6、函数

6.1 概述

作用：将一段经常使用的代码封装起来，减少重复代码

一个较大的程序，一般分为若干个程序块，每个模块实现特定的功能。

6.2 函数的定义

函数的定义一般主要有5个步骤：

1、返回值类型

2、函数名

3、参数表列

4、函数体语句

5、return 表达式

语法：

```
返回值类型  函数名  （参数列表）
{

        函数体语句

        return表达式

}
```

- 返回值类型： 一个函数可以返回一个值。在函数定义中
- 函数名：给函数起个名称
- 参数列表：使用该函数时，传入的数据
- 函数体语句：花括号内的代码，函数内需要执行的语句
- return表达式： 和返回值类型挂钩，函数执行完后，返回相应的数据

示例：定义一个加法函数，实现两个数相加

1、返回值类型 int

2、函数的名称 add

3、参数列表 int num1,intnum2

4、函数体语句 int sum = num1+num2

5、return 表达式 return sum

```
//函数定义
int add(int num1, int num2)
{
    int sum = num1 + num2;
    return sum;
}
```

6.3 函数的调用

功能：使用定义好的函数

语法：`函数名（参数）`

示例：

```
#include<iostream>
#include<ctime> //time系统时间头文件
#include<string>
using namespace std;
//函数的定义
// 语法
// 返回值类型，函数名 参数列表 具体的函数体语句 return表达式
// 加法函数，实现两个整数相加，并且将相加的结果进行返回

// 函数定义的时候，num1和num2并没有真的数据，他只是一个形式上的参数，简称形参
int add(int num1,int num2)
{
    int sum = num1 + num2;
    return sum;
}

int main()
{
    int num1 = 1;
    int num2 = 2;
    // 调用函数
    // 函数调用语法： 函数名称(参数)
    // num1和num2成为实际参数，简称实参；在函数中并称之为形参
    // 当调用函数的时候，实参的值会传递给形参
    int sum = add(num1, num2);
    cout << "sum=" << sum << endl;

    system("pause");
    return 0;
}
```

总结：函数定义里小括号内称为形参，函数调用时传入的参数称为实参

6.4 值传递

- 所谓值传递，就是函数调用时实参将数值传入给形参
- 值传递时，如果形参发生，并不会影响实参

示例：

```
#include<iostream>
#include<ctime> //time系统时间头文件
#include<string>
using namespace std;

//值传递
// 定义函数，两个数字进行交换函数
void swap(int num1, int num2)
{

    cout << "交换前： " << endl;
    cout << "num1=" << num1 << endl;
    cout << "num2=" << num2 << endl;

    int temp = num1;
    num1 = num2;
    num2 = temp;

    cout << "交换前： " << endl;
    cout << "num1=" << num1 << endl;
    cout << "num2=" << num2 << endl;

    return;//或者都不需要写，或者返回值不需要的时候，可以不写return
}

int main()
{

    int a = 10;
    int b = 20;
    cout << "a=" << a << endl;
    cout << "b=" << b << endl;
    // 当我们做值传递的时候，函数的形参发生改变，并不会影响实参
    swap(a, b);

    cout << "a=" << a << endl;
    cout << "b=" << b << endl;
    system("pause");
    return 0;
}
```

总结： 值传递时，形参是修饰不了实参的

6.5 函数的常见样式

常见的函数样式有4种

1. 无参无返

2. 有参无返
3. 无参有返
4. 有参有返

示例：

```cpp
#include<iostream>
#include<ctime> //time系统时间头文件
#include<string>
using namespace std;

//函数常见样式

//1、无参无返
void test01()
{
    cout << "this is test01" << endl;

}
//2、有参无饭
void test02(int a)
{
    cout << "this is test02 a=" << a << endl;
    return;
}
//3、无参有返
int test03()
{
    cout << "this is test03" << endl;
    return 100;

}

//4、有参有返
int test04(int a)
{
    cout << "this is test04 a=" << a << endl;
    return a;
}

int main()
{
     // 无参无返函数调用
    test01();
    // 有参无返函数调用
    test02(100);
    // 无参有返函数调用
    int num1 = test03();
    cout << "num1=" << num1 << endl;
    // 有参有返函数调用
    int num2 = test04(1000);
    cout << "num2=" << num2 << endl;
    system("pause");
```

```
    return 0;
}
```

