程序设计基础

第七章: 数组

刘新国

浙江大学计算机学院
浙江大学 CAD&CG 国家重点实验室

November 18, 2021

内容提要

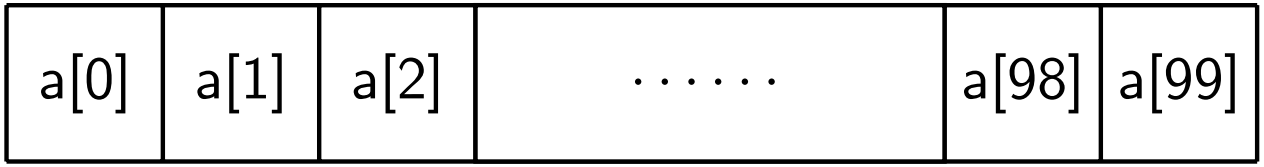
一维数组

二维数组

字符数组和字符串

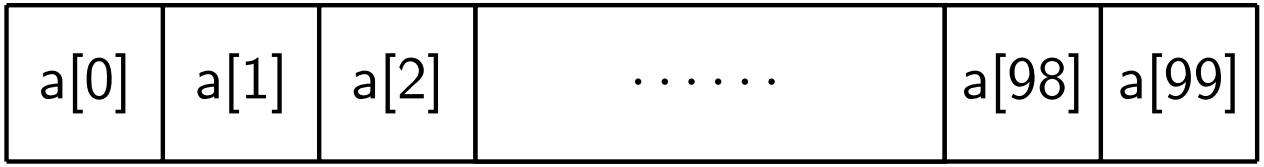
内容提要

一维数组的定义和引用



什么是数组？

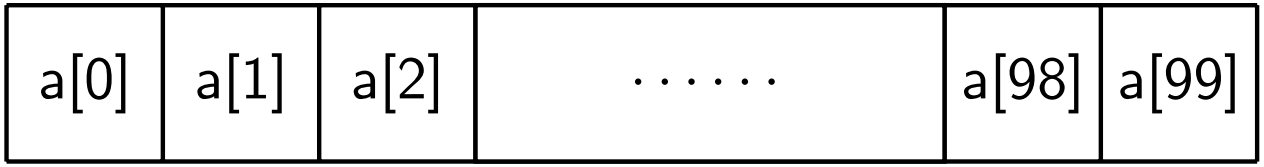
一维数组的定义和引用



什么是数组？

- 一组相同类型数据的有序集合

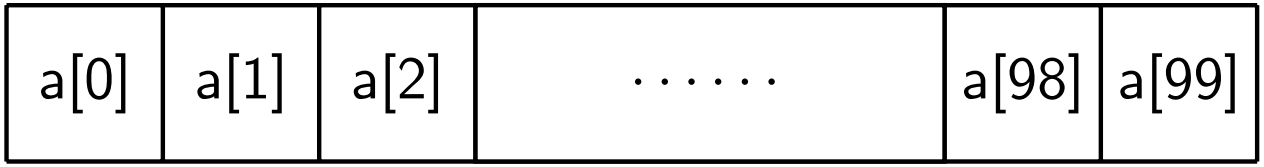
一维数组的定义和引用



什么是数组？

- 一组相同类型数据的有序集合
- 元素在内存中连续存放，具有相同的类型

一维数组的定义和引用



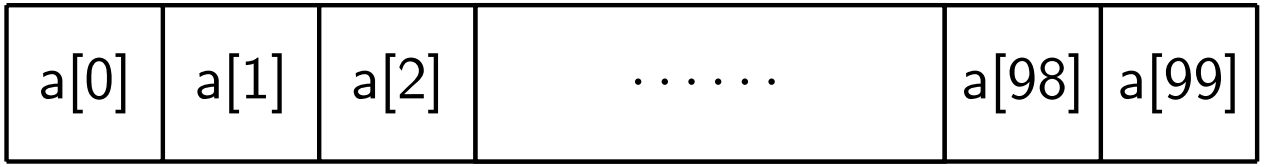
什么是数组？

- 一组相同类型数据的有序集合
- 元素在内存中连续存放，具有相同的类型

一维数组的定义

类型名　数组名 [数组长度]

一维数组的定义和引用



什么是数组？

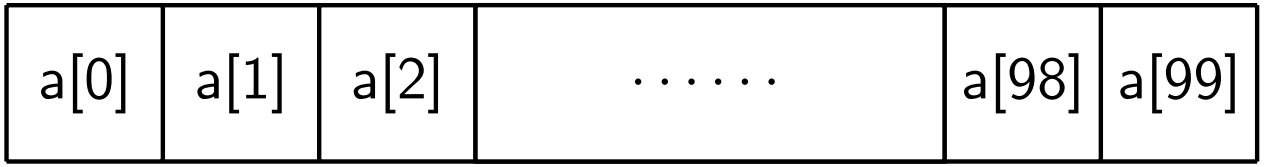
- 一组相同类型数据的有序集合
- 元素在内存中连续存放，具有相同的类型

一维数组的定义

类型名　数组名 [数组长度]

- 类型名 - 定义数组元素的类型

一维数组的定义和引用



什么是数组？

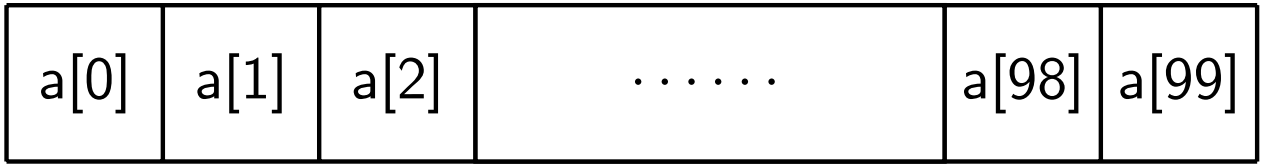
- 一组相同类型数据的有序集合
- 元素在内存中连续存放，具有相同的类型

一维数组的定义

类型名　数组名 [数组长度]

- 类型名 - 定义数组元素的类型
- 数组名 - 定义数组的名称，符合标识符的语法

一维数组的定义和引用



什么是数组？

- 一组相同类型数据的有序集合
- 元素在内存中连续存放，具有相同的类型

一维数组的定义

类型名　数组名 [数组长度]

- 类型名 - 定义数组元素的类型
- 数组名 - 定义数组的名称，符合标识符的语法
- 数组长度 - 整数类型的常量表达式

一维数组的定义和引用

```
int    a[10];    /* 长度为10的整数数组 */
char   c[20*10];/* 长度为200的字符数组 */
int    n = 10;
float  f[n];     /* 非法定义，因为长度
                    n不是常量表达式 */
```

一维数组的定义和引用

```
int    a[10];    /* 长度为10的整数数组 */
char   c[20*10];/* 长度为200的字符数组 */
int    n = 10;
float  f[n];     /* 非法定义，因为长度
                    n不是常量表达式 */
```

数组元素的引用和使用

数组名 [下标表达式]

一维数组的定义和引用

```
int    a[10];    /* 长度为10的整数数组 */
char   c[20*10];/* 长度为200的字符数组 */
int    n = 10;
float  f[n];     /* 非法定义，因为长度
                    n不是常量表达式 */
```

