VIỆN ĐIỆN TỬ - VIỄN THÔNG

School of Electronics and telecommunications

KỸ THUẬT LẬP TRÌNH C/C++

HÀM

Giảng viên: TS. Nguyễn Thị Kim Thoa

Học kỳ: 20201

Nội dung

1. Hàm là gì?
2. Ưu nhược điểm của phân hoạch hàm
3. Khai báo và định nghĩa hàm
4. Các kiểu truyền hàm
5. Cơ chế trả về của hàm
6. Nạp chồng hàm
6. Hàm nội tuyến

@ Hàm là gì?

Hàm là:

- Một đơn vị tổ chức chương trình, một đoạn mã chương trình có cấu trúc để thực hiện một chức năng nhất định, có giá trị sử dụng lại
- Các hàm có quan hệ với nhau thông qua lời gọi, các biến tham số (đầu vào, đầu ra) và giá trị trả về
- Cách thực hiện cụ thể một hàm phụ thuộc nhiều vào dữ kiện (tham số, đối số của hàm):
- Thông thường, kết quả thực hiện hàm mỗi lần đều giống nhau nếu các tham số đầu vào như nhau
- Một hàm không có tham số thì giá trị sử dụng lại rất thấp
- Trong C/C++: Không phân biệt giữa thủ tục và hàm, cả đoạn mã chương trình chính cũng là hàm

Ví dụ về hàm

- Yêu cầu bài toán: Tính tổng một dãy số nguyên (liên tục) trong phạm vi do người sử dụng nhập. In kết quả ra màn hình.
- Các nhiệm vụ:
  - ✓ Nhập số nguyên thứ nhất:
    - Yêu cầu người sử dụng nhập
    - Nhập số vào một biến
  - ✓ Nhập số nguyên thứ hai
    - Yêu cầu người sử dụng nhập
    - Nhập số vào một biến
  - ✓ Tính tổng với vòng lặp
  - ✓ Hiển thị kết quả ra màn hình

Ví dụ: Cách 1

```
#include <iostream.h>
void main() {
      int a, b;
      char c;
      do {
            cout << “Nhap so nguyen thu nhat: ";
            cin >> a;
            cout << “Nhap so nguyen thu hai: ";
            cin >> b;
            int tong = 0;
            for (int i = a; i <= b; ++i)
              tong += i;
            cout << "Tong tu " << a << " den " << b
                 << " la " << tong << endl;
            cout << “ban co muon tinh tiep? (Y/N):";
            cin  >> c;
      } while (c == 'y' || c == 'Y');
}
```

Cách 2 (phân hoạch hàm)

```
#include <iostream.h>

int  Nhap();
int  Tinh_tong(int,int);
void Hien_thi(int a, int b, int kq);

void main() {
     char c;
     do {
          int a = Nhap();
          int b = Nhap();
          int T = Tinh_tong(a,b);
          Hien_thi(a,b,T);
          cout << “Ban co muon tinh tiep? (Y/N):";
          cin  >> c;
     } while (c == 'y' || c == 'Y');
}
```

Cách 2 (...)

```
int Nhap() {
      cout << “Nhap mot so nguyen: ";
      int N;
      cin >> N;
      return N;
}

int Tinh_tong(int a, int b) {
      int T = 0;
      for (int i = a; i <= b; ++i)
            T += i;
      return T;
}

void Hien_thi (int a, int b, int kq) {
      cout << "Tong tu " << a << " den " << b
            << " la " << kq << endl;
}
```

Ưu nhược điểm của phân hoạch hàm

- Chương trình dễ đọc hơn => dễ phát hiện lỗi
- Chương trình dễ mở rộng hơn
- Có giá trị sử dụng lại
- Mã nguồn dài hơn
- Mã chạy lớn hơn
- Chạy chậm hơn

=> Không phải cứ phân hoạch thành nhiều hàm là tốt, mà vấn đề nằm ở cách phân hoạch và thiết kế hàm làm sao cho tối ưu!