6.6 函数的声明

作用： 告诉编译器函数名称及如何调用函数。函数的实际主体可以单独定义。

- 函数的**声明可以多次**，但是函数的**定义只能有一次**

示例：

```
#include<iostream>
#include<ctime> //time系统时间头文件
#include<string>
using namespace std;

// 提前告诉编译器函数的存在，可以利用函数的声明
int max(int a, int b); // 函数声明


// 声明可以多次，定义只能一次
// 函数声明
// 比较函数，实现两个整型数字进行比较，返回较大的值
int max(int a, int b)
{
    return a > b ? a : b; // 三目运算符

}

int main()
{
    int a = 10;
    int b = 20;
    cout << max(a, b) << endl;
    system("pause");
    return 0;
}
```

6.7 函数的分文件编写

作用：让代码结构更加清晰

函数分文件编写一般有4个步骤

1. 创建后缀名为.h的头文件
2. 创建后缀名为.cpp的源文件
3. 在头文件中写函数的声明
4. 在源文件中写函数的定义

示例：

```
//swap.h文件
#include<iostream>
using namespace std;

// 函数的声明
void swap(int a, int b);

```

```
//swap.cpp文件
#include "swap.h" //自定义文件

//函数的定义
void swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
    cout << "a=" << a << endl;
    cout << "b=" << b << endl;

}
```

```
//main函数文件
#include<iostream>
using namespace std;

#include "swap.h"

// 1、创建.h后缀名的头文件 swap.h
// 2、创建.cpp后缀名的源文件 swap.cpp
// 3、在头文件中写函数的声明
// 4、在源文件中先函数的定义

int main()
{
    int a = 10;
    int b = 20;
    swap(a, b);

    system("pause");
    return 0;
}
```

7、指针

<<<<<<< HEAD

7.1 指针的基本概念

指针的作用： 可以通过指针**间接**访问内存

- 内存编号是从0开始记录的，一般用十六进制数字表示
- 可以利用指针变量保存地址

说白了，指针就是一个地址

7.2 指针变量的定义和使用

指针变量定义语法：`数据类型 * 变量名;`

示例：

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

int main()
{
    // 1、定义指针
    int a = 10;
    // 指针定义的语法：  数据类型 * 指针变量名
    int *p;
    // 让指针记录变量a的地址
    p = &a; // &取址变量符
    cout << "a的地址为：" << &a << endl; // 打印的是a的地址
    cout << "指针p等于：" << p << endl; // 打印的是a的地址

    // 2、使用指针
    // 可以通过解引用的方式来找到指针指向的内存
    // 指针前面加 * ，找到指针指向的内存中的数据
    *p = 100;
    cout << "a=" << a << endl;
    cout << "*p=" << *p << endl;

    system("pause");
    return 0;
}
```

指针变量和普通变量的区别

- 普通变量存放的是数据,指针变量存放的是地址
- 指针变量可以通过" * "操作符，操作指针变量指向的内存空间，这个过程称为解引用

> 总结1： 我们可以通过 & 符号 获取变量的地址

> 总结2：利用指针可以记录地址

> 总结3：对指针变量解引用，可以操作指针指向的内存

7.3 指针所占内存空间

提问：指针也是种数据类型，那么这种数据类型占用多少内存空间？

示例：

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

int main()
{
    //指针所占内存空间
    int a = 10;
    //int *p;
    //p = &a; // 指针p指向a的首地址
    int *p = &a;
    //在32位操作系统下，指针是占4个字节空间大小，不管是什么数据类型
    //在64位操作系统下，指针是占8个字节空间大小，不管是什么数据类型
    cout << "sizeof (int *) = " << sizeof(int *) << endl; // sizeof(p)
    cout << "sizeof (float *) = " << sizeof(float *) << endl; // sizeof(p)
    cout << "sizeof (double *) = " << sizeof(double *) << endl; // sizeof(p)
    cout << "sizeof (char *) = " << sizeof(char *) << endl; // sizeof(p)

    system("pause");
    return 0;
}
```

> 总结：所有指针类型在32位操作系统下是4个字节，64位下占8个字节

7.4 空指针和野指针

空指针：指针变量指向内存中编号为0的空间

用途：初始化指针变量

注意：空指针指向的内存是不可以访问的

示例1：空指针

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

int main()
{
     //空指针
    //1、空指针用于给指针变量进行初始化
    int *p = NULL;