数组元素的引用和使用

数组名 [下标表达式]

```
a[0] = 5; /* 将5赋值给a[0]，即首元素*/
a[1] = a[2] + a[3];
scanf("%f%c", &a[0], &c[0]); /* 取地址 */
for( n = 0; n<10; n++ )
     a[n] = n*100;
```

- 像使用普通变量一样，使用数组元素

一维数组的定义和引用

数组元素的引用和使用

数组名 [下标表达式]

一维数组的定义和引用

数组元素的引用和使用

数组名 [下标表达式]

- 下标的范围为[0，　长度-1]，不能越界

一维数组的定义和引用

数组元素的引用和使用

数组名 [下标表达式]

- 下标的范围为[0，　长度-1]，不能越界
- 下标表达式不限于常量，常常是变量

一维数组的定义和引用

数组元素的引用和使用

数组名 [下标表达式]

- 下标的范围为[0, 长度-1]，不能越界
- 下标表达式不限于常量，常常是变量

读入 10 个整数，并输出大于平均值的数

```
int a[10], i, sum, average;
for( sum=i=0; i<10; i++ ) {
    scanf("%d", &a[i])
    sum += a[i];
}
average = sum / 10;
for( i=0; i<10; i++ )
    if( a[i]>average ) printf("%5d", a[i]);
```

一维数组的初始化

和普通变量一样，定义数组时可以同时赋予初值，其一般形式为：

$$\text{类型名数组名 [长度值]} = \text{初值表;}$$

初值表的格式为：

$$\text{初值 1, 初值 2, ..., 初值 n}$$

一维数组的初始化

和普通变量一样，定义数组时可以同时赋予初值，其一般形式为：

类型名数组名 [长度值] = 初值表;

初值表的格式为：

初值 1, 初值 2, ..., 初值 n

例如：

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
float b[] = {4, 6, 8, 10}; /* 长度为4。如果没
    有给出数组长度，那么元素个数为初值的个数 */
static double c[10];
```

一维数组的初始化

和普通变量一样，定义数组时可以同时赋予初值，其一般形式为：

类型名数组名 [长度值] = 初值表;

初值表的格式为：

初值 1, 初值 2, ..., 初值 n

例如：

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
float b[] = {4, 6, 8, 10}; /* 长度为4。如果没
    有给出数组长度，那么元素个数为初值的个数 */
static double c[10];
```

对于静态变量，如果没有给出初值，那么它们的初值为 0，即：所有的 bit 都是 0

程序解析 7-2，用数组计算和输出斐波那契数列

```c
#include <stdio.h>
int main(void)
{
    int i, fib[10] = {1,1}; /*部分初始化*/
    /*部分初始化时，未初始化元素值为0*/

    for( i = 2; i<10; i++ )
        fib[i] = fib[i-1] + fib[i-2];

    for( i = 0; i<10; i++ )
        printf("%6d%c", fib[i],
                i%5!=4 ? ' ' : '\n' );
    return 0;
}
```

程序解析 7-2，用数组计算和输出斐波那契数列

```c
#include <stdio.h>
int main(void)
{
    int i, fib[10] = {1,1}; /*部分初始化*/
    /*部分初始化时，未初始化元素值为0*/

    for( i = 2; i<10; i++ )
        fib[i] = fib[i-1] + fib[i-2];

    for( i = 0; i<10; i++ )
        printf("%6d%c", fib[i],
                i%5!=4 ? ' ' : '\n' );
    return 0;
}
```

数组引用和赋值

程序解析 7-2，用数组计算和输出斐波那契数列

```c
#include <stdio.h>
int main(void)
{
    int i, fib[10] = {1,1}; /*部分初始化*/
    /*部分初始化时，未初始化元素值为0*/

    for( i = 2; i<10; i++ )
        fib[i] = fib[i-1] + fib[i-2];

    for( i = 0; i<10; i++ )
        printf("%6d%c", fib[i],
               i%5!=4 ? ' ' : '\n' );
    return 0;
}
```

数组引用和赋值

条件表达式

程序解析 7-3 顺序查找法

- 首先输入 N 个数据，保存在数组 a[ ] 中；
- 然后输入一个数据 x，并在数组中查找 x 是否存在。
- 如果存在，输出最小的下标。

```
#include <stdio.h>
#define MAXN 5
int main( void )
{
    int a[MAXN], i, n, x;

    printf("Entern n, x:"); /* 提示输入n和x */
    scanf("%d%d", &n, &x);
    printf("Enter %d numbers:", n);
    for( i = 0; i<n; i++ ) /* 输入n个数 */
        scanf("%d", &a[i]);
```

程序解析 7-3 顺序查找法

顺序查找，从 a[0] 开始比较

```
for( i = 0; i<n; i++ )
    if( a[i]==x ) break;
```

程序解析 7-3 顺序查找法

顺序查找，从 a[0] 开始比较

```
for( i = 0; i<n; i++ )
    if( a[i]==x ) break;
```

根据循环结束时的状态，判断和输出查找结果

```
if( i<n )
    printf("The index of %d is %d\n", x, i
        );
else
    printf("%d is Not Found!\n", x);
```

程序解析 7-4 查询数组的最小值和下标

查询数组的最小值，并记录最小值的下标

程序解析 7-4 查询数组的最小值和下标

查询数组的最小值，并记录最小值的下标

顺序查找，从 a[0] 开始比较

- 定义一个变量 m 记录最小值的下标。
- 比较之前，临时的最小值为 a[0]，即 $m = 0$

程序解析 7-4 查询数组的最小值和下标

查询数组的最小值，并记录最小值的下标

顺序查找，从 a[0] 开始比较

- 定义一个变量 m 记录最小值的下标。
- 比较之前，临时的最小值为 a[0]，即 m = 0

```
for( m = 0, i = 1; i<N; i++ )
    if( a[i] <= a[m] ) m = i;
```

程序解析 7-4 查询数组的最小值和下标

查询数组的最小值，并记录最小值的下标

顺序查找，从 a[0] 开始比较

- 定义一个变量 m 记录最小值的下标。
- 比较之前，临时的最小值为 a[0]，即 m = 0

```
for( m = 0, i = 1; i<N; i++ )
    if( a[i] <= a[m] ) m = i;

printf("The minimum value is %d, and its
   index is %d\n", a[m], m);
```

程序解析 7-5 选择法排序

- 找到 a[0], . . ., a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[0] 交换)

程序解析 7-5 选择法排序

- 找到 a[0], . . ., a[N-1] 中最小的元素，并将其交换到最前面（即与 a[0] 交换）
- 找到 a[1], . . ., a[N-1] 中最小的元素，并将其交换到最前面（即与 a[1] 交换）

程序解析 7-5 选择法排序

- 找到 a[0], . . ., a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[0] 交换)
- 找到 a[1], . . ., a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[1] 交换)

. . . . . .

程序解析 7-5 选择法排序

- 找到 a[0], . . ., a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[0] 交换)
- 找到 a[1], . . ., a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[1] 交换)

. . . . . .

