

BAN HỌC TẬP CÔNG NGHỆ PHẦN MỀM

TRAINING CUỐI KỲ HỌC KỲ I NĂM HỌC 2022 – 2023



Sharing is learning



 **BAN HỌC TẬP**

Khoa Công nghệ Phần mềm

Trường Đại học Công nghệ Thông tin

Đại học Quốc gia thành phố Hồ Chí Minh

 **CONTACT**

bht.cnpm.uit@gmail.com

fb.com/bhtcnpm

fb.com/groups/bht.cnpm.uit

TRAINING

NHẬP MÔN LẬP TRÌNH

 **Thời gian:** 7:30 Chủ Nhật ngày 12/02/2022

 **Địa điểm:** Microsoft Teams

 **Trainers:** Huỳnh Lê Đan Linh – KTMP2022.2

Nguyễn Việt Khoa – KTPM2022.2



Sharing is learning

TABLE OF CONTENTS

- | | |
|---|----------------------------|
| I. Các kiểu dữ liệu cơ sở và phép toán | VI. Mảng một chiều |
| II. Cấu trúc điều khiển | VII. Mảng hai chiều |
| III. Một số dạng bài cơ bản | VIII. Kiểu cấu trúc |
| IV. Hàm và cấu trúc chương trình | IX. Con trỏ |
| V. Đệ quy | X. Chuỗi |



I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

1. Các kiểu dữ liệu cơ sở - Kiểu số nguyên

Kiểu	Kích thước	Vùng giá trị
int	2 hoặc 4 bytes	-32,768 tới 32,767 hoặc -2,147,483,648 tới 2,147,483,647
unsigned int	2 hoặc 4 bytes	0 tới 65,535 hoặc 0 tới 4,294,967,295
short	2 bytes	-32,768 tới 32,767
unsigned short	2 bytes	0 tới 65,535
long	4 bytes	-2,147,483,648 tới 2,147,483,647
unsigned long	4 bytes	0 tới 4,294,967,295

I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

1. Các kiểu dữ liệu cơ sở - Kiểu số nguyên

Kiểu	Kích thước	Vùng giá trị
char	1 byte	-128 tới 127 hoặc 0 tới 255
(unsigned) char	1 byte	0 tới 255
(signed) char	1 byte	-128 tới 127

Char vừa là kiểu số nguyên (mã ASCII), cũng vừa là kiểu ký tự.



I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

1. Các kiểu dữ liệu cơ sở - Kiểu số thực

Kiểu	Kích thước	Vùng giá trị	Độ chính xác
float	4 byte	1.2E-38 tới 3.4E+38	6 chữ số
double	8 byte	2.3E-308 tới 1.7E+308	15 chữ số
long double	10 byte	3.4E-4932 tới 1.1E+4932	19 chữ số



I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

1. Các kiểu dữ liệu cơ sở - Kiểu Boolean

- Chỉ nhận một trong hai giá trị **true** (đúng) hoặc **false** (sai) tương ứng với kết quả của mệnh đề toán học trong C++.
- Khi biểu diễn giá trị của biến kiểu bool trên máy tính, giá trị **true** ứng với số **1**, giá trị **false** ứng với số **0**.



I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

1. Các kiểu dữ liệu cơ sở - Kiểu Boolean

- Các cách khai báo biến kiểu bool:

```
1. bool a = true;  
2. bool b(false);  
3. bool c { true };  
4. bool d = !true; //false
```


I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

1. Các kiểu dữ liệu cơ sở - Kiểu Boolean

- Gán các mệnh đề toán học cho biến kiểu bool:

```
1. bool b1 = 1 < 2; //1
2. bool b2 = 5 > 10; //0
3. bool b3 = (1 + 1 == 2); //1
4. cout << b1 << " " << b2 << " " << b3;
```

- Đoạn lệnh trên sẽ cho ra kết quả: 1 0 1

I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

2. Biến

- Khai báo biến:
 - `<kiểu_dữ_liệu> <tên_biến>;`
- Khởi tạo giá trị cho biến:
 - `<kiểu_dữ_liệu> <tên_biến> = <giá_trị_khởi_tạo>;`
- Khai báo hằng:
 - `#define <tên_hằng> <giá_trị_hằng>`
 - `const <kiểu_dữ_liệu> <tên_biến> = <giá_trị_khởi_tạo>;`



I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

2. Biến

- Quy cách đặt tên:

- Tên phải bắt đầu bằng chữ cái hoặc dấu gạch dưới _. Ví dụ: Laptrinh1, _Laptrinh...
- Không được dùng những từ khóa hoặc tên hàm
- Tên không được có các toán tử.
- Hai biến trong cùng một hàm không được trùng tên.
- Tên biến không có dấu cách.



Sharing is learning

I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

3. Các phép toán - Toán tử gán

- Toán tử gán được kí hiệu bởi dấu "=", có tác dụng cập nhật giá trị cho một biến

```
1. int x=3;  
2. int x;  
3. int y = 3+(x=2); //y=5  
4. x=int z=4; //lỗi  
5. int z;  
6. z=4;
```

- Toán tử gán thực hiện từ trái sang phải, cần phân biệt toán tử gán và khởi tạo biến

I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

3. Các phép toán - Toán tử nhập, xuất

- Toán tử nhập có kí hiệu ">>", tác dụng nhập giá trị từ bàn phím để gán vào một biến
- Toán tử xuất có kí hiệu "<<", tác dụng in giá trị của biến ra màn hình

```
1. #include <iostream>
2. using namespace std;
3. int main(){
4.     int x;
5.     cin >> x;
6.     cout << x;
7.     return 0; }
```

I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

3. Các phép toán - Phép toán số học

Toán tử	Ý nghĩa
+	Cộng hai toán hạng
-	Trừ toán hạng thứ hai từ toán hạng đầu
*	Nhân hai toán hạng
/	Phép chia
%	Phép lấy số dư
++	Toán tử tăng (++), tăng giá trị toán hạng thêm một đơn vị
--	Toán tử giảm (--), giảm giá trị toán hạng đi một đơn vị

- Toán tử % chỉ áp dụng trên các toán hạng có kiểu số nguyên
- Kết quả của toán tử chia giữa 2 toán hạng có kiểu int là giá trị có kiểu int

I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

3. Các phép toán - Phép toán số học

- Cho `int A = 10; int B = 20;` là hai biến có kiểu giá trị int

```
1. int c;  
2. c = A + B; //c=30  
3. c = A - B; //c=-10  
4. c = A * B; //c=200  
5. c = A / B; //c=0  
6. c = A % B; //c=20  
7. c = A++; //c=10; A=11;  
8. c = --B; //B=19; c=19;
```

I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

3. Các phép toán - Phép toán logic

Toán tử	Ý nghĩa
&&	Được gọi là toán tử logic AND (và). Nếu cả hai toán tử đều có giá trị khác 0 thì điều kiện trở thành true.
	Được gọi là toán tử logic OR (hoặc). Nếu một trong hai toán tử khác 0, thì điều kiện là true.
!	Được gọi là toán tử NOT (phủ định). Sử dụng để đảo ngược lại trạng thái logic của toán hạng đó. Nếu điều kiện toán hạng là true thì phủ định nó sẽ là false.



I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

3. Các phép toán - Phép toán số học

- Cho `bool A = 1; bool B = 0;` là hai biến có kiểu giá trị boolean

```
1. bool c;  
2. c = A && B; //c=0  
3. c = A || B; //c=1  
4. c = !A; //c=0  
5. c = (2 && 3); //c=1  
6. c = (0 && 3); //c=0  
7. c = (0 || 0); //c=0  
8. c = (0 || 5); //c=1
```

I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

3. Các phép toán - Toán tử quan hệ

Toán tử	Ý nghĩa
==	Kiểm tra nếu 2 toán hạng bằng nhau hay không. Nếu bằng thì điều kiện là true.
!=	Kiểm tra 2 toán hạng có giá trị khác nhau hay không. Nếu không bằng thì điều kiện là true.
>	Kiểm tra nếu toán hạng bên trái có giá trị lớn hơn toán hạng bên phải hay không. Nếu lớn hơn thì điều kiện là true.
<	Kiểm tra nếu toán hạng bên trái nhỏ hơn toán hạng bên phải hay không. Nếu nhỏ hơn thì là true.
>=	Kiểm tra nếu toán hạng bên trái có giá trị lớn hơn hoặc bằng giá trị của toán hạng bên phải hay không. Nếu đúng là true.
<=	Kiểm tra nếu toán hạng bên trái có giá trị nhỏ hơn hoặc bằng toán hạng bên phải hay không. Nếu đúng là true.

I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

3. Các phép toán - Toán tử điều kiện

- `<điều_kiện> ? <biểu_thức_1> : <biểu_thức_2>;`
- Nếu `<điều_kiện>` đúng, kết quả là giá trị của `<biểu_thức_1>`
- Ngược lại, kết quả là giá trị của `<biểu_thức_2>`

```
1. char x;
```

```
2. x = (5 < 10) ? 'A' : 'B'; // x = 'A'
```



Sharing is learning

I. CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

3. Các phép toán - Toán tử điều kiện

- `<điều_kiện> ? <biểu_thức_1> : <biểu_thức_2>;`
- Nếu `<điều_kiện>` đúng, kết quả là giá trị của `<biểu_thức_1>`
- Ngược lại, kết quả là giá trị của `<biểu_thức_2>`

```
1. char x;
```

```
2. x = (5 < 10) ? 'A' : 'B'; // x = 'A'
```



II. CẤU TRÚC ĐIỀU KHIỂN

1. Cấu trúc rẽ nhánh if - else

- Cấu trúc này được dùng khi một lệnh hay một khối lệnh chỉ được thực hiện khi **một điều kiện** nào đó thoả mãn.

```
1. //Cấu trúc if đơn
2. if (<biểu_thức_điều_kiện>)
3. {
4.   khối lệnh; //chạy nếu điều kiện được thoả mãn
5. }
```

II. CẤU TRÚC ĐIỀU KHIỂN

1. Cấu trúc rẽ nhánh if - else

```
1. //Cấu trúc if - else:  
2. if (<biểu_thức_điều_kiện>)  
3. {  
4.     Khối lệnh 1; //Thực hiện nếu điều kiện thỏa  
5. }  
6. else  
7. {  
8.     Khối lệnh 2; //Thực hiện nếu điều kiện không thỏa  
9. }
```

II. CẤU TRÚC ĐIỀU KHIỂN

2. Cấu trúc rẽ nhánh switch - case

```
1. switch (<biểu_thức>)
2. {
3.     case <giá trị thứ 1>: <khối lệnh>;
4.         break;
5.     case <giá trị thứ 2>: <khối lệnh>;
6.         break;
7.     . . .
8.     default: <khối lệnh>;
9. }
```

II. CẤU TRÚC ĐIỀU KHIỂN

2. Cấu trúc rẽ nhánh switch – case

- Thực hiện khối lệnh tương ứng với case là kết quả của biểu thức. Switch – case không duyệt qua từng case mà nhảy thẳng đến case được chọn => nhanh hơn
- <biểu thức> phải là biểu thức số học nhận giá trị nguyên
- Giá trị thứ i ($i=1,2,3,...,n$) là các hằng số chọn tương ứng cho các nhánh chọn case khác nhau
- Nhánh default là nhánh lựa chọn mặc định khi không nhánh nào được chọn
- Nếu không có **break**; chương trình sẽ chạy toàn bộ các lệnh kể từ sau case được chọn.

II. CẤU TRÚC ĐIỀU KHIỂN

3. Cấu trúc lặp for

```
1. for (<biểu thức 1> ; <biểu thức 2> ; <biểu thức 3>)  
2. {  
3.     <khối lệnh>;  
4. }
```

- <biểu thức 1>: thiết lập ban đầu.
- <biểu thức 2>: điều kiện lặp, là biểu thức logic.
- <biểu thức 3>: thiết lập lại.
- Sau khi chạy khối lệnh, chạy <biểu thức 3>. Quá trình lặp dừng lại khi <biểu thức 2> sai.