    //2、空指针是不可以进行访问的
    //0~255之间的内存编号是系统占用的，因此不可以访问
    //*p = 100;
    system("pause");
    return 0;
}
```

野指针：指针变量指向非法的内存空间

示例2：野指针

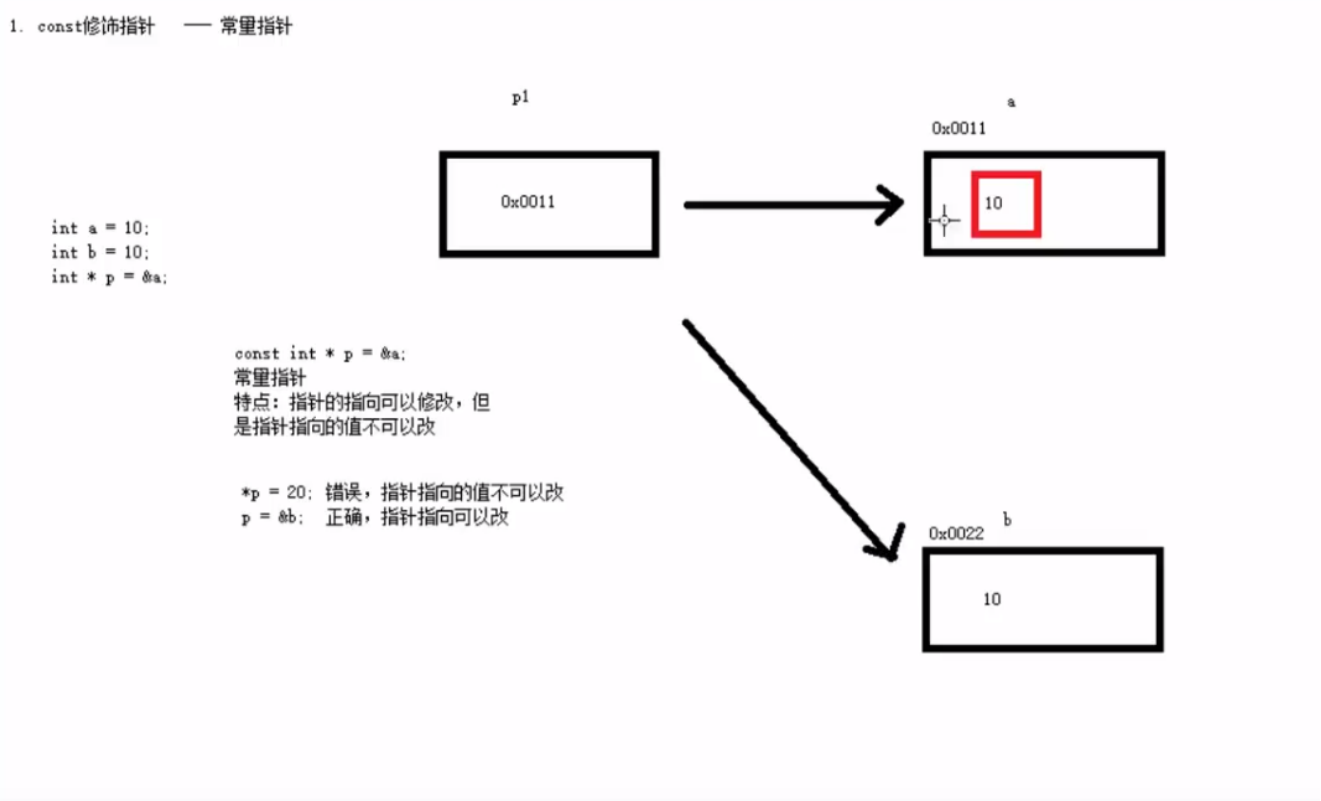
```
#include<iostream>
using namespace std;

int main()
{
    //野指针
    // 在程序中，尽量避免出现野指针
    //指针变量p指向内存地址编号为0x1100的空间
    int *p = NULL; //空指针
    int *p1 = (int *)0x1100;

