LẬP TRÌNH ĐƠN THỂ (HÀM – FUNCTION)

1.Cấu trúc hàm và lý do sử dụng hàm

1.1. Khái niệm

- Phân chia chương trình thành các phần nhỏ - các chương trình con (routine) hay còn gọi là các hàm (function)
- Các hàm có thể được dùng lại nhiều lần → thành lập các thư viện hàm. (vd: stdio, stdlib, conio, math, string,...)
- Một chương trình c là một tập hợp các hàm tương tác bằng cách gọi lẫn nhau và truyền các thông tin qua lại giữa các hàm.
- Trong 1 chương trình của C/C++ có thể có nhiều hàm, tối thiểu có một hàm là hàm main(), nó đánh dấu điểm bắt đầu và điểm kết thúc của chương trình.

1.1. Khái niệm

Hàm?

Hàm là một đoạn chương trình độc lập **thực hiện trọn vẹn một công việc nhất định** sau đó trả về giá trị cho chương trình gọi nó, hay nói cách khác hàm là sự chia nhỏ của chương trình. Một hàm có tên gọi, có đầu vào, đầu ra

1.1. Khái niệm

- Hàm trong C có thể trả về kết quả thông qua tên hàm, có thể không trả về kết quả.
- Một hàm khi được định nghĩa thì có thể được gọi trong chương trình.
- Được gọi nhiều lần với các tham số khác nhau.
- Hàm có hai loại: hàm chuẩn (hàm có sẵn trong thư viện của C++) và hàm tự định nghĩa bởi người sử dụng.

1.1. Khái niệm

Hàm thư viện/hàm chuẩn

- Hàm thư viện là những hàm đã được định nghĩa sẵn trong một thư viện nào đó.
- Muốn sử dụng các hàm thư viện thì phải khai báo thư viện trước khi sử dụng bằng lệnh

#include <tên thư viện.h>

1.1. Khái niệm

Hàm thư viện/hàm chuẩn

- Một số thư viện thường dùng:
  - **a. iostream.h:**
    - Thư viện chứa các hàm vào/ ra chuẩn (*standard input/output*).
    - Gồm các hàm cin(), cout(), printf(), scanf(), getc(), putc(), gets(), puts(), fflush(), fopen(), fclose(), fread(), fwrite(), getchar(), putchar(), getw(), putw()…
  - **b. conio.h** :
    - Thư viện chứa các hàm vào ra trong chế độ DOS Gồm các hàm getch(), getche(), cgets(), cputs(), putch(), clreol(),…

1.1. Khái niệm

Hàm thư viện/hàm chuẩn

- Ý nghĩa của một số thư viện thường dùng:
  - **c. math.h:**
    - Thư viện chứa các hàm toán học.
    - Gồm các hàm abs(), sqrt(), log(). log10(), sin(), cos(), tan(), acos(), asin(), atan(), pow(), exp(),…
  - **d. alloc.h:**
    - Thư viện chứa các hàm liên quan đến việc quản lý bộ nhớ.
    - Gồm các hàm calloc(), realloc(), malloc(), free(), farmalloc(), farcalloc(), farfree(), …

1.1. Khái niệm

Hàm thư viện/hàm chuẩn

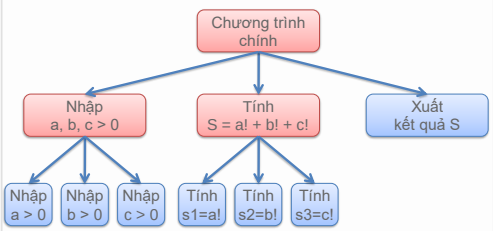
- Ý nghĩa của một số thư viện thường dùng:
  - **e. io.h:**
    - Thư viện chứa các hàm vào ra cấp thấp.
    - Gồm các hàm: open(), _open(), read(), _read(), close(), _close(), creat(), _creat(), creatnew(), eof(), filelength(), lock(),…
  - **f. graphics.h:**
    - Thư viện chứa các hàm liên quan đến đồ họa.
    - Gồm initgraph(), line(), circle(), putpixel(), getpixel(), setcolor(), …
  - …

