned}\sim 100 & \Rightarrow 155 \\ \sim 01100100_2 & \Rightarrow 10011011_2\end{aligned}$$

$$\begin{aligned}\sim 20 & \Rightarrow 235 \\ 00010100_2 & \Rightarrow 11101011_2\end{aligned}$$

- AND theo bits (toán tử $\&$ trong C): 1 = true, 0 = false

$$\begin{array}{rcl}100 & & 01100100_2 \\ \& 20 & \& 00010100_2 \\ \hline & & 00000100_2 \\ & 4 & \end{array}$$

$$\begin{array}{rcl}100 & & 01100100_2 \\ \& 64 & \& 01000000_2 \\ \hline & & 01000000_2 \\ & 64 & \end{array}$$

*Hữu ích để thiết lập bit bất kỳ = 0 (tắt bit)
hoặc đọc giá trị của 1 bit bất kỳ.*

Các phép toán theo bits khác với số nguyên không dấu₍₂₎

- OR theo bits (toán tử | trong C)

100	01100100 ₂
16	00010000 ₂
-----	-----
116	01110100 ₂

Hữu ích để thiết lập bit bất kỳ = 1 (bật bit)

- XOR theo bit (OR loại trừ, toán tử ^ trong C)

100	01100100 ₂
^ 255	^ 11111111 ₂
-----	-----
155	10011011 ₂

$$\begin{aligned}0 \wedge 1 &= 1; \\1 \wedge 1 &= 0; \\1 \wedge 0 &= 1; \\1 \wedge 1 &= 0;\end{aligned}$$

Biểu diễn số nguyên có dấu: Dấu-Trị tuyệt đối

Số nguyên	Biểu diễn
10	00001010
-10	10001010
100	01100100
-100	11100100
8	00001000
-8	10001000
127	01111111
0	00000000
-0	10000000

- MSB biểu diễn dấu: MSB = 0 - Số dương, MSB = 1 - Số âm. Các bits còn lại biểu diễn giá trị tuyệt đối
 - => Phủ định bit dấu để đảo dấu.
- Ví dụ:
 - $100 \Rightarrow 01100100_2$
 - $-100 \Rightarrow \text{neg}(100) \Rightarrow \text{neg}(01100100_2) \Rightarrow 11100100_2$
- Các đặc điểm:
 - Ý tưởng đơn giản;
 - Có 2 biểu diễn số 0;
 - Cần nhiều giải thuật để triển khai phép cộng.

Biểu diễn số nguyên có dấu: Mã bù-1

Số nguyên	Biểu diễn
10	00001010
-10	11110101
100	01100100
-100	10011011
8	00001000
-8	11110111
127	01111111
0	00000000
-0	11111111

- MSB có trọng số = $-(2^{n-1} - 1)$, trong đó n là số lượng bits của biểu diễn.
 - \Rightarrow Phủ định để đảo dấu.
 - $\text{neg}(x) = \sim x$.
- Ví dụ:
 - $25 \Rightarrow 00011001_2$
 - $-25 \Rightarrow \sim(00011001_2) \Rightarrow 11100110_2$
- Các đặc điểm:
 - Ý tưởng đơn giản;
 - Có 2 biểu diễn số 0;
 - Cần nhiều giải thuật để triển khai phép cộng.

Biểu diễn số nguyên có dấu: Mã bù-2

Số nguyên	Biểu diễn
10	00001010
-10	11110110
100	01100100
-100	10011100
8	00001000
-8	11111000
127	01111111
-128	10000000
0	00000000

- MSB có trọng số $= -2^{n-1}$, trong đó n là số lượng bits của biểu diễn.
 - Mã bù-2 của x = mã bù-1 của $x + 1$.
 - Lấy mã bù 2 để đảo dấu.
 - $\text{neg}(x) = \sim x + 1$
- Ví dụ:
 - $20 \Rightarrow 00010100_2$
 - $-20 \Rightarrow \text{neg}(20) \Rightarrow \sim(00010100_2) + 1 \Rightarrow 11101011_2 + 1 \Rightarrow 11101100_2$
- Các đặc điểm:
 - Dải biểu diễn không đối xứng $= [-2^{n-1}, 2^{n-1}-1]$.
 - Một biểu diễn số 0
 - Có thể triển khai phép cộng với 1 giải thuật.