    system("pause");
    return 0;
}
```

> 总结：空指针和野指针都不是我们申请的空间，因此不要访问。

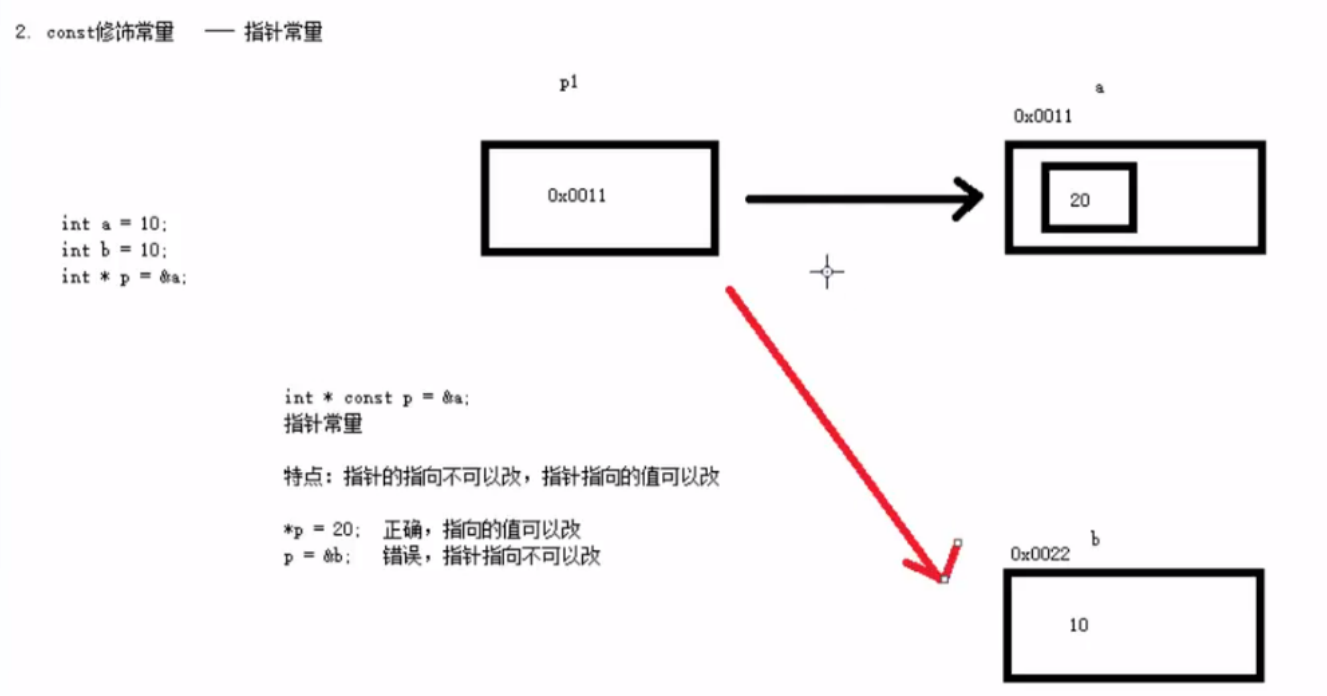
7.5 const修饰指针

const修饰指针有三种情况

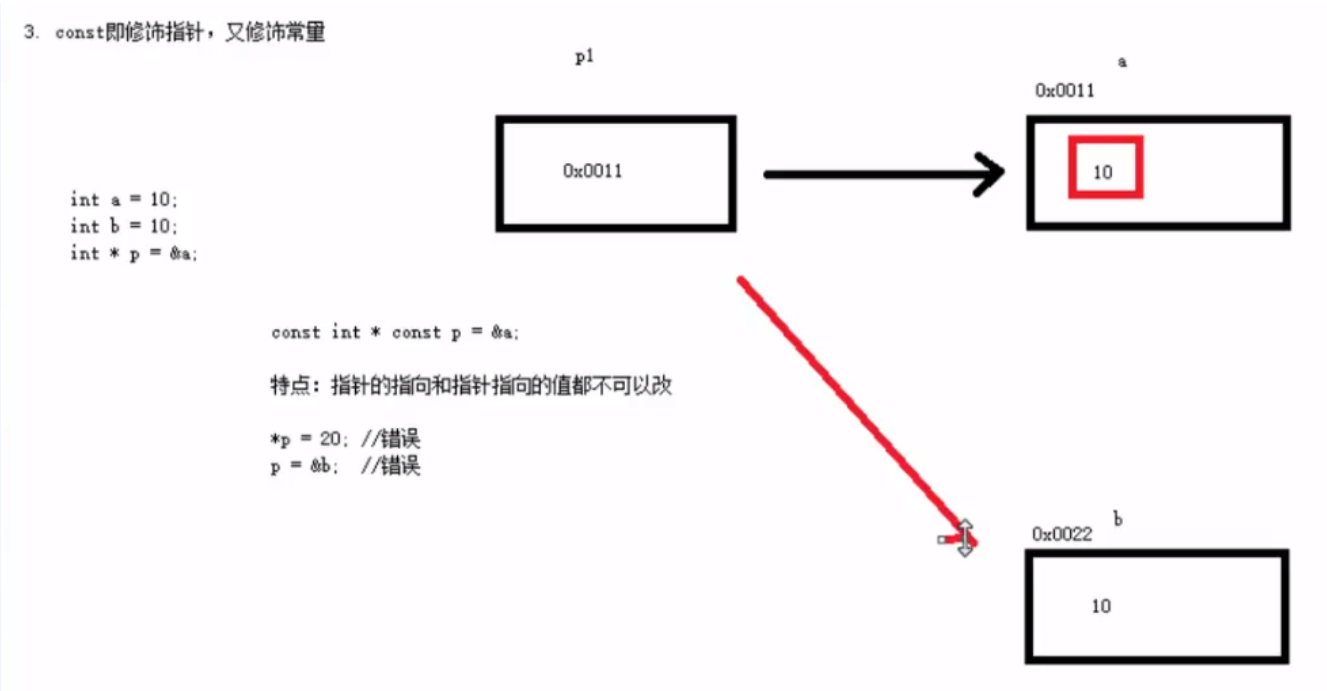
1. **const修饰指针 --- 常量指针**



2. **const修饰常量 --- 指针常量**



3. **const即修饰指针，又修饰常量**



示例：

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

int main()
{

    //1、const修饰指针
    int a = 10;
    int b = 10;

