Trong ngôn ngữ lập trình C, để hiển thị ra màn hình 1 dòng chữ bạn dùng hàm printf(). Để hiểu rõ hơn bạn hãy chạy thử chương trình sau:

#include<stdio.h>

int main() {

printf("Hello Codelearn");

return 0;

}

Kết quả khi chạy chương trình:

Hello Codelearn

Nếu bạn là người mới bắt đầu thì trong đoạn code trên có rất nhiều lệnh bạn chưa biết như #include<stdio.h>, int main(){ và return 0; nhưng bạn chưa cần quan tâm tới các lệnh này vội, hãy coi như những lệnh này là bắt buộc đối với 1 chương trình C.

Bây giờ bạn hãy thay printf("Hello Codelearn"); trong đọan code trên thành printf("You can code"); giống như chương trình sau và chạy thử:

#include<stdio.h>

int main() {

printf("You can code");

return 0;

}

Kết quả khi chạy chương trình:

You can code

Để làm được bài này bạn cần biết cách sử dụng hàm printf()

Trước hết bạn hãy chạy thử chương trình sau:

#include<stdio.h>

int main() {

printf("23 + 56");

return 0;

}

Kết quả khi chạy chương trình:

23 + 56

Có thể thấy đây không phải là kết quả mong muốn, để hiển thị được kết quả của phép tính 23 + 56 bạn cần sử dụng hàm printf() với định dạng %d giống chương trình sau:

#include<stdio.h>

int main() {

printf("%d", 23 + 56);

return 0;

}

Kết quả sau khi chạy:

79

Trong hàm printf() chỗ nào có định dạng %d thì chỗ đó sẽ được thay thế bởi 1 số nguyên. Để hiểu rõ hơn bạn hãy chạy thử chương trình sau:

#include<stdio.h>

int main() {

printf("%d + %d = %d", 8, 7, 8 + 7);

return 0;

}

Kết quả khi chạy chương trình:

8 + 7 = 15

Trong chương trình trên có 3 chỗ có định dạng %d và tương ứng với các định dạng đó là 8, 7 và  8 + 7.

Comment hay còn gọi là chú thích được dùng để giải thích các dòng code, giúp cho người đọc hay chính người viết code sau này hiểu được source code dễ dàng hơn. Các ký tự và đoạn code trong comment sẽ không được chương trình thực thi.

Trong C có 2 loại comment là comment trên một dòng và comment trên nhiều dòng.

Comment trên một dòng sẽ được bắt đầu với //. Ví dụ đơn giản về comment một dòng:

#include<stdio.h>

int main() {

// Dùng hàm printf để hiển thị Hello World! ra màn hình

printf("Hello World!");

return 0;

}

Kết quả khi chạy chương trình:

Hello World!

Bạn cũng có thể đặt comment ngay sau câu lệnh:

#include<stdio.h>

int main() {

printf("Hello World!"); // Dùng hàm printf để hiển thị Hello World! ra màn hình

return 0;

}

Comment trên nhiều dòng trong C sẽ bắt đầu bằng dấu /\* và kết thúc bằng dấu \*/. Ví dụ:

/\*

Chương trình c hiển thị ra màn hình dòng chữ Hello World!

sử dụng hàm printf

\*/

#include<stdio.h>

int main() {

printf("Hello World!");

return 0;

}

Kết quả khi chạy chương trình:

Hello World!

Để hiển thị ra 1324 + 7553 =  bạn sử dụng câu lệnh:

printf("1324 + 7553 = ");

Để hiển thị ra tổng của 1324 và 7553 bạn sử dụng câu lệnh:

printf("%d", 1324 + 7553);

Bạn có thể kết hợp hai câu lệnh này lại như ví dụ sau:

#include<stdio.h>

int main() {

printf("1324 + 7553 = %d", 1324 + 7553);

return 0;

}

Sẽ hiển thị lên màn hình

1324 + 7553 = 8877

Đầu tiên bạn hãy chạy thử chương trình sau:

#include<stdio.h>

int main() {

printf("Name: Codelearn.");

printf("Date of birth: 2019.");

return 0;

}

Chương trình trên sẽ hiển thị lên màn hình:

Name: Codelearn.Date of birth: 2019.

Có thể thấy đoạn code trên không hiển thị được thông tin trên 2 dòng. Để hiển thị được trên nhiều dòng bạn cần dùng tới ký tự xuống dòng '\n' trong hàm printf. Ví dụ chương trình sau:

#include<stdio.h>

int main() {

printf("Line1\n");

printf("Line2");

return 0;

}

Sẽ hiển thị lên màn hình

Line1

Line2

Ngoài ra bạn cũng có thể viết

#include<stdio.h>

int main() {

printf("Line1\nLine2");

return 0;

}

Khái niệm biến trong lập trình cũng giống khái niệm biến trong toán học, biến được dùng để đại diện cho một giá trị nào đó. Cú pháp để khai báo biến kiểu số nguyên trong ngôn ngữ lập trình C như sau:

int tên\_biến;

Ví dụ về cách tạo và sử dụng biến kiểu số nguyên:

#include<stdio.h>

int main() {

// Khai báo biến a kiểu số nguyên

int a;

// Gán giá trị cho a = 5

a = 5;

// Hiển thị ra màn hình giá trị của biến a

printf("a = %d", a);

return 0;

}

Kết quả khi chạy chương trình:

a = 5

Bạn còn có thể khai báo và gán giá trị cho biến trong 1 câu lệnh giống chương trình sau:

#include<stdio.h>

int main() {

// Khai báo biến a kiểu số nguyên và gán giá trị cho a = 438

int a = 438;

// Khai báo biến b kiểu số nguyên và gán giá trị cho b = 238

int b = 238;

// Hiển thị ra màn hình hiệu của a và b

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

return 0;

}

Kết quả khi chạy chương trình:

a - b = 200

Ngoài ra khi đặt tên cho biến bạn cần tuân theo 1 số nguyên tắc sau:

* Tên biến có thể có chữ cái, chữ số và dấu gạch dưới nhưng ký tự đầu tiên của tên biến bắt buộc phải là dấu gạch dưới hoặc chữ cái.
* Tên biến không được có khoảng trắng.
* Tên biến không được trùng với các từ khóa như int, float, static, struct...

