第六章　数组 指针与字符串

（一）

数组的定义与使用

- 数组是具有一定顺序关系的若干相同类型变量的集合体，组成数组的变量称为该数组的元素。

数组的定义

类型说明符　数组名[常量表达式] [常量表达式]..... ；

数组名的构成方法与一般变量名相同。

- 例如：int a[10];

表示a为整型数组，有10个元素：a[0]...a[9]

- 例如: int a[5][3];

表示a为整型二维数组，其中第一维有5个下标（0~4），第二维有3个下标（0~2），数组的元素个数为15，可以用于存放5行3列的整型数据表格。

数组的使用

- 使用数组元素

  必须先声明，后使用。

  只能逐个引用数组元素，而不能一次引用整个数组

  例如：a[0]=a[5]+a[7]-a[2*3]

  例如：b[1][2]=a[2][3]/2

例 6-1

```
#include <iostream>
using namespace std;
int main() {
  int a[10], b[10];
  for(int i = 0; i < 10; i++) {
    a[i] = i * 2 - 1;
    b[10 - i - 1] = a[i];
```

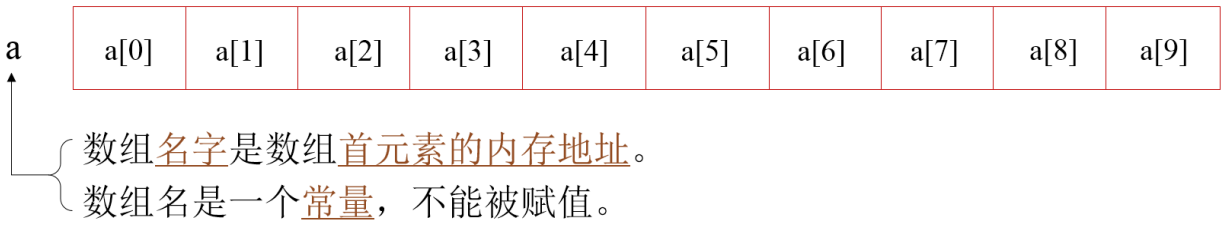
```
  }
  for(int i = 0; i < 10; i++) {
    cout << "a[" << i << "] = " << a[i] << "  ";
    cout << "b[" << I << "] = " << b[i] << endl;
  }
  return 0;
}
```

数组的存储与初始化

一维数组的存储

数组元素在内存中顺次存放，它们的地址是连续的。元素间物理地址上的相邻，对应着逻辑次序上的相邻。

例如：



数组名字是数组首元素的内存地址。

数组名是一个常量，不能被赋值。

一维数组的初始化

在定义数组时给出数组元素的初始值。

- 列出全部元素的初始值

  例如：static int a[10]={0,1,2,3,4,5,6,7,8,9};

- 可以只给一部分元素赋初值

  例如：static int a[10]={0,1,2,3,4};

- 在对全部数组元素赋初值时，可以不指定数组长度

  例如：static int a[]={0,1,2,3,4,5,6,7,8,9}

二维数组的存储

- 按行存放

  例如： float a[3][4];

  可以理解为：

a
- a[0]——a_{00} a_{01} a_{02} a_{03}
- a[1]——a_{10} a_{11} a_{12} a_{13}
- a[2]——a_{20} a_{21} a_{22} a_{23}

其中数组a的存储顺序为：

a_{00} a_{01} a_{02} a_{03} a_{10} a_{11} a_{12} a_{13} a_{20} a_{21} a_{22} a_{23}

二维数组的初始化

- 将所有初值写在一个{}内，按顺序初始化
  - 例如：static int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12
- 分行列出二维数组元素的初值
  - 例如：static int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
- 可以只对部分元素初始化
  - 例如：static int a[3][4]={{1},{0,6},{0,0,11}};
- 列出全部初始值时，第1维下标个数可以省略
  - 例如：static int a[][4]={1,2,3,4,5,6,7,8,9,10,11,12};
  - 或：static int a[][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};

注意：

- 如果不作任何初始化，内部auto型数组中会存在垃圾数据，static数组中的数据默认初始化为0；
- 如果只对部分元素初始化，剩下的未显式初始化的元素，将自动被初始化为零；
- 现在我们来看一个用数组存放Fibonacci数列的例子。

