Chương 3

1. Hàm (Function)
2. Mảng (Array)
3. Chuỗi ký tự (Character sequences)

I. Hàm (Function)

1. Khái niệm

- Một đoạn chương trình có tên, đầu vào và đầu ra.
- Có chức năng giải quyết một số vấn đề chuyên biệt cho chương trình chính.
- Được gọi nhiều lần với các tham số khác nhau.
- Được sử dụng khi có nhu cầu:
  - Tái sử dụng.
  - Sửa lỗi và cải tiến

❖ Khai báo (declare) và định nghĩa (define) function:

Một **function** (hàm) được tạo ra từ những yếu tố sau:

- Kiểu trả về của hàm (data type of output).
- Tên hàm (function name).
- Danh sách tham số (function parameters).
- Khối lệnh (block of statements).

2. Cú pháp

- <Kiểu trả về> Tên hàm ([<Danh sách tham số>])

```
{
        Thân hàm; (Các câu lệnh)
        [return giá_trị/biểu thức];
}
```

<Kiểu trả về> : kiểu bất kỳ của C++ (bool,char, int, long, float,…). Nếu không trả về thì là void.
<Tên hàm>: theo quy tắc đặt tên định danh.
<danh sách tham số> : tham số hình thức đầu vào giống khai báo biến, cách nhau bằng dấu ,
<giá_trị/biểu thức> : trả về cho hàm qua lệnh return

I. Hàm (Function)

- Ví dụ:

```
int tinhTong(int a, int b)
{
        int s;
        s=a+b;
        return s;
}
```

- Sử dụng hàm: **tên_hàm(danh sách đối số);**

Ví dụ: tinhTong(x,y);

- **Lưu ý**: C++ không cho phép các hàm lồng nhau, nghĩa là phần định nghĩa của hàm này phải độc lập hoàn toàn với hàm khác.

I. Hàm (Function)

3. Nguyên tắc hoạt động của hàm

Khi máy gặp lời gọi hàm ở một vị trí nào đó trong chương trình, máy sẽ tạm dời chỗ đó và chuyển đến hàm tương ứng. Quá trình đó diễn ra theo trình tự sau:

- Cấp phát bộ nhớ cho các biến cục bộ.
- Gán giá trị của các đối số cho các tham số tương ứng.
- Thực hiện các câu lệnh trong thân hàm.
- Khi gặp câu lệnh return hoặc dấu } cuối cùng của thân hàm thì máy sẽ xoá các tham số hình thức, biến cục bộ và ra khỏi hàm.

❖ Nếu trở về từ một câu lệnh return có chứa biểu thức thì giá trị của biểu thức được gán cho hàm. Giá trị của hàm sẽ được sử dụng trong các biểu thức chứa nó.

I. Hàm (Function)

4. Tầm vực: Là phạm vi hiệu quả của biến và hàm.

Biến:

- **Toàn cục** (global): khai báo trong ngoài tất cả các hàm (kể cả hàm main) và có tác dụng lên toàn bộ chương trình.
- **Cục bộ** (local): khai báo trong hàm hoặc khối { } và chỉ có tác dụng trong bản thân hàm hoặc khối đó (kể cả khối con nó). Biến cục bộ sẽ bị xóa khỏi bộ nhớ khi kết thúc khối khai báo nó.

Phân biệt: dùng toán tử phạm vi ::

::x;//Global variable

x; //Local variable

I. Hàm (Function)

Câu hỏi:

Nêu cách hoạt động của chương trình sau:

```
1  #include<iostream>
2  using namespace std;
3  bool kiemTraSoNguyenTo(int n)
4  {
5      int dem = 0;
6      if (n<2)
7          return false;
8      for (int i = 1; i <= n; i++)
9      {
10         if (n%i == 0)
11             dem++;
12     }
13     return dem == 2;
14 }
```

```
15 int main()
16 {
17     int n;
18     cout << "Nhap n= ";
19     cin >> n;
20     bool k = kiemTraSoNguyenTo(n);
21     if (k == true)
22         cout << n << " la so nguyen to";
23     else
24         cout << n << " khong la so nguyen to";
25     system("pause");
26     return 0;
27 }
```

I. Hàm (Function)

- **<u>Chú ý</u>: Các bạn tìm hiểu thêm và sẽ trao đổi vào buổi thực hành**
  - Khai báo nguyên mẫu hàm (Function prototype).
  - Tham số hằng (Constant parameter).
  - Tham số mặc định (Default parameter).
  - Nạp chồng hàm (Function overloading).

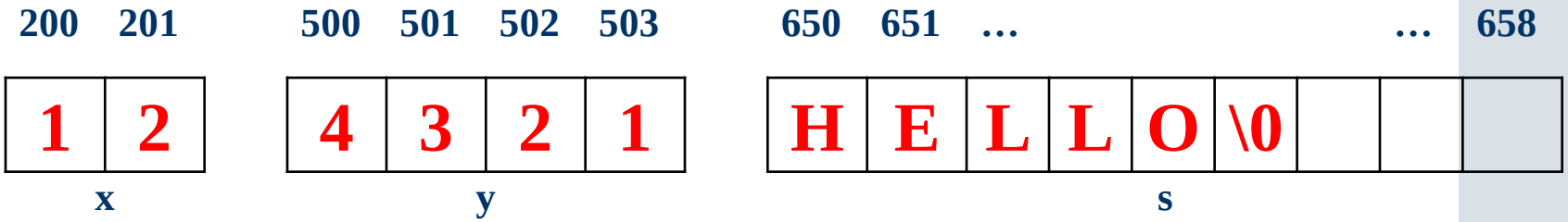
I. Hàm (Function)

5. Địa chỉ của biến (Address of a variable)