Khai báo và định nghĩa hàm

- Định nghĩa hàm: tạo mã thực thi hàm

Kiểu trả về — Tên hàm — Tham số (hình thức)

```
int Tinh_tong(int a, int b) {
    int T = 0;
    for (int i = a; i <= b; ++i)
        T += i;
    return T;
}
```

- Khai báo hàm thuần túy: không tạo mã hàm

```
int Tinh_tong (int a, int b);
```

Kiểu trả về — Tên hàm — Kiểu tham số

- Tại sao và khi nào cần khai báo hàm?

Khai báo hàm và lời gọi hàm

- Ý nghĩa của khai báo hàm:
  - ✓ Khi cần sử dụng hàm (gọi hàm)
  - ✓ Trình biên dịch cần lời khai báo hàm để kiểm tra lời gọi hàm đúng hay sai về cú pháp, về số lượng các tham số, kiểu các tham số và cách sử dụng giá trị trả về.

    int Tinh_tong(**int** a, **int** b);
  - ✓ Có thể khai báo hàm độc lập với việc định nghĩa hàm (tất nhiên phải đảm bảo nhất quán)
- Có thể định nghĩa hàm sau lời gọi hàm
- Gọi hàm: yêu cầu thực thi mã hàm với tham số thực tế (tham trị)

```
int x = 5;
int k = Tinh_tong(x, 10);
```

Tên hàm Đối số (thực tế)

Khai báo hàm C/C++ ở đâu?

- Ở phạm vi toàn cục (ngoài bất cứ hàm nào)
- Một hàm phải được khai báo trước lời gọi đầu tiên trong một tệp tin mã nguồn
- Nếu sử dụng nhiều hàm thì sẽ cần rất nhiều dòng mã khai báo (mất công viết, dễ sai và mã chương trình lớn lên?):
  - ✓ Nếu người xây dựng hàm (định nghĩa hàm) đưa sẵn tất cả phần khai báo vào trong một tệp tin => Header file (*.h, *.hx,...) thì người sử dụng chỉ cần bổ sung dòng lệnh

    #include <filename>
  - ✓ Mã chương trình không lớn lên, bởi khai báo không sinh mã!
- Một hàm có thể khai báo nhiều lần tùy ý!

Định nghĩa hàm ở đâu?

- Ở phạm vi toàn cục (ngoài bất cứ hàm nào)
- Có thể định nghĩa trong cùng tệp tin với mã chương trình chính, hoặc tách ra một tệp tin riêng. Trong Visual C++:

  *.c => C compiler,

  *.cpp => C++ compiler
- Một hàm đã có lời gọi thì phải được định nghĩa chính xác 1 lần trong toàn bộ (dự án) chương trình, trước khi gọi trình liên kết (lệnh Build trong Visual C++)
- Đưa tệp tin mã nguồn vào dự án, không nên:

  #include "xxx.cpp"
- Một hàm được định nghĩa sẵn bằng C, C++, hợp ngữ hoặc bằng một ngôn ngữ khác và dùng trong C/C++ => Sử dụng hàm không cần mã nguồn!
- Một thư viện cho C/C++ bao gồm:
  - ✓ Header file (thường đuôi *.h, *.hxx, ..., nhưng không bắt buộc)
  - ✓ Tệp tin mã nguồn (*.c, *.cpp, *.cxx,...) hoặc mã đích (*.obj, *.o, *.lib, *.dll, ...)

Tham số và đối số

- Cách gọi khác của tham số và đối số là tham biến và tham trị

```
int Tinh_tong (int a, int b) {
    ...
}
```