Muốn sử dụng các hàm thư viện thì ta phải xem cú pháp của các hàm và sử dụng theo đúng cú pháp

1.1. Khái niệm

- Hàm do người dung tự định nghĩa

Đặt vấn đề: *Viết chương trình tính S = a! + b! + c! với a, b, c là 3 số nguyên dương nhập từ bàn phím.*



1.2. Dạng tổng quát của hàm

- Dạng tổng quát của hàm do người dùng định nghĩa:

```
returnType functionName(parameterList)
{
   body of the function
}
```

Kiểu trả về — Tên hàm — Tham số

1.2. Dạng tổng quát của hàm

Các thành phần của hàm:

1. Tên hàm

- Là một định danh (identifier), do đó nó tuân theo các quy định của ngôn ngữ C/C++ cho định danh.
- Nên đặt tên có ý nghĩa.
- Không đặt tên trùng với tên các hàm hệ thống trong C/C++ hoặc các từ khóa.

1.2. Dạng tổng quát của hàm

Các thành phần của hàm:

2. Danh sách tham số:

- xác định các đối số được đưa vào hàm.
- Các đối số được khai báo trong phần mô tả cài đặt của hàm thì được gọi là các tham số hình thức (formal parameters).
- Mỗi tham số hình thức là một cặp: <type><identifier>

Từ khoá ***void*** có thể được dùng nếu không có tham số hình thức nào cần khai báo. Các tham số trong các hàm khác nhau có thể trùng tên.

- Khi gọi hàm, các đối số đưa vào hàm phải đầy đủ và đúng kiểu như đã khai báo.

1.2. Dạng tổng quát của hàm

Các thành phần của hàm:

3. Kiểu trả về: Là bất kỳ kiểu dữ liệu cơ bản nào của C/C++ (char, int, long, float,...)

- Một hàm được phép trả về cho phần chương trình gọi nó một giá trị: *giá trị trả về*, chương trình gọi hàm có thể sử dụng giá trị trả này.
- Một số hàm không cần trả về các giá trị. Từ khóa **void** được dùng trong khai báo giá trị trả về của các hàm này.
- Kiểu int sẽ là kiểu của trị trả về nếu không chỉ rõ kiểu giá trị trả về trong khai báo hàm.

1.2. Dạng tổng quát của hàm

Các thành phần của hàm:

4. Thân hàm:

- { /* các đoạn mã trong thân hàm */ }
- Các biến có thể được khai báo bên trong hàm → biến cục bộ (local variable)
- Biến cục bộ không được trùng tên với tham số hình thức trong khai báo hàm.
- Các biến cục bộ chỉ có giá trị trong phạm vi của hàm.

1.2. Dạng tổng quát của hàm

Các thành phần của hàm:

5. Lệnh return

- Kết thúc hàm và trả quyền điều khiển về cho phần chương trình có lời gọi hàm.
- Cú pháp:

  return Expr;

  hoặc

  return;
- Hàm tự kết thúc khi thực hiện hết lệnh cuối cùng.

1.2. Dạng tổng quát của hàm

```
1  #include<iostream.h>
2  #include<conio.h>
3  int Tonghaiso(int a,int b);
4  void main()
5  {
6      int c,d,kq;
7      cout<<"Nhap c = ";
8      cin>>c;
9      cout<<"Nhap d = ";
10     cin>>d;
11     kq=Tonghaiso(c,d);
12     cout<<"Tong la: "<<kq;
13 }
14 int Tonghaiso(int a,int b)
15 {
16     return a+b;
}
```

