

## 1、typedef类型重定义

- 1.1 回顾变量的定义
- 1.2 将变量名去除，剩余的就是变量对应的类型
- 1.3 使用typedef重新定义新的数据类型
- 1.4 编写代码，测试typedef定义的类型别名
- 1.5 #define和typedef的区别

## 2、内存区域的划分

## 3、结构体 ---> struct

- 3.1 结构体相关的概念
- 3.2 声明结构体类型的格式
- 3.3 定义结构体类型的普通结构体变量并对结构体中的成员进行初始化
  - 3.3.1 先声明结构体类型，在定义结构体类型的变量同时进行初始化
  - 3.3.2 先声明结构体类型，在定义结构体类型的变量同时进行初始化
  - 3.3.3 先声明结构体类型，在定义结构体类型的变量，后进行初始化
  - 3.3.4 声明结构体类型的同时定义结构体类型的变量同时进行初始化
  - 3.3.5 声明结构体类型的同时定义结构体类型的变量最后进行初始化
  - 3.3.6 相同结构体类型的变量可以使用等号进行赋值操作
- 3.4 对于普通的结构体变量中的成员的访问
- 3.5 以上结构体的练习题
- 3.6 定义结构体指针类型的变量
- 3.7 结构体指针类型变量的初始化
  - 3.7.1 定义结构体指针变量，指向普通的结构体变量
  - 3.7.2 定义结构体指针变量，指向堆区的空间
- 3.8 通过结构体指针变量访问结构体中的成员
- 3.9 定义结构体数组
- 3.10 定义结构体指针数组
- 3.11 定义结构体数组指针
- 3.12 结构体的嵌套
- 3.13 作业：学生成绩管理系统

# 1、typedef类型重定义

## 1.1 回顾变量的定义

```
1  1. 定义基本类型的普通变量
2      int a;
3      short b;
4
5  2. 定义数组类型的变量
6      int arr[2];
7      char name[20];
8      int arr2[2][2];
9
10 3. 定义指针类型的变量
11     int *p;
12     char *p;
13     int **pp;
14
15 4. 定义指针数组和数组指针
16     int *p_arr[2];
```

```

17     int (*arr_p)[2];
18
19 5. 函数指针和函数指针数组
20     int (*func_p)(int a, int b);
21     int (*func_p_arr[2])(int a, int b);

```

## 1.2 将变量名去除，剩余的就是变量对应的类型

```

1 1. 定义基本类型的普通变量--->类型
2     int ;
3     short ;
4
5 2. 定义数组类型的变量--->类型
6     int [2];
7     char [20];
8     int [2][2];
9
10 3. 定义指针类型的变量--->类型
11     int *;
12     char *;
13     int **;
14
15 4. 定义指针数组和数组指针--->类型
16     int *[2];
17     int (*)[2];
18
19 5. 函数指针和函数指针数组
20     int (*)(int a, int b);
21     int (*[2])(int a, int b);

```

## 1.3 使用typedef重新定义新的数据类型

```

1 可以使用typedef关键字对1.2小节的数据类型重新定义一个新的名字。
2 使用typedef定义完成新的类型名之后，可以使用新的类型名定义对应类型的变量。
3
4 将typedef放到定义变量的最前边，此时变量名就可以当成使用typedef
5 定义的新的数据类型，然后使用此类型定义对应的变量。
6
7 1. 定义基本类型的普通变量(常用)
8     typedef int myint32_t;      // myint32_t 等价于 int
9     typedef short myint16_t;    // myint16_t 等价于 short
10
11 2. 定义数组类型的变量(不常用)
12     typedef int arr_t[2];       // arr_t 等价于 int [2]
13     typedef char name_t[20];    // name_t 等价于 char [20]
14     typedef int arr2_t[2][2];   // arr2_t 等价于 int [2][2]
15
16 3. 定义指针类型的变(不常用)
17     typedef int *p_t;           // p_t 等价于 int *
18     typedef char *p_t;         // p_t 等价于 char *
19     typedef int **pp_t;        // pp_t 等价于 int **
20
21 4. 定义指针数组和数组指针(不常用)