Tham số (hình thức)

```
int x = 5;
int k = Tinh_tong (x, 10);
...
```

Đối số (thực tế)

```
int a = 2;
k = Tinh_tong (a,x);
```

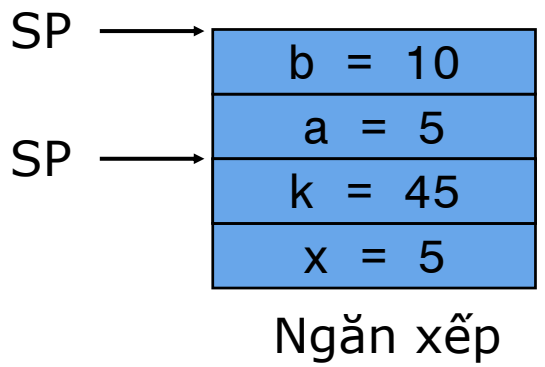
@ Truyền giá trị

```
int Tinh_tong (int, int);

void main() {
    int x = 5;
    int k = Tinh_tong(x, 10);
    ...
}

// Dinh nghia ham
int Tinh_tong(int a, int b) {
    ...


}
```



Ví dụ về truyền giá trị

```
#include <iostream.h>
void func(int N) {
   cin>>N; //nhap vao so 15
}

void main() {
   int x = 5;
   cout<< “Hay nhap so nguyen: ”;
   func(x);
   cout << “Bay gio x la " << x;
   ...
}
```

- Kết quả???

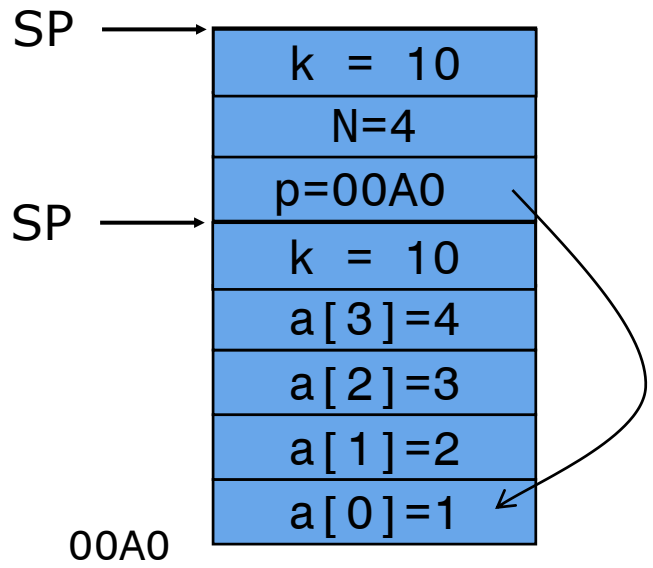
Truyền giá trị (...)

- Truyền giá trị là cách thông thường trong C
- Tham số chỉ nhận được bản sao của đối số (biến đầu vào)
- Thay đổi tham số chỉ làm thay đổi vùng nhớ cục bộ, không làm thay đổi đối số
- Tham số chỉ có thể mang đầu vào của hàm, không chứa được kết quả của hàm
- Truyền giá trị trong nhiều trường hợp kém hiệu quả do mất công sao chép dữ liệu

Truyền địa chỉ

```
int Tong_mang(int* p, int N);
    // Goi ham
void main() {
    int a[] = {1, 2, 3, 4};
    int k = Tong_mang(a,4);
    ...
}

// Dinh nghia ham
int Tong_mang(int* p, int N) {
    int k = 0;
    for (int i=0; i<N; i++)
          k += *p++;
    return k;   }
```



@ Truyền mảng tham số?

```
int Tong_mang(int p[4], int N);
    // Goi ham
void main() {
    int a[] = {1, 2, 3, 4};
    int k = Tong_mang(a,4);
    ...
}

// Dinh nghia ham
int Tong_mang(int p[4], int N) {
    int k = 0;
    for (int i=0; i<N; i++)
          k += *p++;
    return k;
}
```

Bản chất giống truyền địa chỉ

Thử lại ví dụ trước

```
#include <iostream.h>
void func(int* pN) {
   *pN=15;
}

void main() {
   int x = 5;
   cout<<“Nhap so nguyen:“;
   func(&x);
   cout << “Bay gio x la " << x;
   ...
}
```

- Kết quả ???

Khi nào sử dụng truyền địa chỉ(con trỏ)?

