C基础补习

产地：安徽阜阳
姓名：张鹏鹏
所带课程：数据结构、C++、QT
联系方式：
QQ：820018279
V信：13141518517

一、动态内存分配

1.1 静态内存分配

目前为止，你所学习的定义变量都是静态内存分配

```
int a;   int arr[5];    char *p;     double t;
```

1.2 动态内存分配

可以由程序员自己手动在适当的时候，向内存申请自己想要的内存大小的空间，供用户使用

```
man malloc
#include <stdlib.h>           ---> 需要引入的头文件
void *malloc(size_t size);
功能：允许程序员动态地在堆区申请内存空间
参数：
    size:申请空间的大小，以字节为单位
返回值：成功返回申请出来空间的地址
        失败：返回NULL

void free(void *ptr);
功能：释放从堆区申请的空间
参数：
    ptr：要释放的空间的首地址
返回值：无

```

1.3 单个内存空间申请和释放

```c
#include<stdio.h>
#include<stdlib.h>      //#include<malloc.h>
int main(int argc, const char *argv[])
{
    //1、申请空间
    int *p =(int *)malloc(sizeof(int));          //申请一个int类型空间大小
    //使用指针之前要进行判断是否申请成功
    if(NULL == p)
    {
        printf("申请失败\n");
        return -1;
    }

    //使用空间
    *p = 520;

    printf("*p = %d\n", *p);

    return 0;
}

```

1.4 连续内存空间的申请和释放

```c
#include<stdio.h>
#include<stdlib.h>
int main(int argc, const char *argv[])
{
    int n = 10;

    char *p = (char *)malloc( n * sizeof(char));       //申请两个char类型空间的大小
                                                       //并且申请的空间是连续的

    *(p+0) = 'G';          //  *(p+0)    *(p+1)

    *(p+1) = 'K';          //p[1] = 'K';

    //printf("p[0] = %c, p[1] = %c\n", p[0], p[1]);
    for(int i=0; i<n; i++)
```

```
    {
        printf("%d\t", p[i]);
    }
    printf("\n");

    //释放空间
    free(p);
    p = NULL;


    return 0;
}

```

练习：提示用户输入n个学生的个数，使用动态内存分配完成申请空间，然后输入n个学生的成绩，经由升序排序后，输出

```
头文件：
#ifndef __STU_H__
#define __STU_H__

//声明空间申请函数
int *space(int size);

//声明输入函数
void input(int *p, int size);

//排序函数
void sort(int *p, int size);

//输出函数
void output(int *p, int size);

//释放
void my_free(int *p);

#endif


源文件
#include<stdio.h>
#include"stu.h"
#include<stdlib.h>

//空间申请函数
```

```c
int *space(int size)
{
    //判断逻辑
    if(size <= 0)
    {
        printf("所给空间不合法\n");
        return NULL;
    }

    //申请空间
    int *p = (int *)malloc(size*sizeof(int));
    if(NULL == p)
    {
        printf("空间申请失败\n");
        return NULL;
    }

    printf("空间申请成功\n");
    return p;

}

//输入函数
void input(int *p, int size)
{
    for(int i=0; i<size; i++)
    {
        printf("请输入第%d个学生的成绩: ", i+1);
        scanf("%d", &p[i]);
    }
}

//排序函数
void sort(int *p, int size)
{
    for(int i=1; i<size; i++)    //控制趟数，从1开始
    {
        for(int j=0; j<size-i; j++)  //控制元素及比较次数
        {
            if(p[j] > p[j+1])      //大升小降
            {
                //交换三部曲
                int t=p[j]; p[j]=p[j+1]; p[j+1]=t;
            }
        }
```

```
    }
    printf("排序成功\n");
}

//输出函数
void output(int *p, int size)
{
    printf("排序后的分数分别是：");
    for(int i=0; i<size; i++)
    {
        printf("%d\t", *(p+i));
    }
    printf("\n");
}

//释放
void my_free(int *p)
{
    if(NULL != p)
    {
        free(p);
        p = NULL;
    }
}
```

测试文件

```
#include<stdio.h>
#include"stu.h"
int main(int argc, const char *argv[])
{
    int *p;     //接受空间的
    int n;      //人数

    printf("请输入班级人数:");
    scanf("%d", &n);

    p = space(n);

    input(p, n);          //调研输入函数

    sort(p, n);          //排序函数

    output(p, n);        //输出函数

    my_free(p);           //释放空间
```

```
    p = NULL;


    return 0;
}

```

1.5 malloc函数的总结

1> malloc函数可以在程序运行过程中，动态申请空间
2> malloc函数申请的空间在堆区
3> malloc申请的空间，返回的是void*类型的地址，如果想要访问，要强转为自己想要的类型
4> malloc申请的空间，以字节为单位
5> 使用free函数释放空间时，必须给定申请空间的起始地址，不能给定中间地址
6> 释放完空间，指针为悬空指针，此时应尽快将其设置为NULL
7> 一直运行的程序，如果重复大量调用malloc函数，而不调用free函数，会造成“内存泄漏”
8> 当程序结束后，堆区申请的空间，如果没有手动释放，系统会自动回收，此时，不属于内存泄漏