- Để dễ hình dung, trên vùng nhớ RAM (Random access memory) được chia thành các ô, mỗi ô 1 byte và được đánh số từ *0x00000000 đến 0xFFFFFFFF (tùy dung lượng)*
- Các số hệ 16 này được gọi là địa chỉ của biến.
- Khi khai báo một biến, thì biến đó được cấp phát một vùng nhớ và địa chỉ của biến đó là địa chỉ của ô nhớ đầu tiên

I. Hàm (Function)

- Để lấy địa chỉ của biến ta dùng toán tử & (address of operator) : &tên biến;
- Ví dụ:
  - int x;
  - cout<<"Dia chi cua x la:"<<&x;



Biến x chiếm 2 byte nhớ, có địa chỉ là 200, biến y có địa chỉ là 500 và chiếm 4 byte nhớ. Chuỗi s chiếm 6 byte nhớ tại địa chỉ 650. Các byte nhớ của một biến là liền nhau.

I. Hàm (Function)

6. Tham chiếu (reference)

- Là bí danh (alias), tên gọi khác của một biến có sẵn. Về bản chất là 2 biến sử dụng chung 1 vùng nhớ.
- Biến tham chiếu và biến được tham chiếu phải **cùng kiểu**, **duy nhất** và được **khởi tạo** khi khai báo.
- Khai báo: kiểu &biến tham chiếu = biến được tham chiếu;
- Ví dụ:
  - int x=5;
  - int &y=x;
  - cout<<y;

I. Hàm (Function)

- **Câu hỏi:**
  - const int x=5; int &y=x;
  - const int x=5; const int &y=x;
  - int x=5; const int &y=x;

I. Hàm (Function)

7. Cách truyền đối số (Pass parameters)

7.1 Truyền tham trị (Call/pass by value)

- Đối số được truyền ở *dạng giá trị (*sao chép giá trị vào cho tham số thực sự*)*
- Có thể truyền hằng, biến, biểu thức nhưng hàm chỉ nhận giá trị.
- Được sử dụng khi không có nhu cầu thay đổi giá trị của đối số sau khi thực hiện hàm

I. Hàm (Function)

❖ Ví dụ:

```
void hoanVi(int a, int b)
{
        int t=a; a=b;b=t;
}
int main()
{
        int x=5, y=10;
        hoanVi(7,12);
        hoanVi(x,y);
        hoanVi(2*x+y,y-2);
}
```

I. Hàm (Function)

7.2 Truyền tham chiếu (Call/pass by reference)

- Khi truyền đối số chỉ là biến.
- Có nhu cầu thay đổi giá trị của đối số sau khi thực hiện hàm
- Khi khai báo hàm thì các tham số hình thức phải có &

Chú ý: có thể viết int& x hoặc int & x; int &x

I. Hàm (Function)

```
void hoanVi(int &a, int &b)
{
        int t=a; a=b;b=t;
}
int main()
{
        int x=5, y=10;
        hoanVi(7,12);//lỗi
        hoanVi(x,y);
        hoanVi(2*x+y,y-2);//lỗi
}
```

I. Hàm (Function)

7.3. Truyền địa chỉ (Call/pass by address)// Con trỏ sẽ học chương sau

- Đối số phải là địa chỉ của biến.
- Khai báo hàm thì các tham số phải khai báo con trỏ.
- Có nhu cầu thay đổi giá trị của đối số sau khi thực hiện hàm

I. Hàm (Function)

```
void hoanVi(int *a, int *b)
{
        int t=*a; *a=*b;*b=t;
}
int main()
{
        int x=5, y=10;
        hoanVi(7,12);//lỗi
        hoanVi(&x,&y);
        hoanVi(2*x+y,y-2);//lỗi
}
```

I. Hàm (Function)

❖ Lưu ý:

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

ví dụ:

```
void vidu1(int x, int &y)
```

- Có thể sử dụng tham chiếu để trả về giá trị

```
void tinhtong(int x, int y, int &tong)
{
        tong=x+y;
}
```

I. Hàm (Function)

8. Hàm đệ quy

❖**Hàm đệ quy** là hàm mà từ một điểm trong thân của nó có thể gọi tên hàm của chính nó.

❖ **Cấu trúc** tổng quát của hàm đệ qui thường như sau:

```
if (trường hợp cơ sở)
{
  trình bày cách giải // giả định đã có cách giải
}
else    // trường hợp tổng quát
{
Gọi lại hàm với tham đối "bé" hơn
}
```

I. Hàm (Function)

❖ Ví dụ: Viết hàm đệ quy tính s= 1 + 2 +3 +...+n

```
int tinhTong(int n)
{
        if (n==0)
                return 0;
        else
                return n+tinhTong(n-1);
}
```

I. Hàm (Function)

- **Phân loại:**
- **Tuyến tính**: Trong thân hàm có duy nhất một lời gọi hàm gọi lại chính nó.

Ví dụ: Tính tích P=1 x 2 x 3 x ... x n

```
int tinhTich(int n)
{
        if (n==1)
                return 1;
        else
                return n * tinhTich(n-1);
}
```

I. Hàm (Function)

- **Nhị phân**: Trong thân hàm có hai lời gọi hàm gọi lại chính nó.

