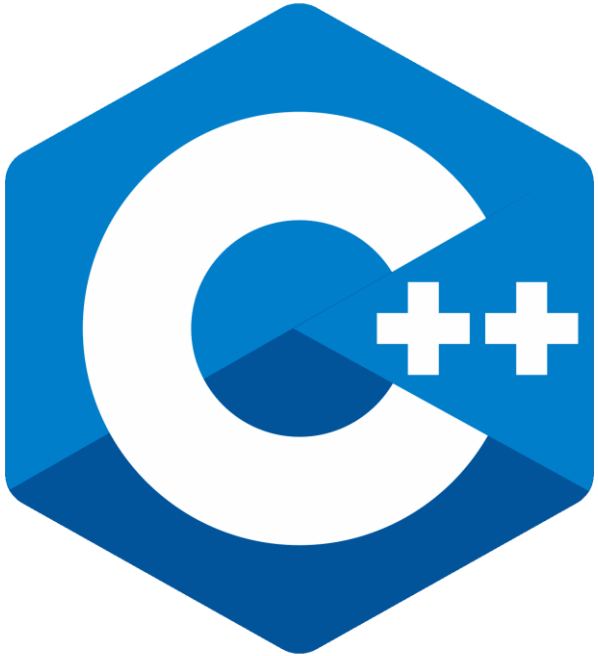
KỸ THUẬT LẬP TRÌNH

TRẦN ĐÌNH LUYỆN

Hàm

Cung cấp các kiến thức hàm và lập trình hướng hàm: khai báo, định nghĩa, truyền tham số và trả về kết quả, nạp chồng tên hàm,...



Định nghĩa

- Function (hàm, chức năng): 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

- Viết chương trình tính tổng một dãy số nguyên (liên tục) trong phạm vi từ a đến b (a, b do người dung nhập) và in kết quả ra màn hình.
- Phân tích các công việc cần làm
  - Nhập số thứ nhất
    - Thông báo nhập
    - Nhập số và đưa vào biến a
  - Nhập số thứ hai
    - Thông báo nhập
    - Nhập số và đưa vào biến b
  - Tính tổng
  - In kết quả ra màn hình

Mã nguồn

```
#include <iostream>
using namespace std;
int main()
{
    int a, b;
    cout << "Nhap so thu nhat: ";
    cin >> a;
    cout << "Nhap so thu hai: ";
    cin >> b;
    int tong = 0;
    for (int i=a;i<= b;++i)
        tong += i;
    cout << a << " -> " << b <<" = " << tong << endl;
   return 0;
}
```

Mã nguồn

```
#include <iostream>
using namespace std;
int main(){
    int a, b;char c;
    do{
        cout << "Nhap so thu nhat: ";
        cin >> a;
        cout << "Nhap so thu hai: ";
        cin >> b;
        int tong = 0;
        for (int i=a;i<= b;++i) tong += i;
        cout << a << " -> " << b <<" = " << tong << endl;
        cout << "Tiep tuc (Y/N)? ";
        cin >> c;
    }while (c == 'Y' || c == 'y');
   return(0);
}
```

Phân chia hàm

```
#include <iostream>
using namespace std;
int NhapSo();
int TinhTong(int, int);
void InKetQua(int a, int b, int tong);
int main(){
    char c;
    do{
        int a=NhapSo();
        int b=NhapSo();
        int tong=TinhTong(a,b);
        InKetQua(a,b,tong);
        cout << "Tiep tuc (Y/N)? ";
        cin >> c;
      }while (c == 'Y' || c == 'y');
    return 0;
}
```

Phân chia hàm

```
int NhapSo(){
    cout << "Nhap vao so nguyen: ";
    int n;
    cin >> n;
    return n;
}
int TinhTong(int a,int b){
    int tong = 0;
    for (int i=a;i<=b;++i)
        tong += i;
    return tong;
}
void InKetQua(int a,int b,int tong){
    cout << a << " + " << b << " = " << tong << endl;
}
```

Không có tham số. Sử dụng lại?

