# BẮT ĐẦU VỚI NGÔN NGỮ C

## PHẦN 1.1 HELLO WORLD

Để tạo một chương trình C đơn giản in dòng chữ "Hello, World " lên màn hình, sử dụng trình soạn thảo văn bản để tạo file mới (ví dụ:

hello.c — phần mở rộng file phải là .c) chứa mã nguồn sau:

**hello.c**

|  |
| --- |
| # include<stdio.h>  int main(void)  {      puts("Hello, World");      return 0;  } |

**Hãy xem từng dòng trong chương trình đơn giản này**

|  |
| --- |
| # include<stdio.h> |

Dòng mã này yêu cầu trình biên dịch cần thêm nội dung của tệp tiêu đề thư viện chuẩn **stdio.h** vào chương trình.

Tệp tiêu đề thường chứa các khai báo của các hàm, macros và kiểu dữ liệu trong thư viện chuẩn.

và bạn cần phải thêm vào tệp tiêu đề trước khi sử dụng chúng. Dòng này thêm vào thư viện **stdio.h** để nó có thể gọi hàm **puts().**

Tìm hiểu thêm về tệp tiêu đề

|  |
| --- |
| int main(void) |

Dòng mã này bắt đầu định nghĩa của một hàm. Nó cho biết tên của hàm (main), kiểu dữ liệu và số lượng các đối số mà nó mong đợi (void, nghĩa là không có) và loại giá trị mà hàm này trả về (int). Theo nguyên tắc, **main** là một tên đặc biệt, nó chỉ dùng để đặt cho function chính của chương trình, và lúc nào chương trình cũng sẽ bắt đầu từ **function main**.

Việc thực thi chương trình bắt đầu trong hàm main().

|  |
| --- |
| {      ...  } |

Dấu ngoặc nhọn (curly braces) được sử dụng thành cặp để chỉ định vị trí bắt đầu và kết thúc một khối mã(khối lệnh). Chúng có thể được sử dụng theo nhiều cách khác nhau, nhưng trong trường hợp này, chúng chỉ định vị trí bắt đầu và kết thúc của một hàm.

|  |
| --- |
| puts("Hello, World"); |

Dòng này gọi hàm puts() để đưa ra văn bản ra đầu ra tiêu chuẩn (mặc định là màn hình), tiếp theo là một dòng mới.

Chuỗi cần đưa ra được bao gồm trong cặp dấu ngoặc đơn().

"Hello, World" là chuỗi sẽ được ghi ra màn hình. Trong ngôn ngữ C, mọi giá trị chuỗi phải nằm trong dấu ngoặc kép "...".

Tìm hiểu thêm về chuỗi.

Trong các chương trình C, mỗi câu lệnh cần được kết thúc bằng dấu chấm phẩy (tức là **;**).

|  |
| --- |
| return 0; |

Khi chúng ta định nghĩa hàm **main(),** chúng ta khai báo nó là một hàm trả về kiểu **int**, có nghĩa là nó cần trả về một giá trị số nguyên. Trong ví dụ này, chúng ta đang trả về giá trị nguyên là **0**, được sử dụng để chỉ rằng chương trình đã kết thúc thành công. Sau câu lệnh **return 0;**, quá trình thực thi sẽ kết thúc.

**Chỉnh sửa chương trình**

Các trình soạn thảo, chỉnh sửa văn bản đơn giản bao gồm **vim** hoặc **gedit** trên **Linux,** hoặc **Notepad** trên **Windows.** Các trình soạn thảo đa nền tảng khác bao gồm **Visual Studio Code** hoặc **Sublime Text.**

Trình chỉnh sửa phải tạo các tệp văn bản thuần túy, không phải RTF hoặc bất kỳ định dạng nào khác.

**Biên dịch và chạy chương trình**

Để chạy chương trình, file nguồn này (hello.c) trước tiên cần được biên dịch thành file thực thi (ví dụ: **hello** trên hệ thống **Unix/Linux** hoặc **hello.exe** trên **Windows**). Điều này được thực hiện bằng cách sử dụng trình biên dịch cho ngôn ngữ C.

**Biên dịch bằng GCC**

GCC **(GNU Compiler Collection)** là một trình biên dịch C được sử dụng rộng rãi. Để sử dụng nó, hãy mở một thiết bị đầu cuối, sử dụng dòng lệnh để điều hướng đến vị trí của tệp nguồn rồi chạy:

|  |
| --- |
| gcc hello.c -o hello |

Nếu không tìm thấy lỗi trong mã nguồn (hello.c), trình biên dịch sẽ tạo một tệp nhị phân, tên của tệp được cung cấp bởi đối số cho tùy chọn dòng lệnh -o (hello). Đây là tập tin thực thi cuối cùng.

Chúng tôi cũng có thể sử dụng các tùy chọn cảnh báo *-Wall -Wextra -Werror*, giúp xác định các sự cố có thể khiến chương trình bị lỗi hoặc tạo ra kết quả không mong muốn. Chúng không cần thiết cho chương trình đơn giản này nhưng đây là cách thêm chúng:

|  |
| --- |
| gcc -Wall -Wextra -Werror -o hello hello.c |

**Sử dụng trình biên dịch clang**

Để biên dịch chương trình bằng **clang**, bạn có thể sử dụng:

|  |
| --- |
| clang -Wall -Wextra -Werror -o hello hello.c |

Theo thiết kế, các tùy chọn dòng lệnh **clang** tương tự như của **GCC.**

**Sử dụng trình biên dịch Microsoft C từ dòng lệnh**

Nếu sử dụng trình biên dịch Microsoft cl.exe trên hệ thống Windows hỗ trợ Visual Studio và nếu tất cả các biến môi trường được đặt, ví dụ C này có thể được biên dịch bằng lệnh sau. Lệnh này sẽ tạo ra tệp hello.exe có thể thực thi được trong thư mục mà lệnh được thực thi trong (Có các tùy chọn cảnh báo như /W3 cho cl, đại khái là tương tự như -Wall vv cho GCC hoặc clang).

|  |
| --- |
| cl hello.c |

**Thực hiện chương trình**

Sau khi được biên dịch, tệp nhị phân sau đó có thể được thực thi bằng cách nhập ./hello trong thiết bị đầu cuối. Khi thực thi, chương trình đã biên dịch sẽ in **Hello, World**, theo sau là một dòng mới, tới dấu nhắc lệnh.

## PHẦN 1.2 BẢN GỐC “HELLO WORLD!” TRONG K&R C

Sau đây là bản gốc "Hello World!" chương trình từ cuốn sách **The C Programming Language** của BrianKernighan và Dennis Ritchie (Ritchie là nhà phát triển ban đầu của ngôn ngữ lập trình C tại Bell Labs), được gọi là "K&R":

|  |
| --- |
| Version = K&R  #include <stdio.h>  main()  {   printf("hello, world\n");  } |

Lưu ý rằng **The C Programming Language** không được chuẩn hóa vào thời điểm viết ấn bản đầu tiên của cuốn sách này (1978), và chương trình này có thể sẽ không được biên dịch trên hầu hết các trình biên dịch hiện đại trừ khi chúng được hướng dẫn chấp nhận mã C90.

Câu ví dụ đầu tiên trong cuốn sách K&R (**The C Programming Language**) hiện được coi là chất lượng kém, một phần là do thiếu một kiểu trả về rõ ràng cho **main()** và một phần là do thiếu câu lệnh **return**. Phiên bản thứ hai của cuốn sách được viết cho tiêu chuẩn C89 cũ. Trong C89, kiểu của main sẽ mặc định là int, nhưng ví dụ K&R không trả về một giá trị xác định cho môi trường. Trong các tiêu chuẩn C99 và sau này, kiểu trả về là bắt buộc, nhưng có thể bỏ qua câu lệnh **return** của main (**và chỉ main**), vì có một trường hợp đặc biệt được giới thiệu với C99 5.1.2.2.3 - nó tương đương với việc trả về 0, biểu thị thành công.

Dạng chính được khuyến nghị và di động nhất cho các hệ thống được lưu trữ là **int main(void)** khi chương trình không sử dụng bất kỳ đối số dòng lệnh nào, hoặc **int main(int argc, char \*\*argv)** khi chương trình sử dụng các đối số ở các dòng lệnh.

**C90 §5.1.2.2.3 Kết thúc chương trình**

Một câu lệnh **return** từ lời gọi ban đầu tới hàm **main** tương đương với việc gọi hàm **exit** với giá trị trả về của hàm **main** là **đối số**. Nếu hàm main thực hiện một câu lệnh **return** mà không xác định giá trị, trạng thái kết thúc trả về cho môi trường chủ không được xác định.

**C90 §6.6.6.4 Câu lệnh return**

Nếu một câu lệnh **return** mà không có biểu thức được thực thi và giá trị của lời gọi hàm được sử dụng bởi người gọi, hành vi đó là không xác định. Đạt đến dấu **}** kết thúc một hàm tương đương với việc thực thi một câu lệnh **return** mà không có biểu thức.