Ví dụ: Tính số hạng thứ n của dãy Fibonacy:

f(0) = f(1) = 1

f(n) = f(n – 1) + f(n – 2) n > 1

ĐK dừng: f(0) = 1 và f(1) = 1

```
long Fibo(int n)
{
  if (n == 0 || n == 1)
    return 1;
  return Fibo(n–1)+Fibo(n–2);
}
```

I. Hàm (Function)

- **Hỗ tương**: Trong thân hàm này có lời gọi hàm tới hàm kia và bên trong thân hàm kia có lời gọi hàm tới hàm này

Tính số hạng thứ n của dãy:

x(0) = 1, y(0) = 0

x(n) = x(n – 1) + y(n – 1)

y(n) = 3*x(n – 1) + 2*y(n – 1)

ĐK dừng: x(0) = 1, y(0) = 0

I. Hàm (Function)

```
long yn(int n);
long xn(int n)
{
   if (n == 0) return 1;
   return xn(n-1)+yn(n-1);
}
long yn(int n)
{
   if (n == 0) return 0;
   return 3*xn(n-1)+2*yn(n-1);
}
```

I. Hàm (Function)

- **Phi tuyến**: Trong thân hàm có lời gọi hàm lại chính nó được đặt bên trong thân vòng lặp.

Tính số hạng thứ n của dãy:

$x(0) = 1$

$x(n) = n^2x(0) + (n-1)^2x(1) + \ldots + 2^2x(n-2) + 1^2x(n-1)$

ĐK dừng: $x(0) = 1$

```
long xn(int n)
{
   if (n == 0) return 1;
   long s = 0;
   for (int i=1; i<=n; i++)
      s = s + i*i*xn(n–i);
   return s;
}
```

I. Hàm (Function)

- **Khử đệ quy**:
  - Vòng lặp
  - Stack
  - ...

I. Hàm (Function)

- Bài tập đệ quy, tính

1. S= 1 +2 +3 +...+n
2. S= 1^2 +2^2 + 3^2 +...+n^2
3. S=1 + 1/2 + 1/3 + ...+1/n
4. S= x + x^2 +x^3+..+x^n
5. Hãy đếm số lượng chữ số của số nguyên dương n
6. Hãy tính tổng các chữ số của số nguyên dương n.
7. Hãy đếm số lượng chữ số lẻ của số nguyên dương n
8. Hãy tính tổng các chữ số chẵn của số nguyên dương n
9. Cho số nguyên dương n. Hãy tìm chữ số đầu tiên của n
10. Hãy tìm chữ số đảo ngược của số nguyên dương n

II. Mảng (Array)

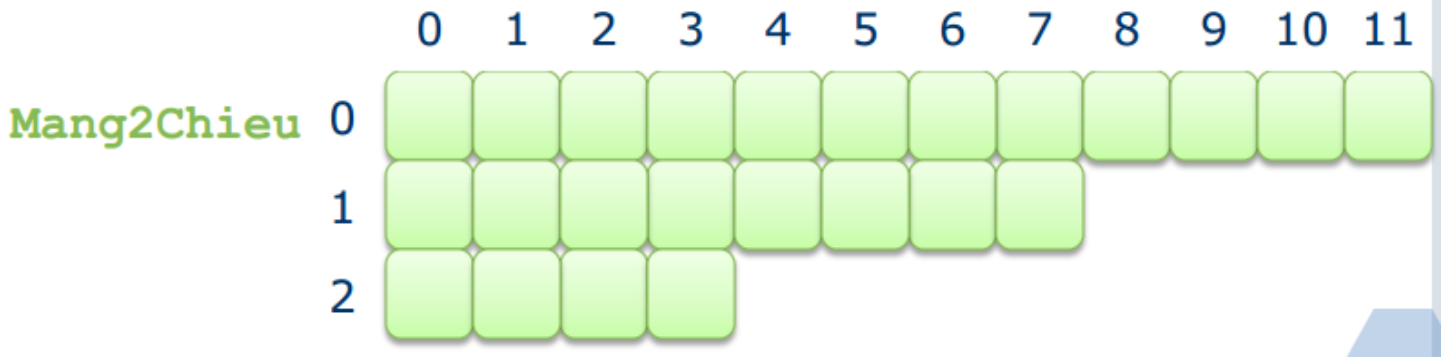
1. Khái niệm

- Mảng là một tập hợp hữu hạn các phần tử có cùng kiểu dữ liệu.
- Các phần tử của mảng được lưu trữ trong một khối gồm các ô nhớ liên tục nhau, có cùng tên (cũng là tên của mảng) nhưng phân biệt với nhau ở chỉ số.
- Chỉ số này xác định vị trí của nó trong mảng, chỉ số từ 0 đến <tổng số phần tử>-1
- Kích thước được xác định ngay khi khai báo và không bao giờ thay đổi.
- Bộ nhớ sử dụng = <tổng số phần tử>*sizeof(<kiểu dữ liệu>).

II. Mảng (Array)

```
int Mang1Chieu[10];
```


```
int Mang2Chieu[3][4];
```



II. Mảng (Array)

2. Mảng 1 chiều (array 1 dimensional)

2.1 Khai báo (declare)

<Kiểu_dữ_liệu> Tên_mảng [số_phần_tử] ;

<Kiểu_dữ_liệu> có thể char, int, float,...

Tên_mảng: theo quy tắc đặt tên

số_phần_tử (kích thước của mảng)

- **Phải được xác định ngay tại thời điểm khai báo**
- **Phải là hằng số**

Ví du:

int a[100];

II. Mảng (Array)

2.2 Khởi tạo giá trị cho mảng một chiều khi khai báo:

Kiểu_dữ_liệu Tên_mảng**[**số_phần_tử**] = {** giá_trị_1**,** giá_trị_2**,…};**

Trong đó, giá_trị_1, giá_trị_2,… là các giá trị tương ứng được khởi tạo cho từng phần tử của mảng theo đúng thứ tự. Số lượng các giá trị không được vượt quá kích thước của mảng.