Khai báo hàm

Gọi hàm

Truyền đối số

Định nghĩahàm

Tham số

1.2. Dạng tổng quát của hàm

Chú ý về khai báo và định nghĩa hàm

- Danh sách đối trong khai báo hàm có thể chứa hoặc không chứa tên đối, thông thường ta chỉ khai báo kiểu đối chứ không cần khai báo tên đối, trong khi ở dòng đầu tiên của định nghĩa hàm phải có tên đối đầy đủ.
- Cuối khai báo hàm phải có dấu chấm phẩy (;), trong khi cuối dòng đầu tiên của định nghĩa hàm không có dấu chấm phẩy.
- Hàm có thể không có đối (danh sách đối rỗng), tuy nhiên cặp dấu ngoặc sau tên hàm vẫn phải được viết. Ví dụ clrscr(), lamTho(), vietGiaoTrinh(), ...
- Một hàm có thể không cần phải khai báo nếu nó được định nghĩa trước khi có hàm nào đó gọi đến nó

1.2. Dạng tổng quát của hàm

```
#include<iostream.h>
int conghaiso(int a,int b);
void main()
{

}
int conghaiso(int a,int b)
{
    return a+b;
}

void ham1(int x,int y)
{
    return x+y;
}
```

SAI

- Khi một hàm muốn trả về một giá trị nào đó thì chúng ta dùng return . Bất kỳ kiểu dữ liệu nào của hàm cũng có thể sử dụng return NGOẠI TRỪ kiểu **void**

Hàm có kiểu void đôi khi được gọi là Thủ Tục

1.3. Gọi và sử dụng hàm

- Một hàm khi đã định nghĩa nhưng 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 thực])

1.3. Gọi và sử dụng hàm

- Khi hàm cần nhận đối số (*arguments*) để thực thi thì khi khai báo hàm cần khai báo danh sách các tham số để nhận giá trị từ chương trình gọi. Các tham số này được gọi là ***tham số hình thức***.

Ví dụ:

```
int min(int a, int b)
{
    if(a<b)
        return a;
    else
        return b;
}
```

Tham số hình thức

1.3. Gọi và sử dụng hàm

- Khi gọi hàm, ta cung cấp các giá trị thật, các giá trị này sẽ được sao chép vào các tham số hình thức và các giá trị thật được gọi là ***tham số thực***.

Ví dụ: Để tìm giá trị nhỏ nhất của 2 số 5 và 6 ta gọi hàm **min(5, 6)**

Tham số thực

min(int a, int b)

1.3. Gọi và sử dụng hàm

Chú ý:

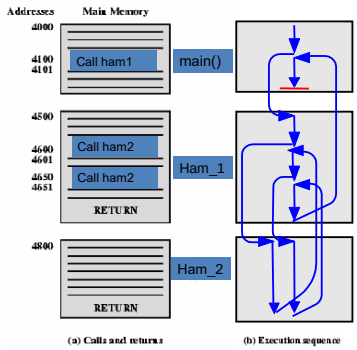
- Danh sách tham đối thực sự còn gọi là danh sách giá trị gồm các giá trị cụ thể để gán lần lượt cho các đối hình thức của hàm
- Danh sách tham đối thực sự truyền cho tham đối hình thức có số lượng bằng với số lượng đối trong hàm và được truyền cho đối theo thứ tự tương ứng
- Các giá trị tương ứng được truyền cho đối phải có kiểu cùng với kiểu đối
- Khi một hàm được gọi, nơi gọi tạm thời chuyển điều khiển đến thực hiện dòng lệnh đầu tiên trong hàm được gọi. Sau khi kết thúc thực hiện hàm, điều khiển lại được trả về thực hiện tiếp câu lệnh sau lệnh gọi hàm của nơi gọi.

