LẬP TRÌNH NÂNG CAO

Bài 2+3: Hàm trong C/C++

Nội dung chính

1. Cấu chúc chung của hàm
2. Hiểu về cách hàm hoạt động
3. Các hàm có sẵn
4. Phạm vi của biến và của hàm
5. Truyền tham số trong hàm
6. Nạp chồng hàm
7. Hàm đệ quy
8. Bài tập

Phần 1

Cấu chúc chung của hàm

Cấu chúc chung của hàm

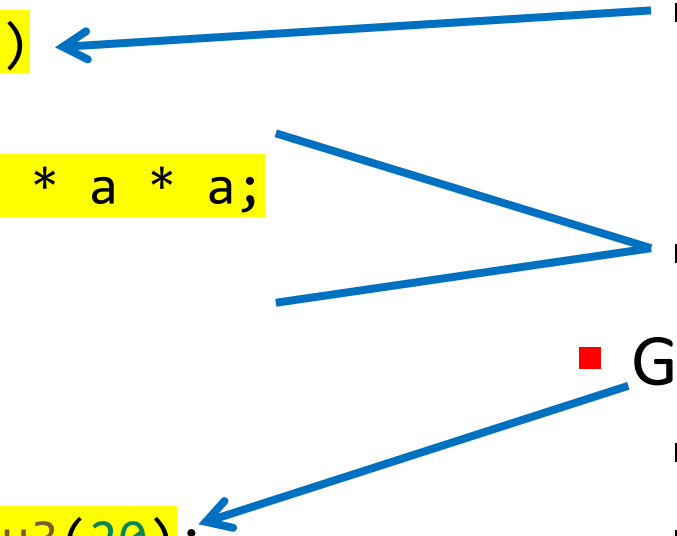
```
#include <iostream>

using namespace std;

// hàm mu3: tính a^3

int mu3(int a)
{
    int b = a * a * a;
    return b;
}

int main() {
    cout << mu3(20);
}
```



- Đã học trong Nhập môn Lập trình
- Định nghĩa hàm (function definition) gồm 2 phần:
  - Phần khai báo (function declaration / function prototype)
  - Phần thân (function body)
- Gọi hàm:
  - Thông qua tên
  - Truyền đối số phù hợp

Cấu chúc chung của hàm

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

// hàm mu3: tính a^3
int mu3(int a);

int main() {
    cout << mu3(20);
}

int mu3(int a) {
    int b = a * a * a;
    return b;
}
```

- Phần khai báo hàm có thể tách riêng
  - Thường viết ở phần đầu của file hoặc tách riêng thành một file (gọi là file header)
- Có thể gọi hàm ngay cả khi chưa biết thân hàm viết thế nào
- Phần định nghĩa hàm viết đầu đủ sau đó

Cấu chúc chung của hàm

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

// hàm mu3: tính a^3
int mu3(int);

int main() {
    cout << mu3(20);
}

int mu3(int a) {
    int b = a * a * a;
    return b;
}
```

- Phần khai báo hàm không cần viết tên tham số
  - Vẫn phải viết kiểu trả về và tên hàm. Riêng phần tham số chỉ cần viết kiểu và bỏ qua phần tên
- Có thể gọi hàm ngay cả khi chưa biết thân hàm viết thế nào
- Phần định nghĩa hàm viết đầu đủ sau đó

Cấu chúc chung của hàm

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

// hàm mu3: tính x^3
int mu3(int x);

int main() {
    cout << mu3(20);
}

int mu3(int a) {
    int b = a * a * a;
    return b;
}
```

- Phần khai báo hàm không cần viết tên tham số
  - Thậm chí tên tham số ở trên viết một đằng ở dưới viết một nẻo vẫn được chấp nhận
- Có thể gọi hàm ngay cả khi chưa biết thân hàm viết thế nào
- Phần định nghĩa hàm viết đầu đủ sau đó

Cấu trúc của một chương trình C/C++

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

const int MAX = 100;
double PI = 3.1415;
int mu3(int);

int main() {
    cout << mu3(20);
}

int mu3(int a) {
    int b = a * a * a;
    return b;
}
```

- Tiền xử lý:
  - #include
  - #define
- Khai báo, định nghĩa:
  - Hằng số
  - Biến
  - Nguyên mẫu hàm
  - Nguyên mẫu lớp
- Mã chính:
  - Các hàm
  - Các lớp

