NGÔN NGỮ LẬP TRÌNH C++

NGÀY 2.1: Con trỏ & hàm

Giảng viên: Th.S Bùi Văn Kiên

Nội dung

- Con trỏ và tham chiếu
  - Biến con trỏ
  - Biến tham chiếu
- Hàm
  - Hàm chuẩn
  - Hàm tự định nghĩa
  - Các bước xây dựng hàm
  - Tham số và đối số
  - Con trỏ hàm
  - Hàm đệ quy
  - Lambda expression

1.1 Con trỏ

Khái niệm

- Con trỏ là một biến đặc biệt, nó chứa địa chỉ của một biến khác. Con trỏ có kiểu là kiểu của biến mà nó trỏ tới

  <Kiểu dữ liệu> *<tên con trỏ>;

- Lấy địa chỉ con trỏ

  <Tên con trỏ> = &<tên biến>

- Lấy giá trị con trỏ

  *<Tên con trỏ>

- Phép gán con trỏ

  <Tên con trỏ1> = <Tên con trỏ2>

1.1 Con trỏ

Toán tử địa chỉ &

cho phép chúng ta xem địa chỉ bộ nhớ nào được gán cho một biến.

```
int x = 10;
cout << x << '\n'; // print the value of variable x
cout << &x << '\n'; // print the memory address of variable x
```

Lưu ý: Toán tử địa chỉ (&) tương tự như toán tử bitwise (&) a&b, nhưng toán tử địa chỉ (&) là toán tử 1 ngôi, trong khi toán tử bitwise (&) là 2 ngôi.

1.1 Con trỏ

Toán tử trỏ đến (*): dereference operator

- Cho phép chúng ta truy cập (get/set) giá trị tại một địa chỉ cụ thể.
- Ví dụ:

```
int x = 10;
cout << x << '\n';      // print the value of variable x
cout << &x << '\n';     // print the memory address of variable x
cout << *&x << '\n';    /// print the value at the memory address of variable x

*&x = 20;
cout << x << '\n';      // print the value of variable x
```

1.1 Con trỏ

Khai báo con trỏ

- C++ cho phép đặt dấu sao cạnh kiểu dữ liệu, cạnh tên biến, hoặc ở giữa. Tuy nhiên, khi khai báo nhiều biến con trỏ, dấu sao phải được đặt cạnh mỗi biến.
- Ví dụ:

```
int *iPtr;          // con trỏ đến 1 địa chỉ chứa giá trị số nguyên

int* iPtr2;         // đúng cú pháp (nhưng không nên sử dụng)
int * iPtr3;        // đúng cú pháp (nhưng không nên sử dụng)

int *iPtr4, *iPtr5; // khai báo 2 con trỏ đến các biến số nguyên
```

1.1 Con trỏ

Gán giá trị cho con trỏ

- Vì con trỏ (pointer) là một biến chứa một địa chỉ bộ nhớ, nên khi giá trị cho con trỏ, giá trị đó phải là 1 địa chỉ.
- Ví dụ:

```
int value = 10;
int *ptr = &value;          // khởi tạo con trỏ ptr là địa chỉ biến value

cout << &value << '\n'; // in địa chỉ biến value
cout << ptr << '\n';       // in địa chỉ của con trỏ ptr đang giữ
cout << *ptr << "\n";     // in giá trị tại địa chỉ mà ptr trỏ đến , tương đương value
```

1.1 Con trỏ

Gán giá trị cho con trỏ

```
int iValue = 5;
double dValue = 7.0;
int *iPtr = &iValue;          // ok
double *dPtr = &dValue; // ok

iPtr = &dValue;   // sai – con trỏ int không thể trỏ đến địa chỉ biến double
dPtr = &iValue;   // sai – con trỏ double không thể trỏ đến địa chỉ biến int
int *newPtr = 5;  // sai – con trỏ chỉ có thể giữ 1 địa chỉ
```

- &iValue là 1 địa chỉ của biến kiểu int, nên nó chỉ có thể gán cho con trỏ kiểu int. Ta có thể kiểm tra kiểu dữ liệu của &iValue:

```
cout << typeid(&iValue).name() << "\n";
```

1.1 Con trỏ

Gán giá trị cho con trỏ

- Sau khi được gán, giá trị con trỏ có thể được gán lại cho một giá trị khác:

```
int value1 = 5;
int value2 = 7;
int *ptr;

ptr = &value1;              // ptr points to value1
cout << *ptr << "\n";       // prints 5

ptr = &value2;              // ptr now points to value2
cout << *ptr << "\n";       // prints 7
```

