**Hàm**

**1.Định Nghĩa Hàm**

Bạn đã từng sử dụng các hàm như sqrt, pow, abs... của các thư viện có sẵn trong C, tuy nhiên để giải quyết bài toán của bạn thì bạn cần phải tự xây dựng các hàm để giải quyết các chức năng nhỏ cho bài toán của mình.

Hàm (function) là một các khối lệnh có nhiệm vụ thực hiện một chức năng nào đó.

Ở các bài học trước bạn đều viết mã nguồn C bên trong hàm main, chương trình sẽ trở nên khó quản lý khi số lượng dòng code trở nên lớn, cùng điểm qua các lợi ích của việc sử dụng hàm



**Cú Pháp :**

data\_type function\_name(type1 parameter1, type2 parameter2...){

//code

}

**Các thành phần của hàm :**

* data\_type : Kiểu trả về của hàm, có thể là các kiểu dữ liệu như int, long long, float, char, double, hoặc void (tương ứng với kiểu trả về là rỗng)
* function\_name : Tên của hàm, cần tuân theo quy tắc như đặt tên biến
* parameter : Tham số của hàm, đây được coi như đầu vào của hàm. Bạn có thể xây dựng bao nhiêu tham số tùy ý và lựa chọn kiểu dữ liệu cho từng tham số.
* code : Các câu lệnh bên trong của hàm

Ví dụ 1 : Hàm có kiểu trả về là int, có 3 tham số là a, b, c

int tong(int a, int b, int c){

int sum = a + b + c;

return sum;

}

Ví dụ 2 : Hàm có kiểu trả về là void, có 3 tham số là a kiểu int, b kiểu long long, c kiểu double

void display(int a, long long b, double c){

printf("%d %lld %.2lf\n", a, b, c);

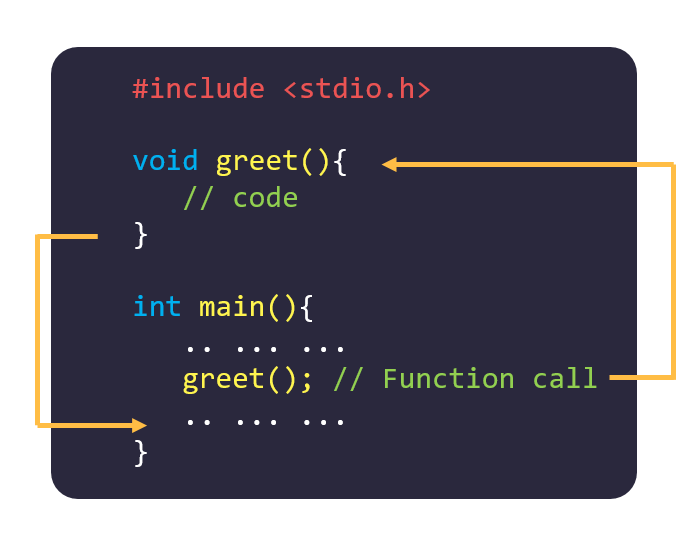
}

**2. Lời Gọi Hàm**

Sau khi xây dựng hàm xong để hàm có thể thực thi bạn cần gọi nó trong hàm main và truyền cho nó tham số nếu cần.

Khi bạn gọi hàm trong hàm main thì các câu lệnh bên trong hàm sẽ được thực thi, sau khi thực thi hết các câu lệnh thì hàm kết thúc và chương trình tiếp tục thực hiện các câu lệnh bên dưới hàm.

Mỗi lần bạn gọi hàm thì các câu lệnh trong hàm sẽ được thực hiện.



Ví dụ 1 :

#include <stdio.h>

void greet(){

printf("Hello BCN !\n");

printf("Team App\n");

}

int main(){

printf("Before\n");

greet(); // Lời gọi hàm

printf("After\n");

return 0;

}

Output :

Before

Hello BCN !