**C99 §5.1.2.2.3 Kết thúc chương trình**

Nếu kiểu trả về của hàm **main** là một kiểu tương thích với **int**, một câu lệnh **return** từ lời gọi ban đầu tới hàm **main** tương đương với việc gọi hàm **exit** với giá trị trả về của hàm main là đối số; đạt đến dấu **}** kết thúc hàm main sẽ trả về giá trị 0. Nếu kiểu trả về không tương thích với int, trạng thái kết thúc trả về cho môi trường chủ không được xác định.

# COMMENTS

Chú thích (comments) được sử dụng để chỉ điều gì đó cho người đọc mã nguồn. Chú thích được coi như một phần trống trên mã nguồn và không thay đổi bất kỳ ý nghĩa thực sự nào của mã. Có hai cú pháp được sử dụng cho chú thích trong ngôn ngữ C, đó là **/\* \*/** (bản gốc) và **//** (bản mới hơn 1 chút). Một số hệ thống tài liệu sử dụng chú thích được định dạng đặc biệt để hỗ trợ tạo tài liệu cho mã nguồn.

## PHẦN 2.1 CHÚ THÍCH BẰNG CÁCH SỬ DỤNG TIỀN XỬ LÝ(PREPROCESSOR)

Một phần lớn mã nguồn cũng có thể bị "chú thích" bằng cách sử dụng các chỉ thị tiền xử lý **#if 0** và **#endif.** Điều này hữu ích khi mã chứa các chú thích nhiều dòng mà không thể lồng nhau nếu không có cơ chế chú thích này.

|  |
| --- |
| #if 0 /\* Bắt đầu "chú thích", bất cứ điều gì từ đây trở đi sẽ bị loại bỏ bởi tiền xử lý \*/  /\* Một lượng mã lớn với các chú thích nhiều dòng \*/  int foo()  {  /\* một số câu lệnh \*/  ...  /\* ... một chú thích mô tả câu lệnh if ... \*/  if (someTest) {   /\* some more comments \*/  return 1;  }  return 0;  }  #endif /\* 0 \*/  /\* Mã từ đây trở đi không bị "chú thích" (được bao gồm trong tệp thực thi biên dịch) \*/ |

## PHẦN 2.2 CHÚ THÍCH ĐƯỢC GIỚI HẠN BẰNG /\* \*/

Một chú thích bắt đầu với một dấu gạch chéo kề ngay sau đó là một dấu hoa thị (/\*) và kết thúc ngay khi gặp một dấu hoa thị kề ngay sau đó là một dấu gạch chéo (\*/). Mọi thứ nằm giữa các cặp ký tự này đều là chú thích và được coi là một phần trống (tức là bị bỏ qua) bởi trình biên dịch.

|  |
| --- |
| /\* Đây là một chú thích \*/ |

Chú thích ở trên là chú thích trên một dòng. Chú thích kiểu /\* này có thể trải dài trên nhiều dòng, như sau:

|  |
| --- |
| /\* Đây là một chú thích  nhiều dòng \*/ |

Mặc dù điều này không bắt buộc, quy ước phong cách thông thường với chú thích trên nhiều dòng là đặt khoảng trắng và dấu hoa thị ở các dòng sau đầu tiên và đặt /\* và \*/ trên các dòng mới, sao cho chúng xếp thành hàng:

|  |
| --- |
| /\* Đây là   \*một chú   \* thích   \*nhiều dòng  \*/ |

Những dấu hoa thị thêm không có tác dụng chức năng nào đối với chú thích vì chúng không có dấu gạch chéo liên quan.

Các chú thích kiểu /\* này có thể được sử dụng trên một dòng riêng biệt, ở cuối một dòng mã hoặc ngay cả trong các dòng mã:

|  |
| --- |
| /\* Chú thích này nằm trên một dòng riêng biệt \*/  if(x && y) { /\* Chú thích này nằm ở cuối dòng mã \*/      if ((complexCondition1) /\* Chú thích này nằm trong một dòng mã \*/          && (complexCondition2)) {      /\* Chú thích này nằm trong một câu lệnh if, trên một dòng riêng biệt \*/      }  } |

Chú thích không thể lồng nhau. Điều này xảy ra vì bất kỳ /\* nào tiếp theo sẽ bị bỏ qua (là một phần của chú thích) và đầu tiên \*/ gặp được sẽ được coi là kết thúc chú thích. Chú thích trong ví dụ dưới đây sẽ không hoạt động:

|  |
| --- |
| /\* Chú thích bên ngoài, có nghĩa là điều này bị bỏ qua => /\* chú thích nội bộ cố gắng \*/ <= kết thúc chú thích,  không phải chú thích này => \*/ |

Để chú thích các khối mã chứa chú thích kiểu này, mà nếu không sẽ bị lồng nhau, bạn có thể tham khảo ví dụ về Chú thích bằng cách sử dụng tiền xử lý.

## PHẦN 2.3 CHÚ THÍCH ĐƯỢC GIỚI HẠN BẰNG DẤU //

Phiên bản ≥ C99

C99 đã giới thiệu việc sử dụng chú thích trên một dòng theo kiểu C++. Loại chú thích này bắt đầu bằng hai dấu gạch chéo kề nhau và kéo dài đến cuối dòng:

|  |
| --- |
| // Mỗi dòng trong số này đều là một chú thích trên một dòng  // Chú ý rằng mỗi dòng phải bắt đầu bằng  // hai dấu gạch chéo kép liên tiếp (//) |

Loại chú thích này có thể được sử dụng trên một dòng riêng biệt hoặc ở cuối một dòng mã. Tuy nhiên, do chú thích này kéo dài đến cuối dòng, nên chúng không thể được sử dụng trong một dòng mã.

|  |
| --- |
| // Chú thích này nằm trên một dòng riêng biệt  if (x && y) { // Chú thích này nằm ở cuối dòng  // Chú thích này nằm trong một câu lệnh if, trên một dòng riêng biệt  } |

## PHẦN 2.4 RỦI RO CÓ THỂ XẢY RA DO TRIGRAPH

Trong quá trình viết chú thích được giới hạn bởi //, có thể xảy ra lỗi chính tả ảnh hưởng đến hoạt động dự kiến của chú thích. Nếu bạn gõ sai như sau:

|  |
| --- |
| int a = 10 //Sao tôi làm điều này??/ |

Dấu gạch chéo / ở cuối là một lỗi chính tả nhưng giờ đây sẽ được hiểu là dấu gạch chéo ngược . Điều này xảy ra vì ??/ tạo thành một trigraph. Trigraph ??/ thực tế là một cách viết dài cho dấu gạch chéo ngược , đây là ký hiệu tiếp tục dòng. Điều này có nghĩa là trình biên dịch sẽ hiểu rằng dòng tiếp theo là tiếp tục của dòng hiện tại, tức là tiếp tục của chú thích, điều này có thể không phải ý định ban đầu.

|  |
| --- |
| int foo = 20; // Bắt đầu từ 20 ??/  int bar = 0;  // Dòng sau sẽ gây lỗi biên dịch (biến 'bar' chưa được khai báo)  // vì 'int bar = 0;' là một phần của chú thích trên dòng trước đó  bar += foo; |

# KIỂU DỮ LIỆU

## PHẦN 3.1 GIẢI THÍCH CÁC KHAI BÁO

Một đặc điểm cú pháp đặc biệt của ngôn ngữ C là các khai báo phản ánh việc sử dụng của đối tượng được khai báo như trong một biểu thức thông thường.

Tập hợp các toán tử sau đây có cùng mức độ ưu tiên và tính kết hợp được tái sử dụng trong các khai báo, bao gồm:

* Toán tử unary \* "dereference" (giải tham chiếu) thể hiện một con trỏ.
* Toán tử binary [] "array subscription" (phần tử mảng) thể hiện một mảng.
* Toán tử (1+n)-ary () "function call" (gọi hàm) thể hiện một hàm.
* Dấu ngoặc () nhóm nhưng ghi đè mức độ ưu tiên và tính kết hợp của các toán tử còn lại được liệt kê.

Ba toán tử trên có độ ưu tiên và tính kết hợp như sau:

|  |  |  |
| --- | --- | --- |
| Toán tử | Mức độ ưu tiên | Tính kết hợp |
| [ ] Phần tử mảng | 1 | Trái sang phải |
| ( ) Gọi hàm | 1 | Trái sang phải |
| \*Giải tham chiếu | 2 | Phải sang trái |

Khi giải thích các khai báo, người đọc cần bắt đầu từ từng định danh và áp dụng các toán tử kề cạnh theo thứ tự đúng như trong bảng trên. Mỗi lần áp dụng một toán tử, ta có thể thay thế nó bằng các từ tiếng Anh sau đây:

|  |  |
| --- | --- |
| Biểu thức | Giải thích |
| Thing[X] | Kích thức của mảng X là… |
| Thing(x1,x2,x3) | 1 hàm nhận x1 x2 x3 và trả về … |
| \*Thing | Một con trỏ trỏ tới .. |

Theo đó phần bắt đầu của việc giải thích bằng tiếng Anh sẽ luôn bắt đầu với định danh và kết thúc với kiểu dữ liệu nằm ở phía bên trái của khai báo.