例: 求 Fibonacci 数列的前 20 项

```
#include <iostream>
using namespace std;
int main() {
  int f[20] = {1,1}; //初始化第0、1个数
  for (int i = 2; i < 20; i++) //求第2～19个数
    f[i] = f[i - 2] + f[i - 1];
  for (i=0;i<20;i++) {      //输出，每行5个数
    if (i % 5 == 0) cout << endl;
      cout.width(12);       //设置输出宽度为12
      cout << f[i];
  }
  return 0;
```

```
}
```

运行结果：

```
   1          1          2          3          5
   8         13         21         34         55
  89        144        233        377        610
 987       1597       2584       4181       6765
```

一维数组应用举例

循环从键盘读入若干组选择题答案，计算并输出每组答案的正确率，直到输入ctrl+z为止。每组连续输入5个答案，每个答案可以是'a'..'d'。

```
#include <iostream>
using namespace std;
int main() {
  const char key[ ] = {'a','c','b','a','d'};
  const int NUM_QUES = 5;
  char c;
  int ques = 0, numCorrect = 0;
  cout << "Enter the " << NUM_QUES << " question tests:" << endl;
  while(cin.get(c)) {
    if(c != '\n') {
        if(c == key[ques]) {
        numCorrect++; cout << " ";
      } else
        cout<<"*";
      ques++;
    } else {
      cout << " Score " << static_cast<float>(numCorrect)/NUM_QUES*100 <<
      "%";
      ques = 0;  numCorrect = 0; cout << endl;
    }
  }
  return 0;
}
```

数组作为函数参数

- 数组元素作实参，与单个变量一样。
- 数组名作参数，形、实参数都应是数组名（实质上是地址，关于地址详见6.2），类型要一样，传送的是数组首地址。对形参数组的改变会直接影响到实参数组。

例 6-2 使用数组名作为函数参数

主函数中初始化一个二维数组，表示一个矩阵，矩阵，并将每个元素都输出，然后调用子函数，分别计算每一行的元素之和，将和直接存放在每行的第一个元素中，返回主函数之后输出各行元素的和。

```
#include <iostream>
using namespace std;
void rowSum(int a[][4], int nRow) {
    for (int i = 0; i < nRow; i++) {
      for(int j = 1; j < 4; j++)
            a[i][0] += a[i][j];
    }
}
int main() {//主函数
    //定义并初始化数组
  int table[3][4] = {{1, 2, 3, 4}, {2, 3, 4, 5}, {3, 4, 5, 6}};


 //输出数组元素
    for (int i = 0; i < 3; i++)    {
      for (int j = 0; j < 4; j++)
            cout << table[i][j] << "   ";
      cout << endl;
    }
    rowSum(table, 3);    //调用子函数，计算各行和
  //输出计算结果
    for (int i = 0; i < 3; i++)
```

```
        cout << "Sum of row " << i << " is " << table[i][0] << endl;
    return 0;
}
```

对象数组

对象数组的定义与访问

- 定义对象数组

  类名　数组名[元素个数]；

- 访问对象数组元素

  通过下标访问

  数组名[下标].成员名

对象数组初始化

- 数组中每一个元素对象被创建时，系统都会调用类构造函数初始化该对象。
- 通过初始化列表赋值。

例：Point a[2]={Point(1,2),Point(3,4)};

- 如果没有为数组元素指定显式初始值，数组元素便使用默认值初始化（调用默认构造函数）。

数组元素所属类的构造函数

- 元素所属的类不声明构造函数，则采用默认构造函数。
- 各元素对象的初值要求为相同的值时，可以声明具有默认形参值的构造函数。
- 各元素对象的初值要求为不同的值时，需要声明带形参的构造函数。
- 当数组中每一个对象被删除时，系统都要调用一次析构函数。