II. CẤU TRÚC ĐIỀU KHIỂN

4. Cấu trúc lặp while, do - while

- Vòng lặp while được sử dụng để thực thi nhiều lần một đoạn chương trình, khi một điều kiện vẫn còn đúng, thường được sử dụng khi số lần lặp không được xác định trước (Không cố định).



II. CẤU TRÚC ĐIỀU KHIỂN

4. Cấu trúc lặp while, do - while

```
1. //while
2. while (<Điều kiện lặp>) {
3.     <khối lệnh>;
4. }
```

- <Điều kiện lặp> là biểu thức logic.
- Trong <khối lệnh> cần có câu lệnh tác động để <điều kiện lặp> sai đi => tránh vòng lặp vô tận.

II. CẤU TRÚC ĐIỀU KHIỂN

4. Cấu trúc lặp while, do - while

```
1. //do - while
2. do {
3.     <khối lệnh>;
4. } while (<Điều kiện lặp>;
```

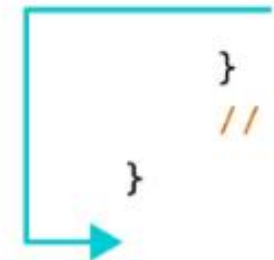
- do – while thực hiện <khối lệnh> ít nhất một lần bất kể đầu vào có đúng hay sai so với <Điều kiện lặp>

II. CẤU TRÚC ĐIỀU KHIỂN

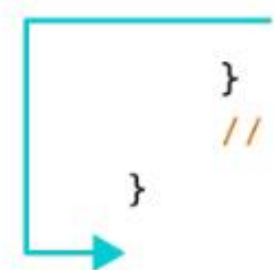
5. Câu lệnh **break**, **continue**

- **break** khiến chương trình lập tức thoát khỏi vòng lặp

```
for (init; condition; update) {  
    // code  
    if (condition to break) {  
        break;  
    }  
    // code  
}
```



```
while (condition) {  
    // code  
    if (condition to break) {  
        break;  
    }  
    // code  
}
```

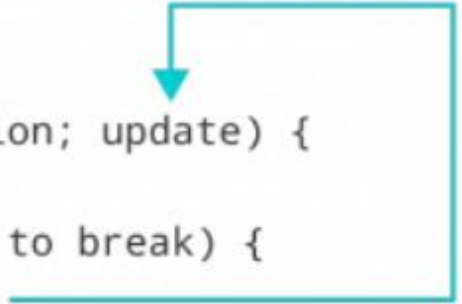


II. CẤU TRÚC ĐIỀU KHIỂN

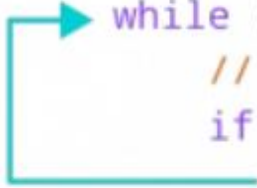
5. Câu lệnh **break**, **continue**

- **continue** khiến chương trình bỏ qua các dòng lệnh phía sau trong vòng lặp hiện tại và chạy vòng lặp kế tiếp

```
for (init; condition; update) {  
    // code  
    if (condition to break) {  
        continue;  
    }  
    // code  
}
```



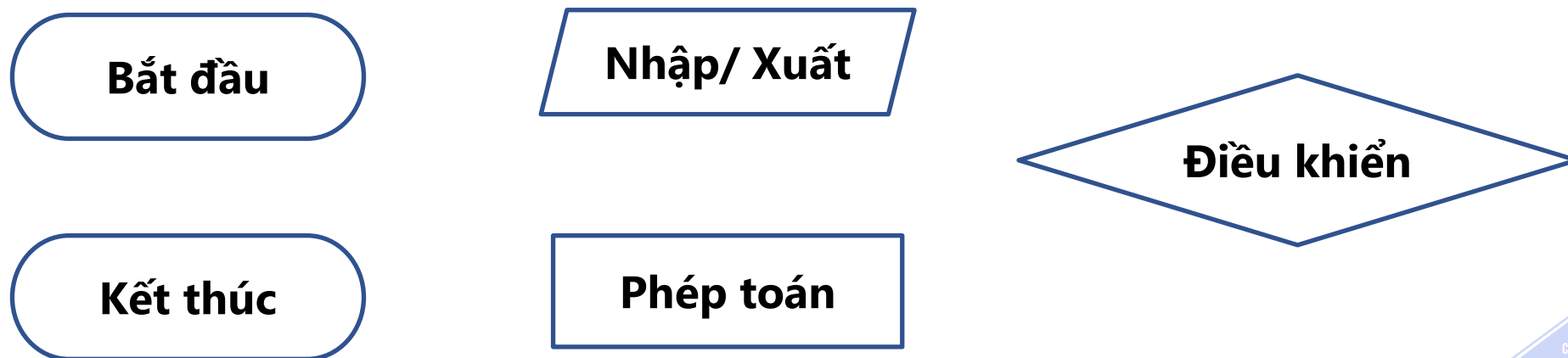
```
while (condition) {  
    // code  
    if (condition to break) {  
        continue;  
    }  
    // code  
}
```



III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

1. Lưu đồ thuật toán

- Lưu đồ thuật toán là công cụ dùng để biểu diễn thuật toán, mô tả nhập (input), dữ liệu xuất (output) và luồng xử lý thông qua các ký hiệu hình học.



III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

2. Một số kỹ thuật cơ bản

a. Cấu trúc điều khiển tuần tự: Dùng thuật toán để giải quyết một số bài tính toán có công thức

- Tính chu vi hình biết tọa độ đỉnh
- Tính diện tích hình biết tọa độ đỉnh
- Đổi độ C sang độ F
- Tính diện tích, chu vi hình tròn
- Đổi radian sang số đo độ

Ví dụ: Tính diện tích của một hình tròn tâm I (8, 8) và một điểm trên đường tròn là A (5, 3)

- Tính bán kính (khoảng cách A – I)
- Tính diện tích



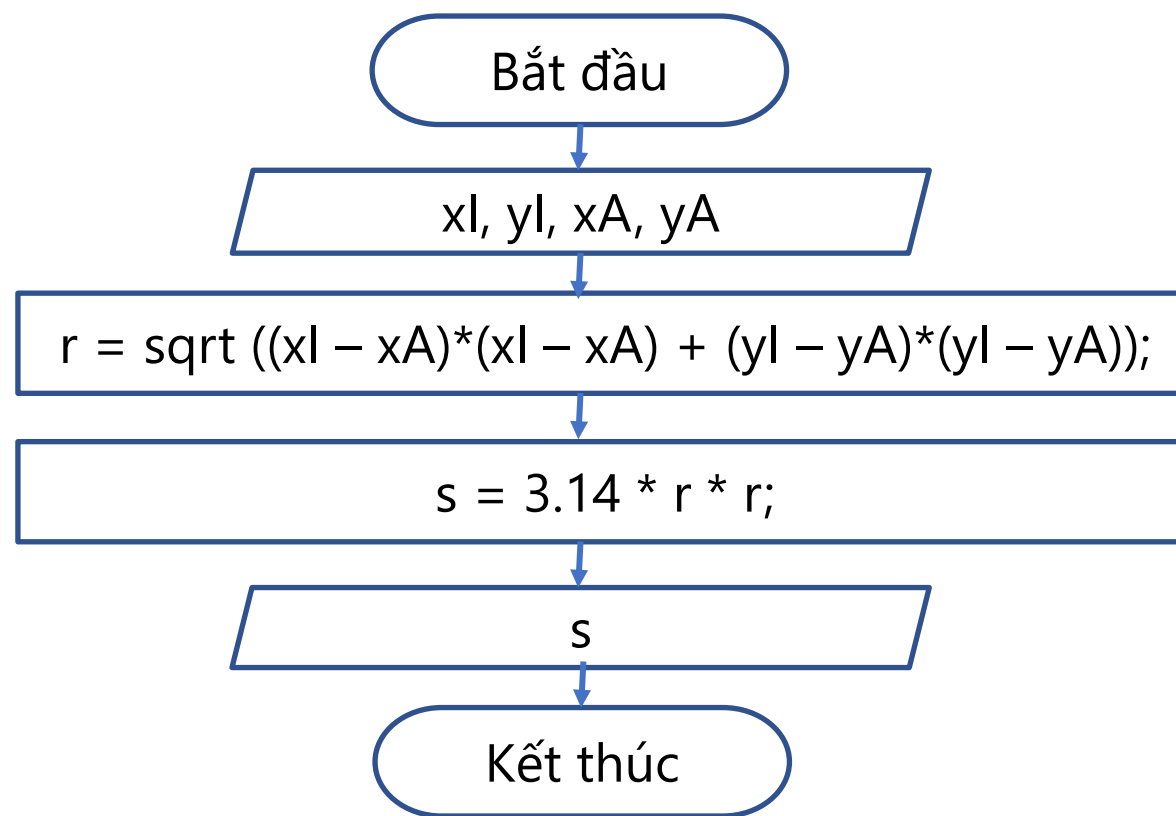
Sharing is learning

III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

2. Một số kỹ thuật cơ bản

- Cấu trúc điều khiển tuần tự: Dùng thuật toán để giải quyết một số bài tính toán có công thức
 - Tính chu vi hình biết tọa độ đỉnh
 - Tính diện tích hình biết tọa độ đỉnh
 - Đổi độ C sang độ F
 - Tính diện tích, chu vi hình tròn
 - Đổi radian sang số đo độ

Ví dụ: Tính diện tích của một hình tròn tâm I (8, 8) và một điểm trên đường tròn là A (5, 3)



III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

2. Một số kỹ thuật cơ bản

a. Cấu trúc điều khiển tuần tự: Dùng thuật toán để giải quyết một số bài tính toán có công thức

- Tính chu vi hình biết tọa độ đỉnh
- Tính diện tích hình biết tọa độ đỉnh
- Đổi độ C sang độ F
- Tính diện tích, chu vi hình tròn
- Đổi radian sang số đo độ

```
1. #include <iostream>
2. using namespace std;
3. int main() {
4.     float xI = 8;
5.     float yI = 8;
6.     float xA = 5;
7.     float yA = 3;
8.     float r = sqrt ((xI - xA)*(xI - xA)
9.                     + (yI - yA)*(yI - yA));
10.    float s = 3.14 * r * r;
11.    cout<<"Dien tich hinh tron la:"<<s<<endl;
12.    return 0;
13. }
```

III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

2. Một số kỹ thuật cơ bản

b. Tính toán tối ưu phép nhân: Dùng thuật toán để tính x^n với ít phép toán nhất

- Khuynh hướng giải chung: Tái sử dụng biến
- **Ví dụ: Tính giá trị x^{16} bằng ít phép toán nhất**

```
1. #include <iostream>
2. using namespace std;
3. int main() {
4.     int x;
5.     cin>>x;
6.     int x2 = x*x;
7.     int x4 = x2*x2;
8.     int x8 = x4*x4;
9.     int x16 = x8*x8;
10.    cout<<"x16: "<< x16 <<endl;
11.    return 0;
12. }
```

III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

2. Một số kỹ thuật cơ bản

c. Chữ số hàng đơn vị - chục – trăm: Lấy ra chữ số hàng đơn vị - chục – trăm của một số được nhập vào

- Khuynh hướng giải chung: Ứng dụng phép chia số nguyên cho số nguyên (kết quả: phần nguyên) và phép chia lấy dư (%)

Ví dụ: Lấy ra các phần tử hàng chục nghìn, hàng nghìn, hàng trăm của số 47298.