**Ví dụ**

|  |
| --- |
| char \*names[20]; |

[] được ưu tiên hơn \*, vì vậy cách hiểu là: **names** là một mảng có kích thước 20 của một con trỏ trỏ tới char.

|  |
| --- |
| char (\*place)[10]; |

Trong trường hợp sử dụng dấu ngoặc () để ghi đè mức độ ưu tiên, toán tử \* được áp dụng trước: **place** là một con trỏ tới một mảng có kích thước là 10 của char.

|  |
| --- |
| int fn(long, short); |

Ở đây không có ưu tiên cần quan tâm: fn là một hàm nhận tham số long và short, và trả về kiểu int.

|  |
| --- |
| int \*fn(void); |

Dấu ngoặc () được áp dụng trước: fn là một hàm nhận tham số void và trả về một con trỏ tới kiểu int.

|  |
| --- |
| int (\*fp)(void); |

Ghi đè mức độ ưu tiên của (): fp là một con trỏ tới một hàm nhận tham số void và trả về kiểu int.

|  |
| --- |
| int arr[5][8]; |

Mảng đa chiều không phải là một ngoại lệ trong quy tắc; các toán tử [] được áp dụng theo thứ tự từ trái sang phải theo tính kết hợp được liệt kê trong bảng: arr là một mảng có kích thước 5 của một mảng có kích thước 8 của kiểu int.

|  |
| --- |
| int \*\*ptr; |

Hai toán tử dereference có cùng mức độ ưu tiên, do đó tính kết hợp có hiệu lực. Các toán tử được áp dụng theo thứ tự từ phải sang trái: ptr là một con trỏ tới một con trỏ tới kiểu int.

**Khai báo nhiều biến**

Dấu phẩy có thể được sử dụng làm dấu phân tách (không hoạt động như toán tử dấu phẩy) để phân cách các khai báo đa biến trong cùng một câu lệnh. Câu lệnh sau chứa năm khai báo:

|  |
| --- |
| int fn(void), \*ptr, (\*fp)(int), arr[10][20], num; |

Các đối tượng được khai báo trong ví dụ trên là:

* fn: một hàm không tham số và trả về kiểu int.
* ptr: một con trỏ tới kiểu int.
* fp: một con trỏ hàm nhận tham số kiểu int và trả về kiểu int.
* arr: một mảng có kích thước 10, mỗi phần tử là một mảng có kích thước 20 kiểu int.
* num: kiểu int.

**Giải thích thay thế**

Bởi vì các khai báo phản ánh việc sử dụng, nên một khai báo cũng có thể được giải thích dựa trên các toán tử có thể được áp dụng lên đối tượng và kiểu dữ liệu cuối cùng của biểu thức đó. Kiểu dữ liệu nằm ở phía bên trái là kết quả cuối cùng sau khi áp dụng tất cả các toán tử.

|  |
| --- |
| /\*   \*Sử dụng toán tử [] để lấy phần tử của "arr" và sau đó dùng toán tử dereference để lấy giá trị,   \*ta được một kết quả kiểu "char".   \*Cụ thể: \*arr[5] có kiểu "char".   \*/  char arr[20];  /\*   \*Gọi hàm "fn" trả về một kết quả kiểu "int".   \*Cụ thể: fn('b') có kiểu "int".   \*/  int fn(char);  /\*   \*Sử dụng toán tử dereference để lấy giá trị của "fp", sau đó gọi nó trả về một kết quả kiểu "int".   \*Cụ thể: (\*fp)() có kiểu "int".   \*/  int (fp)(void);  /\*   \*Sử dụng toán tử [] hai lần để lấy phần tử của "strings", sau đó sử dụng toán tử dereference để lấy giá trị,   \*ta được một kết quả kiểu "char".   \*Cụ thể: \*strings[5][15] có kiểu "char".   \*/  char \*strings[10][20]; |

## PHẦN 3.2 KIỂU SỐ NGUYÊN CÓ ĐỘ RỘNG CỐ ĐỊNH ( KỂ TỪ C99)

Phiên bản ≥ C99 Tiêu đề <stdint.h> cung cấp một số định nghĩa kiểu số nguyên có độ rộng cố định. Các kiểu này là tùy chọn và chỉ được cung cấp nếu nền tảng có một kiểu số nguyên có độ rộng tương ứng và nếu kiểu có dấu tương ứng có biểu diễn bù hai của các giá trị âm. Xem phần lưu ý để biết các gợi ý về việc sử dụng các kiểu có độ rộng cố định.

|  |
| --- |
| /\* Các kiểu phổ biến thường được sử dụng bao gồm \*/  *uint32\_t* u32 = 32; / có độ rộng chính xác 32 bit /  uint8\_t u8 = 255; / có độ rộng chính xác 8 bit /  int64\_t i64 = -65; / có độ rộng chính xác 64 bit trong biểu diễn bù hai \*/ |

## PHẦN 3.3 KIỂU SỐ NGUYÊN VÀ HẰNG SỐ

Các số nguyên có dấu có thể thuộc các kiểu sau đây (int sau từ short hoặc long là tùy chọn):

|  |
| --- |
| signed char c = 127; /\* yêu cầu có độ rộng 1 byte, xem phần lưu ý để biết thêm thông tin \*/  signed short int si = 32767; /\* yêu cầu có ít nhất 16 bit \*/  signed int i = 32767; /\* yêu cầu có ít nhất 16 bit \*/  signed long int li = 2147483647; /\* yêu cầu có ít nhất 32 bit \*/  Phiên bản ≥ C99  signed long long int li = 2147483647; /\* yêu cầu có ít nhất 64 bit \*/ |

Mỗi kiểu số nguyên có dấu này đều có một phiên bản không dấu tương ứng.

|  |
| --- |
| unsigned int i = 65535;  unsigned short = 2767;  unsigned char = 255; |

Đối với tất cả các kiểu, ngoại trừ char, phiên bản có dấu được cho là mặc định nếu phần có dấu hoặc không dấu bị bỏ qua. Kiểu char tạo thành một kiểu ký tự thứ ba, khác với signed char và unsigned char, và sự có dấu (hoặc không) phụ thuộc vào nền tảng.

Có các loại hằng số số nguyên khác nhau (được gọi là chữ số trong ngôn ngữ C) có thể được viết theo các cơ số khác nhau và có độ rộng khác nhau, dựa trên tiền tố hoặc hậu tố của chúng.

|  |
| --- |
| /\* Các biến sau được khởi tạo với cùng một giá trị: \*/  int d = 42; /\* hằng số thập phân (cơ số 10) \*/  int o = 052; /\* hằng số bát phân (cơ số 8) \*/  int x = 0xaf; /\* hằng số thập lục phân (cơ số 16) \*/  int X = 0XAf; /\*\*/ (chữ cái 'a' đến 'f' (không phân biệt chữ hoa chữ thường) biểu thị từ 10 đến 15) \*/ |

Hằng số thập phân luôn luôn có dấu**(signed)**. Hằng số thập lục phân bắt đầu bằng 0x hoặc 0X và hằng số bát phân bắt đầu chỉ với chữ số 0. Hai loại hằng số sau có thể có dấu**(signed)** hoặc không dấu**(unsigned)** tùy thuộc vào việc giá trị có phù hợp với kiểu có dấu hay không.

|  |
| --- |
| /\* Tiền tố để chỉ độ rộng và dấu \*/  long int i = 0x32; /\* không có tiền tố đại diện cho int hoặc long int \*/  unsigned int ui = 65535u; /\* u hoặc U đại diện cho unsigned int hoặc unsigned long int \*/  long int li = 65536l; /\* l hoặc L đại diện cho long int \*/ |

Nếu không có tiền tố, hằng số có kiểu dữ liệu là kiểu đầu tiên mà giá trị của nó phù hợp, nghĩa là hằng số thập phân lớn hơn INT\_MAX sẽ có kiểu long nếu có thể, hoặc long long nếu không được.