例 6-3 对象数组应用举例

```
//Point.h
#ifndef _POINT_H
#define _POINT_H
class Point {       //类的定义
public:             //外部接口
    Point();
```

```
    Point(int x, int y);
    ~Point();
    void move(int newX,int newY);
    int getX() const { return x; }
    int getY() const { return y; }
    static void showCount();   //静态函数成员
private:                       //私有数据成员
    int x, y;
};
#endif      //_POINT_H


//Point.cpp
#include <iostream>
#include "Point.h"
using namespace std;
Point::Point() : x(0), y(0) {
    cout << "Default Constructor called." << endl;
}
Point::Point(int x, int y) : x(x), y(y) {
    cout << "Constructor called." << endl;
}
Point::~Point() {
    cout << "Destructor called." << endl;
}
void Point::move(int newX,int newY) {
    cout << "Moving the point to (" << newX << ", " << newY << ")" << endl;
    x = newX;
    y = newY;
}

//6-3.cpp
#include "Point.h"
```

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

int main() {
    cout << "Entering main..." << endl;
    Point a[2];
    for(int i = 0; i < 2; i++)
      a[i].move(i + 10, i + 20);
    cout << "Exiting main..." << endl;
    return 0;
}
```

基于范围的 for 循环

```
int main()
{
int array[3] = {1,2,3};
int *p;
for(p = array; p < array + sizeof(array) / sizeof(int); ++p)
{
   *p += 2;
   std::cout << *p << std::endl;
}

return 0;
}
```

```
int main()
{
int array[3] = {1,2,3};
for(int & e : array)
{
   e += 2;
   std::cout<<e<<std::endl;
}

return 0;
}
```

指针的概念、定义和指针运算

内存空间的访问方式

- 通过变量名访问
- 通过地址访问

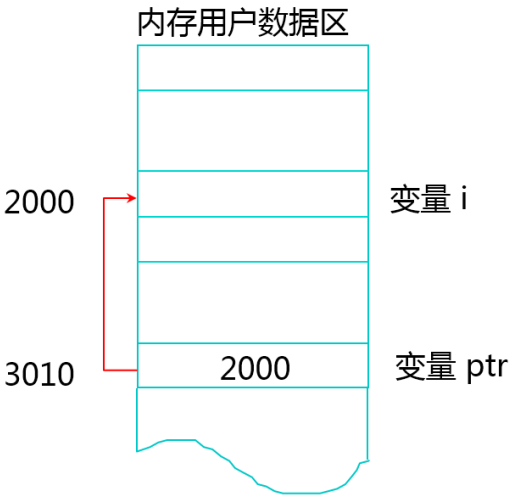
指针的概念

- 指针：内存地址，用于间接访问内存单元
- 指针变量：用于存放地址的变量

指针变量的定义

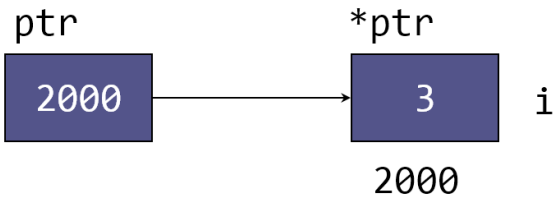
- 例：

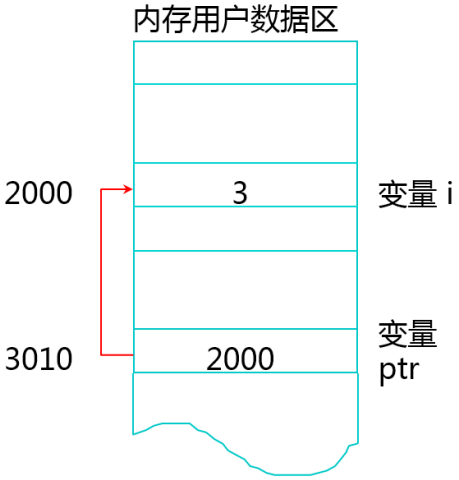
```
static int i;
static int*  ptr = &i;
```



- 例：