- Tạo một biến a chứa giá trị 47298
- Hàng trăm: Chia a cho 100, rồi gán kết quả vào b. Chữ số hàng trăm là kết quả của $b\%10$
- Hàng nghìn: Chia a cho 1000, rồi gán kết quả vào b. Chữ số hàng nghìn là kết quả của $b\%10$
- Hàng chục nghìn: Chia a cho 10000, rồi gán kết quả vào b. Chữ số hàng nghìn là giá trị của b

III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

2. Một số kỹ thuật cơ bản

c. Chữ số hàng đơn vị - chục – trăm: Lấy ra chữ số hàng đơn vị - chục – trăm của một số được nhập vào

- Khuynh hướng giải chung: Ứng dụng phép chia số nguyên cho số nguyên (kết quả: phần nguyên) và phép chia lấy dư (%)

Ví dụ: Lấy ra các phần tử hàng chục nghìn, hàng nghìn, hàng trăm của số 47298.

```
1. #include <iostream>
2. using namespace std;
3. int main() {
4.     int a = 47298; int b;
5.     b = a/100;
6.     cout<<"Hang tram: "<<b%10<<"\n";
7.     b = a/1000;
8.     cout<<"Hang nghin: "<<b%10<<"\n";
9.     b = a/10000;
10.    cout<<"Hang chuc nghin: "<<b<<"\n";
11. }
```

III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

2. Một số kỹ thuật cơ bản

d. Hoán vị: Kỹ thuật sử dụng biến trung gian

- Đổi chỗ hai biến a và b
- Khuynh hướng giải chung: Dùng một biến t tạm thời để chứa giá trị của một số a, thay đổi giá trị của a theo b và gán lại b theo t

Ví dụ: Đổi chỗ 2 số a = 8 và b = 9

```
1. #include <iostream>
2. using namespace std;
3. int main() {
4.     int a = 8; int b = 9;
5.     cout<<"a: "<<a<<" b: "<<b<<"\n";
6.     int t = a; //t=8
7.     a = b; //a=9
8.     b = t; //b=8;
9.     cout<<"a: "<<a<<" b: "<<b<<"\n";
10.    return 0;
11. }
```

III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

2. Một số kỹ thuật cơ bản

e. Dùng cấu trúc vòng lặp để tính toán

- Các tổng chuỗi có tính quy luật (tổng các lũy thừa tăng dần, tổng đan dấu...)
 - Các tích chuỗi có tính quy luật
 - Tính số hạng thứ n của dãy số
- Khuyňh hướng giải chung: Sử dụng biến i trong vòng lặp while/ for/ do...while... để tính giá trị của chuỗi và thay đổi toán hạng trong chuỗi.



III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

2. Một số kỹ thuật cơ bản

e. Dùng cấu trúc vòng lặp để tính toán

- Các tổng chuỗi có tính quy luật (tổng các lũy thừa tăng dần, tổng đan dấu...)
- Các tích chuỗi có tính quy luật
- Tính số hạng thứ n của dãy số

Ví dụ: $S(n) = 1*2*3*4*...*n$, Tính $S(10)$

- Xác định công thức tổng quát của chuỗi (biểu diễn S_n theo $S(n-1)$, $S(n-2)$...)
- Xác định giá trị khởi đầu, thông thường: tổng – 0, tích – 1
- Xác định giá trị kết thúc (giới hạn vòng lặp)
- Xác định bước nhảy (sau khi kết thúc vòng lặp thì giá trị lặp i tăng/ giảm bao nhiêu)

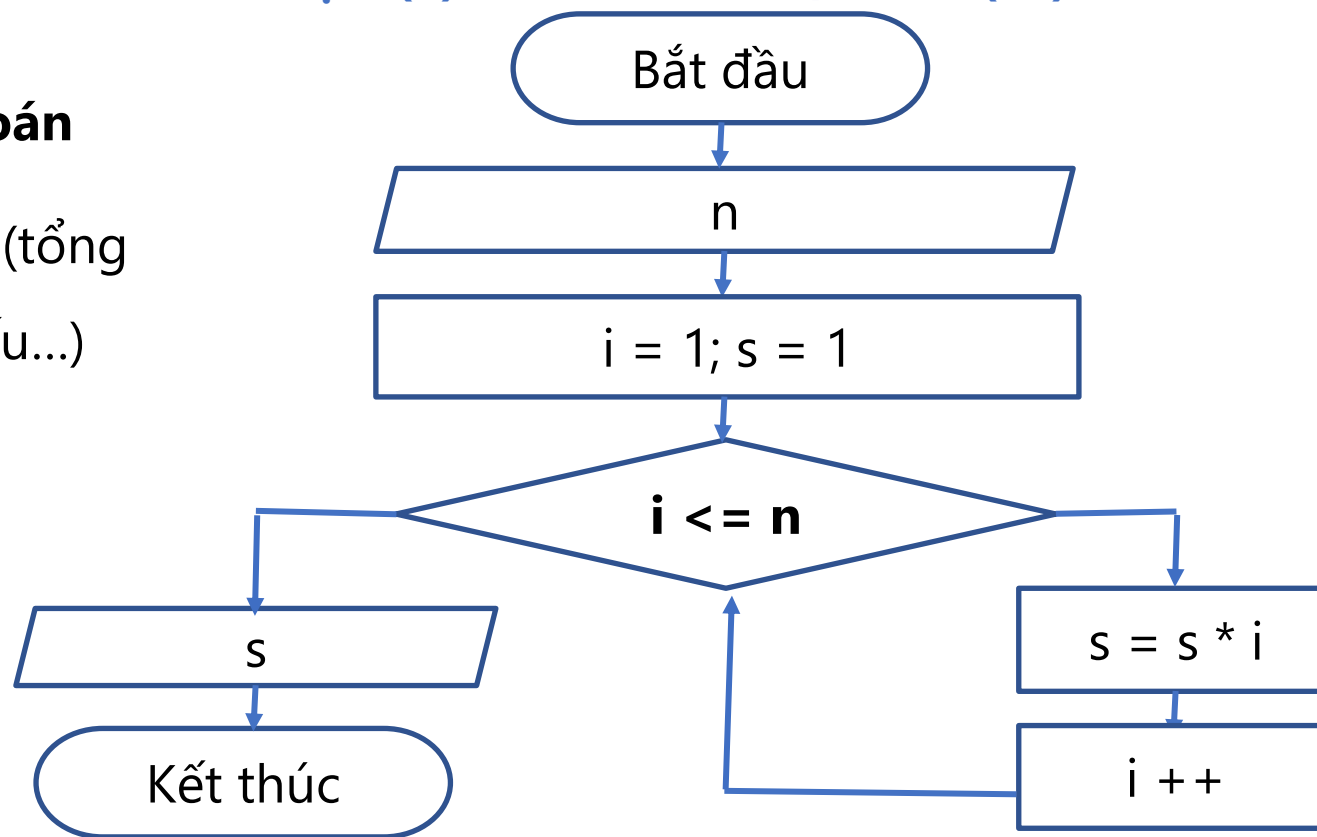
III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

2. Một số kỹ thuật cơ bản

e. Dùng cấu trúc vòng lặp để tính toán

- Các tổng chuỗi có tính quy luật (tổng các lũy thừa tăng dần, tổng đan dấu...)
- Các tích chuỗi có tính quy luật
- Tính số hạng thứ n của dãy số

Ví dụ: $S(n) = 1*2*3*4*...*n$, Tính $S(10)$



III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

2. Một số kỹ thuật cơ bản

e. Dùng cấu trúc vòng lặp để tính toán

- Các tổng chuỗi có tính quy luật (tổng các lũy thừa tăng dần, tổng đan dấu...)
- Các tích chuỗi có tính quy luật
- Tính số hạng thứ n của dãy số

Ví dụ: $S(n) = 1*2*3*4*...*n$, Tính $S(10)$

```
1. #include <iostream>
2. using namespace std;
3. int main() {
4.     int n;
5.     cin>>n;
6.     int s = 1;
7.     for (int i = 1; i<=n; i++)
8.         s = s * i;
9.     cout<<"S(" <<n<<" ) = "<<s;
10.    return 0;
11. }
```

III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

2. Một số kỹ thuật cơ bản

e. Dùng cấu trúc rẽ nhánh

- Kiểm tra ba độ dài cạnh a, b, c cho trước có tạo thành một tam giác hay không
 - Kiểm tra tam giác vuông
 - Kiểm tra lớn – bé – bằng
 - Kiểm tra năm nhuận
- Khuynh hướng giải chung: Dùng cấu trúc rẽ nhánh để kiểm tra điều kiện của biến, viết các phép toán ứng cho trường hợp điều kiện đúng hay sai
 - Xác định biến cần kiểm tra
 - Xác định điều kiện kiểm tra
 - Phép toán trong trường hợp đúng
 - Phép toán trong trường hợp sai



III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

2. Một số kỹ thuật cơ bản

f. Kỹ thuật đặt lính canh

- Tìm số lớn nhất/ bé nhất trong n số cho trước
- Khuynh hướng giải chung: Dùng một biến lính canh (lc) mang giá trị của một số bất kì trong các số đã cho, liên tục so sánh lc và các số còn lại, thay đổi biến lc khi thỏa điều kiện so sánh.

Ví dụ: Tìm số lớn nhất trong tập gồm 6 phần tử $A = \{121, 115, 12, 41, 123, 122\}$

```
1. #include <iostream>
2. using namespace std;
3. int main() {
4.     int A[6]={121, 115, 12, 41, 123, 122};
5.     int n = sizeof(A)/sizeof(int);
6.     int lc = A[0];
7.     for (int i = 1; i<n; i++)
8.         if (A[i]>lc) lc = A[i];
9.     cout<<"So lon nhat: " <<lc;
10.    return 0;
11. }
```

III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

2. Một số kỹ thuật cơ bản

g. Kỹ thuật đếm

- Đếm số chữ số của số nguyên dương
- Đếm số năm nhuận
- Khuynh hướng giải chung: Dùng một biến đếm (dem) với giá trị khởi đầu bằng 0 để lưu số đếm. Kiểm tra điều kiện của đề, nếu điều kiện đúng thì biến dem tăng thêm 1 lượng đơn vị

Ví dụ: Đếm số chữ số của số 8712311231

```
1. #include <iostream>
2. using namespace std;
3. int main() {
4.     long int a = 8712311231;
5.     int dem = 0;
6.     if (a==0) dem = 1;
7.     while (a!= 0){
8.         dem ++;
9.         a/=10; }
10.    cout<<"So chu so: " <<dem;
11.    return 0;
12. }
```

III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

2. Một số kỹ thuật cơ bản

h. Kỹ thuật đặt cờ hiệu

- Kiểm tra số nguyên tố
- Kiểm tra số đối xứng
- Khuynh hướng giải chung: Dùng một biến cờ hiệu (flag) với một giá trị khởi đầu. Kiểm tra điều kiện của đề, nếu đạt điều kiện thì thay đổi giá trị của flag, kiểm tra flag để đưa ra kết luận

Ví dụ: Kiểm tra tính đối xứng "abcdcba"

```
1. #include <iostream>
2. using namespace std;
3. int main() {
4.     char s[7] = {'a','b','c','d','c','b','a'};
5.     int flag = 1;
6.     for (int i = 0; i < (7/2); i++)
7.         if (s[i] != s[7 - i - 1]) flag = 0;
8.     if (flag == 0) cout<<"Khong doi xung";
9.     else cout<<"Doi xung";
10.    return 0;
11. }
```

III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

3. Một số dạng ứng dụng nhiều

- **Tìm số đảo ngược**
- Kiểm tra số hoàn thiện
- Kiểm tra số nguyên tố, số chính phương
- Kiểm tra số đối xứng
- Kiểm tra toàn lẻ/ toàn chẵn
- Ước chung lớn nhất/ bội chung nhỏ nhất
- Kiểm tra số có dạng a^n hay không (a: hằng số)

```
1.  #include <iostream>
2.  using namespace std;
3.  int main() {
4.      int n;
5.      cin>>n;
6.      int t = n;
7.      int dn = 0;
8.      while (t!= 0){
9.          dn = dn*10 + (t%10);
10.         t/=10; }
11.     cout<<dn;
12.     return 0;
13. }
```

III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

3. Một số dạng ứng dụng nhiều

- Tìm số đảo ngược
- **Kiểm tra số hoàn thiện**
- Kiểm tra số nguyên tố, số chính phương
- Kiểm tra số đối xứng
- Kiểm tra toàn lẻ/ toàn chẵn
- Ước chung lớn nhất/ bội chung nhỏ nhất
- Kiểm tra số có dạng a^n hay không (a: hằng số)

```
1.  #include <iostream>
2.  using namespace std;
3.  int main() {
4.      int n;
5.      cin>>n;
6.      int tongUoc = 0;
7.      for (int i = 1; i<=n/2; i++)
8.          if (n%i != 0) tongUoc+=i;
9.      if (tongUoc == n) cout<<1;
10.     else cout<<0;
11.     return 0;
12. }
```