```

```
    const int * p = &a; //常量指针，指针指向的值不可以改，指针指向可以改
    //*p = 20; 错误
    p = &b;//正确

    //2、const修饰常量     指针常量
    // 指针的指向不可以改，指针指向的值可以改
    int * const p2 = &a; //
    *p2 = 100; //正确
    //p2 = &b;//错误，指针的指向不可以改

    //3、const修饰常量和指针
    const int * const p3 = &a;
    //*p3 = 100; 错误
    //p3 = &b; 错误

    system("pause");
    return 0;
}
```

> 技巧：看const右侧紧跟着的是指针还是常量，是指针就是常量指针，是常量就是指针常量

7.6 指针和数组

作用：利用指针访问数组中元素

示例：

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

int main()
{

    // 指针和数组
    // 利用指针访问数组中的元素

    int arr[] = { 1,2,3,4,5,6,7,8,9,10 };
    cout << "第一个元素为" << arr[0] << endl;

    int * p = arr; // arr就是数组的首地址
    cout << "利用指针来访问第一个元素：" << *p << endl;
    //p++;//让指针向后便宜4个字节
    //cout << "利用指针来访问第二个元素：" << *p << endl;

    cout << "利用指针遍历数组" << endl;
    for (int i = 0; i < 10; i++)
    {
        //cout << arr[i] << endl;
        cout << *p << endl;
        p++;

    }

```

```
    system("pause");
    return 0;
}
```

7.7 指针和函数

作用：利用指针作函数参数，可以修改实参的值

示例：

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

//实现两个数字进行交换
void swap01(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
    cout << "swap01 a=" << a << endl;
    cout << "swap01 b=" << b << endl;

}

void swap02(int *p1, int *p2)
{
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;

}

int main()
{
    //指针和函数
    //1、值传递
    int a = 10;
    int b = 20;
    //swap01(a, b);

    //2、地址传递
    //如果是地址传递，可以修饰实参
    swap02(&a, &b);

    cout << "a=" << a << endl;
    cout << "b=" << b << endl;

    system("pause");
    return 0;
}
```