- 找到 a[k], . . ., a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[k] 交换)

程序解析 7-5 选择法排序

- 找到 a[0], . . ., a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[0] 交换)
- 找到 a[1], . . ., a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[1] 交换)

  . . . . . .
- 找到 a[k], . . ., a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[k] 交换)

  . . . . . .

程序解析 7-5 选择法排序

- 找到 a[0], . . ., a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[0] 交换)
- 找到 a[1], . . ., a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[1] 交换)

  . . . . . .
- 找到 a[k], . . ., a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[k] 交换)

  . . . . . .
- 找到 a[N-2] 和 a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[N-2] 交换)

程序解析 7-5 选择法排序

- 找到 a[0], . . ., a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[0] 交换)
- 找到 a[1], . . ., a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[1] 交换)

  . . . . . .
- 找到 a[k], . . ., a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[k] 交换)

  . . . . . .
- 找到 a[N-2] 和 a[N-1] 中最小的元素, 并将其交换到最前面 (即与 a[N-2] 交换)

程序解析 7-5 选择法排序

- 找到 a[0], ..., a[N-1] 中最小的元素，并将其交换到最前面 (即与 a[0] 交换)
- 找到 a[1], ..., a[N-1] 中最小的元素，并将其交换到最前面 (即与 a[1] 交换)

  ……
- 找到 a[k], ..., a[N-1] 中最小的元素，并将其交换到最前面 (即与 a[k] 交换)

  ……
- 找到 a[N-2] 和 a[N-1] 中最小的元素，并将其交换到最前面 (即与 a[N-2] 交换)

```
for( k = 0; k<N-1; k++ ) {
    找到a[k], ..., a[N-1]中最小的元素
    并将其交换到最前面 (即与a[k]交换)
}
```

程序解析 7-5 选择法排序

```
for( k = 0; k<N-1; k++ ) {
    找到a[k], ..., a[N-1]中最小的元素
    并将其交换到最前面 (即与a[k]交换)
}
```

程序解析 7-5 选择法排序

```
for( k = 0; k<N-1; k++ ) {
     找到a[k], ..., a[N-1]中最小的元素
     并将其交换到最前面 (即与a[k]交换)
}
```

转换成 C 语言代码

```
for( k = 0; k<N-1; k++ ) {
     /*找a[k],...,a[N-1]中的最小元素 */
     for( m = k, i = k+1; i<N; i++ )
          if( a[i] < a[m] ) m = i;
     /*交换a[m]与a[i]交换 */
     temp = a[m]; a[m] = a[i]; a[i] = temp;
}
```

完整的程序见例 7-5

程序解析 7-6 二分查找法

假设有一个排序过的数组，任务是在该数组中快速查找一个给定的数值。

数组为 a[0]-a[9]

```
#define N 10
int main()
{
    int low, high, mid;
    int a[N] = {1,2,3,4,5,6,7,8,9,10}, x; //有
        序数组
```

程序解析 7-6 二分查找法

假设有一个排序过的数组，任务是在该数组中快速查找一个给定的数值。

数组为 a[0]-a[9]

```
#define N 10
int main()
{
    int low, high, mid;
    int a[N] = {1,2,3,4,5,6,7,8,9,10}, x; //有
        序数组
```

输入待查找的数值 x

```
    printf("Enter x: ");
    scanf("%d", &x);
```

程序解析 7-6 二分查找法

二分法查找：逐步缩小查找范围，直到找到了，或范围为空

初始化查找范围为整个数组

```
low = 0;     //查找起点
high = N-1;//查找终点
```

程序解析 7-6 二分查找法

二分法查找：逐步缩小查找范围，直到找到了，或范围为空

初始化查找范围为整个数组

```
low = 0;    //查找起点
high = N-1;//查找终点
```

不断地比较测试，缩小查找范围

```
while( low<=high ) {
    mid = (low + high)/2;
    if( x==a[mid] )
        break; //找到了!
    else if ( x<a[mid] )
        high = mid-1;//到左侧继续查找
    else
        low = mid+1; //到右侧继续查找
}
```

程序解析 7-6 二分查找法

二分法查找：逐步缩小查找范围，直到找到了，或范围为空

确定和输出查找结果

```
if( low<=high )
    //查找范围不空，但停止了，必然找到了
    printf("Found!, index is %d\n", mid);
else
    //查找范围为空，未找到
    printf("Not found!\n");
```