- Khi cần thay đổi "biến đầu vào" (truy nhập trực tiếp vào ô nhớ, không qua bản sao)
- Khi kích cỡ kiểu dữ liệu lớn => tránh sao chép dữ liệu vào ngăn xếp
- Truyền tham số là một mảng => bắt buộc truyền địa chỉ
- Lưu ý: Sử dụng con trỏ để truyền địa chỉ của vùng nhớ dữ liệu đầu vào. Bản thân con trỏ có thể thay đổi được trong hàm nhưng địa chỉ vùng nhớ không thay đổi (nội dung của vùng nhớ đó thay đổi được)

Truyền tham chiếu (C++)

```
#include <iostream.h>
void func(int& N) {
   N=15;
}

void main() {
   int x = 5;
   cout<<“Nhap so nguyen:“;
   func(x);
   cout << “Bay gio x la " << x;
   ...
}
```

- Kết quả: ???

Ví dụ hàm hoán vị

- Viết một hàm hoán đổi giá trị của hai phần tử nguyên, sau đó viết chương trình ứng dụng hàm đó.

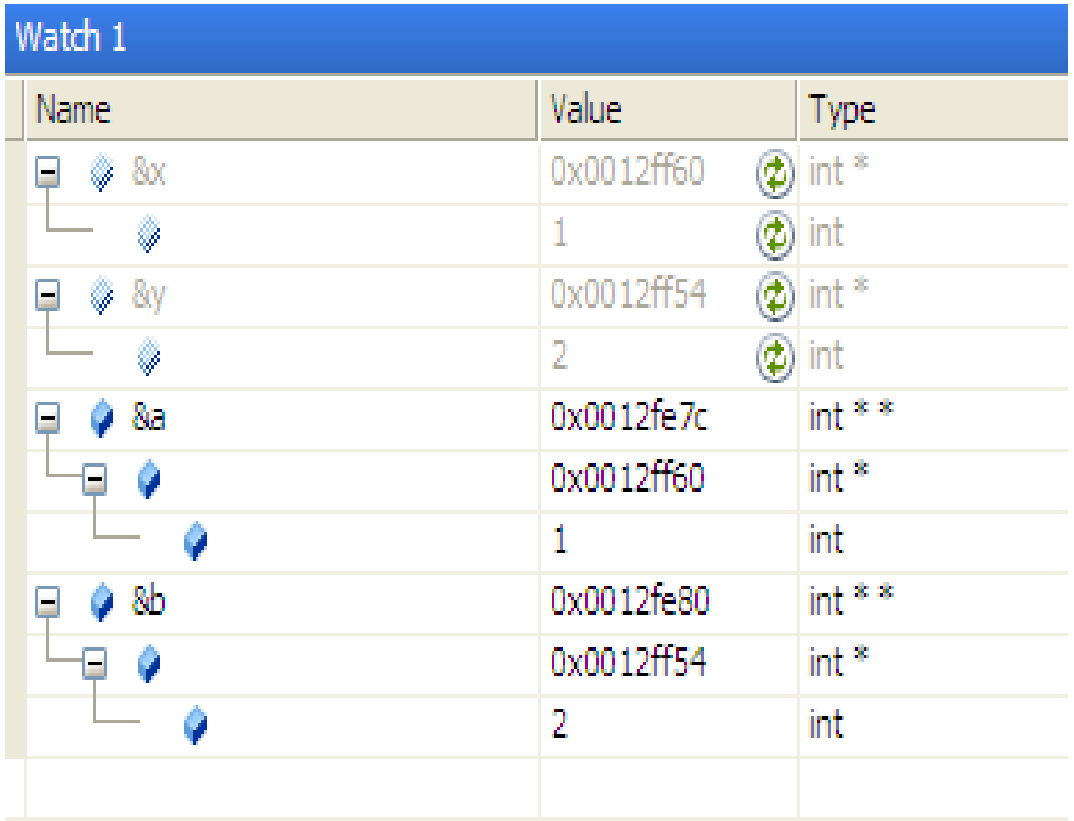
```
#include <iostream.h>
void swap(int& a, int& b) {
   int temp = a;
   a = b;
   b = temp;
}
void main() {
   int x, y;
   cout << “nhap vao 2 so nguyen”;
   cin >>x >>y;
   swap(x,y);
   cout << "Now x is " << x << ", y is " << y;
}
```

- Còn cách nào giải quyết bài toán trên?