1.1 Con trỏ

Gán giá trị cho con trỏ

- Khi địa chỉ của biến value được gán cho con trỏ ptr:
  - ptr tương đương với &value
  - *ptr được xử lý giống như value
- Vì *ptr được xử lý giống như value, nên ta có thể gán giá trị cho *ptr như 1 biến thông thường:

```
int value = 5;
int *ptr = &value;          // ptr points to value

*ptr = 7;                   // *ptr tương đương với value
cout << value;              // prints 7
```

1.1 Con trỏ

Kích thước của con trỏ:

- Kích thước của con trỏ phụ thuộc vào kiến trúc mà tập tin thực thi được biên dịch.
  - Kiến trúc x86, con trỏ sẽ có kích thước 32-bit (4 bytes)
  - Kiến trúc x64, con trỏ sẽ có kích thước 64-bit (8 bytes)

1.1 Con trỏ

Đặc trưng

- Hiệu quả nhưng khó điều khiển
- Quan hệ chặt chẽ với mảng và xâu
- Có thể khai báo kiểu con trỏ cho bất kỳ kiểu dữ liệu nào
- Con trỏ khởi tạo bằng 0, NULL (Không trỏ vào gì cả), hoặc một địa chỉ.
- Ứng dụng nhiều trong xử lý ảnh và lập trình nhúng vì tốc độ xử lý nhanh.

1.2 Biến tham chiếu

Định nghĩa

- Ngoài biến giá trị, biến con trỏ, C++ cho phép sử dụng loại biến thứ ba là biến tham chiếu (reference variables). So với 2 loại biến nói trên, biến tham chiếu có những đặc điểm sau:
  - Biến tham chiếu không được cấp phát bộ nhớ, không có địa chỉ riêng.
  - Nó dùng làm bí danh cho một biến (kiểu giá trị) nào đó và nó sử dụng vùng nhớ của biến này.
- Một tham chiếu được khai báo bằng cách sử dụng toán tử & giữa kiểu dữ liệu và tên biến:

```
int value = 10;
int &ref = value; // ref tham chiếu đến biến value
```

1.2 Biến tham chiếu

Tham chiếu dưới dạng bí danh (another name)

- Một tham chiếu hoạt động như một bí danh cho đối tượng đang được tham chiếu.

```
int value = 10;
int &ref = value;          // ref tham chiếu đến biến value
value = 15;                // value = 15
ref = 20;                  // value = 20

cout << value << "\n";     // 20
++ref;                     // tương đương với ++value
cout << value << "\n";     // 21

cout << &value << "\n";    // in địa chỉ value và ref
cout << &ref << "\n";
```

1.2 Biến tham chiếu

Tham chiếu và con trỏ

- Tham chiếu tới một biến giống như một con trỏ được ngầm tham chiếu khi truy cập.

```
int value = 10;
int *ptr = &value;
int &ref = value;
*ptr = 15;

cout << *ptr << "\n";
cout << ref << "\n";
cout << value << "\n";

ref = 20;
cout << *ptr << "\n";
cout << ref << "\n";
cout << value << "\n";
```

1.2 Biến tham chiếu

Lưu ý khi sử dụng tham chiếu

- Nếu một vấn đề có thể được giải quyết bằng một tham chiếu hoặc một con trỏ, thì tham chiếu thường được ưu tiên. Con trỏ chỉ nên được sử dụng trong các tình huống mà tham chiếu không hỗ trợ (ví dụ như cấp phát bộ nhớ động).
- Không giống như con trỏ, tham chiếu không không thể giữ giá trị null. Vì vậy, tham chiếu phải được khởi tạo khi khai báo.
- Không thể tham chiếu đến một đối tượng khác sau khi khởi tạo.

```
int value1 = 5;
int value2 = 6;
int &ref = value1;          // ok
ref = value2;               // ref = value1 = 6
// ref luôn tham chiếu đến value1, và không thể thay đổi
```

2. Hàm trong C++

- Tại sao phải dùng chương trình con:
  - Có công việc cần phải được thực hiện tại nhiều nơi trong chương trình → tách công việc đó thành chương trình con.
  - Để thuận tiện trong quản lý, trình bày và phát triển.

- Trong C++, một chương trình con gọi là hàm: có tên, đầu vào và đầu ra, và có chức năng giải quyết một số vấn đề chuyên biệt cho chương trình chính.

