CHƯƠNG 3

HÀM

Nội dung

1. Chương trình con
2. Khai báo hàm và định nghĩa hàm
3. Gọi hàm
4. Truyền tham số
5. Giá trị trả về
6. Phạm vi của biến
7. Biến mảng
8. Biến mảng là tham số của hàm

1. Chương trình con (1)

- **Chương trình con**: là một phần mã trong một chương trình lớn hơn, phần mã này thực hiện một tác vụ cụ thể và tương đối độc lập với phần mã còn lại.
- Một chương trình con thường được viết mã sao cho nó có thể được gọi nhiều lần từ nhiều nơi trong thời gian chạy của một chương trình (có thể được gọi bởi chính nó).
- Các chương trình con thường được tập trung thành các thư viện, là một cơ chế quan trọng cho việc chia sẻ và tái sử dụng mã.

1. Chương trình con (2)

- Chương trình con có 2 loại: Thủ tục (Procedure) và hàm (Function):
  - Thủ tục (PROCEDURE): Dùng để thực hiện một hay nhiều nhiệm vụ nào đó.
  - Hàm (FUNCTION): Trả về một giá trị nào đó (có kiểu vô hướng, kiểu string hoặc kiểu con trỏ). Hàm có thể sử dụng trong các biểu thức.

1. Chương trình con (3)

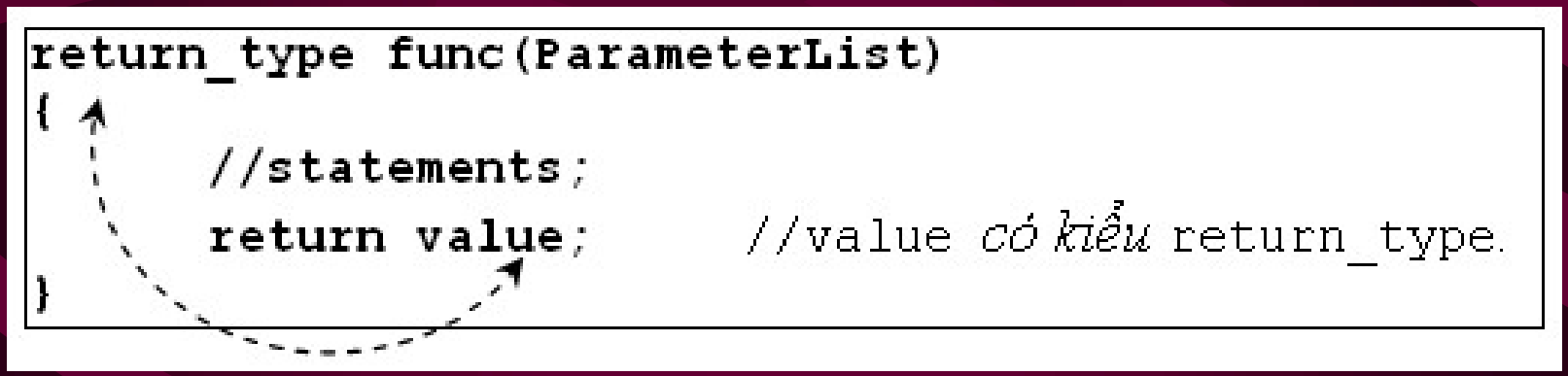
- Khi nào thì nên dùng thủ tục/hàm:
  - Dùng hàm:
    - Kết quả của bài toán trả về 1 giá trị duy nhất (kiểu vô hướng, kiểu string hoặc kiểu con trỏ).
    - Phát biểu gọi CHƯƠNG TRÌNH CON cần nằm trong các biểu thức tính toán.
  - Dùng thủ tục:
    - Kết quả của bài toán không trả về giá trị nào hoặc trả về nhiều giá trị hoặc trả về kiểu dữ liệu có cấu trúc (Array, Record, File).
    - Phát biểu gọi CHƯƠNG TRÌNH CON không nằm trong các biểu thức tính toán.

1. Chương trình con (4)

- Chương trình con được dùng khi xây dựng các chương trình lớn nhằm:
  - giảm đáng kể kích thước và chi phí của một chương trình
  - làm cho chương trình dễ theo dõi,
  - dễ sửa chữa,
  - nâng cao độ tin cậy của chương trình.
- Một đặc điểm nổi bật của chương trình con là nó có tính đệ quy nhờ thế mà nhiều bài toán được giải quyết dễ dàng.
- Chương trình con trong ngôn ngữ C là *hàm*.

2. Khai báo hàm, định nghĩa hàm (1)

- *Định nghĩa hàm* gồm tên hàm, các tham số và thân hàm (chứa các phát biểu chương trình), thực thi một việc cụ thể.
- Dạng định nghĩa hàm:



2. Khai báo hàm, định nghĩa hàm (2)

trong đó:

- Kiểu trả về (**return_type**, còn gọi là *kiểu hàm*) tương ứng với kiểu của giá trị mà hàm trả về thông qua phát biểu **return**.
- Tên hàm (func_name) được đặt theo nguyên tắc đặt tên, nhưng nên đặt tên sao cho dễ hiểu.
- **ParameterList** là danh sách tham số, mỗi tham số được xác định bởi *kiểu dữ liệu* và *tên*. Các tham số *phân cách nhau bởi dấu phẩy*. Có thể là danh sách rỗng.
- Phần thân hàm nằm giữa cặp ngoặc { và }.