Ví dụ hàm swap

- Hàm truyền địa chỉ

```
#include <iostream>
using namespace std;
void swap1(int* a, int* b) {
        int temp = *a;
        *a = *b;
        *b = temp;
}
void main() {
        int x, y;
        cout << "nhap vao 2 so nguyen";
        cin >>x >>y;
        swap1(&x,&y);
        cout << "Now x is " << x << ", y
is " << y;
}
```



Khi nào sử dụng truyền tham chiếu?

- Truyền tham chiếu chỉ dùng trong C++
- Khi cần thay đổi "biến đầu vào" (truy nhập trực tiếp vào ô nhớ, không qua bản sao)
- Một tham số tham chiếu có thể đóng vai trò là đầu ra (chứa kết quả), hoặc có thể vừa là đầu vào và đầu ra
- Khi kích cỡ kiểu dữ liệu lớn => tránh sao chép dữ liệu vào ngăn xếp.

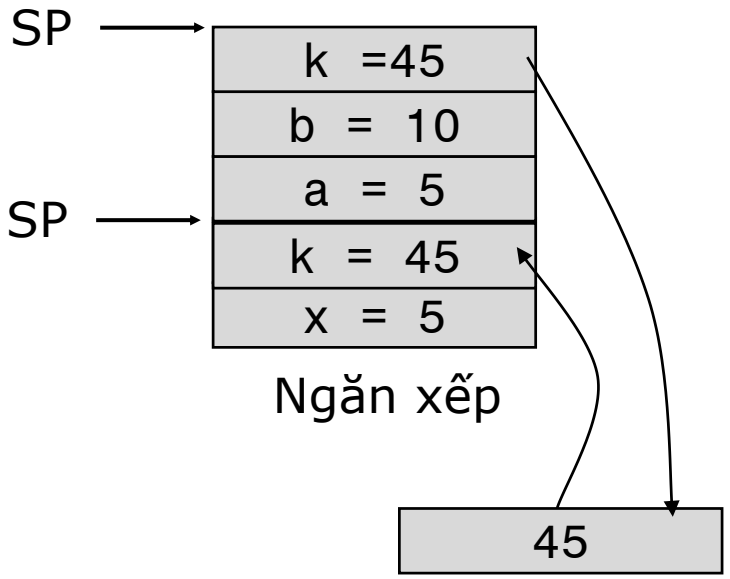
Kiểu trả về

- Kiểu trả về gần như tùy ý.
- Không thể trả về trực tiếp một mảng
- Có thể trả về kiểu:
  - ✓ Giá trị
  - ✓ Con trỏ (địa chỉ)
  - ✓ Tham chiếu
- Tuy nhiên, cần rất thận trọng với trả về địa chỉ hoặc tham chiếu:
  - ✓ Không bao giờ trả về con trỏ hoặc tham chiếu vào biến cục bộ
  - ✓ Không bao giờ trả về con trỏ hoặc tham chiếu vào tham số truyền qua giá trị

Cơ chế trả về

```
int Tinh_tong(int a, int b) {
    int k = 0;
    for (int i=a; i <= b; ++i)
          k +=i;
    return k;
}

void main() {
    int x = 5, k = 0;
    k = Tinh_tong(x,10);
    ...
}
```



Trả về con trỏ

- Viết hàm tìm giá trị lớn nhất của một mảng?
- Viết hàm trả về địa chỉ của phần tử lớn nhất trong một mảng và giá trị của phần tử đó?
- Viết chương trình ứng dụng các hàm trên.

```
//Hàm trả về giá trị lớn nhất của mảng
void Max(int* p, int n, int& max) {
   int *p2 = p + n; max=*p;
   while (p < p2) {
        if (*p > max)
              max = *p;
        ++p;
   }
}
```

```
//Hàm trả về giá trị lớn nhất của mảng và
   địa chỉ của phần tử lớn nhất đó
int* Max(int* p, int n, int& max)
   {
   int *pMax = p; int *p2 = p + n;
   while (p < p2) {
        if (*p > *pMax)
              pMax = p;
        ++p;
   }
   max=*pMax;
   return pMax;
}
void main() {
   int s[5] = { 1, 2, 3, 4, 5};
   int *p = Max(s,5);
}
```