二、内存划分


```
ubuntu@ubuntu:day1$ vi 04test.c
ubuntu@ubuntu:day1$ cat 04test.c
#include<stdio.h>

int m;              //在静态区的.bss段
int n = 1314;       //在静态区的.data段
char *r = "hello";   //r在.data段    “hello”在.ro段
```

```c
char brr[20] = "world";   //brr在.data段，“world”在.ro段
static int k;             //在.bss段
static int g = 666;           //在.data段
//int x = n;

//char *x = malloc(sizeof(char));    //在堆区申请空间

int main(int argc, const char *argv[])
{
    int a;                  //定义在栈区，没有初始化，默认为随机值
    int b = 520;             //定义在栈区
    char *p = "hello";          //指针p在栈区    “hello”在静态区的.ro段  不能通过指针改变字符串的值
    char *q = malloc(sizeof(char));    //在堆区申请空间，q在栈区
    char arr[10] = "hello";           //"hello"在.ro段，  arr数组在栈区，因为是多个变量，所以数组元素可以被重新赋值
    static int c = 100;              //在静态区的.data段
    static int s;                   //在静态区的.bss段


    return 0;
}
```

三、存储类型

3.1 存储类型的种类

```
auto    static   extern     const       register    volatile

auto:自动类型的变量，目前你现在定义的局部变量都是auto，默认省略不写
    注意：非自动类型：全局变量、static修饰的变量都不能加该关键字

static:
    1、修饰局部变量时，可以延长该变量的生命周期
    2、修饰全局变量或者全局函数，限制该变量或者函数的作用域，表示该变量或者该函数只能在该文件中被引用

extern:
    1、可以引入外部文件的全局变量或者全局函数
    2、可以使用该关键字解决全局变量和局部变量同名时，想要使用全局变量的情况

const: 该关键字修饰的变量作为只读变量，起到保护数据不被更改的作业，但是注意，不能说该关键字定义的时常量
```

```
    1、修饰普通变量时，该变量作为只读变量，不能通过该变量更改该变量内存中的值
    2、const修饰指针变量，放在不同位置，保护的对象不同
        const int *p=&n;        //不能通过指针更改内存中的值，但是可以改变指针的指向
        int const *p = &n;      //不能通过指针更改内存中的值，但是可以改变指针的指向
        int * const p = &n;     //不能改变指针的指向，但是可以通过指针改变内存中的值
        const int * const p=&n;  //既不能改变指针指向，也不能通过指针改变值

register:
    1、修饰变量时，该变量直接定义在寄存器中，大大提高数据的存储和访问速度
    2、该关键字修饰的变量，不能做取地址运算，因为取地址运算是取得内存的地址，而该关键字修饰的变量不在内存中
    3、一个内核中，寄存器的个数是有限的，所以一个程序中，尽可能少的定义寄存器变量

volatile：防止代码优化功能
    1、该关键字修饰的变量，每次cpu去数据时，都会到内存中直接获取
    2、对应硬件寄存器时要使用volatile（ARM）
    3、多线程处理全局变量时，要使用volatile修饰（IO进程线程）
    4、中断访问非自动类型数据（ARM中）
```

auto案例

```c
#include<stdio.h>
//auto int m = 1314;          //报错，因为全局变量不能定义成auto类型
int main(int argc, const char *argv[])
{
     auto static int x = 999;
    auto int n = 520;

    printf("n = %d\n", n);

//  printf("m = %d\n", m);
    printf("x = %d\n", x);

    return 0;
}
```

---

static修饰局部变量

```c
#include<stdio.h>
```

```c

void fun()
{
    int a = 0;                  //该变量是局部变量，每次都是跟随函数空间的开辟而获得内存并初始化
    static int b = 0;           //该语句在编译时已经完成初始化

    printf("a = %d\n", a);
    printf("b = %d\n", b);

    a++;
    b++;
}

int main(int argc, const char *argv[])
{
    fun();          //a=0   b=0;
    fun();          //a=0   b=1;
    fun();          //a=0   b=2;
    //printf("b = %d\n", b);            //报错，因为静态局部变量也是局部变量，其他函数无权访问
    return 0;
}
```

---

static修饰全局变量和全局函数

外部文件：

```c
static int a = 1314;    //该全局变量只能在06fun.c中使用
                         //别的文件无权使用

static void fun()        //该函数只能在06fun.c中使用
{
    printf("hello world\n");
}
```