Ok, tốt

Nhiều tham số, hiệu năng?

Lợi ích phân chia hàm

- Chương trình dễ đọc hơn => dễ phát hiện lỗi
- Chương trình dễ mở rộng hơn
- Hàm **TinhTong** có thể sử dụng lại tốt
- Mã nguồn dài hơn
- Mã chạy lớn hơn
- Chạy chậm hơn
- Không phải cứ phân hoạch thành nhiều hàm là tốt, vấn đề nằm ở cách phân hoạch và thiết kế hàm làm sao cho tối ưu!

Phân chia hàm

```
#include <iostream>
using namespace std;
int NhapSo(const char*);
int TinhTong(int, int);
int main(){
    char c;
    do{
        int a=NhapSo("Nhap so thu nhat: ");
        int b=NhapSo("Nhap so thu hai: ");
        int tong=TinhTong(a,b);
        cout << a << " + " << b << " = " << tong << endl;
        cout << "Tiep tuc (Y/N)? ";
        cin >> c;
        cout << c;
    }while (c == 'Y' || c == 'y');
    return 0;
}
```

Phân chia hàm

```
int NhapSo(const char* noiDungHienThi){
    cout << noiDungHienThi;
    int n;
    cin >> n;
    return n;
}
int TinhTong(int a,int b){
    int tong = 0;
    for (int i=a;i<=b;++i)
        tong += i;
    return tong;
}
```

Khai báo và định nghĩa hàm

- Khai báo hàm: không tạo mã hàm

```
int TinhTong(int, int);
```

- Định nghĩa hàm: tạo mã thực thi hàm

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

- Khi nào cần khai báo hàm và tại sao?

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 TinhTong(int a, int b);
  - Có thể khai báo hàm độc lập với việc định nghĩa hàm (phải đảm bảo nhất quán)
- Gọi hàm: yêu cầu thực thi mã hàm với tham số thực tế

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

Nơi khai báo hàm

- Khai báo 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 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!

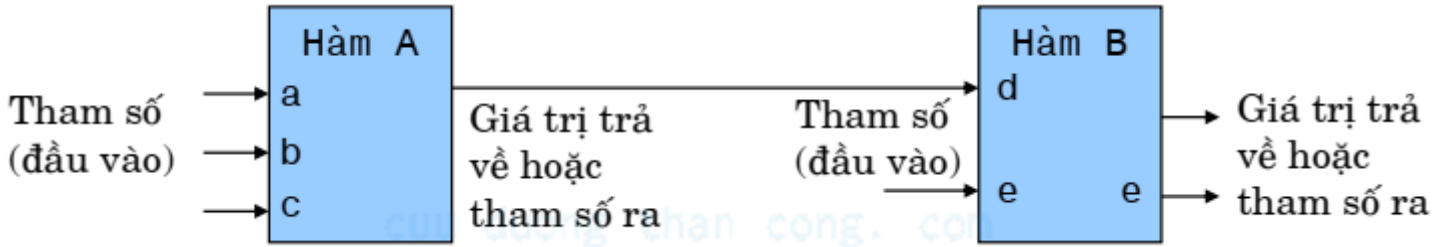
Nơi khai báo hàm

- Có thể định nghĩa trong cùng tệp tin với mã chương trình chính, hoặc tách ra một tệp tin riêng. (*.c, *.cpp)
- 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)
- Đưa tệp tin mã nguồn vào dự án: #include "xxx.cpp"
- Một thư viện cho C/C++ bao gồm:
  - Header file (*.h, *.hxx, ...,)
  - Tệp tin mã nguồn (*.c, *.cpp, *.cxx,...) hoặc mã đích (*.obj, *.o, *.lib, *.dll, ...)