```

```

22     typedef int *p_arr_t[2];    // p_arr_t 等价于 int *[2]
23     typedef int (*arr_p_t)[2]; // arr_p_t 等价于 int (*)[2]
24
25 5. 函数指针(常用)和函数指针数组
26     typedef int (*func_p_t)(int a, int b);
27         // func_p_t 等价于 int (*)(int a, int b);
28     typedef int (*func_p_arr_t[2])(int a, int b);
29         // func_p_arr_t 等价于 int (*func_p_arr_t[2])(int a, int b);

```

## 1.4 编写代码，测试typedef定义的类型别名

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4  typedef int    myint32_t;
5  typedef char   name_t[20];
6  typedef int    arr2_t[2][2];
7  typedef int    *p_t;
8  typedef int    (*arr_p_t)[2];
9  typedef int    (*func_p_t)(int a, int b);
10
11 int add_func(int a, int b)
12 {
13     return (a+b);
14 }
15 // int cal_func(int a, int b, int (*func_p)(int a, int b))
16 int cal_func(int a, int b, func_p_t func_p)
17 {
18     return func_p(a, b);
19 }
20 int main(int argc, const char *argv[])
21 {
22     /*your code*/
23     myint32_t a = 100; // 等价于int a = 100
24     printf("a = %d\n", a);
25
26     name_t name;      // 等价于 char name[20]
27     strcpy(name, "zhangsan");
28     printf("name = %s\n", name);
29
30     p_t p = NULL;      // 等价于 int *p = NULL;
31     p = (int *)malloc(sizeof(int));
32     *p = 10086;
33     printf("*p = %d\n", *p);
34     free(p);
35     p = NULL;
36
37     arr2_t arr2 = {1,2,3,4}; // 等价于int arr2[2][2] = {1,2,3,4};
38     arr_p_t arr_p = arr2;    // 等价于 int (*arr_p)[2] = arr2;
39     for (int i = 0; i < 2; i++)
40     {
41         for (int j = 0; j < 2; j++)
42         {
43             printf("%d ", *((arr_p + i) + j));

```

```

44     }
45     putchar('\n');
46 }
47
48 printf("100 + 200 = %d\n", cal_func(100,200,add_func));
49 return 0;
50 }

```

## 1.5 #define和typedef的区别

```

1  #define  myint32_t  int
2  typedef  int  myint32_t;
3
4  区别：
5      1. 宏定义是在预处理阶段进行替换的，
6          typedef定义的类型参与到编译阶段。
7
8      2. 宏定义的结尾不能有分号
9          typedef的结尾必须有分号
10
11     3. 定义新的类型名时，尽量使用typedef不要使用#define
12         #define  p_d  int*
13         typedef  int *p_t;
14
15         p_d  p,q;          // 替换之后的结果为 int *p, q;
16
17         p_t  p,q;          // 编译之后的结果为 int *p, *q;
18

```

## 2、内存区域的划分

- 1 32位系统的内存的划分：32位操作系统可以访问的虚拟地址空间位0-4G。
- 2 目前课上使用%p打印的地址都属于虚拟地址，操作系统为了更加安全，
- 3 在系统之上禁止操作物理地址内存的空间。
- 4
- 5 MMU：内存管理单元，这是一个芯片中的硬件，完成物理地址到虚拟地址的映射。

0xFFFF_FFFF(4G)			
内核空间			
0xC000_0000(3G)			
用户空间	栈区		局部变量在栈区分配空间， 由操作系统自动分配自动回收
	堆区		程序员自己手动分配的地址空间 由程序员手动分配，手动释放。 malloc/free
	静态区	.bss段	未初始化的全局变量或者使用 static修饰的未初始化的全局变量或者局部变量
		.data段	初始化的全局变量或者使用 static修饰的初始化的全局变量或者局部变量
		.rodata段(read only data)	只读的数据存储在此段，不可以被修改。
0x0000_0000 (0G)		.text段	代码段

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4  // 全局变量
5  int a = 100;          // a : .data段
6  int b ;               // b : .bss段
7  const short c = 1000; // rodata
8  static int d = 2000;  // .data段
9  static int e;         // .bss段
10 char name[20] = "zhangsan"; // .data段
11 char sex[5];          // .bss段
12 char *p = "hello";    // p : .data段 "hello" : rodata
13 char *q ;             // .bss段
14
15 void set_init(int i, int j)
16 {
17     // i , j : 栈区
18
19     // 局部变量
20     int a = 100;        // 栈区
21     int b ;             // 栈区
22     const short c = 1000; // 栈区
23     static int d = 2000;  // .data段
24     static int e;        // .bss段
25     char name[20] = "zhangsan"; // 栈区
26     char sex[5];         // 栈区
27     char *p = "hello";   // p : 栈区 "hello" : rodata
28     char *q;             // 栈区
29     q = (char *)malloc(10); // 指针变量p指向堆区空间。
30 }
31
32 int main(int argc, const char *argv[])
33 {
34     q = (char *)malloc(10); // 全局变量q执行堆区空间
35
36     return 0;
37 }

