CHƯƠNG 6
CHƯƠNG TRÌNH CON

Giảng Viên: Nguyễn Văn Thắng

KHÁI NIỆM VỀ HÀM TRONG C++

- Trong những chương trình lớn, có những đoạn chương trình cần lặp lại nhiều lần.
- Để tránh sự lặp lại và việc kiểm tra chương trình được thuận lợi đạt hiệu quả cao, khi viết chương trình, người ta thường phân chia chương trình thành nhiều module, mỗi module giải quyết một công việc nào đó. Các module như vậy gọi là các chương trình con.

Giảng Viên: Nguyễn Văn Thắng

PHÂN LOẠI HÀM

- ***Hàm có hai loại***:
  - Hàm chuẩn : là những hàm do ngôn ngữ cung cấp
  - **Hàm tự định nghĩa** : do người dùng tự xây dựng

Giảng Viên: Nguyễn Văn Thắng

HÀM TOÁN HỌC

- C++ cung cấp một số hàm toán học để có thể sử dụng trong chương trình.
- Muốn sử dụng các hàm toán học thì trong chương trình ta phải khai báo:

#include <math.h>

- Cú pháp chung của một hàm là:

functionName (arguments)

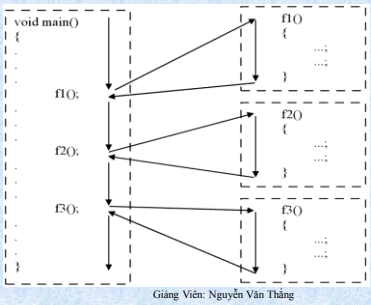
MỘT SỐ HÀM THÔNG DỤNG

Tên Hàm	Công Dụng	Kiểu dữ liệu trả về
abs(x)	Tính trị tuyệt đối của x	Int
fabs()		Double
labs(x)		long int
pow(x1,x2)	tính x1 lũy thừa x2	Double
sqrt(x)	tính căn bậc 2 của x	Double

MỘT SỐ HÀM THÔNG DỤNG

Tên Hàm	**Công Dụng**	**Kiểu dữ liệu trả về**
sin(x)	tính sin x (x tính bằng radian)	Double
cos(x)	tính cos x (x tính bằng radian)	Double
tan(x)	tính tan x (x tính bằng radian)	Double
log(x)	ln(x)	Double
log10(x)	logarit cơ số 10 của x	Double
exp(x)	Ex	Double

Định nghĩa hàm



Định nghĩa hàm

Dạng 1

<kiểu kết quả> **Tên hàm ([*<kiểu tham số>* *<tham số>*|[,…]])**

{

[Khai báo biến cục bộ và các câu lệnh thực hiện hàm]

return <giá trị>;

}

Dạng 2

void **Tên hàm ([*<kiểu tham số>* *<tham số>*|[,…]])**

{

[Khai báo biến cục bộ và các câu lệnh thực hiện hàm]

}

Ví dụ:tìm số lớn nhất của 2 số

```
int max(int a, int b)
{
    return (a>b) ? a:b;
    /* if(a>b)
       return a
    else
       return b
    */
}
```

Dạng nào?

```
void  tinh(int a)
{
    int x=0;
    x=a*k; // k ??
    y=2*x;// y??
}
```

Dạng nào?

Ví dụ :

Viết hàm tìm ước chung lớn nhất của 2 số nguyên a, b.

```
int uscln(int a, int b)
{
    a=abs(a);
    b=abs(b);
    while(a!=b)
    {
        if(a>b)
            a-=b;
        else
            b-=a;
    }
    return a; //hoặc return b;
}
```


Lệnh return

- Lệnh return dùng để thoát khỏi một hàm và có thể trả về một giá trị nào đó.
- return ; /*không trả về giá trị*/
- return <biểu thức>; /*Trả về giá trị của biểu thức*/
- **Lưu ý:**

  Nếu hàm có kết quả trả về, thì bắt buộc phải sử dụng câu lệnh ***return*** *để trả về kết quả cho hàm.*

