TRƯỜNG ĐẠI HỌC KINH TẾ KỸ THUẬT CÔNG NGHIỆP
Khoa Công Nghệ Thông Tin

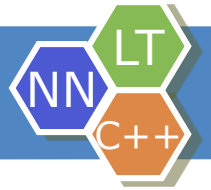
FUTURE TECHNOLOGY
CÔNG NGHỆ TƯƠNG LAI

Phần 2. NN lập trình C++

CHƯƠNG 8
HÀM (FUNCTION)

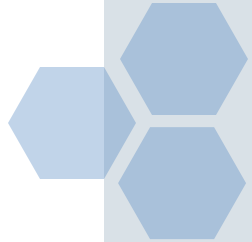
Chương 7 – Con trỏ (Pointer)

1. Khai báo và định nghĩa hàm
2. Các tham số của hàm
3. Cấp lưu trữ và phạm vi của đối tượng
4. Hàm đệ quy



Lý do sử dụng hàm

- Các hàm cho phép chia một vấn đề thành các vấn đề nhỏ hơn
  - Cho phép giải quyết vấn đề khó khăn dễ dàng hơn
- Một chương trình rõ ràng hơn khi sử dụng các hàm
  - Chúng ta chỉ cần biết hàm làm gì mà không cần quan tâm đến cách thực hiện của nó
- Chúng cho phép khái quát hóa một số nhóm câu lệnh lặp lại nhiều lần
  - Ngăn việc viết lặp lại một nhóm câu lệnh nhiều lần



Giới thiệu hàm

- Tư tưởng Chia để trị - Divide and conquer
  - Xây dựng một chương trình từ các thành phần (component) nhỏ hơn
  - Quản lý từng thành phần dễ quản lý hơn quản lý chương trình ban đầu
- Xây dựng chương trình theo hướng phân tích từ trên xuống (Top – Down Anlysis)
- Trong C++ có các module: các hàm(function) và lớp(class)

Giới thiệu hàm

- Trong C++ có hai loại hàm được sử dụng:
  - Các hàm và lớp do lập trình viên tự định nghĩa
  - Các hàm và lớp đóng gói sẵn từ thư viện chuẩn
- Lời gọi hàm - function call
  - tên hàm và các thông tin (các đối số - arguments) mà nó cần
- Để sử dụng các hàm đóng gói sẵn
  - Chỉ ra thư viện trong định hướng tiền xử lý
  - Trong chương trình, sử dụng lời gọi hàm
- Để sử dụng hàm do lập trình viên tự định nghĩa
  - Định nghĩa hàm
  - Trong chường trình, sử dụng lời gọi hàm.

Giới thiệu hàm

- **Ví dụ:** các tính toán toán học thông thường
  - #include <math.h>
- Cách gọi hàm
  - tên_hàm (đối_số); **hoặc**
  - tên_hàm(đối_số_1, đối_số_2, …);
- Ví dụ

  cout << sqrt(900.0);
  - Mọi hàm trong thư viện toán đều trả về giá trị kiểu **double**
- các đối số (argument) cho hàm có thể là
  - hằng - Constants
    - **sqrt(4);**
  - biến - Variables
    - **sqrt(x);**
  - biểu thức - Expressions
    - **sqrt(sqrt(x)) ;**
    - **sqrt(3 - 6x);**

Method	Description	Example
ceil(x)	làm tròn *x* tới số nguyên nhỏ nhất không nhỏ hơn *x*	ceil(9.2) is 10.0 ceil(-9.8) is -9.0
cos(x)	cos của *x* (lượng giác) (*x* tính theo đơn vị radian)	cos(0.0) is 1.0
exp(x)	hàm mũ: *e mũ x*	exp(1.0) is 2.71828 exp(2.0) is 7.38906
fabs(x)	giá trị tuyệt đối của *x*	fabs(5.1) is 5.1 fabs(0.0) is 0.0 fabs(-8.76) is 8.76
floor(x)	làm tròn *x* xuống số nguyên lớn nhất không lớn hơn *x*	floor(9.2) is 9.0 floor(-9.8) is -10.0
fmod(x, y)	phần dư của phép chia *x/y* , tính bằng kiểu số thực	fmod(13.657, 2.333) is 1.992
log(x)	loga tự nhiên của x (cơ số *e*)	log(2.718282) is 1.0 log(7.389056) is 2.0
log10(x)	loga cơ số 10 của *x*	log10(10.0) is 1.0 log10(100.0) is 2.0
pow(x, y)	*x* mũ *y*	pow(2, 7) is 128 pow(9, .5) is 3
sin(x)	sin *x* (lượng giác) (*x* tính theo radian)	sin(0.0) is 0
sqrt(x)	căn bậc hai của *x*	sqrt(900.0) is 30.0 sqrt(9.0) is 3.0
tan(x)	tang *x* (lượng giác) (*x* tính theo radian)	tan(0.0) is 0

Fig. 3.2 Math library functions.