III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

3. Một số dạng ứng dụng nhiều

- Tìm số đảo ngược
- Kiểm tra số hoàn thiện
- **Kiểm tra số nguyên tố**, số chính phương
- Kiểm tra số đối xứng
- Kiểm tra toàn lẻ/ toàn chẵn
- Ước chung lớn nhất/ bội chung nhỏ nhất
- Kiểm tra số có dạng a^n hay không (a: hằng số)

```
1.  #include <iostream>
2.  using namespace std;
3.  int main() {
4.      int n;
5.      cin>>n;
6.      bool kiểmtra = 1;
7.      for (int i = 2; i<n; i++){
8.          if (n<2 || n%i == 0 && i!=n){
9.              kiểmtra = 0;
10.             break; }
11.     }
12.     cout<< kiểmtra;
13.     return 0;
14. }
```

III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

3. Một số dạng ứng dụng nhiều

- Tìm số đảo ngược
- Kiểm tra số hoàn thiện
- Kiểm tra số nguyên tố, **số chính phương**
- Kiểm tra số đối xứng
- Kiểm tra toàn lẻ/ toàn chẵn
- Ước chung lớn nhất/ bội chung nhỏ nhất
- Kiểm tra số có dạng a^n hay không (a: hằng số)

```
1. #include <iostream>
2. using namespace std;
3. int main() {
4.     int n;
5.     cin>>n;
6.     int sqroot = sqrt(n);
7.     if (sqroot*sqroot == n)
8.         cout<<1;
9.     else cout<<0;
10.    return 0;
11. }
```

III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

3. Một số dạng ứng dụng nhiều

- Tìm số đảo ngược
- Kiểm tra số hoàn thiện
- Kiểm tra số nguyên tố, số chính phương
- **Kiểm tra số đối xứng - ứng dụng số đảo ngược**
- Kiểm tra toàn lẻ/ toàn chẵn
- Ước chung lớn nhất/ bội chung nhỏ nhất
- Kiểm tra số có dạng a^n hay không (a: hằng số)



III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

3. Một số dạng ứng dụng nhiều

- Tìm số đảo ngược
- Kiểm tra số hoàn thiện
- Kiểm tra số nguyên tố, số chính phương
- Kiểm tra số đối xứng
- **Kiểm tra toàn lẻ/ toàn chẵn**
- Ước chung lớn nhất/ bội chung nhỏ nhất
- Kiểm tra số có dạng a^n hay không (a: hằng số)

```
1. //toan le
2. #include <iostream>
3. using namespace std;
4. int main() {
5.     int n;
6.     cin>>n;
7.     bool kiểmtra = 1;
8.     do{
9.         if((n % 10) % 2 == 0){
10.             kiểmtra = 0;
11.             break; }
12.     }while(n/= 10);
13.     cout<< kiểmtra;
14.     return 0;
15. }
```

III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

3. Một số dạng ứng dụng nhiều

- Tìm số đảo ngược
- Kiểm tra số hoàn thiện
- Kiểm tra số nguyên tố, số chính phương
- Kiểm tra số đối xứng
- Kiểm tra toàn lẻ/ toàn chẵn
- **Ước chung lớn nhất/ bội chung nhỏ nhất**
- Kiểm tra số có dạng a^n hay không (a: hằng số)

```
1. //uoc chung lon nhat
2. #include <iostream>
3. using namespace std;
4. int main() {
5.     int a, b;
6.     cin>>a>>b;
7.     while (a*b != 0){
8.         if (a > b) a %= b;
9.         else b %= a;
10.    }
11.    cout<<"UCLN: "<<a+b;
12.    return 0;
13. }
```

III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

3. Một số dạng ứng dụng nhiều

- Tìm số đảo ngược
- Kiểm tra số hoàn thiện
- Kiểm tra số nguyên tố, số chính phương
- Kiểm tra số đối xứng
- Kiểm tra toàn lẻ/ toàn chẵn
- **Ước chung lớn nhất/ bội chung nhỏ nhất**
- Kiểm tra số có dạng a^n hay không (a: hằng số)

```
1. //boi chung nho nhât
2. #include <iostream>
3. using namespace std;
4. int main() {
5.     int a, b, bcnn;
6.     cin>>a>>b;
7.     int step = (a>b) ? a : b;
8.     for (int i=step;i<=a*b;i+=step){
9.         if(i%a==0 && i%b == 0){
10.             bcnn = i;
11.             break;
12.         }
13.     }
14.     cout<<"BCNN: "<<bcnn;
15.     return 0;
16. }
```

III. MỘT SỐ DẠNG BÀI TẬP CƠ BẢN

3. Một số dạng ứng dụng nhiều

- Tìm số đảo ngược
- Kiểm tra số hoàn thiện
- Kiểm tra số nguyên tố, số chính phương
- Kiểm tra số đối xứng
- Kiểm tra toàn lẻ/ toàn chẵn
- Ước chung lớn nhất/ bội chung nhỏ nhất
- **Kiểm tra số có dạng a^n (a: hằng số)**

```
1.  //a = 3
2.  #include <iostream>
3.  using namespace std;
4.  int main() {
5.      int n;
6.      cin>>n;
7.      int t = n, i = 1;
8.      while (t>1){
9.          i *= 3;
10.         t/=3;
11.     }
12.     if (i == n) cout<<1;
13.     else cout<<0;
14.     return 0;
15. }
```

IV. HÀM VÀ CẤU TRÚC MỘT CHƯƠNG TRÌNH

1. Phạm vi của biến

- Biến cục bộ là biến được khai báo trong một khối lệnh, phạm vi sử dụng trong khối lệnh (hoặc hàm) mà nó được khai báo.
- Biến toàn cục so với một khối lệnh (hoặc hàm) là biến được khai báo bên ngoài khối lệnh (ngoài các hàm), được hiểu trong phạm vi từ vị trí khai báo đến khi kết thúc khối lệnh chứa biến.

=> *Biến toàn cục được khai báo bên ngoài tất cả các hàm thì được hiểu cho toàn chương trình.*

IV. HÀM VÀ CẤU TRÚC MỘT CHƯƠNG TRÌNH

1. Phạm vi của biến

```
1. #include <iostream>
2. using namespace std;
3. int n = 90;
4. int main() {
5.     int a = 10;
6.     { int a = 12;
7.         cout<<a<<"\n"; //12
8.     }
9.     cout<<a<<"\n"; //10
10.    cout<<n; //90
11.    return 0;
12. }
```

IV. HÀM VÀ CẤU TRÚC MỘT CHƯƠNG TRÌNH

2. Hàm

- Một **chương trình con** được đặt tên và có chức năng chuyên biệt để xử lí các vấn đề chuyên biệt của chương trình chính.
 - Tái sử dụng được với nhiều đối số được truyền vào
 - Dễ dàng sửa lỗi và cải tiến
 - Chương trình chính trở nên dễ hiểu hơn



IV. HÀM VÀ CẤU TRÚC MỘT CHƯƠNG TRÌNH

3. Các cú pháp khi dùng hàm

a. Định nghĩa hàm

```
1. <Kiểu dữ liệu trả về><Tên hàm>(<danh sách tham số>)  
2. {  
3.     lệnh trong hàm;  
4. }
```

- Nếu hàm không có giá trị trả về, phần <Kiểu dữ liệu trả về> đặt là "void"

IV. HÀM VÀ CẤU TRÚC MỘT CHƯƠNG TRÌNH

3. Các cú pháp khi dùng hàm

a. Định nghĩa hàm

```
1. int chuongTrinhCon1 (bool thamso1, int& thamso2, int thamsoarray[])
2. {
3.     return <giá trị trả về_int>;
4. }
5. void chuongTrinhCon2 (int thamso3, int& thamso2)
6. {
7.     //Khong co lenh return
8. }
```

IV. HÀM VÀ CẤU TRÚC MỘT CHƯƠNG TRÌNH

3. Các cú pháp khi dùng hàm

b. Gọi hàm

1. <Tên hàm>(danh sách đối số);

Nếu không có đối số cần truyền vào, để trống danh sách đối số

Nếu chỉ gọi tên hàm mà không có () hay (<danh sách đối số>), giá trị trả về luôn bằng 1



IV. HÀM VÀ CẤU TRÚC MỘT CHƯƠNG TRÌNH

3. Các cú pháp khi dùng hàm

b. Gọi hàm

```
1. int main (){  
2.     bool doiso1 = true;  
3.     int doiso2 = 8;  
4.     int doiso3 = 10;  
5.     int doisoarray = [10, 8, 9];  
6.     int cTC1 = chuongTrinhCon1(doiso1, doiso2, doisoarray);  
7.     chuongTrinhCon2(doiso3, doiso2);  
8.     cout<<chuongTrinhCon2; //1  
9.     return 0;  
10. }
```

IV. HÀM VÀ CẤU TRÚC MỘT CHƯƠNG TRÌNH

3. Các cú pháp khi dùng hàm

c. Khai báo hàm (nguyên mẫu của hàm)

1. <Kiểu dữ liệu trả về><Tên hàm>(<danh sách tham số>);

2. //hoặc:

3. <Kiểu dữ liệu trả về><Tên hàm>(<danh sách kiểu dữ liệu của tham số>);

- Dùng trong trường hợp: Có lời gọi đến hàm nhưng nội dung hàm không được định nghĩa trước.



IV. HÀM VÀ CẤU TRÚC MỘT CHƯƠNG TRÌNH

3. Các cú pháp khi dùng hàm

c. Khai báo hàm (nguyên mẫu của hàm)

```
1.  chuongTrinhCon2(int thamso3, int& thamso2);  
2.  chuongTrinhCon2(int, int&);
```



IV. HÀM VÀ CẤU TRÚC MỘT CHƯƠNG TRÌNH

4. Tham số và đối số

- **Tham số:** Phần trong dấu () khi khai báo hàm/ định nghĩa hàm. Đây là các biến cục bộ của hàm.
- **Đối số:** Phần trong dấu () ở lời gọi hàm. Khi gọi hàm, các đối số được truyền vào tham số tương ứng ở hàm để thực hiện tính toán



IV. HÀM VÀ CẤU TRÚC MỘT CHƯƠNG TRÌNH

4. Tham số và đối số

- Hoạt động truyền đối số cho tham số:
- **Truyền tham trị** (<Kiểu dữ liệu> <Tên tham số>): Giá trị của đối số được truyền vào không thay đổi khi hàm được thực thi
- **Truyền tham chiếu** (<Kiểu dữ liệu>& <Tên tham số>): Giá trị của đối số được truyền vào **thay đổi** khi hàm được thực thi



IV. HÀM VÀ CẤU TRÚC MỘT CHƯƠNG TRÌNH

4. Tham số và đối số

- **Truyền tham trị:** Giá trị của đối số được truyền vào không thay đổi khi hàm được thực thi

```
1. int chuongTrinhThamTri (int a, int b)
2. {
3.     int sum = 0;
4.     for (int i = 0; i < a; i ++ )
5.     {
6.         sum +=b;
7.         b+=2;
8.     }
9.     return sum;
10. }
```

```
11. int main()
12. {
13.     int a = 2;
14.     int b = 10;
15.     int sum = chuongTrinhThamTri (a,b);
16.     cout<<sum; //sum = 22
17.     cout<<b; //b = 10
18.     return 0;
19. }
```

IV. HÀM VÀ CẤU TRÚC MỘT CHƯƠNG TRÌNH

4. Tham số và đối số

- **Truyền tham chiếu:** Giá trị của đối số được truyền vào **thay đổi** khi hàm được thực thi

```
1. int chuongTrinhThamChieu (int a, int &b)
2. {
3.     int sum = 0;
4.     for (int i = 0; i < a; i ++ )
5.     {
6.         sum +=b;
7.         b+=2;
8.     }
9.     return sum;
10. }
```

```
11. int main()
12. {
13.     int a = 2;
14.     int b = 10;
15.     int sum = chuongTrinhThamChieu (a,b);
16.     cout<<sum; //sum = 22
17.     cout<<b; //b = 14
18.     return 0;
19. }
```