Team App

After

Giải thích :

1. Hàm main thực hiện câu lệnh đầu tiên in ra "Before"
2. Câu lệnh thứ 2 trong main là lời gọi hàm greet(), chương trình tiến hành nhảy vào bên trong hàm greet() và thực hiện lần lượt 2 câu lệnh in ra "Hello BCN !" và "Team App”
3. Sau khi thực hiện xong 2 câu lệnh thì hàm greet() kết thúc tương đương câu lệnh thứ 2 trong main thực hiện xong
4. Hàm main tiếp tục thực hiện câu lệnh thứ 3 in ra "After" sau đó kết thúc chương trình

**3. Đối Số Và Tham Số**

**Tham số** (Parameter) hay tham số hình thức là các thành phần khi bạn xây dựng hàm, xem xét ví dụ dưới đây thì a, b, c sẽ được gọi là tham số

**Đối số** (Argument) hay tham số chính thức là các giá trị bạn truyền vào cho hàm khi gọi hàm, xem xét ví dụ dưới đây thì m, n, p được gọi là đối số

Khi bạn gọi hàm thì lần lượt giá trị của các đối số sẽ được gán cho tham số, trong ví dụ dưới thì m được gán cho a, n được gán cho b, p được gán cho c. Những gì thay đổi trên tham số sẽ không có ảnh hưởng gì tới đối số

**Chú ý :** Kiểu dữ liệu của đối số và tham số nên trùng nhau hoặc của tham số nên là kiểu dữ liệu lớn hơn kiểu dữ liệu của đối số. Ví dụ bạn xây dựng 1 hàm có tham số là long long thì nó có thể áp dụng với 1 số int nhưng ngược lại thì không.

Ví dụ 1 :

#include <stdio.h>

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

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

}

int main(){

int m = 100, n = 200, p = 300;

display(m, n, p);

return 0;

}

Output :

100 200 300

Ví dụ 2 :

#include <stdio.h>

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

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

}

int main(){

int x = 30;

display(100.2, 200.3, x);

return 0;

}

Output :

200 100 30

Giải thích :

1. a được gán giá trị là 100.2 nhưng do a có kiểu là int nên chỉ lưu được 100
2. b được gán giá trị là 200.3 nhưng do b có kiểu là int nên chỉ lưu được 200
3. c được gán giá trị của x tương tương với 30
4. Câu lệnh in ra b, c, a lần lượt là 200 100 30

Ví dụ 3 : Thay đổi tham số sẽ không ảnh hưởng gì tới đố số

#include <stdio.h>

void thaydoi(int n){

n += 28;

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

}

int main(){

int a = 100;

thaydoi(a);

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

return 0;

}

Output :

128

100

Giải thích :

1. Tham số n được gán giá trị của đối số a nên n = 100
2. n += 28 => n = 128, câu lệnh printf bên trong hàm thay đổi in ra n sẽ là 128
3. Hàm kết thúc, câu lệnh in ra a thì a vẫn là 100

**4.Câu Lệnh Return**

Khi hàm của bạn thực hiện các chức năng tính toán và mong muốn trả về 1 giá trị cụ thể thì bạn cần câu lệnh return, trong các ví dụ trên thì hàm của mình đều không cần trả về giá trị nào nên mình để kiểu trả về là void.

Giả sử bạn cần viết hàm tính tổng của 3 số nguyên, khi đó hàm sẽ nhận vào tham số là 3 số nguyên và cần trả về tổng 3 số. Vậy bạn cần xác định tổng của 3 số đó sẽ có kiểu là gì để làm kiểu trả về cho hàm và bổ sung thêm câu lệnh return kèm giá trị bạn muốn trả về.

Có thể hiểu đơn giản giá trị đi kèm với return chính là kết quả mà hàm trả về cho bạn khi bạn gọi hàm.