Giới thiệu hàm

- **Hàm (chương trình con)**
  - Module hóa một chương trình
  - khả năng tái sử dụng
- **Các biến địa phương – Local variables**
  - khai báo trong hàm nào thì chỉ được biết đến bên trong hàm đó
  - biến được khai báo bên trong định nghĩa hàm là biến địa phương
- **Các tham số – Parameters**
  - là các biến địa phương với giá trị được truyền vào hàm khi hàm được gọi
  - cung cấp thông tin từ bên ngoài hàm

Giới thiệu hàm

Định nghĩa hàm

Lời gọi hàm

```
#include <iostream.h>

// Function prints a greeting

void printHello ()
{
   cout<<"Hello World!\n";
}


// Calling the greeting function

int main()
{
   printHello();
   return 0;
}
```

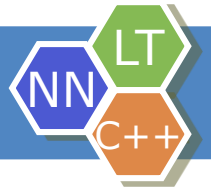
Giới thiệu hàm

```
1   #include <iostream.h>
2
3   int square( int );   // function prototype
4
5   int main()
6   {
7           // loop 10 times and calculate and output
8           // square of x each time
9           for ( int x = 1; x <= 10; x++ )
10                  cout << square( x ) << "  ";  // function call
11          cout << endl;
12          return 0;  // indicates successful termination
13
14  } // end main
15
16  // square function definition returns square of an integer
17  int     square( int y )  // y is a copy of argument to function
18  {
19          return y * y;     // returns square of y as an int
20   } // end function square
```

Function prototype: chỉ rõ kiểu dữ liệu của đối số và giá trị trả về. `square` cần một số `int`, và trả về `int`.

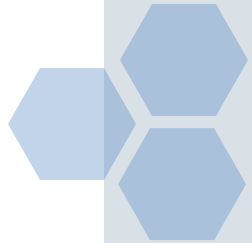
Cặp ngoặc `()` dùng khi gọi hàm. Khi chạy xong, hàm trả kết quả.

Định nghĩa hàm `square`. `y` là một bản sao của đối số được truyền vào. Hàm trả về `y * y`.



8.1 Khai báo và định nghĩa hàm

- Viết một hàm cần xác định:
  - tên của hàm
  - các tham số của hàm
  - Giá trị trả về của hàm
  - các câu lệnh được thực hiện khi hàm được gọi
- Các câu lệnh được gọi là "phần thân hàm"



8.1 Khai báo và định nghĩa hàm

- **Khai báo hàm:**

  *[<kiểu giá trị trả về>] <tên hàm>***(** *<danh sách kiểu tham số>* **)**

- **Định nghĩa hàm:**

  *[<kiểu giá trị trả về>] <tên hàm>***(***<danh sách tham số>* **)**

  {

  // các khai báo cục bộ và các câu lệnh

  }

  - danh sách tham số – Parameter list
    - dấu phảy tách các tham số
    - Nếu không có tham số, sử dụng **void** hoặc để trống
  - giá trị trả về – Return-value-type
    - kiểu của giá trị trả về (sử dụng **void** nếu không trả về giá trị gì)

8.1 Khai báo và định nghĩa hàm

Định nghĩa hàm

Ví dụ về hàm

```
int square(int y )
{
    return y * y;
}

int main()
{
    ...
    cout << square(x);
    ...
}
```

- Từ khóa **return**
  - trả dữ liệu về, và trả điều khiển lại cho nơi gọi (caller)
    - nếu không trả về, sử dụng **return;**
  - hàm kết thúc khi chạy đến ngoặc phải (**}**)
    - điều khiển cũng được trả về cho nơi gọi
- Không thể định nghĩa một hàm bên trong một hàm khác