2. Hàm trong C++

- Phân loại:
  - Hàm trả giá trị (return function)
  - Hàm không trả về giá trị (void function)

- Một hàm khi được định nghĩa thì có thể được gọi trong chương trình.
- Có thể được gọi nhiều lần với các tham số khác nhau.
- Trong C/C++, hàm main() được gọi thực hiện đầu tiên.
- Hàm có hai loại: hàm chuẩn (hàm được thư viện trong C/C++ viết sẵn) và hàm tự định nghĩa bởi người sử dụng.

2.1 Hàm chuẩn

- Hàm thư viện / hàm chuẩn 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>

2.1 Hàm chuẩn

Ý nghĩa của một số thư viện thường dùng:

- 1. iostream:
  - Thư viện chứa các hàm vào/ ra chuẩn (standard input/output).
  - Gồm các hàm cin, cout…
- 2. cmath:
  - Thư viện chứa các hàm tính toán.
  - Gồm các hàm abs(), sqrt(), log(). log10(), sin(), cos(), tan(), acos(), asin(), atan(), pow(), exp(),…
- 3. algorithm

  Chứa hàm về thuật toán đặc biệt như sort (quick-sort) của STL
- 4. vector

  Kiểu dữ liệu mảng động vector

2.1 Hàm chuẩn

Ý nghĩa của một số thư viện thường dùng:

- 5. stdio.h:
  - Thư viện chứa các hàm vào/ ra chuẩn (standard input/output).
  - Gồm các hàm printf(), scanf(), getc(), putc(), gets(), puts(), fflush(), fopen(), fclose(), fread(), fwrite(), getchar(), putchar(), …
- 6. string:
  - Thư viện chứa các hàm xử lý xâu string.
  - Gồm các hàm size(), length(), substr(), …
- 7. 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(), …

2.2 Hàm tự định nghĩa

- C/C++ cho phép lập trình viên tự định nghĩa hàm khi cần.
- Ví dụ:

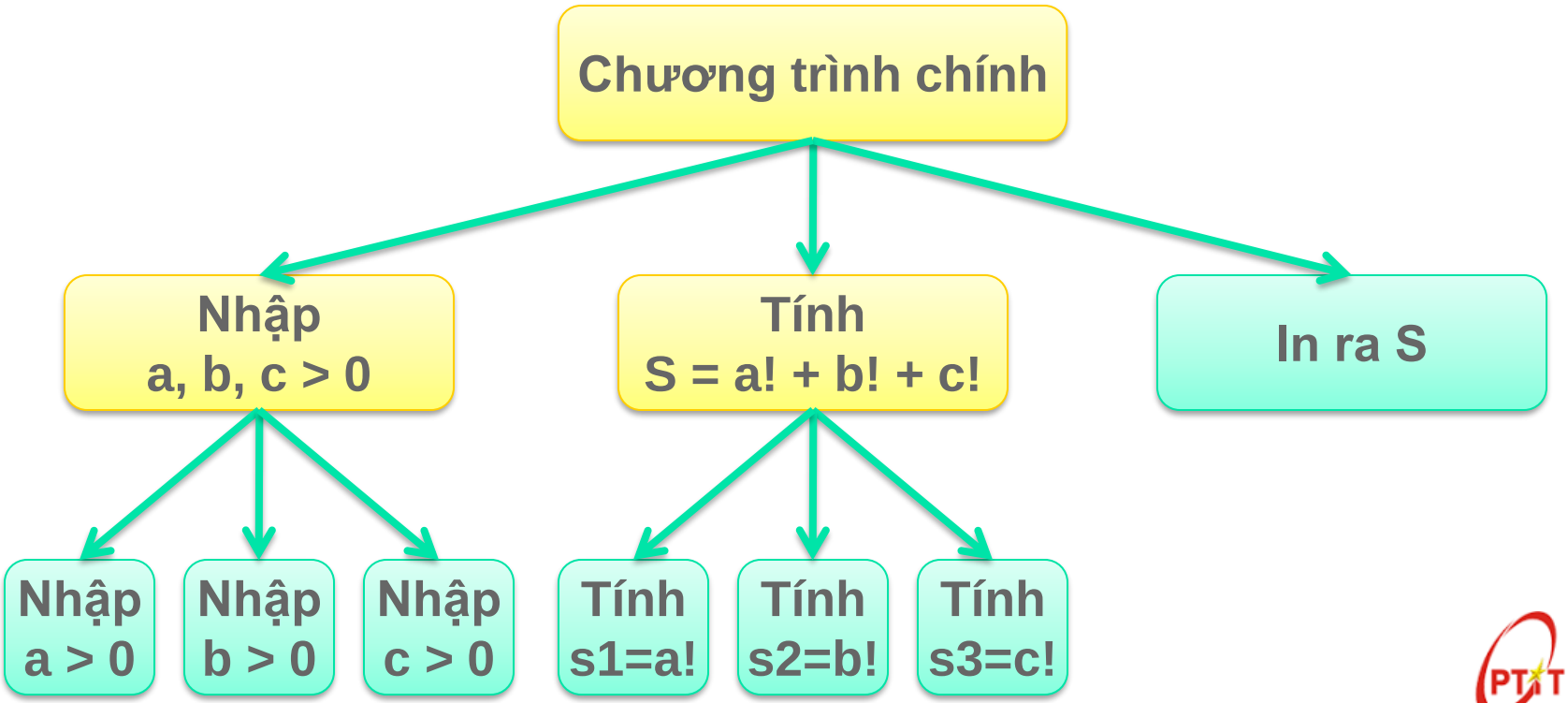
Khi cần vẽ một vòng tròn (circle) và tô màu (color). Có thể định nghĩa hai hàm để giải quyết vấn đề:

- Hàm createCircle()
- Hàm color()

2.2 Hàm tự định nghĩa

- Đặt vấn đề

Viết chương trình tính S = a! + b! + c! với a, b, c là các số nguyên dương nhập từ bàn phím.



2.2 Hàm tự định nghĩa

- Code thông thường: phần tính giai thừa code bị lặp lại

```
int main() {
    cin >> a >> b >> c;
    if (a <= 0 || b <= 0 || c <= 0) {
        cout << "Invalid input" << endl;
        return 0;
    }
    int s1 = 1;
    for(int i = 1; i <= a; i++)
        s1 = s1 * i;

    int s2 = 1;
    for(int i = 1; i <= b; i++)
        s2 = s2 * i;

    int s3 = 1;
    for(int i = 1; i <= c; i++)
        s3 = s3 * i;

    int S = s1+s2+s3;
    cout << S << endl;

    return 0;
}
```

2.2 Hàm tự định nghĩa

- Code thông thường: phần tính giai thừa code bị lặp lại, chỉ khác nhau tham số đầu vào (a, b, c) và tham số đầu ra (s1, s2, s3)

- Giải pháp: viết 1 hàm tính giai thừa → tái sử dụng code

```
int giaithua(int n) {
    int ans = 1;
    for(int i = 1; i <= n; i++)
        ans = ans*i;
    return ans;
}

int S = giaithua(a) + giaithua(b) + giaithua(c);
```

2.2 Hàm tự định nghĩa

Cú pháp hàm trả về giá trị:

```
<kiểu trả về> <tên hàm>([danh sách tham số])
{
        <các câu lệnh>
        [return <giá trị>;]
}
```

Trong đó

- <kiểu trả về> : kiểu bất kỳ của C (char, int, long, float,…).
- <tên hàm>: theo quy tắc đặt tên định danh.
- <danh sách tham số> : tham số hình thức đầu vào giống khai báo biến, cách nhau bằng dấu ,
- <giá trị> : trả về cho hàm qua lệnh return.

2.2 Hàm tự định nghĩa

Cú pháp hàm không trả về giá trị:

```
void <tên hàm>([danh sách tham số])
{
        <các câu lệnh>
}
```

Trong đó

- Bắt đầu: void
- <tên hàm>: theo quy tắc đặt tên định danh.
- <danh sách tham số> : tham số hình thức đầu vào giống khai báo biến, cách nhau bằng dấu ,
- Có thể dùng return ở giữa chừng để kết thúc hàm sớm.

2.3 Các bước xây dựng hàm

- Cần xác định các thông tin sau đây:
  - Tên hàm.
  - Hàm sẽ thực hiện công việc gì.
  - Các đầu vào (nếu có).
  - Đầu ra (nếu có).

