结构体

一、为什么需要结构体？

为了表示一些复杂的事物，而普通的基本类型无法满足实际要求。

[为什么需要结构体.cpp]

```
#include<stdio.h>
//struct 结构
struct Student{//构造了一个数据类型,这个数据类型里面有age、score、sex; //构造的这个新的数据类型代表的学生类型
    int age;//年龄
    float score;//成绩
    char sex;//性别
}; //这就是一个结构体
int main (void){
    struct Student st ={80, 60.0, 'f'};
        //利用上面构造的数据类型定义一个变量st，st里面有3个成员(age、score、sex)。

    return 0;
}
```

二、什么叫结构体？

把一些基本类型数据组合在一起形成的一个新的复合数据类型，这个就叫结构体。

三、如何定义结构体？

三种方式，推荐使用第一种。

[如何定义结构体.cpp]

```
#include<stdio.h>
//第一种方式(推荐使用)只是构造了一个新的数据类型，并没有定义变量
struct Student1{
    int age;
    float score;
    char sex;
};
//第二种方式(在定义的同时定义变量名)
struct Student2{
    int age;
    float score;
    char sex;
}st2;
//第三种方式(最不好的一种，连构造的变量类型是什么都不知道)
struct{
    int age;
    float score;
    char sex;
}st3;
int main(void){

    struct Student1 st = {80, 66.6, 'f'};
    return 0;
}
```

四、怎样使用结构体变量？

1、赋值和初始化

第一种定义赋值：定义的同时可以整体赋初始值；

第二种定义赋值：定义完后，则只能单个赋值。+

[结构体变量的赋值和初始化.cpp]

```
#include<stdio.h>
//定义结构体的第一种方式(推荐使用)
struct Student{
    int age;
    float score;
    char sex;
};
int main (void){
    //第一种定义赋值     定义的同时可以整体赋值初始化
    struct Student st1 = {80, 66.6, 'f'};//初始化    利用构造的数据类型定义一个变量，同时赋值。
    /*
        类似
        int a[5] = {1,2,3,4,5};
    */

    //第二种定义赋值     定义完后，只能单个赋值
    struct Student st2; //先定义
    st2.age = 88;       //再单个赋值
    st2.score = 66.6;
    st2.sex = 'f';
    /*
        类似
        int a[5];
        a[0] = 1;
        a[1] = 2;
        ...
    */

    //输出
    printf("%d %f %c\n",st1.age, st1.score, st1.sex);
    printf("%d %f %c\n",st2.age, st2.score, st2.sex);

    return 0;
}
```

2、如何取出结构体变量中的每一个成员【重点】

Ⅰ、结构体变量名.成员名

Ⅱ、指针变量名->成员名（常用）

指针变量名->成员名 在计算机内部会被转化成（*指针变量名).成员名的方式来执行

所以说这两种方式是等价的

[如何取出结构体变量中的每一个成员.cpp]

```
#include <stdio.h>
//定义结构体的第一种方式(推荐使用)
struct Student{
    int age;
    float score;
    char sex;
};
int main(void){
    struct Student st = {80, 66.6f, 'f'};//初始化
    struct Student * pst = &st;//表示pst可以存放前面那个类型变量的地址
                               //&st1不能写成st1,&不可以省略
    st.age = 88;    //第一种方式
    pst->score = 66.6f; //第二种方式  pst->age在计算机内部会被转化成(*pst).age
                        /*
                        66.6在C语言中默认是double类型，如果希望一个实数是float类型，
                        则必须在末尾加f或F，因此66.6是double类型的，但是66.6f或66.6F是float类型的
                        */

    printf("%d\n%f", pst->age, st.score);
    return 0;
}
/*
    在dev c++中的输出结果是：
    ------------------------
    88
    66.599998- - 因为浮点型不能保证每一个数字都准确存储
    ------------------------
*/
```