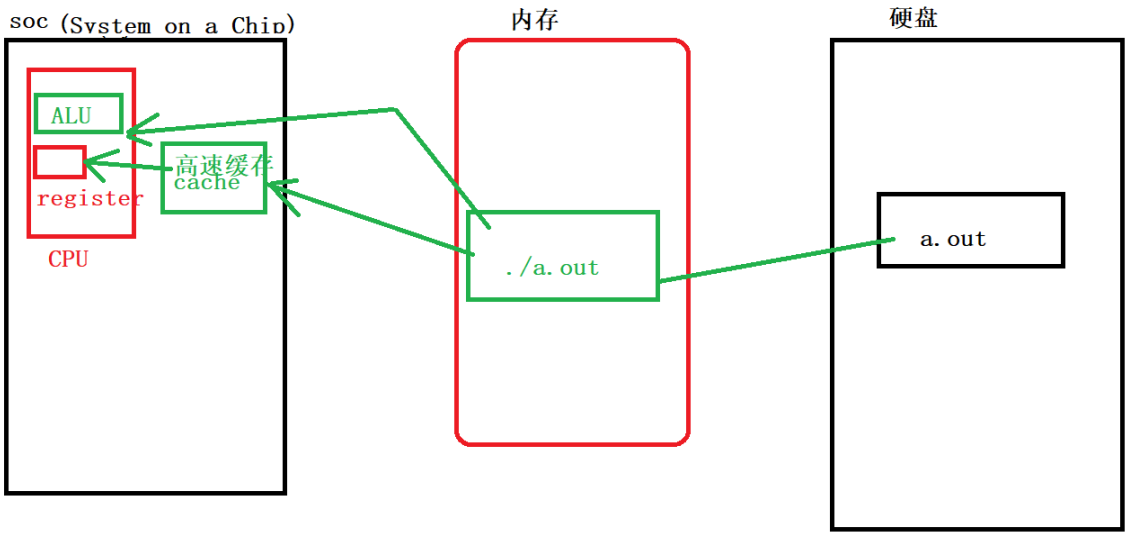
主函数：

```c
#include<stdio.h>

//extern int a;         //声明该变量为外部传过来的变量
//extern void fun();    //引用外部函数时，extern可加可不加

int main(int argc, const char *argv[])
{
    //printf("a = %d\n", a);
```

```
    fun();

    return 0;
}

-------------------------------------------------------------------------------------
extern 该关键字解决全局变量和局部变量同名时，想要使用全局变量的情况
int n = 520;

int main(int argc, const char *argv[])
{
    int n = 1314;

    printf("n = %d\n", n);              //1314

    if(1)
    {
        extern int n;
        printf("n = %d\n", n);          //此时使用的是全局变量的n
    }
    return 0;
}
-------------------------------------------------------------------------------------
 const修饰普通变量
 int main(int argc, const char *argv[])
{
    int n = 520;
    printf("n = %d\n", n);              //可以输出
    n = 1314;                   //可以通过变量名对该变量所在内存空间进行更改
    printf("n = %d\n", n);              //可以输出

    printf("-------------------------------------------------\n");
    const int m = 666;          //const修饰的为只读变量
    printf("m = %d\n", m);          //666
    n = m;                          //也可以
//  m = 999;                //?不可以，因为m是只读变量，不能进行更改

    printf("-----------------------------------------\n");
    int *p = &m;
    *p = 999;
    printf("m = %d\n", m);
```

```
103
104     printf("-----------------------------------------\n");
105     const int k;            //没有初始化的const变量，无意义，是随机值，后期也不允许更改
106     //k = 888;
107     printf("k = %d\n", k);
108     return 0;
109 }
110
111 ---------------------------------------------------------------------------
112 const 修饰指针变量
113  #include<stdio.h>
114 int main(int argc, const char *argv[])
115 {
116     int n = 520;
117     int m = 1314;
118
119     const int *p = &n;    //不能通过指针更改内存中的值，但是可以改变指针指向
120     //*p = m;                 //更改内存中的值
121     p = &m;                 //更改指针指向
122
123
124     int * const q = &n;   //不能改变指针指向，但是可以通过指针改变值
125     *q = m;         //可以通过指针改变内存中的值
126     //q = &m;         //报错，指针指向不能改变
127
128     const int * const r = &n;     //两个都不能改变
129     //*r = m;           //报错
130     //r = &m;             //报错
131
132
133     return 0;
134 }
135
136 ---------------------------------------------------------------------------
137 register关键字
138 #include<stdio.h>
139 int main(int argc, const char *argv[])
140 {
141     register int a = 520;          //该变量为寄存器遍历
142
```

```
    printf("&a = %p\n", &a);          //报错，register修饰的变量，不能取地址

    return 0;
}
```



四、typedef类型重定义

4.1 格式

```
typedef int size_t;
typedef unsigned char uint_8;
typedef unsigned int uint_32;
```

4.2 不同类型的重定义

```
//定义变量
int a;              //定义整形变量
int arr[3];         //定义数组
int *p;             //定义一级指针
int **p;            //定义二级指针
int *arr[3];         //定义指针数组
int (*p)[3];         //定义数组指针
int *func(int , int);    //定义指针函数
int (*funp)(int, int);   //定义函数指针
int (*funp[3])(int, int);    //定义函数指针数组

```

```
//提取类型
int ;
int [3];
int *;
int **;
int *[3];
int (*)[3];
int *(int , int);
int (*)(int, int);
int (*[3])(int, int);

//给类型重新起名：在前面加关键字typedef，将原本写变量名的地方，写该类型的新名字即可
typedef int A;        //A就是int的别名
typedef int ARR[3];
typedef int *P;
typedef int **PP;
typedef int *PARR[3];
typedef int (*ARRP)[3];
typedef int *PFUN(int , int);
typedef int (*FUNP)(int, int);
typedef int (*FUNPARR[3])(int, int);

```

```
typedef与define的区别
#include<stdio.h>

typedef int*  B;          //此时A就是int类型

#define A int*

int main(int argc, const char *argv[])
{
    A m,n;              //int *m, n     --->m是指针，n是整形变量
    B t,k;              //int *t; int *k;      --->两个都是指针

    int g;

    m = &g;          //不报错
    n = &g;          //会报错
    t = &g;          //不报错
    k = &g;          //不报错

    return 0;
}
```

```
22
23  ____________________________________________________________________
    ____________________________________________
24  typedef定义函数指针类型
25  #include<stdio.h>
26
27  typedef void (*FUNP)(int, int);     //给函数指针重新类型定义
28
29  void fun(int m, int n)
30  {
31      printf("%d\n", m+n);
32  }
33
34  int main(int argc, const char *argv[])
35  {
36      FUNP p1 = fun;      //void (*p)(int, int) = fun;
37
38      p1(520, 1314);    //fun(520, 1314)
39      return 0;
40  }
41
42  ____________________________________________________________________
    ____________________________________
43  数组重定义
44  #include<stdio.h>
45
46  typedef int ARR[3];
47
48  int main(int argc, const char *argv[])
49  {
50      ARR n;        //int n[3];
51
52      n[0] = 520;
53      n[1] = 1314;
54      n[2] = 999;
55
56      //n = 100;      //此时n是数组名
57
58      return 0;
59  }
60
61
```