Một số tên biến hợp lệ:

int a;

int \_a;

int a10\_;

Một số tên biến không hợp lệ:

int 10a;

int a 10;

int int;

Để khai báo và gán giá trị cho biến kiểu số thực bạn bạn dùng từ khóa double:

// Khai báo biến a kiểu số thực và gán giá trị cho a = 10.5

double a = 10.5;

Để hiển thị một số thực ra màn hình bạn dùng hàm printf() với định dạng %f. Ví dụ chương trình tạo và hiển thị biến kiểu số thực ra màn hình:

#include<stdio.h>

int main() {

// Khai báo biến a kiểu số thực và gán giá trị cho a = 1.5

double a = 1.5;

printf("a = %f", a);

return 0;

}

Kết quả khi chạy chương trình:

a = 1.500000

Ngoài double ra thì trong ngôn ngữ lập trình C còn kiểu dữ liệu dùng để lưu trữ số thực nữa là float nhưng trong hầu hết mọi trường hợp bạn nên sử dụng kiểu double.

**Lưu ý:** bạn không thể dùng biến kiểu số nguyên để lưu trữ số thực, để hiểu rõ hơn hãy xem chương trình sau:

#include<stdio.h>

int main() {

int a = 1.5;

printf("a = %f\n", a);

printf("a = %d", a);

return 0;

}

Kết quả khi chạy chương trình:

a = 0.000000

a = 1

Có thể thấy khi bạn dùng biến kiểu số nguyên để lưu trữ số thực thì kết quả sẽ không được chính xác.

Kiểu ký tự trong C là kiểu dữ liệu chỉ lưu trữ được 1 ký tự trong bảng mã [ASCII](https://vi.wikipedia.org/wiki/ASCII), ký tự này có thể là một chữ cái (a, b, c, ... x, y, z), một chữ số  (0, 1, 2,..., 9), một phép toán (+, -, \*, /) hay một ký tự bất kỳ khác (!, &, ...).

Biến kiểu ký tự được khai báo bằng từ khóa char (char là viết tắt của **character**), để hiển thị biến kiểu ký tự bạn dùng định dạng %c. Ví dụ chương trình tạo và hiển thị biến kiểu ký tự:

#include<stdio.h>

int main() {

// Khai báo biến c kiểu ký tự và gán trị cho c = 'a'

char c = 'a';

// Hiển thị biến c ra màn hình

printf("%c", c);

return 0;

}

Chương trình này sẽ hiển thị lên màn hình:

a

Mục đích của bài này là giúp bạn hiểu hơn về kiểu ký tự nên hãy tìm cách điền code vào dấu ... để màn hình hiển thị ra ký tự 'd'.

Kiểu ký tự bản chất là một kiểu số nguyên, bạn có thể hiện giá trị nguyên của một ký tự bằng định dạng %d giống như chương trình sau:

#include<stdio.h>

int main() {

printf("a = %d\n", 'a');

printf("b = %d\n", 'b');

printf("c = %d\n", 'c');

printf("d = %d\n", 'd');

printf("e = %d\n", 'e');

printf("...\n");

printf("x = %d\n", 'x');

printf("y = %d\n", 'y');

printf("z = %d\n", 'z');

return 0;

}

Kết quả khi chạy chương trình:

a = 97

b = 98

c = 99

d = 100

e = 101

...

x = 120

y = 121

z = 122

Giá trị số của các chữ cái từ 'a' đến 'z' tăng dần, mỗi giá trị liên tiếp cách nhau 1 đơn vị.

Do kiểu ký tự là một kiểu số nguyên nên bạn có thể áp dụng các phép toán +, -, \*, /. Từ đoạn code bên trên có thể thấy để hiển thị ký tự 'd' bạn có thể lấy ký tự 'a' + 3, để hiển thị ký tự 'e' bạn có thể lấy ký tự 'd' + 1 như chương trình sau:

#include<stdio.h>

int main() {

char d = 'a' + 3;

char e = 'd' + 1;

printf("%c\n", d);

printf("%c", e);

return 0;

}

Màn hình sẽ hiển thị ra:

d

e

Trong ngôn ngữ lập trình C có các kiểu dữ liệu lưu trữ số nguyên như: short, int, long, long long...

Sự khác biệt giữa các kiểu dữ liệu này là miền giá trị:

Miền giá trị của kiểu short là từ -32768 tới 32767.

Miền giá trị của kiểu int là từ -2147483648 tới 2147483647.

Miền giá trị của kiểu long là từ -2147483647 tới 2147483647 (giống với miền dữ liệu của kiểu int).

Miền giá trị của kiểu long long là từ -9223372036854775808 tới 9223372036854775807.

Có thể thấy miền giá trị của kiểu long long là lớn nhất, do đó bạn có thể dùng kiểu dữ liệu này để thay cho các kiểu dữ liệu khác nhưng bù lại dùng kiểu dữ liệu này sẽ tốn bộ nhớ hơn (kiểu dữ liệu có miền giá trị càng lớn sẽ càng tốn bộ nhớ). Do đó bạn cần sử dụng các kiểu dữ liệu một cách hợp lý, ví dụ biến để lưu trữ số học sinh trong một lớp học nên là kiểu short vì thường một lớp chỉ có vài chục học sinh.

**Lưu ý:** Miền giá trị của các kiểu dữ liệu còn phụ thuộc vào yếu tố khác nhưng thường sẽ giống như trên.

Quay lại bài này, bạn không thể dùng kiểu int để lưu giá trị cho biến a và b do kết quả của phép nhân 2 số kiểu int sẽ là 1 số kiểu int mà miền giá trị của kiểu int là không đủ để lưu trữ kết quả của phép nhân 384847522 \* 988347273. Ví dụ chương trình:

#include<stdio.h>

int main() {

int a = 384847522;

int b = 988347273;

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

return 0;

}

Sẽ cho ra kết quả:

-628954446

Có thể thấy kết quả này là sai hoàn toàn, bạn cần dùng kiểu dữ liệu long long cho biến a và biến b. Để hiển thị biến kiểu long long ra màn hình bạn cần dùng hàm printf() với định dạng %lld.