```

## 3、结构体 ---> struct

### 3.1 结构体相关的概念

1. 结构体属于构造类型。
2. 结构体可以将不同的类型的变量封装成一个新的类型，这样的类型称为构造类型。  
结构体中的成员在内存中占用的空间是连续的。

### 3.2 声明结构体类型的格式

```
1 struct 结构体名 {  
2     数据类型 成员变量名1;  
3     数据类型 成员变量名2;  
4     数据类型 成员变量名3;  
5     .....  
6     数据类型 成员变量名n;  
7 };
```

- 1> 结构体中的所有成员在内存中是连续的，每个成员都有自己对应的内存空间。
- 2> 结构体中成员的数据类型可以为，基本类型，指针类型，数组类型，函数指针，数组指针，指针数组，结构体类型，联合体类型，枚举类型。
- 3> 以上只是单纯的声明了一个结构体类型，并没有定义结构体类型的变量。
- 4> “struct 结构体名”可以当成一个结构体类型使用，使用此结构体类型可以定义对应的结构体变量，定义结构体变量就会在内存中分配对应的空间。

### 3.3 定义结构体类型的普通结构体变量并对结构体中的成员进行初始化

1. 定义普通结构体变量的语法格式  
struct 结构体名 结构体变量名;

#### 3.3.1 先声明结构体类型，在定义结构体类型的变量同时进行初始化

```
1 1. 声明结构体类型  
2     struct Complex {  
3         int m_i;    // 实部  
4         int m_j;    // 虚部  
5     };  
6  
7 2. 定义结构体类型的变量并同时进行初始化  
8     struct Complex c1 = {10, 20};
```

#### 3.3.2 先声明结构体类型，在定义结构体类型的变量同时进行初始化

```

1  1. 声明结构体类型
2      struct Complex {
3          int m_i;    // 实部
4          int m_j;    // 虚部
5      };
6
7  2. 定义结构体类型的变量并同时进行初始化
8      struct Complex c1 = {
9          .m_i = 10,
10         .m_j = 20
11     }; // 好处: 可以不按照顺序进行初始化, 或者只对部分成员初始化

```

### 3.3.3 先声明结构体类型, 在定义结构体类型的变量, 后进行初始化

```

1  1. 声明结构体类型
2      struct Complex {
3          int m_i;    // 实部
4          int m_j;    // 虚部
5      };
6
7  2. 定义结构体类型的变量
8      struct Complex c1;
9  3. 后进行初始化(要求对每个成员单独进行初始化)
10     c1.m_i = 10;
11     c1.m_j = 20;