Gọi hàm

- **Gọi hàm**: Một hàm khi định nghĩa thì chúng vẫn chưa được thực thi, hàm chỉ được thực thi khi trong chương trình có một lời gọi đến hàm đó.
- **Cú pháp gọi hàm:**

<Tên hàm>([Danh sách các tham số])

Ví dụ 1:

```
#include <iostream.h>
#include <conio.h>
int main()
{
   int a, b, c;
   cout<<“a= “; cin>>a;
   cout<<“b=“; cin>>b;
   cout<<“ So lon la “<<max(a, b);
getch();
return 0;
}
```

```
int max(int a, int b)
{
   return (a>b) ? a:b;
   /* if(a>b)
     return a
   else
     return b
   */
}
```


Ví dụ 2:

```
void main()
{
   unsigned int a, b, USC;
   cout<<“Nhap a,b: ”;
   cin>>a>>b;
   USC = ucsln(a,b);
   cout<<“Uoc chung lon nhat la: ”, USC); // có cách ??
   getch();
}
```


Nguyên tắc hoạt động của hàm

```
void main()
{
   unsigned int a, b, USC;
   cout<<“Nhap a,b: ”;
   cin>>a>>b;
   USC = uscln(a,b);
   cout<<“Uoc chung lon
   nhat la: ”, USC);
   getch();
}
```

```
int uscln(int a, int b)
{
        a=abs(a);
        b=abs(b);
        while(a!=b)
        {
                if(a>b)
                        a-=b;
                else
                        b-=a;
        }
        return a;
}
```

Prototype của hàm

- Dùng để loại trừ việc bắt buộc phải định nghĩa hàm trước khi gọi.
- Prototype khai báo giống như header của hàm :

Ví dụ:

Header : void dispayMessage()
Prototype : void dispayMessage()

- *Sử* dụng prototype của hàm : tương tự như viết định nghĩa hàm mà không có thân của hàm.

Prototype của hàm

- Chương trình **bắt buộc phải có prototype của hàm** hoặc phải **bắt buộc viết định nghĩa của hàm trước khi được gọi.**
- Sau khi đã sử dụng prototype của hàm, ta có thể viết định nghĩa chi tiết hàm ở bất kỳ vị trí nào trong chương trình.

Ví dụ:

```
#include<istream.h>
// Function prototypes
void first();
void second();
int main()
{  cout<<“I am starting in function main.\n”;
   first();      // gọi hàm first
   second(); // gọi hàm second
   cout <<“Back in function main again.\n”;
   return 0;
}
```

Ví dụ:

```
void first()        // định nghĩa function first
{
  cout<<"I am now inside the function first().";
}
void second()     // định nghĩa function first
{
  cout<<"I am now inside the function second().";
}
```


Prototype của hàm

```
//void first();?? Chương trình có chạy không?
int main()
{   first();// gọi hàm first
    return 0;
}
void first()// định nghĩa hàm
{
    cout<<"I am now inside the function first().";
}
```


Truyền tham số cho hàm

Tham số giá trị (pass value parameter)

- Mặc định, việc truyền tham số cho hàm trong C/C++ là truyền theo giá trị, nghĩa là khi gọi ***hàm có tham số***, ta truyền các ***giá trị*** cho hàm
- Khi gọi hàm, giá trị được truyền vào hàm gọi là ***đối số (argument)***
- Biến trong hàm dùng để giữ giá trị được truyền vào thông qua đối số được gọi là **tham số *(parameter).***
- Tham số này còn được gọi là ***đối số hình thức (formal argument)***

Ví dụ :

```
void displayValue (int num)
{
        cout<<"The value is "<<num<<endl;
}
```

num : là tham số hay còn gọi là đối số hình thức

Khi gọi : ***displayValue(5);***

5 chính là đối số của hàm ***displayValue***

Ví dụ:

```
int tong(int a, int b)
{
   return a+b;
}
void main()
{
   int x=5; int y=3; int z;
   z=tong(x,y);
   cout<<z;
}
```

Ví dụ:

- Giả sử chúng ta gọi hàm **addition** như sau:
- int x=5, y=3, z;
  z = addition (x , y);

```
int addition (int a, int b)

                  5     3
z = addition (  x   ,   y  );
```

- Trong trường hợp này khi gọi hàm **addition** thì các giá trị **5** and **3** được truyền cho hàm

Ví dụ:

```
#include <iostream.h>
#include<conio.h>
void Foo (int num)
{
   num = 0;
   cout << "num = " << num << '\n';
}
void main ()
{
   int x = 10;
   Foo(x);
   cout << "x = " << x << '\n';
   getch();
}
```


Lưu ý:

Đối với những hàm có tham số *(parameter)*:

- ***Prototype của hàm*** phải bao gồm kiểu dữ liệu của từng tham số mà có thể không cần quan tâm đến tên của tham số đó :
  void displayValue(int) //prototype của hàm displayValue
  hoặc: void displayValue(int num)
- ***Header của hàm*** phải bao gồm cụ thể từng ***kiểu dữ liệu đi kèm với tất cả tên của các tham số***
  void displayValue(int num) //header-hàm displayValue

Lưu ý:

- Khi đưa đối số vào cho tham số của hàm, đối số có thể được tự động thay đổi cho phù hợp trong trường hợp kiểu dữ liệu của đối số không phù hợp với kiểu dữ liệu của tham số.

Ví dụ :

```
void displayValue(int)//prototype
```

Gọi : displayValue(4.7) //đối số thuộc real

Đối số mặc định

- **Đối số mặc định** *(default argument)* dùng để chuyển tự động cho tham số khi đối số thực sự không xuất hiện trong hàm khi hàm được gọi
- ***Đối số mặc định bắt buộc phải được khai báo liệt kê trong prototype của hàm.***

Ví dụ:

```
void evenOrOdd(int = 0);
void showArea(float = 20.0, float = 10.0) Hoặc
void showArea(float length= 20.0, float width= 10.0)
```


Đối số mặc định

- Nếu hàm không được khai báo prototype trước, vẫn có thể khai báo các đối số mặc định trong header của hàm
- Ví dụ:

```
void showArea(float length= 20.0, float width= 10.0)
{
    float area = length*width;
    cout<<"The area is "<< area<<endl;
}
```

Đối số mặc định

```
#include<iostream.h>
void displayStars(int=10, int=1);
void main()
{
   displayStars(); cout<<endl;
   displayStars(5); cout<<endl;
   displayStars(7,3);
}
void displayStars(int cot, int dong)
{  for (int i=0;i<dong;i++)
   {     for (int j=0;j<cot; j++)
                cout<<"*";
         cout<<endl;
   }
}
```


Đối số mặc định

- Khi gọi hàm có đối số mặc định, nếu không có tham số đầu tiên thì cũng không được có các tham số sau.

  Ví dụ : gọi hàm displayStars (,3)//thiếu đối số cols(sai)
- Các hàm có sử dụng nhiều tham số, thì có thể một số tham số có đối số mặc định, một số khác thì không

  int getSum(int, int=0, int=0);
- Các tham số không có đối số mặc định bắt buộc phải có giá trị khi gọi hàm.

  Gọi cout<<getSum(3);

  cout<<getSum();// SAI

Đối số mặc định

- Trong hàm vừa có đối số không mặc định, vừa có đối số mặc định, thì bắt buộc các tham số có đối số mặc định phải được khai báo ở sau cùng.

  int getSum(int, int=0, int=0);// ĐÚNG

  int getSum(int, int=0, int); // SAI

  void displayStars(int =10, int)//SAI
- Khi một đối số không xuất hiện khi hàm được gọi thì bắt buộc các đối số phía sau cũng không được xuất hiện

  sum = getSum(num1, num2); // ĐÚNG

  sum = getSum(num1, , num3); // SAI

  sum = getSum(num1); // ĐÚNG

  displayStars(,x);//SAI

Tham số là tham trị (call by value)

- Khi truyền tham số giá trị cho hàm, và khi hàm thực thi thì ***đối số thực sự sẽ không bị ảnh hưởng***
- Đối số được truyền vào cho tham số thực chất chỉ là bản sao chép của đối số thực.