Để nhập dữ liệu cho biến kiểu số nguyên bạn có thể dùng hàm scanf() với định dạng %d. Ví dụ chương trình sau:

#include<stdio.h>

int main() {

// Khai báo biến a kiểu số nguyên

int a;

// Nhập dữ liệu cho biến a từ bàn phím

scanf("%d", &a);

// Hiển thị

printf("a = %d", a);

return 0;

}

Sau khi chạy nếu bạn nhập

77

Và bấm phím **Enter** thì chương trình sẽ hiển thị lên màn hình:

a = 77

Đọc tới đây bạn đã có thể quay lại phần bài tập và làm thử.

Nếu bạn vẫn chưa làm được bài này thì có thể xem hướng dẫn ở bên dưới.

**Lưu ý:** Một số lỗi thường gặp khi sử dụng nhập dữ liệu từ bàn phím với scanf() mà bạn dễ gặp phải:

* Thiếu dấu nháy đôi: scanf("%d, &a"); -> scanf("%d", &a);
* Thiếu &tên\_biến: scanf("%d", a); -> scanf("%d", &a);
* Để lấy phần dư của một phép chia bạn có thể sử dụng toán tử %. Ví dụ chương trình:
* #include<stdio.h>
* int main() {
* printf("%d", 5 % 3);
* return 0;
* }
* Sẽ hiển thị lên màn hình:
* 2
* Do 2 là phần dư của phép chia 5/3.
* Để nhập dữ liệu cho biến kiểu số thực (double) bạn dùng hàm scanf với định dạng %lf. Ví dụ chương trình nhập và tính tích 2 số thực:
* #include<stdio.h>
* int main() {
* double a, b;
* // Dùng định dạng %lf để nhập dữ liệu cho biến kiểu số thực
* scanf("%lf%lf", &a, &b);
* printf("a \* b = %f", a \* b);
* return 0;
* }
* Để làm được bài này bận cần biết cách sử dụng mệnh đề if hoặc mệnh đề if-else
* **Mệnh đề if:**
* Mệnh đề if được dùng để kiểm tra một biểu thức điều kiện có đúng hay không, nếu đúng thì các câu lệnh bên trong mệnh đề if sẽ được thực thi. Ví dụ để kiểm tra xem n có chia hết cho 2 không bạn có thể làm như sau:
* if (n % 2 == 0) {
* // Nếu n chia hết cho 2 thì hiển thị n is an even number
* printf("n is an even number");
* }
* if (n % 2 != 0) {
* // Nếu n không chia hết cho 2 thì hiển thị n is an odd number
* printf("n is an odd number");
* }
* Trong ngôn ngữ lập trình C, == là toán tử kiểm tra xem 2 giá trị có bằng nhau không, != là toán tử kiểm tra xem 2 giá trị có khác nhau không. Như ở ví dụ trên if (n % 2 == 0) là câu lệnh kiểm tra xem n chia dư cho 2 có bằng 0 không, if (n % 2 != 0) là câu lệnh kiểm tra xem n chia dư cho 2 có khác 0 không.
* Lưu ý: Một số bạn hay nhầm giữa toán tử = và toán tử ==, toán tử = là toán tử gán giá trị chứ không phải toán tử so sánh nên nếu bạn dùng toán tử = để so sánh thì chương trình sẽ cho ra kết quả sai.
* **Mệnh đề if-else:**
* Tương tự như mệnh đề if nhưng nếu biểu thức điều kiện trong if sai thì đoạn code bên trong else sẽ được thực thi:
* if (n % 2 == 0) {
* // Nếu n chia hết cho 2 thì hiển thị n is an even number
* printf("n is an even number");
* } else {
* // Ngược lại (n không chia hết cho 2) thì hiển thị n is an odd number
* printf("n is an odd number");
* }
* Để kiểm tra một số có bằng 0 không bạn có thể sử dụng mệnh đề if và toán tử ==. Ví dụ:
* if (n == 0) {
* // Nếu n = 0 thì hiển thị ra màn hình n is equal to 0
* printf("n is equal to 0");
* }
* Để kiểm tra một số là nguyên âm hay nguyên dương bạn có thể sử dụng mệnh đề if với các toán tử > và <. Ví dụ chương trình sau:
* #include<stdio.h>
* int main() {
* int n = 5;
* if (n > 0) {
* // Nếu n lớn hơn 0 thì hiển thị n is a positive number
* printf("n is a positive number");
* }
* if (n < 0) {
* // Nếu n nhỏ hơn 0 thì hiển thị n is a negative number
* printf("n is a negative number");
* }
* return 0;
* }
* Kết quả khi chạy chương trình:
* n is a positive number
* Để kiểm tra một số có lớn hơn hoặc bằng số khác không bạn có thể sử dụng toán tử >=. Ví dụ:
* if (a >= b) {
* /\*
* Nếu a lớn hơn hoặc bằng b thì hiển thị ra màn hình
* a is greater than or equal to b
* \*/
* printf("a is greater than or equal to b");
* }
* Để kiểm tra xem một số có khác 0 không rất đơn giản, bạn có thể sử dụng mệnh đề if và toán tử !=. Ví dụ để kiểm tra xem biến a có khác 0 không bạn có thể làm như sau:
* if (a != 0) {
* // Nếu a khác 0 thì hiển thị a is not equal to zero
* printf("a is not equal to 0");
* }
* Vậy để kiểm tra xem cả a và b có khác 0 không bạn cần kết hợp được 2 câu lệnh if (a != 0) và if (b != 0). Trong C bạn có thể kết hợp 2 câu lệnh này bằng toán tử &&. Ví dụ:
* if (a != 0 && b != 0) {
* // Nếu cả a và b đều khác 0 thì hiển thị a is not equal to 0 and b is not equal to 0
* printf("a is not equal to 0 and b is not equal to 0");
* }
* Bài này bạn có thể dùng mệnh đề if và toán tử && như bài trước hoặc cũng có thể dùng mệnh đề if và toán tử ||. Ngược với toán tử &&, toán tử || dùng để kiểm tra xem một trong các biểu thức có đúng không, nếu có thì đoạn code bên trong mệnh đề if sẽ được thực thi (với toán tử && thì tất cả các biểu thức phải đúng thì code bên trong mệnh đề if mới được thực thi, hiểu đơn giản hơn thì || là toán tử **hoặc** còn && là toán tử **và**). Ví dụ để kiểm tra biến score có nhỏ hơn 0 hoặc lớn hơn 10 bạn có thể làm như sau:
* if (score < 0 || score > 10) {
* // Nếu score nhỏ hơn 0 hoặc score lớn hơn 10 thì đoạn code trong khối lệnh này sẽ được thực thi
* }
* Đọc tới đây bạn đã biết cách dùng toán tử ||, hãy sử dụng cả 2 toán tử là && và || để làm bài này.
* Nếu bạn chưa làm được thì có thể xem hướng dẫn ở bên dưới.
* Để nhập dữ liệu cho biến kiểu ký tự bạn có thể dùng hàm scanf với định dạng %c.
* Bài này bạn hoàn toàn có thể làm bằng mệnh đề **if** như đã học nhưng cách tốt hơn để làm bài này là sử dụng lệnh **switch case.**
* Cấu trúc của lệnh **switch case**:
* switch (x) {
* case giá\_trị\_1:
* // Nếu x có giá trị bằng giá\_trị\_1 thì đoạn code trong khối lệnh này sẽ được thực thi
* break;
* case giá\_trị\_2:
* // Nếu x có giá trị bằng giá\_trị\_2 thì đoạn code trong khối lệnh này sẽ được thực thi
* break;
* case giá\_trị\_3:
* // Nếu x có giá trị bằng giá\_trị\_3 thì đoạn code trong khối lệnh này sẽ được thực thi
* break;
* ...
* case giá\_trị\_n:
* // Nếu x có giá trị bằng giá\_trị\_n thì đoạn code trong khối lệnh này sẽ được thực thi
* break;
* default:
* /\*
* Nếu x không bằng giá trị nào trong các giá trị trên
* thì code trong khối lệnh này sẽ được thực thi