Tập tin tiêu đề <limits.h> mô tả các giới hạn của số nguyên như sau. Các giá trị xác định bởi người thực hiện (implementation-defined) phải có giá trị tuyệt đối lớn hơn hoặc bằng giá trị được hiển thị dưới đây, với cùng dấu.

|  |  |  |
| --- | --- | --- |
| MARCO | Kiểu dữ liệu | Giá trị |
| CHAR\_BIT | smallest object that is not a bit-field (byte) 8 |  |
| SCHAR\_MIN | signed char | -127 / -(27 - 1) |
| SCHAR\_MAX | signed char | +127 / 27 - 1 |
| UCHAR\_MAX | unsigned char | 255 / 28 - 1 |
| CHAR\_MIN | char | see below |
| CHAR\_MAX | char | see below |
| SHRT\_MIN | short int | -32767 / -(215 - 1) |
| SHRT\_MAX | short int | +32767 / 215 - 1 |
| USHRT\_MAX | unsigned short int | 65535 / 216 - 1 |
| INT\_MIN | int | -32767 / -(215 - 1) |
| INT\_MAX | int | +32767 / 215 - 1 |
| UINT\_MAX | unsigned int | 65535 / 216 - 1 |
| LONG\_MIN | long int | -2147483647 / -(231 - 1) |
| LONG\_MAX | long int | +2147483647 / 231 - 1 |
| ULONG\_MAX | unsigned long int | 4294967295 / 232 - 1 |
| Version ≥ C99 |  |  |
| LLONG\_MIN | long long int | -9223372036854775807 / -(263 - 1) |
| LLONG\_MAX | long long int | +9223372036854775807 / 263 - 1 |
| ULLONG\_MAX | unsigned long long int | 18446744073709551615 / 264 - 1 |

Nếu giá trị của một đối tượng có kiểu char được mở rộng dấu khi sử dụng trong một biểu thức, giá trị của CHAR\_MIN sẽ giống như SCHAR\_MIN và giá trị của CHAR\_MAX sẽ giống như SCHAR\_MAX. Nếu giá trị của một đối tượng có kiểu char không được mở rộng dấu khi sử dụng trong một biểu thức, giá trị của CHAR\_MIN sẽ là 0 và giá trị của CHAR\_MAX sẽ giống như UCHAR\_MAX.

Phiên bản ≥ C99

Tiêu chuẩn C99 đã thêm một tiêu đề mới, <stdint.h>, chứa các định nghĩa cho các số nguyên có chiều rộng cố định. Xem ví dụ số nguyên chiều rộng cố định để được giải thích sâu hơn.

## PHẦN 3.4 HẰNG SỐ THỰC

Ngôn ngữ C có ba kiểu số thực dấu phẩy động bắt buộc, đó là **float**, **double** và **long double**.

|  |
| --- |
| float f = 0.314f; /\* Hậu tố f hoặc F chỉ định kiểu float \*/  double d = 0.314; /\* Không có hậu tố chỉ định kiểu double \*/  long double ld = 0.314l; /\* Hậu tố l hoặc L chỉ định kiểu long double \*/  /\* Các phần khác nhau trong một định nghĩa số thực là tùy chọn \*/  double x = 1.; /\* Hợp lệ, phần thập phân là tùy chọn \*/  double y = .1; /\* Hợp lệ, phần nguyên là tùy chọn \*/  /\* Chúng cũng có thể được định nghĩa trong dạng ký hiệu khoa học \*/  double sd = 1.2e3; /\* Phần thập phân 1.2 được nhân với 10^3, tức là 1200.0 \*/ |

Tiêu đề <float.h> xác định các giới hạn khác nhau cho các hoạt động của dấu phẩy động Số học dấu phẩy động được xác định triển khai. Tuy nhiên, hầu hết các nền tảng hiện đại (arm, x86, x86\_64, MIPS) đều sử dụng các hoạt động của dấu phẩy động IEEE 754. C cũng có ba loại dấu phẩy động phức tạp tùy chọn bắt nguồn từ loại trên.

## PHẦN 3.5 CHUỖI KÍ TỰ

Một chuỗi ký tự trong ngôn ngữ C là một chuỗi các ký tự, kết thúc bằng một ký tự số không (literal zero)

|  |
| --- |
| char\* str = "hello, world"; /\* chuỗi ký tự nguyên mẫu (string literal) \*/  /\* chuỗi ký tự nguyên mẫu có thể được sử dụng để khởi tạo mảng \*/  char a1[] = "abc"; /\* a1 là mảng char[4] chứa {'a','b','c','\0'} \*/  char a2[4] = "abc"; /\* tương tự a1 \*/  char a3[3] = "abc"; /\* a3 là mảng char[3] chứa {'a','b','c'}, thiếu ký tự '\0' \*/ |

Chuỗi ký tự nguyên mẫu trong C **không thể sửa đổi**, và việc cố gắng thay đổi giá trị của chúng sẽ dẫn đến hành vi không xác định. Chuỗi ký tự nguyên mẫu thường được lưu trữ trong bộ nhớ chỉ đọc (read-only memory), chẳng hạn như phần .rodata, và mọi cố gắng sửa đổi giá trị của chúng có thể dẫn đến kết quả không mong muốn.

|  |
| --- |
| char\* s = "foobar";  s[0] = 'F'; /\* hành vi không xác định \*/  /\* Thực hành tốt là đánh dấu chuỗi nguyên mẫu là hằng số bằng cách sử dụng const \*/  char const s1 = "foobar";  s1[0] = 'F'; /\* lỗi biên dịch! \*/ |

Nhiều chuỗi ký tự được nối vào thời gian biên dịch, có nghĩa là bạn có thể viết cấu trúc như thế này.

|  |
| --- |
| Phiên bản trước C99  /\* Chỉ có thể nối hai chuỗi ký tự hẹp (narrow) hoặc hai chuỗi ký tự rộng (wide) \*/  char\* s = "Hello, " "World";  Phiên bản C99 trở lên  /\* Từ phiên bản C99 trở đi, có thể nối nhiều hơn hai chuỗi \*/  /\* Quá trình nối chuỗi phụ thuộc vào triển khai cụ thể \*/  char\* s1 = "Hello" ", " "World";  /\* Một số cách sử dụng phổ biến là nối chuỗi định dạng (format strings) \*/  char\* fmt = "%" PRId16; /\* Macro PRId16 được định nghĩa từ C99 \*/ |

Chuỗi ký tự nguyên mẫu, tương tự như hằng số ký tự, hỗ trợ các bộ ký tự khác nhau.

|  |
| --- |
| /\* Chuỗi ký tự thông thường, kiểu char[] \*/  char\* s1 = "abc";  /\* Chuỗi ký tự rộng (wide character), kiểu wchar\_t[] \*/  *wchar\_t*\* s2 = L"abc";  Phiên bản C11 trở lên  /\* Chuỗi ký tự UTF-8, kiểu char[] \*/  char\* s3 = u8"abc";  /\* Chuỗi ký tự rộng 16-bit, kiểu char16\_t[] \*/  *char16\_t*\* s4 = u"abc";  /\* Chuỗi ký tự rộng 32-bit, kiểu char32\_t[] \*/  *char32\_t*\* s5 = U"abc"; |

# TOÁN TỬ

Toán tử trong ngôn ngữ lập trình là một ký hiệu cho biết trình biên dịch hoặc trình thông dịch thực hiện một phép toán, phép toán quan hệ hoặc logic cụ thể và tạo ra kết quả cuối cùng

Ngôn ngữ C có nhiều toán tử mạnh mẽ. Nhiều toán tử trong C là toán tử nhị phân, có nghĩa là chúng có hai toán hạng. Ví dụ, trong biểu thức a / b, / là một toán tử nhị phân nhận hai toán hạng (a, b). Ngoài ra, còn có một số toán tử một ngôi (unary operators) chỉ yêu cầu một toán hạng (ví dụ: ~, ++), và chỉ có một toán tử 3 ngôi duy nhất ?

## PHẦN 4.1 TOÁN TỬ QUAN HỆ

Các toán tử quan hệ trong ngôn ngữ C kiểm tra xem một mối quan hệ cụ thể giữa hai toán hạng có đúng hay không. Kết quả được đánh giá là 1 (đúng) hoặc 0 (sai). Kết quả này thường được sử dụng để ảnh hưởng đến luồng điều khiển (thông qua câu lệnh if, while, for), nhưng cũng có thể được lưu trữ trong các biến.

### Bằng “==”

Kiểm tra xem các toán hạng được cung cấp có bằng nhau không.

|  |
| --- |
| 1 == 0; /\* đánh giá thành 0.(Sai) \*/  1 == 1; /\* đánh giá thành 1.(Đúng) \*/  int x = 5;  int y = 5;  int \*xptr = &x, yptr = &y;  xptr == yptr; /\* đánh giá thành 0, các toán hạng giữ các địa chỉ vùng nhớ khác nhau.(Sai) \*/  \*xptr == yptr; /\* đánh giá thành 1, các toán hạng trỏ tới các vị trí vùng nhớ chứa cùng một giá trị.(Đúng) \*/ |

**Chú ý:** Không nên nhầm lẫn toán tử này với toán tử gán “=” !!!

### Không bằng (Khác) “!=”

Kiểm tra xem các toán hạng được cung cấp có bằng nhau không

|  |
| --- |
| 1 == 0; /\* đánh giá thành 0.(Sai) \*/  1 == 1; /\* đánh giá thành 1.(Đúng) \*/  int x = 5;  int y = 5;  int \*xptr = &x, yptr = &y;  xptr == yptr; /\* đánh giá thành 0, các toán hạng giữ các địa chỉ vùng nhớ khác nhau.(Sai) \*/  \*xptr == yptr; /\* đánh giá thành 1, các toán hạng trỏ tới các vị trí vùng nhớ chứa cùng một giá trị.(Đúng) \*/ |

Toán tử này trả về kết quả ngược lại với kết quả của toán tử **bằng bằng** (==).