Quy tắc

- Khai báo hàm: cung cấp thông tin nguyên mẫu của hàm
  - Mô tả đủ thông tin để có thể phát lời gọi hàm
  - Phải viết trước bất kỳ lời gọi hàm nào
  - Phải có kiểu trả về của hàm
  - Phải có tên hàm
  - Phải có kiểu của từng tham số
  - Không nhất thiết phải có tên tham số
  - `double mu_x(int, double);`
  - `double mu_x(int a, double d);`
- Gọi hàm: gọi tên hàm và các đối số cần thiết
  - `d = mu_x(3, 0.5);`
  - Những giá trị thực sự được dùng trong lời gọi hàm được gọi là đối số (argument) hoặc tham số thực (actual parameter)

Quy tắc

- Định nghĩa hàm: viết đầy đủ cả phần khai báo và phần thân hàm
  - Tất nhiên phải viết đầy đủ tên các tham số (parameter) để có thể sử dụng được chúng trong phần thân hàm
  - Còn gọi là các tham số hình thức (formal parameter)
  - Trả về kết quả thông qua lệnh `return`
  - Nếu hàm không có kết quả tính toán (chẳng hạn hàm in N số ra màn hình), thì khai báo kiểu `void` và không cần `return` nữa
  - ```
    double mu_x(int a, double d) {
        double k = 1;
        for (int i = 0; i < a; i++)
            k *= d;
        return k
    }
    ```
    ```

Thảo luận

- Mục đích của việc sử dụng hàm?
  - Tái sử dụng: Mã được viết một lần, sử dụng nhiều lần
  - Giảm chi phí: Sửa lỗi, nâng cấp ở một đoạn mã
  - Dễ phát triển: Chia chương trình phức tạp thành nhiều đơn thể, giảm độ phức tạp khi viết các khối mã
- Tại sao phải tách phần nguyên mẫu và phần thân hàm?
  - Tập trung vào các chức năng
  - Phát triển song song
- Hàm: hiện thực hóa ý tưởng "trừu tượng hóa chức năng" (functional abstraction)
  - Người dùng chỉ cần biết đến chức năng của mã
  - Người dùng không cần quan tâm đến chi tiết của mã

Phần 2

Hiểu về cách hàm hoạt động

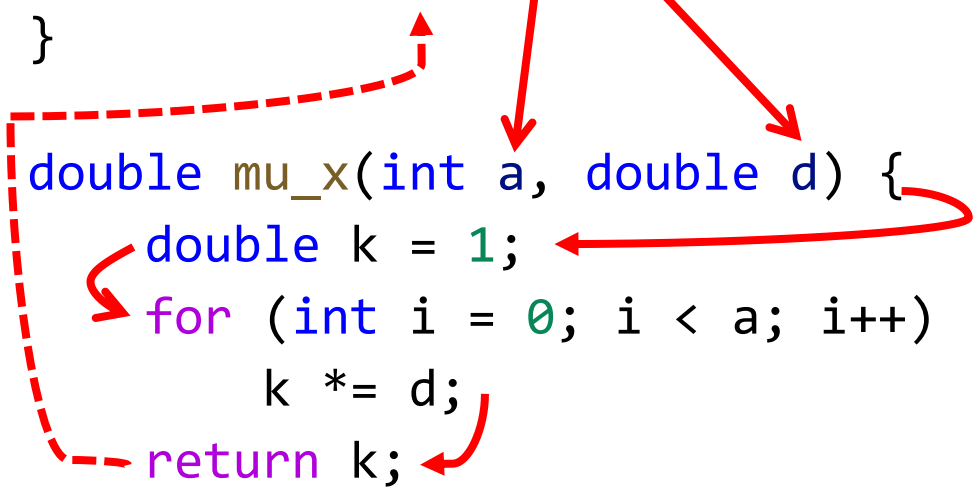
Hiểu về cách hàm hoạt động

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

double mu_x(int, double);

int main() {
    cout << mu_x(3, 0.5);
}

double mu_x(int a, double d) {
    double k = 1;
    for (int i = 0; i < a; i++)
        k *= d;
    return k;
}
```