* \*/
* break;
* }
* Ví dụ về chương trình nhập vào một số từ 1 tới 7 và hiển thị ngày trong tuần tương ứng với số đó:
* #include<stdio.h>
* int main() {
* int n;
* scanf("%d", &n);
* switch (n) {
* case 1:
* printf("Monday");
* break;
* case 2:
* printf("Tuesday");
* break;
* case 3:
* printf("Wednesday");
* break;
* case 4:
* printf("Thursday");
* break;
* case 5:
* printf("Friday");
* break;
* case 6:
* printf("Saturday");
* break;
* case 7:
* printf("Sunday");
* break;
* default:
* printf("Invalid input! Please enter week number between 1-7");
* }
* return 0;
* }
* Lưu ý: Biến hoặc biểu thức bên trong lệnh **switch** phải là một số nguyên và giá trị trong các lệnh **case** phải là một hằng số.
* Bài này sẽ được giải bằng vòng lặp for. Cách đơn giản nhất để hiểu được vòng lặp trong C là xem nhiều các ví dụ.
* Ví dụ về chương trình sử dụng vòng lặp for để in ra các số từ 1 tới 5:
* #include<stdio.h>
* int main() {
* // i++ tương đương với i = i + 1 hoặc i += 1
* for (int i = 1; i <= 5; i++) {
* printf("%d ", i);
* }
* return 0;
* }
* Kết quả khi chạy chương trình:
* 1 2 3 4 5
* Ví dụ về chương trình sử dụng vòng lặp for để in ra các số từ 10 về 1:
* #include<stdio.h>
* int main() {
* // i-- tương đương với i = i - 1 hoặc i -= 1
* for (int i = 10; i > 0; i--) {
* printf("%d ", i);
* }
* return 0;
* }
* Kết quả khi chạy chương trình:
* 10 9 8 7 6 5 4 3 2 1
* Ví dụ về chương trình sử dụng vòng lặp for để in ra các số chẵn từ 2 tới 100:
* #include<stdio.h>
* int main() {
* for (int i = 2; i <= 100; i += 2) {
* printf("%d ", i);
* }
* return 0;
* }
* Kết quả khi chạy chương trình:
* 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 94 96 98 100
* Tới đây chắc bạn cũng đã hiểu sơ qua về cách sử dụng vòng lặp for. Vòng lặp được hiểu đơn giản là lặp lại 1 đoạn code. Ví dụ vòng for:
* for (int i = 1; i <= 5; i++) {
* printf("%d ", i);
* }
* Sẽ hoạt động theo thứ tự như sau:
* Bước 1. Tạo ra biến i kiểu nguyên và gán giá trị cho i = 1
* Bước 2. Nếu i <= 5 thì thì thực hiện đoạn code bên trong dấu { và } (đoạn code này chính là printf("%d ", i);). Ngược lại nếu i > 5 thì dừng không chạy tiếp bước 3.
* Bước 3. Tăng i lên 1 đơn vị và quay lại bước 2 (i++ tương đương với i = i + 1 hoặc i += 1).
* Bài này bạn có thể làm bằng vòng lặp for như các bài trước nhưng tôi sẽ hướng dẫn bạn sử dụng vòng lặp while để làm.
* Cấu trúc vòng lặp while:
* while (biểu\_thức){
* // Nếu biểu thức đúng thì thực hiện đoạn code trong khối lệnh này
* }
* Có thể thấy cách sử dụng vòng lặp while rất đơn giản, khi biểu\_thức còn đúng thì thực hiện đoạn code bên trong vòng while.
* Hãy xem một vài ví dụ về vòng lặp while để hiểu rõ hơn:
* #include<stdio.h>
* int main() {
* int n = 5;
* while (n != 0) {
* printf("%d ", n);
* n--;
* }
* return 0;
* }
* Chương trình này sẽ hiển thị ra màn hình:
* 5 4 3 2 1
* Đoạn code trên có thể hiểu đơn giản là: Khi n còn khác 0 thì hiển thị ra màn hình giá trị của n và giảm n đi 1.
* Ví dụ khác về hiển thị các số chẵn từ 1 tới 100 sử dụng vòng lặp while:
* #include<stdio.h>
* int main() {
* int n = 1;
* while (n <= 100) {
* if (n % 2 == 0) {
* printf("%d ", n);
* }
* n++;
* }
* return 0;
* }
* Bài này bạn sẽ học được cách dừng vòng lặp với câu lệnh break.
* Câu lệnh break là câu lệnh dùng để dừng một vòng lặp, cách sử dụng câu lệnh break rất đơn giản, ví dụ chương trình sau:
* #include<stdio.h>
* int main() {
* for (int i = 1; i <= 20; i++) {
* if (i == 11) {
* break;
* }
* printf("%d ", i);
* }
* return 0;
* }
* Sẽ hiển thị ra
* 1 2 3 4 5 6 7 8 9 10
* thay vì
* 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
* do khi i = 11 thì câu lệnh break sẽ được thực thi và tất cả code trong vòng lặp sẽ không được thực hiện tiếp.
* Trong chương trình trên nếu bạn đặt câu lệnh printf("%d ", i); lên trước câu lệnh break; giống như sau:
* #include<stdio.h>
* int main() {
* for (int i = 1; i <= 20; i++) {
* printf("%d ", i);
* if (i == 11) {
* break;
* }
* }
* return 0;
* }
* Thì màn hình sẽ hiển thị ra:
* 1 2 3 4 5 6 7 8 9 10 11
* Do khi i = 11 thì câu lệnh printf("%d ", i); sẽ được thực thi trước câu lệnh break;
* Bạn cũng có thể dùng câu lệnh break với vòng lặp while giống chương trình sau:
* #include<stdio.h>
* int main() {
* int i = 1;
* while (i <= 20) {
* if (i == 11) {
* break;
* }
* printf("%d ", i);
* i++;
* }
* return 0;
* }
* Kết quả khi chạy chương trình:
* 1 2 3 4 5 6 7 8 9 10
* ài này bạn sẽ học được cách sử dụng câu lệnh continue.
* Khi gặp câu lệnh continue trong vòng lặp, các đoạn code bên dưới câu lênh này sẽ không được thực thi.
* Ví dụ chương trình hiển thị các số chẵn từ 1 tới 100 sử dụng câu lệnh continue:
* #include<stdio.h>
* int main() {
* for (int i = 1; i <= 100; i++) {
* if (i % 2 != 0) {
* continue;
* }
* printf("%d ", i);
* }
* return 0;
* }
* Trong đoạn code trên khi câu lệnh continue được thực thi thì câu lệnh printf("%d ", i); sẽ không được thực thi (do câu lệnh này nằm dưới câu lệnh continue).
* Bạn cũng có thể dùng câu lệnh continue với vòng lặp while.
* Ở các bài trước bạn đã được học về vòng lặp for và while, bài này bạn sẽ học được cách sử dụng vòng lặp do-while. Cấu trúc vòng lặp do-while:
* do {
* // Khối lệnh
* } while (biểu\_thức);
* Đầu tiên khối lệnh bên trong vòng lặp do-while sẽ được thực thi trước. Sau đó vòng lặp do-while sẽ kiểm tra biểu thức điều kiện, nếu biểu thức điều kiện đúng thì chương trình sẽ tiếp tục thực thi khối lệnh trong do-while, ngược lại nếu biểu thức điều kiện sai thì dừng vòng lặp.
* Vòng lặp do-while khác vòng lặp while ở chỗ vòng lặp while sẽ kiểm tra điều kiện trước, còn vòng lặp do-while sẽ thực hiện khối lệnh trước nên khối lệnh trong do-while sẽ được thực thi ít nhất 1 lần.
* Ví dụ về sự khác nhau giữa 2 vòng lặp:
* #include<stdio.h>
* int main() {
* int i = 1;
* do {
* printf("do-while");
* } while (i < 1);
* while (i < 1) {
* printf("while");
* }
* return 0;
* }
* Chương trình trên sẽ hiển thị ra:
* do-while
* **Lưu ý:** Vòng lặp do-while kết thúc bằng dấu chấm phẩy.