### Not “!”

Kiểm tra xem một đối tượng có bằng 0 hay không.

Toán tử "!" cũng có thể được sử dụng trực tiếp với một biến như sau:

|  |
| --- |
| !someVal |

Điểu này có tác dụng tương tự như

|  |
| --- |
| someVal == 0 |

### So sánh lớn hơn “>”

Kiểm tra xem toán hạng bên trái có giá trị lớn hơn toán hạng bên phải hay không

|  |
| --- |
| 5 > 4 /\* trả về 1.(Đúng) \*/  4 > 5 /\* trả về 0.(Sai) \*/   1. > 4 /\* trả về 0.(Sai) \*/ |

### 4.1.5 So sánh bé hơn “<”

Kiểm tra xem toán hạng bên trái có giá trị bé hơn toán hạng bên phải hay không

|  |
| --- |
| 5 < 4 /\* trả về 0.(Sai) \*/  4 < 5 /\* trả về 1.(Đúng) \*/   1. < 4 /\* trả về 0.(Sai) \*/ |

### 4.1.6 So sánh lớn hơn hoặc bằng “>=”

Kiểm tra xem toán hạng bên trái có giá trị lớn hơn hoặc bằng toán hạng bên phải hay không

|  |
| --- |
| 5 >= 4 /\* trả về 1.(Đúng) \*/  4 >= 5 /\* trả về 0.(Sai) \*/   1. >= 4 /\* trả về 1.(Đúng) \*/ |

### 4.1.7 So sánh bé hơn hoặc bằng “<=”

Kiểm tra xem toán hạng bên trái có giá trị bé hơn hoặc bằng toán hạng bên phải hay không

|  |
| --- |
| 5 <= 4 /\* trả về 0.(Sai) \*/  4 <= 5 /\* trả về 1.(Đúng) \*/  4 <= 4 /\* trả về 1.(Đúng) \*/ |

## PHẦN 4.2 TOÁN TỬ ĐIỀU KIỆN/ TOÁN TỬ 3 NGÔI

Toán tử 3 ngôi "?" trong ngôn ngữ C được sử dụng để thực hiện một phép toán 3 ngôi (ternary operation). Nó đánh giá toán hạng đầu tiên và nếu giá trị kết quả khác 0, nó đánh giá toán hạng thứ hai. Ngược lại, nếu giá trị kết quả bằng 0, nó đánh giá toán hạng thứ ba.

|  |
| --- |
| a = b ? c : d;  /\*Tương đương với\*/  if (b)      a = c;  else      a = d; |

Đoạn mã sau đây mô tả cú pháp của toán tử 3 ngôi: **điều kiện ? kết quả nếu đúng : kết quả nếu sai**. Mỗi giá trị có thể là kết quả của một biểu thức đã được đánh giá.

|  |
| --- |
| int x = 5;      int y = 42;      printf("%i, %i\n", 1 ? x : y, 0 ? x : y); /\* Kết quả "5, 42" \*/ |

Toán tử điều kiện có thể được lồng vào nhau. Ví dụ: đoạn mã sau xác định số lớn hơn trong ba số:

|  |
| --- |
| big= a > b  ? (a > c ? a : c)              : (b > c ? b : c); |

Ví dụ sau ghi số nguyên chẵn vào một tệp và số nguyên lẻ vào tệp khác

|  |
| --- |
| #include<stdio.h>  int main()  {  *FILE* \*even, \*odds;      int n = 10;  *size\_t* k = 0;      even = fopen("even.txt", "w");      odds = fopen("odds.txt", "w");      for(k = 1; k < n + 1; k++)      {          k%2==0 ? fprintf(even, "\t%5d\n", k)                  : fprintf(odds, "\t%5d\n", k);      }      fclose(even);      fclose(odds);      return 0;  } |

Toán tử điều kiện liên kết từ phải sang trái. Hãy xem xét những điều sau đây:

|  |
| --- |
| exp1 ? exp2 : exp3 ? exp4 : exp5 |

Vì liên kết từ phải sang trái, biểu thức trên được đánh giá là:

|  |
| --- |
| exp1 ? exp2 : ( exp3 ? exp4 : exp5 ) |

## PHẦN 4.3 TOÁN TỬ BITWISE

Toán tử bitwise có thể được sử dụng để thực hiện thao tác cấp độ bit trên các biến. Dưới đây là danh sách tất cả sáu toán tử bitwise được hỗ trợ trong C:

|  |  |
| --- | --- |
| Kí hiệu | Toán tử |
| & | Toán tử AND |
| | | Toán tử OR |
| ^ | Toán tử XOR |
| ~ | Toán tử phủ định NOT |
| << | Toán tử dịch trái |
| >> | Toán tử dịch phải |

Chương trình sau đây minh họa việc sử dụng tất cả các toán tử bitwise:

|  |
| --- |
| #include <stdio.h>  int main()  {      unsigned int a = 29; /\* 29 = 0001 1101 \*/      unsigned int b = 48; /\* 48 = 0011 0000 \*/      int c = 0;      c = a & b; /\* 32 = 0001 0000 \*/      printf("%d & %d = %d\n", a, b, c );      c = a | b; /\* 61 = 0011 1101 \*/      printf("%d | %d = %d\n", a, b, c );      c = a ^ b; /\* 45 = 0010 1101 \*/      printf("%d ^ %d = %d\n", a, b, c );      c = ~a; /\* -30 = 1110 0010 \*/      printf("~%d = %d\n", a, c );      c = a << 2; /\* 116 = 0111 0100 \*/      printf("%d << 2 = %d\n", a, c );      c = a >> 2; /\* 7 = 0000 0111 \*/      printf("%d >> 2 = %d\n", a, c );      return 0;  } |

**Kết quả:**

|  |
| --- |
| **29 & 48 = 16**  **29 | 48 = 61**  **29 ^ 48 = 45**  **~29 = -30**  **29 << 2 = 116**  **29 >> 2 = 7** |

Các phép toán bitwise với kiểu dữ liệu có dấu nên được sử dụng cẩn thận vì bit dấu của kiểu dữ liệu đó có ý nghĩa riêng. Có các hạn chế đặc biệt áp dụng cho các toán tử dịch chuyển (shift operators):

* Dịch trái một bit 1 vào bit dấu của kiểu dữ liệu có dấu được coi là sai và dẫn đến hành vi không xác định. Điều này có nghĩa rằng kết quả của phép dịch như vậy không được xác định rõ ràng và có thể khác nhau tùy thuộc vào trình biên dịch và nền tảng.
* Dịch phải một giá trị âm (với bit dấu được đặt là 1) được xác định bởi từng trình biên dịch và do đó không có tính chất di động. Điều này có nghĩa rằng hành vi của phép dịch như vậy có thể khác nhau giữa các phiên bản khác nhau của ngôn ngữ C và không được đảm bảo di động trên các nền tảng hoặc trình biên dịch khác nhau.
* Nếu giá trị của toán hạng phải của toán tử dịch chuyển là số âm hoặc lớn hơn hoặc bằng độ rộng của toán hạng trái được gán, hành vi là không xác định. Điều này có nghĩa là kết quả của phép dịch như vậy không được chỉ định và có thể cho ra kết quả không mong đợi hoặc không nhất quán.

Để đảm bảo tính di động và tránh hành vi không xác định, thường khuyến nghị sử dụng kiểu dữ liệu không dấu khi thực hiện các phép toán bitwise liên quan đến dịch chuyển.

**Masking:**

Masking (hoặc còn gọi là áp dụng mặt nạ) đề cập đến quá trình trích xuất các bit mong muốn từ một biến (hoặc biến đổi các bit mong muốn trong biến) bằng cách sử dụng các phép toán logic bitwise. Toán hạng (một hằng số hoặc biến) được sử dụng để thực hiện việc áp dụng mặt nạ được gọi là mặt nạ (mask).

Trong quá trình masking, mặt nạ là một giá trị có các bit được thiết lập một cách đặc biệt để thực hiện việc trích xuất hoặc biến đổi các bit trong biến gốc. Khi thực hiện phép toán bitwise AND (&) giữa biến gốc và mặt nạ, chỉ có các bit tương ứng mà mặt nạ đặt thành 1 và các bit còn lại trong biến gốc sẽ bị xóa đi (thiết lập thành 0). Điều này cho phép trích xuất hoặc biến đổi các bit cần thiết trong biến gốc và bỏ qua các bit không mong muốn.

Mặt nạ được sử dụng theo nhiều cách khác nhau

* Để quyết định mẫu bit của một biến số nguyên.
* Để sao chép một phần của mẫu bit đã cho sang một biến mới, trong khi phần còn lại của biến mới được điền bằng 0 (sử dụng bitwise AND)
* Để sao chép một phần của mẫu bit đã cho sang một biến mới, trong khi phần còn lại của biến mới được lấp đầy bằng 1 (sử dụng OR theo bit).
* Để sao chép một phần của mẫu bit đã cho sang một biến mới, trong khi phần còn lại của mẫu bit ban đầu được đảo ngược trong biến mới (sử dụng OR loại trừ bit).