Lý do trả về con trỏ hoặc tham chiếu

- Tương tự như lý do truyền địa chỉ hoặc truyền tham chiếu:
- Tránh sao chép dữ liệu lớn không cần thiết
- Để có thể truy cập trực tiếp và thay đổi giá trị đầu ra
- Có thể trả về con trỏ hoặc tham chiếu vào đâu?
- Vào biến toàn cục
- Vào tham số truyền cho hàm qua địa chỉ hoặc qua tham chiếu
- Nói chung: vào vùng nhớ mà còn tiếp tục tồn tại sau khi kết thúc hàm

Tham số mặc định của hàm

- VD:

```
void func(int a, int b=0, int c=0)
{ ……  }
void main()
{
   int x=4, y=5, z=6;
   func(x,y,z);//OK
   func(x,y);//OK
   func(x);//OK
}
```

Tham số mặc định của hàm (...)

- Các tham số mặc phải nằm ở cuối cùng của dãy tham số (tính từ trái sang phải)
- Tham số thiếu vắng trong lời gọi hàm sẽ tương ứng với các tham số mặc định cuối cùng (tính từ trái sang phải)
- Đối với một hàm có tiền khai báo và định nghĩa hàm, đối số mặc định có thể được khai báo ở một trong hai, nhưng không phải cả hai.

```
int sum(int a, int b, int c = 0);
int sum(int a, int b, int c)
{
    return a + b + c;
}
```

Hoặc

```
int sum(int a, int b, int c);
int sum(int a, int b, int c = 0)
{
    return a + b + c;
}
```

Ví dụ tìm min, max của mảng

- Viết một hàm đồng thời tìm giá trị lớn nhất và giá trị nhỏ nhất của một mảng. Viết một chương trình ứng dụng hàm đó?

Ví dụ tìm min, max của mảng

```
#include <iostream>
using namespace std;
void min_max(int *p, int N, int& max, int& min)
 {
        int* p2=p+N;
        max=p[0];
        min = p[0];
        while (p<p2){
                if (max<*p)
                        max=*p;
                if(min>*p)
                        min = *p;
                p++;
        }
}
```

```
void main()
{
        //const int N=7;
        int a[]={1,8,9,5,7,99,-9};
        int min,max;
        min_max(a, 7, max, min);
        cout<<"min = "<<min<<" max =
"<<max<<endl;
}
```

Nạp chồng tên hàm trong C++

- Trong C++ có thể xây dựng nhiều hàm có cùng tên, ví dụ:

```
int    max(int a, int b);
double max(double a, double b);
double max(double a, double b, double c);
double max(double *seq, int n);
```

- Mục đích của nạp chồng tên hàm:
  - ✓ Đơn giản hóa cho người xây dựng hàm trong việc chọn tên (thay vì maxInt, maxDouble, maxDouble3, maxDoubleSequence,...)
  - ✓ Đơn giản hóa cho người sử dụng hàm, chỉ cần nhớ 1 tên quen thuộc thay cho nhiều tên phức tạp

Ví dụ: định nghĩa các hàm max()

```
int max(int a, int b) {                          // (1)
   return (a > b)? a : b;
}
double max(double a, double b) {                 // (2)
   return (a > b)? a : b;
}
double max(double a, double b, double c); {      // (3)
   if (a < b) a = b;
   if (a < c) a = c;
   return a;
}
double max(double *seq, int n) {                 // (4)
   int i = 0, kq = seq[0];
   while (i < n) {
     if (kq < seq[i])kq = seq[i];
     ++i;
   }
   return kq;
}
```

Ví dụ: sử dụng các hàm max()

```
int max(int a, int b);                          // (1)
double max(double a, double b);                 // (2)
double max(double a, double b, double c);       // (3)
double max(double *seq, int n);                 // (4)

void main() {
   int k = max(5,7);                // call?
   double d = max(5.0,7.0);         // call?
   double a[] = {1,2,3,4,5,6};
   d = max(d, a[1], a[2]);          // call?
   d = max(a, 5);                   // call?
   d = max(5,7);                    // call?
   d = max(d, 5);                   // call?
}
```

- Đẩy trách nhiệm kiểm tra và tìm hàm phù hợp cho compiler!

