加油！当自己的实力不能满足自己的目标时，**就静下心去学习**！

@[toc]

# 结构体

## 结构体的定义和使用

**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：**

//结构体定义  
struct student  
{  
 //成员列表  
 string name; //姓名  
 int age; //年龄  
 int score; //分数  
}stu3; //结构体变量创建方式3   
  
  
int main() {  
  
 //结构体变量创建方式1  
 struct student stu1; //struct 关键字可以省略  
  
 stu1.name = "张三";  
 stu1.age = 18;  
 stu1.score = 100;  
   
 cout << "姓名：" << stu1.name << " 年龄：" << stu1.age << " 分数：" << stu1.score << endl;  
  
 //结构体变量创建方式2  
 struct student stu2 = { "李四",19,60 };  
  
 cout << "姓名：" << stu2.name << " 年龄：" << stu2.age << " 分数：" << stu2.score << endl;  
  
  
 stu3.name = "王五";  
 stu3.age = 18;  
 stu3.score = 80;  
   
  
 cout << "姓名：" << stu3.name << " 年龄：" << stu3.age << " 分数：" << stu3.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  
2：创建结构体变量时，关键字struct可以省略  
3：结构体变量利用操作符 ''.'' 访问成员

## 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：**

//结构体定义  
struct student  
{  
 //成员列表  
 string name; //姓名  
 int age; //年龄  
 int score; //分数  
}  
  
int main() {  
   
 //结构体数组  
 struct student arr[3]=  
 {  
 {"张三",18,80 },  
 {"李四",19,60 },  
 {"王五",20,70 }  
 };  
  
 for (int i = 0; i < 3; i++)  
 {  
 cout << "姓名：" << arr[i].name << " 年龄：" << arr[i].age << " 分数：" << arr[i].score << endl;  
 }  
  
 system("pause");  
  