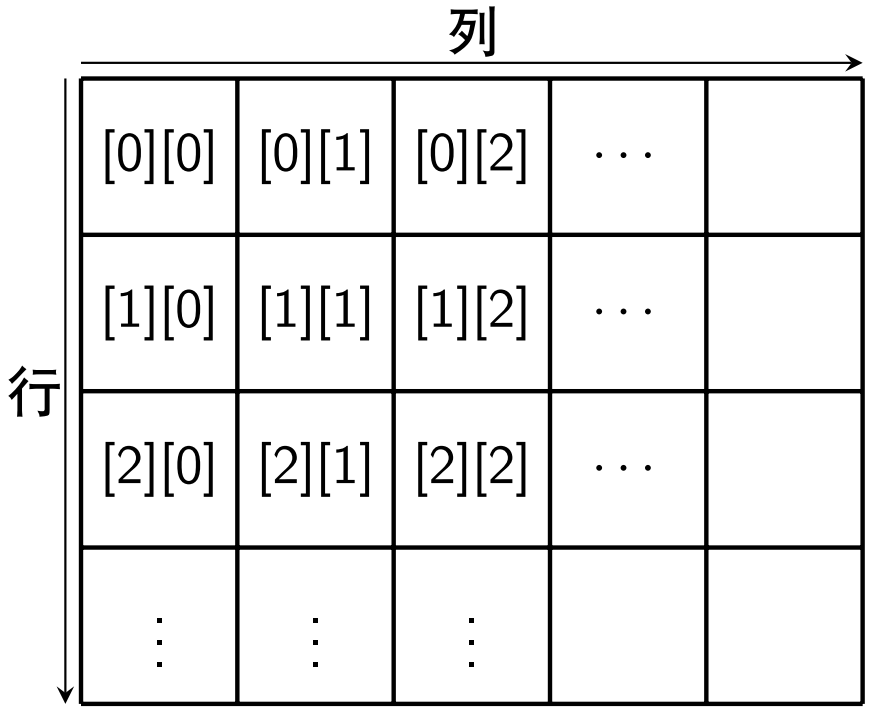
内容提要

一维数组

二维数组

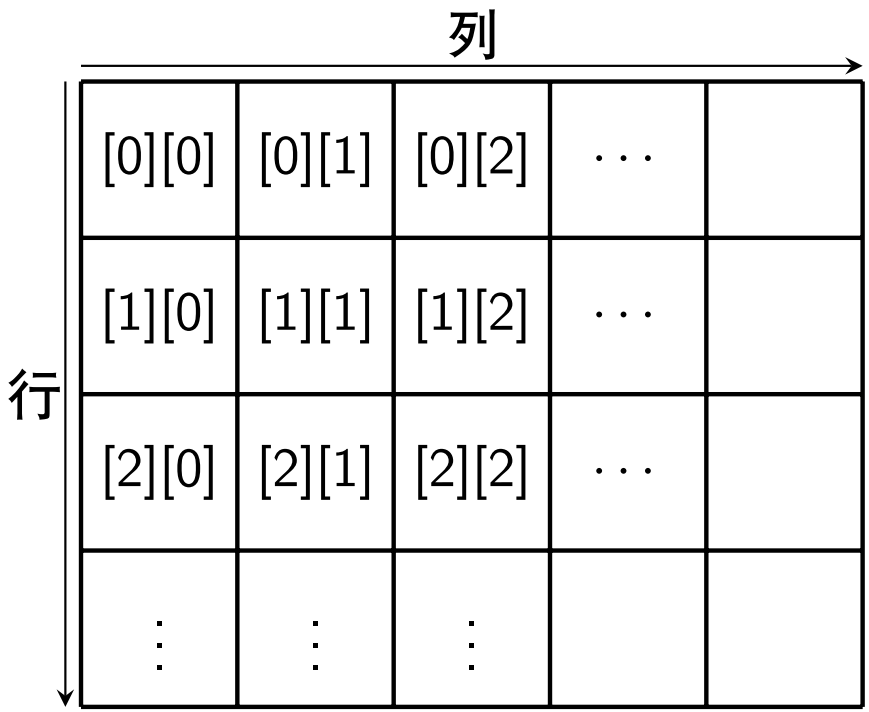
字符数组和字符串

二维数组的定义和引用



什么是二维数组？

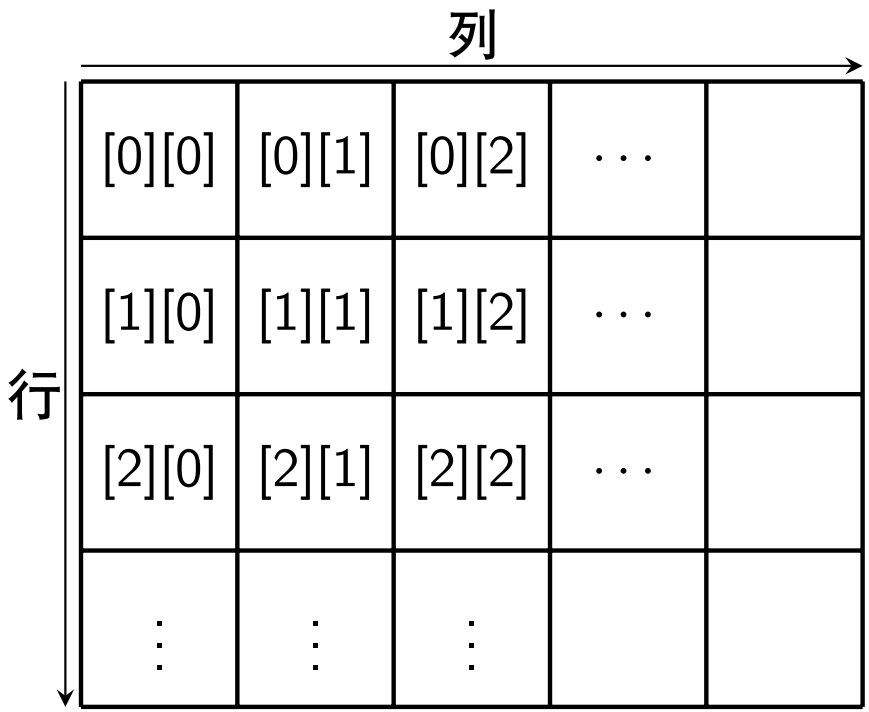
二维数组的定义和引用



什么是二维数组？

- 与一组数组一样，也是相同类型数据的有序集合

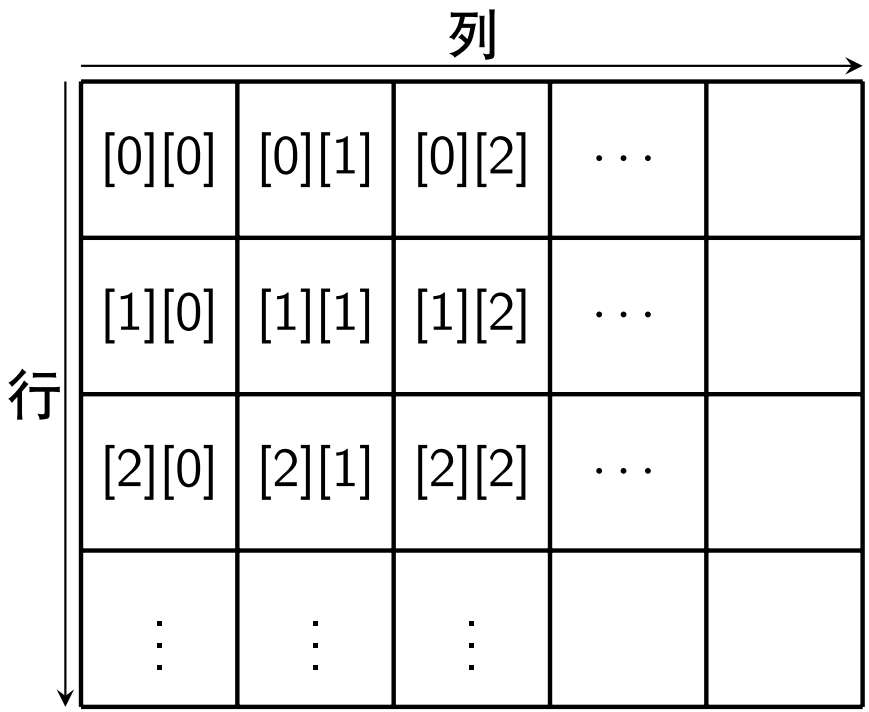
二维数组的定义和引用



什么是二维数组？

- 与一组数组一样，也是相同类型数据的有序集合
- 元素在内存中连续存放，具有相同的类型

二维数组的定义和引用



什么是二维数组？

- 与一组数组一样，也是相同类型数据的有序集合
- 元素在内存中连续存放，具有相同的类型
- 逻辑上具有行、列两个维度

二维数组的定义

类型名　数组名 [行数][列数];

二维数组的定义

类型名　数组名 [行数][列数];

- 类型名 - 定义数组元素的类型

二维数组的定义

类型名　数组名 [行数][列数];

- 类型名 - 定义数组元素的类型
- 数组名 - 定义数组的名称，符合标识符的语法

二维数组的定义

类型名　数组名 [行数][列数];

- 类型名 - 定义数组元素的类型
- 数组名 - 定义数组的名称，符合标识符的语法
- 行数 - 整数类型的常量表达式，指定有多少行

二维数组的定义

类型名　数组名 [行数][列数];

- 类型名 - 定义数组元素的类型
- 数组名 - 定义数组的名称，符合标识符的语法
- 行数 - 整数类型的常量表达式，指定有多少行
- 列数 - 整数类型的常量表达式，指定有多少列

二维数组的定义

类型名　数组名 [行数][列数];

- 类型名 - 定义数组元素的类型
- 数组名 - 定义数组的名称，符合标识符的语法
- 行数 - 整数类型的常量表达式，指定有多少行
- 列数 - 整数类型的常量表达式，指定有多少列