Ví dụ 1: Khởi tạo giá trị cho tất cả các phần tử của mảng:

int A[5]={1,5,-8,7,0};

Ví dụ 2: Khởi tạo vài giá trị đầu của mảng, các phần tử sau mặc định bằng 0: **int A[5]={2,4,1};**

Ví dụ 3: Khởi tạo giá trị 0 cho tất cả các phần tử:

int A[5]={0};

Ví dụ 4: Khởi tạo mảng mà không khai báo kích thước:

int A[]={1,5,-8};

Khi đó mảng A sẽ tự động có kích thước bằng 3 vì nó có 3 phần tử.

II. Mảng (Array)

2.3 Truy xuất dữ liệu trong mảng:

Truy xuất các phần tử của mảng theo cú pháp:

Tên_mảng[chỉ_số]

Trong đó chỉ_số là số nguyên bắt đầu từ 0 đến n-1, với n là kích thước của mảng.

Ví dụ: Mảng A có 4 phần tử, phần tử thứ 2 là A[1]= - 7

Chỉ số	0	1	2	3
Mảng A	4	-7	3	2
Phần tử	A[0]	A[1]	A[2]	A[3]

II. Mảng (Array)

2.4 Gán dữ liệu kiểu mảng

Không được sử dụng phép gán thông thường, mà phải gán trực tiếp giữa các phần tử tương ứng.

Ví dụ:

```
int a[4];
int b[4];
a=b; //lỗi
a[2]=10;
for (int i=0;i<4;i++)
        a[i]=b[i];
```

II. Mảng (Array)

2.5 Truyền mảng cho hàm

Tham số kiểu mảng trong khai báo hàm giống như khai báo biến mảng

```
void sapXepTang(int a[100]);
```

Tham số kiểu mảng truyền cho hàm chính là địa chỉ của phần tử đầu tiên của mảng

- Có thể bỏ số lượng phần tử hoặc sử dụng con trỏ.
- Mảng có thể thay đổi nội dung sau khi thực hiện hàm.

```
void sapXepTang(int a[100]);
void sapXepTang(int a[]);
void sapXepTang(int *a);
```

* Lời gọi hàm

```
sapXepTang(a);
```

II. Mảng (Array)

2.6. Một số kỹ thuật cơ bản

Viết hàm thực hiện từng yêu cầu sau:

- Nhập mảng
- Xuất mảng
- Tìm kiếm một phần tử trong mảng
- Kiểm tra tính chất của mảng
- Tách mảng / Gộp mảng
- Tìm giá trị nhỏ nhất/lớn nhất của mảng
- Sắp xếp mảng giảm dần/tăng dần
- Thêm/Xóa/Sửa một phần tử vào mảng

II. Mảng (Array)

3. Mảng 2 chiều (array 2 dimensional)

3.1 Khai báo

Kiểu_dữ_liệu Tên_mảng**[**số_hàng**][**số_cột**] ;**

Khi đó kích thước mảng sẽ là tích (số_hàng*số_cột)

Ví dụ:

float A[3][4]; ***/*Mảng số thực A gồm 12 phần tử được chia thành 3 hàng, 4 cột*/***

3.2 Khởi tạo giá trị cho mảng hai chiều khi khai báo:

Ví dụ 1: Khởi tạo mảng số nguyên A[3][2] giống mảng 1 chiều:

```
int A[3][2] = {2, 7, 9, 0, 4, -3};
```

II. Mảng (Array)

Ví dụ 2: Khởi tạo theo từng dòng (hàng), các phần tử của mỗi dòng sẽ nằm trong một cặp dấu {}:

```
int A[3][2]={{2, 7}, {9, 0}, {4, 3}};
```

Ví dụ 3: Khởi tạo giá trị cho vài phần tử ở đầu mảng, các phần tử còn lại sẽ tự động nhận giá trị 0:

```
int A[3][2]={{2, 7}, {9}, {4, 3}};
```

Ví dụ 4: Khởi tạo giá trị 0 cho tất cả các phần tử của mảng:

```
int A[3][2]={0};
```

Ví dụ 5: Khởi tạo giá trị cho mảng mà không bao gồm khai báo chiều thứ nhất hay số hàng (chiều thứ hai hay số cột bắt buộc phải có):

```
int A[][2]={{2, 7}, {9, 0}, {4, 3}};
```

II. Mảng (Array)

3.3. Truy xuất các phần tử của mảng:

Mỗi phần tử của mảng có dạng:

Tên_mảng[chỉ_số_hàng][chỉ_số_cột]

Trong đó, chỉ_số_hàng có giá trị từ 0 đến (số_hàng -1)và chỉ_số_cột có giá trị từ 0 đến (số_cột - 1).

Ví dụ: Mảng int A[3][2] được minh họa như hình dưới:

Chỉ số	0, 0	0, 1	1, 0	1, 1	2, 0	2, 1
Mảng A						
Phần tử	[0][0]	[0][1]	[1][0]	[1][1]	[2][0]	[2][1]

	0	1
0	6	2
1	-9	0
2	1	5

II. Mảng (Array)

3.4 Gán dữ liệu kiểu mảng

Không được sử dụng phép gán thông thường, mà phải gán trực tiếp giữa các phần tử tương ứng.