Tham số tham chiếu (pass reference parameter)

- Cơ chế truyền tham số tham chiếu cho phép hàm làm việc thẳng với đối số thật sự khi chương trình con được gọi.
- Mỗi sự thay đổi của tham số sẽ ảnh hưởng trực tiếp tới đối số tương ứng.
- Cơ chế truyền truyền tham số tham chiếu cho phép hàm có thể thay đổi dữ liệu của môi trường nơi gọi hàm.
- Cho phép một hàm có khả năng trả về nhiều hơn một giá trị.

Tham số tham chiếu (pass reference parameter)

- Để thực hiện cơ chế truyền tham số tham chiếu, ta sử dụng biến tham chiếu ***reference variable.***

Kiểu dữ liệu & tên biến

- Biến tham chiếu là một đại diện của một biến khác sử dụng ở nơi gọi hàm.
- Tất cả mọi sự xảy ra ở biến tham khảo thực chất là xảy ra trên một biến khác mà biến tham khảo làm đại diện.

Tham số tham chiếu (pass reference parameter)

Ví dụ:

```
void doubleNum(int &);//Khai báo prototype
void doubleNum(int &refVar)// Định nghĩa hàm cũng phải có &
{
        refVar*=2;
}
```


Ví dụ:

```
void duplicate (int& a, int& b, int& c)
{
   a*=2;
   b*=2;
   c*=2;
}
void main ()
{
   int x=1, y=3, z=7;
   duplicate (x, y, z);
    cout << "x=" << x << ", y=" << y << ", z=" << z;
    getch();
}
```


Tham số tham chiếu (pass reference parameter)

- Khi truyền tham số dưới dạng tham biến, ta đang truyền chính biến đó. Vì vậy, bên trong hàm nếu có bất kì sự thay đổi giá trị nào được thực hiện với tham số đó sẽ ảnh hưởng trực tiếp đến giá trị của biến tương ứng với tham số đó.

```
void duplicate (int& a,int& b,int& c)
                  x     y     z
    duplicate (   x  ,  y  ,  z  );
```

```
void duplicate(int a, int b)
{
        a*=2;
        b*=2;
}
void main()
{
        int x=5;
        int y=7;
        duplicate(x,y)
        cout<<“x=“<<x<<endl;
        cout<<“y=“<<y;
        getch();
}
```

Kết quả: x=5 , y=7

```
void duplicate(int&a, int &b)
{
        a*=2
        b*=2;
}
void main()
{
        int x=5;
        int y=7;
        duplicate(x,y)
        cout<<“x=“<<x<<endl;
        cout<<“y=“<<y;
        getch();
}
```

Kết quả: x=10 , y=14


```
void duplicate(int &a, int b)
{
        a*=2;
        b*=2;
}
void main()
{
        int x=5;
        int y=7;
        duplicate(x,y)
        cout<<“x=“<<x<<endl;
        cout<<“y=“<<y;
        getch();
}
```

Kết quả: x=10 , y=7

```
void duplicate(int&a, int &b)
{
        a*=2
        b*=2;
}
void main()
{
        int x=5;
        int y=7;
        duplicate(x,y)
        cout<<“x=“<<x<<endl;
        cout<<“y=“<<y;
        getch();
}
```

Kết quả: x=10 , y=14

Ví dụ Hàm swap có công dụng đổi giá trị của hai biến được dùng để truyền dữ liệu cho các tham số của nó.

```
#include <iostream.h>
#include <conio.h>
void  swap(int&, int&);
void main()
{ int    i=5,j=10;
   cout <<"\n Truoc khi goi ham swap\n“;
   cout <<"i= "<<i<<"  "<<"j= "<<j <<endl;
   swap(i,j);
```