2.3 Các bước xây dựng hàm

Ví dụ 1

- Tên hàm: XuatTong
- Công việc: tính và xuất tổng 2 số nguyên
- Đầu vào: hai số nguyên x và y
- Đầu ra: không có

```
void XuatTong(int x, int y) {
        int s;
        s = x + y;
        cout << x << " cong " << y << " bang " << s << endl;
}
```

2.3 Các bước xây dựng hàm

Ví dụ 2

- Tên hàm: TinhTong
- Công việc: tính và trả về tổng 2 số nguyên
- Đầu vào: hai số nguyên x và y
- Đầu ra: một số nguyên có giá trị x + y

```
int TinhTong(int x, int y) {
        int s;
        s = x + y;
        return s;
}
```

2.3 Các bước xây dựng hàm

Ví dụ 3

- Tên hàm: NhapXuatTong
- Công việc: nhập và xuất tổng 2 số nguyên
- Đầu vào: không có
- Đầu ra: không có

```
void NhapXuatTong() {
        int x, y;
        cout << “Nhap 2 so nguyen: ”;
        cin >> x >> y;
        cout << x << “ cong “ << y << “ bang “ << x+y << endl;
}
```

2.3 Các bước xây dựng hàm

- Một số chương trình, người ta thường đặt phần tiêu đề hàm/nguyên mẫu hàm (prototype) trên hàm main và phần định nghĩa hàm dưới hàm main.

```
void XuatTong(int x, int y);      // prototype
int main() {
        …
}

void XuatTong(int x, int y){
        cout << x << " cong " << y << " bang " << x+y << endl;
}
```

2.4 Tham số và lời gọi hàm

- Các đối số trong C/C++ là các biến được sử dụng để truyền các giá trị nhất định.

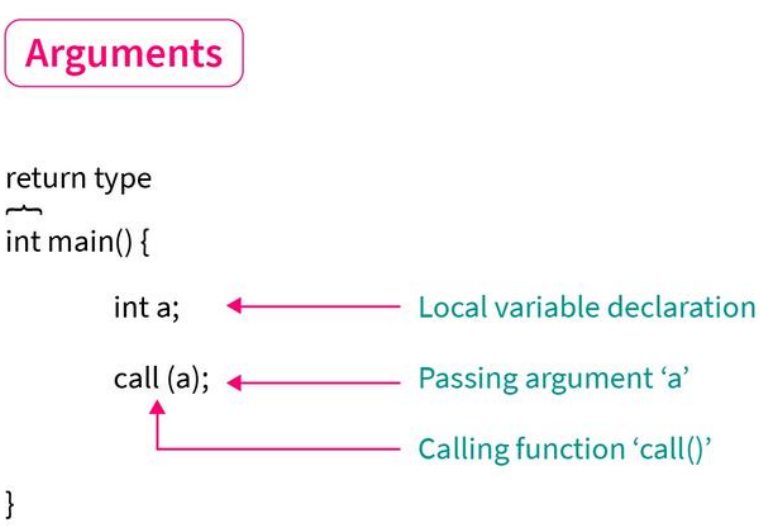
- Khi viết một hàm, có thể truyền dữ liệu dưới dạng đối số cho hàm qua lời gọi. Dữ liệu được truyền có thể là một giá trị số nguyên, giá trị số thực, v.v. Mảng hoặc chuỗi cũng có thể được truyền dưới dạng dữ liệu cho các hàm.

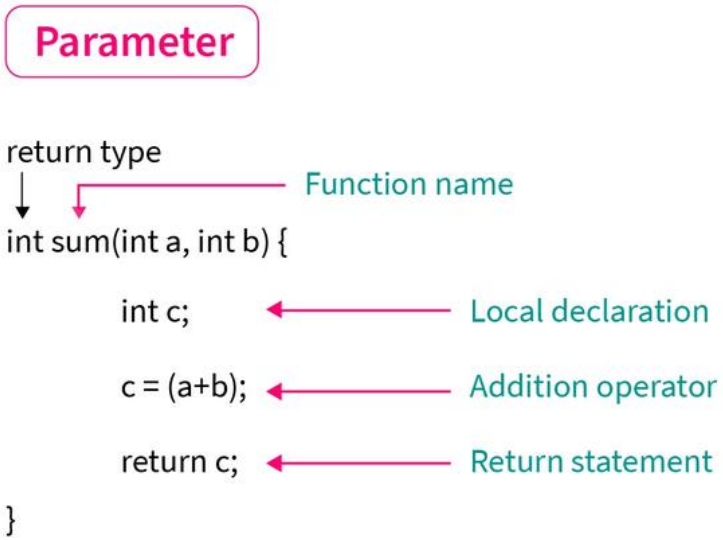
- Tham số trong C++ đề cập đến bất kỳ khai báo biến nào trong dấu ngoặc đơn trong quá trình khai báo hàm. Chúng được liệt kê trong định nghĩa của hàm, được phân tách bằng dấu phẩy.

