

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：

```
1 返回值类型 函数名 （参数列表）
2  {
3
4      函数体语句
5
6      return表达式
7
8  }
```

- 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

```

1 //函数定义
2 int add(int num1, int num2)
3 {
4     int sum = num1 + num2;
5     return sum;
6 }

```

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：

```

1 #include<iostream>
2 #include<ctime> //time系统时间头文件
3 #include<string>
4 using namespace std;
5 //函数的定义
6 // 语法
7 // 返回值类型, 函数名 参数列表 具体的函数体语句 return表达式
8 // 加法函数, 实现两个整数相加, 并且将相加的结果进行返回
9
10 // 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
11 int add(int num1, int num2)
12 {
13     int sum = num1 + num2;
14     return sum;
15 }
16
17 int main()
18 {
19     int num1 = 1;
20     int num2 = 2;
21     // 调用函数
22     // 函数调用语法: 函数名称(参数)
23     // num1和num2成为实际参数, 简称实参; 在函数中并称之为形参
24     // 当调用函数的时候, 实参的值会传递给形参
25     int sum = add(num1, num2);
26     cout << "sum=" << sum << endl;
27
28     system("pause");
29     return 0;
30 }

```

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：

```
1  #include<iostream>
2  #include<ctime> //time系统时间头文件
3  #include<string>
4  using namespace std;
5
6  //值传递
7  // 定义函数，两个数字进行交换函数
8  void swap(int num1, int num2)
9  {
10
11     cout << "交换前: " << endl;
12     cout << "num1=" << num1 << endl;
13     cout << "num2=" << num2 << endl;
14
15     int temp = num1;
16     num1 = num2;
17     num2 = temp;
18
19     cout << "交换后: " << endl;
20     cout << "num1=" << num1 << endl;
21     cout << "num2=" << num2 << endl;
22
23     return; //或者都不需要写，或者返回值不需要的时候，可以不写return
24 }
25
26 int main()
27 {
28
29     int a = 10;
30     int b = 20;
31     cout << "a=" << a << endl;
32     cout << "b=" << b << endl;
33     // 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
34     swap(a, b);
35
36     cout << "a=" << a << endl;
37     cout << "b=" << b << endl;
38     system("pause");
39     return 0;
40 }
```

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：

```
1  #include<iostream>
2  #include<ctime> //time系统时间头文件
3  #include<string>
4  using namespace std;
5
6  //函数常见样式
7
8  //1、无参无返
9  void test01()
10 {
11     cout << "this is test01" << endl;
12
13 }
14 //2、有参无返
15 void test02(int a)
16 {
17     cout << "this is test02 a=" << a << endl;
18     return;
19 }
20 //3、无参有返
21 int test03()
22 {
23     cout << "this is test03" << endl;
24     return 100;
25
26 }
27
28 //4、有参有返
29 int test04(int a)
30 {
31     cout << "this is test04 a=" << a << endl;
32     return a;
33 }
34
35 int main()
36 {
37     // 无参无返函数调用
38     test01();
39     // 有参无返函数调用
40     test02(100);
41     // 无参有返函数调用
42     int num1 = test03();
43     cout << "num1=" << num1 << endl;
44     // 有参有返函数调用
45     int num2 = test04(1000);
46     cout << "num2=" << num2 << endl;
47     system("pause");
```

```
48     return 0;
49 }
```

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：

```
1  #include<iostream>
2  #include<ctime> //time系统时间头文件
3  #include<string>
4  using namespace std;
5
6  // 提前告诉编译器函数的存在，可以利用函数的声明
7  int max(int a, int b); // 函数声明
8
9
10 // 声明可以多次，定义只能一次
11 // 函数声明
12 // 比较函数，实现两个整型数字进行比较，返回较大的值
13 int max(int a, int b)
14 {
15     return a > b ? a : b; // 三目运算符
16 }
17
18
19 int main()
20 {
21     int a = 10;
22     int b = 20;
23     cout << max(a, b) << endl;
24     system("pause");
25     return 0;
26 }
```