i 5 j 10

Ví dụ Hàm swap có công dụng đổi giá trị của hai biến được dùng để truyền dữ liệu cho các tham số của nó.

```
void  swap(int& x, int& y)
 {
      int tam;
      tam=x;
      x=y;
      y=tam;
 }
```

x 5 y 10

5 tam

Ví dụ Hàm swap có công dụng đổi giá trị của hai biến được dùng để truyền dữ liệu cho các tham số của nó.

```
  cout <<"Sau khi goi ham swap\n“;
   cout <<"i= "<<i<<"  " <<"j= "<<j <<endl;
  getch();
}
```

i 10 j 5

Giảng Viên: Nguyễn Văn Thắng

Khi nào dùng tham trị -tham biến

Truyền tham trị khi:

- Đối số cần truyền là một hằng
- Đối số cần truyền là một biến nhưng không cần thay đổi giá trị khi hàm thực thi xong.
- Khi hàm thực thi xong chỉ cần trả về một giá trị , nên dùng cơ chế truyền tham trị và hàm return để trả về kết quả .

Giảng Viên: Nguyễn Văn Thắng

Khi nào dùng tham trị -tham biến

Truyền tham biến khi:

Khi hai hay nhiều biến cần truyền vào hàm và sau khi hàm thực thi xong, thì giá trị các biến này phải nhận được kết quả mới.

Ví dụ:

```
int addNums(int, int); //Prototype function
void getNums(int &, int &); //Prototype function
int main()
{
   int n1, n2;
   getNums(n1, n2);
   cout<<“gia tri cua n1 va n2:  ”<<n1<<“ va “<<n2<<endl;
   cout<<“tong cua n1 va n2 la ”<<addNums(n1,n2)<,endl;
   return 0;
}
```


Ví dụ(tt)

```
void getNums(int &a, int &b)
{       cout<<“a=”; cin>>a;
        cout<<“b= ”;cin>>b;
}
void addNums(int &num1, int &num2)
{
                return num1+num2;
}
```

Phạm vi của biến (scope variables)

Biến cục bộ (local variables)

- Biến cục bộ là biến được khai báo trong thân của một hàm, các biến này chỉ được hiểu bên trong phạm vi của hàm khai báo nó.

Ví dụ:

```
void  sub_fun()
{
   int n2=30;
   cout <<"Trong ham sub_fun() n2= " <<n2<<endl //30
   return;
}
```


Biến toàn cục (global variables)

- Biến toàn cục là biến được khai báo bên ngoài các hàm, những biến này được dùng chung cho tất cả các hàm được khai báo sau nó.

```
#include <iostream.h>
#include <conio.h>
int n1;
void  sub_fun()
void main()
{
    int  n2;
    n1=10; n2=20;//n2 là biến cục bộ của hàm main
    cout <<"Trong ham main() n1= "<<n1<<endl  //10
         <<"Trong ham main() n2= "<<n2<<endl; //20
    sub_fun();
    cout <<"Trong ham main() sau khi goi sub_fun n1= "<<n1<<endl //40
         <<"Trong ham main() sau khi goi sub_fun n2= "<<n2<<endl;//20
    getch();
    return;
}
void  sub_fun()
 {
    int n2=30; //biến cục bộ của hàm sub_fun()
    cout <<"Trong ham sub_fun() n1= " <<n1<<endl //10
    cout<<"Trong ham sub_fun() n2= " <<n2<<endl;  //30
    n1=40;//biến toàn cục
    return;
 }
```

Biến toàn cục (global variables)

- Các biến toàn cục không được khởi tạo, sẽ được khởi tạo tự động là ***0 nếu là kiểu số***, và ***NULL nếu là kiểu ký tự***.
- Các ***biến hoặc hàm toàn cục*** được định nghĩa một lần ở ***mức toàn cục***.
- Biến hay hàm toàn cục được truy xuất tại bất kỳ vị trí nào trong chương trình.