2. Khai báo hàm, định nghĩa hàm (3)

- Nếu không xác định **return_type**, mặc định sẽ là kiểu **int**.
- Nếu hàm không trả về giá trị, dùng **void** (thay cho **return_type**).
- Không được phép đặt định nghĩa hàm này trong một hàm khác.
- Các hàm được định nghĩa *không phải theo một thứ tự* nào.
- Nếu có phát biểu gọi hàm trước khi hàm được định nghĩa thì cần có một *khai báo hàm trước* lời gọi hàm đó.

```
double tongDanDau(int n)
{
        double S= 0, t= 1.0;
        for (int i=1; i<=n; i++)
        {
                S += t/i;
                t=-t;
        }
        return S;
}
```

```
void main()
{
        double ss;
        ss= tongDanDau(5);
        cout<<ss;    //→ 0.7833
        int k= 7;
        ss= tongDanDau(k);
        cout<<ss;
}
```

định nghĩa hàm

gọi hàm

```
int toHop(int n,int k)
{
    long gThua(int n)
    {
        //...
    }
    return gThua(n)/(gThua(k)*(gThua(n-k)));
}
```

không được phép định nghĩa hàm ở đây

```
#include<iostream.h>

long gThua(int n);
int toHop(int n,int k);

void main()
{
	int cnk;
	cnk = toHop(5,3);
	cout<<cnk;
}
```

```
int toHop(int n,int k)
{
	return gThua(n)/(gThua(k)*(gThua(n-k));
}
long gThua(int n)
{
    long gt= 1;
    int i=1;
    for(; i<=n; i++)
        gt *= i;
    return gt;
}
```

2. Khai báo hàm và định nghĩa hàm

- Khai báo hàm là đưa ra một "mẫu hàm", mô tả tên hàm, kiểu trả về và danh sách tham số.
- Kết thúc khai báo hàm với dấu chấm phẩy ";".
- Dạng khai báo hàm:

  return_type func_name (ParameterList)
- Xét các ví dụ sau:

```
void f1(int i, int j, float k);
void f2(int a, b, float c);   //??
f3();
```

3. Gọi hàm

- Chỉ với định nghĩa hàm, hàm đó *chưa thực thi*. Hàm chỉ thực thi khi nó *được gọi*.
- Để *"yêu cầu"* một hàm thực thi, ta *"gọi"* tên hàm cùng với các tham số thực sự:

func(arg1, arg2,…); *//Lưu ý: không có kiểu dữ liệu*

```
double tongDanDau(int n)
{
      double S= 0, t= 1.0;
      for (int i=1; i<=n; i++)
      {
            S += t/i;
            t=-t;
      }
      return S;
}
```

```
void main()
{
      double ss;
      ss= tongDanDau(5);
      cout<<ss;    //→ 0.7833
      int k= 7;
      ss= tongDanDau(k);
      cout<<ss;
}
```

định nghĩa hàm

gọi hàm

Tham số trong chương trình con

- Chương trình con có thể không cần tham số mà chỉ có các biến riêng (biến cục bộ).
- Trường hợp cần chuyển các giá trị cho hàm khi gọi hàm thì cần định nghĩa danh sách tham số của hàm, còn gọi là các *tham số hình thức*.
- Mỗi giá trị thực chuyển cho hàm khi gọi hàm được gọi là *đối số* (hay *tham số thực*).
- Mỗi khi gọi hàm, có thể chuyển các đối số khác nhau.

4. Truyền tham số (1)

- Để một hàm thực thi, cần gọi hàm với tên và chuyển các đối số tương ứng với danh sách tham số hình thức cả về kiểu và thứ tự.
- ***Truyền bằng tham trị***:
  - Là khi giá trị của đối số được *sao chép* vào cho tham số hình thức. Như vậy, các thay đổi cho tham số hình thức (trong hàm) không làm thay đổi đối với tham số thực.
  - Mặc định, với cách khai báo danh sách tham số với kiểu và tên, ta có cách chuyển tham trị.

4. Truyền tham số (2)

- ***Truyền bằng tham chiếu***:
  - Khi muốn tham số hình thức và tham số thực cùng địa chỉ (bản chất là cùng ô nhớ nhưng khác tên), ta dùng cách chuyển tham chiếu cho hàm.
  - *Khai báo* tham số của hàm với kí tự '&' ngay trước tên (giữa KDL và tên).
  - Như vậy, mọi thay đổi đối với tham số hình thức cũng làm thay đổi tham số thực.
  - Có thể dùng chuyển tham chiếu để *trả về giá trị* cho nơi gọi hàm.

Ví dụ

- So sánh 2 hàm hoán vị sau đây, dùng chuyển tham chiếu và tham trị:

```
void hoanVi_1(int a, int b)
{
        int t= a;
        a= b,
        b= t;
   }
void hoanVi_2(int &a, int &b)
   {
        int t= a;
        a= b,
        b= t;
   }
```

```
void main()
   {
        int x= 3, y= 4;
   cout<<x<<' '<<y<<endl;
   hoanVi_1(x, y);
   cout<<x<<' '<<y<<endl;
   hoanVi_2(x, y);
   cout<<x<<' '<<y<<endl;
   }
//Nhận xét các giá trị được in ra
```

5. Giá trị trả về

- Một hàm không trả về giá trị khi hàm được khai báo có kiểu là **void**.
- Ngược lại, hàm phải trả về giá trị có kiểu cùng với kiểu trả về đã khai báo.
- Phát biểu **return** nhằm dừng thực thi hàm, trở về nơi gọi nó; và còn được dùng để trả giá trị (tính toán được) về cho nơi gọi hàm.
- Trong một hàm, có thể có nhiều phát biểu return, nhưng chỉ 1 phát biểu return được thực thi.
- Trong một phát biểu return chỉ có 1 giá trị được trả về.