```

### 3.3.4 声明结构体类型的同时定义结构体类型的变量同时进行初始化

```

1  1. 声明结构体类型,同时定义结构体类型的变量, 同时进行初始化
2      struct Complex {
3          int m_i;    // 实部
4          int m_j;    // 虚部
5      }c1={10,20},c2={.m_i = 30, .m_j = 40};
6

```

### 3.3.5 声明结构体类型的同时定义结构体类型的变量最后进行初始化

```

1  1. 声明结构体类型的同时定义结构体类型的变量
2      struct Complex {
3          int m_i;    // 实部
4          int m_j;    // 虚部
5      }c1;
6
7  2. 最后进行初始化
8      c1.m_i = 10;
9      c1.m_j = 20;

```

### 3.3.6 相同结构体类型的变量可以使用等号进行赋值操作

```
1  1. 声明结构体类型
2      struct Complex {
3          int m_i;    // 实部
4          int m_j;    // 虚部
5      };
6  2. 定义结构体类型的变量
7      struct Complex c1 = {10, 20}, c2;
8
9  3. 相同结构体类型的变量进行赋值操作
10     c2 = c1;
```

## 3.4 对于普通的结构体变量中的成员的访问

```
1  普通的结构变量名.成员变量名;
2
3  写到等号的左边就是赋值，写到等号的右边就是读值。
```

## 3.5 以上结构体的练习题

```
1  #include <stdio.h>
2  // 1. 声明结构体类型
3  struct Complex {
4      int m_i;    // 实部
5      int m_j;    // 虚部
6  }c4={1,2}, c5={.m_i=3,.m_j=4},c6;
7
8  // 相同结构体类型变量之间可以进行赋值
9  void print(struct Complex c)
10 {
11     // 访问结构体中的成员
12     printf("%d + %di\n", c.m_i, c.m_j);
13 }
14
15 int main(int argc, const char *argv[])
16 {
17     /*your code*/
18     // 1. 定义结构体变量的同时进行初始化
19     struct Complex c1 = {10 , 20};
20     print(c1);
21
22     // 2. 定义结构体变量的同时进行初始化
23     struct Complex c2 = {
24         .m_j = 40,
25         .m_i = 30
26     };
27     print(c2);
28     // 3. 先定义结构体变量，后进行初始化
29     struct Complex c3;
30     c3.m_i = 50;
31     c3.m_j = 60;
32     print(c3);
```



```

33
34 // 4. 声明结构体类型的同时定义结构变量同时初始化
35 print(c4);
36 print(c5);
37
38 // 5. 声明结构体类型的同时定义结构变量,后进行初始化
39 c6.m_i = 5;
40 c6.m_j = 6;
41 print(c6);
42
43 return 0;
44 }

```

```

1 练习题:
2 声明一个学生的结构体类型, 成员有名字(char name[20]), 性别(char),
3   年龄(unsigned short), 成绩(int)
4
5 使用此结构体类型定义结构体变量, 并进行初始化
6 #include <stdio.h>
7 #include <string.h>
8 #include <stdlib.h>
9 // 1. 声明一个结构体类型
10 struct Student{
11     char name[20];
12     char sex;
13     unsigned short age;
14     int score;
15 };
16
17 void print(struct Student s)
18 {
19     printf("姓名: %s 性别: %c 年龄: %d 成绩: %d\n",
20           s.name, s.sex, s.age, s.score);
21 }
22 int main(int argc, const char *argv[])
23 {
24     /*your code*/
25     // 1. 定义结构体变量的, 同时进行初始化
26     struct Student stu1 = {"zhangsan", 'M', 18, 99};
27     print(stu1);
28
29     // 2. 定义结构体变量的, 同时进行初始化
30     struct Student stu2 = {
31         .name = "lisi",
32         .sex = 'W',
33         .age = 19,
34         .score = 88
35     };
36     print(stu2);
37
38     // 3. 先定义结构体变量, 后进行初始化
39     struct Student stu3;
40     strcpy(stu3.name, "ruhua");
41     stu3.sex = 'M';
42     stu3.age = 19;