 return 0;  
}

## 结构体指针

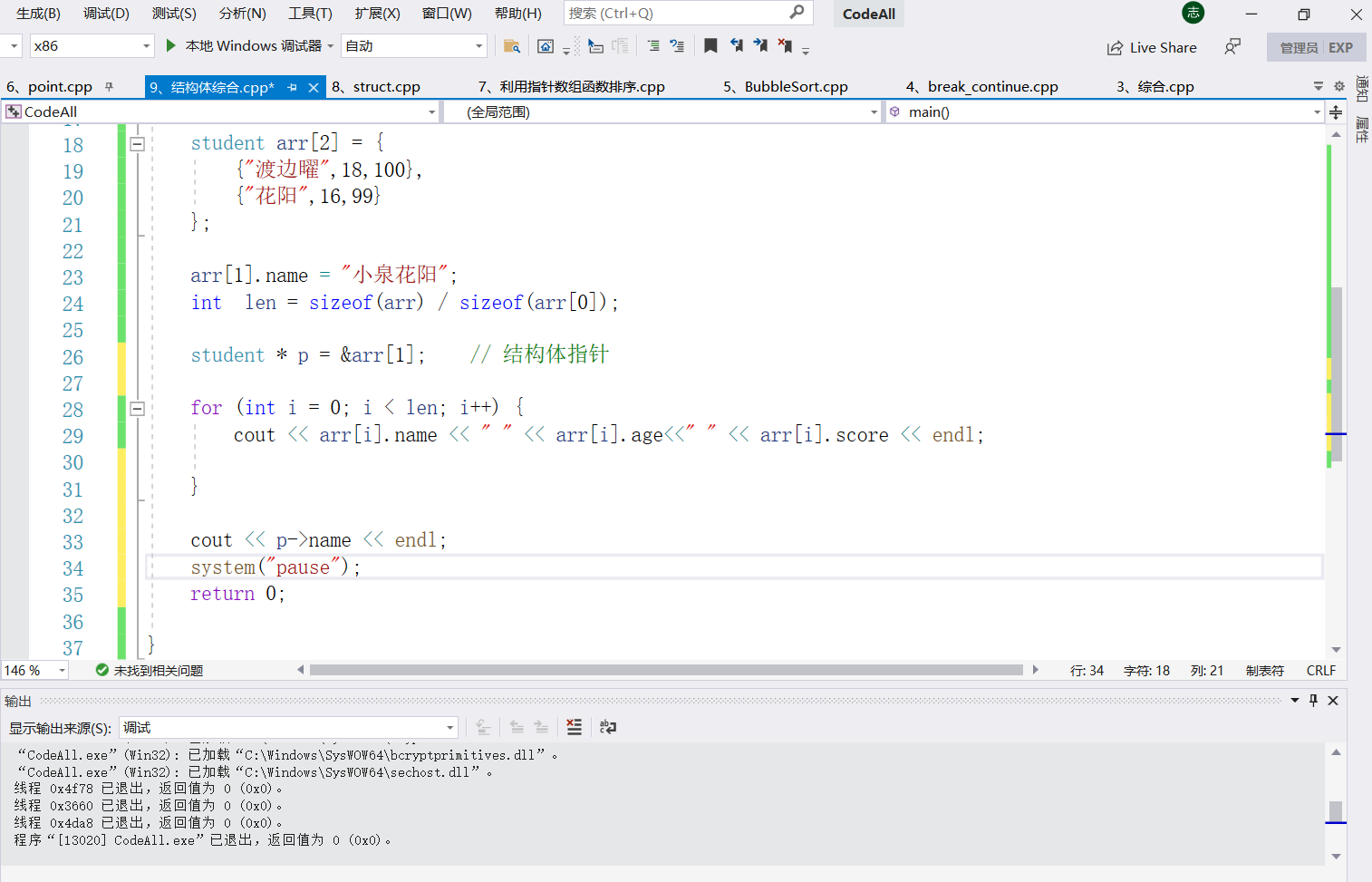
**作用** ： 通过指针访问结构体中的成员

* 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>  
using namespace std;  
#include <iostream>  
//结构体定义  
struct student  
{  
 //成员列表  
 string name; //姓名  
 int age; //年龄  
 int score; //分数  
};  
  
  
int main() {  
   
 struct student stu = { "张三",18,100, };  
   
 struct student \* p = &stu;  
   
 p->score = 80; //指针通过 -> 操作符可以访问成员  
  
 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



Code:

#include <iostream>  
using namespace std;  
#include <iostream>  
  
/// <summary>  
/// 结构体指针 结构体数组  
/// </summary>  
  
struct student {  
 string name;  
 int age;  
 int score;  
  
};  
  
int main() {  
  
 student arr[2] = {  
 {"渡边曜",18,100},  
 {"花阳",16,99}  
 };  
  
 arr[1].name = "小泉花阳";  
 int len = sizeof(arr) / sizeof(arr[0]);  
   
 student \* p = &arr[1]; // 结构体指针  
  
 for (int i = 0; i < len; i++) {  
 cout << arr[i].name << " " << arr[i].age<<" " << arr[i].score << endl;  
   
 }  
  
 cout << p->name << endl;  
 system("pause");  
 return 0;  
  
}

注意：代码中

## 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：**

//学生结构体定义  
struct student  
{  
 //成员列表  
 string name; //姓名  
 int age; //年龄  
 int score; //分数  
};  
  
//教师结构体定义  
struct teacher  
{  
 //成员列表  
 int id; //职工编号  
 string name; //教师姓名  
 int age; //教师年龄  
 struct student stu; //子结构体 学生  
};  
  
  
int main() {  
  
 struct teacher t1;  
 t1.id = 10000;  
 t1.name = "老王";  
 t1.age = 40;  
  
 t1.stu.name = "张三";  
 t1.stu.age = 18;  
 t1.stu.score = 100;  
  
 cout << "教师 职工编号： " << t1.id << " 姓名： " << t1.name << " 年龄： " << t1.age << endl;  
   
 cout << "辅导学员 姓名： " << t1.stu.name << " 年龄：" << t1.stu.age << " 考试分数： " << t1.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

## 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：**

//学生结构体定义  
struct student  
{  
 //成员列表  
 string name; //姓名  
 int age; //年龄  
 int score; //分数  
};  
  
//值传递  
void printStudent(student stu )  
{  
 stu.age = 28;  
 cout << "子函数中 姓名：" << stu.name << " 年龄： " << stu.age << " 分数：" << stu.score << endl;  
}  
  
//地址传递  
void printStudent2(student \*stu)  
{  
 stu->age = 28;  
 cout << "子函数中 姓名：" << stu->name << " 年龄： " << stu->age << " 分数：" << stu->score << endl;  
}  
  
int main() {  
  
 student stu = { "张三",18,100};  
 //值传递  
 printStudent(stu);  
 cout << "主函数中 姓名：" << stu.name << " 年龄： " << stu.age << " 分数：" << stu.score << endl;  
  
 cout << endl;  
  
 //地址传递  
 printStudent2(&stu);  
 cout << "主函数中 姓名：" << stu.name << " 年龄： " << stu.age << " 分数：" << stu.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

注意：地址传递格式

void printStudent2(student \*stu) { }  
  
printStudent2(&stu);

## 结构体中 const使用场景

* **产生原因**： 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
* **作用** : 用const来防止误操作

**示例：**

//学生结构体定义  
struct student  
{  
 //成员列表  
 string name; //姓名  
 int age; //年龄  
 int score; //分数  
};  
  
//const使用场景  
void printStudent(const student \*stu) //加const防止函数体中的误操作  
{  
 //stu->age = 100; //操作失败，因为加了const修饰  
 cout << "姓名：" << stu->name << " 年龄：" << stu->age << " 分数：" << stu->score << endl;  
  
}  
  
int main() {  
  
 student stu = { "张三",18,100 };  
  
 printStudent(&stu);  
  
 system("pause");  
  
 return 0;  
}

## Tips

### C++随机数构造

#include <ctime>  
srand((unsigned int)time(NULL));

* 随机数种子 头文件

rand() % 61 + 40;

* 前者 0~60 --> 40~100