4.3 define与typedef的区别

1> define只是简单的字符串的替换，不做计算，也不做正确性检查，发生在预处理阶段

2> typedef是类型的替换，发生在编译阶段

3> define是一个预处理指令，以#开头，没有分号结束

4> typedef是一条语句，需要分号结尾

5> 最明显的例子是定义两个指针变量时有很大区别

五、结构体

5.1 引入目的

系统提供的数据类型不够开发使用了，此时，没有条件，程序员自己定义一个自定义的数据类型，供自己使用

5.2 定义

由多个相同数据类型或者不同数据类型构成的数据集合，属于构造数据类型

5.3 定义格式

```
struct 类型名
{
    成员变量1;
    成员变量2;
    ...
    成员变量n;
};

解释：
1> struct:定义结构体的关键字，不能省略
2> 所有的成员变量都是用一对{}包裹起来
3> 成员变量可以是基本数据类型（int、float、double、char、short、long）,也可以是构造数据类型（数组、结构　　体、共同体）
4> 成员中可以封装各种类型的变量名，但是不能封装函数，但是能封装函数指针
5> 这是一条声明语句，花括号后面的分号不能省
6> 成员变量分配的空间是连续的
7> 声明结构体类型时不分配空间，只有拿着该类型定义变量时才分配内存空间
8> 常用的定义变量的格式：
        struct 类型名　变量名列表;
9> 相同结构体变量之间是可以相互赋值的
```

```

```

```c
#include<stdio.h>

struct Stu
{
    int num;          //学号
    char name[20];        //姓名
    double score;      //分数
};

int main(int argc, const char *argv[])
{
    struct Stu s1;           //此时定义了一个学生类型的变量
    //printf("sizeof s1 = %ld\n", sizeof(s1));//32

    return 0;
}
```

5.4 结构体变量的定义

```
定义格式一：
struct 类型名
{
    成员变量1;
    成员变量2;
    ...
    成员变量n;
};
struct 类型名   变量名列表;
例子：
struct Stu
{
    int num;          //学号
    char name[20];        //姓名
    double score;      //分数
};

int main(int argc, const char *argv[])
{
    struct Stu s1;           //此时定义了一个学生类型的变量
    //printf("sizeof s1 = %ld\n", sizeof(s1));//32

    return 0;
}
```

```
-------------------------------------------------------------------
-----------------------------------
定义格式2：
struct 类型名
{
    成员变量1;
    成员变量2;
    ...
    成员变量n;
}变量列表;
例子：
struct Stu
{
    int num;          //学号
    char name[20];         //姓名
    double score;       //分数
}s1,s2,s3;
-------------------------------------------------------------------
-----------------------------------------
定义格式3：
struct
{
    成员变量1;
    成员变量2;
    ...
    成员变量n;
}变量列表;
例子：
struct
{
    int num;          //学号
    char name[20];         //姓名
    double score;       //分数
}s1,s2,s3;
```

5.5 结构体变量的初始化

```
#include<stdio.h>

struct Stu
{
    int num;          //学号
    char name[20];         //姓名
    double score;       //分数
```

```c
}s2 = {1002, "秋雅", 99};            //初始化方式二

int main(int argc, const char *argv[])
{
    struct Stu s1 = {1001, "夏洛", 67};          //初始化方式一


    struct Stu s3 = {.name="元华", .score=98};      //初始化方式三

    return 0;
}

```

5.6 结构体变量引用成员

引入成员运算符 点（.）,个人读作"的"

```c
#include<stdio.h>
#include<string.h>

struct Stu
{
    int num;          //学号
    char name[20];         //姓名
    double score;       //分数
};


int main(int argc, const char *argv[])
{
    struct Stu s1 = {1001, "夏洛", 67};          //初始化方式一

    struct Stu s2;
    s2.num = 1002;          //给变量s2的num成员赋值
    //s2.name = "马冬梅";      //因为字符串赋值不能直接使用等号
    strcpy(s2.name, "马冬梅");
    s2.score = 75.5;

    //输出信息
    printf("s1的信息为：学号：%d,姓名：%s，分数：%lf\n", s1.num, s1.name, s1.score);
    printf("s2的信息为：学号：%d,姓名：%s，分数：%lf\n", s2.num, s2.name, s2.score);

    //定义s3变量
```

```
    struct Stu s3;
    printf("请输入s3的学号：");
    scanf("%d", &s3.num);
    printf("请输入s3的姓名：");
    scanf("%s", s3.name);
    printf("请输入s3的成绩：");
    scanf("%lf", &s3.score);
    printf("s3的信息为：学号：%d,姓名：%s，分数：%lf\n", s3.num, s3.name, s3.score);
    return 0;
}