① Phát lời gọi hàm

② Gán giá trị thực cho tham số: a = 3, d = 0.5

③ Vào thân hàm

④ Khai báo biến k = 1

⑤ Thực hiện vòng `for`

⑥ Trả về kết quả qua lệnh `return` và thoát khỏi hàm

Vấn đề: hàm không thể thay đổi giá trị của đối số

Phần 3

Các hàm có sẵn

Các hàm có sẵn

- Các hàm có sẵn do các lập trình viên khác viết ra và cung cấp cho chúng ta sử dụng
  - Hàm chuẩn của C/C++ đi kèm với trình biên dịch
  - Hàm do các đồng nghiệp trong cùng dự án viết cho chúng ta
- Cung cấp ở dạng thư viện, chỉ việc khai báo và sử dụng
  - Thư viện thường gồm 2 loại file:
    - File header: chỉ chứa các khai báo hàm (.h, .hpp hoặc không đuôi)
    - File source: chứa phần thân hàm (.c, .cpp)
  - Khai báo thư viện thông qua phát biểu #include
  - Phát biểu #include phải chỉ ra file header sẽ sử dụng
  - `#include <iostream>` ← tìm file trong thư mục chuẩn
  - `#include "mylib"` ← tìm file trong thư mục hiện tại

Các hàm có sẵn

Tên	Mô tả	Kiểu đối số	Kiểu giá trị trả về	Ví dụ	Giá trị	Thư viện
sqrt	Căn bậc hai	double	double	sqrt(4.0)	2.0	cmath
pow	Lũy thừa	double	double	pow(2.0,3.0)	8.0	cmath
abs	Giá trị tuyệt đối kiểu int	int	int	abs(-7) abs(7)	7 7	cstdlib
labs	Giá trị tuyệt đối kiểu long	long	long	labs(-70000) labs(70000)	70000 70000	cstdlib
fabs	Giá trị tuyệt đối kiểu double	double	double	fabs(-7.5) fabs(7.5)	7.5 7.5	cmath

Các hàm có sẵn

Tên	Mô tả	Kiểu đối số	Kiểu giá trị trả về	Ví dụ	Giá trị	Thư viện
ceil	Làm tròn lên	double	double	ceil(3.2) ceil(3.9)	4.0 4.0	cmath
floor	Làm tròn xuống	double	double	floor(3.2) floor(3.9)	3.0 3.0	cmath
exit	Kết thúc chương trình	int	void	exit(1);	Không có	cstdlib
rand	Số ngẫu nhiên	Không có	int	rand( )	Thay đổi	cstdlib
srand	Thiết lập hạt giống cho rand	unsigned int	void	srand(42);	Không có	cstdlib

Tạo số ngẫu nhiên

- Một trong những vấn đề thú vị nhất
  - Tạo các tình huống ngẫu nhiên trong chương trình, trò chơi
  - Tạo các biến ngẫu nhiên trong tính toán khoa học
  - Chỉ là giả-ngẫu-nhiên
- Một số kĩ thuật lưu ý:
  - `rand()`: trả về giá trị nguyên giữa 0 & RAND_MAX
    - RAND_MAX tùy thuộc vào từng thư viện và trình biên dịch
  - Thu hẹp phạm vi: `rand() % 6`
    - Trả về số ngẫu nhiên giữa 0 & 5
  - Tịnh tiến: `rand() % 6 + k`
    - Trả về số ngẫu nhiên giữa k & 5+k
  - Số thực ngẫu nhiên: `rand() / (double) RAND_MAX`
  - Khởi tạo nhân cho việc tạo số ngẫu nhiên: `srand(time(0))`

Phần 4

Phạm vi của biến và của hàm

Quy tắc

- Biến chỉ có thể truy cập sau khi đã khai báo
- Biến được khai báo bên trong khối nào (giữa cặp ngoặc {} nào) thì chỉ được truy cập bên trong khối đó
  - Biến không nằm trong bất kỳ cặp ngoặc nào: biến toàn cục (global variable)
  - Biến nằm trong hàm: biến cục bộ (local variable)
- Biến tham số của hàm được sử dụng trong hàm
- Biến global:
  - Có thể truy cập từ bất kỳ đâu trong chương trình
  - Chú ý: một chương trình có thể gồm nhiều file
  - Có thể truy cập biến ở trong file khác: từ khóa `extern`
  - Rất cẩn thận khi sử dụng