8.1 Khai báo và định nghĩa hàm

Tên hàm

Thân hàm

```
#include <iostream.h>

int factorial (int a)
{
    int i, fac = 1;
    for(i=1; i<=a; i++)
        fact = fac * i;
    return fac;
}

int main()
{   int n;

    cout<<"Nhap so nguyen n ";
    cin>>n;

    cout<<n<<"! = "<<factorial(n);
}
```

8.1 Khai báo và định nghĩa hàm

Nguyên mẫu hàm - Function Prototype

- Function prototype bao gồm
  - Tên hàm
  - Các tham số (số lượng và kiểu dữ liệu)
  - Kiểu trả về (**void** nếu không trả về giá trị gì)
- Function prototype chỉ cần đến nếu định nghĩa hàm đặt sau lời gọi hàm (function call)
- Prototype phải khớp với định nghĩa hàm
  - Function prototype

    ```
    double maximum( double, double, double );
    ```
  - Function definition

    ```
    double maximum( double x, double y, double z )
    {
     ...
    }
    ```

8.1 Khai báo và định nghĩa hàm

Nguyên mẫu hàm

Nguyên mẫu hàm

```
#include <iostream.h>

int giaithua (int);

int main()
 { int n;

   cout<<"Nhap so nguyen n";
   cin>>n;

   cout<<n<<"! = "<< giaithua(n);
}
```

Định nghĩa hàm

```
int  giaithua (int a)
{   int i, gt = 1;
   for(i=1; i<=a; i++)
      gt = gt * i;
   return gt;
}
```

8.1 Khai báo và định nghĩa hàm

```
1   #include <iostream.h>
2
3    double maximum( double, double, double ); // function prototype
4
5   int main()
6   {
7      double number1;
8      double number2;
9      double number3;
10
11       cout << “Nhap vao 3 so thuc: ";
12       cin >> number1 >> number2 >> number3;
13
14       // number1, number2 and number3 are arguments to
15       // the maximum function call
16       cout << “So max la: "
17            << maximum( number1, number2, number3 ) << endl;
18
19    return 0;  // indicates successful termination
20  } // end main
```

Hàm `maximum` lấy 3 tham số (cả 3 là `double`) và trả về một `double`.

8.1 Khai báo và định nghĩa hàm

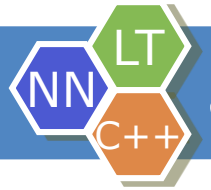
```
21  double maximum( double x, double y, double z )
22  {
23     double max = x;   // assume x is largest
24
25     if ( y > max )    // if y is larger,
26        max = y;       // assign y to max
27
28     if ( z > max )    // if z is larger,
29        max = z;       // assign z to max
30
31     return max;       // max is largest value
32
33  } // end function maximum
```

dấu phảy phân tách các tham số.

```
Nhap vao 3 so thuc: 99.32 37.3 27.1928
So max la: 99.32

Nhap vao 3 so thuc : 1.1 3.333 2.22
So max la: 3.333

Nhap vao 3 so thuc : 27.9 14.31 88.99
So max la: 88.99
```



8.2 Các tham số của hàm

- Tham số là thông tin được chuyển đến một hàm
- Các tham số “hình thức" là các biến cục bộ được khai báo trong khai báo hàm.
- Tham số "thực sự" là các giá trị được truyền cho hàm khi nó được gọi
  - Là biến cục bộ của hàm có giá trị được xác định sau mỗi lần gọi.
- Do đó, các tham số có giá trị khác nhau trong mỗi lần chúng được gọi.
  - Các tham số chỉ có thể được truy cập trong một hàm.
  - Khi gọi một hàm, tất cả các tham số phải có giá trị.

8.2 Các tham số của hàm

```
#include <iostream.h>

int addOne (int x )
{
    x = x + 1;
    return x;
}

int  main()
{
  int i = 3;

  cout<<addOne(i);
  cout<<i;

 return 0;
}
```

Khai báo một tham số hình thức như một biến cục bộ

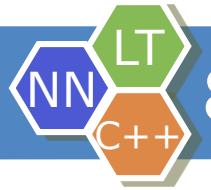
Thay đổi giá trị của biến cục bộ

Truyền giá trị của i trong hàm main cho hàm addOne

Output:

4

3



8.2 Các tham số của hàm

Giá trị trả về

- Câu lệnh return được sử dụng để trả về giá trị cho một hàm.
- Một hàm có thể có một số câu lệnh trả về. Trả về đầu tiên mà chương trình đáp ứng sẽ kết thúc hàm.
- Một hàm không trả về gì phải được khai báo với kiểu trả về void. Trong trường hợp này, không cần giá trị trả về.