```

5.7 结构体指针引用成员

```
格式：
    指针名->成员名
    （*指针名）.成员名
-------------------------------------------------------------------------------------
#include<stdio.h>
#include<string.h>

struct Stu
{
    int num;          //学号
    char name[20];          //姓名
    double score;       //分数
};


int main(int argc, const char *argv[])
{
    struct Stu s1 = {1001, "夏洛", 67};          //初始化方式一

    struct Stu s2;
    s2.num = 1002;          //给变量s2的num成员赋值
    //s2.name = "马冬梅";
    strcpy(s2.name, "马冬梅");
    s2.score = 75.5;

    //输出信息
    printf("s1的信息为：学号：%d,姓名：%s，分数：%lf\n", s1.num, s1.name, s1.score);

```

```c
    printf("s2的信息为：学号：%d,姓名：%s，分数：%lf\n", s2.num, s2.name, s2.score);

    //定义s3变量
    struct Stu s3;
    printf("请输入s3的学号：");
    scanf("%d", &s3.num);
    printf("请输入s3的姓名：");
    scanf("%s", s3.name);
    printf("请输入s3的成绩：");
    scanf("%lf", &s3.score);
    printf("s3的信息为：学号：%d,姓名：%s，分数：%lf\n", s3.num, s3.name, s3.score);


printf("**********************************************************\n");
    struct Stu *p1;
    p1 = &s1;            //将指针指向s1

    //结构体指针变量访问成员要用运算符：箭头（->）
    p1->score = 87;
    strcpy(p1->name, "沈腾");
    printf("s1的信息为：学号：%d,姓名：%s，分数：%lf\n", s1.num, s1.name, s1.score);


printf("**********************************************************\n");
    struct Stu *p2 = &s2;        //*p  <==> s2
    (*p2).score = 20;
    printf("s2的信息为：学号：%d,姓名：%s，分数：%lf\n", s2.num, s2.name, s2.score);


printf("**********************************************************\n");
    struct Stu *p3 = (struct Stu *)malloc(sizeof(struct Stu));
 //在堆区申请空间，将地址赋值给p3
    p3->num = 1004;
    strcpy(p3->name, "小明");
    p3->score = 100;
    printf("p3的信息为：学号：%d,姓名：%s， 分数：%lf\n", p3->num, p3->name, p3->score);
    free(p3);       //释放申请的空间
```

```
    p3 = NULL;
}
```

5.8 结构体数组

```
定义格式：struct 结构体名 数组名[常量];
#include<stdio.h>

struct Stu
{
    int num;          //学号
    char name[20];         //姓名
    double score;       //分数
};

int main(int argc, const char *argv[])
{
    struct Stu s[3];           //此时定义了三个学生，s[0]  s[1]  s[2]

    for(int i=0; i<3; i++)       //循环输入三个学生的所有信息
    {
        printf("请输入第%d个学生的学号：", i+1);
        scanf("%d", &s[i].num);
        printf("请输入第%d个学生的姓名：", i+1);
        scanf("%s", s[i].name);
        printf("请输入第%d个学生的成绩：", i+1);
        scanf("%lf", &s[i].score);
    }

    //输出
    printf("学号\t姓名\t\t成绩\n");
    for(int i=0; i<3; i++)
    {
        printf("%d\t%s\t\t%.2lf\n", s[i].num, s[i].name, s[i].score);
    }


printf("*****************************************************\n");
    struct Stu *p[3]= {&s[0], &s[1], &s[2]};         //指针数组
    p[0]->num = 1001;

```

```
    return 0;
}
```

练习：用户提示输入学生个数，调用函数（void input(struct Stu *p， int n)）完成n的学生信息的录入，调用有参无返回值函数void my_max(struct Stu *p， int n)输出分数最高学生的信息， 调用输出函数（void output(struct Stu *p， int n)），将所有学生信息输出

```
头文件
#ifndef __TEST_H__
#define __TEST_H__

#define MAX 20

//定义学生类型
struct Stu
{
    int num;    //学号
    char name[10];    //姓名
    int score;          //成绩
};

//输入
void input(struct Stu *p, int n);

//求最大
void my_max(struct Stu *p, int n);

//输出
void output(struct Stu *p, int n);


#endif
-----------------------------------------------------------------------
-----------------------------------
源文件
#include"test.h"
#include<stdio.h>


//输入
void input(struct Stu *p, int n)
{
    for(int i=0; i<n; i++)
    {
```

```
        printf("请输入第%d个学生的学号：", i+1);
        scanf("%d", &p[i].num);
        printf("请输入第%d个学生的姓名：", i+1);
        scanf("%s", p[i].name);
        printf("请输入第%d个学生的分数：", i+1);
        scanf("%d", &p[i].score);
        printf("\n");
    }
    printf("录入成功\n");
}

//求最大
void my_max(struct Stu *p, int n)
{
    struct Stu max = p[0];          //存储最值
    for(int i=0; i<n; i++)
    {
        if(max.score < p[i].score)
        {
            max = p[i];
        }
    }
    printf("最高分的信息：学号：%d,姓名:%s, 成绩：%d\n", max.num,
max.name, max.score);
}

//输出
void output(struct Stu *p, int n)
{
    printf("学号\t姓名\t\t分数\n");
    for(int i=0; i<n; i++)
    {
        printf("%d\t%s\t\t%d\n", p[i].num, p[i].name,
p[i].score);
    }
}
```