IV. HÀM VÀ CẤU TRÚC MỘT CHƯƠNG TRÌNH

5. Cấu trúc một chương trình

- **Khối khai báo:** Khai báo sử dụng thư viện, không gian tên, các biến toàn cục và các hàm sẽ sử dụng trong chương trình (khai báo nguyên mẫu hàm)
- **Khối hàm main:** Viết hàm main
- **Khối định nghĩa hàm:** Viết chương trình con cho các hàm được khai báo



IV. HÀM VÀ CẤU TRÚC MỘT CHƯƠNG TRÌNH

5. Cấu trúc một chương trình

- Ví dụ: Chương trình tìm số ước chung của hai số a và b được nhập vào

```
1. #include <iostream>
2. using namespace std;
3. int Nhap();
4. void uocSoChung(int, int);
5. int main() {
6.     int a = Nhap();
7.     int b = Nhap();
8.     uocSoChung(a, b);
9.     return 0;
10. }

11. int Nhap()
12. {
13.     int x;
14.     cout<<"Nhap: ";
15.     cin>>x;
16.     return x;
17. }

18. void uocSoChung (int m, int n){
19.     int i = 1;
20.     int dem = 0;
21.     while ((m/i != 0) || (n/i!=0))
22.     {
23.         if (m%i==0 && n%i==0)
24.             dem++;
25.         i ++; }
26.     cout << dem;
27. }
```

V. LẬP TRÌNH ĐỀ QUY

1. Một bài toán đệ quy

- Một bài toán đệ quy có thể được giải quyết bằng cách đưa bài toán về dạng thức đơn giản hơn theo một quy tắc có thể xác định được.
- Trường hợp đệ quy: Trường hợp mà hạng tử sau của bài toán có thể được xác định qua hạng tử trước của bài toán theo một quy tắc nhất định.
- Trường hợp cơ bản: Trường hợp mà hạng tử của bài toán không định nghĩa bằng quy tắc mà cho trước một giá trị



V. LẬP TRÌNH ĐỆ QUY

1. Một bài toán đệ quy

- Chẳng hạn, trong dãy Fibonacci: 1 1 2 3 5 8 13 21 34 ...
 - **Quy tắc:** $u(n) = u(n-1) + u(n-2)$ Với $n \geq 2$ – trường hợp đệ quy
 - Có hai **trường hợp cơ bản:** $u_0 = 1; u_1 = 1$



V. LẬP TRÌNH ĐỆ QUY

2. Hàm đệ quy

- Hàm chứa phần gọi đệ quy (**gọi lại chính hàm đó**) cho đến khi **đạt điều kiện dừng** ở trường hợp cơ bản. Sử dụng hàm đệ quy để xử lí các bài toán có tính đệ quy.

$$S(n) = 0 + 1 + 2 + 3 + \dots + n;$$

$$\text{Trường hợp đệ quy: } S(n) = S(n-1) + n;$$

$$\text{Trường hợp cơ bản: } S(0) = 0;$$

```
1. int tinhSn (int n)
2. {
3.     if (n < 1) return 0;
4.     return tinhSn(n-1) + n;
5. }
```



V. LẬP TRÌNH ĐỆ QUY

2. Hàm đệ quy

- **Đệ quy tuyến tính:** Hàm đệ quy chỉ gọi lại chính nó **một lần duy nhất** trong thân hàm.
- Hàm đệ quy tuyến tính dùng để xử lý những bài toán có giá trị sau được định nghĩa qua **một** giá trị đứng trước nó.

Tính giai thừa của một số nguyên n

$$P(n) = 1.2.3.4...n = 1. (1.2.3.4...n)$$

Trường hợp cơ bản: $P(0) = 1$;

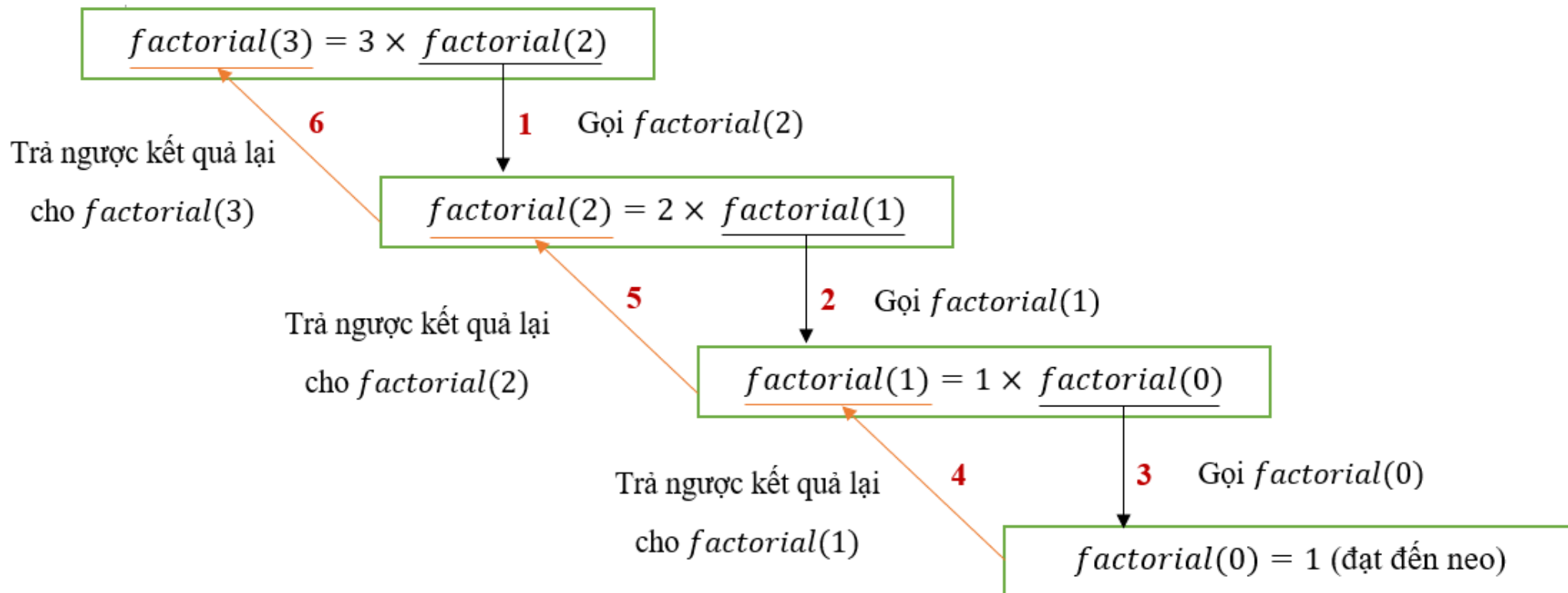
Trường hợp đệ quy: $P(n) = P(n-1)*n$;

```
1. int tinhPn (int n)
2. {
3.     if (n == 0) return 1;
4.     return tinhPn(n-1) *n;
5. }
```

V. LẬP TRÌNH ĐỆ QUY

2. Hàm đệ quy

```
1. int tinhPn (int n)
2. {
3.     if (n == 0) return 1;
4.     return tinhPn(n-1) *n;
5. }
```



V. LẬP TRÌNH ĐỆ QUY

2. Hàm đệ quy

- **Đệ quy tuyến tính:** Hàm đệ quy chỉ gọi lại chính nó **một lần duy nhất** trong thân hàm.
- Hàm đệ quy tuyến tính dùng để xử lý những bài toán có giá trị sau được định nghĩa qua **một** giá trị đứng trước nó.

Định nghĩa hàm tính tổng sau:

$$S(n) = 8 + 10 + 21 + 43 + 87 + \dots + S(n-1) + n + 1;$$

Trường hợp cơ bản: $S(0) = 8$;

Trường hợp đệ quy: $S(n) = 2 * S(n - 1) + n + 1$;

```
1. int tinhSn (int n)
2. {
3.     if (n < 1) return 8;
4.     return 2*tinhSn(n-1) + n + 1;
5. }
```

V. LẬP TRÌNH ĐỆ QUY

2. Hàm đệ quy

- **Đệ quy nhị phân:** Hàm đệ quy gọi lại chính nó **hai lần** trong thân hàm.
- Sử dụng để giải quyết các bài toán về tìm kiếm, sắp xếp hoặc các phép toán trên cây.
- Giá trị sau được định nghĩa qua hai giá trị đứng trước nó.

Tính số thứ n của dãy Fibonacci

Trường hợp cơ bản: $u(0) = 1$ và $u(1) = 1$

Trường hợp đệ quy: $u(n) = u(n-1) + u(n-2)$ Với $n \geq 2$

```
1. int tinhFib (int n)
2. {
3.     if (n <= 1) return 1;
4.     return tinhFib(n-1) + tinhFib(n-2);
5. }
```

V. LẬP TRÌNH ĐỆ QUY

2. Hàm đệ quy

- **Đệ quy hỗ tương:** Hai hàm đệ quy gọi cho nhau. Chẳng hạn có hai hàm A() và B(), A() gọi B() trong thân hàm và B() gọi A() trong thân hàm.
- Khi dùng đệ quy hỗ tương nhất thiết phải khai báo hàm trước. Dùng cho các bài toán có biến số phụ thuộc vào nhau.

Tính X(n), Y(n)

Trường hợp cơ bản: $X(0)=1, Y(0)=1$

Trường hợp đệ quy: $X(n) = X(n-1)+Y(n-1)$

$$Y(n) = X(n-1)*Y(n-1)$$

```
1.  int Y (int n); int X (int n);
2.  int X (int n)
3.  {
4.      if(n==0) return 1;
5.      return X(n-1) + Y(n-1);
6.  }
7.  int Y (int n)
8.  {
9.      if(n==0) return 1;
10.     return X(n-1)*Y(n-1);
11. }
```

V. LẬP TRÌNH ĐỆ QUY

3. Phương pháp chung để lập trình đệ quy

- **Xác định trường hợp đệ quy:** Xác định quy tắc mà bài toán đệ quy đưa ra
- **Xác định trường hợp cơ bản** (điều kiện dừng)
- Trong hàm, tại điểm mà tham số được truyền vào đạt điều kiện dừng thì trả lại giá trị của trường hợp cơ bản, tại các điểm khác thì trả về biểu thức của trường hợp đệ quy.



VI. MẢNG MỘT CHIỀU

Mảng một chiều là dãy các phần tử có cùng kiểu dữ liệu

1. Khai báo mảng 1 chiều:

- **Cú pháp:** KDL <TenBien> [SoPhanTuToiDa]
- **VD:** `int a[100];` (a là mảng 1 chiều có tối đa 100 phần tử. Mỗi phần tử trong mảng có kiểu dữ liệu là số nguyên)
- **VD:** `char str[20];` (str là mảng 1 chiều có tối đa 20 ptử. Mỗi phần tử trong mảng có kiểu là kiểu ký tự.)