Từ khóa `static`

- Từ khóa này có thể dụng cho cả hàm, biến toàn cục và biến cục bộ
- Dùng với hàm: quy định rằng hàm này chỉ được dùng trong phạm vi của file hiện tại
- Dùng với biến toàn cục: quy định rằng biến này chỉ được truy cập trong phạm vi của file hiện tại
- Dùng với biến cục bộ: quy định rằng biến này là "tĩnh", không bị hủy đi khi kết thúc hàm
  - Chỉ khởi tạo một lần
  - Sống theo vòng đời của chương trình, không bị hủy với hàm
  - Hàm lần sau sử dụng giá trị còn tồn lại từ lần gọi trước

Ví dụ về quy tắc phạm vi

```
// biến toàn cục, truy cập từ bất kỳ đâu, kể cả từ file khác
long long g = 0;
// biến toàn cục, được khai báo ở file khác
extern double d;
// biến module, chỉ được truy cập từ đoạn mã cùng file
static int c = 1;
// hàm toàn cục, có thể gọi từ bất kỳ đâu, kể cả từ file khác
void change(int a) {
    static int x = 0;   // biến module, nhưng phạm vi hàm
    ...
// hàm module, chỉ gọi được từ đoạn mã cùng file
static void dosmt(bool a) {
    bool b = true;      // biến cục bộ, phạm vị hàm
    ...
// hàm toàn cục, định nghĩa hàm được viết ở file khấc
extern int somefuction(int n);
```

Ví dụ về biến static, hãy chạy thử xem nào!

```
#include <iostream>

using namespace std;

void change(int a) {
    static int x = 0;
    cout << "X = " << ++x << endl;
    a = 10;
}

int main() {
    int b = 5;
    change(b);
    change(b);
    change(b);
}
```

Phần 5

Truyền tham số trong hàm

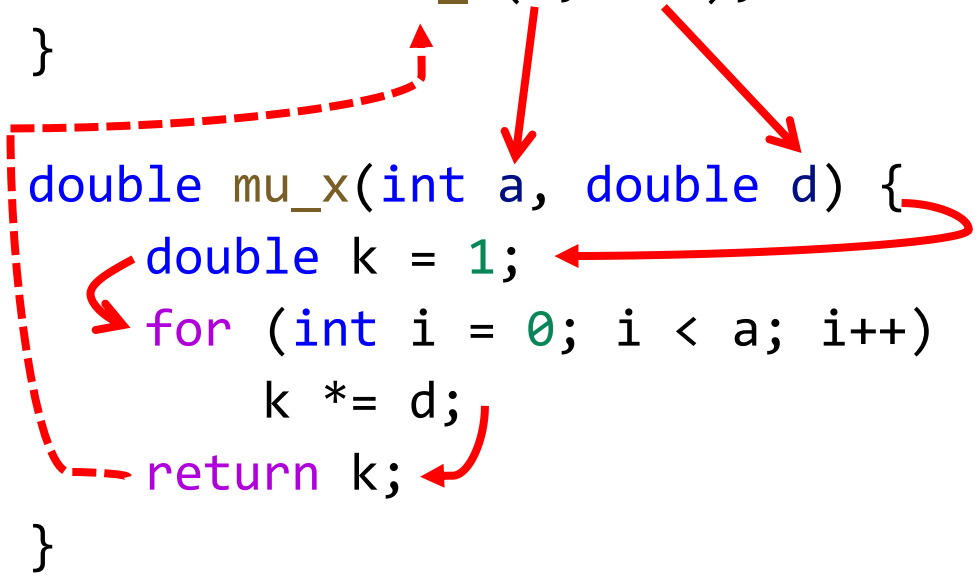
Trở lại ví dụ về cách hàm hoạt động

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

double mu_x(int, double);

int main() {
    cout << mu_x(3, 0.5);
}

double mu_x(int a, double d) {
    double k = 1;
    for (int i = 0; i < a; i++)
        k *= d;
    return k;
}
```



① Phát lời gọi hàm

② Gán giá trị thực (đối số) cho tham số: a = 3, d = 0.5

③ Khai báo biến k = 1

④ Thực hiện vòng `for`

⑤ Trả về kết quả qua lệnh `return` và thoát khỏi hàm

Vấn đề: Biến a và d là biến của hàm, việc thay đổi giá trị chỉ có tác dụng nội bộ

Một ví dụ rõ ràng hơn