Ví dụ:

- Trong các định nghĩa hàm sau đây, có những định nghĩa hàm không hợp lệ.

```
int f1()      {}     //?
void f2()     { return 0;}
   //?
int f3()      { return ; }
   //?
```

```
void f4()     { return;}
int f5()      { return 0;}
int f6(){ return 0.5;} //?
```

```
int IsPrime(int n)
{
        if (n <= 1)
                return 0;
        for (int i= 2; i<n; i++)
        if (n%i == 0)
                return 0;
        return 1;
}
```

```
void main()
{
  int n;
  cout<<"Hay nhap 1 so nguyen:";
  cin>>n;
  if (IsPrime(n) == 1)
     cout<<n<<"la so nguyen to";
  else
     cout<<n<<"KHONG la so nguyen to";
}
```

6. Phạm vi của biến

- Có 3 nơi cơ bản mà biến được khai báo:
  - trong một hàm,
  - trong định nghĩa danh sách tham số của hàm,
  - ngoài tất cả các hàm.
- Tương ứng với vị trí xuất hiện của biến, có:
  - Biến địa phương
  - Tham số hình thức
  - Biến toàn cục

Biến địa phương

- Các biến (hằng) được khai báo *trong một hàm* được gọi là các *biến địa phương*.
- Biến có thể được khai báo bất kì đâu trong hàm, chỉ các phát biểu trong *cùng khối* mới có thể truy xuất.
- Biến địa phương chỉ tồn tại (thời gian sống) trong khi *khối* lệnh có chứa khai báo biến đó thực thi.
- Khối lệnh hoặc hàm khác không thể truy xuất chúng.

Tham số hình thức

- Dùng tham số hình thức để chuyển các giá trị cho hàm.
- Các tham số hình thức được dùng *như biến địa phương*.Nghĩa là:
  - biến chỉ được sinh ra khi hàm được gọi thực thi và bị hủy khi hàm thực thi xong.
  - Chỉ được truy xuất bởi các phát biểu trong hàm đó.

Biến toàn cục

- Để tạo biến (hằng) toàn cục, khai báo biến (hằng) ngoài tất cả các hàm.
- Biến toàn cục có thể được truy xuất bởi các phát biểu ở bất kì đâu trong chương trình kể từ sau khi nó được định nghĩa (khai báo).
- Thời gian sống của biến toàn cục là suốt quá trình chương trình thực thi.

Ví dụ

```
#include<iostream.h>
double a = 3.0, r = 0.0;
double f1(){
      r = 2*a;
      return r;
}
double f2(){
      return (a*a);
}
```

```
void main()
{
      cout<<f1();
      cout<<f2();
      cout<<r;
}
```

Trường hợp trùng tên biến

- Không thể định nghĩa hai biến trùng tên trong cùng khối.
- Nếu có biến địa phương trong hàm trùng tên với biến toàn cục, thì trong hàm đó, mặc định sẽ truy xuất đến biến địa phương.
- Để truy xuất đến biến toàn cục, ta dùng phép toán phân định phạm vi, là dấu hai chấm kép [::] ngay trước tên biến.

Ví dụ

```
#include<iostream.h>
double a = 3.0, r = 0.0;
double f1(double a)
{
      double r  = 2*a;
   return r;
}
double f2(double r)
{
   ::r  = r*r;
      return ::r;
}
void main()
{
   cout<<f1(a);
   cout<<f2(a);
   cout<<r;
}
```

```
//Chú ý trường hợp:

int f3(int a)
   {
      int b = a;
      if (a<0)
      int b = -a;
      return b;
   }
//Chạy chương trình bằng "tay"
   xem thử !
```

7. Biến mảng

- Mảng là một nhóm các biến có cùng tên, cùng kiểu dữ liệu.
- Mảng có thể là một hoặc nhiều chiều.
- Mỗi phần tử (mỗi biến) của mảng được truy xuất thông qua chỉ số

- Mảng *một chiều* **a** có 8 phần tử, hiện có 4 phần tử đã được gán giá trị.

Ví dụ: **a**

3	-2	4		
7	-8	6		

☞ Mảng *2 chiều* **a** có 4×5=20 phần tử, hiện có 2×3=6 phần tử đã được gán giá trị.

Định nghĩa mảng một chiều (1-D)

Mẫu khai báo: <*kiểu dữ liệu*> **`TenBien[MAX] ;`**

- **`TenBien`**: theo nguyên tắc đặt tên,
- **`MAX`**: phải là ***hằng*** (hằng khai báo, hằng giá trị,…)

Ví dụ:

```
#define MAX 20
int a[10];
int b[MAX]
double m[MAX]
```

Định nghĩa mảng một chiều (2)

Với khai báo trên:

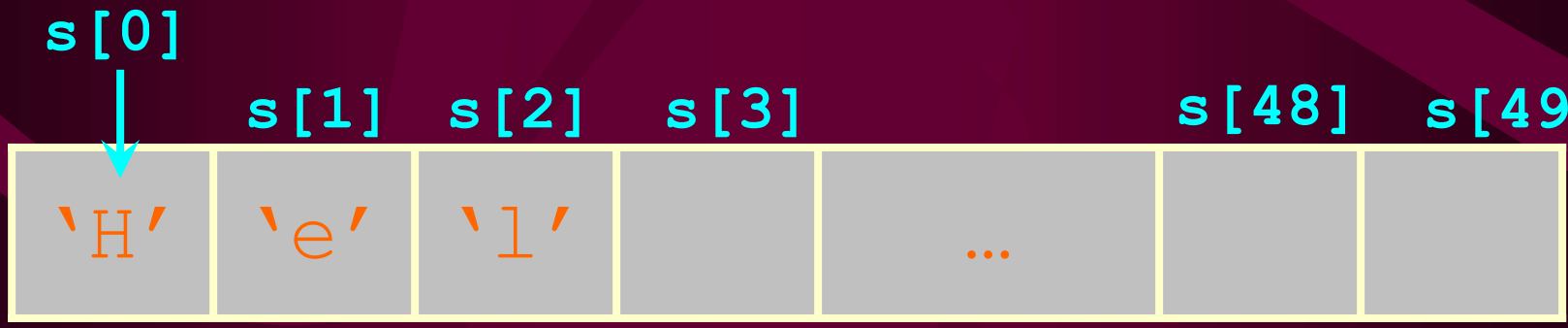
- Là ta đã định nghĩa một mảng có MAX phần tử *(= đã định nghĩa MAX biến)*
- Tất cả phần tử *(biến)* này có cùng kiểu dữ liệu.
- Mỗi phần tử của mảng có *chỉ số* từ `[0]`→`[MAX-1]`.