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：

```
1 //swap.h文件
2 #include<iostream>
3 using namespace std;
4
5 // 函数的声明
6 void swap(int a, int b);
7
```

```
1 //swap.cpp文件
2 #include "swap.h" //自定义文件
3
4 //函数的定义
5 void swap(int a, int b)
6 {
7     int temp = a;
8     a = b;
9     b = temp;
10    cout << "a=" << a << endl;
11    cout << "b=" << b << endl;
12
13 }
```

```
1 //main函数文件
2 #include<iostream>
3 using namespace std;
4
5 #include "swap.h"
6
7 // 1、创建.h后缀名的头文件 swap.h
8 // 2、创建.cpp后缀名的源文件 swap.cpp
9 // 3、在头文件中写函数的声明
10 // 4、在源文件中先函数的定义
11
12 int main()
13 {
14     int a = 10;
15     int b = 20;
16     swap(a, b);
17
18     system("pause");
19     return 0;
20 }
```

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：

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      // 1、定义指针
7      int a = 10;
8      // 指针定义的语法： 数据类型 * 指针变量名
9      int *p;
10     // 让指针记录变量a的地址
11     p = &a; // &取址变量符
12     cout << "a的地址为：" << &a << endl; // 打印的是a的地址
13     cout << "指针p等于：" << p << endl; // 打印的是a的地址
14
15     // 2、使用指针
16     // 可以通过解引用的方式来找到指针指向的内存
17     // 指针前面加 *，找到指针指向的内存中的数据
18     *p = 100;
19     cout << "a=" << a << endl;
20     cout << "*p=" << *p << endl;
21
22     system("pause");
23     return 0;
24 }
```

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：

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      //指针所占内存空间
7      int a = 10;
8      //int *p;
9      //p = &a; // 指针p指向a的首地址
10     int *p = &a;
11     //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
12     //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
13     cout << "sizeof (int *) = " << sizeof(int *) << endl; // sizeof(p)
14     cout << "sizeof (float *) = " << sizeof(float *) << endl; // sizeof(p)
15     cout << "sizeof (double *) = " << sizeof(double *) << endl; // sizeof(p)
16     cout << "sizeof (char *) = " << sizeof(char *) << endl; // sizeof(p)
17
18     system("pause");
19     return 0;
20 }
```

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

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      //空指针
7      //1、空指针用于给指针变量进行初始化
8      int *p = NULL;
9
10     //2、空指针是不可以进行访问的
11     //0~255之间的内存编号是系统占用的，因此不可以访问
12     /*p = 100;
13     system("pause");
14     return 0;
15 }
```


野指针：指针变量指向非法的内存空间

示例2：野指针

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      //野指针
7      // 在程序中，尽量避免出现野指针
8      //指针变量p指向内存地址编号为0x1100的空间
9      int *p = NULL; //空指针
10     int *p1 = (int *)0x1100;
11
12     system("pause");
13     return 0;
14 }
```