Cộng các số nguyên có dấu (mã bù-2)

$$\begin{array}{r}
 10 \\
 + 20 \\
 \hline
 30
 \end{array}
 \quad
 \begin{array}{r}
 00001010_2 \\
 + 00010100_2 \\
 \hline
 00011110_2
 \end{array}$$

++: Kết
quả đúng

$$\begin{array}{r}
 100 \\
 + 100 \\
 \hline
 -56
 \end{array}
 \quad
 \begin{array}{r}
 11 \quad 1 \\
 01100100_2 \\
 + 01100100_2 \\
 \hline
 11001000_2
 \end{array}$$

++:
Tràn số

$$\begin{array}{r}
 10 \\
 + -20 \\
 \hline
 -10
 \end{array}
 \quad
 \begin{array}{r}
 1 \\
 00001010_2 \\
 + 11101100_2 \\
 \hline
 11110110_2
 \end{array}$$

+-, -+: Kết
quả luôn
đúng

$$\begin{array}{r}
 -10 \\
 + 20 \\
 \hline
 10
 \end{array}
 \quad
 \begin{array}{r}
 1111 \quad 1 \\
 11110110_2 \\
 + 00010100_2 \\
 \hline
 00001010_2
 \end{array}$$

$$\begin{array}{r}
 -10 \\
 + -20 \\
 \hline
 -30
 \end{array}
 \quad
 \begin{array}{r}
 111111 \\
 11110110_2 \\
 + 11101100_2 \\
 \hline
 11100010_2
 \end{array}$$

--: Kết
quả đúng

$$\begin{array}{r}
 -100 \\
 + -30 \\
 \hline
 126
 \end{array}
 \quad
 \begin{array}{r}
 10 \\
 10011100_2 \\
 + 11100010_2 \\
 \hline
 01111110_2
 \end{array}$$

--: Tràn
số

Trừ các số nguyên có dấu

Trừ tuần tự có nhớ

1111 1

10	00001010 ₂
- 20	- 00010100 ₂
----	-----
-10	11110110 ₂

111111

10	00001010 ₂
- -20	- 11101100 ₂
----	-----
30	00011110 ₂

1

-10	11110110 ₂
- -20	- 11101100 ₂
-----	-----
10	00001010 ₂

Cộng với mã bù-2

1

10	00001010 ₂
+ -20	+ 11101100 ₂
----	-----
-10	11110110 ₂

10	00001010 ₂
+ 20	+ 00010100 ₂
----	-----
30	00011110 ₂

1111 1

-10	11110110 ₂
+ 20	+ 00010100 ₂
-----	-----
10	00001010 ₂

Dịch chuyển số nguyên có dấu

- Dịch các bit sang trái (toán tử << trong C): Lấp đầy phần bên phải với các giá trị 0

```
10          << 3 => 80
000010102 << 3 => 010100002

-10         << 3 => -80
111101102 << 3 => 101100002
```

*Tương đương với phép
toán đại số nào?
!Lưu ý: bit dấu*

Dịch chuyển số nguyên có dấu₍₂₎

- Dịch chuyển sang phải: Có nhiều phương án triển khai khác nhau, phía trái có thể được lấp đầy bằng bit dấu (đại số) hoặc được lấp đầy bằng bit 0 (lô-gic).

Lấp đầy phía trái bằng bit 0

10 >> 3 => 1
00001010₂ >> 3 => 00000001

-10 >> 3 => 30
11110110₂ >> 3 => 00011110₂

Lấp đầy phía trái bằng bit dấu

10 >> 3 => 1
00001010₂ >> 3 => 00000001

-10 >> 3 => -1
11110110₂ >> 3 => 11111110₂

- Trong C (toán tử >>), dịch chuyển sang phải có thể là đại số hoặc lô-gic, không có quy ước cụ thể cho trường hợp số có dấu được dịch chuyển là số âm.
 - Nên tránh dịch chuyển số nguyên có dấu.