```

```

43     stu3.score = 77;
44
45     print(stu3);
46     return 0;
47 }

```

## 3.6 定义结构体指针类型的变量

```

1  1. 声明结构体类型
2      struct Complex {
3          int m_i;
4          int m_j;
5      };
6
7  2. 定义结构体指针变量
8      struct 结构体名 *结构体指针变量名;
9      struct Complex *c_p = NULL;
10
11 3. 结构体指针变量占用的内存空间的大小
12     32位操作系统，结构体指针变量占用4字节空间；
13     64位操作系统，结构体指针变量占用8字节空间；
14
15 4. 结构体指针变量只是对结构体指针变量分配空间，
16     并没有指向对应的空间，及对结构体中的成员没有分配空间。

```

## 3.7 结构体指针类型变量的初始化

### 3.7.1 定义结构体指针变量，指向普通的结构体变量

```

1  struct 结构体名 *结构体指针变量名 = &普通的结构体变量名;
2
3  举例：
4  1. 声明结构体类型
5      struct Complex {
6          int m_i;
7          int m_j;
8      };
9  2. 定义结构体变量
10     struct Complex c1 = {1,2}; // 定义普通的结构体变量
11     struct Complex *p = &c1; // 定义结构指针变量，并指向普通的结构体变量
12

```

### 3.7.2 定义结构体指针变量，指向堆区的空间

```

1  struct 结构体名 *结构体指针变量名 =
2      (struct 结构体名 *)malloc(sizeof(struct 结构体名));
3
4  对结构体指针变量指向的空间的成员进行初始化：结构体指针变量名->成员名 = 初始值；
5

```

```

6  举例：
7  1. 声明结构体类型
8      struct Complex {
9          int m_i;
10         int m_j;
11     };
12 2. 定义结构体指针变量
13     struct Complex *p =
14         (struct Complex *) malloc(sizeof(struct Complex));
15
16 3. 对结构体指针变量指向的空间中的每个成员进行初始化
17     p->m_i = 3;
18     p->m_j = 4;
19

```

```

1  struct Complex *p = {
2      ->m_i =3,
3      ->m_j =4,
4  };      // 错误的,没有这种写法

```

## 3.8 通过结构体指针变量访问结构体中的成员

```

1  结构体指针变量名->成员变量名；
2      // 前提：结构体指针变量必须有指向才可以使用，否则访问的是非法的空间
3
4
5  总结：结构体中成员的访问方式：
6  普通的结构体变量：    普通结构体变量名.成员名；
7
8  结构指针变量：        结构体指针变量名->成员名；

```

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  // 1. 声明结构体类型
6  struct Complex {
7      int m_i;    // 实部
8      int m_j;    // 虚部
9  };
10
11 void print(struct Complex c)
12 {
13     printf("%d + %di\n", c.m_i, c.m_j);
14     // printf("%d + %di\n", (&c)->m_i, (&c)->m_j);
15 }
16 /*
17  * 当结构体类型作为函数的形参时，推荐使用结构体指针类型的形参
18  * 1. 结构体指针变量相对于普通结构体变量占用的栈区空间更少

```

```

19  * 2. 结构体指针变量进行赋值时，比普通结构体变量进行赋值时效率高
20  */
21  void show(struct Complex *p)
22  {
23      printf("%d + %di\n", p->m_i, p->m_j);
24      // printf("%d + %di\n", (*p).m_i, (*p).m_j);
25  }
26
27  int main(int argc, const char *argv[])
28  {
29      /*your code*/
30      // 1. 定义结构体指针变量，指向普通的结构体变量
31      struct Complex *c_p1 = NULL;
32      struct Complex c1 = {10,20};
33      c_p1 = &c1;
34      printf("*c_p1 = ");
35      print(*c_p1);
36      printf("*c_p1 = ");
37      show(c_p1);
38
39      // 2. 定义结构体指针变量，指向堆区的空间
40      struct Complex *c_p2 =
41          (struct Complex *)malloc(sizeof(struct Complex));
42      if (c_p2 == NULL)
43      {
44          printf("malloc failed!\b");
45          return -1;
46      }
47      // 对结构体指针变量指向堆区的空间初始化
48      c_p2->m_i = 40;
49      c_p2->m_j = 50;
50
51      printf("*c_p2 = ");
52      show(c_p2);
53
54      // 释放堆区的空间
55      free(c_p2);
56      c_p2 = NULL;
57
58
59      return 0;
60  }

```