Định nghĩa mảng một chiều (3)

Ví dụ:

`char s[50];` ⇒ định nghĩa 50 biến kiểu *char* gồm: `s[0], s[1], s[2],... s[49]`.

`s[0] = 'H', s[1]='e', s[2]='l';`



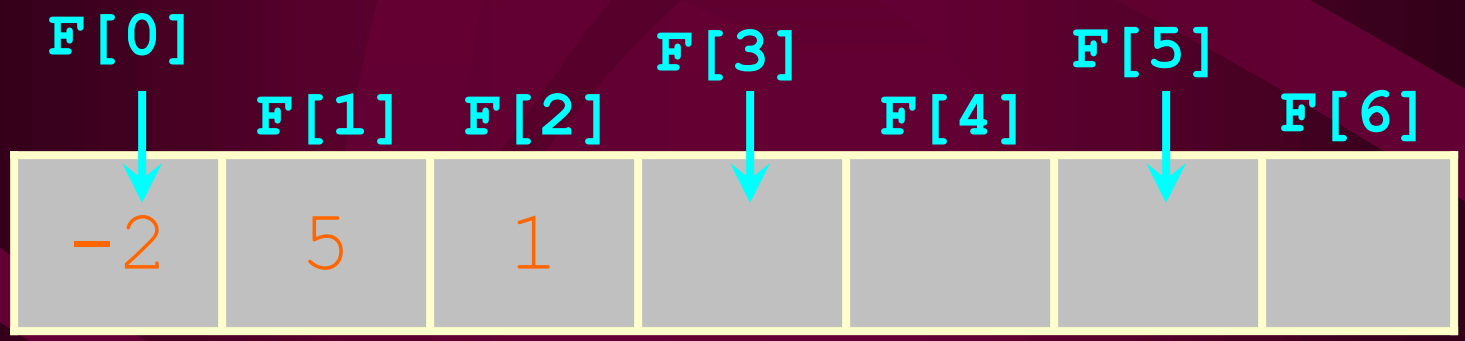
Định nghĩa mảng một chiều (3)

Ví dụ:

`int F[7];` ⇒ định nghĩa 7 biến kiểu *int* gồm:

`F[0], F[1],f[2],… F[6].`

`F[0]=-2, F[1]=5, F[2]=1;`



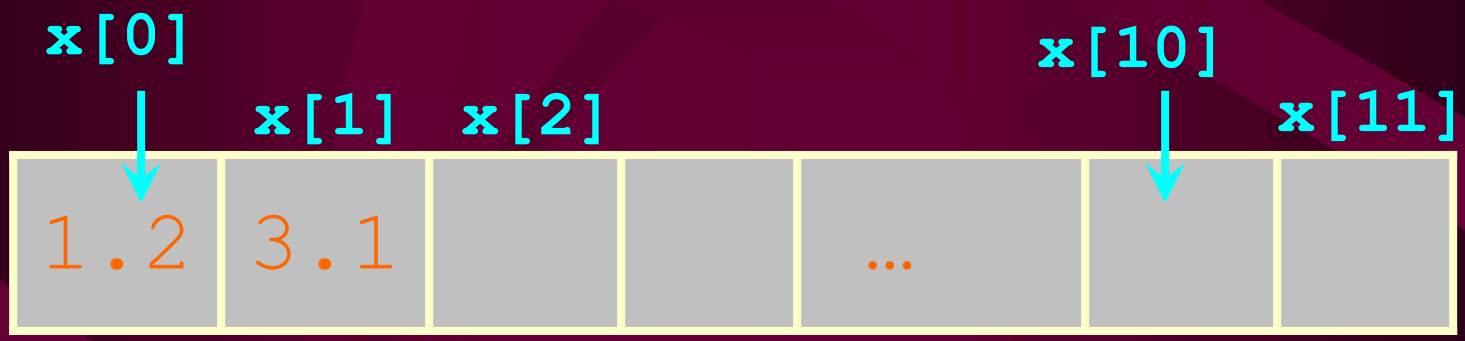
Định nghĩa mảng một chiều (4)

Ví dụ:

`double x[12];` ⇒định nghĩa 12 biến kiểu *double*

gồm: `x[0], x[1], x[2],… x[11]`.

```
x[0]=1.2, x[1]=3.1;
```



Định nghĩa mảng một chiều (5)

Ví dụ:

- Để khai báo mảng số nguyên có 25 phần tử với tên là **mg**,

= Khai báo 25 biến có tên chung là **mg**:

```
int mg[25];
```

hay:

```
#define MAX 25
int mg[MAX];
```

Lưu trữ

Khi mảng được định nghĩa,

- Vùng nhớ cho các phần tử của mảng được cấp phát là các ô nhớ liền nhau.
- Mỗi phần tử chiếm số bytes tùy thuộc kiểu dữ liệu.
- `sizeof(`<*tên biến mảng*>`);` → cho biết tổng số bytes của mảng.