1.4. Nguyên tắc hoạt động của hàm

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

```
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;
}
```

Hoạt động của CT khi có hàm



1.5. Phạm vi truy nhập của biến

- Phạm vi truy cập (scope) của biến xác định vùng chương trình có thể truy cập đến biến.
- Biến được khai báo trong khối lệnh (nằm giữa { }) có thể được truy cập bởi các lệnh nằm trong cùng khối và các lệnh thuộc các khối con.
- Biến được khai báo "ngoài cùng" có phạm vi truy cập trong toàn chương trình → biến toàn cục (global variables).
- Biến cục bộ (local variable) được khai báo và sử dụng trong phạm vi một khối lệnh và các khối lệnh con.
- Biến thuộc phạm vi trong cùng được tham chiếu đến đầu tiên.

1.5. Phạm vi truy nhập của biến

```
int a;
    int Ham1()
    {
        int a1;
    }

    int Ham2()
    {
        int a2;
        {
            int a21;
        }
    }

    void main()
    {
        int a3;
    }
```

2. Truyền tham số cho hàm: tham trị, tham chiếu

2.1. Truyền Tham số cho hàm

Trong C++ có 2 cách truyền tham số cho hàm:

- Truyền giá trị (*call by value*): các đối số đưa vào hàm được sao chép vào các tham số hình thức. các giá trị của các đối số được sử dụng trong hàm nhưng những thay đổi của tham số hình thức trong hàm không làm thay đổi giá trị của các đối số truyền vào.

```
void truyenGiaTri(int x)
{
    ...
    x++;
}
```

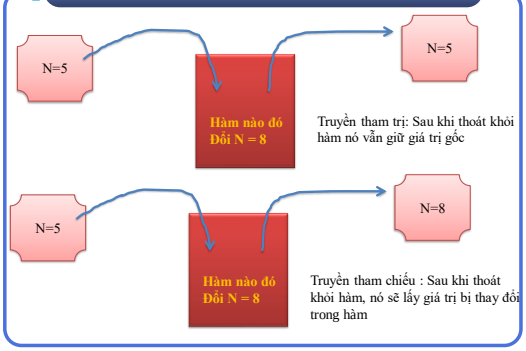
2.1. Truyền Tham số cho hàm

Trong C++ có 2 cách truyền tham số cho hàm:

- Truyền tham chiếu (*call by reference*): các tham chiếu đến các tham số hình thức là tham chiếu đến các đối số (địa chỉ của tham đối được truyền vào trong hàm). Giá trị của các đối số có thể được thay đổi từ xử lý bên trong hàm.

```
void truyenThamChieu(int &x)
{
        ...
        x++;
}
```

2.1. Truyền Tham số cho hàm



2.1. Truyền Tham số cho hàm

```
//Truyền tham trị
#include <iostream.h>
void hoanVi(int x, int y)
{  int Temp;
   Temp = x;
   x    = y;
   y    = Temp;
  cout<<"\nTrong ham-";
  cout<<"Left = "<<x<<" Right = "<<y;
}
int main(void)
{  int Left, Right;
   Left = 5; Right = 7;
   hoanVi(Left, Right);
   cout<<"\nNgoai ham -";
   cout<<"Left = "<<Left<<" Right = "<<Right;
    return 0;
}
```

Kết quả ????

```
Trong ham - Left = 7 Right = 5
Ngoai ham - Left = 5 Right = 7
```

2.1. Truyền Tham số cho hàm

```
//Truyền tham chiếu
#include <iostream.h>
void hoanVi(int &x, int &y)
{  int Temp;
   Temp = x;
   x    = y;
   y    = Temp;
  cout<<"\nTrong ham-";
  cout<<"Left = "<<x<<" Right = "<<y;
}
int main(void)
{  int Left, Right;
   Left = 5; Right = 7;
   hoanVi(Left, Right);
   cout<<"\nNgoai ham -";
   cout<<"Left = "<<Left<<" Right = "<<Right;
    return 0;
}
```

Kết quả ????

```
Trong ham - Left = 7 Right = 5
Ngoai ham - Left = 7 Right = 5
```