Bài này có thể làm bằng cách thông thường là tạo ra 10 biến như sau:

#include<stdio.h>

int main() {

int a0, a1, a2, a3, a4, a5, a6, a7, a8, a9;

scanf("%d%d%d%d%d%d%d%d%d%d", &a0, &a1, &a2, &a3, &a4, &a5, &a6, &a7, &a8, &a9);

printf("%d", a0 + a1 + a2 + a3 + a4 + a5 + a6 + a7 + a8 + a9);

return 0;

}

Có thể thấy cách làm này không được hay, bạn phải tạo ra rất nhiều biến. Nếu chương trình yêu cầu nhập 100 số nguyên thì bạn cũng phải tạo ra 100 biến. Rất may là vấn đề này có thể giải quyết bằng mảng.

Mảng được hiểu đơn giản là một tập hợp các biến. Ví dụ bài này nếu làm bằng mảng thì sẽ như sau:

#include<stdio.h>

int main() {

// Khai báo mảng a kiểu số nguyên với 10 phần tử

int a[10];

// Khai báo biến lưu trữ tổng các phần tử trong mảng

int sum = 0;

// Dùng vòng for để nhập dữ liệu cho các phần tử trong mảng

for (int i = 0; i < 10; i++) {

scanf("%d", &a[i]);

}

// Dùng vòng for để tính tổng các phần tử trong mảng

for (int i = 0; i < 10; i++) {

sum += a[i];

}

// In ra tổng các phần tử trong mảng

printf("%d", sum);

return 0;

}

Có thể thấy thay vì tạo ra 10 biến thì bạn chỉ cần tạo ra 1 mảng với 10 phần tử. Thay vì nhập dữ liệu và tính tổng trên từng biến thì bạn nhập dữ liệu và tính tổng trên từng phần tử của mảng.