Lưu trữ

Ví dụ:

```
double x[25];
```

$\Rightarrow$ `sizeof(x)` $\equiv 200$

$\Rightarrow$ `sizeof(x[0])` $\equiv$ `sizeof(x[1])` $\equiv \ldots \equiv 8$

```
long a[20];
```

$\Rightarrow$ `sizeof(a)` $\equiv 80$

$\Rightarrow$ `sizeof(a[0])` $\equiv$ `sizeof(a[1])` $\equiv \ldots \equiv 4$

Lưu ý (1)

Không thể dùng biến mảng như thông thường.

```
int a[6];
a = -2;     //err
cout<<a;    //err
```

Mà phải dùng từng phần tử trong chúng.

Mỗi phần tử được dùng như một biến đơn.

```
int a[6];
a[1] = -2;
cin>>a[0];
cout<<a[0]*a[1];
```

Lưu ý (2)

- Ta thường dùng phát biểu lặp để truy xuất các phần tử của mảng.
- Đồng thời dùng thêm một biến kiểu nguyên, cho biết số phần tử của mảng thực sự đang được dùng.

d:\temp\vdmang01.cpp

```
#include<iostream.h>
#define MAX 20
void main()
{
    int a[MAX], n;
    do{
        cout<<"Ban muon bao nhieu phan tu: ";
        cin>>n;
    }while(n>MAX || n<0);
    for (int i=0; i<n; i++)
    {
        cout<<"Nhap a["<<i<<"]: ";
        cin>>a[i];
    }
    cout<<"Cac phan tu cua mang: ";
    for (i=0; i<n; i++)
        cout<<a[i]<<"\t";
}
```

Vd01

Khởi tạo

Một số cách mà ngôn ngữ C cho phép khởi tạo giá trị của mảng:

(1) Số phần tử = số giá trị

```
double x[3]= {1.0, 2.71828, 3.14159};
```

⇔ `x[0]= 1.0, x[1]= 2.71828,...`

⇒ Cấp phát 3 ô nhớ kiểu double, gán 3 giá trị cho chúng.

Khởi tạo

(2) Số phần tử > số giá trị

```
char s[10]= {'c','h','a','o','e','m'};
```

⇔ `s[0]= 'c', s[1]= 'h',…,s[5]='m';`

⇒ Cấp phát 10 ô nhớ kiểu char, lần lượt gán 6 giá trị cho 6 ô nhớ đầu

Khởi tạo

(3) Không định số phần tử

```
int SoNgay[]= {31, 28, 31, 30, 31, 30,
               31, 31, 30, 31, 30, 31};
```

⇔ Cấp phát số phần tử = số giá trị được gán.

⇒ Cấp phát 12 ô nhớ kiểu **`int`**, lần lượt gán 12 giá trị cho chúng.

d:\temp\vdmang02.cpp

Vd02

```
#include<iostream.h>
void main()
{
    int SoNgay[]= {31, 28, 31, 30, 31, 30,
                   31, 31, 30, 31, 30, 31};
    int n = sizeof(SoNgay)/sizeof(SoNgay[0]);

    for (int i=0; i<n; i++)
    {
        cout<<"Thang "<<i+1<<" co "
            <<SoNgay[i]<<" ngay !\n";
    }
    cout<<"(Xet thang 2 cua nam khong nhuan)";
}
```

8. Biến mảng là tham số của hàm

- Tham số được khai báo như khai báo biến mảng.
- Có thể không cần xác định số phần tử của mảng
- Mảng được chuyển tham chiếu

Vd: Hàm xuất nội dung một mảng các số nguyên:

```
void xuatMang(int a[], int N);
```

Vd: Hàm nhập nội dung một mảng các số nguyên:

```
void nhapMang(int a[], int &N);
```

// lưu ý khi nhập N trong hàm

d:\temp\vdmang03.cpp

Vd03

```
#include<iostream.h>
#define MAX 20
void nhap(int a[], int &n);
void xuat(int a[], int n);
void main()
{
      int a[MAX], n;
      nhap(a,n);
      xuat(a,n);
}
```

```
void nhap(int a[], int &n)
{
    do{
        cout<<"Ban muon bao nhieu phan tu: ";
        cin>>n;
    }while(n>MAX || n<0);
    for (int i=0; i<n; i++)
    {
        cout<<"Nhap a["<<i<<"]: ";
        cin>>a[i];
    }
}
void xuat(int a[], int n)
{
    cout<<"Cac phan tu cua mang: ";
    for (int i=0; i<n; i++)
        cout<<a[i]<<"\t";
}
```

- Tham số được khai báo như khai báo biến mảng.
- Có thể không cần xác định số phần tử của mảng
- Mảng được chuyển tham chiếu