8.2 Các tham số của hàm

Biến toàn cục (global variable)

- Biến được khai báo trong một hàm là biến cục bộ.
- Biến này chỉ có thể truy cập trong hàm của nó.
- Biến toàn cục được khai báo bên ngoài phạm vi hàm và có thể được sử dụng bởi các hàm khác nhau. Ví dụ: `int global;`

```
void f (void) {global = 0; }
void f (void) {global = 1; }
```

8.2 Các tham số của hàm

- Khi biến toàn cục và biến cục bộ trong một hàm có cùng tên thì biến cục bộ sẽ được sử dụng.
- Ví dụ:

```
int i;
void f()
{  int i;
   i++;  // chỉ thay đổi biến cục bộ i
}
void g()
{  i ++; // thay đổi biến toàn cục i
}
```

8.2 Các tham số của hàm

- Các tham chiếu là các biệt danh (alias) của các biến khác
  - chỉ tới cùng một biến
  - có thể được dùng bên trong một hàm

```
int count = 1;      // khai báo biến nguyên count
int &cRef = count; // tạo cRef là một biệt danh của count
++cRef;            // tăng count (sử dụng biệt danh của count)
```

- Các tham chiếu phải được khởi tạo khi khai báo
  - Nếu không, trình biên dịch báo lỗi
  - Tham chiếu lạc (Dangling reference)
    - tham chiếu tới biến không xác định

8.2 Các tham số của hàm

```
1   #include <iostream.h>
2
3   int main()
4   {
5      int x = 3;
6
7      // y refers to (is an alias for) x
8      int &y = x;
9
10     cout << "x = " << x << endl << "y = " << y << endl;
11     y = 7;
12     cout << "x = " << x << endl << "y = " << y << endl;
13
14     return 0;  // indicates successful termination
15
16  } // end main
```

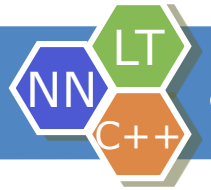
y được khai báo là một tham chiếu tới x.

```
x = 3
y = 3
x = 7
y = 7
```

8.2 Các tham số của hàm

```
1   #include <iostream.h>
2
3   int main()
4   {
5      int   x = 3;
6      int  &y;     // Error: y must be initialized
7
8      cout << "x = " << x << endl << "y = " << y << endl;
9      y = 7;
10     cout << "x = " << x << endl << "y = " << y << endl;
11
12     return 0;  // indicates successful termination
13
14  } // end main
```

Lỗi biên dịch – tham chiếu không được khởi tạo.



8.2 Các tham số của hàm

Tham chiếu

- Gọi bằng giá trị - Call by value
  - Bản sao của dữ liệu được truyền cho hàm
  - Thay đổi đối với bản sao không ảnh hưởng tới dữ liệu gốc
  - Ngăn chặn các hiệu ứng phụ không mong muốn
- Gọi bằng tham chiếu - Call by reference
  - Hàm có thể truy nhập trực tiếp tới dữ liệu gốc
  - Các thay đổi thể hiện tại dữ liệu gốc

8.2 Các tham số của hàm

Tham số là tham chiếu

- Tham số tham chiếu - Reference parameter
  - Ý nghĩa: Là biệt danh (alias) của biến được truyền vào lời gọi hàm
    - 'truyền tham số bằng tham chiếu' hay 'truyền tham chiếu'
  - Cú pháp: Đặt ký hiệu **&** sau kiểu dữ liệu tại prototype của hàm
    - **`void myFunction( int &data )`**
    - có nghĩa "**`data`** là một tham chiếu tới một biến kiểu **`int`**"
  - dạng của lời gọi hàm không thay đổi
    - tuy nhiên dữ liệu gốc khi được truyền bằng tham chiếu có thể bị sửa đổi
- Con trỏ (chương 7)
  - Một cách truyền tham chiếu khác

8.2 Các tham số của hàm

```
1   #include <iostream.h>
2
3   int squareByValue( int );        // function prototype
4   void squareByReference( int & ); // function prototype
5
6   int main()
7   {
8      int x = 2;
9      int z = 4;
10
11     // demonstrate squareByValue
12     cout << "x = " << x << " before squareByValue\n";
13     cout << "Value returned by squareByValue: “  << squareByValue( x ) << endl;
14     cout << "x = " << x << " after squareByValue\n" << endl;
15  // demonstrate squareByReference
16     cout << "z = " << z << " before squareByReference" << endl;
17     squareByReference( z );
18     cout << "z = " << z << " after squareByReference" << endl;
19
20     return 0;  // indicates successful termination
21  } // end main
```

Lưu ý ký hiệu & có nghĩa truyền tham chiếu (pass-by-reference).