Các phần tử của mảng được truy xuất bằng toán tử []. Ví dụ:

* a[0] tương đương với phần tử thứ nhất.
* a[1] tương đương với phần tử thứ hai.
* ...

Nhìn chung cách làm bài này với mảng cũng giống việc tạo ra nhiều biến, chỉ khác ở chỗ thay vì phải viết a0, a1, a2, ..., a9 thì bạn có thể dùng vòng lặp for và viết là a[i].

Ngoài ra bạn còn có thể khai báo và khởi tạo giá trị cho mảng bằng 1 số cách dưới đây:

// Khai báo mảng a 5 phần tử với giá trị từ 1 tới 5

int a[5] = {1, 2, 3, 4, 5};

// Khai báo mảng a 5 phần tử với giá trị từ 1 tới 5

int a[] = { 1, 2, 3, 4, 5 };

// Khai báo mảng a 20 phần tử và gán giá trị cho 5 phần tử đầu tiên

int a[20] = { 1, 2, 3, 4, 5 };

Mảng 2 chiều hay còn được gọi là ma trận là một mảng các mảng 1 chiều. Mảng 2 chiều giống như một cái bảng, mảng 1 chiều là các dòng trong bảng và các phần tử của mảng là các ô.

Cú pháp để khai báo mảng 2 chiều với 3 hàng và 5 cột:

int arr[3][5];

Ví dụ chương trình nhập vào một mảng 2 chiều và in mảng 2 chiều này ra màn hình:

#include<stdio.h>

int main() {

int arr[100][100];

int n, m;

scanf("%d%d", &n, &m);

for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) {

scanf("%d", &arr[i][j]);

}

}

for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) {

printf("arr[%d][%d] = %d\n", i, j, arr[i][j]);

}

}

return 0;

}

Nếu bạn nhập

3 4

3 4 8 2

5 6 4 9

2 3 8 1

Thì chương trình sẽ hiển thị ra:

arr[0][0] = 3

arr[0][1] = 4

arr[0][2] = 8

arr[0][3] = 2

arr[1][0] = 5

arr[1][1] = 6

arr[1][2] = 4

arr[1][3] = 9

arr[2][0] = 2

arr[2][1] = 3

arr[2][2] = 8

arr[2][3] = 1

Xâu bản chất là 1 mảng các ký tự được lưu trữ trên các ô nhớ liên tiếp và kết thúc bằng ký tự '\0', bạn có thể khởi tạo giá trị cho xâu giống như cho mảng:

* char str[] = "abcd";
* char str[50] = "abcd";
* char str[] = {'a', 'b', 'c', 'd', '\0'};
* char str[5] = {'a', 'b', 'c', 'd', '\0'};

Để nhập dữ liệu và hiển thị một xâu ra màn hình bạn dùng định dạng %s giống như chương trình sau:

#include<stdio.h>

int main() {

// Khai báo xâu str, xâu str lưu trữ được 99 phần tử

char str[100];

// Nhập dữ liệu vào xâu str

scanf("%s", str);

// Hiển thị xâu str ra màn hình

printf("%s", str);

return 0;

}

Đoạn code trên tạo ra xâu str lưu trữ được 99 ký tự (không phải là 100 do ký tự cuối cùng của một xâu luôn là '\0'), nhập dữ liệu cho xâu str và hiển thị xâu này ra màn hình.

**Lưu ý:** Khi nhập dữ liệu cho xâu thì không sử dụng toán tử & trong hàm scanf().

Xâu bản chất là 1 mảng các ký tự nên bạn có thể lấy ra phần tử thứ k trong xâu s bằng s[k - 1];

Như đã học ở bài đầu tiên về xâu, xâu là một mảng các ký tự kết thúc bằng ký tự '\0' nên bạn có thể lấy ra độ dài xâu bằng các duyệt xâu và tìm phần tử có giá trị bằng '\0'. Ví dụ chương trình hiển thị ra độ dài của một xâu:

#include<stdio.h>

int main() {

int len = 0;

char str[] = "Codelearn.io";

while (str[len] != '\0') {

len++;

}

printf("%d", len);

return 0;

}

Kết quả khi chạy chương trình:

12

Ngoài ra bạn còn có thể sử dụng hàm strlen() (strlen là viết tắt của **string length**) trong thư viện string.h để lấy ra độ dài của 1 xâu giống chương trình sau:

#include<stdio.h>

#include<string.h>

int main() {

char str[] = "Codelearn.io";

printf("%d", strlen(str));

return 0;

}

Kết quả khi chạy chương trình:

12

Khi làm thực tế bạn nên dùng hàm strlen() vì nó ngắn hơn.

Về bản chất ký tự cũng chỉ là một số nguyên, việc bạn chuyển 1 ký tự từ in thường sang in hoa thực chất là chuyển đổi từ 1 số nguyên sang 1 số nguyên khác. Như đã được học, để hiển thị giá trị nguyên (mã **ASCII**) của một ký tự bạn có thể làm như sau:

#include<stdio.h>

int main() {

printf("a = %d\n", 'a');

printf("b = %d\n", 'b');

printf("c = %d\n", 'c');

printf("d = %d\n\n", 'd');

printf("A = %d\n", 'A');

printf("B = %d\n", 'B');

printf("C = %d\n", 'C');

printf("D = %d\n", 'D');

return 0;

}

Chương trình này sẽ hiển thị lên màn hình:

a = 97

b = 98

c = 99

d = 100

A = 65

B = 66

C = 67

D = 68

Có thể quan sát thấy giá trị của các ký tự in thường luôn bằng giá trị của các ký tự in hoa cộng với 32. Từ đó bạn có thể chuyển 1 ký tự từ in thường sang in hoa bằng cách trừ đi 32. Ví dụ chương trình:

#include<stdio.h>

int main() {

printf("%c", 'd' - 32);

return 0;

}

Sẽ hiển thị ra màn hình

D

Từ đó bạn có thể giải bài này bằng cách duyệt qua các ký tự và nếu ký tự đó là in thường thì trừ đi 32.