VI. MẢNG MỘT CHIỀU

Mảng một chiều là dãy các phần tử có cùng kiểu dữ liệu

1. Khai báo mảng 1 chiều:

- Minh họa:

Mảng một chiều các số nguyên:

5	9	20	19
---	---	----	----

Mảng một chiều các ký tự:

S	A	I	G	O	N
---	---	---	---	---	---



Sharing is learning

VI. MẢNG MỘT CHIỀU

Mảng một chiều là dãy các phần tử có cùng kiểu dữ liệu

1. Khai báo mảng 1 chiều:

- Chỉ số mảng: (Mảng có n phần tử thì đánh chỉ số từ chỉ số 0 đến chỉ số $n - 1$ là phần tử cuối cùng trong mảng)



Sharing is learning

VI. MẢNG MỘT CHIỀU

Mảng một chiều là dãy các phần tử có cùng kiểu dữ liệu

1. Khai báo mảng 1 chiều:

- Một số ví dụ về khởi tạo và sử dụng mảng một chiều:

- `int c[5] = {0};`

Mảng c có tối đa 5 phần tử và đang sử dụng 5 phần tử (***Khởi tạo giá trị 0 cho mọi phần tử của mảng***)

- `int d[] = {8, 1, 2, 19, 88};`

Mảng d có tối đa 5 phần tử và đang sử dụng cả 5 phần tử. (***Tự động xác định số lượng phần tử***)



VI. MẢNG MỘT CHIỀU

Mảng một chiều là dãy các phần tử có cùng kiểu dữ liệu

1. Khai báo mảng 1 chiều:

- Một số ví dụ về khởi tạo và sử dụng mảng một chiều:

- `int a[5] = {15, 0, 1, 19, 89};`

Mảng a có tối đa 5 phần tử và đang sử dụng cả 5 phần tử. ***(Khởi tạo giá trị cho mọi phần tử trong mảng)***

- `int b[5] = {22, 5};`

Mảng b có tối đa 5 phần tử, nhưng hiện tại sử dụng có 2 phần tử tương ứng là `b[0] = 22` và `b[1] = 5`, còn 3 phần tử chưa được sử dụng. ***(Khởi tạo giá trị của một số phần tử trong mảng)***

VI. MẢNG MỘT CHIỀU

Mảng một chiều là dãy các phần tử có cùng kiểu dữ liệu

2. Kỹ thuật nhập/xuất mảng 1 chiều:

a. Định nghĩa hàm nhập:

```
11. void Nhap(int a[], int &n)
12. {
13.     cout<<"Nhap n:";
14.     cin>>n;
15.     for(int i=0; i<n; i++)
16.     {
17.         cout<<"Nhap a["<<i<<"]:";
18.         cin>>a[i];
19.     }
20. }
```



VI. MẢNG MỘT CHIỀU

Mảng một chiều là dãy các phần tử có cùng kiểu dữ liệu

2. Kỹ thuật nhập/xuất mảng 1 chiều:

b. Định nghĩa hàm xuất:

```
11. void Xuat(int a[], int n)
12. {
13. |   for(int i=0; i<n; i++)
14. |       cout<<setw(4)<<a[i];
15. }
```



VI. MẢNG MỘT CHIỀU

Mảng một chiều là dãy các phần tử có cùng kiểu dữ liệu

2. Kỹ thuật nhập/xuất mảng 1 chiều:

c. Một số cách truyền mảng cho hàm và lời gọi hàm:

```
void function_C1(int arr[100], int n);
```

Tên hàm: function_C1

Tham số: kiểu mảng số nguyên arr và số lượng phần tử mảng n

Giá trị trả về: không có giá trị trả về void

```
int function_C2(int arr[], int n);
```

Tên hàm: function_C2

Tham số: kiểu mảng số nguyên arr và số lượng phần tử mảng n

Giá trị trả về: kiểu số nguyên int

```
int function_C3(int *arr, int n);
```

Tên hàm: function_C3

Tham số: kiểu mảng con trỏ số nguyên arr và số lượng phần tử mảng n

Giá trị trả về: kiểu số nguyên int



VI. MẢNG MỘT CHIỀU

Mảng một chiều là dãy các phần tử có cùng kiểu dữ liệu

3. Một số kỹ thuật khác về mảng một chiều:

a. Kỹ thuật tính toán:

— Định nghĩa hàm tính tổng các giá trị âm trong mảng một chiều các số thực?

— Định nghĩa hàm.

```
11.float TongAm(float a[],int n)
12.{
13.    float s = 0;
14.    for(int i=0;i<n;i++)
15.        if(a[i]<0)
16.            s = s + a[i];
17.    return s;
18.}
```


VI. MẢNG MỘT CHIỀU

Mảng một chiều là dãy các phần tử có cùng kiểu dữ liệu

3. Một số kỹ thuật khác về mảng một chiều:

b. Kỹ thuật đếm:

— Định nghĩa hàm đếm số lượng số nguyên tố nhỏ hơn 100 trong mảng?

— Định nghĩa hàm.

```
11.int DemNguyenTo(int a[],int n)
12.{
13.    int dem = 0;
14.    for(int i=0;i<n;i++)
15.        if(a[i]<100 && ktNguyenTo(a[i])==true)
16.            dem = dem + 1;
17.    return dem;
18.}
```



Sharing is learning

VI. MẢNG MỘT CHIỀU

Mảng một chiều là dãy các phần tử có cùng kiểu dữ liệu

3. Một số kỹ thuật khác về mảng một chiều:

c. Kỹ thuật tìm kiếm:

— Định nghĩa hàm tìm giá trị lớn nhất trong mảng một chiều các số thực.

— Định nghĩa hàm.

```
11.float LonNhat(float a[],int n)
12.{
13.    float lc = a[0];
14.    for(int i=0;i<n;i++)
15.        if(a[i]>lc)
16.            lc = a[i];
17.    return lc;
18.}
```



Sharing is learning

VI. MẢNG MỘT CHIỀU

Mảng một chiều là dãy các phần tử có cùng kiểu dữ liệu

3. Một số kỹ thuật khác về mảng một chiều:

d. Kỹ thuật đặt cờ hiệu:

— Định nghĩa hàm kiểm tra trong mảng các số nguyên có tồn tại giá trị chẵn nhỏ hơn 2004 hay không?

— Định nghĩa hàm.

```
11.bool KiemTra(int a[],int n)
12.{
13.    bool flag = false;
14.    for(int i=0;i<n;i++)
15.        if(a[i]%2==0 && a[i]<2004)
16.            flag = true;
17.    return flag;
18.}
```



Sharing is learning

VII. MẢNG HAI CHIỀU

Mảng 2 chiều (Ma trận) là một tập hợp các phần tử trên dòng và các phần tử trên cột, có cùng kiểu dữ liệu và cùng tên

1. Khai báo mảng hai chiều:

- KDL<TenBien>[Sodongtoida][Socottoida]
- **VD:** float arr[20][20]; (arr là ma trận có tối đa 20 dòng và tối đa 20 cột. Mỗi phần tử trong ma trận có kiểu dữ liệu là kiểu số thực)
- **VD:** char str[10][30]; (str là ma trận có tối đa 10 dòng và tối đa 30 cột. Mỗi phần tử trong ma trận có kiểu là kiểu ký tự (char))

VII. MẢNG HAI CHIỀU

1. Khai báo mảng hai chiều:

- Chỉ số: Trong ma trận có m dòng và n cột (hay ma trận $m \times n$) thì các dòng trong ma trận được đánh số từ 0 đến $(m - 1)$, các cột trong ma trận được đánh số từ 0 đến $(n - 1)$.
- **VD:** `int arr[10][15];`
arr là ma trận có tối đa 10 dòng và 15 cột. Các dòng trong ma trận được đánh chỉ số từ 0 đến 9, các cột trong ma trận được đánh chỉ số từ 0 đến 14.



VII. MẢNG HAI CHIỀU

1. Khai báo mảng hai chiều:

- Hình ảnh minh họa:

char A[2][6]

Kiểu dữ liệu: char

Tên biến mảng: A

Mảng có 2 dòng và 6 cột

Số lượng phần tử trong mảng

$= 2 * 6 = 12$

D	A	I	H	O	C
U	I	T	H	C	M

int Mang2Chieu[3][4]

Kiểu dữ liệu: int

Tên biến mảng: Mang2Chieu

Mảng có 3 dòng và 4 cột

Số lượng phần tử trong mảng

$= 3 * 4 = 12$

5	8	0	1
7	2	4	9
8	3	6	0



VII. MẢNG HAI CHIỀU

1. Khai báo mảng hai chiều:

- Cách khai báo, sử dụng mảng 2 chiều:

Khởi tạo giá trị cho mọi phần tử của mảng

```
int A[3][5] =  
{  
    { 11, 22, 3, 52, 75 },  
    { 16, 7, 38, 89, 10 },  
    { 31, 7, 56, 14, 71 }  
};
```

→ Mảng A gồm 3 dòng x 5 cột và đang được sử dụng toàn bộ 15 phần tử.

Khởi tạo giá trị cho một số phần tử đầu dòng

```
int a[3][5] =  
{  
    { 31, 24 },  
    { 76, 17, 48 },  
    { 11 },  
};
```

→ Mảng a gồm 3 dòng x 5 cột và có tối đa 15 phần tử nhưng hiện tại chỉ sử dụng 6 phần tử, còn 9 phần tử chưa được sử dụng.

Tự động xác định số lượng phần tử

```
int arr[][4] =  
{  
    { 15, 82, 40, 1 },  
    { 7, 22, 67, 93 }  
};
```

→ Mảng arr gồm 2 dòng x 4 cột và đang sử dụng toàn bộ 12 phần tử.

11	22	3	52	75
16	7	38	89	10
31	7	56	14	71

31	24			
76	17	48		
11				

15	8	40	1
7	22	67	93



VII. MẢNG HAI CHIỀU

1. Một số kĩ thuật lập trình trên mảng 2 chiều:

a. Kĩ thuật nhập:

Định nghĩa hàm nhập ma trận các số nguyên

```
11. void Nhap(int a[][100], int &m, int &n)
12. {
13.     cout<<"Nhap m: ";
14.     cin>>m;
15.     cout<<"Nhap n: ";
16.     cin>>n;
17.     for(int i=0; i<m; i++)
18.         for(int j=0; j<n; j++)
19.         {
20.             cout<<"Nhap a["<<i<<"]["<<j<<"]: ";
21.             cin>>a[i][j];
22.         }
23. }
```



Sharing is learning

VII. MẢNG HAI CHIỀU

1. Một số kĩ thuật lập trình trên mảng 2 chiều:

a. Kĩ thuật nhập:

Định nghĩa hàm nhập ma trận các số nguyên

```
11. void Nhap(int a[][100], int &m, int &n)
12. {
13.     cout<<"Nhap m: ";
14.     cin>>m;
15.     cout<<"Nhap n: ";
16.     cin>>n;
17.     for(int i=0; i<m; i++)
18.         for(int j=0; j<n; j++)
19.         {
20.             cout<<"Nhap a["<<i<<"]["<<j<<"]: ";
21.             cin>>a[i][j];
22.         }
23. }
```

— Phân tích:

- + Tên hàm: **Nhap**.
- + Số lượng tham số: ba tham số.
- + Tên tham số: Khai báo hàm ko cần nói tên tham số.
- + Loại tham số: cả ba đều là tham biến.
- + Kiểu dữ liệu trả về: không có.



Sharing is learning

VII. MẢNG HAI CHIỀU

1. Một số kỹ thuật lập trình trên mảng 2 chiều:

b. Kỹ thuật xuất:

Định nghĩa xuất nhập ma trận các số nguyên

```
1. void Xuat(int a[][100], int m, int n)
2. {
3.     for(int i=0; i<m; i++)
4.     {
5.         for(int j=0; j<n; j++)
6.             cout<<a[i][j];
7.         cout<<"\n";
8.     }
9. }
```



VII. MẢNG HAI CHIỀU

1. Một số kỹ thuật lập trình trên mảng 2 chiều:

c. Kỹ thuật đếm:

— Định nghĩa hàm đếm số lượng giá trị chẵn.

```
1. int DemChan (int a[][100], int m, int n)
2. {
3.     int dem=0;
4.     for(int i=0; i<m; i++)
5.         for(int j=0; j<n; j++)
6.             if(a[i][j]%2==0)
7.                 dem++;
8.     return dem;
9. }
```



VII. MẢNG HAI CHIỀU

1. Một số kĩ thuật lập trình trên mảng 2 chiều:

d. Kĩ thuật tìm kiếm:

— Định nghĩa hàm tìm giá trị lớn nhất.

```
1. float LonNhat(float a[][100], int m, int n)
2. {
3.     float lc = a[0][0];
4.     for(int i=0; i<m; i++)
5.         for(int j=0; j<n; j++)
6.             if(a[i][j]>lc)
7.                 lc = a[i][j];
8.     return lc;
9. }
```



VII. MẢNG HAI CHIỀU

1. Một số kĩ thuật lập trình trên mảng 2 chiều:

e. Kỹ thuật đặt cờ hiệu:

— Định nghĩa hàm kiểm tra trong ma trận các số nguyên có tồn tại giá trị chẵn nhỏ hơn 2004 hay ko.

```
1. int TonTaiChan(int a[][100], int m, int n)
2. {
3.     int flag = 0;
4.     for(int i=0; i<m; i++)
5.         for(int j=0; j<n; j++)
6.             if(a[i][j]%2==0 && a[i][j]<2004)
7.                 flag = 1;
8.     return flag;
```



VIII. KIỂU CẤU TRÚC

Kiểu cấu trúc trong C++, hay còn gọi là kiểu struct trong C++ là một tập hợp các thuộc tính liên quan tới cùng một đối tượng. Ví dụ điển hình của kiểu cấu trúc là tập hợp các thuộc tính liên quan tới một người như tên, tuổi và giới tính.