Vd: Hàm tính tổng các phần tử của mảng nguyên

(vd: **a = [3 7 4 9] → S= 23**)

```
int tinhTong(int a[], int N);
```

d:\temp\vdmang04.cpp

Vd04

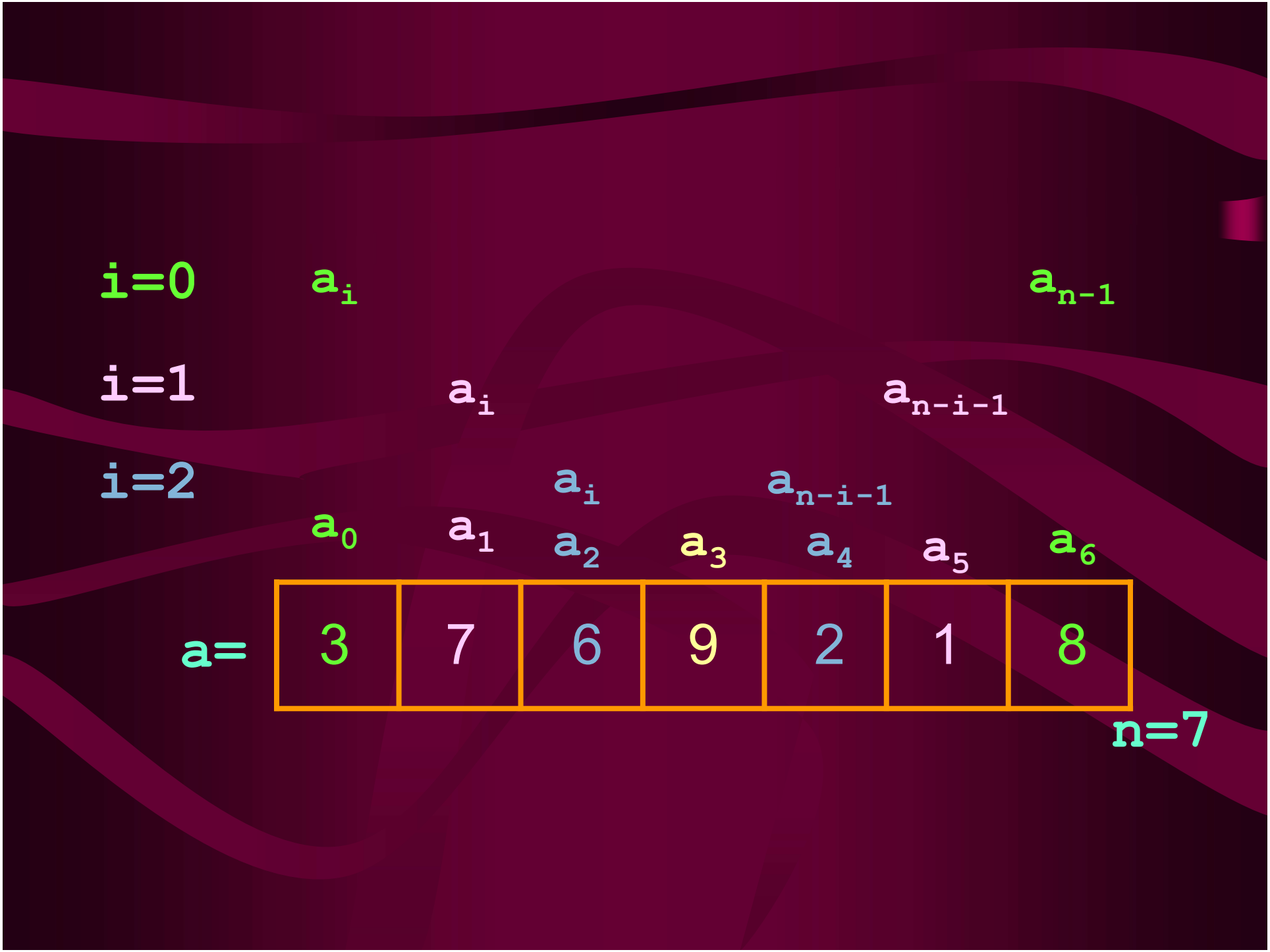
```
#include<iostream.h>
#define MAX 20
void nhap(int a[], int &n);
void xuat(int a[], int n);
int tinhTong(int a[], int n);
void main()
{
     int a[MAX], n;
     nhap(a,n);
     xuat(a,n);
     int S= tinhTong(a,n);
     cout<<"\n Tong cac phan tu cua mang: "<<S;
}
int tinhTong(int a[], int n)
{
     int S= 0;
     for (int i=0; i<n; i++)
          S += a[i];
     return S;
}
void nhap(int a[], int &n)
{
```

8. Biến mảng là tham số của hàm

Vd1: Đảo ngược nội dung của mảng. Ví dụ:

a= [3 7 4 9] → a = [9 4 7 3]

```
void daoMang(int a[], int N)
   {
      for (int i= 0; i<N/2; i++)
          HoanVi(a[i], a[n-i-1]);
   }
```


i=0
a_i
a_{n-1}
i=1
a_i
a_{n-i-1}
i=2
a_i
a_{n-i-1}
a_0
a_1
a_2
a_3
a_4
a_5
a_6
a=
3
7
6
9
2
1
8
n=7

d:\temp\vdmang05.cpp

Vd05

```
#include<iostream.h>
#define MAX 20
void nhap(int a[], int &n);
void xuat(int a[], int n);

void hoanVi(int &x, int &y);
void daoMang(int a[], int N);

void main()
{
    int a[MAX], n;
    nhap(a,n);
    xuat(a,n);
    daoMang(a,n);
    cout<<"\n Sau khi mang duoc dao ";
    xuat(a,n);
}
void daoMang(int a[], int N)
```

Vd2: Hàm sắp xếp tăng các phần tử của mảng. Ví dụ:

a= [3 7 4 9] → a = [3 4 7 9]

```
    sapTang(a,n);
    cout<<"\nSau khi duoc sap ";
    xuat(a,n);
}
void sapTang(int a[], int N)    //interchange
{
    for (int i= 0; i<N-1; i++)
        for (int j= i+1; j<N; j++)
            if (a[i] > a[j])
                hoanVi(a[i], a[j]);
}
```