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

1. const修饰指针 --- 常量指针

```
int a = 10;
int b = 10;
int * p = &a;
```

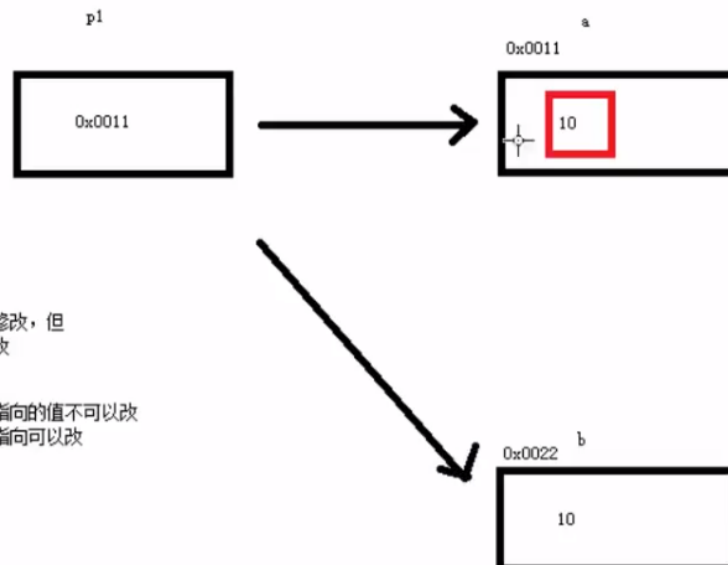
```
const int * p = &a;
```

常量指针

特点：指针的指向可以修改，但是指针指向的值不可以改

*p = 20; 错误，指针指向的值不可以改

p = &b; 正确，指针指向可以改



2. const修饰常量 --- 指针常量

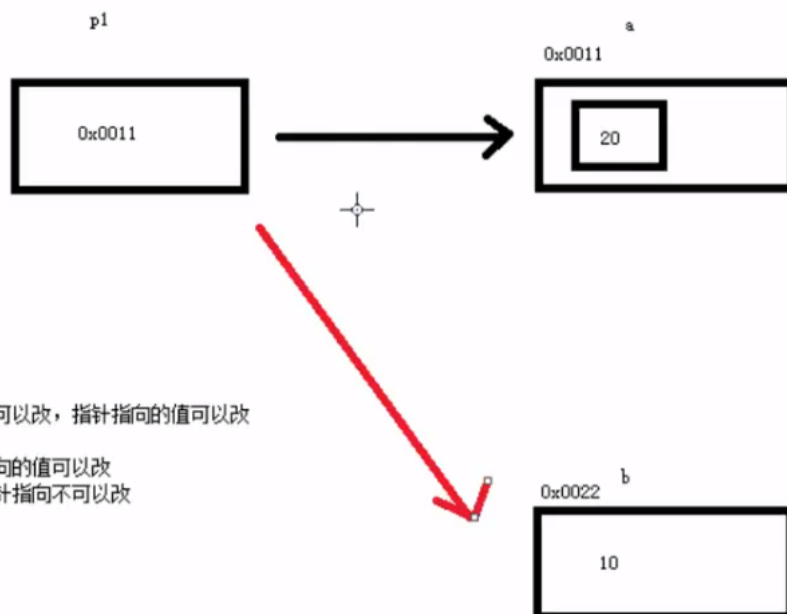
2. const修饰常量 —— 指针常量

```
int a = 10;  
int b = 10;  
int * p = &a;
```

```
int * const p = &a;  
指针常量
```

特点：指针的指向不可以改，指针指向的值可以改

*p = 20; 正确，指向的值可以改
p = &b; 错误，指针指向不可以改



3. const即修饰指针，又修饰常量

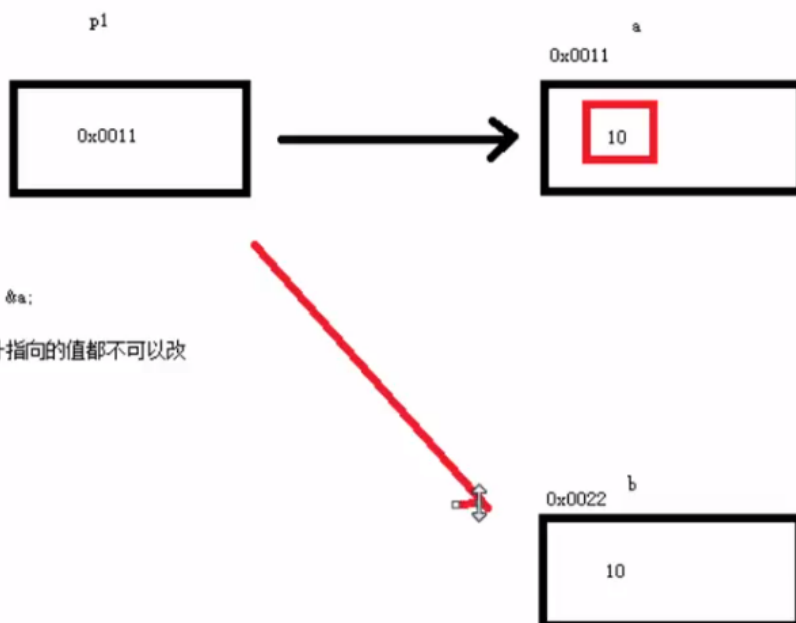
3. const即修饰指针，又修饰常量

```
int a = 10;  
int b = 10;  
int * p = &a;
```

```
const int * const p = &a;
```

特点：指针的指向和指针指向的值都不可以改

*p = 20; //错误
p = &b; //错误



示例：

```
1  #include<iostream>  
2  using namespace std;  
3  
4  int main()  
5  {  
6  
7      //1、const修饰指针  
8      int a = 10;  
9      int b = 10;  
10
```

```

11     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
12     /*p = 20; 错误
13     p = &b; //正确
14
15     //2、const修饰常量    指针常量
16     // 指针的指向不可以改，指针指向的值可以改
17     int * const p2 = &a; //
18     *p2 = 100; //正确
19     //p2 = &b; //错误，指针的指向不可以改
20
21     //3、const修饰常量和指针
22     const int * const p3 = &a;
23     /*p3 = 100; 错误
24     //p3 = &b; 错误
25
26     system("pause");
27     return 0;
28 }