Ví dụ:

```
int a[5][10], b[5][10];
b = a; // Sai
int i, j;
for (i = 0; i < 5; i++)
        for (j = 0; j < 10; j++)
                b[i][j] = a[i][j];
```

II. Mảng (Array)

3.5. Truyền mảng cho hàm

Tham số kiểu mảng trong khai báo hàm giống như khai báo biến mảng

```
void NhapMaTran(int a[50][100]);
```

Tham số kiểu mảng truyền cho hàm chính là địa chỉ của phần tử đầu tiên của mảng

- Có thể bỏ số lượng phần tử của dòng, hoặc sử dụng con trỏ.
- Mảng có thể thay đổi nội dung sau khi thực hiện hàm.

```
void NhapMaTran(int a[][100]);
void NhapMaTran(int (*a)[100]);
```

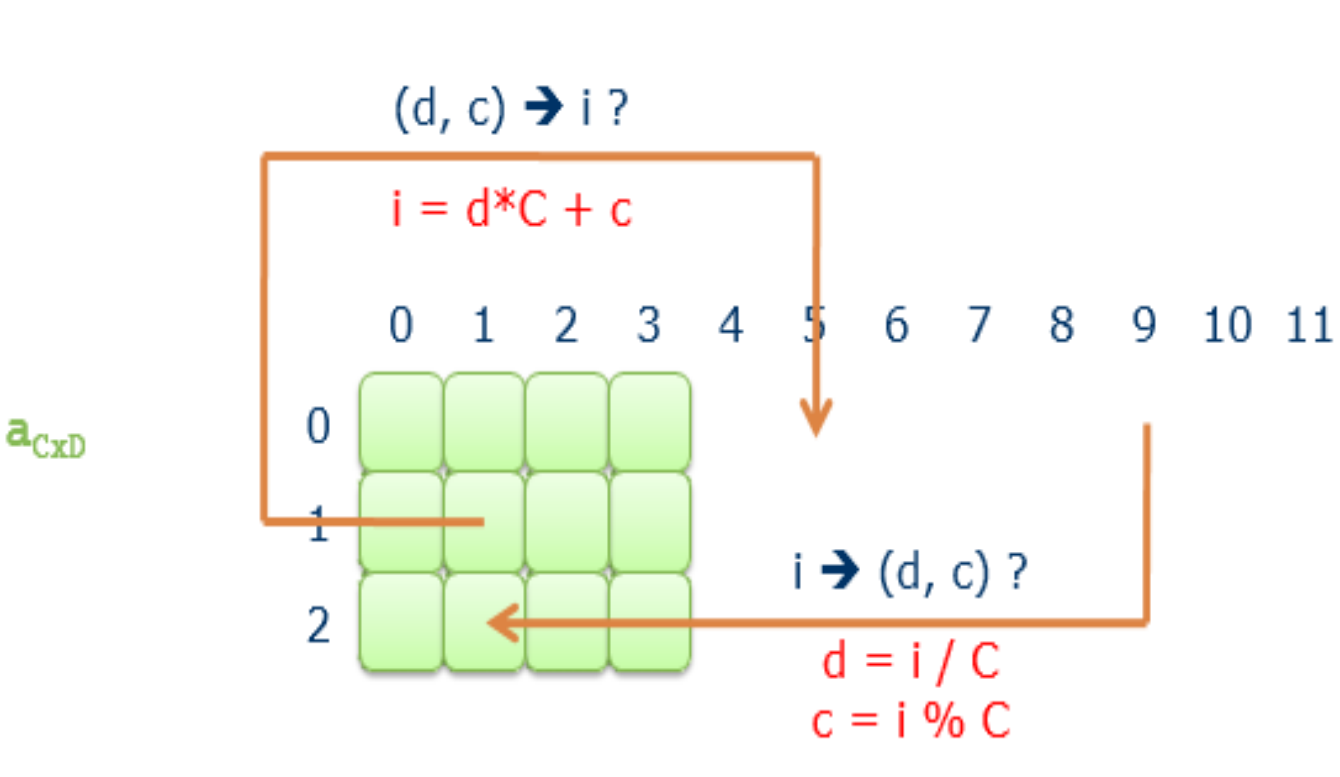
II. Mảng (Array)

* Lời gọi hàm

```
        void NhapMaTran(int a[][100], int &m, int &n);
        void XuatMaTran(int a[][100], int m, int n);
int main()
{
        int a[50][100], m, n;
        NhapMaTran(a, m, n);
        XuatMaTran(a, m, n);
}
```

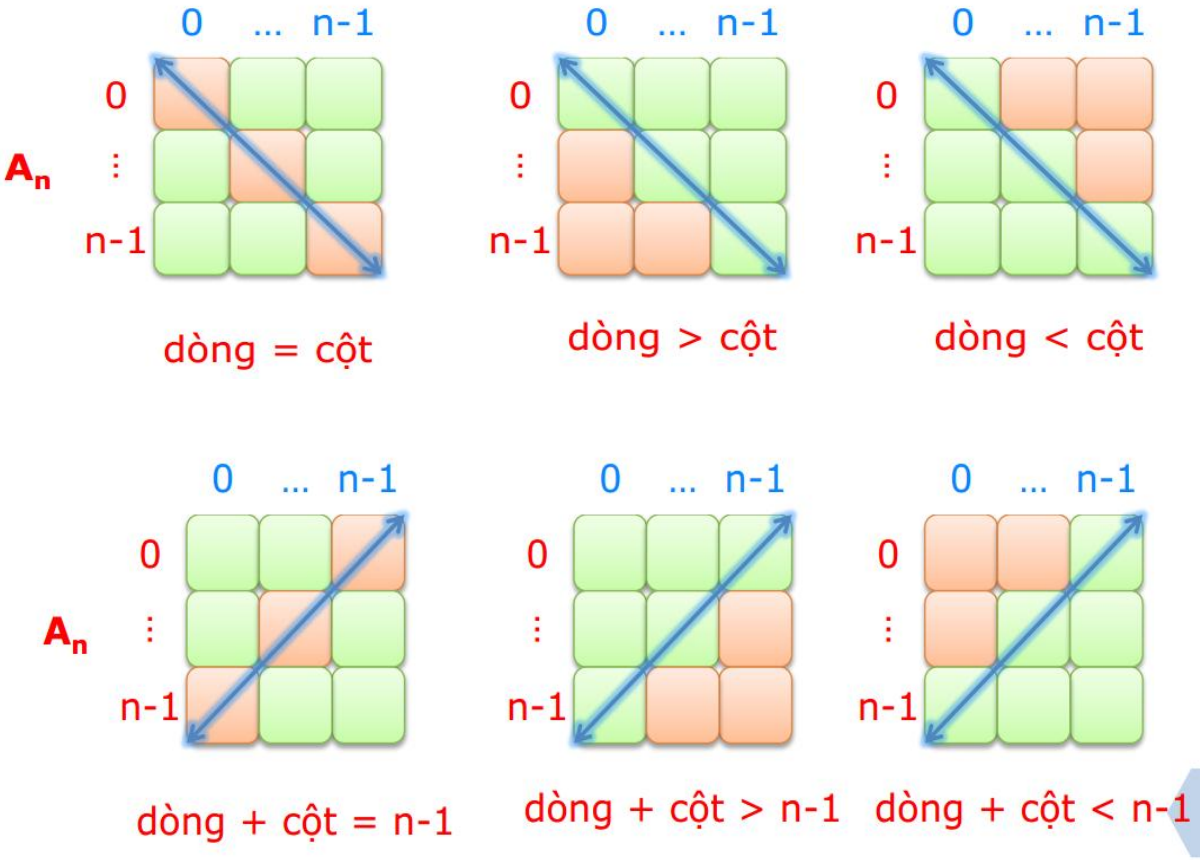
II. Mảng (Array)