Truyền tham số và trả về kết quả

- Truyền tham số và trả về kết quả là phương pháp cơ bản để tổ chức quan hệ giữa các hàm (giữa các chức năng trong hệ thống)
- Ngoài ra, còn có các cách khác:
  - Sử dụng biến toàn cục: nói chung là không nên!
  - Sử dụng các tệp tin, streams: dù sao vẫn phải sử dụng tham số để nói rõ tệp tin nào, streams nào
  - Các cơ chế giao tiếp hệ thống khác => vẫn cần các tham số bổ sung
- Truyền tham số & trả về kết quả là một vấn đề cốt lõi trong xây dựng và sử dụng hàm, một trong những yếu tố ảnh hưởng quyết định tới chất lượng phần mềm!

Truyền tham số và trả về kết quả

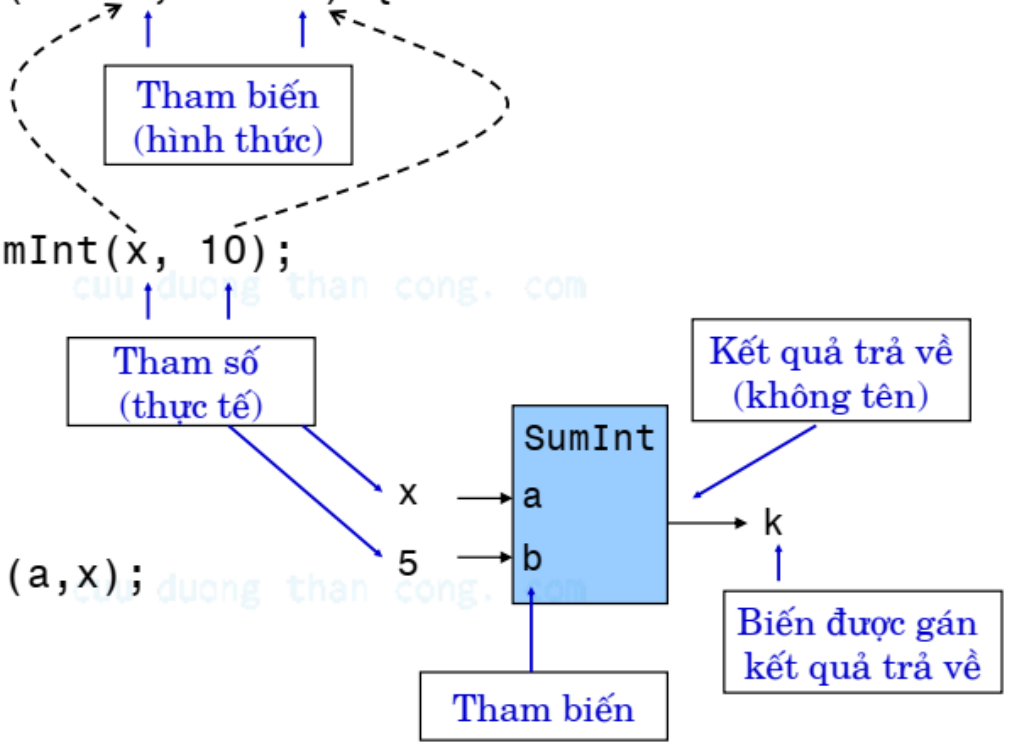


Tham số thực tế

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

int x = 5;
int k = SumInt(x, 10);
...