d:\temp\vdmang06.cpp

Vd06

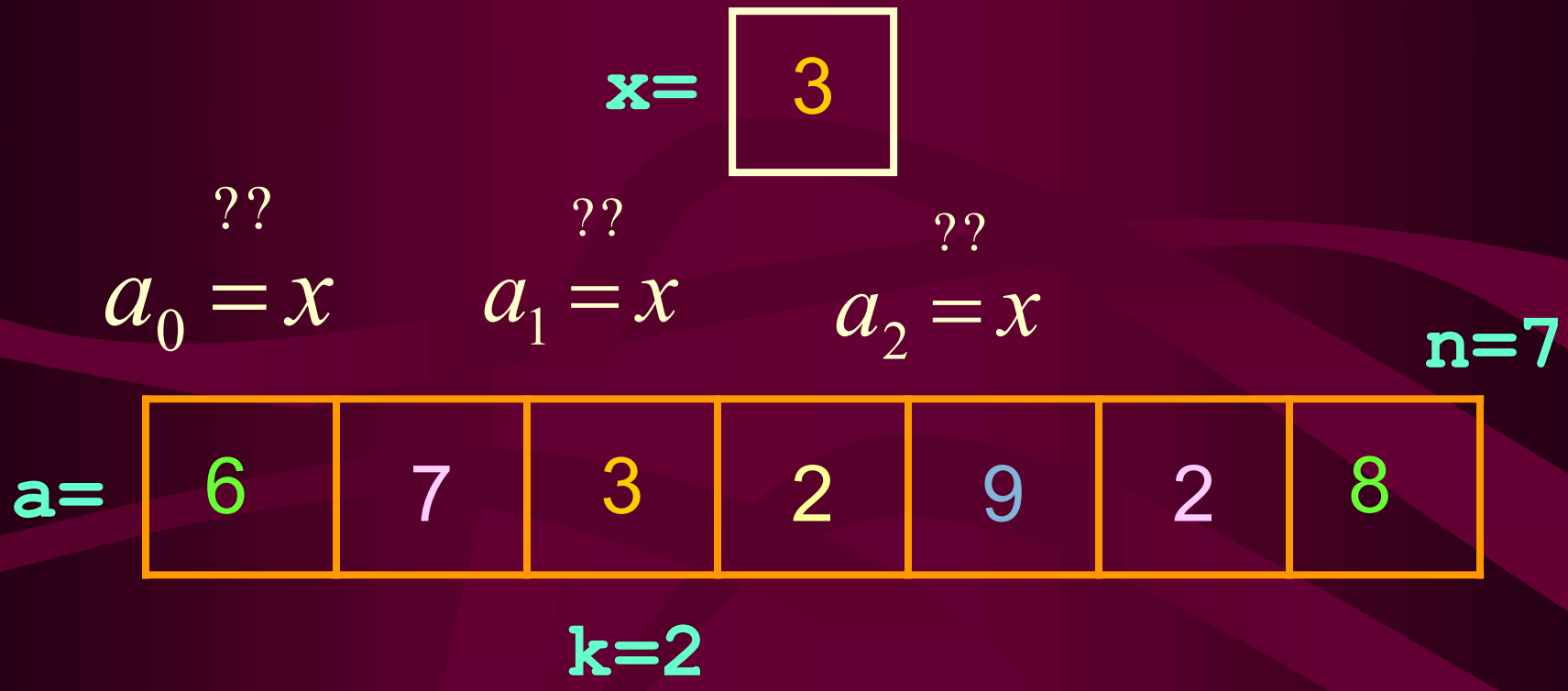
```
#include<iostream.h>
#define MAX 20
void nhap(int a[], int &n);
void xuat(int a[], int n);

void sapTang(int a[], int n);
void hoanVi(int &x, int &y);

void main()
{
    int a[MAX], n;
    nhap(a,n);
    xuat(a,n);
    sapTang(a,n);
    cout<<"\nSau khi duoc sap ";
    xuat(a,n);
}
void sapTang(int a[], int N)     //interchange
```

Vd3: Tìm xem một giá trị x có trong mảng ? Ví dụ:

a= [3 7 4 9], x= 4

→ **x** nằm ở vị trí 2 trong mảng **a**

```
int timKiem(int a[], int n, int x)
{
    for (int i=0; i<n; i++)
        if (a[i]==x)
            return i;
    return -1;
}
```


x=
3
??
$a_0 = x$
??
$a_1 = x$
??
$a_2 = x$
n=7
a=
6
7
3
2
9
2
8
k=2

x= 4

$a_0 \overset{??}{=} x \quad a_1 \overset{??}{=} x \quad a_2 \overset{??}{=} x \quad a_3 \overset{??}{=} x \quad a_4 \overset{??}{=} x \quad a_5 \overset{??}{=} x \quad a_6 \overset{??}{=} x$

a=

6	7	3	2	9	2	8

n=7

k=-1 *(khong co x trong mang)*

d:\temp\vdmang07.cpp

Vd07

```
#include<iostream.h>
#define MAX 20
void nhap(int a[], int &n);
void xuat(int a[], int n);
int timKiem(int a[], int n, int x);

void main()
{
    int a[MAX], n;
    nhap(a,n);
    xuat(a,n);
    int k= timKiem(a,n,4);
    if (k!=-1)
        cout<<"\n Tim thay x tai vi tri "<<k;
    else
        cout<<"\n Khong co x trong mang !";
}
```