2.1. Truyền Tham số cho hàm

Ví dụ: Viết chương trình hoán vị 2 phần tử

```
#include<stdio.h>

void hoanVi1 (int x, int y)
{ int temp = x;
  x = y;
  y = temp;
}
void hoanVi2 (int *x, int *y)
{ int temp = *x;
  *x = *y;
  *y = temp;
}
void hoanVi3 (int &x, int &y)
{
  int temp = x;
  x = y;
  y = temp;
}
```

```
int main()
{ int m=12; n=28;
  hoanVi1(m,n);
  cout<<"m= "<<m<<" n = "<<n);
  hoanVi2(&m,&n);
  cout<<"m= "<<m<<" n = "<<n);
  hoanVi3(m,n);
  cout<<"m= "<<m<<" n = "<<n);
return 0;
}
```

Kết quả ????

2.1. Truyền Tham số cho hàm

- Lưu ý
  - Trong một hàm, các tham số có thể truyền theo nhiều cách.

```
void honHop(int x, int &y)
{
        ...
        x++;
        y++;
}
```

➢ ***Khi nào sử dụng truyền tham trị/tham chiếu????***

2.2. Prototype (nguyên mẫu)của hàm

- Chương trình **bắt buộc phải có prototype của hàm** hoặc phải **bắt buộc viết định nghĩa của hàm trước khi 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.

2.2. Prototype (nguyên mẫu)của hàm

```
#include <iostream.h>// Khai báo thư viện iostream.h
int max(int x, int y);// khai báo nguyên mẫu hàm max
 int main()//hàm main (sẽ gọi các hàm thực hiện)
{
    int a, b;// khai báo biến
    cout<<" Nhap vao 2 so a, b ";
    cin>>a>>b;
    cout<<"so lon nhat la:"<< max(a,b);
}
int max(int x, int y)// Định nghĩa hàm max(a,b)
{
    return (x>y) ? x:y;
}
```

2.3. Nguyên tắc xây dựng hàm

1. Mỗi hàm chỉ thực hiện một công việc duy nhất.
2. Độc lập với các hàm khác.
3. Mỗi hàm nên che giấu thông tin.
4. Tham số của hàm cần được chỉ rõ.
5. Tính tái sử dụng càng cao tốt.
6. Các dữ liệu mà hàm sử dụng:
   - Giữ cho các mỗi liên hệ càng đơn giản càng tốt. Tránh sử dụng biến toàn cục càng nhiều càng tốt.
   - Nếu sử dụng biến toàn cục, viết hướng dẫn chi tiết.

3. Hàm và mảng dữ liệu

3.1. Truyền mảng 1 chiều cho hàm

Có 2 cách truyền:

Cách thứ nhất đối được khai báo bình thường như khai báo biến mảng nhưng không cần có số phần tử kèm theo.

ví dụ:

```
int x[ ];
float x[ ];
```

Cách thứ hai khai báo đối như một con trỏ kiểu phần tử mảng.

ví dụ:

```
int *p;
float *p;
```

3.1. Truyền mảng 1 chiều cho hàm

Ví dụ: : Hàm nhập và in giá trị 1 vectơ

```
void nhap(int x[ ], int n)        // hoặc: void nhap(int *x, int n)
{   int i;
    for (i=0; i<n; i++)
        cin >> x[i];              // hoặc cin >> *(x+i)
}
void in(int int x[], int n)
{   int i;
    for (i=0; i<n; i++)
        cout <<"  "<<x[i] ;
}
int main()
{   int a[10] ;          // mảng a chứa tối đa 10 phần tử
    nhap(a,7);           // vào 7 phần tử đầu tiên cho a
    in(a,3);             // ra 3 phần tử đầu tiên của a
}
```

3.2. Truyền mảng 2 chiều cho hàm

Ta có hai cách khai báo:

1- Khai báo theo đúng bản chất của mảng 2 chiều (float x[m][n]) do C++ qui định, tức x là mảng 1 chiều m phần tử, mỗi phần tử của nó có kiểu float[n]

```
float x[ ][n] ;          // mảng với số phần tử không
                         // định trước, mỗi phần tử là n số
float (*x)[n] ;
```

2- Xem mảng float x[m][n] thực sự là mảng một chiều float x[m*n] và sử dụng cách khai báo như trong mảng một chiều, sử dụng con trỏ float *p để truy cập được đến từng phần tử của mảng.

```
k = *(p + i*n + j)
i = k/n
j = k%n
```

3.2. Truyền mảng 2 chiều cho hàm

ví dụ:

Tính tổng các phần tử trong ma trận

```
float tong(float x[ ][10], int m, int n)
// hoặc float tong(float (*x)[10], int m, int n)
{   // m: số dòng, n: số cột
    float t = 0;
    int i, j ;
    for (i=0; i<m; i++)
          for (j=0; j<n; j++)
               t += x[i][j] ;
    return t;
}
```

4. Đệ Qui

3. Đệ qui

- Một hàm được gọi là đệ qui nếu một lệnh trong thân hàm gọi đến chính hàm đó.
- Đệ qui giúp giải quyết bài toán theo cách nghĩ thông thường một cách tự nhiên.
- Đệ qui phải xác định được điểm dừng. Nếu không xác định chính xác thì làm bài toán bị sai và có thể bị lặp vĩnh cửu (Stack Overhead)

3. Đệ qui

- Ví dụ: Định nghĩa giai thừa của một số nguyên dương n như sau:
- 5!=5*4!
- 4!=4*3!
- Tức là nếu ta biết được (n-1) giai thừa thì ta sẽ tính được n giai thừa, vì n!=n*(n-1)!
- Thấy n=0 hoặc n=1 thì giai thừa luôn = 1 ➔ chính là điểm dừng

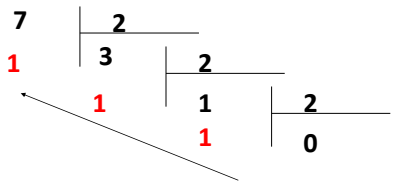
n!=1* 2 * 3 *…* (n-1) *n = (n-1)! *n (với 0!=1)

3. Đệ qui

```
int giaiThua(int n)
{
   if(n<=1)
     return(1);
   return n*giaiThua(n-1); // goi de qui
}
```

3. Đệ qui

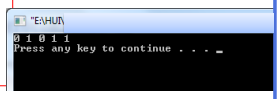
Ví dụ : Dùng đệ qui để chuyển đổi thừ hệ thập phân sang nhị phân.



3. Đệ qui

```
void H10toH2(int n)
{
 int t=n%2;
 if(n>0)
   H10toH2(n/2);
 cout<<t<<" ";
}
```

```
void main()
{
  H10toH2(11);
}
```



3. Đệ qui

***Phân loại đệ qui**

- *Đệ qui tuyến tính.
- *Đệ qui nhị phân.
- *Đệ qui phi tuyến.
- *Đệ qui hỗ tương.

3. Đệ qui tuyến tính

Trong thân hàm có duy nhất một lời gọi hàm gọi lại chính nó một cách tường minh.

<Kiểu dữ liệu hàm> **TenHam** (<danh sách tham số>)
{
if (điều kiện dừng)
{
...
//Trả về giá trị hay kết thúc công việc
}
//Thực hiện một số công việc (nếu có)
. . . **TenHam** (<danh sách tham số>);
//Thực hiện một số công việc (nếu có)
}

3. Đệ qui tuyến tính

Ví dụ:

Tính $S(n) = 1 + 2 + 3 + L + n$

- *Điều kiện dừng: S(0) = 0.*
- *Qui tắc (công thức) tính: S(n) = S(n-1) + n.*

```
long TongS (int n)
{
   if(n==0)
   return 0;
   return ( TongS(n-1) + n );
}
```

3.2. Đệ qui nhị phân

Trong thân của hàm có hai lời gọi hàm gọi lại chính nó một cách tường minh.