Hàm sau sử dụng mặt nạ để hiển thị mẫu bit của một biến:

|  |
| --- |
| #include <limits.h>  void bit\_pattern(int *u*)  {      int i, x, word;      unsigned mask = 1;      word = CHAR\_BIT \* sizeof(int);      mask = mask << (word - 1); /\* shift 1 to the leftmost position \*/      for(i = 1; i <= word; i++)      {          x = (*u* & mask) ? 1 : 0; /\* identify the bit \*/          printf("%d", x); /\* print bit value \*/          mask >>= 1; /\* shift mask to the right by 1 bit \*/      }  } |

## PHẦN 4.4 HOẠT ĐỘNG NGẮN MẠCH CỦA TOÁN TỬ LOGIC

## PHẦN 4.5 TOÁN TỬ DẤU PHẨY

Toán tử dấu phẩy (,) cho phép bạn đánh giá nhiều biểu thức ở bất kỳ nơi nào cho phép một biểu thức duy nhất. Toán tử dấu phẩy đánh giá toán hạng bên trái, sau đó là toán hạng bên phải, rồi trả về kết quả của toán hạng bên phải.

|  |
| --- |
| int x = 42, y = 42;  printf("%i\n", (x \*= 2, y)); /\* Outputs "42". \*/ |

Toán tử dấu phẩy giới thiệu một điểm thứ tự giữa các toán hạng của nó.

**Lưu ý** rằng dấu phẩy được sử dụng trong các hàm gọi các đối số riêng biệt đó **KHÔNG** phải là toán tử dấu phẩy, thay vào đó, nó được gọi là dấu phân cách khác với toán tử dấu phẩy. Do đó, nó không có các thuộc tính của toán tử dấu phẩy.

Lệnh gọi printf() ở trên chứa cả toán tử dấu phẩy và dấu phân cách.

|  |
| --- |
| printf("%i\n", (x \*= 2, y)); /\* Outputs "42". \*/  /\*           ^        ^ Đây là toán tử dấu phẩy \*/  /\*           Đây là dấu phân cách \*/ |

Toán tử dấu phẩy thường được sử dụng trong phần khởi tạo cũng như trong phần cập nhật của vòng lặp for. Ví dụ:

|  |
| --- |
| for(k = 1; k < 10; printf("*\%*d\\n", k), k += 2); /\*outputs the odd numbers below 9/\*  /\* outputs sum to first 9 natural numbers \*/  for(sumk = 1, k = 1; k < 10; k++, sumk += k)      printf("*\%*5d*\%*5d\\n", k, sumk); |

## PHẦN 4.6 TOÁN TỬ SỐ HỌC

### 4.6.1 Số học cơ bản

Trả về một giá trị là kết quả của việc áp dụng toán hạng bên trái cho toán hạng bên phải, sử dụng phép toán liên quan. Các quy tắc giao hoán toán học thông thường được áp dụng (nghĩa là phép cộng và phép nhân có tính chất giao hoán, phép trừ, phép chia và mô đun thì không).

### 4.6.2 Toán tử cộng

Toán tử cộng (+) được sử dụng để cộng hai toán hạng lại với nhau. Ví dụ:

|  |
| --- |
| #include<stdio.h>  int main(){      int a = 5;      int b = 7;      int c = a + b;      printf("%d + %d = %d",a,b,c);      return 0;  } |

**Kết quả**

|  |
| --- |
| **5 + 7 = 12** |

### 4.6.3 Toán tử trừ

Toán tử trừ được sử dụng để trừ toán hạng thứ 2 cho toán hạng thứ nhất. Ví dụ

|  |
| --- |
| #include<stdio.h>  int main(){      int a = 5;      int b = 7;      int c = a - b;      printf("%d - %d = %d",a,b,c);      return 0;  } |

**Kết quả**

|  |
| --- |
| **5 - 7 = -2** |

### 4.6.4 Toán tử nhân

Toán tử nhân (\*) được sử dụng để nhân cả hai toán hạng. Ví dụ:

|  |
| --- |
| #include<stdio.h>  int main(){      int a = 5;      int b = 7;      int c = a \* b;      printf("%d \* %d = %d",a,b,c);      return 0;  } |

**Kết quả**

|  |
| --- |
| **5 \* 7 = 35** |

**Đừng nhầm lẫn với con trỏ (\*)**

### 4.6.5 Toán tử chia

Toán tử chia (/) chia toán hạng thứ nhất cho toán hạng thứ hai. Nếu cả hai toán hạng của phép chia là số nguyên, nó sẽ trả về một giá trị nguyên và loại bỏ phần còn lại (sử dụng toán tử modulo % để tính toán và lấy phần còn lại).

Nếu một trong các toán hạng là một giá trị dấu chấm động, thì kết quả là một phép tính gần đúng của phân số.

**Ví dụ:**

|  |
| --- |
| #include<stdio.h>  int main(){      int a = 19 / 2 ;      int b = 18 / 2 ;      int c = 255 / 2;      int d = 44 / 4 ;      double e = 19 / 2.0 ;      double f = 18.0 / 2 ;      double g = 255 / 2.0;      double h = 45.0 / 4 ;      printf("19 / 2 = %d\n", a);      printf("18 / 2 = %d\n", b);      printf("255 / 2 = %d\n", c);      printf("44 / 4 = %d\n", d);      printf("19 / 2.0 = %g\n", e);      printf("18.0 / 2 = %g\n", f);      printf("255 / 2.0 = %g\n", g);      printf("45.0 / 4 = %g\n", h);  } |

**Kết quả**

|  |
| --- |
| **19 / 2 = 9**  **18 / 2 = 9**  **255 / 2 = 127**  **44 / 4 = 11**  **19 / 2.0 = 9.5**  **18.0 / 2 = 9**  **255 / 2.0 = 127.5**  **45.0 / 4 = 11.25** |

**Chú ý: Ép kiểu**

|  |
| --- |
| #include<stdio.h>  int main(){      int a = 9;      int b = 2;      float c;      //c = (float) a / b;      //c = 9.0 / 2;      c = 9 / 2.0;      printf("%d / %d = %f",a,b,c);  } |

**Kết quả**

|  |
| --- |
| **9 / 2 = 4.500000** |

### 4.6.6 Toán tử modulo (Phép chia lấy dư)

Toán tử modulo (%) chỉ nhận toán hạng số nguyên và được sử dụng để tính phần còn lại sau khi toán hạng thứ nhất được chia cho toán hạng thứ hai. Toán tử mô đun (còn được gọi là toán tử chia lấy phần dư) là một toán tử trả về phần còn lại sau khi thực hiện phép chia số nguyên.

Ví dụ:

|  |
| --- |
| #include <stdio.h>  int main (void) {      int a = 25 % 2; /\* a holds value 1 \*/      int b = 24 % 2; /\* b holds value 0 \*/      int c = 155 % 5; /\* c holds value 0 \*/      int d = 49 % 25; /\* d holds value 24 \*/      printf("25 *%* 2 = %d\n", a); /\* Will output "25 % 2 = 1" \*/      printf("24 *%* 2 = %d\n", b); /\* Will output "24 % 2 = 0" \*/      printf("155 *%* 5 = %d\n", c); /\* Will output "155 % 5 = 0" \*/      printf("49 *%* 25 = %d\n", d); /\* Will output "49 % 25 = 24" \*/      return 0;  } |

### 4.6.7 Toán tử tăng giảm

Các toán tử tăng (a++) và giảm (a--) khác nhau ở chỗ chúng thay đổi giá trị của biến mà bạn áp dụng chúng mà không có toán tử gán. Bạn có thể sử dụng toán tử **tăng** và **giảm** trước hoặc sau biến. Vị trí của toán tử thay đổi thời gian tăng/giảm giá trị thành trước hoặc sau khi gán nó cho biến.

Ví dụ:

|  |
| --- |
| #include <stdio.h>  int main(void)  {      int a = 1;      int b = 4;      int c = 1;      int d = 4;      a++;      printf("a = %d\n",a); /\* Sẽ in ra "a = 2" \*/      b--;      printf("b = %d\n",b); /\* Sẽ in ra "b = 3" \*/      if (++c > 1) { /\* c được tăng thêm 1 trước khi được so sánh trong điều kiện \*/          printf("This will print\n"); /\* Dòng này sẽ được in ra \*/      } else {          printf("This will never print\n"); /\* Dòng này sẽ không được in ra \*/      }      if (d-- < 4) { /\* d được giảm sau khi được so sánh \*/          printf("This will never print\n"); /\* Dòng này sẽ không được in ra \*/      } else {          printf("This will print\n"); /\* Dòng này sẽ được in ra \*/      }  } |

Kết quả

|  |
| --- |
| a = 2  b = 3  This will print  This will print |

Như ví dụ cho c và d cho thấy, cả hai toán tử đều có hai dạng, là ký hiệu tiền tố và ký hiệu hậu tố. Cả hai đều có tác dụng giống nhau trong việc tăng (++) hoặc giảm (--) biến, nhưng khác nhau về giá trị mà chúng trả về: các phép toán tiền tố thực hiện thao tác trước rồi trả về giá trị, trong khi các phép toán hậu tố trước tiên xác định giá trị được trả lại, và sau đó thực hiện thao tác.