测试文件

```
#include<stdio.h>
#include"test.h"
int main(int argc, const char *argv[])
{
    struct Stu s[MAX];   //定义数组
    int n;          //学生个数
```

```
79
80      printf("请输入学生个数：");
81      scanf("%d", &n);
82
83      input(s, n);     //调用输入函数
84      my_max(s, n);    //调用求最值函数
85      output(s, n);   //调用输出函数
86
87
88      return 0;
89  }
90
```

5.9 结构体的嵌套

```
1   struct A
2   {
3       int xx;
4       double dd;
5   };
6
7   struct B
8   {
9       struct A aa;
10      int bb;
11  };
12
13  //访问数据
14  struct B b;
15  b.bb = 520;                 //访问B类型中的bb成员
16  b.aa;                     //访问B类型中的aa成员，但是该成员是个结构体
17  b.aa.xx;                 //访问B类型中的aa成员中的xx成员
18  b.aa.dd;                  //访问B类型中的aa成员的dd成员
19  ------------------------------------------------------------------
    ----
20  struct B *p = (struct B*)malloc(sizeof(struct B));
21  p->bb;                   //访问B类型中的bb成员,该数据是整形变量
22  p->aa;                   //访问B类型中的aa成员，但是该成员是个结构体
23  p->aa.xx;                 //访问B类型中的aa成员中的xx成员
24  p->aa.bb;           //访问B类型中的aa成员的dd成员
25
26  ____________________________________________________________________
    ____________________________________
27  struct A
```

```
{
    int arr[2];
    double aa;
};

struct B
{
    struct A brr[2];
    int bb;
};
struct B t;           //定义结构体变量
t.bb;               //访问B类型中的bb成员，是个普通变量
t.brr[0];
t.brr[1];           //访问B中的数组，是一个结构体变量
t.brr[0].aa;          //普通变量
t.brr[0].arr[0];       //普通变量
t.brr[0].arr[1];       //普通变量
t.brr[1].aa;          //普通变量
t.brr[1].arr[0];       //普通变量
t.brr[1].arr[1];       //普通变量
-----------------------------------------------------
struct B h[2];        //定义结构体数组  h[0]   h[1]
h[0] .bb;               //访问B类型中的bb成员，是个普通变量
h[0] .brr[0];
h[0] .brr[1];           //访问B中的数组，是一个结构体变量
h[0] .brr[0].aa;          //普通变量
h[0] .brr[0].arr[0];       //普通变量
h[0] .brr[0].arr[1];       //普通变量
h[0] .brr[1].aa;          //普通变量
h[0] .brr[1].arr[0];       //普通变量
h[0] .brr[1].arr[1];       //普通变量
h[1] .bb;               //访问B类型中的bb成员，是个普通变量
h[1] .brr[0];
h[1] .brr[1];           //访问B中的数组，是一个结构体变量
h[1] .brr[0].aa;          //普通变量
h[1] .brr[0].arr[0];       //普通变量
h[1] .brr[0].arr[1];       //普通变量
h[1] .brr[1].aa;          //普通变量
h[1] .brr[1].arr[0];       //普通变量
h[1] .brr[1].arr[1];       //普通变量
-----------------------------------------------------------
struct B * r[2] = {&h[0], &h[1]};         //定义结构体指针数组
r[0] r[1]
r[0]->bb;               //访问B类型中的bb成员，是个普通变量
r[0]->brr[0];
```

```
r[0]->brr[1];           //访问B中的数组，是一个结构体变量
r[0]->brr[0].aa;          //普通变量
r[0]->brr[0].arr[0];       //普通变量
r[0]->brr[0].arr[1];       //普通变量
r[0]->brr[1].aa;          //普通变量
r[0]->brr[1].arr[0];       //普通变量
r[0]->brr[1].arr[1];       //普通变量
r[1]->bb;               //访问B类型中的bb成员，是个普通变量
r[1]->brr[0];
r[1]->brr[1];           //访问B中的数组，是一个结构体变量
r[1]->brr[0].aa;          //普通变量
r[1]->brr[0].arr[0];       //普通变量
r[1]->brr[0].arr[1];       //普通变量
r[1]->brr[1].aa;          //普通变量
r[1]->brr[1].arr[0];       //普通变量
r[1]->brr[1].arr[1];       //普通变量
```

定义一个班级结构体（包括学生的数组，班级人数），学生信息（学号、姓名、成绩），完成对班级人数的输入输出

```
#include<stdio.h>
#define MAX 20              //班级最大容纳量
//定义学生结构体
struct Stu
{
    int num;
    char name[20];
    int score;
};

//定义班级结构体
struct my_class
{
    struct Stu s[MAX];           //学生的数组
    int len;                    //班级现有人数
};
/*************************添加函数*************************/
void add(struct my_class *c, struct Stu k)
{
    //判满操作
    if(c->len >= MAX)
    {
        printf("添加失败\n");
        return ;
    }
```

```c

    //将学生信息放入班级
    c->s[c->len] = k;
    c->len++;
    printf("添加成功\n");
}


/**********************主函数*******************************/
int main(int argc, const char *argv[])
{
    struct my_class class1;             //定义一个班级
    class1.len = 0;                    //初始化一个空班级

    for(;;)
    {
        struct Stu k;
        printf("请输入要录入学生的学号：");
        scanf("%d", &k.num);
        printf("请输入要录入学生的姓名：");
        scanf("%s", k.name);
        printf("请输入要录入学生的成绩：");
        scanf("%d", &k.score);

        add(&class1, k);                    //调用添加函数

        char judge;
        printf("是否继续录入学生信息（N|Y）：");
        getchar();
        scanf("%c", &judge);

        if(judge=='n' || judge=='N')
        {
            break;
        }
    }

    //输出信息
    printf("目前班级学生信息如下：\n");
    printf("学号\t姓名\t\t分数\n");
    for(int i=0; i<class1.len; i++)
    {
        printf("%d\t%s\t\t%d\n", class1.s[i].num,
class1.s[i].name, class1.s[i].score);
    }
```

```
    return 0;
}