> 总结：如果不想修改实参，就用值传递，如果想修改实参，就用地址传递

7.8 指针、数组、函数

案例描述：封装一个函数，利用冒泡排序，实现对整型数组的升序排序

例如数组：int arr[10] = { 4,3,6,9,1,2,10,8,7,5 };

示例：

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

//冒泡排序函数 参数1：数组的首地址，参数2：数组长度
void bubbleSort(int *arr, int len)
{
    for (int i = 0; i < len; i++)
    {
        for (int j = 0; j < len-i-1; j++)
        {
            //如果j>j+1的值，交换数字
            if (arr[j]>arr[j+1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;

            }
        }
    }

}

//打印数组
void printArray(int *arr, int len)
{
    for (int i = 0; i < len; i++)
    {
        cout << arr[i] << endl;

    }

}

int main()
{
    //1、先创建一个数组
    int arr[] = { 4,3,6,9,1,2,10,8,7,5 };
    int len = sizeof(arr) / sizeof(arr[0]); // 数组的长度

    //2、创建一个函数，实现冒泡排序
    bubbleSort(arr,len);

```

```
    //3、打印排序后的数组
    printArray(arr, len);

    system("pause");
    return 0;
}
```

> 总结：当数组名传入到函数作为参数时，被退化为指向首元素的指针

8 结构体

8.1 结构体基本概念

结构体属于用户自定义的数据类型，允许用户存储不同的数据类型

8.2 结构体定义和使用

语法： `struct 结构体名 { 结构体成员列表 };`

通过结构体创建变量的方式有三种：

- struct 结构体名 变量名
- struct 结构体名 变量名 = { 成员1值， 成员2值...}
- 定义结构体时顺便创建变量

示例：

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

//1、 创建学生数据类型：学生包括（姓名，年龄，分数）
// 自定义数据类型，一些类型集合组成的一个类型
// 语法 struct 类型名称 {成员列表};
struct Student
{
    //成员列表
    //姓名
    string name;
    //年龄
    int age;
    //分数
    int score;

}s3;// 顺便创建结构体变量--不建议使用第三种


//2、 通过学生类型创建具体学生
int main()
{

```

```
    //2.1 struct Student s1
        //struct关键字可以不写

    struct Student s1;
    //给s1属性赋值，通过. 访问结构体变量中的属性
    s1.name = "jjk";
    s1.age = 18;
    s1.score = 100;
    cout << "姓名: " << s1.name << "年龄: " << s1.age << "分数: " << s1.score << endl;

    //2.2 struct Student s2 = {...}
    struct Student s2 = { "贾继康",10,349 };
    cout << "姓名: " << s2.name << "年龄: " << s2.age << "分数: " << s2.score << endl;

    //2.3 定义结构体时顺便创建结构体变量
    s3.name = "王五";
    s3.age = 20;
    s3.score = 23;
    cout << "姓名: " << s3.name << "年龄: " << s3.age << "分数: " << s3.score << endl;

    system("pause");
    return 0;
}
```

> 总结1：定义结构体时的关键字是struct，不可省略

> 总结2：创建结构体变量时，关键字struct可以省略

> 总结3：结构体变量利用操作符 "." 访问成员

8.3 结构体数组

作用：将自定义的结构体放入到数组中方便维护

语法：`struct 结构体名 数组名[元素个数] = { {} , {} , ... {} }`

示例：

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

//结构体数组
//1、结构体定义
struct Student
{
    string name; // 姓名
    int age; //年龄
    int  score;//分数
};

int main()
{
```

```
    //2、创建结构体数组
    struct Student stuArray[3] =
    {
        {"张三",13,23},
        {"李四",23,435},
        {"王五",34,56}

    };


    //3、给结构体数组中的元素赋值
    stuArray[2].name = "赵柳";
    stuArray[2].age = 343;
    stuArray[2].score = 34;

    //4、遍历结构体数组
    for (int i = 0; i < 3; i++)
    {
        cout << "姓名: " << stuArray[i].name
             << "年龄: " << stuArray[i].age
             << "分数: " << stuArray[i].score << endl;