```

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：

```

1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6
7      // 指针和数组
8      // 利用指针访问数组中的元素
9
10     int arr[] = { 1,2,3,4,5,6,7,8,9,10 };
11     cout << "第一个元素为" << arr[0] << endl;
12
13     int * p = arr; // arr就是数组的首地址
14     cout << "利用指针来访问第一个元素：" << *p << endl;
15     //p++; //让指针向后便宜4个字节
16     //cout << "利用指针来访问第二个元素：" << *p << endl;
17
18     cout << "利用指针遍历数组" << endl;
19     for (int i = 0; i < 10; i++)
20     {
21         //cout << arr[i] << endl;
22         cout << *p << endl;
23         p++;
24     }
25 }
26

```

```
27     system("pause");
28     return 0;
29 }
```

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：

```
1  #include<iostream>
2  using namespace std;
3
4  //实现两个数字进行交换
5  void swap01(int a, int b)
6  {
7      int temp = a;
8      a = b;
9      b = temp;
10     cout << "swap01 a=" << a << endl;
11     cout << "swap01 b=" << b << endl;
12
13 }
14
15 void swap02(int *p1, int *p2)
16 {
17     int temp = *p1;
18     *p1 = *p2;
19     *p2 = temp;
20
21 }
22
23 int main()
24 {
25     //指针和函数
26     //1、值传递
27     int a = 10;
28     int b = 20;
29     //swap01(a, b);
30
31     //2、地址传递
32     //如果是地址传递，可以修饰实参
33     swap02(&a, &b);
34
35     cout << "a=" << a << endl;
36     cout << "b=" << b << endl;
37
38     system("pause");
39     return 0;
40 }
```

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：

```
1  #include<iostream>
2  using namespace std;
3
4  //冒泡排序函数 参数1: 数组的首地址, 参数2: 数组长度
5  void bubbleSort(int *arr, int len)
6  {
7      for (int i = 0; i < len; i++)
8      {
9          for (int j = 0; j < len-i-1; j++)
10         {
11             //如果j>j+1的值, 交换数字
12             if (arr[j]>arr[j+1])
13             {
14                 int temp = arr[j];
15                 arr[j] = arr[j + 1];
16                 arr[j + 1] = temp;
17             }
18         }
19     }
20 }
21
22 //打印数组
23 void printArray(int *arr, int len)
24 {
25     for (int i = 0; i < len; i++)
26     {
27         cout << arr[i] << endl;
28     }
29 }
30
31 int main()
32 {
33     //1、先创建一个数组
34     int arr[] = { 4,3,6,9,1,2,10,8,7,5 };
35     int len = sizeof(arr) / sizeof(arr[0]); // 数组的长度
36
37     //2、创建一个函数, 实现冒泡排序
38     bubbleSort(arr, len);
39 }
```

```
44 //3、打印排序后的数组
45 printArray(arr, len);
46
47 system("pause");
48 return 0;
49 }
```

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：

```
1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  //1、 创建学生数据类型：学生包括（姓名，年龄，分数）
6  // 自定义数据类型，一些类型集合组成的一个类型
7  // 语法 struct 类型名称 {成员列表};
8  struct Student
9  {
10     //成员列表
11     //姓名
12     string name;
13     //年龄
14     int age;
15     //分数
16     int score;
17
18 }s3; // 顺便创建结构体变量——不建议使用第三种
19
20
21 //2、 通过学生类型创建具体学生
22 int main()
23 {
24
```

```

25 //2.1 struct Student s1
26 //struct关键字可以不写
27
28 struct Student s1;
29 //给s1属性赋值，通过. 访问结构体变量中的属性
30 s1.name = "jjk";
31 s1.age = 18;
32 s1.score = 100;
33 cout << "姓名: " << s1.name << "年龄: " << s1.age << "分数: " << s1.score << endl;
34
35 //2.2 struct Student s2 = {...}
36 struct Student s2 = { "贾继康", 10, 349 };
37 cout << "姓名: " << s2.name << "年龄: " << s2.age << "分数: " << s2.score << endl;
38
39 //2.3 定义结构体时顺便创建结构体变量
40 s3.name = "王五";
41 s3.age = 20;
42 s3.score = 23;
43 cout << "姓名: " << s3.name << "年龄: " << s3.age << "分数: " << s3.score << endl;
44
45 system("pause");
46 return 0;
47 }

```

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:

```

1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  //结构体数组
6  //1、结构体定义
7  struct Student
8  {
9      string name; // 姓名
10     int age; //年龄
11     int score; //分数
12 };
13
14 int main()
15 {