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



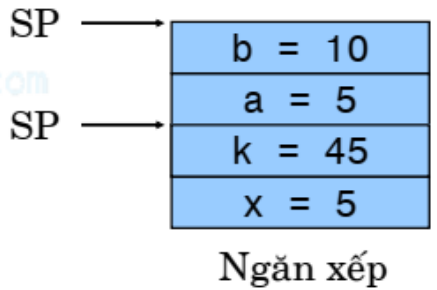
Truyền giá trị

```
int SumInt(int, int);

// Function call
void main() {
    int x = 5;
    int k = SumInt(x, 10);
    ...
}

// Function definition
int SumInt(int a, int b) {
    ...


}
```



Ngăn xếp

Ví dụ

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   void NhapSoNguyen(const char* userPrompt, int N) {
4       cout << userPrompt;
5       cin >> N;
6   }
7   int main() {
8       int x = 5;
9       NhapSoNguyen("Nhap mot so nguyen:", x);
10      cout << "Gia tri cua x la: " << x;
11      return 0;
12  }
```

Kết quả x như thế nào?

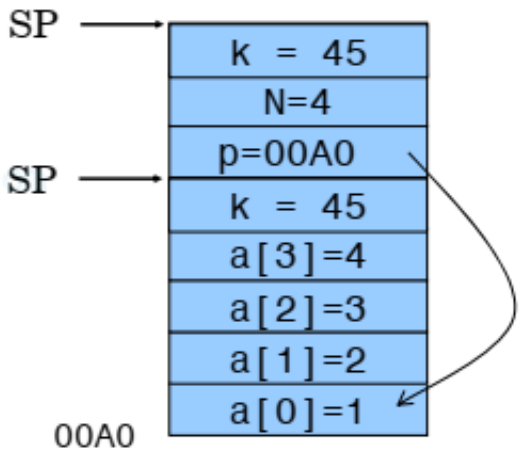
Truyền giá trị

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

Truyền địa chỉ

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

// Function definition
int SumInt(int* p, int N) {
    int *p2 = p + N, k = 0;
    while (p < p2)
        k += *p++;
    return k;
}
```



Truyền mảng tham số

```
1  #include <bits/stdc++.h>
2  #define fu(i,n) for (int i=0;i<n;i++)
3  using namespace std;
4  int SumInt(int p[4], int N);
5  void main() {
6      int a[] = {1, 2, 3, 4};
7      int k = SumInt(a,4);
8      return 0;
9  }
10 int SumInt(int p[4], int N) {
11     int *p2 = p + N, k = 0;
12     while (p < p2)
13     k += *p++;
14     return k;
15 }
```

Bản chất là truyền địa chỉ như slide trước.

Ví dụ

```
#include <bits/stdc++.h>
#define fu(i,n) for (int i=0;i<n;i++)
using namespace std;
int NhapSoNguyen(const char* userPrompt, int* N){
    cout << userPrompt;
    cin >> *N;
}
int main() {
    int x = 5;
    NhapSoNguyen("Nhap mot so nguyen: ", &x);
    cout << "Gia tri cua x la: " << x;
    return 0;
}
```

Kết quả x như thế nào?

Khi nào dùng địa chỉ?

- 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)

Kiểu tham chiếu (C++)

- Một biến tham chiếu là một biến đại diện trực tiếp cho một biến khác (thay cho con trỏ)
- Sử dụng chủ yếu về sau trong truyền tham số cho hàm *(sẽ đề cập kỹ hơn trong phần hàm)*

```
int main()
{
    double d = 2.0;
    double& r = d; // r represents d
    double *p1 = &d, *p2 = &r;
    r = 1.0; // OK, d = 1.0
    double& r2; // error, r has to be assigned to a var.
    double& r3 = 0; // error, too
    double d2 = 0;
    r = d2; // r = 0, d=0
    r = 1.0; // r = d = 1, d2 =0
    return 0;
}
```

Truyền tham chiếu (C++)

```
#include <bits/stdc++.h>
#define fu(i,n) for (int i=0;i<n;i++)
using namespace std;
int NhapSoNguyen(const char* userPrompt, int& N){
    cout << userPrompt;
    cin >> N;
}
int main() {
    int x = 5;
    NhapSoNguyen("Nhap mot so nguyen: ", x);
    cout << "Gia tri cua x la: " << x;
    return 0;
}
```

Ví dụ hoán vị 2 số - con trỏ

```
#include <iostream>
using namespace std;
void swap(int* a, int* b) {
	int temp = *a;
	*a = *b;
	*b = temp;
}

int main()
{
	int x = 5, y = 10;
	swap(&x,&y);
	cout << "Now x is " << x << ", y is " << y;
	return 0;
}
```

Ví dụ hoán vị 2 số - tham chiếu

```
#include <iostream>
using namespace std;
void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

int main()
{
    int x = 5, y = 10;
    swap(x,y);
    cout << "Now x is " << x << ", y is " << y;
    return 0;
}
```

Khi nào sử dụng?