    }
    system("pause");
    return 0;
}
```

8.4 结构体指针

作用：通过指针访问结构体中的成员

- 利用操作符 `->` 可以通过结构体指针访问结构体属性

示例：

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

//结构体指针
//定义学生的结构体
struct student
{
    string name; //姓名
    int age; //年龄
    int score; //分数
};

int main()
{
    //1、创建学生结构体变量
```

```
    struct student s = { "张三",23,45 };

    //2、创建指针指向结构体变量
    struct student *p = &s;

    //3、通过指针访问结构体变量中的数据
    //通过结构体指针，访问结构体中的属性，需要利用'->'
    cout << "姓名: " << p->name << "年龄" << p->age << "分数: " << p->score << endl;

    system("pause");
    return 0;
}
```

> 总结：结构体指针可以通过 -> 操作符 来访问结构体中的成员

8.5 结构体嵌套结构体

作用： 结构体中的成员可以是另一个结构体

例如：每个老师辅导一个学员，一个老师的结构体中，记录一个学生的结构体

示例：

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

//学生结构体定义
struct student
{
    //学生姓名，年龄，考试分数
    string name;
    int age;
    int score;
};


//老师结构体定义
struct teacher
{
    int id; //教师编号
    string name; // 教师姓名
    int age; //教师年龄
    struct student stu; //老师有自己的学生

};

int main()
{
    //结构体嵌套结构体
    //创建老师
```

```
    struct teacher t;
    t.id = 10000;
    t.name = "老王";
    t.age = 50;
    t.stu.name = "小三";
    t.stu.age = 20;
    t.stu.score = 34;
    cout << "老师姓名: "<< t.name
        << "老师编号: " << t.id
        << "老师年龄: " << t.age
        << "老师辅导的学生姓名: " << t.stu.name
        << "学生年龄: " << t.stu.age
        << "学生成绩: " << t.stu.score << endl;

    system("pause");
    return 0;
}
```

总结: 在结构体中可以定义另一个结构体作为成员，用来解决实际问题

8.6 结构体做函数参数

作用: 将结构体作为参数向函数中传递

传递方式有两种:

- 值传递
- 地址传递

示例:

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


struct student
{
    //学生姓名，年龄，分数
    string name;
    int age;
    int score;
};

//打印学生信息函数
//1、值传递
void printStudent1(struct student s)
{
    cout << "子函数1中打印姓名: " << s.name << "年龄: " << s.age << "分数: " << s.score << endl;

}
//2、地址传递
```

```
void printStudent2(struct student * p)
{
    cout << "子函数2中打印姓名:" << p->name << "年龄:" << p->age << "分数:" << p->score << endl;
}

int main()
{

     //结构体做函数参数
    //将学生传入到一个参数中，打印学生身上的所有信息

    //创建结构体变量
    struct student s;
    s.name = "张三";
    s.age = 23;
    s.score = 34;

    printStudent1(s);
    printStudent2(&s);

    //cout << "main函数中打印姓名:" << s.name << "年龄:" << s.age << "分数:" << s.score << endl;

    system("pause");
    return 0;
}
```

总结：如果不想修改主函数中的数据，用值传递，反之用地址传递

8.7 结构体中 const使用场景

作用：用const来防止误操作

示例：

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

//const 使用场景

struct student
{
    //姓名，年龄，分数
    string name;
    int age;
    int score;

};
//打印函数
//将函数中的形参改为指针，可以减少内存空间，而且不会复制新的副本出来
//需要注意到：main函数的文件会因为打印函数的修改而随之修改，所以在形参中加上const，以致不能修改
```

```cpp
void printStudents(const struct student *s)
{
    //s->age = 100; //假如const之后，一旦有修改的操作就会报错，可以防止我们的误操作
    //cout << "姓名: " << s.name << "年龄: " << s.age << "分数: " << s.score << endl;
    cout << "姓名: " << s->name << "年龄: " << s->age << "分数: " << s->score << endl;
}

int main()
{
    //创建结构体变量及其初始化赋值
    struct student s = { "张三", 20, 34 };

    //通过函数打印结构体变量信息
    //printStudents(s);//值传递
    printStudents(&s); //地址传递

    system("pause");
    return 0;
}
```

8.8 结构体案例

8.8.1 案例1

案例描述:

学校正在做毕设项目，每名老师带领5个学生，总共有3名老师，需求如下

设计学生和老师的结构体，其中在老师的结构体中，有老师姓名和一个存放5名学生的数组作为成员

学生的成员有姓名、考试分数，创建数组存放3名老师，通过函数给每个老师及所带的学生赋值

最终打印出老师数据以及老师所带的学生数据。

示例:

```cpp
#include<iostream>
#include<string>
#include<ctime>
using namespace std;

//学生的结构体定义
struct student
{
    string name;
    int score;
};

//老师的结构体定义
struct teacher
{
    string name;
```

```
    struct student sArray[5];
};

//给老师和学生赋值的函数
void allocateSpace(struct teacher tArray[],int len)
{
    string nameSeed = "ABCDE";
    //给老师开始赋值
    for (int i = 0; i < len; i++)
    {

        tArray[i].name = "Teacher_";
        tArray[i].name += nameSeed[i];

        //通过循环给每名老师所带的学生赋值
        for (int j = 0; j < 5; j++)
        {
            tArray[i].sArray[j].name = "Student_";
            tArray[i].sArray[j].name += nameSeed[j];

            int random = rand() % 61+40;//  40-99
            tArray[i].sArray[j].score = random;

        }
    }
}

//打印所有信息
void printInfo(struct teacher tArray[], int len)
{
    for (int i = 0; i < len; i++)
    {
        cout << "老师的姓名: " << tArray[i].name << endl;
        for (int j = 0; j < 5; j++)
        {
            cout << "\t学生姓名: " << tArray[i].sArray[j].name
                 << "考试分数: " << tArray[i].sArray[j].score << endl;
        }
    }
}

int main()
{
     //随机数种子
    srand((unsigned int)time(NULL));
     //创建3名老师的数组
    struct teacher tArray[3];
    int len = sizeof(tArray) / sizeof(tArray[0]);

    //通过函数给3名老师的信息赋值，并且老师带的学生赋值
    allocateSpace(tArray,len);

    //打印所有老师及所带的学生信息
```

```
    printInfo(tArray,len);

    system("pause");
    return 0;
}
```

8.8.2 案例2

案例描述：

设计一个英雄的结构体，包括成员姓名，年龄，性别;创建结构体数组，数组中存放5名英雄。

通过冒泡排序的算法，将数组中的英雄按照年龄进行升序排序，最终打印排序后的结果。

五名英雄信息如下：

```
        {"刘备",23,"男"},
        {"关羽",22,"男"},
        {"张飞",20,"男"},
        {"赵云",21,"男"},
        {"貂蝉",19,"女"},
```

示例

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

//英雄的结构体
struct Hero
{
    string name;
    int age;
    string sex;

};

//冒泡排序 实现年龄升序排列
void bubbleSort(struct Hero heroArray[], int len)
{
    for (int i = 0; i < len-1; i++)
    {
        for (int j = 0; j < len-i-1; j++)
        {
            //如果j下标的元素 大于 j+1下标的元素的年龄，交换两个元素
            if (heroArray[j].age>heroArray[j+1].age)
            {
                struct Hero temp = heroArray[j];
                heroArray[j] = heroArray[j + 1];
                heroArray[j + 1] = temp;
```

```
            }
        }
    }
}

//打印函数
void printHero(struct Hero heroArray[], int len)
{
    for (int i = 0; i < len; i++)
    {
        cout << "英雄的姓名：" << heroArray[i].name
            << " 英雄的年龄：" << heroArray[i].age
            << "英雄的性别：" << heroArray[i].sex << endl;
    }
}

int main()
{
    //1、设计一个英雄的结构体
    //2、创建数组存放5名英雄
    struct Hero heroArray[5] =
    {
        {"刘备",23,"男"},
        {"关羽",22,"男"},
        {"张飞",20,"男"},
        {"赵云",21,"男"},
        {"貂蝉",19,"女"},
    };

    int len = sizeof(heroArray) / sizeof(heroArray[0]); // 获取数组的长度
    for (int i = 0; i < len; i++)
    {
        cout <<"英雄的姓名：" <<heroArray[i].name
             <<" 英雄的年龄：" <<heroArray[i].age
             <<"英雄的性别：" <<heroArray[i].sex << endl;

    }

    //3、对数组进行排序，按照年龄升序排序
    bubbleSort(heroArray,len);

    //4、将排序后的结果打印输出
    cout << "排序后的结果：" << endl;
    printHero(heroArray, len);

    system("pause");
    return 0;
}
```