二维数组的定义

类型名　数组名 [行数][列数];

- 类型名 - 定义数组元素的类型
- 数组名 - 定义数组的名称，符合标识符的语法
- 行数 - 整数类型的常量表达式，指定有多少行
- 列数 - 整数类型的常量表达式，指定有多少列

```
int a[3][2];    /* 3行2列的整数数组 */
```

二维数组的定义

类型名　数组名 [行数][列数];

- 类型名 - 定义数组元素的类型
- 数组名 - 定义数组的名称，符合标识符的语法
- 行数 - 整数类型的常量表达式，指定有多少行
- 列数 - 整数类型的常量表达式，指定有多少列

```
int a[3][2];    /* 3行2列的整数数组 */

int n = 5;
float f[n][n];    /* 非法定义，因为长度
                     n不是常量表达式 */
```

二维数组的定义

类型名　数组名 [行数][列数];

- 类型名 - 定义数组元素的类型
- 数组名 - 定义数组的名称，符合标识符的语法
- 行数 - 整数类型的常量表达式，指定有多少行
- 列数 - 整数类型的常量表达式，指定有多少列

```
int a[3][2];    /* 3行2列的整数数组 */

int n = 5;
float f[n][n];  /* 非法定义，因为长度
                   n不是常量表达式 */
```

数组的大小必须是常量表达式

二维数组的定义和初始化

类型名　数组名 [行数][列数] = {初值表};

逐行赋初值

```
int a[3][4] = {
    {1, 2, 3, 4 }, /* 第0行 初值*/
    {5, 6, 7, 8 }, /* 第1行 初值*/
    {9,10,11, 12}  /* 第2行 初值*/
};
```

二维数组的定义和初始化

类型名　数组名 [行数][列数] = {初值表};

逐行赋初值

```
int a[3][4] = {
    {1, 2, 3, 4 }, /* 第0行 初值*/
    {5, 6, 7, 8 }, /* 第1行 初值*/
    {9,10,11, 12}  /* 第2行 初值*/
};
```

部分赋初值

```
int a[3][4] = {
    {1, 2, 3 }, /* 第0行 初值，末尾无初值*/
    {}, /* 第1行无初值，默认为0*/
    {5, 6, 7, 8 }, /* 第2行 初值*/
};
```

二维数组的存放方式

- 二维数组在逻辑上具有行、列两个维度
- 但在内存中依然是线性顺序存放的
  - 计算机的内存只有一个地址值
- 按照地址值从低到高，依次存放数组的：

  第 0 行，第 1 行，第 2 行，$\cdots$

二维数组的存放方式

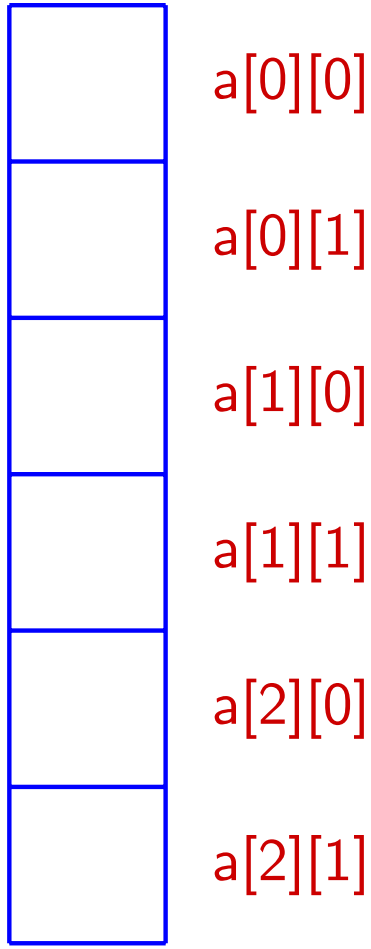
- 二维数组在逻辑上具有行、列两个维度
- 但在内存中依然是线性顺序存放的
  - 计算机的内存只有一个地址值
- 按照地址值从低到高，依次存放数组的：

  第 0 行，第 1 行，第 2 行，$\cdots$

假设定义了

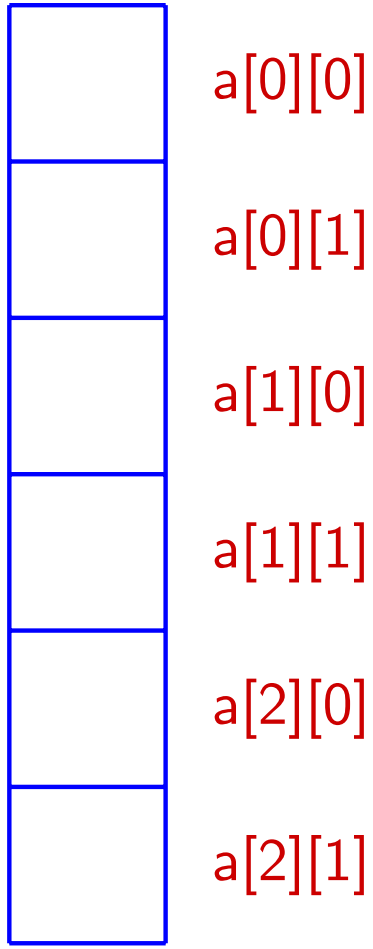
```
int a[3][2];
```

那么数组 a 在内存中的排列顺序如图所示意。$\Longrightarrow$



二维数组的存放方式

- 二维数组在逻辑上具有行、列两个维度
- 但在内存中依然是线性顺序存放的
  - 计算机的内存只有一个地址值
- 按照地址值从低到高，依次存放数组的：
  第 0 行，第 1 行，第 2 行，$\cdots$



假设定义了

```
int a[3][2];
```

那么数组 a 在内存中的排列顺序如图所示意。$\Longrightarrow$

定义二维数组的时候，可以采用顺序赋初值初始化

```
int a[3][2] = { 1, 2, 3, 4, 5, 6};
```

二维数组的使用，例程 7-8

定义一个二维数组 a

$$a[\,i\,][\,j\,] = i + j \quad (0 \le i \le 2,\ 0 \le j \le 2)$$

并且按矩阵的形式输出 a

```
#include <stdio.h>
int main( void )
{
    int i, j, a[3][2];
    /* 给二维数组赋值 */
    for( i = 0; i < 3; i++ )
        for( j = 0; j < 2; j++ )
            a[i][j] = i + j;
```

二维数组的使用，例程 7-8

定义一个二维数组 a

$$a[\,i\,][\,j\,] = i + j \quad (0 \le i \le 2,\ 0 \le j \le 2)$$

并且按矩阵的形式输出 a

```
    /* 按矩阵输出 */
    for( i = 0; i < 3; i++ ) {
        for( j = 0; j < 2; j++ )
            printf("%4d", a[i][j]);
        printf("\n"); /* 换行 */
    }
    return 0;
}
```

二维数组的使用，例程 7-9 矩阵转置

矩阵转置：将元素 a_{ij} 和 a_{ji} 互换

```
for( i = 0; i<n; i++ )
    for( j = 0; j<i; j++ ) {
        /* 转置：交换a[i][j]和a[j][i] */
        temp    = a[i][j];
        a[i][j] = a[j][i];
        a[j][i] = temp;
    }
```