- Chỉ trong C++
- Khi cần thay đổi "biến đầu vào" (truy nhập trực tiếp vào ô nhớ, không qua bản sao)
- Một tham biến 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

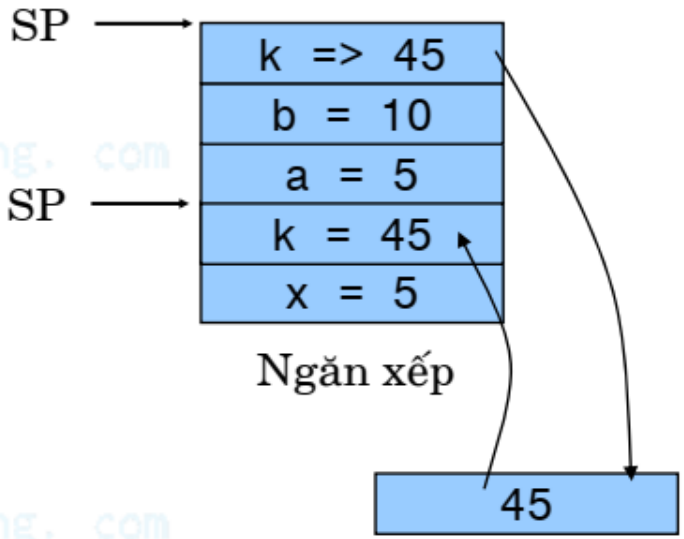
Kiểu trả về của hàm

- Kiểu trả về: gần như tùy ý (**không trả về một mảng**)
- Có thể trả về kiểu:
  - Giá trị
  - Con trỏ
  - Tham chiếu
- Tuy nhiên, cần rất thận trọng với trả về địa chỉ hoặc tham chiế
  - 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 biến truyền qua giá trị
- Với người lập trình ít có kinh nghiệm: chỉ nên trả về kiểu giá trị

Ví dụ

```
int SumInt(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 = SumInt(x,10);
    ...
}
```



Hàm trả về con trỏ (tìm phần tử lớn nhất)

```
#include <iostream>
using namespace std;
int* FindMax(int* p, int n) {
    int *pMax = p;
    int *p2 = p + n;
    while (p < p2) {
        if (*p > *pMax)
        pMax = p;
        ++p;
    }
    return pMax;
}
int main() {
    int s[5] = { 1, 2, 3, 4, 5};
    int *p = FindMax(s,5);
    return 0;
}
```

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?
  - Biến toàn cục
  - Tham số truyền cho hàm qua địa chỉ hoặc qua tham chiếu
  - Vào vùng nhớ mà còn tiếp tục tồn tại sau khi kết thúc hàm

Ví dụ trả về con trỏ


```
int* FindMax2(int* p, int n) {
    int Max = *p;
    int *p2 = p + n;
    while (p < p2) {
        if (*p > Max)
        Max = *p;
        ++p;
    }
    return &Max;
}
void main() {
    int s[5] = { 1, 2, 3, 4, 5};
    int *p = FindMax2(s,5);
}
```