Các thao tác trên dãy bits khác

- Phủ định theo bits (toán tử \sim trong C)
 - AND theo bits (toán tử $\&$ trong C)
 - OR theo bits (toán tử $|$ trong C)
- Tương tự như với số nguyên không dấu*

Nên tránh sử dụng các thao tác trên dãy bits với số nguyên có dấu. Nếu không thể thì cố gắng chỉ sử dụng các giá trị không âm.

Các kiểu số nguyên trong C

- Quy chuẩn C mô tả 5 cặp kiểu số nguyên (có dấu/không dấu):

Có dấu	Không dấu
signed char	unsigned char
short int	unsigned short int
int	unsigned int
long int	unsigned long int
long long int	unsigned long long int

- Quy chuẩn C không quy định kích thước cụ thể của các kiểu số nguyên, tuy nhiên có một số ràng buộc:
 - Kích thước của kiểu int được lựa chọn theo kích thước tự nhiên trong kiến trúc của môi trường thực thi;
 - Kích thước của kiểu có dấu phải bằng kích thước của kiểu không dấu tương ứng;
 - Miền giá trị phải bao gồm miền giá trị tối thiểu được quy định trong quy chuẩn. Các giới hạn thực tế được mô tả trong <limits.h>

Các kiểu số nguyên trong C₍₂₎

Các kiểu số nguyên trong C có thể được mô tả theo nhiều cách khác nhau nhưng có cùng ý nghĩa

- Có thể giản lược (ngầm định) từ khóa int trong mô tả kiểu và từ khóa signed cho các kiểu số nguyên có dấu.
- Quy chuẩn C không quy định biểu diễn của kiểu char, trình biên dịch có thể mặc định char là signed char hoặc unsigned char (thường là signed char):
 - Người lập trình không nên coi char là signed char hay unsigned char để duy trì tính khả chuyển.

Các kiểu số nguyên trong C₍₃₎

Các kiểu số nguyên trong C có thể được mô tả theo nhiều cách khác nhau nhưng có cùng ý nghĩa

Thường gặp	Ý nghĩa
<i>Có dấu (thường là mã bù-2)</i>	
short	signed short int
int	signed int
long	signed long int
long long	signed long long int
<i>Không dấu (thuần nhị phân)</i>	
unsigned short	unsigned short int
unsigned	unsigned int
unsigned long	unsigned long int
unsigned long long	unsigned long long int

Kiểu số nguyên và biểu diễn số nguyên

- Với kiểu số nguyên không dấu:
 - Quy chuẩn C quy định biểu diễn giá trị ở dạng thuần nhị phân, nhưng không quy định bit căn chỉnh
 - Các triển khai thường sử dụng bit 0 để căn chỉnh
 - => Biểu diễn số nguyên không dấu như trong bài giảng này.
 - Tràn số với kiểu số nguyên không dấu là hành vi xác định, kết quả trong trường hợp tràn số = phần dư của kết quả đúng khi chia cho 2^n .
- Với kiểu số nguyên có dấu:
 - Quy chuẩn C không quy định sử dụng biểu diễn dấu-trị tuyệt đối, mã bù-1 hay mã bù-2, không quy định bit căn chỉnh.
 - Tuy các triển khai thường sử dụng mã bù-2 (GCC sử dụng mã bù-2), nhưng tràn số với số nguyên có dấu là hành vi bất định (*trình biên dịch có thể làm bất kỳ điều gì*).

Các giới hạn đối với kiểu số nguyên

Các giá trị thường gặp trong môi trường 64-bits

Kiểu	#bits	Min	Max
Có dấu (thường là mã bù-2)			
signed char	8	-128	127
short	16	-32768	32767
int	32	-2147483648	2147483647
long	64	-9223372036854775808	9223372036854775807
long long	64		
Không dấu (thuần nhị phân)			
unsigned char	8	0	255
unsigned short	16		65535
unsigned	32		4294967295
unsigned long	64		18446744073709551615
unsigned long long	64		

Nội dung

- Cấu trúc chương trình C
- Các hệ đếm
- **Kiểu dữ liệu & biến**
 - Biểu diễn & Kiểu số nguyên
 - Biểu diễn & Kiểu số thực
 - Khai báo biến thuần

Biểu diễn số thực dấu chấm động

Chuẩn IEEE 754 (2008, 1985)