二维数组的使用，例程 7-9 矩阵转置

矩阵转置：将元素 a_{ij} 和 a_{ji} 互换

```
for( i = 0; i<n; i++ )
    for( j = 0; j<i; j++ ) {
        /* 转置：交换a[i][j]和a[j][i] */
        temp    = a[i][j];
        a[i][j] = a[j][i];
        a[j][i] = temp;
    }
```

二维数组的使用，例程 7-9 矩阵转置

矩阵转置：将元素 a_{ij} 和 a_{ji} 互换

```
for( i = 0; i<n; i++ )
    for( j = 0; j<i; j++ ) {
        /* 转置：交换a[i][j]和a[j][i] */
        temp    = a[i][j];
        a[i][j] = a[j][i];
        a[j][i] = temp;
    }
```

注意：不要交换两次

二维数组的使用，例程 7-10 年月日计算

给定年月日，计算它是该年中的第几天。

```
#include <stdio.h>
int day_of_year(int year, int month, int day )
{
  int k, leap = year%4==0 && year%100!=0 ||
     year%400==0;
  int days[][13] = { /* 每个月的天数 */
    {0,31,28,31,30,31,30,31,31,30,31,30,31},
    {0,31,29,31,30,31,30,31,31,30,31,30,31}};

  /* 加上本月之前的每个月的天数 */
  for( k = 1; k<month; k++ )
    day = day + days[leap][k];

  return day;
}
```

二维数组的使用，例程 7-10 年月日计算

定义一个二维数组保存天数，简化程序设计

```
/* 加上本月之前的每个月的天数 */
for( k = 1; k<month; k++ )
   day = day + days[leap][k];
```

二维数组的使用，例程 7-10 年月日计算

定义一个二维数组保存天数，简化程序设计

```
/* 加上本月之前的每个月的天数 */
for( k = 1; k<month; k++ )
   day = day + days[leap][k];
```

如果不定义二维数组保存天数，怎么设计程序？

二维数组的使用，例程 7-10 年月日计算

定义一个二维数组保存天数，简化程序设计

```
/* 加上本月之前的每个月的天数 */
for( k = 1; k<month; k++ )
    day = day + days[leap][k];
```

如果不定义二维数组保存天数，怎么设计程序？

```
/* 加上本月之前的每个月的天数 */
for( k = 1; k<month; k++ ) {
    int dk;
    if( k>7 ) dk = 31 - k%2;
    else if( k == 2 ) dk = 28 + leap;
    else dk = 30 + k%2;
    day += dk;
}
```

内容提要

一维字符数组

元素类型为 char 的数组

- 适用普通数组的定义、初始化、引用

```
char a[10];
char h[20] = {'H', 'e', 'l', 'l', 'o', '\0'};
char str[] = {'c', 'h', 'a', 'r'};
```

读入一串字符，保存在数组中

```
int i, k;
char line[80];
for(k = 0; k<sizeof(line); k++ )
    if( (line[k]=getchar()) == '\n' )
        break;
```

字符串

字符串

一个有序的字符序列，称为字符串。

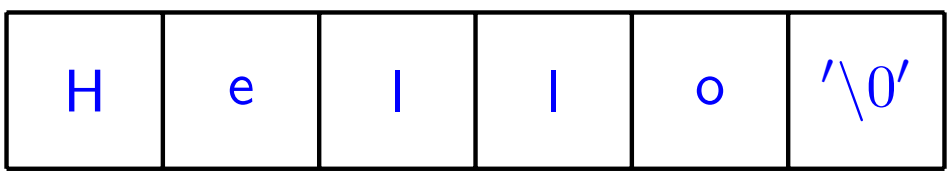
字符串常量

一对双引号括起来的字符序列。例如

"Happy", "Hello world", "Country"

字符串的存储

将字符序列按顺序保存在一块连续内存中，并以特殊字符$'\backslash 0'$结尾。例如字符串"Hello" 的存储为：



注意：不存储双引号

字符串与字符数组

用字符数组存储字符串

```
char s[10];
/* 将字符串 "Hello" 保存在s中 */
s[0] = 'H'; s[1] = 'e';
s[2] = 'l'; s[3] = 'l';
s[4] = 'o'; s[5] = '\0';
```

注意：必须以 '\0' 结尾

字符串与字符数组

用字符数组存储字符串

```
char s[10];
/* 将字符串 "Hello" 保存在s中 */
s[0] = 'H'; s[1] = 'e';
s[2] = 'l'; s[3] = 'l';
s[4] = 'o'; s[5] = '\0';
```

注意：必须以 '\0' 结尾

用字符串常量初始化字符数组

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

字符串与字符数组

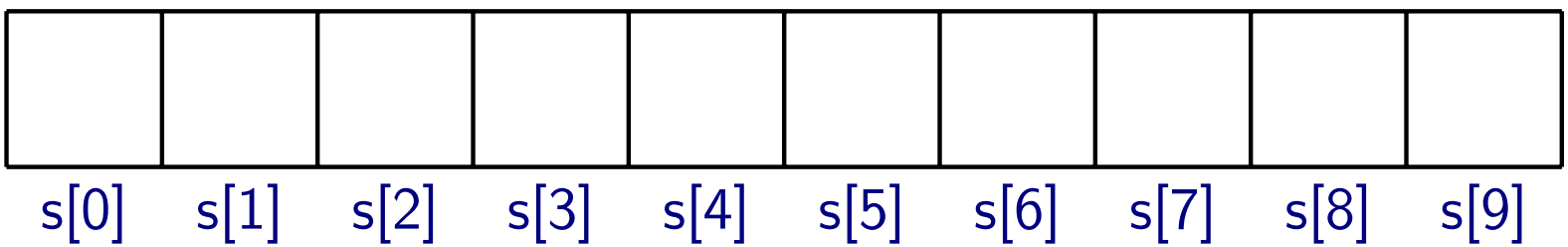
用字符数组存储字符串

```
char s[10];
/* 将字符串 "Hello" 保存在s中 */
s[0] = 'H'; s[1] = 'e';
s[2] = 'l'; s[3] = 'l';
s[4] = 'o'; s[5] = '\0';
```

注意：必须以 '\0' 结尾

用字符串常量初始化字符数组

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



字符串与字符数组

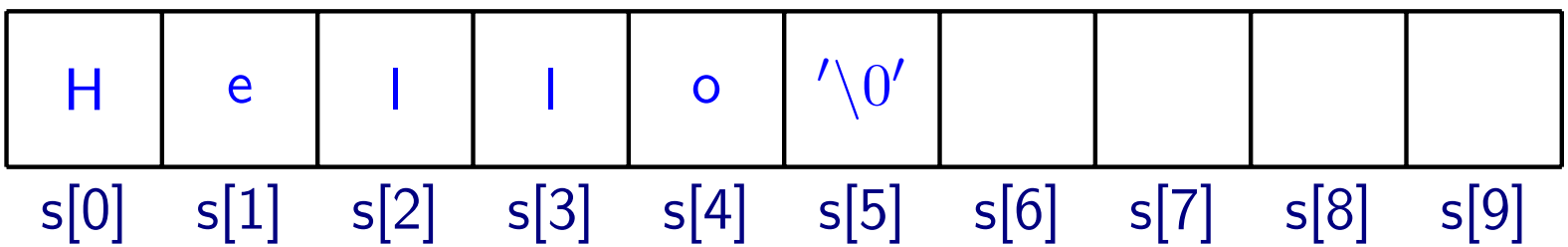
用字符数组存储字符串

```
char s[10];
/* 将字符串 "Hello" 保存在s中 */
s[0] = 'H'; s[1] = 'e';
s[2] = 'l'; s[3] = 'l';
s[4] = 'o'; s[5] = '\0';
```