Ví dụ 1 : Hàm tính tổng 3 số nguyên int

#include <stdio.h>

int tong(int a, int b, int c){

int sum = a + b + c;

return sum;

}

int main(){

printf("%d\n", tong(10, 20, 30));

printf("%d\n", tong(28, 28, 28));

return 0;

}

Output :

60

84

Ví dụ 2 : Hàm tính giai thừa của số n

#include <stdio.h>

long long factorial(int n){

long long gt = 1;

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

gt \*= i;

}

return gt;

}

int main(){

printf("%lld\n", factorial(5));

printf("%lld\n", factorial(10));

return 0;

}

Output :

120

3628800

**Chú ý :**

1. Hàm của bạn sẽ kết thúc ngay khi gặp câu lệnh return và giá trị return đầu tiên đó sẽ được trả về cho hàm
2. Kiểu dữ liệu trả về của hàm với kiểu của biến mà bạn sử dụng trong câu lệnh return cần giống nhau. Ví dụ hàm factorial trả về long long thì khi trả về biến gt thì biến gt cũng nên có kiểu long long, nếu bạn sử dụng gt kiểu int có thể bị tràn dữ liệu, khi đó kiểu trả về là long long cũng không tự khôi phục cho bạn được kết quả đúng

Ví dụ 3 :

#include <stdio.h>

int tong(int a, int b){

int res = a + b;

return 28; // kết thúc và trả về luôn giá trị 28

return res; // không được thực hiện

}

int main(){

printf("%d\n", tong(10, 20));

printf("%d\n", tong(100, 200));

return 0;

}

Output :

28

28

Ví dụ 4 :

#include <stdio.h>

void greet(int n){

printf("BCN\n");

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

return; // Kết thúc ngay hàm void

printf("TeamApp\n");

}

int main(){

greet(28);

return 0;

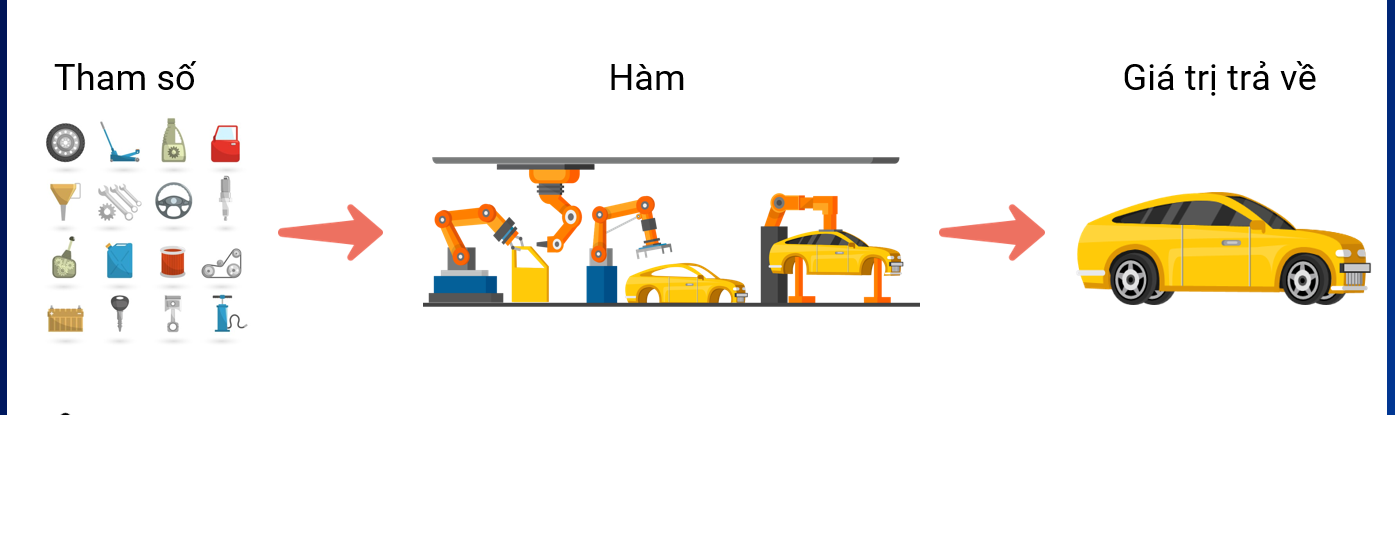
}

Output :