3.6. Liên hệ giữa chỉ số mảng 1 chiều và chỉ số mảng 2 chiều



II. Mảng (Array)

3.7 Ma trận vuông: Số dòng = số cột



II. Mảng (Array)

3.8 Một số kỹ thuật cơ bản

Viết chương trình con thực hiện các yêu cầu sau

Nhập mảng

Xuất mảng

Tìm kiếm một phần tử trong mảng

Kiểm tra tính chất của mảng

Tính tổng các phần tử trên dòng/cột/toàn ma trận/đường chéo chính/nửa trên/nửa dưới

Tìm giá trị nhỏ nhất/lớn nhất của mảng

III. Chuỗi ký tự (Character sequences)

1. Khái niệm

- Kiểu char chỉ chứa được một ký tự. Để lưu trữ một chuỗi (nhiều ký tự) ta sử dụng mảng (một chiều) các ký tự.
- Chuỗi ký tự kết thúc bằng ký tự '\0' (null)
- Độ dài chuỗi = kích thước mảng – 1

Ví dụ

```
char Hoten[30]; // Dài tối đa 29 ký tự
char NgaySinh[9]; //Dài tối đa 8 ký tự
```

III. Chuỗi ký tự (Character sequences)

2. Khai báo

char Tên_chuỗi[độ dài chuỗi];

3. Khởi tạo

- Độ dài cụ thể:

```
char s[10]={'c', 'h', 'a', 'o', ' ', 'b', 'a', 'n','\0'};
char s[10]="chao ban"; // Tự động thêm '\0'
```

- Tự xác định độ dài

```
char s[]={'c', 'h', 'a', 'o', ' ', 'b', 'a', 'n','\0'};
char s[]="chao ban"; // Tự động thêm '\0'
```

III. Chuỗi ký tự (Character sequences)

4. Xuất/nhập

4.1. Xuất: cout

4.2. Nhập

- Nhập ký tự: **cin.get**(Biến ký tự);
- Nhập chuỗi ký tự: **cin.getline**(s, n);

Ví dụ:

```
char c,Str[100];
cout<<"Nhap mot ki tu: ";
cin.get(c);
cin.ignore(1);                  // xóa bộ nhớ đệm
cout<<"Nhap mot chuoi(xau): ";
cin.getline(Str,20);  // 20 là số kí tự tối đa của chuỗi.
cout<<"\nKi tu vua nhap la: "<<c<<"\n";
cout<<"Chuoi vua nhap la: "<<Str<<"\n";
```

III. Chuỗi ký tự (Character sequences)

5. Một số hàm xử lý chuỗi trong #include<string.h>

5.1. strcpy(s, t) ;

Hàm sẽ sao chép toàn bộ nội dung của xâu t (kể cả kí tự kết thúc xâu '\0') vào cho xâu s. Để sử dụng hàm này cần đảm bảo độ dài của mảng s ít nhất cũng bằng độ dài của mảng t. Trong trường hợp ngược lại kí tự kết thúc xâu sẽ không được ghi vào s và điều này có thể gây treo máy khi chạy chương trình.

Ví dụ:

```
#include<iostream>
#include<string.h>
using namespace std;
int main()
{
    char s[10], t[10];
    strcpy(t,"Face");// được, gán "Face" cho t
    strcpy(s,t);// được, sao chép t sang s
    cout << s << " to " << t<<'\n'; // in ra: Face to Face }
```

III. Chuỗi ký tự (Character sequences)

5.2. strncpy(s, t, n) ;

Sao chép n kí tự của t vào s. Hàm này chỉ làm nhiệm vụ sao chép, không tự động gắn kí tự kết thúc xâu cho s. Do vậy NSD phải thêm lệnh đặt kí tự '\0' vào cuối xâu s sau khi sao chép xong.