注意：必须以 $'\backslash 0'$ 结尾

用字符串常量初始化字符数组

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



字符串与字符数组

省略长度的字符数组定义

```
char b[] = "Happy";
```

字符串与字符数组

省略长度的字符数组定义

```
char b[] = "Happy";
```

数组 b 长度为几？

- 等于初值表里初值的个数
- 初值表是字符串常量，包含：'H'，'e'，'l'，'l'，'o'，'\0'
- 共计 6 个元素。注意不要忘记'\0'

字符串与字符数组

如何将字符串存入字符数组

- 一是如前所述，用字符串常量初始化
- 或者将字符逐个存入数组，最后添加结束标志字符'\0'

字符串与字符数组

如何将字符串存入字符数组

- 一是如前所述，用字符串常量初始化
- 或者将字符逐个存入数组，最后添加结束标志字符'\0'

例如：将字符串"a"存入字符数组 s 中

```
char s[10];
s[0] = 'a';
s[1] = '\0';
```

字符串与字符数组

如何将字符串存入字符数组

- 一是如前所述，用字符串常量初始化
- 或者将字符逐个存入数组，最后添加结束标志字符'\0'

例如：将字符串"a"存入字符数组 s 中

```
char s[10];
s[0] = 'a';
s[1] = '\0';
```

注意：字符'a'和字符串"a"的区别

- "a"是字符串，包含了两个字符：'a'和'\0'

判断字符数组 line 中读入的内容是否为回文

读入一串字符，保存为字符串（在字符数组中）

```
#define MAXLINE 80 //定义符号常量（宏）

int main(void)
{
    int i, k;
    char line[MAXLINE];

    //输入字符串
    printf("Enter a string:");
    k = 0;
    while( (line[k]=getchar())!='\n' )
        k++;
    line[k] = '\0'; //别忘记\0
```

判断字符数组 line 中读入的内容是否为回文

比较判断字符串是否为回文

```
i= 0;
k = k-1;
while ( i<k ) {
    if( line[i]!=line[k] )
        break; //不是回文
    i++; //指向下一个字符
    k--; //指向前一个字符
}

if( i>=k ) // 检查循环终止情况
    printf("是的, 是回文\n");
else
    printf("不, 不是回文\n");
```

课后试试：改写循环

```
// 将while循环改为for循环
for( i = 0, k = k-1; i<k; i++, k-- )
    if( line[i]!=line[k] )
        break;
```

```
// 或简化while循环
i = 0;
k = k-1;
while( i<k && line[i]==line[k] )
    i++, k--;
```

```
// 或简化while循环
i = 0;
while( i<--k && line[i]==line[k] )
    i++;
```

字符串编程，统计数字字符的个数

输入以回车为结束标志的一串字符（<80 个），统计其中数字字符的个数

字符串编程，统计数字字符的个数

输入以回车为结束标志的一串字符（<80 个），统计其中数字字符的个数

定义变量

```
#include <stdio.h>
int main( void )
{
    int count, i;
    char str[81];
```

字符串编程，统计数字字符的个数

输入以回车为结束标志的一串字符（<80 个），统计其中数字字符的个数

定义变量

```
#include <stdio.h>
int main( void )
{
    int count, i;
    char str[81];
```

读入以回车为结束标志的一串字符

```
    printf("输入字符串:");
    i = 0;
    while( i<80 && (str[i]=getchar())!='\n' )
        i++;
    str[i] = '\0'; /* 添加字符串结束标志*/
```

字符串编程，统计数字字符的个数

统计数字字符的个数

```
    for( count=0, i=0; str[i]!='\0'; i++ )
        if( str[i]>='0' && str[i]<='9' )
            count ++;
    printf("Count = %d\n", count);
    return 0;
}
```

字符串处理的循环条件：

```
str[i] != '\0'
```

字符串编程，例 7-12，凯撒密码

为了防止信息泄漏，需要将电码明文通过加密的方法变换为密文。

字符串编程，例 7-12，凯撒密码

为了防止信息泄漏，需要将电码明文通过加密的方法变换为密文。

读入电码明文

```
#define MAXLINE 80
#define M 26 //定义符号常量

int main(void)
{
    int i, offset;
    char str[MAXLINE];

    printf("输入电码明文字符串: ");
    for( i=0; (str[i]=getchar())!='\n'; i++ )
        ;
    str[i] = '\0'; // 添加结束标志
```

读入加密偏移量

```
printf("输入加密偏移量: ");
scanf("%d", &offset);
offset = offset % M; //限制偏移量<M
```

逐个字母加密：加上偏移量

对大写字母加密

```
for( i = 0; str[i]!='\0'; i++ ) {
    if( str[i]>='A' && str[i]<='Z' ) {
        str[i] += offset;
        if( str[i]>'Z' ) str[i] -= M;
    }
```

对小写加密

```
    if( str[i]>='a' && str[i]<='z' ) {
        str[i] += offset;
        if( str[i]>'z' ) str[i] -= M;
    }
```

字符串编程，例 7-12，凯撒密码

输出加密后的字符串

```
printf("加密后的密文是：%s\n", str);
```

字符串输出方法

- 使用格式控制字符%s，调用 printf 函数
- 调用函数 puts，例如 puts(str)；该函数会自动换行
- 循环调用 putchar，逐个输出字符

```
for( i=0; str[i]!='\0'; i++ )
    putchar(str[i]);
```

字符串编程，例程 7-13，转换为 10 进制数

- 输入以回车为结束标志的一串字符（$<$10 个）
- 将其中的数字字符转换为 10 进制输出

字符串编程，例程 7-13，转换为 10 进制数

- 输入以回车为结束标志的一串字符（<10 个）
- 将其中的数字字符转换为 10 进制输出

读入字符串的部分与前面的程序雷同，略

定义变量 number 保存十进制的结果

```
int number, i;
```

将数字字符转换为 10 进制的数

```
for( number=0, i=0; str[i]!='\0'; i++ )
    if( str[i]>='0' && str[i]<='9' )
        number = number * 10 + str[i]-'0';
printf("Digit=%d\n", number);
```

字符串编程，例程 7-13，转换为 10 进制数

- 输入以回车为结束标志的一串字符（<10 个）
- 将其中的数字字符转换为 10 进制输出

读入字符串的部分与前面的程序雷同，略

定义变量 number 保存十进制的结果

```
int number, i;
```

将数字字符转换为 10 进制的数

```
for( number=0, i=0; str[i]!='\0'; i++ )
    if( str[i]>='0' && str[i]<='9' )
        number = number * 10 + str[i]-'0';
printf("Digit=%d\n", number);
```