```

```

16 //2、创建结构体数组
17 struct Student stuArray[3] =
18 {
19     {"张三", 13, 23},
20     {"李四", 23, 435},
21     {"王五", 34, 56}
22
23 };
24
25
26 //3、给结构体数组中的元素赋值
27 stuArray[2].name = "赵柳";
28 stuArray[2].age = 343;
29 stuArray[2].score = 34;
30
31 //4、遍历结构体数组
32 for (int i = 0; i < 3; i++)
33 {
34     cout << "姓名: " << stuArray[i].name
35         << "年龄: " << stuArray[i].age
36         << "分数: " << stuArray[i].score << endl;
37
38 }
39 system("pause");
40 return 0;
41 }

```

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：

```

1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  //结构体指针
6  //定义学生的结构体
7  struct student
8  {
9      string name; //姓名
10     int age; //年龄
11     int score; //分数
12 };
13
14 int main()
15 {
16     //1、创建学生结构体变量

```



```

17     struct student s = { "张三",23,45 };
18
19     //2、创建指针指向结构体变量
20     struct student *p = &s;
21
22     //3、通过指针访问结构体变量中的数据
23     //通过结构体指针，访问结构体中的属性，需要利用'-'>'
24     cout << "姓名: " << p->name << "年龄" << p->age << "分数: " << p->score << endl;
25
26     system("pause");
27     return 0;
28 }

```

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：

```

1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  //学生结构体定义
6  struct student
7  {
8      //学生姓名，年龄，考试分数
9      string name;
10     int age;
11     int score;
12 };
13
14
15 //老师结构体定义
16 struct teacher
17 {
18     int id; //教师编号
19     string name; // 教师姓名
20     int age; //教师年龄
21     struct student stu; //老师有自己的学生
22
23 };
24
25 int main()
26 {
27     //结构体嵌套结构体
28     //创建老师

```

```

29     struct teacher t;
30     t.id = 10000;
31     t.name = "老王";
32     t.age = 50;
33     t.stu.name = "小三";
34     t.stu.age = 20;
35     t.stu.score = 34;
36     cout << "老师姓名: " << t.name
37         << "老师编号: " << t.id
38         << "老师年龄: " << t.age
39         << "老师辅导的学生姓名: " << t.stu.name
40         << "学生年龄: " << t.stu.age
41         << "学生成绩: " << t.stu.score << endl;
42
43     system("pause");
44     return 0;
45 }

```

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：

```

1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5
6  struct student
7  {
8      //学生姓名, 年龄, 分数
9      string name;
10     int age;
11     int score;
12 };
13
14 //打印学生信息函数
15 //1、值传递
16 void printStudent1(struct student s)
17 {
18     cout << "子函数1中打印姓名: " << s.name << "年龄: " << s.age << "分数: " << s.score <<
19     endl;
20 }
21 //2、地址传递

```

```

22 void printStudent2(struct student * p)
23 {
24     cout << "子函数2中打印姓名:" << p->name << "年龄:" << p->age << "分数:" << p->score <<
endl;
25 }
26
27 int main()
28 {
29
30     //结构体做函数参数
31     //将学生传入到一个参数中，打印学生身上的所有信息
32
33     //创建结构体变量
34     struct student s;
35     s.name = "张三";
36     s.age = 23;
37     s.score = 34;
38
39     printStudent1(s);
40     printStudent2(&s);
41
42     //cout << "main函数中打印姓名:" << s.name << "年龄:" << s.age << "分数:" << s.score <<
endl;
43
44     system("pause");
45     return 0;
46 }

```

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：

```

1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  //const 使用场景
6
7  struct student
8  {
9      //姓名，年龄，分数
10     string name;
11     int age;
12     int score;
13
14 };
15 //打印函数
16 //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
17 //需要注意：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

```

```

18 void printStudents(const struct student *s)
19 {
20     //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
21     //cout << "姓名: " << s.name << "年龄: " << s.age << "分数: " << s.score << endl;
22     cout << "姓名: " << s->name << "年龄: " << s->age << "分数: " << s->score << endl;
23 }
24
25 int main()
26 {
27     //创建结构体变量及其初始化赋值
28     struct student s = { "张三", 20, 34 };
29
30     //通过函数打印结构体变量信息
31     //printStudents(s); //值传递
32     printStudents(&s); //地址传递
33
34     system("pause");
35     return 0;
36 }