Ví dụ:

```
#include<iostream>
#include<string.h>
using namespace std;
int main()
{
    char s[15], t[20] = "Tin hoc dai cuong";
    strncpy(s, t, 3); // copy 3 kí tự "Tin" vào s
    s[3]=' '; //Gán kí tự khoảng trống cho s[3]
    strncpy(s+4, t+8, 9); //copy "dai cuong" vào s từ vị trí thứ 4
    s[13] = '\0'; // kết thúc xâu s
    cout <<"Chuoi s la: " << s<<'\n'; //Xuất ra "Tin dai cuong"
}
```

III. Chuỗi ký tự (Character sequences)

5.3. strcat(s, t);

Nối một bản sao của t vào sau s (thay cho phép +). Hiển nhiên hàm sẽ loại bỏ kí tự kết thúc xâu s trước khi nối thêm t. Việc nối sẽ đảm bảo lấy cả kí tự kết thúc của xâu t vào cho s (nếu s đủ chỗ) vì vậy NSD không cần thêm kí tự này vào cuối xâu.

Ví dụ:

```
#include<iostream>
#include<string.h>
using namespace std;
int main()
{
        char a[100] = "Nam", b[4] = "Bac";
        strcat(a, " va ");
        strcat(a, b);
        cout << a; //Sẽ xuất ra "Nam va Bac"
}
```

III. Chuỗi ký tự (Character sequences)

5.4. strncat(s, t, n);

Nối bản sao **n kí tự đầu tiên** của xâu t vào sau xâu s. Hàm tự động đặt thêm dấu kết thúc xâu vào s sau khi nối xong.

Tương tự, có thể sử dụng cách viết **strncat(s, t+k, n)** để nối n kí tự từ vị trí thứ k của xâu t cho s.

5.5. strcmp(s, t);

Hàm so sánh 2 xâu s và t (thay cho các phép toán so sánh). Giá trị trả lại là hiệu 2 kí tự khác nhau đầu tiên của s và t. Từ đó, nếu s1 < s2 thì hàm trả lại giá trị âm, bằng 0 nếu s1==s2, và dương nếu s1 > s2. Trong trường hợp chỉ quan tâm đến so sánh bằng, nếu hàm trả lại giá trị 0 là 2 xâu bằng nhau và nếu giá trị trả lại khác 0 là 2 xâu khác nhau.

III. Chuỗi ký tự (Character sequences)

5.6. strncmp(s, t, n) ;

Giống hàm strcmp(s, t) nhưng chỉ so sánh tối đa n kí tự đầu tiên của hai xâu.

5.7. stricmp(s, t) ;

Như strcmp(s, t) nhưng không phân biệt chữ hoa, thường.

5.8. strupr(s);

Hàm đổi xâu s thành in hoa, và cũng trả lại xâu in hoa đó.

5.9. strlwr(s);

Hàm đổi xâu s thành in thường, kết quả trả lại là xâu s.

5.10. strlen(s) ;

Hàm trả giá trị là độ dài của xâu s.

5.11. strrev(s);

Đảo ngược thứ tự các ký tự trong xâu s (Trừ ký tự '\0')

III. Chuỗi ký tự (Character sequences)

Bài 1: Xem thêm một số hàm khác như:

- atoi, atol, atof : đổi chuỗi thành số.
- itoa, ltoa, ultoa: đổi số thành chuỗi.
- strtok

Bài 2: Viết hàm nhận vào một chuỗi và trả về chuỗi tương ứng (giữ nguyên chuỗi đầu vào):

- Các ký tự thành ký tự thường (giống strlwr).
- Các ký tự thành ký tự hoa (giống strupr).
- Các ký tự đầu tiên mỗi từ thành ký tự hoa.
- Chuẩn hóa chuỗi (xóa khoảng trắng thừa)

III. Chuỗi ký tự (Character sequences)

Bài 3: Viết hàm nhận vào một chuỗi s và trả về chuỗi tương ứng sau khi xóa các khoảng trắng.

Bài 4: Viết hàm nhận vào một chuỗi s và đếm xem có bao nhiêu từ trong chuỗi đó.

Bài 5: Viết hàm nhận vào một chuỗi s và xuất các từ trên các dòng liên tiếp.

Bài 6: Viết hàm tìm từ có chiều dài lớn nhất và xuất ra màn hình từ đó và độ dài tương ứng

III. Chuỗi ký tự (Character sequences)

6. Mảng chuỗi

Mảng chuỗi là mảng trong đó các phần tử của nó là một chuỗi.

Cú pháp khai báo :

```
char Tên_mảng[Kích_thước][độ_dài_chuỗi];
```

Trong đó : kích_thước là số lượng phần tử tối đa của mảng, độ_dài_chuỗi là kích thước của các phần tử.

Để truy cập đến các phần tử ta chỉ cần viết tên mảng kèm chỉ số của nó ở trong cặp dấu **[]** đầu tiên.

III. Chuỗi ký tự (Character sequences)

```
#include<iostream>
using namespace std;
int main()
{
	char Sinh_vien[10][25]; //Mảng Sinh_vien gồm 10 chuỗi
	int i;
	for( i=0; i<10; i++)
	{
		cout<<"Nhap vao ho va ten sinh vien thu " <<i+1<<" :\n";
		cin.getline(Sinh_vien[i], 24);//Nhập tối đa 24 kí tự cho chuỗi Sinh_vien[i]
	}
	cout<<"Danh sach sinh vien vua nhap la: \n";
	for( i=0; i<10; i++)
	cout<<"Sinh vien thu "<<i+1<<": "<<Sinh_vien[i]<<"\n";
}
```

7. Tổ chức chương trình

- Các loại biến và phạm vi
  - *Biến cục bộ*
  - *Biến ngoài hay biến toàn cục*
- Biến với mục đích đặc biệt
  - *Biến hằng và từ khóa* ***const***
  - *Biến tĩnh và từ khóa* ***static***
  - *Biến thanh ghi và từ khóa* ***register***
  - *Biến ngoài và từ khóa* ***extern***
- Các chỉ thị tiền xử lý
  - *Chỉ thị bao hàm tệp* ***#include***
  - *Chỉ thị macro* ***#define***
  - *Chỉ thị* ***#ifdef*** *và* ***#ifndef***