字符串处理的循环条件：不是结束字符'\0'

```
str[i] != '\0'
```

字符串编程，16 进制转换为 10 进制，例程 7-14

- 输入一串字符，以 # 为结束标志
- 输出其中的 16 进制字符，和对应的 10 进制数值

字符串编程，16 进制转换为 10 进制，例程 7-14

- 输入一串字符，以 # 为结束标志
- 输出其中的 16 进制字符，和对应的 10 进制数值

读入部分省略。复制 16 进制的字符到新的数组 hexad 中

```
for( k = i = 0; str[i]!='\0'; i++ )
    if( str[i]>='0'&&str[i]<='9' ||
        str[i]>='a'&&str[i]<='f' ||
        str[i]>='A'&&str[i]<='F' )
        hexad[k++] = str[i];
hexad[k] = '\0'; /* 添加字符串结束标志*/
```

字符串编程，16 进制转换为 10 进制，例程 7-14

- 输入一串字符，以 # 为结束标志
- 输出其中的 16 进制字符，和对应的 10 进制数值

读入部分省略。复制 16 进制的字符到新的数组 hexad 中

```
for( k = i = 0; str[i]!='\0'; i++ )
    if( str[i]>='0'&&str[i]<='9' ||
        str[i]>='a'&&str[i]<='f' ||
        str[i]>='A'&&str[i]<='F' )
        hexad[k++] = str[i];
hexad[k] = '\0'; /* 添加字符串结束标志*/
```

复制语句带有后缀运算符 ++

执行顺序是什么？

字符串编程，16 进制转换为 10 进制，例程 7-14

- 输入一串字符，以 # 为结束标志
- 输出其中的 16 进制字符，和对应的 10 进制数值

读入部分省略。复制 16 进制的字符到新的数组 hexad 中

```
for( k = i = 0; str[i]!='\0'; i++ )
    if( str[i]>='0'&&str[i]<='9' ||
        str[i]>='a'&&str[i]<='f' ||
        str[i]>='A'&&str[i]<='F' )
        hexad[k++] = str[i];
hexad[k] = '\0'; /* 添加字符串结束标志*/
```

复制语句带有后缀运算符 ++

执行顺序是什么？

先忽略 ++，计算完表达式之后，再对变量执行 ++ 运算

字符串编程，16 进制转换为 10 进制

```
hexad[k++] = str[i];
```

- ++ 是单目运算符，右结合，高优先级（但是比 [ ] 低）
- 执行的时候，++ 和 −− 作为后缀运算时都是在最后执行

  总结：优先结合，滞后执行

因此

```
hexad[k++] = str[i];
```

等价于以下两个语句：

```
hexad[k] = str[i];
k++;
```

注意：前缀运算则相反

```
hexad[++k] = str[i];
```

等价于：

```
++k;
hexad[k] = str[i];
```

字符串编程，16 进制转换为 10 进制

输出将 16 进制字符串 hexad

```
printf("The Hexa string is:%s\n", hexad);
```

字符串的输入和输出

- printf 用%s作为格式，输出字符串。
  - 逐个输出字符，直到结束字符'\0' 为止
  - 要求被输出的参数是以'\0' 结尾的字符序列，即字符串
- scanf 用%s作为格式，读入字符串
  - 读入字符串的时候，碰到空格、回车、制表符的时候停止
  - 并在字符串的末尾添加字符串结束字符'\0'

字符串编程，16 进制转换为 10 进制，例程 7-14

将字符串 hexad 转换为 10 进制的数值

```
for( number=0, i=0; hexad[i]!='\0'; i++ )
    if( str[i]>='0' && str[i]<='9' )
        number = number * 16 + str[i]-'0';
    else if( str[i]>='a' && str[i]<='f' )
        number = number * 16 + str[i]-'a' + 10;
    else if( str[i]>='A' && str[i]<='F' )
        number = number * 16 + str[i]-'A' + 10;
/* 输出数值 */
printf("Number = %ld\n", number);
```

字符串编程，16 进制转换为 10 进制，例程 7-14

将字符串 hexad 转换为 10 进制的数值

```
for( number=0, i=0; hexad[i]!='\0'; i++ )
    if( str[i]>='0' && str[i]<='9' )
        number = number * 16 + str[i]-'0';
    else if( str[i]>='a' && str[i]<='f' )
        number = number * 16 + str[i]-'a' + 10;
    else if( str[i]>='A' && str[i]<='F' )
        number = number * 16 + str[i]-'A' + 10;
/* 输出数值 */
printf("Number = %ld\n", number);
```

字符串编程，16 进制转换为 10 进制，例程 7-14

将字符串 hexad 转换为 10 进制的数值

```
for( number=0, i=0; hexad[i]!='\0'; i++ )
    if( str[i]>='0' && str[i]<='9' )
        number = number * 16 + str[i]-'0';
    else if( str[i]>='a' && str[i]<='f' )
        number = number * 16 + str[i]-'a' + 10;
    else if( str[i]>='A' && str[i]<='F' )
        number = number * 16 + str[i]-'A' + 10;
/* 输出数值 */
printf("Number = %ld\n", number);
```

字符串编程，16 进制转换为 10 进制，例程 7-14

代码改造

```
for( number=0, i=0; hexad[i]!='\0'; i++ ) {
    int d = str[i];
    if( d>='0' && d<='9' )
        d = d - '0';
    else if( d>='a' && d<='f' )
        d = d - 'a' + 10;
    else //if( d>='A' && d<='F' )
        d = d - 'A' + 10;
    number = number * 16 + d;
}
```

字符串编程，16 进制转换为 10 进制，例程 7-14

代码改造

```
for( number=0, i=0; hexad[i]!='\0'; i++ ) {
    int d = str[i];
    if( d>='0' && d<='9' )
        d = d - '0';
    else if( d>='a' && d<='f' )
        d = d - 'a' + 10;
    else //if( d>='A' && d<='F' )
        d = d - 'A' + 10;
    number = number * 16 + d;
}
```

字符串编程，16 进制转换为 10 进制，例程 7-14

代码改造

```
for( number=0, i=0; hexad[i]!='\0'; i++ ) {
    int d = str[i];
    if( d>='0' && d<='9' )
        d = d - '0';
    else if( d>='a' && d<='f' )
        d = d - 'a' + 10;
    else //if( d>='A' && d<='F' )
        d = d - 'A' + 10;
    number = number * 16 + d;
}
```

字符串编程，16 进制转换为 10 进制，例程 7-14

代码改造

```
for( number=0, i=0; hexad[i]!='\0'; i++ ) {
    int d = str[i];
    if( d>='0' && d<='9' )
        d = d - '0';
    else if( d>='a' && d<='f' )
        d = d - 'a' + 10;
    else //if( d>='A' && d<='F' )
        d = d - 'A' + 10;
    number = number * 16 + d;
}
```

总结

- 数组的定义和使用
  - 数组元素的遍历，下标范围
  - 元素查找
  - 选择排序法
- 二维数组
- 字符数组
- 字符串使用和处理

今天到此为止