```

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:

```

1  #include<iostream>
2  #include<string>
3  #include<ctime>
4  using namespace std;
5
6  //学生的结构体定义
7  struct student
8  {
9      string name;
10     int score;
11 };
12
13 //老师的结构体定义
14 struct teacher
15 {
16     string name;

```

```

17     struct student sArray[5];
18 };
19
20 //给老师和学生赋值的函数
21 void allocateSpace(struct teacher tArray[],int len)
22 {
23     string nameSeed = "ABCDE";
24     //给老师开始赋值
25     for (int i = 0; i < len; i++)
26     {
27
28         tArray[i].name = "Teacher_";
29         tArray[i].name += nameSeed[i];
30
31         //通过循环给每名老师所带的学生赋值
32         for (int j = 0; j < 5; j++)
33         {
34             tArray[i].sArray[j].name = "student_";
35             tArray[i].sArray[j].name += nameSeed[j];
36
37             int random = rand() % 61+40;// 40-99
38             tArray[i].sArray[j].score = random;
39
40         }
41     }
42 }
43
44 //打印所有信息
45 void printInfo(struct teacher tArray[], int len)
46 {
47     for (int i = 0; i < len; i++)
48     {
49         cout << "老师的姓名: " << tArray[i].name << endl;
50         for (int j = 0; j < 5; j++)
51         {
52             cout << "\t学生姓名: " << tArray[i].sArray[j].name
53                 << "考试分数: " << tArray[i].sArray[j].score << endl;
54         }
55     }
56 }
57
58 int main()
59 {
60     //随机数种子
61     srand((unsigned int)time(NULL));
62     //创建3名老师的数组
63     struct teacher tArray[3];
64     int len = sizeof(tArray) / sizeof(tArray[0]);
65
66     //通过函数给3名老师的信息赋值, 并且老师带的学生赋值
67     allocateSpace(tArray,len);
68
69     //打印所有老师及所带的学生信息

```

```

70     printInfo(tArray, len);
71
72     system("pause");
73     return 0;
74 }

```

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:

```

1     {"刘备", 23, "男"},
2     {"关羽", 22, "男"},
3     {"张飞", 20, "男"},
4     {"赵云", 21, "男"},
5     {"貂蝉", 19, "女"},

```

示例

```

1  #include<iostream>
2  #include<string>
3  #include<ctime>
4  using namespace std;
5
6  //英雄的结构体
7  struct Hero
8  {
9      string name;
10     int age;
11     string sex;
12
13 };
14
15 //冒泡排序 实现年龄升序排列
16 void bubbleSort(struct Hero heroArray[], int len)
17 {
18     for (int i = 0; i < len-1; i++)
19     {
20         for (int j = 0; j < len-i-1; j++)
21         {
22             //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
23             if (heroArray[j].age > heroArray[j+1].age)
24             {
25                 struct Hero temp = heroArray[j];
26                 heroArray[j] = heroArray[j + 1];
27                 heroArray[j + 1] = temp;

```

```

28     }
29 }
30 }
31 }
32
33 //打印函数
34 void printHero(struct Hero heroArray[], int len)
35 {
36     for (int i = 0; i < len; i++)
37     {
38         cout << "英雄的姓名: " << heroArray[i].name
39             << " 英雄的年龄: " << heroArray[i].age
40             << "英雄的性别: " << heroArray[i].sex << endl;
41     }
42 }
43
44 int main()
45 {
46     //1、设计一个英雄的结构体
47     //2、创建数组存放5名英雄
48     struct Hero heroArray[5] =
49     {
50         {"刘备", 23, "男"},
51         {"关羽", 22, "男"},
52         {"张飞", 20, "男"},
53         {"赵云", 21, "男"},
54         {"貂蝉", 19, "女"},
55     };
56
57     int len = sizeof(heroArray) / sizeof(heroArray[0]); // 获取数组的长度
58     for (int i = 0; i < len; i++)
59     {
60         cout << "英雄的姓名: " << heroArray[i].name
61             << " 英雄的年龄: " << heroArray[i].age
62             << "英雄的性别: " << heroArray[i].sex << endl;
63     }
64 }
65
66 //3、对数组进行排序, 按照年龄升序排序
67 bubbleSort(heroArray, len);
68
69 //4、将排序后的结果打印输出
70 cout << "排序后的结果: " << endl;
71 printHero(heroArray, len);
72
73 system("pause");
74 return 0;
75 }

```