```

1  练习题：
2  声明一个学生的结构体类型，成员有名字(char name[20])，性别(char)，
3  年龄(unsigned short)，成绩(int)
4
5  使用此结构体类型定义结构体指针变量，并进行初始化。
6  #include <stdio.h>
7  #include <string.h>
8  #include <stdlib.h>
9  // 1. 声明一个结构体类型
10 struct Student{
11     char name[20];
12     char sex;

```

```

13     unsigned short age;
14     int score;
15 };
16
17 void print(struct Student *stu_p)
18 {
19     printf("姓名: %s 性别: %c 年龄: %d 成绩: %d\n",
20           stu_p->name, stu_p->sex, stu_p->age, stu_p->score);
21 }
22
23 int main(int argc, const char *argv[])
24 {
25     /*your code*/
26     // 1. 定义结构体指针变量, 指向普通的结构体变量
27     struct Student stu = {
28         .name = "zhangsan",
29         .sex = 'M',
30         .age = 18,
31         .score = 88
32     };
33     struct Student *stu_p1 = &stu;
34     print(stu_p1);
35
36     // 2. 定义结构体指针变量, 指向堆区空间
37     struct Student *stu_p2 =
38         (struct Student *)malloc(sizeof(struct Student));
39     strcpy(stu_p2->name, "lisi");
40     stu_p2->sex = 'W';
41     stu_p2->age = 19;
42     stu_p2->score = 99;
43
44     print(stu_p2);
45
46     free(stu_p2);
47     stu_p2 = NULL;
48
49     return 0;
50 }

```

## 3.9 定义结构体数组

- 1 1. 结构体数组的本质就是一个数组, 数组的每个成员都是一个结构体类型。
- 2
- 3 2. 定义结构体类型的数组
- 4 struct 结构体名 结构体数组名[元素个数];
- 5
- 6 3. 定义结构体数组的初始化
- 7 3.1 定义结构体数组的同时进行初始化
- 8 struct Complex arr[2] = {{10,20},{30,40}};
- 9 | |---> 数组第1个元素
- 10 |---> 数组第0个元素

```

11
12 3.2 定义结构体数组的同时进行初始化
13 struct Complex arr[2] = {
14     [0] = {
15         .m_i = 10,
16         .m_j = 10
17     },
18     [1] = {
19         .m_i = 20,
20         .m_j = 30
21     }
22 };
23
24 3.3 先定义结构体类型的数组，后进行初始化
25 struct Complex arr[2]; // 先得到每个数组的元素，在对元素的成员初始化
26 arr[0].m_i = 10;
27 arr[0].m_j = 20;
28 arr[1].m_i = 30;
29 arr[1].m_j = 40;
30
31 (arr + 0)->m_i = 10;
32 (arr + 0)->m_j = 20;
33 (arr + 1)->m_i = 30;
34 (arr + 1)->m_j = 40;
35
36 4. 结构体数组中的成员的访问
37 for (int i = 0; i < 2; i++)
38 {
39     printf("%d + %di\n", arr[i].m_i, arr[i].m_j);
40     printf("%d + %di\n", (arr + i)->m_i, (arr + i)->m_j);
41     printf("%d + %di\n", (*(arr + i)).m_i, (*(arr + i)).m_j);
42 }

```

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  struct Book {
6      char boot_name[50];
7      char author[20];
8      int id;
9      float price;
10 };
11 void print(struct Book *p, int len)
12 {
13     #if 0
14         for (int i = 0; i < len; i++)
15         {
16             // 指针变量名当成数组名使用
17             printf("书名: %s\t", p[i].boot_name);
18             printf("作者: %s\t", p[i].author);
19             printf("条码: %d\t", p[i].id);
20             printf("售价: %.2f\t", p[i].price);
21             putchar('\n');
22         }