2.4 Tham số và lời gọi hàm

- Đối số

Tham số





2.4 Tham số và lời gọi hàm

Ví dụ: Viết chương trình đổi chỗ 2 phần tử

```
#include<iostream>
using namespace std;

// Pass by value
1  void Swap1 (int x, int y){
2     int temp = x;
3     x = y;
4     y = temp;
5  }
6  // Pass by address (pointer)
   void Swap2 (int *x, int *y){
7     int temp = *x;
8     *x = *y;
9     *y = temp;
10 }
11 // Pass by reference
   void Swap3 (int &x, int &y){
12    int temp = x;
13    x = y;
14    y = temp;
15 }
```

```
int main() {
    int m=1, n=28;
    Swap1(m,n);
    cout << m << " " << n << endl;

    Swap2(&m,&n);
    cout << m << " " << n << endl;

    Swap3(m,n);
    cout << m << " " << n << endl;

    return 0;
}
```

2.4 Tham số và lời gọi hàm

Truyền tham trị (Pass by value)

- Truyền đối số cho hàm ở dạng giá trị.
- Có thể truyền hằng, biến, biểu thức nhưng hàm chỉ sẽ nhận giá trị.
- Được sử dụng khi không có nhu cầu thay đổi giá trị của tham số sau khi thực hiện hàm.

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

2.4 Tham số và lời gọi hàm

Truyền địa chỉ (Pass by address)

- Truyền đối số cho hàm ở dạng địa chỉ (con trỏ).
- Không được truyền giá trị cho tham số này.
- Được sử dụng khi có nhu cầu thay đổi giá trị của tham số sau khi thực hiện hàm.

```
void TruyenDiaChi(int *x){
        ...
        *x++;
}
```

2.4 Tham số và lời gọi hàm

Truyền tham chiếu (Pass by reference)

- Truyền đối số cho hàm ở dạng địa chỉ (con trỏ). Được bắt đầu bằng & trong khai báo.
- Không được truyền giá trị cho tham số này.
- Được sử dụng khi có nhu cầu thay đổi giá trị của tham số sau khi thực hiện hàm.

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

2.4 Tham số và lời gọi hàm

Lưu ý

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

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

2.4 Tham số và lời gọi hàm

Lưu ý

- Sử dụng tham chiếu là một cách để trả về giá trị cho chương trình.

```
int TinhTong(int x, int y){
        return x + y;
}
void TinhTong(int x, int y, int &tong) {
        tong = x + y;
}
void TinhTongHieu(int x, int y, int &tong, int &hieu) {
        tong = x + y; hieu = x – y;
}
```

Lưu ý khi xây dựng hàm

- Chức năng

Cố gắng mỗi hàm thực hiện giải quyết một vấn đề duy nhất

- Xác định kiểu trả về
  - Nếu hàm trả giá trị thì xác định kiểu dữ liệu tương ứng, return phù hợp
  - Nếu hàm chỉ thực hiện thủ tục, không mang giá trị thì kiểu trả về là void
- Mỗi hàm chỉ trả về được 1 giá trị thông qua kiểu trả về, muốn trả về nhiều giá trị thì có thể sử dụng tham chiếu
- **Đặt tên: đặt tên hàm có ý nghĩa**
  - Viết toàn bộ bằng chữ thường, giữa các chữ là dấu _

  is_prime(), cal_rectangle_area, cal_rect_area, …
  - Viết liền, chữ cái đầu tiên của từ tiếp theo được viết hoa

  isPrime, calRectangleArea, calRectArea, …

2 Bài tập

- 1. Viết hàm tìm số lớn nhất trong 3 số a, b, c
- 2. Viết hàm kiểm tra số nguyên tố
- 3. Hãy liệt kê tất cả các số nguyên tố có N chữ số sao cho tổng các chữ số của số đó đúng bằng S cho trước.
  - Input: Hai số nguyên N, S (1 <= N <= 6, 0 <= S <= 54).
  - Output: Các số thỏa mãn, in ra theo thứ tự tăng dần

2 Bài tập

- 4. Viết chương trình đếm tất cả các số tự nhiên K nếu thỏa tất cả mãn điều kiện dưới đây
  - K là số có N chữ số
  - K là số nguyên tố
  - Số viết ngược lại của K là 1 số nguyên tố
  - Tổng các chữ số của K cũng là số nguyên tố
  - Mỗi chữ số trong K cũng là số nguyên tố

QUESTIONS & ANSWERS