8.2 Các tham số của hàm

```
22  int squareByValue( int number )
23  {
24      return number *= number;  // caller's argument not modified
25
26   } // end function squareByValue
27
28  void squareByReference( int &numberRef )
29  {
30       numberRef  *= numberRef;   // caller's argument modified
31
32  } // end function squareByReference
```

thay đổi number, nhưng đối số gốc (x) không bị thay đổi.

thay đổi numberRef, một biệt danh của đối số gốc. Do đó, z bị thay đổi.

```
x = 2 before squareByValue
Value returned by squareByValue: 4
x = 2 after squareByValue

z = 4 before squareByReference
z = 16 after squareByReference
```

8.3 Cấp lưu trữ và phạm vi của các đối tượng

- Biến có các thuộc tính
  - đã biết: tên, kiểu, kích thước, giá trị
  - kiểu lưu trữ – Storage class
    - biến tồn tại bao lâu trong bộ nhớ
  - Phạm vi – Scope
    - biến có thể được sử dụng tại những nơi nào trong chương trình
  - Liên kết – Linkage
    - Đối với những chương trình gồm nhiều file (multiple-file program), những file nào có thể sử dụng biến đó

8.3 Cấp lưu trữ và phạm vi của các đối tượng

- Loại biến tự động – Automatic storage class
  - biến được tạo khi chương trình chạy vào một khối chương trình (block)
  - và bị hủy bỏ khi chương trình ra khỏi block
  - Chỉ có các biến địa phương của các hàm mới có thể là biến tự động
    - mặc định là tự động
    - từ khóa **`auto`** dùng để khai báo biến tự động
  - từ khóa **`register`**
    - gợi ý đặt biến vào thanh ghi tốc độ cao
    - có lợi cho các biến thường xuyên được sử dụng (con đếm vòng lặp)
    - Thường là không cần thiết, trình biên dịch tự tối ưu hóa
  - Chỉ dùng một trong hai từ **`register`** hoặc **`auto`**.
    - **`register int counter = 1;`**

8.3 Cấp lưu trữ và phạm vi của các đối tượng

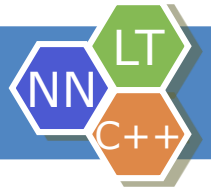
- Loại biến tĩnh – Static storage class
  - Biến tồn tại trong suốt chương trình
  - Có thể không phải nơi nào cũng dùng được, do áp dụng quy tắc phạm vi (scope rules)
- từ khóa **`static`**
  - dành cho biến địa phương bên trong hàm
  - giữ giá trị giữa các lần gọi hàm
  - chỉ được biết đến trong hàm của biến đó
- từ khóa **`extern`**
  - mặc định với các biến/hàm toàn cục (global variables/functions)
    - toàn cục: được định nghĩa bên ngoài các hàm
  - được biết đến tại mọi hàm nằm sau biến đó

8.3 Cấp lưu trữ và phạm vi của các đối tượng

- Phạm vi – Scope
  - Phạm vi của một định danh (tên) là phần chương trình nơi có thể sử dụng định danh đó
- Phạm vi file – File scope
  - được định nghĩa bên ngoài một hàm và được biết đến tại mọi hàm trong file
  - các biến toàn cục (global variable), định nghĩa và prototype của các hàm.
- Phạm vi hàm – Function scope
  - chỉ có thể được dùng đến bên trong hàm chứa định nghĩa
  - Chỉ áp dụng cho các nhãn (label), ví dụ: các định danh đi kèm một dấu hai chấm (**`case:`**)

8.3 Cấp lưu trữ và phạm vi của các đối tượng

- Phạm vi khối – Block scope
  - Bắt đầu tại nơi khai báo, kết thúc tại ngoặc phải **}**
    - chỉ có thể được dùng trong khoảng này
  - Các biến địa phương, các tham số hàm
  - các biến **static** cũng có phạm vi khối
    - loại lưu trữ độc lập với phạm vi
- Function-prototype scope
  - danh sách tham số của function prototype
  - không bắt buộc phải chỉ rõ các tên trong prototype
    - Trình biên dịch bỏ qua
  - Trong một prototype, mỗi tên chỉ được dùng một lần