```
#include <iostream>

using namespace std;

void change(int a) {
    a = 10;
    cout << "a = " << a << endl;
}

int main() {
    int b = 5;
    cout << "b = " << b << endl;
    change(b);
    cout << "b = " << b << endl;
}
```

① In ra b (b = 5)

② Phát lời gọi hàm

③ Gán giá trị thực (đối số) cho tham số: a = b (a = 5)

④ In ra a (a = 10)

⑤ Thoát khỏi hàm

⑥ In ra b (b = 5)

Việc thay đổi biến a trong hàm không tác động gì đến biến b ở hàm chính

Cách giải quyết: tham chiếu (&)

- Đây là kiến thức có trong môn học Nhập môn Lập trình
- Biến tham chiếu:
  - Alias (nickname) của một biến khác
  - Khai báo như biến, nhưng thêm dấu & vào trước tên biến
  - Tác động vào tham chiếu cũng như tác động vào biến
- Ví dụ:

```
int x;                      // biến x
int & y = x, z = y;         // tham chiếu y, biến z
x = 10;                     // y = 10
z = 9                       // x và y vẫn là 10
y = 20;                     // x = 20
```

Ví dụ viết lại bằng tham chiếu

```
#include <iostream>

using namespace std;

void change(int & a) {
    a = 10;
    cout << "a = " << a << endl;
}

int main() {
    int b = 5;
    cout << "b = " << b << endl;
    change(b);
    cout << "b = " << b << endl;
}
```

① In ra b (b = 5)

② Phát lời gọi hàm, gán tham số: a = b (a ≈ b)

③ Thay đổi a = 10 (b = 10)

④ In ra a (a = 10)

⑤ Thoát khỏi hàm

⑥ In ra b (b = 10)

Muốn tham số bị thay đổi giá trị trong hàm: khai báo nó ở dạng tham chiếu

Tham chiếu có ưu điểm và có điểm bất tiện

```
#include <iostream>

using namespace std;

void change(int & a) {
    a = 10;
    cout << "a = " << a << endl;
}

int main() {
    int b = 5;
    cout << "b = " << b << endl;
    change(b);
    cout << "b = " << b << endl;
    change(100);    // lỗi
}
```

- Tiết kiệm bộ nhớ (biến tham chiếu luôn có kích thước 4 byte – tùy OS)
- Nhanh (vì kích thước nhỏ)
- Chỉ sử dụng được với biến, không làm việc với dữ liệu trực trị (giá trị viết trực tiếp vào đối số)
- Trả về tham chiếu đến biến cục bộ có thể gây những lỗi bộ nhớ

Quy tắc chung

- Chú ý: Những quy tắc này không phải luôn luôn đúng
  - Muốn thay đổi giá trị đối số thì hãy sử dụng tham chiếu (thêm dấu & vào trước tên biến)
  - Muốn giảm bớt yêu cầu bộ nhớ và tăng tốc độ của hàm cũng có thể sử dụng tham chiếu
  - Muốn ngăn chặn việc vô ý thay đổi dữ liệu tham chiếu thì thêm từ khóa const vào trước tham chiếu
- Ví dụ:

```
// dùng khi cần thay đổi d
int change(string & d)
// dùng khi cần hàm nhanh hơn
int change(string & d)
// dùng khi cần hàm nhanh hơn và không thay đổi d
int change(const string & d)
```

Phần 6

Nạp chồng hàm

Khái niệm

- Nạp chồng hàm (function overloading) chỉ việc có thể viết nhiều hàm cùng tên nhau trong một chương trình
- Ví dụ: ta viết ba hàm đều tên là trungbinh dùng để tính trung bình cộng của 2, 3 hoặc 4 số thực