d:\temp\vdmag07b.cpp

Vd07b

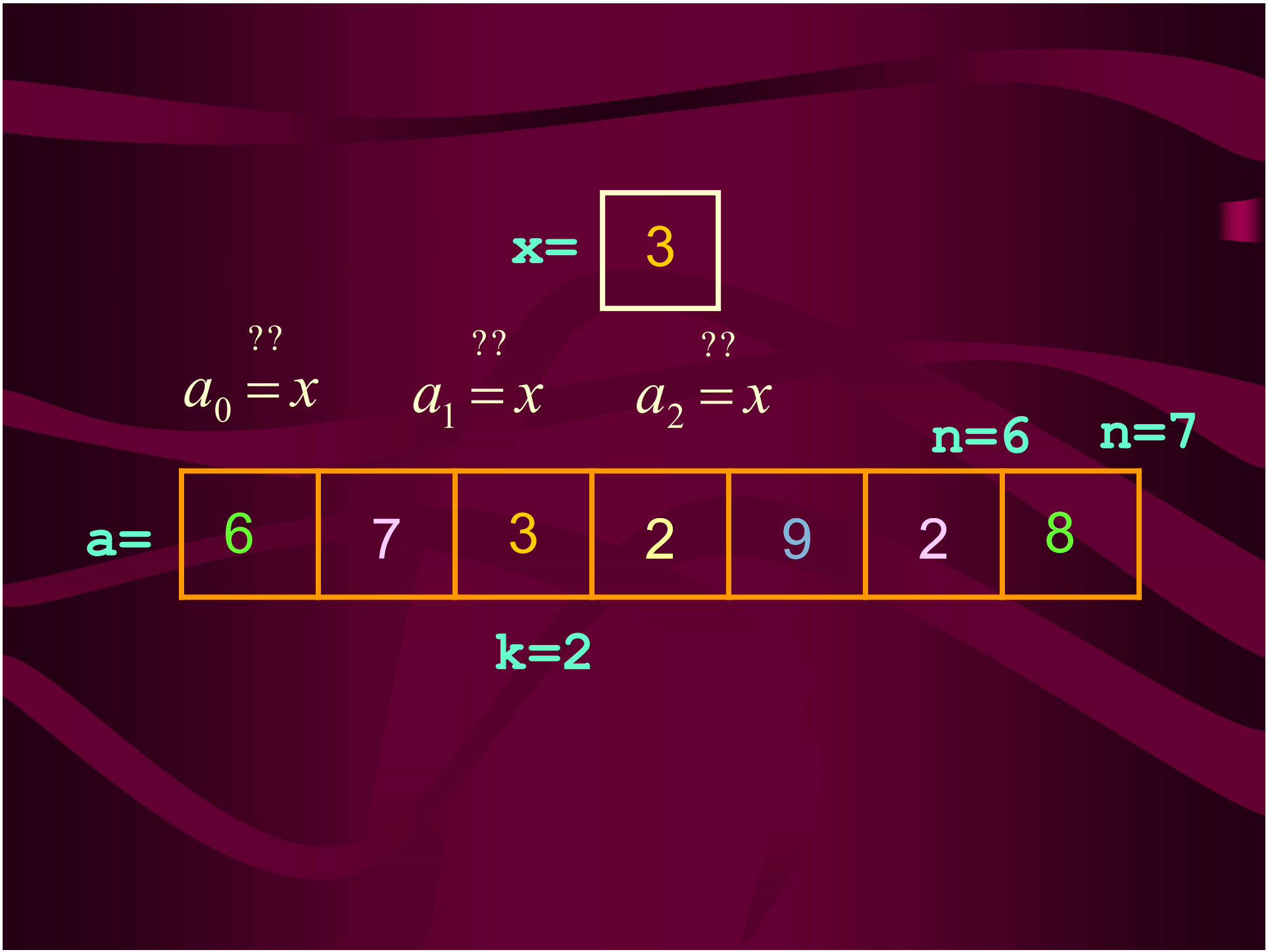
```
#include<iostream.h>
#define MAX 20
void nhap(int a[], int &n);
void xuat(int a[], int n);
int timKiem(int a[], int n, int x);

void main()
{
     int a[MAX], n, x, k;
     nhap(a,n);
     xuat(a,n);
     cout<<"\n Ban muon tim phan tu x= ";
     cin>>x;
     k= timKiem(a,n,x);
     if (k!=-1)
          cout<<"\n Tim thay x tai vi tri "<<k;
     else
          cout<<"\n Khong co x trong mang !";
}
```

Vd4: Xóa phần tử x khỏi mảng (nếu có). Ví dụ:

a= [6 7 3 2 9 2 8],n=7

x= 3 → a= [6 7 2 9 2 8], n= 6

x= 2 → a= [6 7 9 8], n= 4


x=
3
??
$a_0 = x$
??
$a_1 = x$
??
$a_2 = x$
n=6
n=7
a=
6
7
3
2
9
2
8
k=2

e:\vdmag08a.cpp

```
#include<iostream.h>
#define MAX 20
void nhap(int a[], int &n);
void xuat(int a[], int n);

int timKiem(int a[], int n, int x);
void xoaX(int a[], int &n, int x);

void main()
{
        int a[MAX], n, x;
        nhap(a,n);
        xuat(a,n);
        cout<<"\n Nhap phan tu muon xoa: ";
        cin>>x;
        xoaX(a,n,x);
        cout<<"\n Sau khi da xoa x: ";
        xuat(a,n);
}
void xoaX(int a[], int &N, int x)
{
```

Xóa phần tử x khỏi mảng (nếu có).

```
    xoaX(a,n,x);
    cout<<"\n Sau khi da xoa x: ";
    xuat(a,n);
}
void xoaX(int a[], int &N, int x)
{
    int k, i;
    k= timKiem(a,N,x);
    if (k!=-1)
    {
        for (i= k; i<=N-2; i++)
            a[i]= a[i+1];
        --N;
    }
}
```

Vd08a

Xóa tất cả phần tử x khỏi mảng (nếu có).

```
    xoaX(a,n,x);
    cout<<"\n Sau khi da xoa x: ";
    xuat(a,n);
}
void xoaX(int a[], int &N, int x)
{
    int k, i;
    do
    {
        k= timKiem(a,N,x);
        if (k!=-1)
        {
            for (i= k; i<=N-2; i++)
                a[i]= a[i+1];
            --N;
        }
    }
    while(k!=-1);
}
```

Vd08b