①pst->age 在计算机内部会被转化成(*pst).age 没有为什么，这就是->的含义，这是一种硬性规定；

②所以 pst->age 等价于(*pst).age 也等价于 st.age

③我们之所以知道 pst->age 等价于 st.age，是因为 pst->age 是被转化成了(*pst).age 来执行

④pst->age 的含义

pst 所指向的那个结构体变量中 age 这个成员

3、结构体变量和结构体变量指针作为函数参数传递的问题

推荐使用结构体指针变量作为函数参数来传递

[结构体变量和结构体变量指针作为函数参数传递的问题.cpp]

```
/*
    2021年1月8日
    通过函数完成对结构体变量的输入和输出
*/
#include <stdio.h>
#include <string.h>
//定义了一个数据类型 名字是:struct Student
struct Student {
    int age;
    char sex;
    char name[100];
};//分号不能省略

/*  本函数无法修改main函数st的值    所以本函数是错误的
    void InputStudent(struct Student str)
    strcopy(str.name,"张三");
    str.age = 20;
    str.sex = 'M';
    原因:   不能这样写,用struct Student数据类型名分配了105个字节,
            是静态内存，函数执行完空间会被释放，不会改变st的值
*/

//正确写法  因为想在别的函数修改st的值,必须发送st的地址
void InputStudent(struct Student * pst) {
    //pst只占4个字节 pst只存放st的首字节地址，但是pst指向st这个变量(因为struct Student *)
    //  (*pst).age = 20;等价于 st.age = 20;
    pst->age = 20;
    pst->sex = 'M';
    strcpy(pst->name, "张三");  //strcpy是字符串拷贝,将" 张三" 拷贝到name里面
    //必须加个头文件#include<string.h>
}

void  OutputStudent(struct Student * Outst) { //如果函数内部不修改st变量的值,就不需要使用指针
    printf("%d\n%c\n%s", Outst->age, Outst->sex, Outst->name);
}
int main(void) {
    struct Student st;
    printf("%d\n",sizeof(st));//st这个变量占多少个字节
    //对结构体变量输入
    InputStudent(&st);//在其他函数修改本函数内变量的值,实参为变量的地址,被调用函数的形参为指针类型.
    //对结构体变量输出
        /*
            发送地址还是发送内容(如果发送内容会将st的108个字节发送出去,也会在函数中分配108个字节,浪费内存)
            指针的优点之一:
                快速传递内容
                耗用内存小
                执行速度快
        */
    OutputStudent(&st);//将st发送给输出函数，相当于Outst是st的拷贝，所以输出Outst相当于输出st

//  printf("%d\n%c\n%s", st.age, st.sex, st.name);
    return 0;
}
/*
    对结构体输入,必须发送st的地址,因为需要在别的函数中输入和修改st的值,
    所以输入函数的形参Inst需要是(struct Student *)
    对结构体输出,输出函数可以接收st的地址也可以接收st的内容,但为了减少内存的损耗,
    也为了提高执行速度,推荐发送地址.
*/
```

4、结构体变量的运算

结构体变量不能相加，不能相减，也不能相互乘除，但结构体变量可以相互赋值。

例子：

```
struct Student{
    int age;
    char sex;
    char name[100];//分号不能省略
};
struct Student st1, st2;
st1+st2; st1*st2; st1/st2;//都是错误的
st1=st2; 或者 st2=st1;//都是正确的
```

举例：

动态构造存放学生信息的结构体数组(动态构造一个数组，存放学生的信息，然后按分数排序输出)

[学生管理系统、结构体、动态数组.cpp]

```
#include<stdio.h>
#include<malloc.h>
struct Student{
    int age;
    float score;
    char name[100];
};
int main(void){
    int len;
    //构造动态数组
    printf("请输入学生的个数:\n");
    printf("len = ");
    scanf("%d", &len);
    struct Student * p =(struct Student *)malloc(len * sizeof(struct Student));

    int i;
    for(i=0; i<len; i++){
        printf("请输入第%d个学生的信息\n", i+1);
        printf("age=");
        scanf("%d", &p[i].age);
        printf("score=");
        scanf("%f", &p[i].score);
        printf("name=");
        scanf("%s", p[i].name);
    }
    //输出(数组输出一定用for循环)
    int j;
    for(j=0; j<len; j++){
        printf("这是第%d个学生的信息:", j+1);
        printf("age=%d, score=%f, name=%s\n", p[j].age, p[j].score, p[j].name);
    }
    //排序 按学生成绩降序
    int k, l;
    struct Student t;//因为目的是将学生排序,所以临时变量的数据类型应该是结构体数据类型
    for(k=0; k<len-1; k++){
        for(l=0; l<len-1-k; l++){
            if(p[l].score < p[l+1].score){//比较的是结构体成绩这个成员 //>升序 <降序
                //互换的是整体,将结构体的整体根据成绩排序
                t = p[l];
                p[l] = p[l+1];
                p[l+1] = t;
            }
        }
    }
    //按成绩降序输出学生的信息
    //在排序中已经将学生信息根据成绩排序了
    printf("按成绩顺序输出:\n");
    for(l=0; l<len; l++){
        printf("age=%d, score=%f, name=%s\n",p[l].age, p[l].score, p[l].name);
    }
    return 0;
}
```