- *Biến tĩnh và từ khóa* ***static***
  - Biến tĩnh được tạo ra bên trong một khối lệnh (**cục bộ**) có khả năng lưu giữ giá trị của nó cho dù chương trình đã chạy ra bên ngoài khối lệnh chứa nó.
  - Biến tĩnh chỉ cần được khai báo một lần duy nhất, và tiếp tục được duy trì sự tồn tại xuyên suốt cho đến khi chương trình kết thúc.

```
void staticVariablesPrint()
{
    static int s_value = 1;
    ++s_value;
    cout << s_value << '\n';
}
int main()
{
    staticVariablesPrint();
    staticVariablesPrint() ;
    staticVariablesPrint() ;
    return 0;
}
```

Một số hàm chuẩn C/C++

Tên hàm	Khai báo	Ý nghĩa
rand()	stdlib.h	Cho 1 giá trị ngẫu nhiên từ 0 đến 32767
random(x)	stdlib.h	Cho 1 giá trị ngẫu nhiên từ 0 đến x
pow(x,y)	math.h	Tính x mũ y
sqrt(x)	math.h	Tính căn bậc 2 của x
sin(x), cos(x), tan(x)	math.h	Tính sin, cosin, tang của góc x có số đo x radian
abs(a)	stdlid.h	Cho giá trị tuyệt đối của số nguyên a
labs(a)	stdlid.h	Cho giá trị tuyệt đối của số nguyên dài a
fabs(a)	stdlid.h	Cho giá trị tuyệt đối của số thực
exp(x)	math.h	Tính e mũ x
log(x)	math.h	Tính logarit cơ số e của x
log10(x)	math.h	Tính logarit cơ số 10 của x
ceil(x)	math.h	Phần nguyên nhỏ nhất không nhỏ hơn x
floor(x)	math.h	Phần nguyên lớn nhất không lớn hơn x

Một số hàm chuẩn C/C++

Tên hàm	Khai báo	Ý nghĩa
atof(str)	stdlib.h	Chuyển chuỗi str sang giá trị số thực
atoi(str)	stdlib.h	Chuyển chuỗi str sang số nguyên
itoa(x,str,y)	stdlib.h	Chuyển số nguyên x sang chuỗi str theo cơ số y
tolower(x)	ctype.h	Đổi chữ hoa sang chữ thường
toupper(x)	ctype.h	Đổi chữ thường sang chữ hoa
strcat(a,b)	string.h	Thêm chuỗi b vào sau chuỗi a
strcpy(a,b)	string.h	Sao chép chuỗi b vào a
strlwr(s)	string.h	Chuyển chuỗi s sang chữ thường
strupr(s)	string.h	Chuyển chuỗi s sang chữ hoa
strlen(s)	string.h	Trả về độ dài chuỗi s
strcmp(s,t)	string.h	Trả về hiệu của 2 kí tự khác nhau đầu tiên trong chuỗi, nếu hàm trả về giá trị 0 thì s bằng t.

Giới thiệu về string

1. #include<string>
2. using namespace std;
3. **Khai báo:**

    string tên_chuỗi;

Ví dụ: string s; // string (class), s (object).

4. **Hàm tạo**

    string s; char s1[]="abc";
    s="nguyen van a";
    string diachi("Vung Tau");
    string diachi1(diachi); //string s2(s1); chuyển mảng s1 thành s2.
    string nA(20,'a');

5. **Nhập/ xuất**

Xuất ra màn hình: cout (cout<<s;)

Nhập vào từ bàn phím: getline(cin, đối tượng, ký tự kết thúc);//Nhận cả ký tự trống(space, tab).

Giới thiệu về string

Mặc định là ký tự xuống dòng (‘\n’)
getline(cin,đối tượng): khi gặp ký tự ‘\n’ kết thúc chuỗi.
getline(cin,đối tượng,’.’): khi gặp ký tự ‘.’ kết thúc chuỗi.

6. Truy cập từng phần tử

tên_chuỗi[chỉ số]: như mảng ký tự.
tên_chuỗi.at(chỉ số);

7. Một số phương thức cơ bản:

- length(), size(): độ dài của chuỗi. Trả về int
- push_back(): thêm 1 ký tự vào cuối chuỗi. Trả về string
- pop_back(): xóa 1 ký tự ở cuối chuỗi. Trả về string
- append(chuỗi nối vào): nối chuỗi. s.append(s1): nối s1 vào s
- compare(chuỗi so sánh): trả về int. s.compare(s1): so sánh s với s1(0: s==s1, 1: s>s1, -1: s<s1)
- insert(vị trí, chuỗi cần chèn): string. Chèn chuỗi cần chèn vào chuỗi từ vị trí.

Giới thiệu về string

- find(chuỗi cần tìm): int. Trả về vị trí đầu tiên của chuỗi cần tìm. Nếu không có trả về giá trị rác.
- substr(vị trí, độ dài chuỗi con): string. Cho chuỗi con từ vị trí có độ dài chuỗi con.
- clear(): void. Xóa chuỗi.
- empty(): bool. true nếu chuỗi rỗng, ngược lại false.
- erase(vị trí, số ký tự cần xóa): string
- c_str(): const char* . Chuyển chuỗi string sang mảng chuỗi.
- capacity(): int. Trả về số bytes của string
- Lưu ý:
- Về string:
  - Không quan tâm đến cấp phát bộ nhớ, ký tự null.
  - Hỗ trợ +,+=,=,>,>=,<,<=,==,!=
  - Tốn bộ nhớ hơn so với mảng ký tự.
- Về mảng ký tự: quan tâm đến cấp phát bộ nhớ, ký tự null, nhưng ít tốn bộ nhớ hơn so với string.