8.4 Hàm đệ quy – Recursion

- Các hàm đệ quy – Recursive functions
  - các hàm tự gọi chính mình
  - chỉ giải quyết một trường hợp cơ bản (base case)
- Nếu không phải trường hợp cơ bản
  - Chia bài toán thành các bài toán nhỏ hơn
  - Gọi bản sao mới của hàm để giải quyết vấn đề nhỏ hơn (gọi đệ quy (recursive call) hoặc bước đệ quy(recursive step))
    - hội tụ dần dần về trường hợp cơ bản
    - hàm gọi chính nó tại lệnh return
  - Cuối cùng, trường hợp cơ bản được giải quyết
    - câu trả lời đi ngược lên, giải quyết toàn bộ bài toán

8.4 Hàm đệ quy – Recursion

- Ví dụ: tính giai thừa (factorial)

  $n! = n * (n - 1) * (n - 2) * \ldots * 1$

  - Quan hệ đệ quy (n! = n * (n – 1)!)

    $5! = 5 * 4!$

    $4! = 4 * 3!\ldots$

  - Trường hợp cơ bản (1! = 0! = 1)

8.4 Hàm đệ quy – Recursion

```
1   #include <iostream.h>
2   #include <iomanip.h>
3
4   unsigned long giaithua( unsigned long ); // function prototype
5
6   int   main()
7   {
8         for ( int i = 0; i <= 10; i++ )
9               cout << setw( 2 ) << i << "! = " << giaithua( i ) << endl;
10
11        return 0;  // indicates successful termination
12  } // end main
13
14  unsigned long giaithua( unsigned long number )
15  {
16     // base case
17     if ( number <= 1 )
18        return 1;
19
20     // recursive step
21     else
22        return number * giaithua( number - 1 );
23
24  } // end function factorial
```

Kiểu dữ liệu unsigned long có thể lưu số nguyên trong khoảng từ 0 đến 4 tỷ.

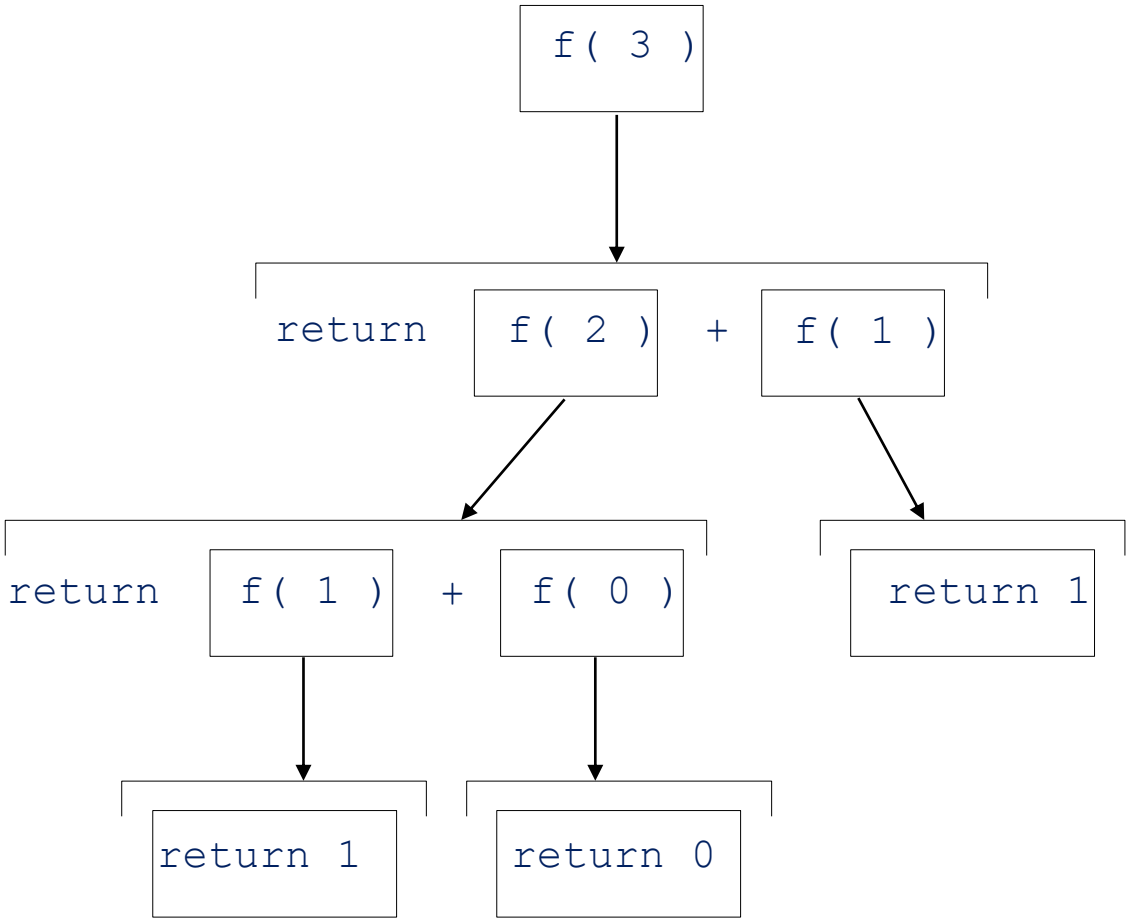
Trường hợp cơ bản xảy ra khi ta có 0! hoặc 1!.
Mọi trường hợp khác phải được chia nhỏ (bước đệ qui).