```
*ptr = 3;
```





与地址相关的运算——“*”和“&”

- 指针运算符
- 地址运算符：&

指针的初始化和赋值

指针变量的初始化

- 语法形式

  存储类型 数据类型 *指针名 = 初始地址；

- 例：

int *pa = &a;

- 注意事项
  - 用变量地址作为初值时，该变量必须在指针初始化之前已声明过，且变量类型应与指针类型一致。
  - 可以用一个已有合法值的指针去初始化另一个指针变量。
  - 不要用一个内部非静态变量去初始化 static 指针。

指针变量的赋值运算

- 语法形式

指针名=地址

注意："地址" 中存放的数据类型与指针类型必须相符

- 向指针变量赋的值必须是地址常量或变量，不能是普通整数，例如：
  - 通过地址运算 "&" 求得已定义的变量和对象的起始地址
  - 动态内存分配成功时返回的地址
- 例外：整数0可以赋给指针，表示空指针。
- 允许定义或声明指向 void 类型的指针。该指针可以被赋予任何类型对象的地址。

  例： void *general;

指针空值 nullptr

- 以往用0或者NULL去表达空指针的问题：
  - C/C++的NULL宏是个被有很多潜在BUG的宏。因为有的库把其定义成整数0，有的定义成 (void*)0。在C的时代还好。但是在C++的时代，这就会引发很多问题。
- C++11使用nullptr关键字，是表达更准确，类型安全的空指针

例 6-5 指针的定义、赋值与使用

```
//6_5.cpp
#include <iostream>
```

```
using namespace std;
int main() {
    int i;                      //定义int型数i
    int *ptr = &i;  //取i的地址赋给ptr
    i = 10;                     //int型数赋初值
    cout << "i = " << i << endl;              //输出int型数的值
    cout << "*ptr = " << *ptr << endl;      //输出int型指针所指地址的内容
    return 0;
}
```

运行结果：

```
i = 10
*ptr = 10
```

例6-6 void类型指针的使用

```
#include <iostream>
using namespace std;
int main() {
    //!void voidObject;  错，不能声明void类型的变量
    void *pv;               //对，可以声明void类型的指针
    int i = 5;
    pv = &i;                //void类型指针指向整型变量
    int *pint = static_cast<int *>(pv);  //void指针转换为int指针
    cout << "*pint = " << *pint << endl;
    return 0;
}
```

指向常量的指针

- 不能通过指向常量的指针改变所指对象的值，但指针本身可以改变，可以指向另外的对象。
- 例

```
int a;
const int *p1 = &a;         //p1是指向常量的指针
int b;
p1 = &b;     //正确，p1本身的值可以改变
```

```
*p1 = 1;      //编译时出错，不能通过p1改变所指的对象
```

指针类型的常量

- 若声明指针常量，则指针本身的值不能被改变。
- 例

```
int a;
int * const p2 = &a;
p2 = &b;        //错误，p2是指针常量，值不能改变
```

指针的算术运算、关系运算

指针类型的算术运算

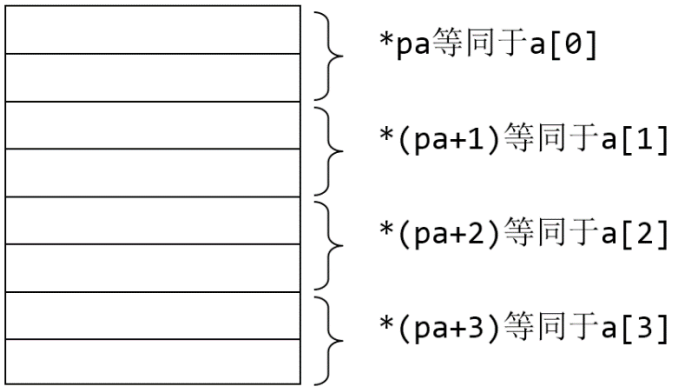
- 指针与整数的加减运算
- 指针++，--运算

指针类型的算术运算

- 指针p加上或减去n
  - 其意义是指针当前指向位置的前方或后方第n个数据的起始位置。
- 指针的++、--运算
  - 意义是指向下一个或前一个完整数据的起始。
- 运算的结果值取决于指针指向的数据类型，总是指向一个完整数据的起始位置。
- 当指针指向连续存储的同类型数据时，指针与整数的加减运和自增自减算才有意义。

指针与整数相加的意义

```
short a[4];
short *pa=a
```



指针类型的关系运算

- 指向相同类型数据的指针之间可以进行各种关系运算。

- 指向不同数据类型的指针，以及指针与一般整数变量之间的关系运算是无意义的。
- 指针可以和零之间进行等于或不等于的关系运算。

例如：p==0或p!=0