BCN

28

**5.Chú Ý Khi Xây Dựng Hàm**

****

1. Hàm này có cần trả về giá trị không, nếu có thì trả về kiểu dữ liệu là gì ?
2. Hàm này có bao nhiêu tham số, các tham số có kiểu dữ liệu là gì ?
3. Hàm của bạn xây dựng đã đủ tổng quát chưa hay quá quá chi tiết và chỉ phù hợp cho 1 bài toán cụ thể
4. Bạn gọi hàm có đúng thứ tự tham số mà mình mong muốn hay không, kiểu dữ liệu của tham số hình thức và tham số chính thức có hợp lí hay không?
5. Nếu hàm của khác void, thì hàm sẽ kết thúc ngay lập tức khi gặp câu lệnh return một giá trị nào đó, còn nếu hàm void mà các bạn muốn kết thúc tại thời điểm nào đó các bạn có thể sử dụng câu lệnh return; là đủ.

Ví dụ bài toán : Tính tổng các ước của N và in ra màn hình

Cách 1 :

#include <stdio.h>

void xuly(){

int n;

scanf("%d", &n);

int tong = 0;

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

if(n % i == 0){

tong += i;

}

}

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

}

int main(){

xuly();

return 0;

}

Phân tích : Cách làm này không sai, code này sẽ tiến hành nhập, tính tổng và in ra luôn trong hàm xử lý. Tất cả các công việc đều được xử lý bởi hàm xuly(), tuy nhiên cách làm này không phù hợp vì hàm chưa đủ tổng quát.

Ví dụ nếu muốn tính tổng các ước của từng số từ 1 tới n thì hàm này không dùng được, hoặc chỉ muốn lấy ra tổng của 1 số để thực hiện các chức năng khác thì hàm này cũng không sử dụng được.

Cách 2 :

#include <stdio.h>

int tonguoc(int n){

int tong = 0;

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

if(n % i == 0){

tong += i;

}

}

return tong;

}

int main(){

int n;

scanf("%d", &n);

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

return 0;

}

Phân tích : Cách làm này tổng quát hơn, hàm tính tổng ước của bạn có thể sử dụng bất cứ lúc nào bạn cần tính tổng ước của 1 số.

Các thao tác nhập và xuất bạn nên xử lý trong hàm main, hàm chỉ nên có chức năng xử lý nhiệm vụ nhất định.

**Nguyên Mẫu Hàm**

Khai báo nguyên mẫu hàm nhằm cung cấp các thông tin của hàm cho trình biên dịch bao gồm

* Kiểu trả về của hàm
* Tên hàm
* Số lượng tham số của hàm
* Dánh sách kiểu dữ liệu của các tham số

Ví dụ 1 :

#include <stdio.h>

//function prototype

void greet();

int tong(int, int, int);

long long solve(int, long long);

void greet(){

printf("TeamApp\n");

}

int tong(int a, int b, int c){

return a + b + c;

}

long long solve(int a, long long b){

return 2 \* a + 3 \* b;

}

int main(){

greet();

printf("%d\n", tong(10, 20, 30));

printf("%lld\n", solve(10, 20));

return 0;

}

Output :

TeamApp

60

80

Ví dụ 2 : Chương trình bị lỗi biên dịch khi xây dựng hàm không đúng thứ tự

#include <stdio.h>

void B(){

printf("Team App\n");

A();

}

void A(){

printf("BCN\n");

}

int main(){

B();

return 0;

}

Giải thích :

Ví dụ trong chương trình dưới trên, hàm B() gọi hàm A() thì hàm A() cần xây dựng trước hàm B(). Nếu không thì chương trình sẽ bị lỗi biên dịch, điều tương tự xảy ra khi bạn gọi các hàm trong hàm main thì cần xây dựng các hàm này bên trên hàm main.

Khi sử dụng khai báo nguyên mẫu hàm bạn có thể tùy ý gọi các hàm trong chương trình mà không cần xét đến thứ tự.

Ví dụ 3 :

#include <stdio.h>

//function prototype

void A();

void B();

int main(){

B();

return 0;

}

void B(){

printf("Team App\n");

A();

}

void A(){

printf("BCN\n");

}

Output :

Team App

BCN