8.4 Hàm đệ quy – Recursion

- Chuỗi Fibonacci: 0, 1, 1, 2, 3, 5, 8...
  - Mỗi số là tổng của hai số đứng liền trước
  - Ví dụ một công thức đệ quy:
    - *fib(n) = fib(n-1) + fib(n-2)*
- Mã C++ cho hàm Fibonacci

```
long fib( long n )
{ if (n == 0 || n == 1)  // base case
        return  n;
  else
        return fib(n - 1) + fib(n – 2);
}
```

8.4 Hàm đệ quy – Recursion



8.4 Hàm đệ quy – Recursion

- Thứ tự thực hiện
  - `return fib( n - 1 ) + fib( n - 2 );`
- Không xác định hàm nào được thực hiện trước
  - C++ không qui định
  - Chỉ có các phép `&&, ||` và `?:` đảm bảo thứ tự thực hiện từ trái sang phải
- Các lời gọi hàm đệ quy
  - Mỗi tầng đệ quy nhân đôi số lần gọi hàm
    - số thứ 30 cần 2^30 ~ 4 tỷ lời gọi hàm
  - Độ phức tạp lũy thừa (Exponential complexity)

8.4 Hàm đệ quy – Recursion

```
1   #include <iostream.h>
2   unsigned long fib( unsigned long ); // function prote
3
4   int main()
5   {
6    unsigned long result, number;
7
8    // obtain integer from user
9    cout << "Enter an integer: ";
10     cin >> number;
11
12     // calculate fibonacci value for number input by user
13     result = fib(number );
14
15     // display result
16     cout << "Fibonacci(" << number << ") = " << result << endl;
17
18     return 0;  // indicates successful termination
19     }
```

Các số Fibonacci tăng rất nhanh và đều là số không âm.
Do đó, ta dùng kiểu

8.4 Hàm đệ quy – Recursion

```
20  // recursive definition of function fibonacci
21  unsigned long fib( unsigned long n )
22  {
23     // base case
24     if ( n == 0 || n == 1 )
25        return n;
26
27     // recursive step
28     else
29        return fib( n - 1 ) + fib( n - 2 );
30
31  } // end function fibonacci
```

```
Enter an integer: 0
Fibonacci(0) = 0

Enter an integer: 1
Fibonacci(1) = 1

Enter an integer: 2
Fibonacci(2) = 1

Enter an integer: 3
Fibonacci(3) = 2
```