Do hành vi có khả năng phản trực giác này, việc sử dụng các toán tử tăng/giảm bên trong các biểu thức đang gây tranh cãi.

## PHẦN 4.7 TOÁN TỬ TRUY CẬP

Các toán tử truy cập thành viên (dấu chấm . và mũi tên ->) được sử dụng để truy cập một thành viên của cấu trúc(struct).

### 4.7.1 Thuộc tính của đối tượng

Được đánh giá thành giá trị, chỉ định đối tượng là thành viên của đối tượng được truy cập.

|  |
| --- |
| struct *MyStruct*  {      int x;      int y;  };  struct *MyStruct* myObject;  myObject.x = 42;  myObject.y = 123;  printf(".x = %i, .y = %i\n", myObject.x, myObject.y); /\* Outputs ".x = 42, .y = 123". \*/ |

### 4.7.2 Thành viên của đối tượng được trỏ tới

Toán tử mũi tên (->) được sử dụng để tiện lợi hóa quá trình giải tham chiếu và truy cập thành viên của một cấu trúc. Thay vì viết (\*x).y để truy cập thành viên y của đối tượng được giải tham chiếu bởi con trỏ x, chúng ta có thể sử dụng x->y. Điều này giúp làm rõ hơn cách thao tác và làm cho mã nguồn dễ đọc hơn, đặc biệt là khi có các con trỏ cấu trúc lồng nhau.

|  |
| --- |
| #include <stdio.h>  int main(){  struct *MyStruct*  {      int x;      int y;  };  struct *MyStruct* myObject;  struct *MyStruct* \*p = &myObject;  p->x = 42;  p->y = 123;  printf(".x = %i, .y = %i\n", p->x, p->y); /\* Kết quả ".x = 42, .y = 123". \*/  printf(".x = %i, .y = %i\n", myObject.x, myObject.y); /\* Kết quả cũng sẽ là ".x = 42, .y = 123". \*/  } |

### 4.7.3 Lấy địa chỉ

Toán tử & (unary address of operator) trong ngôn ngữ C được sử dụng để lấy địa chỉ của một biến. Khi áp dụng toán tử & cho một biến, nó trả về một con trỏ chứa địa chỉ của biến đó.

**Ví dụ:**

|  |
| --- |
| #include <stdio.h>  int main(){  int x = 3;  int \*p = &x;  printf("%p = %p\n", &x, (p)); /\* Xuất ra "A = A",Với A là một giá trị do triển khai cụ thể quy định ở đây là giá trị địa chỉ ngẫu nhiên.\*/  } |

**Kết quả**

|  |
| --- |
| 0061FF18 = 0061FF18 |

### 4.7.4 Dereference

The unary \* operator dereferences a pointer. It evaluates into the lvalue resulting from dereferencing the pointer that results from evaluating the given expression.

|  |
| --- |
| int x = 42;  int \*p = &x;  printf("x = %d, \*p = %d\n", x, \**p*); /\* Xuất ra "x = 42, \*p = 42". \*/  \*p = 123;  printf("x = %d, \*p = %d\n", x, \**p*); /\* Xuất ra "x = 123, \*p = 123". \*/ |

### 4.7.5 Chỉ số mảng

Trong C, cú pháp indexing (chỉ số mảng) thực tế là một cách viết tắt cho việc thực hiện phép cộng con trỏ và sau đó giải tham chiếu. Biểu thức dạng **a[i]** tương đương với **\*(a + i)**, trong đó a là một con trỏ và i là chỉ số hoặc độ lệch.

Trong C, mảng sẽ tự động chuyển thành con trỏ tới phần tử đầu tiên của nó trong hầu hết các ngữ cảnh. Khi bạn sử dụng cú pháp chỉ số **a[i],** trình biên dịch sẽ coi a như một con trỏ tới phần tử đầu tiên của mảng và thực hiện phép cộng con trỏ với độ lệch **i**. Sau đó, con trỏ kết quả được giải tham chiếu sử dụng toán tử **\*** để truy cập vào giá trị tại vị trí đó trong bộ nhớ.

|  |
| --- |
| int arr[] = { 1, 2, 3, 4, 5 };  printf("arr[2] = %i\n", arr[2]); /\* Xuất ra "arr[2] = 3". \*/ |

### 4.7.6 Khả năng hoán đổi của chỉ số mảng

Thêm một con trỏ vào một số nguyên là một phép toán giao hoán (nghĩa là thứ tự của các toán hạng không làm thay đổi kết quả) vì vậy con trỏ + số nguyên == số nguyên + con trỏ.

Hệ quả của điều này là arr[3] và 3[arr] là tương đương.

|  |
| --- |
| printf("3[arr] = %i\n", 3[arr]); /\* Xuất ra "3[arr] = 4". \*/ |

## PHẦN 4.8 TOÁN TỬ SIZEOF

### 4.8.1 With a type as operand

Đánh giá kết quả là kích thước trong byte, của kiểu size\_t, của đối tượng thuộc kiểu đã cho. Yêu cầu có dấu ngoặc đơn xung quanh kiểu dữ liệu.

|  |
| --- |
| printf("%zu\n", sizeof(int)); /\* Hợp lệ, in ra kích thước của một đối tượng int,                                 \*có thể thay đổi theo nền tảng. \*/  printf("%zu\n", sizeof *int*); /\* Không hợp lệ, các kiểu dữ liệu truyền vào cần được                                \*bao bọc bởi dấu ngoặc đơn! \*/ |

### 4.8.2 With an expression as operand

Đánh giá kết quả là kích thước trong byte, kiểu size\_t, của các đối tượng có cùng kiểu dữ liệu với biểu thức được cung cấp. Biểu thức chính nó không được đánh giá. Dấu ngoặc đơn không bắt buộc; tuy nhiên, vì biểu thức được cung cấp phải là toán tử một ngôi, nên luôn tốt nhất là sử dụng chúng.

|  |
| --- |
| char ch = 'a';  printf("%zu\n", sizeof(ch)); /\* Hợp lệ, sẽ in ra kích thước của một đối tượng char,                                \* luôn là 1 cho mọi nền tảng. \*/  printf("%zu\n", sizeof *ch*); /\* Hợp lệ, sẽ in ra kích thước của một đối tượng char,                               \*luôn là 1 cho mọi nền tảng. \*/ |

## PHẦN 4.9 TOÁN TỬ ÉP KIỂU

Thực hiện một chuyển đổi rõ ràng sang kiểu dữ liệu đã chỉ định từ giá trị thu được sau khi đánh giá biểu thức cung cấp.

|  |
| --- |
| int x = 3;  int y = 4;  printf("%f\n", (double)x / y); /\* In ra "0.750000". \*/ |

Ở đây, giá trị của x được chuyển đổi thành kiểu double, phép chia cũng chuyển đổi giá trị của y thành kiểu double và kết quả của phép chia, một giá trị double, được truyền vào hàm printf để in ra màn hình.

## PHẦN 4.10 TOÁN TỬ GỌI HÀM

Toán tử gọi hàm trong ngôn ngữ lập trình yêu cầu toán hạng đầu tiên phải là một con trỏ tới hàm (function pointer) hoặc một biểu thức chỉ định hàm (function designator). Đối tượng này xác định hàm mà ta muốn gọi. Các toán hạng còn lại, nếu có, được gọi là đối số của cuộc gọi hàm. Toán tử này đánh giá thành giá trị trả về từ việc gọi hàm thích hợp với các đối số tương ứng.

|  |
| --- |
| int myFunction(int *x*, int *y*)  {      return *x* \* 2 + *y*;  }  int (\*fn)(int, int) = &myFunction;  int x = 42;  int y = 123;  printf("(\*fn)(%i, %i) = %i\n", x, y, (fn)(x, y)); /\* In ra "fn(42, 123) = 207". \*/  printf("fn(%i, %i) = %i\n", x, y, fn(*x*, y)); /\* Một dạng khác: bạn không cần phải giải tham chiếu một cách rõ ràng \*/ |

## PHẦN 4.11 TOÁN TỬ TĂNG/ TOÁN TỬ GIẢM

Các toán tử tăng và giảm tồn tại ở dạng tiền tố và hậu tố.

|  |
| --- |
| int a = 1;  int b = 1;  int tmp = 0;  tmp = ++a; /\* tăng a lên một đơn vị và trả về giá trị mới; a == 2, tmp == 2 \*/  tmp = a++; /\* tăng a lên một đơn vị nhưng trả về giá trị cũ; a == 3, tmp == 2 \*/  tmp = --b; /\* giảm b đi một đơn vị và trả về giá trị mới; b == 0, tmp == 0 \*/  tmp = b--; /\* giảm b đi một đơn vị nhưng trả về giá trị cũ; b == -1, tmp == 0 \*/ |

Hiểu đơn giản

Toán tử ++: Tăng giá trị lên 1 đơn vị.