Hàm và thư viện

- Xây dựng một chương trình chạy tốt đã khó, viết một thư viện hàm tốt còn khó hơn!
- Một thư viện hàm định nghĩa:
  - một tập hợp các hàm (có liên quan theo một chủ đề chức năng)
  - những kiểu dữ liệu sử dụng trong các hàm
  - một số biến toàn cục (rất hạn chế)
- Một thư viện hàm tốt cần phải:
  - Thực hiện những chức năng hữu ích
  - Đơn giản, dễ sử dụng
  - Hiệu suất và độ tin cậy cao
  - Trọn vẹn, nhất quán và đồng bộ

Thiết kế hàm

- Phân tích yêu cầu: làm rõ các dữ kiện (đầu vào) và kết quả (đầu ra); tìm ra các chức năng cần thực hiện
- Đặt tên hàm: ngắn gọn, ý nghĩa, tự miêu tả
  - Hàm chỉ hành động: chọn tên hàm là một động từ kết hợp với kiểu đối tượng, ví dụ **printVector**, **displayMatrix**, **addComplex**, **sortEventQueue**, **filterAnalogSignal**,...
  - Hàm truy nhập thuộc tính: Có thể chọn là động từ hoặc danh từ kết hợp kiểu đối tượng chủ thể, ví dụ **length**, **size**, **numberOfColums**, **getMatrixElem**, **putShapeColor**
- Trong C++ nhiều hàm có thể giống tên (nạp chồng tên hàm), có thể chọn tên ngắn, ví dụ **sort**, **print**, **display**, **add**, **putColor**, **getColor**

Thiết kế hàm

- Trong C++ còn có thể định nghĩa hàm toán tử để có thể sử dụng các ký hiệu toán tử định nghĩa sẵn như *, /, +, - thay cho lời gọi hàm.
- Tham số đầu vào (=> tham biến)
  - Đặc tả ý nghĩa: Thể hiện rõ vai trò tham số
  - Đặt tên: Ngắn gọn, tự mô tả
  - Chọn kiểu: Kiểu **nhỏ nhất** mà **đủ** biểu diễn
  - Chọn cách truyền tham số: cân nhắc giữa truyền giá trị hay truyền địa chỉ/tham chiếu vào kiểu hằng
- Tham số đầu ra (=> tham biến truyền qua địa chỉ/qua tham chiếu hoặc sử dụng giá trị trả về): đặc tả ý nghĩa, đặt tên, chọn kiểu tương tự như tham số đầu vào

Thiết kế hàm

- Định nghĩa bổ sung các kiểu dữ liệu mới như cần thiết
- Mô tả rõ tiền trạng (pre-condition): điều kiện biên cho các tham số đầu vào và các điều kiện ngoại cảnh cho việc gọi hàm
- Mô tả rõ hậu trạng (post-condition): tác động của việc sử dụng hàm tới ngoại cảnh, các thao tác bắt buộc sau này,...
- Thiết kế thân hàm dựa vào các chức năng đã phân tích, sử dụng lưu đồ thuật toán với các cấu trúc điều kiện/rẽ nhánh (vòng lặp) => có thể phân chia thành các hàm con nếu cần

Ví dụ-Hàm tìm N số nguyên tố đầu tiên

- Phân tích:
  - Dữ kiện: N - số số nguyên tố đầu tiên cần tìm
  - Kết quả: Một dãy N số nguyên tố đầu tiên
  - Các chức năng cần thực hiện:
    - Nhập dữ liệu? CÓ/KHÔNG?
    - Kiểm tra dữ kiện vào (N)? Có/không (Nếu kiểm tra mà N nhỏ hơn 0 thì hàm làm gì?)
    - Cho biết k số nguyên tố đầu tiên, xác định số nguyên tố tiếp theo NHƯ THẾ NÀO?
    - Lưu trữ kết quả mỗi lần tìm ra vào một cấu trúc dữ liệu phù hợp (dãy số cần tìm)
    - In kết quả ra màn hình? CÓ/KHÔNG?