Câu hỏi củng cố bài

1. Hàm nào sau đây là một hàm đầy đủ?

   A. int funct();

   B. int funct(int x) {return x=x+1;}

   C. void funct(int) {printf(“Hello”);

   D. void funct(x) {printf(“Hello”); }

Multiple Choice

Câu hỏi củng cố bài

2. Cách khai báo hàm nào sau đây là đúng?

A. <kieu tra ve>: <ten_ham>(thamso1, …) {Khoi lenh}

B. <kieu tra ve> <ten_ham>(thamso1, …) {Khoi lenh}

C. <ten_ham>(thamso1, …) {Khoi lenh}

D. <ten_ham> {Khoi lenh}

Multiple Choice

Câu hỏi củng cố bài

3. Thế nào là truyền tham biến?

A. Truyền địa chỉ vào biến của hàm

B. Truyền bản sao của biến vào hàm chứ không phải bản thân biến

C. Truyền giá trị của tham số vào biến

D. Truyền bản sao của tham số vào biến.

Multiple Choice

Câu hỏi củng cố bài

4. Thế nào là truyền tham trị?

A. Truyền địa chỉ vào biến của hàm

B. Truyền bản sao của biến vào hàm chứ không phải bản thân biến

C. Truyền giá trị của tham số vào biến

D. Truyền bản sao của tham số vào biến.

Multiple Choice

Câu hỏi củng cố bài

5. Đoạn lệnh sau cho kết quả thế nào?

A. 5

B. 6

C. 0

D. Lỗi biên dịch

```
int divide (int a,int b) { return (a / b); }
void output(int a) { cout << a; }
int main() {
int x = 100, y, z;
output(divide(12));
return 0;
}
```

Multiple Choice

Câu hỏi củng cố bài

6. Đoạn lệnh sau cho kết quả thế nào?

```
int addition (int a,int b) { return (a * b); }
int main() {
int z = addition(5,3);
cout << z;
return 0;
}
```

A. 15

B. 8

C. 2

D. Lỗi biên dịch

Multiple Choice

Câu hỏi củng cố bài

7. Đoạn lệnh sau cho kết quả thế nào?

A. 18

B. 9

C. 8

D. 12

```
int addition (int a,int b) { return (a - b); }
int main() {
int x = 5, y = 3, z = 10 + addition(x,y);
cout << z;
return 0;
}
```

Multiple Choice

Câu hỏi củng cố bài

8. Đoạn lệnh sau cho kết quả thế nào?

A. 120

B. 5

C. 0

D. Lỗi biên dịch

```
long facto (long a) {
if (a > 1) return (a * facto(a – 1));
else return (0);
}

int main() {
cout << facto(5);
return 0;
}
```

Multiple Choice

Câu hỏi củng cố bài

9. Đoạn lệnh sau cho kết quả thế nào?

A. 0, 0

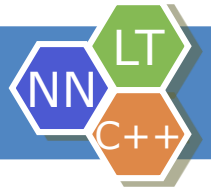
B. 10, 20

C. 20, 10

D. Báo lỗi cú pháp

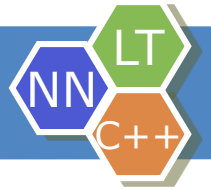
```
void functionS w(int &x, int &y) {
    int tmp = x;
    x = y;
    y = tmp;
}
int main() {
    int i = 10, j = 20;
    functionSw(i,j);
    cout << i << ", " << j << endl;
}
```

Multiple Choice



Câu hỏi lý thuyết

1. Nêu cách khai báo nguyên mẫu hàm? Cho ví dụ.
2. Nêu cách định nghĩa hàm? Cho ví dụ.
3. Nêu khái niệm tham số hình thức, tham số thực sự, biến toàn cục và biến cục bộ.



Bài tập

1. Viết hàm tìm số lớn nhất trong 2 số. Áp dụng tìm số lớn nhất trong 4 số nhập vào từ bàn phím.
2. Viết hàm tìm UCLN của 2 số nguyên dương, áp dụng tìm UCLN của 4 số nguyên dương a, b, c, d nhập vào từ bàn phím.
3. Viết hàm kiểm tra một số có phải là số nguyên tố hay không? Áp dụng in ra các số nguyên tố trong phạm vi n.

Bài tập

4. Viết hàm kiểm tra một số có phải là số hoàn hảo hay không? Áp dụng in ra các số hoàn hảo trong mảng.
5. Viết hàm đệ quy tính

$$S = 1 + 2 + 3 + \dots + (n-1) + n.$$

6. Viết hàm đệ quy tính giai thừa, áp dụng tính tổ hợp chập k của n.

$$C^{k}_{n} = \frac{n!}{k! * (n-k)!}$$

Bài tập

7. Viết hàm đệ quy tính Sn= $1^1 * 2^2 * 3^3 * 4^4 *\ldots* n^n$.
8. Viết hàm đệ quy tính UCLN của 2 số nguyên dương p, q. Áp dụng tìm UCLN của 3 số nguyên dương a, b, c.