**• x++:** thực hiện lệnh trước rồi mới tăng x lên 1 đơn vị.

**• ++x:** tăng x lên 1 đơn vị rồi mới thực hiện lệnh.

Toán tử --: Giảm giá trị đi 1 đơn vị.

**• x--:** thực hiện lệnh trước rồi mới giảm x đi 1 đơn vị.

**• --x:** giảm x đi 1 đơn vị rồi mới thực hiện lệnh.

Lưu ý rằng các phép toán số học không tạo ra điểm chuỗi (sequence points), do đó, một số biểu thức sử dụng toán tử tăng (++) hoặc giảm (--) có thể dẫn đến hành vi không xác định.

## PHẦN 4.12 TOÁN TỬ GÁN

Gán giá trị của toán hạng bên phải cho vị trí lưu trữ được đặt tên bởi toán hạng bên trái và trả về giá trị.

|  |
| --- |
| int x = 5; /\* Biến x chứa giá trị 5. Trả về 5. /  char y = 'c'; /\* Biến y chứa giá trị 99. Trả về 99                 \*(vì ký tự 'c' được đại diện bởi 99 trong bảng ASCII). \*/  float z = 1.5; /\* Biến z chứa giá trị 1.5. Trả về 1.5. \*/  char const\* s = "foo"; /\* Biến s chứa địa chỉ của ký tự đầu tiên trong chuỗi 'foo'. \*/ |

Một số phép tính số học có toán tử gán phức hợp.

|  |
| --- |
| a += b /\* tương đương với: a = a + b \*/  a -= b /\* tương đương với: a = a - b \*/  a = b /\* tương đương với: a = a \* b \*/  a /= b /\* tương đương với: a = a / b \*/  a %= b /\*tương đương với: a = a % b \*/  a &= b /\* tương đương với: a = a & b \*/  a |= b /\* tương đương với: a = a | b \*/  a ^= b /\* tương đương với: a = a ^ b \*/  a <<= b /\* tương đương với: a = a << b \*/  a >>= b /\*tương đương với: a = a >> b \*/ |

Một tính năng quan trọng của các toán tử gán kết hợp này là biểu thức ở phía bên trái (a) chỉ được đánh giá một lần. Ví dụ, nếu p là một con trỏ:

|  |
| --- |
| \*p += 5; |

Trong trường hợp này, giá trị của biểu thức \*p sẽ chỉ được tính toán một lần và sau đó, giá trị mới được gán vào vị trí mà con trỏ p trỏ tới. Điều này giúp tránh các lỗi không xác định hoặc không mong muốn khi giá trị của biểu thức thay đổi trong quá trình đánh giá.

|  |
| --- |
| \*p = \*p + 5 |

Cũng cần lưu ý rằng kết quả của một phép gán chẳng hạn như a = b là cái được gọi là giá trị. Do đó, phép gán thực sự có một giá trị mà sau đó có thể được gán cho một biến khác. Điều này cho phép xâu chuỗi các phép gán để đặt nhiều biến trong một câu lệnh.

Giá trị này có thể được sử dụng trong các biểu thức kiểm soát của câu lệnh if (hoặc vòng lặp hoặc câu lệnh chuyển đổi) để bảo vệ một số mã trên kết quả của một biểu thức hoặc lời gọi hàm khác. Ví dụ:

|  |
| --- |
| char \*buffer;  if ((buffer = malloc(1024)) != NULL)  {   /\* thực hiện một số công việc sử dụng buffer \*/      free(buffer);  }  else  {   /\* báo cáo lỗi khi không cấp phát được bộ nhớ \*/  } |

Do đó, cần phải cẩn thận để tránh mắc lỗi đánh máy phổ biến có thể dẫn đến các lỗi bí ẩn.

|  |
| --- |
| int a = 2;  /\* ... \*/  if (a = 1)  /\* Xóa tất cả các tập tin trên ổ cứng của tôi \*/ |

Điều này sẽ dẫn đến kết quả tai hại, vì a = 1 sẽ luôn đánh giá bằng 1 và do đó, biểu thức kiểm soát của câu lệnh if sẽ luôn đúng (đọc thêm về cạm bẫy phổ biến này tại đây). Tác giả gần như chắc chắn có ý định sử dụng toán tử đẳng thức (==) như hình dưới đây:

|  |
| --- |
| int a = 2;  /\* ... \*/  if (a == 1)  /\* Xóa tất cả các tập tin trên ổ cứng của tôi \*/ |

**Tính kết hợp của toán tử**

|  |
| --- |
| int a, b = 1, c = 2;  a = b = c; |

Điều này gán c cho b, trả về b, được gán cho a. Điều này xảy ra bởi vì tất cả các toán tử gán có tính kết hợp phải, điều đó có nghĩa là phép toán ngoài cùng bên phải trong biểu thức được ước tính trước và tiến hành từ phải sang trái.

## PHẦN 4.13 TOÁN TỬ LOGIC

### 4.13.1 Logic AND

Toán tử logic AND (&&) trong C thực hiện phép AND logic giữa hai toán hạng và trả về giá trị 1 nếu cả hai toán hạng đều khác 0. (Có nghĩa là đúng khi cả 2 đều đúng) Toán tử AND logic có kiểu int.

|  |
| --- |
| 0 && 0 /\* Trả về 0. \*/  0 && 1 /\* Trả về 0. \*/  2 && 0 /\* Trả về 0. \*/  2 && 3 /\* Trả về 1. \*/ |

### 4.13.2 Logic OR

Toán tử logic OR (||) trong C thực hiện phép OR logic giữa hai toán hạng và trả về giá trị 1 nếu bất kỳ một trong hai toán hạng là khác không. Toán tử OR logic có kiểu int.

|  |
| --- |
| 0 || 0 /\* Trả về 0. \*/  0 || 1 /\* Trả về 1. \*/  2 || 0 /\* Trả về 1. \*/  2 || 3 /\* Trả về 1. \*/ |

### 4.13.3 Logic NOT

Toán tử logic NOT (!) trong C thực hiện phép phủ định logic của một biểu thức và trả về giá trị 1 nếu biểu thức là sai (bằng 0), và trả về giá trị 0 nếu biểu thức là đúng (khác 0). Toán tử NOT logic có kiểu int.

Các tính chất quan trọng chung cho cả phép toán logic AND (&&) và phép toán logic OR (||) là như sau, và rất quan trọng để hiểu cách chúng hoạt động trong việc đánh giá ngắn gọn (short-circuit evaluation):

Toán hạng bên trái (LHS) được đánh giá hoàn toàn trước khi toán hạng bên phải (RHS) được đánh giá:

* Trong biểu thức A && B, A được đánh giá trước, và chỉ khi A là đúng (khác không) thì B mới được đánh giá. Nếu A là sai (bằng không), B sẽ không được đánh giá vì kết quả tổng thể của phép toán AND đã xác định là sai.
* Trong biểu thức A || B, A được đánh giá trước, và chỉ khi A là sai (bằng không) thì B mới được đánh giá. Nếu A là đúng (khác không), B sẽ không được đánh giá vì kết quả tổng thể của phép toán OR đã xác định là đúng.

Có một điểm chuỗi (sequence point) giữa việc đánh giá toán hạng bên trái và toán hạng bên phải:

* Điểm chuỗi là điểm trong việc thực thi chương trình nơi tất cả các hiệu ứng phụ của các đánh giá trước đó đã được thực hiện. Điểm chuỗi đảm bảo rằng các đánh giá xảy ra theo một thứ tự được xác định, tránh các vấn đề liên quan đến hành vi không xác định (undefined behavior).

Toán hạng bên phải không được đánh giá nếu kết quả của toán hạng bên trái xác định kết quả tổng thể:

* Đánh giá ngắn gọn (short-circuit evaluation) tránh việc thực hiện các tính toán không cần thiết. Ví dụ, trong A && B, nếu A là sai, kết quả của biểu thức tổng thể đã biết là sai bất kể giá trị của B. Do đó, không cần phải đánh giá B.
* Tương tự, trong A || B, nếu A là đúng, kết quả của biểu thức tổng thể đã biết là đúng bất kể giá trị của B. Do đó, không cần phải đánh giá B.

Các tính chất này là rất quan trọng để viết mã hiệu quả và an toàn, đặc biệt khi sử dụng các biểu thức có hiệu ứng phụ tiềm ẩn hoặc các đánh giá phức tạp. Nó cho phép các lập trình viên tối ưu hóa mã của họ và tránh công việc không cần thiết khi đánh giá biểu thức logic.

## PHẦN 4.14 TOÁN HỌC TRÊN CON TRỎ

### 4.14.1 Cộng con trỏ

### 4.14.2 Trừ con trỏ

## PHẦN 4.15 \_Alignof