```

5.10 与typedef共同使用

```
格式一、
typedef struct 类型名
{
    成员变量1;
    成员变量2;
    ...
    成员变量n;
}自定义类型名;

格式一、
typedef struct
{
    成员变量1;
    成员变量2;
    ...
    成员变量n;
}自定义类型名;
```

```
#include<stdio.h>

typedef struct Stu
{
    int num;
    char name[20];
}STU, *S;           //此时STU是普通结构体类型，S是结构体指针类型


int main(int argc, const char *argv[])
{
    struct Stu s1;
    STU s2;           //也是一个结构体变量

    s1 = s2;

    S s3;             //此时s3是指针类型

    s3 = &s1;

```

```
    return 0;
}
```

5.11 结构体大小问题（非常重要）

```
字节对齐方式：
1、在32位系统下，如果所有成员都是低于4字节，对齐方式一最大的对齐
2、在所有成员已经计算好容量后，最终的容量要是最大的成员的整数倍，超过4字节，一4字节为倍数
3、在64位系统下，以8字节对齐
```

```
#include<stdio.h>

struct A
{
    char aa;
    char ab;
};

struct B           //4
{
    char ba;
    short bb;
};

struct C        //4
{
    char ca;
    char cb;
    short cc;
};

struct D    //6
{
    char da;
    short db;
    char dc;
};

struct E     //8
{
    char ea;
    int eb;
};
```

```
34
35  struct F    //8
36  {
37      char fa;
38      short fb;
39      int fc;
40  };
41
42  struct G       //16
43  {
44      char ga;
45      int fb;
46      short fc;
47  };
48
49  struct H        //
50  {
51      short hq;    //2
52      char hb[7];   //10
53      int hc;        //4
54      short hd;       //2
55  };
56
57  struct I         //?
58  {
59      struct H ia;             //此时，该结构体类型大小已经定型了
60      char ib;
61  };
62
63  struct K
64  {
65      char ka;         //2
66      struct B kb;
67
68  };
69
70  /********************主函数************************/
71  int main(int argc, const char *argv[])
72  {
73      printf("sizeof A = %d\n", sizeof(struct A));        //2
74      printf("sizeof B = %d\n", sizeof(struct B));        //4
75      printf("sizeof C = %d\n", sizeof(struct C));        //4
76      printf("sizeof D = %d\n", sizeof(struct D));        //6
77      printf("sizeof E = %d\n", sizeof(struct E));        //8
78      printf("sizeof F = %d\n", sizeof(struct F));        //8
```

```
    printf("sizeof G = %d\n", sizeof(struct G));        //12
    printf("sizeof H = %d\n", sizeof(struct H));        //20
    printf("sizeof I = %d\n", sizeof(struct I));        //24
    printf("sizeof K = %d\n", sizeof(struct K));        //6

    return 0;
}
```

六、共用体

6.1 定义

多个相同数据类型或不同数据类型共同使用同一片存储空间的构造数据类型

6.2 定义格式

```
union 类型名
{
    成员变量1;
    成员变量2;
    ...
    成员变量n;
};

union 类型名 变量名;

1>  使用方法跟结构体一样
2>  共用体的地址和每个成员的地址都是同一地址
3>  共用体变量不能作为函数参数传递

```

```
#include<stdio.h>

union data
{
    char c;
    int i;
    double d;
};


int main(int argc, const char *argv[])
```

```
{
    union data k;          //此时定义了一个共用体变量
    printf("sizeof k = %ld\n", sizeof(k));

    k.c = 10;
    printf("k.c = %d, k.i = %d, k.d = %.2lf\n", k.c, k.i, k.d);          // 10
    k.i = 520;
    printf("k.c = %d, k.i = %d, k.d = %.2lf\n", k.c, k.i, k.d);          //520
    k.d = 3.14;

    printf("k.c = %d, k.i = %d, k.d = %.2lf\n", k.c, k.i, k.d);          //3.14

    //验证共用体的地址和每个成员的地址都是同一地址
    printf("&k = %p, &k.c = %p, &k.i = %p, &k.d = %p\n", &k, &k.c, &k.i, &k.d);


    return 0;
}
```

```
#include<stdio.h>

struct Stu
{
    int num;
    char name[20];
    char title;          //S表示学生  T表示老师
    union
    {
        double score;          //学生的成绩
        char subj[10];          //老师所带课程
    };
};

struct Class
{
    struct Stu student[20];
    int len;                //目前的容量
};