VD: NGAY x; // x có kiểu cấu trúc ngày



VIII. KIỂU CẤU TRÚC

1. Khai báo kiểu cấu trúc

```
struct <struct name>  
{  
    <type1> <member1>;  
    <type2> <member2>;  
    ...  
};
```



VIII. KIỂU CẤU TRÚC

1. Khai báo kiểu cấu trúc

Ví dụ:

```
struct people
```

```
{
```

```
    int old;
```

```
    char *name;
```

```
    int height;
```

```
    char *sex;
```

```
};
```



Sharing is learning

VIII. KIỂU CẤU TRÚC

1. Khai báo kiểu cấu trúc

- Typedef: typedef trong C++ là một câu lệnh có tác dụng định nghĩa lại tên của một kiểu đã được định nghĩa bởi một tên khác.
- Ví dụ, chúng ta có thể định nghĩa lại kiểu int bằng một tên mới là int_new như sau: `typedef int int_new;`
- Khi đó, tất cả các biến thuộc kiểu int_new trong chương trình cũng sẽ có kiểu dữ liệu là int.



VIII. KIỂU CẤU TRÚC

2. Kích thước struct trong C++

- Kích thước struct trong C++ là kích thước mà struct đó chiếm trong bộ nhớ máy tính. Kích thước struct trong C++ được tính bởi đơn vị byte, và kích thước của nó được tính bằng tổng kích thước của tất cả các thành viên trong nó, cộng thêm một bộ nhớ đệm (padding memory) ở giữa các thành viên nhằm tăng tốc độ truy cập của máy tính. Ví dụ, chúng ta có thể định nghĩa lại kiểu int bằng một tên mới là int_new như sau: `typedef int int_new;`
- Khác với các loại dữ liệu khác, ví dụ như mảng với kích thước mảng bằng đúng tổng kích thước của các phần tử tạo nên mảng, thì kích thước của struct trong C++ thường lớn hơn tổng kích thước của các phần tử tạo nên nó.
- Nguyên nhân là do các thành viên của struct được bố trí ở những vị trí thuận tiện cho việc truy cập nhằm tăng tốc độ truy cập của máy tính. Kết quả là, một khoảng trống (được gọi là đệm, tiếng anh là padding) được tạo ra giữa các thành viên và làm kích thước bộ nhớ tổng thể tăng lên.



VIII. KIỂU CẤU TRÚC

2. Kích thước struct trong C++

- Toán tử **sizeof()**
- Toán tử sizeof là một toán tử để kiểm tra kích thước bộ nhớ của các đối tượng trong chương trình C++ như biến, struct chẳng hạn. Toán tử sizeof trả về kích thước bộ nhớ của một biến hoặc một struct được tính bằng đơn vị byte.
- VD: sizeof (name);

Trong đó name chính là tên của thực thể của struct chúng ta cần lấy kích thước.



VIII. KIỂU CẤU TRÚC

2. Mảng cấu trúc

- Mảng cấu trúc trong C++ là mảng chứa các thực thể được tạo ra từ một kiểu cấu trúc bên trong nó. Do thực thể tạo ra từ cấu trúc cũng là một loại giá trị, nên chúng ta cũng có thể lưu trữ chúng như là phần tử trong cùng một mảng. Và các phần tử trong mảng cấu trúc đều có chung kiểu là kiểu cấu trúc đã sử dụng để tạo nên thực thể.
- Mảng cấu trúc trong C++ có đặc tính của cả kiểu mảng và kiểu cấu trúc, do đó chúng ta có thể truy cập vào các thực thể trong mảng cấu trúc thông qua index, cũng như là truy cập vào các thành viên của từng thực thể theo cách thông thường.



VIII. KIỂU CẤU TRÚC

2. Mảng cấu trúc

- Khai báo: `struct_name array_name[length];`
- VD: Ta có kiểu cấu trúc sau:

```
typedef struct {  
    char tenloai[20];  
    char loai[100];  
    int gioitinh;  
    double height;  
    double weight;  
} dongvat;
```

- Khai báo mảng cấu trúc có 3 phần tử kiểu `dongvat`:
`dongvat d[3];`



IX. CON TRỎ

1. Địa chỉ một biến trong máy tính

- Các thiết bị nhớ cung cấp bộ nhớ tạm thời (là bộ nhớ được sử dụng trong quá trình máy tính làm việc để lưu trữ dữ liệu) đều được tạo nên bởi các ô nhớ liên tiếp nhau, mỗi ô nhớ tương ứng với một byte và đều có một số thứ tự đại diện cho vị trí của ô nhớ đó trong thiết bị. Số thứ tự đó được gọi là địa chỉ của ô nhớ.
- Các địa chỉ của ô nhớ là những con số ảo được tạo ra bởi hệ điều hành, mà con người chúng ta rất khó đọc. Hãy cứ tưởng tượng các ô nhớ được đánh số từ 0,0, và địa chỉ cuối cùng được đánh số tương đương với số ô nhớ của thiết bị đó.



IX. CON TRỎ

2. Con trỏ

Một **con trỏ (a pointer)** là một **biến được dùng để lưu trữ địa chỉ của biến khác.**

a. Khai báo con trỏ:

```
{Kiểu_dữ_liệu} *{Tên_con_trỏ};
```

VD: Khai báo con trỏ có kiểu int: `int *ptr;`



Sharing is learning

IX. CON TRỎ

2. Con trỏ

Một **con trỏ (a pointer)** là một **biến được dùng để lưu trữ địa chỉ của biến khác.**

b. Con trỏ NULL

- Khi khai báo một con trỏ mà chưa khởi tạo địa chỉ trỏ đến cho nó, thì việc in ra giá trị của con trỏ có thể gây ra lỗi và chương trình sẽ bị đóng luôn. Nguyên nhân là do khi chưa khởi tạo, thì con trỏ sẽ nắm giữ một giá trị rác nào đó, có thể là một địa chỉ vượt quá giới hạn của bộ nhớ ảo.
- Để khắc phục, khi khởi tạo một con trỏ mà chưa sử dụng đến ngay, các bạn nên gán cho nó một giá trị là NULL hoặc nullptr. Đây là các macro được định nghĩa sẵn trong C++, khi gán một con trỏ bằng NULL hoặc nullptr nghĩa là con trỏ đó chưa trỏ đến giá trị nào cả. Nó được định danh sẵn trong C++



IX. CON TRỎ

2. Các phép toán cơ bản với con trỏ

a. Phép gán

- Ta chỉ được phép gán giá trị của con trỏ bằng với **địa chỉ** của một biến khác (hoặc một con trỏ khác) cùng kiểu dữ liệu với nó.
- Muốn gán địa chỉ của biến thông thường cho con trỏ, trước hết cần sử dụng toán tử `&` để lấy ra địa chỉ ảo của biến, sau đó mới gán địa chỉ đó cho con trỏ được. Còn nếu như gán một con trỏ khác cho con trỏ thì chỉ cần chúng cùng kiểu là được.

Ví dụ:

```
int x = 5;
```

```
int* ptr = &x;
```

```
int* ptr_1 = ptr;
```



Sharing is learning

IX. CON TRỎ

2. Các phép toán cơ bản với con trỏ

b. Truy xuất giá trị ở vùng nhớ mà con trỏ trỏ đến

Khi đã có một con trỏ trỏ đến địa chỉ nào đó trong thiết bị nhớ, muốn đưa ra giá trị của vùng nhớ mà con trỏ đang trỏ tới, các bạn sử dụng toán tử $*$ ở phía trước biến con trỏ.



IX. CON TRỎ

2. Các phép toán cơ bản với con trỏ

b. Truy xuất giá trị ở vùng nhớ mà con trỏ trỏ đến

Ví dụ:

```
1  #include <iostream>
2
3
4
5  using namespace std;
6
7
8
9  main()
10 {
11
12     int* ptr;
13
14     int value = 5;
15
16
17
18     ptr = &value;
19
20
21
22
23     cout << "Giá trị ở vùng nhớ mà con trỏ trỏ đến: " << *ptr;
24
25
26
27     return 0;
28 }
29
30
31
```

IX. CON TRỎ

2. Các phép toán cơ bản với con trỏ

b. Truy xuất giá trị ở vùng nhớ mà con trỏ trỏ đến

Ví dụ:

Kết quả của chương trình:

Giá trị ở vùng nhớ mà con trỏ trỏ đến: 5

```
1  #include <iostream>
2
3
4
5  using namespace std;
6
7
8
9  main()
10 {
11
12     int* ptr;
13
14     int value = 5;
15
16
17
18     ptr = &value;
19
20
21
22
23     cout << "Giá trị ở vùng nhớ mà con trỏ trỏ đến: " << *ptr;
24
25
26
27     return 0;
28 }
29
30
31
```

IX. CON TRỎ

2. Các phép toán cơ bản với con trỏ

b. Tăng và giảm con trỏ Ví dụ:

Giống như các biến thông thường, các con trỏ cũng có thể sử dụng những toán tử tăng giảm, chúng bao gồm: ++ , -- , + , - , += , -=



Sharing is learning

IX. CON TRỎ

2. Các phép toán cơ bản với con trỏ

b. Tăng và giảm con trỏ Ví dụ:

VD: Cho đoạn chương trình sau

```
1  #include <iostream>
2  | using namespace std;
3  main()
4
5  {
6
7      int value = 0;
8      int* ptr = &value;
9      cout << ptr << endl;
10     ++ptr;
11     cout << ptr;
12     return 0;
13
14 }
15
```

IX. CON TRỎ

2. Các phép toán cơ bản với con trỏ

b. Tăng và giảm con trỏ. Ví dụ:

VD: Cho đoạn chương trình sau

Kết quả:

0x104dfee8

0x104dfec

```
1  #include <iostream>
2  | using namespace std;
3  main()
4
5  {
6
7      int value = 0;
8      int* ptr = &value;
9      cout << ptr << endl;
10     ++ptr;
11     cout << ptr;
12     return 0;
13
14 }
15
```

IX. CON TRỎ

2. Các phép toán cơ bản với con trỏ

b. Tăng và giảm con trỏ Ví dụ:

Kết quả:

`0x104dfee8`

`0x104dfeec`

Kí hiệu 0x ở đầu địa chỉ thể hiện số đứng phía sau là thập lục phân. Quy đổi hai giá trị 104dfee8 và 104dfeec ra hệ thập phân, ta được hai giá trị:

- Trước khi tăng: 273 546 984273546984 (bytes).
- Sau khi tăng: 273 546 988273546988 (bytes).

Hai giá trị này chênh nhau đúng 4 đơn vị, vừa bằng kích thước của kiểu dữ liệu int là 4 byte. Như vậy, toán tử ++ sẽ làm con trỏ trỏ đến địa chỉ tiếp theo trên bộ nhớ ảo, với khoảng cách đúng bằng kích thước của kiểu dữ liệu đã khai báo cho nó.