```
double trungbinh(double a, double b) {
    ...
}

double trungbinh(double a, double b, double c) {
    ...
}

double trungbinh(double a, double b, double c, double d) {
    ...
}
```

Khái niệm

- Tất nhiên ta có thể đặt tên hàm khác đi, chẳng hạn như trungbinh2, trungbinh3 và trungbinh4
  - Nhưng cách làm này có tính đối phó
  - Làm mất ý nghĩa của tên hàm: trungbinh2 có thể hiểu là trung bình bình phương?
- Một số ngôn ngữ lập trình không cho hàm trùng tên
- C/C++ thì việc này là được phép: trình dịch sẽ tự tìm ra hàm hợp lý nhất trong số các hàm trùng tên
  - Dựa trên số lượng tham số của hàm
  - Dựa trên kiểu dữ liệu của các tham số của hàm
  - Không dựa trên kết quả trả về của hàm
  - Cơ chế này gọi là tự động phân giải nạp chồng (automatic overload resolution)

Nạp chồng hàm: tình huống đơn giản

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

void print(int a, int b)    { cout << "0" << endl; }
void print(int a, double b) { cout << "1" << endl; }
void print(double a, int b) { cout << "2" << endl; }

int main() {
    print(10, 20);      // print(int, int)
    print(0.5, 100);    // print(double, int)
    print(5, 0.5);      // print(int, double)
    print(1.5, 0.5);    // Lỗi
}
```

Nạp chồng hàm: tình huống chuyển đổi kiểu

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

void print(long a, long b)     { cout << "0" << endl; }
void print(long a, double b)   { cout << "1" << endl; }
void print(double a, long b)   { cout << "2" << endl; }
void print(double a, double b) { cout << "3" << endl; }

int main() {
    print(10, 20);       // Lỗi
    print(0.5, 100L);    // print(double, long)
    print(5L, 0.5);      // print(long, double)
    print(1.5, 0.5);     // print(double, double)
}
```

Nạp chồng hàm: tham số mặc định

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

double area(double dai, double rong = 1) {
    return dai * rong;
}

int main() {
    cout << area(10, 20) << endl;   // dai = 10, rong = 20
    cout << area(10) << endl;       // dai = 10, rong = 1
}
```

Nạp chồng hàm giúp thiết kế linh hoạt hơn

```
// Nhập n rồi nhập mảng a
void nhap(int & n, int a[]) {

}
// Nhập mảng a có n phần tử (n biết trước)
void nhap(int a[], int n) {

}
// Nhập n, nhập mảng a rồi trả n về
int nhap(int a[]) {

}
```

Phần 7

Hàm đệ quy

Khái niệm đệ quy

- Hàm đệ quy = Hàm trong lúc thực thi có gọi lại chính nó
  - Đệ quy trực tiếp: gọi lại chính nó ngay trong thân hàm
  - Đệ quy gián tiếp: gọi lại chính nó thực hiện trong các hàm con
- Một số ngôn ngữ lập trình không cho phép đệ quy
- Phù hợp với lối suy nghĩ top-down (từ trên xuống), rất phổ biến trong toán học, tin học, vật lý,...
- Chương trình viết rất ngắn
- Khá giống với tư duy quy nạp:
  - Giải bài toán với trường hợp nhỏ nhất
  - Giải bài toán lớn dựa trên lời giải từ bài toán con
- Chạy chậm!!!

Ví dụ: tính n!

- Công thức toán: n! = 1 x 2 x ... x (n-1) x n
- Trường hợp n = 1: n! = 1
- Trường hợp n > 1: biến đổi công thức
  - n! = 1 x 2 x ... x (n-1) x n = (1 x 2 x ... x (n-1)) x n = (n-1)! x n
- Viết mã:

```
// phiên bản dài dòng
int giaithua(int n) {
    if (n == 1) return 1;
    return giaithua(n - 1) * n;
}
// phiên bản ngắn hơn
int gt(int n) {
    return n == 1 ? 1 : gt(n - 1) * n;
}
```

Hàm đệ quy thực hiện như thế nào?