Biến toàn cục (global variables)

- Thời gian sống của biến tùy thuộc vào phạm vi của nó. Biến toàn cục tồn tại suốt thời gian thực hiện chương trình
- Các biến cục bộ chỉ tồn tại trong thời gian hàm chứa nó thực thi.
- Không gian bộ nhớ cho các biến toàn cục được dành riêng trước khi sự thực hiện của chương trình bắt đầu
- Không gian bộ nhớ cho các biến cục bộ được cấp phát ở thời điểm thực hiện chương trình.

Toán tử phạm vi (scope resolution)

- Phạm vi cục bộ ghi chồng lên phạm vi toàn cục nên một ***biến cục bộ*** có ***cùng tên với biến toàn cục*** làm cho biến toàn cục không thể truy xuất được tới phạm vi cục bộ.

Toán tử phạm vi (scope resolution)

Ví dụ:

```
#include  <iostream.h>
#include  <conio.h>
float n=42.8;
void sub();
void main()
{
    clrscr();
    float n=30.5;
    cout <<"Giá trị của  n= " <<n<<endl;  //30.5;
    sub();
    getch();
}
void sub();
{
    cout <<"trong sub n= "<<n <<endl; //42.8
}
```

Trong trường hợp trên nếu muốn hàm main in ra giá trị của biến toàn cục thì phải sử dụng toán tử scope resolution :: ngay trước tên biến.

```
#include  <iostream.h>
#include  <conio.h>
float n=42.8;
void sub();
void main()
{  float n=30.5;
   cout <<"Giá trị của  n= " <<::n<<endl;  //42.8;
   sub();
   getch();
}
void sub()
{
   cout <<"trong sub n= "<<n <<endl; //42.8
}
```

Đệ Quy

- Đệ quy tuyết tính
- Đệ quy nhi phân
- Đệ quy phi tuyến
- Đệ quy hỗ tương

Đệ quy Tuyến Tính

```
KDL TenHam(<Danh sách tham số>)
{
    if(<Điều kiện dừng>)
{
    …
    return <Giá trị trả về>;
}
…
…TenHam(<Danh sách tham số>);
…
}
```

```
long GiaiThua( int n)
{
   if(n==1)
     return 1;
   else
    return GiaiThua(n-1) *n;
}
```


Đệ quy nhi phân

```
KDL TenHam(<Danh sách tham số>)
{
        if(<Điều kiện dừng>)
{
        …
        return <Giá trị trả về>;
}
…
…TenHam(<Danh sách tham số>);
…
…TenHam(<Danh sách tham số>);
}
```

```
long int Fibonaci(int n)
{
 if((n==0 )|| (n==1))
    return 1;
 else
   return Fibonaci(n-1)+Fibonaci(n-2);
}
```

Đệ Quy Đệ quy phi tuyến

```
KDL TenHam(<Danh sách tham số>)
{
 For (int i=1; i<=n; i++)
 {
      If (<Điều kiện dừng>)
      {
                    …
      }
      else
      {
            …
            TenHam(<Danh sách tham số>);
            …
      }
 }
}
```

```
x(0)=0
x(n)=n2x(0)+(n-1)2x(0)+(n-
1)2x(0)+  +22x(n-2)+12x(n-1)
long int TinhXn(int n)
{
  if(n==0)
      return 1;
 long int s=0;
 for(int i=1; i<=n; i++)
   s+=i*i*TinhXn(n-i);
  return s;
}
```


Đệ quy hỗ tương

```
KDL TenHam2(<Danh sách tham số>);
KDL TenHam1(<Danh sách tham số>)
{
            …
TenHam2(<Danh sách tham số>);
…
}
KDL TenHam2(<Danh sách tham số>)
{
            …
TenHam1(<Danh sách tham số>);
…
}
```

```
x(0)=1
y(0)=0
x(n)=x(n-1)+y(n-1)                khi n>0
y(n)=3*x(n-1)-2*y(n-1)                    khi n>0
long int TinhYn(int n);
long int TinhXn(int n)
{
  if(n==0)
    return 1;
  return  TinhXn(n-1)+TinhYn(n-1);
}
long int TinhYn(int n)
{
  if(n==0)
    return 0;
    return (3*TinhXn(n-1)-2*TinhYn(n-1));
}
```