- Được sử dụng bởi hầu hết các hệ thống máy tính hiện nay
- Định dạng dấu chấm động: $r = (-1)^S * d_0.d_1....d_t * 2^e$
- Biểu diễn của số thực v bao gồm 3 thành phần:
 - Dấu: S ($0 \Rightarrow$ số dương, $1 \Rightarrow$ số âm)
 - Mũ lệch: $E = e + \text{bias}$ (độ lệch, $\text{bias} = e_{\text{max}}$)
 - Dãy chữ số sau dấu . (phần thập phân): $T = d_1 d_2 ... d_{p-1}$
 - Chữ số trước dấu . (d_0) được ngầm định trong E

1 bit

w bits

$t (= p - 1)$ bits

S (Dấu)	E (Mũ lệch)	T (Dãy chữ số có nghĩa sau dấu .)
------------	----------------	--------------------------------------

- Với $0 < E < 2^w - 1$ thì $v = (-1)^S * 2^{E - \text{bias}} * (1 + 2^{1-p} * T)$
 - d_0 được ngầm định = 1 trong trường hợp này.

Biểu diễn số thực dấu chấm động: Các quy ước

- Nếu $E = 2^w - 1$ và $T \neq 0$ thì v là NaN
- Nếu $E = 2^w - 1$ và $T == 0$ thì $v = (-1)^S * (+\infty)$
- Nếu $E = 0$ và $T \neq 0$ thì $v = (-1)^S * 2^{1 - \text{bias}} * (0 + 2^{1-p} * T)$
 - d_0 được ngầm định = 0 trong trường hợp này.
- Nếu $E = 0$ và $T = 0$ thì $v = (-1)^S * (+0)$

Các định dạng cơ bản

- binary32 (độ chính xác đơn) - 32 bits
 - $w = 8$, bias = 127
 - $t = 23$ ($p = 24$)
- binary64 (độ chính xác kép) - 64 bits
 - $w = 11$, bias = 1023
 - $t = 52$ ($p = 53$)
- binary128 (độ chính xác x 4) - 128 bits
 - Có trong phiên bản năm 2008 (thay thế phiên bản 1985).
 - $w = 15$, bias = 16383
 - $t = 112$ ($p = 113$)

Ví dụ 2.5. Số thực dấu chấm động

Phân tích biểu diễn 32-bits:

1**10000****100****101000000000000000000000000000**

- Dấu (1 bit)
 - $S = 1 \Rightarrow$ Số âm
- Mũ lệch (8 bits):
 - $E = 10000100_2 = 132$
 - $\Rightarrow e = 131 - 127 = 5$
- Phần thập phân:
 - $T = 101000000000000000000000000000$
- Giá trị số: $v = (-1) * 2^5 * 1.101_2 =$
$$(-1) * 2^5 * (1 + 1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3}) = -32 * 1.625 = -52$$
 - Hoặc: $v = -1.101_2 * 2^5 = -110100_2 = -52$

Kiểu dữ liệu số thực dấu chấm động trong C

- Quy chuẩn C mô tả 3 kiểu số thực cơ bản:
 - float
 - double
 - long double
 - Không quy định kích thước cụ thể, nhưng có ràng buộc $\text{sizeof(float)} \leq \text{sizeof(double)} \leq \text{sizeof(long double)}$
- Các triển khai phổ biến hiện nay:
 - float - 32 bits (4 bytes) - tương thích với IEEE 754
 - double - 64 bits (8 bytes) - tương thích với IEEE 754
 - long double - có nhiều triển khai với các biểu diễn khác nhau. Trong đó 2 biểu diễn tiêu biểu là:
 - Định dạng 80-bits (IEC 60559)
 - Định dạng 128-bits (IEEE 754, IEC 60559).
 - Một số phiên bản GCC sử dụng biểu diễn 80-bits (IEC 60559) nhưng có thể sử dụng nhiều hơn 10 bytes cho lưu trữ (12 hoặc 16).

Các hệ quả của phương pháp biểu diễn

Giá trị vô cùng nhỏ (epsilon): Là số dương nhỏ nhất mà bạn có thể cộng vào 1.0 và có được kết quả $\neq 1.0$.