```

```c

int main(int argc, const char *argv[])
{
    struct my_class class1;             //定义一个班级
    class1.len = 0;                    //初始化一个空班级
    for(;;)
    {
        struct Stu k;
        printf("请输入要录入学生的学号：");
        scanf("%d", &k.num);
        printf("请输入要录入学生的姓名：");
        scanf("%s", k.name);
        printf("请输入要录入学生的身份（T|S）：");
        scanf(" %c", &k.title);
        if(k.title == 'S')
        {
            printf("请输入要录入的分数：");
            scanf("%lf", &k.score);
        }else if(k.title == 'T')
        {
            printf("请输入所带课程：");
            scanf("%s", k.subj);
        }

    //  add(&class1, k);                //调用添加函数

        char judge;
        printf("是否继续录入学生信息（N|Y）：");
        getchar();
        scanf("%c", &judge);

        if(judge=='n' || judge=='N')
        {
            break;
        }
    }

    return 0;
}

```

作业：

完成简单学生管理系统

1> 定义班级类型（存储学生的数组、班级现有人数）、定义学生类型（学号、姓名、分数）
2> 定义一个班级，完成初始化
3> 完成学生信息的添加
4> 定义函数完成输出所有学生信息
5> 定义函数完成输出最高成绩学生的信息
6> 定义函数完成按成绩排序功能
7> 定义函数完成，要求用户输入一个名字，查找是否是该班级的学生，如果是输出“该学生在该班级”，如果不是输出“查　　找失败”

```
头文件
#ifndef __TEST_H__
#define __TEST_H__
#define MAX 20
//定义学生结构体
typedef struct
{
    int num;
    char name[20];
    int score;
}STU;

//定义班级结构体
typedef struct
{
    STU data[MAX];    //学生数据
    int len;        //长度
}Class;

//创建班级
Class *create();

//添加学生信息
void add(Class *p);

//展示学生信息
void display(Class *p);

//排序
void sort(Class *p);

//查找
void search(Class *p);
```

```

#endif
-------------------------------------------------------------------
-------------------------------------
源文件
#include<stdio.h>
#include"test.h"
#include<stdlib.h>
#include<string.h>

//创建班级
Class *create()
{
    Class *p = (Class*)malloc(sizeof(Class));
    if(NULL == p)
    {
        printf("班级创建失败\n");
        return NULL;
    }

    //给班级初始化
    p->len = 0;

    printf("创建成功\n");
    return p;
}

//添加学生信息
void add(Class *p)
{
    STU s;      //要添加的学生

    printf("请输入要添加学生的学号：");
    scanf("%d", &s.num);
    printf("请输入要添加学生的姓名：");
    scanf("%s", s.name);
    printf("请输入要添加学生的分数：");
    scanf("%d", &s.score);

    //将学放入班级中
    if(p->len >= MAX)
    {
        printf("班级已满添加失败\n");
        return ;
    }
```

```
78
79      //添加逻辑
80      p->data[p->len] = s;
81      p->len++;
82      printf("添加成功\n");
83  }
84
85  //展示学生信息
86  void display(Class *p)
87  {
88      printf("学号\t姓名\t\t分数\n");
89      for(int i=0; i<p->len; i++)
90      {
91          printf("%d\t%s\t\t%d\n", p->data[i].num, p-
    >data[i].name, p->data[i].score);
92      }
93  }
94
95
96  //排序
97  void sort(Class *p)
98  {
99      for(int i=1; i<p->len; i++)
100     {
101         for(int j=0; j<p->len-i; j++)
102         {
103             if(p->data[j].score > p->data[j+1].score)
104             {
105                 STU t = p->data[j];
106                 p->data[j] = p->data[j+1];
107                 p->data[j+1] = t;
108             }
109         }
110     }
111     printf("排序成功\n");
112 }
113
114
115 //查找
116 void search(Class *p)
117 {
118     char mz[20];    //要查找的姓名
119
120     printf("请输入您要查找的姓名：");
121     scanf("%s", mz);
```

```
122
123     //开始遍历整个班级
124     for(int i=0; i<p->len; i++)
125     {
126         if(strcmp(mz, p->data[i].name) == 0)
127         {
128             printf("您要查找的人在该班级\n");
129             return ;
130         }
131     }
132
133     printf("查找失败\n");
134 }
135 ------------------------------------------------------------------
    ------------------------------------
```

主函数

```
136 #include"test.h"
137 #include<stdio.h>
138 #include<stdlib.h>
139 int main(int argc, const char *argv[])
140 {
141     int id;
142     Class *p;         //记录班级的指针
143
144     for(;;)
145     {
146         printf("\t\t====学生管理系统=======\n");
147         printf("\t\t功能一、创建班级\n");
148         printf("\t\t功能二、添加学生信息\n");
149         printf("\t\t功能三、展示学生信息\n");
150         printf("\t\t功能四、最高分显示\n");
151         printf("\t\t功能五、排序\n");
152         printf("\t\t功能六、查找\n");
153         printf("\t\t功能0、退出\n");
154
155         //提示用户输入功能编号
156         printf("请输入功能编号：");
157         scanf("%d", &id);
158
159         switch(id)
160         {
161         case 1:
162             {
163                 p = create();
164                 if(NULL == p)
165
```

```
                {
                    return -1;
                }
            }
            break;
        case 2:
            {
                add(p);
            }
            break;
        case 3:
            {
                display(p);
            }
            break;
        case 4:
            {
            }
            break;
        case 5:
            {
                sort(p);
            }
            break;
        case 6:
            {
                search(p);
            }
            break;
        case 0:exit(0);       //退出整个程序进程
        default:printf("您输入的功能有误，请重新输入!\n");
        }
    }
    return 0;
}

```