Bạn không thể sử dụng toán tử == để kiểm tra 2 xâu có giống nhau hay không vì nếu áp dụng toán tử này lên 2 xâu thì đó là phép so sánh địa chỉ của phần tử đầu tiên của 2 xâu. Cụ thể hơn chương trình sau:

#include<stdio.h>

#include<string.h>

int main() {

char str1[] = "coding";

char str2[] = "coding";

if (str1 == str2) {

printf("str1 is equal to str2");

} else {

printf("str1 is not equal to str2");

}

return 0;

}

Sẽ hiển thị lên màn hình:

str1 is not equal to str2

Để kiểm tra xem 2 xâu có giống nhau hay không bạn có thể dùng hàm strcmp() trong thư viện string.h. Hàm này sẽ trả về 0 nếu 2 xâu giống nhau, ngược lại hàm này sẽ trả về một số khác 0 nếu 2 xâu khác nhau. Ví dụ chương trình sau:

#include<stdio.h>

#include<string.h>

int main() {

char str1[] = "coding";

char str2[] = "coding";

char str3[] = "codelearn.io";

printf("%d %d", strcmp(str1, str2), strcmp(str1, str3));

return 0;

}

Kết quả khi chạy chương trình:

0 1

Hàm được hiểu đơn giản là 1 đoạn code, khi bạn gọi hàm thì thực ra là bạn đang gọi tới đoạn code này (bạn cũng có thể hiểu hàm là viết tắt của 1 đoạn code). Cách khai báo và sử dụng hàm rất đơn giản, trước hết bạn hãy xem một vài ví dụ về hàm:

Ví dụ về hàm hiển thị ra các số từ 1 tới 10:

#include<stdio.h>

// Khai báo hàm show

void show() {

// Hiển thị ra màn hình các số từ 1 đến 10

for (int i = 1; i <= 10; i++) {

printf("%d ", i);

}

}

int main() {

// Gọi hàm show

show();

return 0;

}

Kết quả khi chạy chương trình:

1 2 3 4 5 6 7 8 9 10

Ví dụ về hàm hiển thị ra bảng chữ cái tiếng anh:

#include<stdio.h>

// Khai báo hàm show

void show() {

// Hiển thị bảng chữ cái tiếng anh ra màn hình

for (char c = 'A'; c <= 'Z'; c++) {

printf("%c", c);

}

}

int main() {

// Gọi hàm show

show();

return 0;

}

Kết quả khi chạy chương trình:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Như đã nói ở trên, việc bạn gọi hàm thực ra là bạn đang gọi tới đoạn code bên trong hàm nên bạn có thể hiển thị ra 2 bảng chữ cái bằng cách gọi hàm 2 lần:

#include<stdio.h>

// Khai báo hàm show

void show() {

// Hiển thị bảng chữ cái tiếng anh ra màn hình

for (char c = 'A'; c <= 'Z'; c++) {

printf("%c", c);

}

}

int main() {

// Gọi hàm show

show();

show();

return 0;

}

Kết quả khi chạy chương trình:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Tới đây chắc bạn cũng đã hiểu cách tạo và sử dụng hàm, cú pháp để khai báo hàm sẽ trông giống như sau:

void tên\_hàm() {

/\*

Khối lệnh bên trong thân hàm

Khối lệnh này sẽ được thực thi khi bạn gọi hàm

\*/

}

Để sử dụng hàm bạn chỉ đơn giản là gọi tới tên\_hàm();

Bài này sẽ giúp bạn học được cách sử dụng hàm có đầu vào. Trước hết hãy xem ví dụ đơn giản về hàm có đầu vào:

#include<stdio.h>

// Khai báo hàm show với đầu vào là biến name kiểu string

void show(char name[100]) {

// Hiển thị biến name ra màn hình

printf("Hello %s", name);

}

int main() {

char name[] = "Codelearn";

show(name);

return 0;

}

Kết quả khi chạy chương trình:

Hello Codelearn

Hàm show() trong ví dụ trên là hàm có đầu vào là 1 mảng các ký tự.

Khi gọi một hàm có đầu vào bạn luôn cần cung cấp giá trị cho các đầu vào của hàm. Để hiểu rõ hơn bạn hãy xem ví dụ khác về hàm hiển thị số lớn nhất trong 3 số đầu vào:

#include<stdio.h>

// Khai báo hàm show với đầu vào là 3 số nguyên a, b, c

void show(int a, int b, int c) {

int maxValue = a;

if (maxValue < b) {

maxValue = b;

}

if (maxValue < c) {

maxValue = c;

}

// Hiển số lớn nhất trong 3 số đầu vào

printf("%d\n", maxValue);

}

int main() {

// Gọi hàm show với giá trị của các đầu vào là a = 4, b = 6, c = 3

show(4, 6, 3);

// Gọi hàm show với giá trị của các đầu vào là a = 2, b = 1, c = 5

show(2, 1, 5);

// Gọi hàm show với giá trị của các đầu vào là a = 3, b = 3, c = 3

show(3, 3, 3);

return 0;

}

Kết quả khi chạy chương trình:

6

5

3

Có thể thấy khi gọi hàm show(); bạn cần truyền giá trị cho 3 đầu vào.