<Kiểu dữ liệu hàm> **TenHam** (<danh sách tham số>)
{
if (điều kiện dừng)
{
...
//Trả về giá trị hay kết thúc công việc
}
//Thực hiện một số công việc (nếu có)
. . .**TenHam** (<danh sách tham số>); //Giải quyết vấn đề nhỏ hơn
//Thực hiện một số công việc (nếu có)
. . . **TenHam** (<danh sách tham số>); //Giải quyết vấn đề còn lại
//Thực hiện một số công việc (nếu có)
}

3.2. Đệ qui nhị phân

Ví dụ: Tính số hạng thứ n của dãy Fibonaci được định nghĩa như sau:

$f_1 = f_0 = 1$; (n>1)

$f_n = f_{n-1} + f_{n-2}$;

Điều kiện dừng: f(0) = f(1) = 1.

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

3. 3. Đệ qui phi tuyến

Trong thân của hàm có lời gọi hàm gọi lại chính nó được đặt bên trong vòng lặp.

```
<Kiểu dữ liệu hàm> TenHam (<danh sách tham số>)
{
for (int i = 1; i<=n; i++)
{
//Thực hiện một số công việc (nếu có)
   if (điều kiện dừng)
{ ...
//Trả về giá trị hay kết thúc công việc
}
else
{ //Thực hiện một số công việc (nếu có)
TenHam (<danh sách tham số>);
}
}
}
```

3. 3. Đệ qui phi tuyến

Ví dụ: Tính số hạng thứ n của dãy {Xn} được định nghĩa như sau:

$X_0 = 1$;

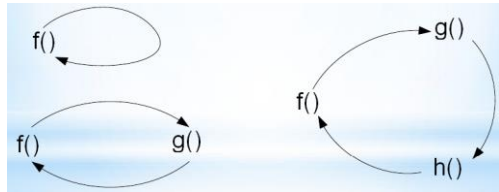
$Xn = n^2X_0 + (n-1)^2X_1 + \ldots + 1^2X_{n-1}$; (n≥1)

Điều kiện dừng:X(0) = 1.

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

3. 3. Đệ qui hỗ tương

Trong thân của hàm này có lời gọi hàm đến hàm kia và trong thân của hàm kia có lời gọi hàm tới hàm này.



3. 3. Đệ qui hỗ tương

```
<Kiểu dữ liệu hàm> TenHam2 (<danh sách tham số>);
<Kiểu dữ liệu hàm> TenHam1 (<danh sách tham số>)
{
    //Thực hiện một số công việc (nếu có)
    …TenHam2 (<danh sách tham số>);
    //Thực hiện một số công việc (nếu có)
}
<Kiểu dữ liệu hàm> TenHam2 (<danh sách tham số>)
{
    //Thực hiện một số công việc (nếu có)
    …TenHam1 (<danh sách tham số>);
    //Thực hiện một số công việc (nếu có)
}
```

3. 3. Đệ qui hỗ tương

Ví dụ: Tính số hạng thứ n của hai dãy {Xn}, {Yn} được định nghĩa như sau:

$X_0 = Y_0 = 1$;

$X_n = X_{n-1} + Y_{n-1}$; (n>0)

$Y_n = n^2X_{n-1} + Y_{n-1}$; (n>0)

- Điều kiện dừng:X(0) = Y(0) = 1.

```
long TinhYn(int n);
long TinhXn (int n)
{
    if(n==0)
    return 1;
    return TinhXn(n-1) + TinhYn(n-1);
    }
long TinhYn (int n)
{
if(n==0)
return 1;
return n*n*TinhXn(n-1) + TinhYn(n-1);

}
```

3. 4. Cách hoạt động của hàm đệ qui

Ví dụ tính n! với n=5