```

```

23 #else
24     for (int i = 0; i < len; i++)
25     {
26         // 指针变量名当成数组名使用
27         printf("书名: %s\t", (p+i)->boot_name);
28         printf("作者: %s\t", (p+i)->author);
29         printf("条码: %d\t", (p+i)->id);
30         printf("售价: %.2f\t", (p+i)->price);
31         putchar('\n');
32     }
33 #endif
34 }
35 int main(int argc, const char *argv[])
36 {
37     /*your code*/
38     // 1. 定义结构体数组, 并对结构数组的每个元素初始化
39     struct Book arr1[2] = {
40         {"C语言程序设计", "谭浩强", 10001, 50.6},
41         {"C++ primer", "王刚", 10002, 100.5}
42     };
43
44     print(arr1, sizeof(arr1)/sizeof(struct Book));
45
46     // 2. 定义结构体数组, 并对结构体的元素的每个成员初始化
47     struct Book arr2[2] = {
48         [0] = {
49             .boot_name = "C语言数据结构",
50             .author = "冯舜玺",
51             .id = 10003,
52             .price = 45.6,
53         },
54         [1] = {
55             .boot_name = "UNIX网络编程",
56             .author = "史蒂文斯",
57             .id = 10004,
58             .price = 78.5,
59         },
60     };
61     print(arr2, sizeof(arr2)/sizeof(struct Book));
62
63     // 先定义结构体变量, 后进行初始化
64     struct Book arr3[2];
65     // 通过终端输入的方式进行初始化
66     for (int i = 0; i < 2; i++)
67     {
68         scanf("%s%s%d%f", arr3[i].boot_name,
69             arr3[i].author,
70             &arr3[i].id,
71             &arr3[i].price);
72     }
73
74     print(arr3, sizeof(arr3)/sizeof(struct Book));
75
76
77     return 0;

```

```

1  练习题：
2  声明一个学生的结构体类型，成员有名字(char name[20])，性别(char)，
3      年龄(unsigned short)，成绩(int)
4
5  使用此结构体类型定义结构体数组，数组有5个成员，
6  采用从终端输入的方式对数组的5个成员分别进行初始化操作。
7
8  使用冒泡排序的方式，按照成绩对结构体数组进行排序。
9  #include <stdio.h>
10 #include <string.h>
11 #include <stdlib.h>
12 // 1. 声明一个结构体类型
13 struct Student{
14     char name[20];
15     char sex;
16     unsigned short age;
17     int score;
18 };
19
20 void print(struct Student *stu_p, int len)
21 {
22     for(int i = 0; i < len; i++)
23     {
24         printf("姓名: %s 性别: %c 年龄: %d 成绩: %d\n",
25             (stu_p+i)->name, (stu_p+i)->sex,
26             (stu_p+i)->age, (stu_p+i)->score);
27     }
28 }
29
30 void init_student(struct Student *stu_p, int len)
31 {
32     for (int i = 0; i < len; i++)
33     {
34         scanf("%s", (stu_p + i)->name);
35         getchar();
36         scanf("%c", &(stu_p + i)->sex);
37         scanf("%hd", &(stu_p+i)->age);
38         scanf("%d", &(stu_p+i)->score);
39     }
40 }
41
42 void bubble_sort(struct Student *stu_p, int len)
43 {
44     for (int i = 0; i < len - 1; i++)
45     {
46         for (int j = 0; j < len - i - 1; j++)
47         {
48             if (stu_p[j].score < stu_p[j+1].score)
49             {
50                 struct Student tmp;
51                 tmp = stu_p[j];
52                 stu_p[j] = stu_p[j+1];
53                 stu_p[j+1] = tmp;