X. CHUỖI

1. Khái niệm

Kiểu	Kích thước	Vùng giá trị
char	1 byte	-128 tới 127
unsigned char	1 byte	0 tới 255
signed char	1 byte	-128 tới 127

- Kiểu kí tự là một kiểu dữ liệu có độ lớn **1 byte (8 bits)** dùng để lưu trữ 1 kí tự trong vùng nhớ máy tính.
- Kí tự trong C++ được hiểu là ký tự trong bảng mã ASCII.
- Kiểu ký tự char luôn được đặt ở giữa 2 dấu nháy đơn

X. CHUỖI

2. Khai báo, khởi tạo

- Các kiểu khai báo chuỗi:

```
char s1[100];
```

```
char s2[];
```

```
char *s3;
```

- Các kiểu khởi tạo chuỗi:

Trường hợp độ dài cụ thể:

```
char s[10]={'U','I','T','\0'};
```

```
char s[10]="UIT";
```

\\Hệ thống tự động thêm kí tự kết thúc chuỗi '\0'

Trường hợp tự xác định độ dài:

```
chars[]={ 'U','I','T','\0'};
```

```
char s[]="UIT";
```

\\Hệ thống tự động thêm kí tự kết thúc chuỗi '\0'



X. CHUỖI

2. Khai báo, khởi tạo

- Trường hợp độ dài cụ thể:

0	1	2	3	4	5	6	7	8	9
'U'	'I'	'T'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'

- Trường hợp tự xác định độ dài:

0	1	2	3
'U'	'I'	'T'	'\0'



X. CHUỖI

3. Nhập/xuất chuỗi và một số hàm thông dụng trong thư viện

a. Nhập chuỗi không khoảng trắng:

Ví dụ :

```
#include<iostream>
```

```
using namespace std;
```

```
int main() {
```

```
    char s1[100];
```

```
    cout << "Nhap chuoi 1: ";
```

```
    cin >> s1;
```

```
    cout << "Chuoi 1 la: " << s1;
```

```
    return 0;
```

```
}
```



X. CHUỖI

3. Nhập/xuất chuỗi và một số hàm thông dụng trong thư viện

a. Nhập chuỗi không khoảng trắng:

Ví dụ :

Chương trình trên cho ra các kết quả

```
Nhap chuỗi 1: abcd
Chuỗi 1 là: abcd
```

```
Nhap chuỗi 1: abc def
Chuỗi 1 là: abc
```

Khi nhập chuỗi "abcde" và nhấn phím Enter để tạo ra kí tự xuống dòng '\n'

```
Nhap chuỗi 1: abcde
Chuỗi 1 là: abcde
```

Nhận xét: Khi đọc thông tin từ bàn phím, đối tượng **cin** sẽ đọc các ký tự cho đến khi gặp ký tự **khoảng trắng** ' ', hoặc ký tự **enter** '\n'

X. CHUỖI

3. Nhập/xuất chuỗi và một số hàm thông dụng trong thư viện

a. Nhập chuỗi không khoảng trắng:

Ví dụ :

Chương trình trên cho ra các kết quả

```
Nhap chuỗi 1: abcd
Chuỗi 1 là: abcd
```

```
Nhap chuỗi 1: abc def
Chuỗi 1 là: abc
```

Khi nhập chuỗi "abcde" và nhấn phím Enter để tạo ra kí tự xuống dòng '\n'

```
Nhap chuỗi 1: abcde
Chuỗi 1 là: abcde
```

Nhận xét: Khi đọc thông tin từ bàn phím, đối tượng **cin** sẽ đọc các ký tự cho đến khi gặp ký tự **khoảng trắng** ' ', hoặc ký tự **enter** '\n'

X. CHUỖI

3. Nhập/xuất chuỗi và một số hàm thông dụng trong thư viện

b. Nhập chuỗi không khoảng trắng:

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char* s2 = new char[100];
    cout << "Nhap chuoi 2: ";
    fgets(s2, sizeof(s2), stdin);
    // hoặc cin.getline(s2, 100);||gets_s(s2, 100);
    cout << "Chuoi 2 la: " << s2;
}
```

Chương trình bên cho ra các kết quả:

```
Nhap chuoi 2: abc def
Chuoi 2 la: abc def
```



Sharing is learning

X. CHUỖI

3. Nhập/xuất chuỗi và một số hàm thông dụng trong thư viện

c. Nhập chuỗi có chứa kí tự xuống dòng

- Bình thường, khi nhập 1 chuỗi ký tự hoặc số vào bàn phím, các chuỗi này sẽ được đẩy vào bộ nhớ đệm dùng cho việc lưu trữ trước khi được gán vào 1 biến.
- Việc quên xóa bộ nhớ đệm sẽ dẫn đến việc lưu trữ giá trị trong biến bị sai.



X. CHUỖI

3. Nhập/xuất chuỗi và một số hàm thông dụng trong thư viện

c. Nhập chuỗi có chứa kí tự xuống dòng

- Ví dụ:

```
1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4  int main() {
5      int n;
6      char s3[100];
7      cout << "Nhập n: ";
8      cin >> n;
9      cout << "Nhập chuỗi 3: "<<endl;
10     fgets(s3, sizeof(s3), stdin);
11     cout < "Chuỗi 3 là: " <<s3;
12     cout << "Hello";
13 }
```

Kết quả đoạn chương trình bên:

```
Nhap n: 3
Nhap chuoi 3:
Chuoi 3 la:
Hello
```

- Ở trên, chuỗi ký tự s3 không nhận được giá trị do bị ký tự **Enter** có trong bộ nhớ đệm chèn vào.
- **cin.ignore()** là 1 phương thức của đối tượng cin trong C++. Câu lệnh này có tác dụng **xóa ký tự đầu tiên** trong bộ nhớ đệm.

X. CHUỖI

3. Nhập/xuất chuỗi và một số hàm thông dụng trong thư viện

c. Nhập chuỗi có chứa kí tự xuống dòng

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    int n;
    char s3[100];
    cout << "Nhap n: ";
    cin >> n;
    cin.ignore();
    cout << "Nhap chuoi 3: ";
    fgets(s3, sizeof(s3), stdin);
    cout << "Chuoi 3 la: " << s3;
    cout << "Hello";
} |
```

Kết quả đoạn chương trình bên:

```
Nhap n: 3
Nhap chuoi 3: abc def
Chuoi 3 la: abc def
Hello
```



Sharing is learning

X. CHUỖI

3. Nhập/xuất chuỗi và một số hàm thông dụng trong thư viện

d. Một số hàm thường dùng liên quan đến chuỗi ký tự

Hàm	Công dụng
strlen	hàm tính độ dài chuỗi ký tự
Strcpy	hàm sao chép chuỗi ký tự
strdup	hàm tạo bản sao
strlwr/strupr	hàm chuyển chuỗi thành chuỗi viết thường/hoa
strrev	hàm đảo ngược
strcmp	hàm so sánh 2 chuỗi không phân biệt viết hoa thường
Strcat	hàm nối 2 chuỗi
strstr	hàm tìm chuỗi trong chuỗi



X. CHUỖI

3. Nhập/xuất chuỗi và một số hàm thông dụng trong thư viện

Một số lưu ý về kiểu dữ liệu char

- Mỗi kí tự hay gặp đều là các ký tự trong bảng mã ASCII. Bảng mã này có
- 256 giá trị từ 0 - 255. Mỗi kí tự sẽ được gán một mã ASCII. Các mã ASCII cần nắm được
 - Các kí tự từ a - z có mã ASCII từ 97-122.
 - Các kí tự từ A - Z có mã ASCII từ 65-90
 - Các kí tự từ 0 - 9 có mã ASCII từ 48-57
- Chú ý: Kiểu dữ liệu char là kí tự nhưng có thể sử dụng nó như một số, chính là mã ASCII đại diện cho nó để cộng, trừ, nhân, chia như là với số nguyên thông thường.



X. CHUỖI

4. Các thao tác trên chuỗi kí tự và ví dụ minh họa

- a. Định nghĩa hàm đếm số lượng kí tự khoảng trắng, in hoa, in thường có trong chuỗi:

```
void demKiTu(char str[])
{
    int kt=0, dt=0, dh=0;
    for(int i=0; i<strlen(str);i++)
    {
        if(str[i]==' ')
            kt++;
        else if((str[i]>='a')&&(str[i]<='z'))
            dt++;
        else if((str[i]>='A')&&(str[i]<='Z'))
            dh++;
    }

    cout<<"So ki tu khoang trang: "<<kt<<endl;
    cout<<"So ki tu thuong: "<<dt<<endl;
    cout<<"So ki tu hoa: "<<dh;
}
```

X. CHUỖI

4. Các thao tác trên chuỗi kí tự và ví dụ minh họa

b. So sánh 2 chuỗi kí tự

- Khi so sánh lớn nhỏ hai ký tự đơn, chúng ta đơn giản so sánh mã ký tự ASCII của chúng. Ví dụ, do mã unicode của ký tự a là 97 sẽ lớn hơn của ký tự A là 65 nên phép so sánh sau sẽ cho ra kết quả true: `'a' > 'A' //true`
- Có thể sử dụng hàm `strcmp()` để so sánh 2 chuỗi trong C++ với cú pháp sau đây:

`strcmp(str1, str2);`

Trong đó `str1` và `str2` là 2 chuỗi cần được so sánh.



Sharing is learning

X. CHUỖI

4. Các thao tác trên chuỗi kí tự và ví dụ minh họa

b. So sánh 2 chuỗi kí tự

Sau khi đã xác định được cặp **ký tự đầu tiên khác nhau giữa 2 chuỗi**, hàm strcmp() sẽ trả về **hiệu mã ký tự ASCII giữa chúng**, và bằng cách so sánh giá trị này với số 0, chúng ta có thể so sánh 2 chuỗi như sau:

Biểu thức	Giá trị trả về	Kết quả
strcmp(str1, str2)	> 0	str1 > str2
strcmp(str1, str2)	= 0	str1 = str2
strcmp(str1, str2)	< 0	str1 < str2



X. CHUỖI

4. Các thao tác trên chuỗi kí tự và ví dụ minh họa

c. Nối 2 chuỗi và xuất ra màn hình:

Không sử dụng hàm:

```
#include <iostream>

#include<cstring>

using namespace std;

int main() {

    char s1[10] = "dkhp";

    char s2[] = ".com";

    int i, j;

    int n1 = strlen(s1);

    int n2 = strlen(s2);

    j=0;

    for(i = n1; i<n1+n2; i++ ) {

        s1[i] = s2[j];

        j++;

    }

    s1[i] = '\0';

    cout<<"\nKet qua sau khi noi chuoi la:\n"<<s1;

    return 0;

}
```


X. CHUỖI

4. Các thao tác trên chuỗi kí tự và ví dụ minh họa

c. Nối 2 chuỗi và xuất ra màn hình:

Sử dụng hàm:

```
#include <iostream>

#include<cstring>

using namespace std;

int main() {

    char str1[100];

    char str2[100];

    char str3[100];

    int len;

    fgets(str1, sizeof(str1), stdin);

    fgets(str2, sizeof(str2), stdin);

    strcpy(str3, str1); // sao chép chuỗi str1 vào str3.

    strcat(str3, str2); // nối chuỗi str2 với str1 vừa được sao chép.

    cout<<"\nNoi chuoi: "<< str3;

    return 0;

}
```

X. CHUỖI

4. Các thao tác trên chuỗi kí tự và ví dụ minh họa

d. Chuyển đổi chữ in/thường

```
/*Hàm chuyển chữ thường thành chữ hoa trong C++*/
char upper(char chr){
    if('a' <= chr && chr <= 'z'){
        chr = chr -32;
    }
    return chr;
}

/*Hàm chuyển chữ hoa thành chữ thường trong C++*/
char lower(char chr){
    if('A' <= chr && chr <= 'Z'){
        chr = chr +32;
    }
    return chr;
}
```





BAN HỌC TẬP CÔNG NGHỆ PHẦN MỀM

TRAINING CUỐI KỲ HỌC KỲ I NĂM HỌC 2022 – 2023



Sharing is learning

HẾT

**CẢM ƠN CÁC BẠN ĐÃ THEO DÕI
CHÚC CÁC BẠN CÓ KẾT QUẢ THI THẬT TỐT!**

 **BAN HỌC TẬP**

Khoa Công nghệ Phần mềm

Trường Đại học Công nghệ Thông tin

Đại học Quốc gia thành phố Hồ Chí Minh

 **CONTACT**

bht.cnpm.uit@gmail.com

fb.com/bhtcnpm

fb.com/groups/bht.cnpm.uit