- Đối với kiểu float (binary32): $\epsilon \approx 1 \cdot 10^{-7}$
 - Macro: FLT_EPSILON
 - Biến kiểu float không lưu được số: 1.000000001
 - \Rightarrow Tính toán: Trong giới hạn tầm 6 chữ số
 - Macro: FLT_DIG
- Đối với kiểu double (binary64): $\epsilon \approx 2 \cdot 10^{-16}$
 - Macro: DBL_EPSILON
 - \Rightarrow Tính toán: Trong giới hạn tầm 15 chữ số
 - Macro: DBL_DIG

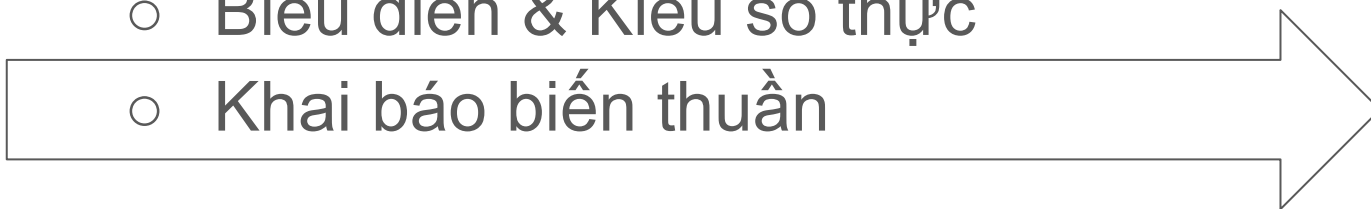
Các giá trị số được cung cấp trong slide này là các giá trị tương đối!

Các hệ quả của phương pháp biểu diễn₍₂₎

- Tương tự như trong hệ cơ số 10, biểu diễn của số hữu tỉ (số thực) có thể có vô hạn chữ số:
 - Ví dụ: giá trị (số) thực của $1/3$ có vô số chữ số
- Khi chuyển số thực từ HCS 10 sang hệ nhị phân cũng có các trường hợp biểu diễn ở hệ nhị phân có vô số chữ số:
 - Ví dụ: Biểu diễn của 0.1 (HCS 10) trong hệ nhị phân
- **!Cần thận trọng với vấn đề làm tròn trong các so sánh**
 - Kết quả gần đúng do biểu diễn không đầy đủ
 - Sai số có thể được tích lũy qua nhiều bước
 - Cần thận trọng khi so sánh 2 số thực
 - Sử dụng giới hạn chính xác (epsilon) trong trường hợp có sai số.
 - $a == b$ vs. $\text{fabs}(a - b) \leq \text{epsilon}$
 - $a > b$ vs. $a - b > \text{epsilon}$

Nội dung

- Cấu trúc chương trình C
- Các hệ đếm
- **Kiểu dữ liệu & biến**
 - Biểu diễn & Kiểu số nguyên
 - Biểu diễn & Kiểu số thực
 - Khai báo biến thuần



Cú pháp

- Trong trường hợp đơn giản nhất chúng ta có thể khai báo biến theo cú pháp:
 - 1 biến: Tên-kiểu tên-biến;
 - Nhiều biến: Tên-kiểu tên-biến-1, tên-biến-2, ...;
- hoặc kết hợp khai báo với khởi tạo:
 - 1 biến: Tên-kiểu tên-biến = giá-trị;
 - Nhiều biến:
Tên-kiểu tên-biến-1 = giá-trị-1, tên-biến-2 = giá-trị-2, ...

Nội dung chi tiết về khai báo sẽ được cung cấp sau

Ví dụ 2.6. Các khai báo biến thuần

```
int a;
```

```
int b, c;
```

```
// - Khai báo a, b, c là các biến kiểu int, khởi tạo theo quy định
```

```
int x = 10, y = 20;
```

```
// - Khai báo và khởi tạo các biến x và y, kiểu int
```

```
float f; // - Biến f có kiểu float, khởi tạo theo quy định
```

```
double d1 = 1.15, d2 = 3.14;
```

```
// Khai báo và khởi tạo các biến d1 và d2, kiểu double
```

Giá trị ban đầu của biến được khai báo nhưng không khởi tạo bằng bao nhiêu?

(Có nhiều trường hợp, nội dung chi tiết được cung cấp sau)