Trước tiên bạn hãy xem code khi làm bài này mà không sử dụng hàm:

#include<stdio.h>

#include<string.h>

int main() {

char s1[100], s2[100], s3[100];

scanf("%s %s %s", s1, s2, s3);

int len1 = strlen(s1);

for (int i = 0; i < len1; i++) {

if (s1[i] >= 'A' && s1[i] <= 'Z') {

s1[i] += 32;

}

}

int len2 = strlen(s2);

for (int i = 0; i < len2; i++) {

if (s2[i] >= 'A' && s2[i] <= 'Z') {

s2[i] += 32;

}

}

int len3 = strlen(s3);

for (int i = 0; i < len3; i++) {

if (s3[i] >= 'A' && s3[i] <= 'Z') {

s3[i] += 32;

}

}

printf("%s\n%s\n%s", s1, s2, s3);

return 0;

}

Chắc bạn cũng đã nhận thấy vấn đề với cách làm này: code ở cả 3 vòng for bị trùng lặp nhau. Rõ ràng bạn cần tìm cách để tái sử dụng lại đoạn code chuyển 1 xâu từ in hoa sang in thường thay vì cứ phải lặp lại đoạn code này mỗi khi sử dụng.

Qua các bài đã học về hàm, bạn đã biết rằng hàm được dùng để đại diện cho 1 đoạn code, do đó bạn có thể đặt đoạn code bị trùng lặp vào trong hàm và mỗi khi sử dụng bạn chỉ cần gọi hàm.

Để thấy được lợi ích của hàm bạn hãy xem code khi làm bài này bằng hàm:

#include<stdio.h>

#include<string.h>

void show(char s[100]) {

int len = strlen(s);

// Chuyển xâu s từ in hoa sang in thường

for (int i = 0; i < len; i++) {

if (s[i] >= 'A' && s[i] <= 'Z') {

s[i] += 32;

}

}

// Hiển thị xâu s ra màn hình

printf("%s\n", s);

}

int main() {

char s1[100], s2[100], s3[100];

scanf("%s %s %s", s1, s2, s3);

show(s1);

show(s2);

show(s3);

return 0;

}

Có thể thấy khi sử dụng hàm code trở nên ngắn và dễ hiểu hơn.

Ở các bài trước bạn đã biết cách tạo và sử dụng hàm không có giá trị trả về. Hàm không có giá trị trả về được khai báo bằng từ khóa void, các hàm dạng này được sử dụng để thực thi các đoạn code trong thân hàm và không cần lấy ra kết quả sau khi thực thi xong.

Hàm có giá trị trả về khác hàm không có giá trị trả về ở chỗ là các hàm dạng này sau khi thực thi các đoạn code thì sẽ trả về một kết quả. Ví dụ về hàm trả về tổng các số từ 1 tới n:

#include<stdio.h>

// Khai báo hàm sum với kiểu trả về là int và đầu vào là số n

int sum(int n) {

// Biến lưu trữ tổng các số từ 1 tới n

int add = 0;

// Tính tổng các số từ 1 tới n và lưu vào biến add

for (int i = 1; i <= n; i++) {

add += i;

}

// Trả về add (gán kết quả của hàm sum cho add)

return add;

}

int main() {

// Hiển thị giá trị của hàm sum với các đầu vào khác nhau ra màn hình

printf("%d %d %d", sum(3), sum(4), sum(5));

return 0;

}

Kết quả khi chạy chương trình:

6 10 15

Trong ví dụ trên hàm sum() được khai báo với từ khóa int nên sẽ có kiểu trả về là một số nguyên (kết quả của hàm này là một số nguyên).

Để trả về một kết quả cho hàm bạn dùng câu lệnh return, như ở ví dụ trên thì hàm sum() có gọi tới câu lệnh return add; nên kết quả của hàm sẽ chính là biến add.

Do sum() là hàm có giá trị trả về là một số nguyên nên bạn có thể sử dụng hàm này như sử dụng một số nguyên (bạn có thể gán int a = sum(3); hay printf("%d", sum(3)), ...).

Nếu bạn vẫn chưa quen với hàm có giá trị trả về thì có thể xem tiếp một ví dụ khác đơn giản hơn về hàm trả về tổng 2 số:

#include<stdio.h>

int sum(int a, int b) {

return a + b;

}

int main() {

printf("%d %d %d", sum(2, 3), sum(1, 4), sum(3, 7));

return 0;

}

Kết quả khi chạy chương trình:

5 5 10

Lưu ý: Hàm sẽ kết thúc và trả về giá trị ngay khi gặp câu lệnh return. Xem ví dụ sau để hiểu hơn:

#include<stdio.h>

using namespace std;

int sum(int a, int b) {

return 1;

return a + b;

}

int main() {

printf("%d %d %d", sum(2, 3), sum(1, 4), sum(3, 7));

return 0;

}

Kết quả khi chạy chương trình:

1

1

1

Hàm đệ quy là hàm mà gọi tới chính nó, ví dụ một hàm đệ quy sẽ trông giống như sau:

void recurse() {

...

recurse();

...

}

Do tính chất tự gọi lại chính nó nên để tránh việc hàm đệ quy chạy không bao giờ dừng bạn luôn cần có điểm dừng (điểm dừng được hiểu đơn giản là tới 1 lúc nào đó, hàm đệ quy sẽ không gọi lại chính nó nữa).

Mô tả hàm đệ quy tính 5!:

factorial(5)

= 5 \* factorial(4)

= 5 \* 4 \* factorial(3)

= 5 \* 4 \* 3 \* factorial(2)

= 5 \* 4 \* 3 \* 2 \* factorial(1)

= 5 \* 4 \* 3 \* 2 \* 1

= 120

Điểm dừng ở ví dụ trên chính là khi đầu vào của hàm factorial bằng 1 thì hàm factorial sẽ trả về 1 thay vì gọi tiếp tới chính nó.

Cụ thể hơn xem ví dụ về hàm đệ quy tính tổng các số từ 1 tới n:

#include<stdio.h>

int sumOfArray(int n) {

if (n == 0) return 0;

return n + sumOfArray(n - 1);

}

int main() {

printf("%d", sumOfArray(10));

return 0;

}

Điểm dừng ở đây chính là khi n = 0 thì hàm sumOfArray trả về 0 thay vì gọi tiếp tới chính nó.