Ví dụ-Hàm tìm N số nguyên tố đầu tiên

- Đặt tên hàm: **findPrimeSequence**
- Tham số vào: **1**
  - Ý nghĩa: số các số nguyên tố cần tìm
  - Tên: **N**
  - Kiểu: số nguyên đủ lớn (**int/long**)
  - Truyền tham số: qua giá trị
- Tham số ra: **1**
  - Ý nghĩa: dãy **N** số nguyên tố đầu tiên tính từ **1**
  - Giá trị trả về hay tham biến? **Tham biến**!
  - Tên: **primes**
  - Kiểu: mảng số nguyên (của **int/long**)
  - Truyền tham số: qua địa chỉ (**int*** hoặc **long***)

Ví dụ-Hàm tìm N số nguyên tố đầu tiên

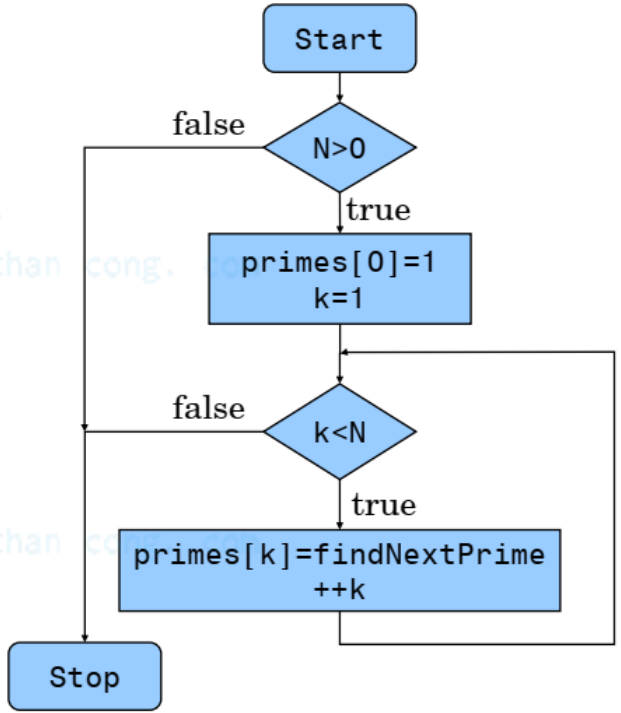
- Tiền trạng:
  - Tham số **N** phải là số không âm (có nên chọn kiểu **unsigned**?)
  - **primes** phải mang dịa chỉ của mảng số nguyên có ít nhất **N** phần tử

Ví dụ-Hàm tìm N số nguyên tố đầu tiên

- Khai báo hàm:

```
void findPrimeSequence(int N, int* primes);
```

- Thiết kế thân hàm
  - Lưu đồ thuật toán như hình
  - Phân chia, bổ sung một hàm mới: **findNextPrime**
- Lặp lại qui trình thiết kế hàm cho **findNextPrime**



Thư viện chuẩn

- Thư viện vào/ra (nhập/xuất) **<stdio.h>**
- Xử lý ký tự và chuỗi ký tự **<string.h>, <ctype.h>**
- Thư viện hàm toán **<math.h>, <float.h>**
- Thời gian, ngày tháng **<time.h>, <locale.h>**
- Cấp phát bộ nhớ động **<stdlib.h>**
- Các hàm ký tự rộng **<wchar.h>,<wctype.h>**
- Các hàm khác **<stdlib.h>, ..**

Nạp chồng tên hàm

- 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:
  - 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

Nạp chồng tên hàm - Ví dụ

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

Nạp chồng tên hàm - 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);
int main() {
    int k = max(5,7);
    double d = max(5.0,7.0);
    double a[] = {1,2,3,4,5,6};
    d = max(d, a[1], a[2]);
    d = max(a, 5);
    d = max(5,7);
    d = max(d, 5);
    return 0;
}
```

Kiểm tra và tìm hàm phù hợp do compiler thực hiện

Một số quy tắc

- 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)*
- Why?
  - 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)

Tìm hiểu thêm

Hàm inline

Q&A