```



```

54     }
55     }
56 }
57 }
58 int main(int argc, const char *argv[])
59 {
60     /*your code*/
61     // 1. 定义结构体数组,
62     struct Student class[5];
63
64     init_student(class, 5);
65
66     printf("排序之前\n");
67     print(class, 5);
68
69     bubble_sort(class, 5);
70
71     printf("排序之后\n");
72     print(class, 5);
73     return 0;
74 }

```

## 3.10 定义结构体指针数组

- 1 1. 结构体指针数组:本质是一个数组, 数组的每个成员都是一个结构体指针类型的地址
- 2
- 3 2. 定义结构体指针数组
- 4 struct 结构体名 \* 结构体指针数组名[元素个数];
- 5
- 6 3. 结构体指针数组的初始化
- 7 struct 结构体名 \* 结构体指针数组名[元素个数] =
- 8 {&普通结构体变量0, &普通结构体变量1, .... }
- 9
- 10 4. 结构体指针数组中每个成员的访问
- 11 结构体指针数组名[下标]->成员变量名;

## 3.11 定义结构体数组指针

- 1 1. 结构体数组指针 :本质是一个指针, 指向的是一个二维数组
- 2
- 3 2. 定义结构体数组指针
- 4 struct 结构体名 (\*结构体数组指针变量名)[列宽];
- 5
- 6 3. 结构体数组指针的初始化
- 7 struct 结构体名 二维数组名[行宽][列宽] = {};
- 8 struct 结构体名 (\*结构体数组指针变量名)[列宽] = 二维数组名;
- 9
- 10 4. 结构体数组指针指向的二维数组的成员的访问
- 11 4.1> 将结构体数组指针当成二维数组名使用
- 12 结构体数组指针变量名[行下标][列下标].成员变量名;

```
13
14 4.2> 采用地址加偏移量的方式
15      (*(结构体数组指针变量名 + 行偏移) + 列偏移)->成员变量名;
16
17      (*(*(结构体数组指针变量名 + 行偏移) + 列偏移)).成员变量名;
```

## 3.12 结构体的嵌套

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4  // 声明结构体类型
5  struct Person {
6      char name[20];
7      char sex;
8      int age;
9  };
10
11 struct Student {
12     struct Person per;
13     float score;
14 };
15
16 void print(struct Student *stu_p)
17 {
18     // 结构类型的变量不管如何嵌套,
19     // 重要掌握当前结构体类型的变量的类型,
20     // 最终就可以缺点使用.还是->访问成员
21     printf("姓名: %s 性别: %c 年龄: %d 成绩: %f\n",
22           stu_p->per.name, stu_p->per.sex,
23           stu_p->per.age, stu_p->score);
24 }
25 void show(struct Student stu)
26 {
27     printf("姓名: %s 性别: %c 年龄: %d 成绩: %f\n",
28           stu.per.name, stu.per.sex,
29           stu.per.age, stu.score);
30 }
31 int main(int argc, const char *argv[])
32 {
33     /*your code*/
34
35     // 1. 定义普通的结构体变量
36     struct Person per = {"zhangsan", 'M', 18};
37     struct Student stu1 = {per, 88};
38
39     print(&stu1);
40
41     // 2. 定义结构体指针变量
42     struct Student *stu_p =
43         (struct Student*) malloc(sizeof(struct Student));
44     strcpy(stu_p->per.name, "lisi");
45     stu_p->per.sex = 'w';
```

```
46     stu_p->per.age = 20;
47     stu_p->score = 99;
48     print(stu_p);
49
50     return 0;
51 }
```

## 3.13 作业：学生成绩管理系统

---

1 | 完成学生成绩管理系统。

1 | 结构体  
2 | 枚举  
3 | 联合体  
4 | Makefile

##