Một số qui tắc về nạp chồng tên hàm

- Các hàm cùng tên được định nghĩa cùng trong một file/ trong một thư viện hoặc sử dụng trong cùng một chương trình phải khác nhau ít nhất về:
  - ✓ Số lượng các tham số, hoặc
  - ✓ Kiểu của ít nhất một tham số (int khác short, const int khác int, int khác int&, ...)
- Không thể chỉ khác nhau ở kiểu trả về
- Tại sao vậy?
  - ✓ Compiler cần có cơ sở để quyết định gọi hàm nào
  - ✓ Dựa vào cú pháp trong lời gọi (số lượng và kiểu các tham số thực tế) compiler sẽ chọn hàm có cú pháp phù hợp nhất
  - ✓ Khi cần compiler có thể tự động chuyển đổi kiểu theo chiều hướng hợp lý nhất (vd short=>int, int => double)

Một số qui tắc về nạp chồng tên hàm

- Hàm có đối số mặc định có thể được nạp chồng.

Ví dụ:

```
void print(int a);
void print(double a = 0);
```

- Các tham số có đối số mặc định không được sử dụng để xác định tính duy nhất trong nạp chồng hàm.

Ví dụ:

```
void print(int a);
void print(int a, int b = 0); // lỗi
```

Hàm nội tuyến (inline) trong C++

- Vấn đề: Hàm tiện dụng, nhưng nhiều khi hiệu suất không cao, đặc biệt khi mã thực thi hàm ngắn vì
  - ✓ Các thủ tục như nhớ lại trạng thái chương trình, cấp phát bộ nhớ ngăn xếp, sao chép tham số, sao chép giá trị trả về, khôi phục trạng thái chương trình mất nhiều thời gian
  - ✓ Nếu mã thực thi hàm ngắn thì sự tiện dụng không bõ so với sự lãng phí thời gian

Dùng hàm inline trong C++

- Điều duy nhất cần làm là thêm từ khóa inline vào đầu dòng khai báo và định nghĩa hàm

```
inline int max(int a, int b) {
    return (a > b)? a : b;
}
```

- Hàm inline khác gì hàm bình thường?
  - ✓ "Hàm inline" thực chất không phải là một hàm!
  - ✓ Khi gọi hàm thì lời gọi hàm được thay thế một cách thông minh bởi mã nguồn định nghĩa hàm, không thực hiện các thủ tục gọi hàm

Khi nào nên dùng hàm inline

- Ưu điểm của hàm inline:
  - ✓ Tiện dụng như hàm bình thường
  - ✓ Hiệu suất như viết thẳng mã, không gọi hàm
  - ✓ Tin cậy, an toàn.
- Nhược điểm của hàm inline:
  - ✓ Nếu gọi hàm nhiều lần trong chương trình, mã chương trình có thể lớn lên nhiều (mã thực hiện hàm xuất hiện nhiều lần trong chương trình)
  - ✓ Mã định nghĩa hàm phải để mở => đưa trong header file
- Lựa chọn xây dựng và sử dụng hàm inline khi:
  - ✓ Mã định nghĩa hàm nhỏ (một vài dòng lệnh, không chứa vòng lặp)
  - ✓ Yêu cầu về tốc độ đặt ra trước dung lượng bộ nhớ

Ví dụ

- Yêu cầu bài toán: Tính tổng một dãy số nguyên (liên tục) trong phạm vi do người sử dụng nhập. In kết quả ra màn hình. (sử dụng hàm inline)