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

int gt(int n) {
    if (n == 1) return 1;
    return gt(n - 1) * n;
}

int main() {
    cout << gt(4);
}
```

① Phát lời gọi gt(4)

② Vào hàm gt(n=4)

③ Bỏ qua lệnh if

④ Tính công thức gt(3)*4

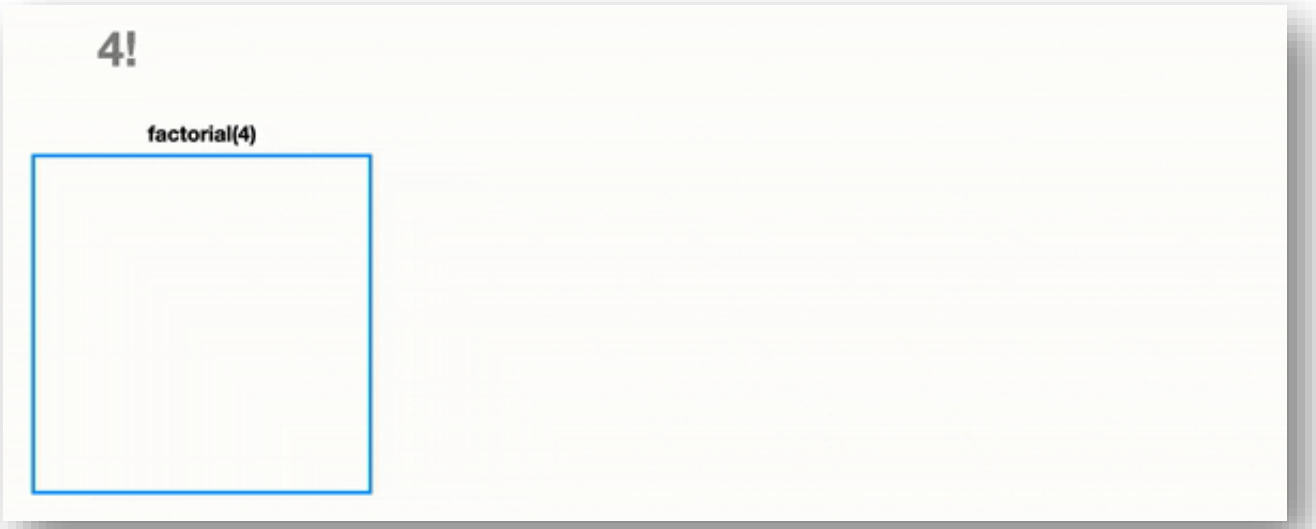
- ⑤ Vào hàm gt(n=3)
- ⑥ Bỏ qua lệnh if
- ⑦ Tính công thức gt(2)*3
  - ⑧ Vào hàm gt(n=2)
  - ⑨ Bỏ qua lệnh if
  - ⑩ Tính công thức gt(1)*2
    - ⑪ Vào hàm gt(n=1)
    - ⑫ Chạy lệnh if, trả về 1

Hàm đệ quy thực hiện như thế nào?

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

int gt(int n) {
    if (n == 1) return 1;
    return gt(n - 1) * n;
}

int main() {
    cout << gt(4);
}
```



Phần 8

Bài tập

Bài tập

1. Viết khai báo các hàm sau:
    1. In các số nguyên từ a đến b
    2. Kiểm tra xem a và b có phải nguyên tố cùng nhau không?
    3. Tính tổng N số đầu tiên dãy số thực A
    4. Sắp xếp N phần tử đầu tiên của dãy số nguyên A
    5. Tìm vị trí của phần tử có giá trị nhỏ nhất trong dãy A
    6. Sinh ra một chuỗi ngẫu nhiên
2. Nhập số N và in ra màn hình đúng N số thực ngẫu nhiên trong khoảng 0..1
3. Viết phần mềm gieo xúc sắc: mỗi khi người dùng nhập "next" sẽ gieo và trả về giá trị ngẫu nhiên từ 1 đến 6. Người dùng nhập "quit" sẽ thoát chương trình

Bài tập

4. Viết hàm sort nhận ba số thực a, b và c; hàm thực hiện sắp xếp lại các giá trị này theo thứ tự giảm dần
5. Viết hàm thaythe nhận tham số là số phần tử n, mảng a và hai số thực x y. Hàm sẽ tìm các giá trị x trong mảng a có n phần tử và thay thế bằng giá trị y
6. Viết phần thân của ba hàm thuộc slide 37.
7. Viết hàm bin nhận tham số là một số tự nhiên n và in ra màn hình dạng nhị phân của n.
8. Viết hàm bin nhận tham số là một số tự nhiên n và in ra màn hình dạng nhị phân của n. Yêu cầu: không sử dụng vòng lặp.

Bài tập

9. Viết hàm tính tổ hợp chập k của n (C_n^k)
10. Cho hai điểm P và Q trên đường thẳng là tọa độ tâm của hai hình tròn có bán kính là R và C (hình minh họa ở dưới). Hãy tính diện tích phần giao của hai hình tròn với độ chính xác 6 số thập phân.