1、typedef类型重定义

1.1 回顾变量的定义

```
1. 定义基本类型的普通变量
    int a;
    short b;

2. 定义数组类型的变量
    int arr[2];
    char name[20];
    int arr2[2][2];

3. 定义指针类型的变量
    int *p;
    char *p;
    int **pp;

4. 定义指针数组和数组指针
    int *p_arr[2];
```

```
    int (*arr_p)[2];

5. 函数指针和函数指针数组
    int (*func_p)(int a, int b);
    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. 定义基本类型的普通变量--->类型
    int ;
    short ;

2. 定义数组类型的变量--->类型
    int [2];
    char [20];
    int [2][2];

3. 定义指针类型的变量--->类型
    int *;
    char *;
    int **;

4. 定义指针数组和数组指针--->类型
    int *[2];
    int (*)[2];

5. 函数指针和函数指针数组
    int (*)(int a, int b);
    int (*[2])(int a, int b);
```

1.3 使用typedef重新定义新的数据类型

```
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。
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。

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
定义的新的数据类型，然后使用此类型定义对应的变量。

1. 定义基本类型的普通变量(常用)
    typedef int myint32_t;      // myint32_t  等价于 int
    typedef short myint16_t;    // myint16_t  等价于 short

2. 定义数组类型的变量(不常用)
    typedef int arr_t[2];       // arr_t 等价于 int [2]
    typedef char name_t[20];    // name_t 等价于  char [20]
    typedef int arr2_t[2][2];   // arr2_t  等价于  int [2][2]

3. 定义指针类型的变(不常用)
    typedef int *p_t;           // p_t  等价于  int *
    typedef char *p_t;          // p_t  等价于  char *
    typedef int **pp_t;         // pp_t  等价于  int **

4. 定义指针数组和数组指针(不常用)
```

```
    typedef int *p_arr_t[2];    // p_arr_t 等价于 int *[2]
    typedef int (*arr_p_t)[2];  // arr_p_t 等价于 int (*)[2]

5. 函数指针(常用)和函数指针数组
    typedef int (*func_p_t)(int a, int b);
        // func_p_t 等价于 int (*)(int a, int b);
    typedef int (*func_p_arr_t[2])(int a, int b);
        // 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

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef  int   myint32_t;
typedef  char  name_t[20];
typedef  int   arr2_t[2][2];
typedef  int   *p_t;
typedef  int   (*arr_p_t)[2];
typedef  int  (*func_p_t)(int a, int b);

int add_func(int a, int b)
{
    return (a+b);
}
// int cal_func(int a, int b, int (*func_p)(int a, int b))
int cal_func(int a, int b, func_p_t func_p)
{
    return func_p(a, b);
}
int main(int argc, const char *argv[])
{
    /*your code*/
    myint32_t  a = 100; // 等价于int a = 100
    printf("a = %d\n" , a);

    name_t  name;      // 等价于 char name[20]
    strcpy(name, "zhangsan");
    printf("name = %s\n", name);

    p_t p = NULL;        // 等价于 int *p = NULL;
    p = (int *)malloc(sizeof(int));
    *p = 10086;
    printf("*p = %d\n", *p);
    free(p);
    p = NULL;

    arr2_t  arr2 = {1,2,3,4};  // 等价于int arr2[2][2] = {1,2,3,4};
    arr_p_t  arr_p = arr2;      // 等价yu int (*arr_p)[2] = arr2;
    for (int i = 0; i < 2;i++)
    {
        for (int j = 0; j < 2; j++)
        {
            printf("%d ", *(*(arr_p + i) + j));
```

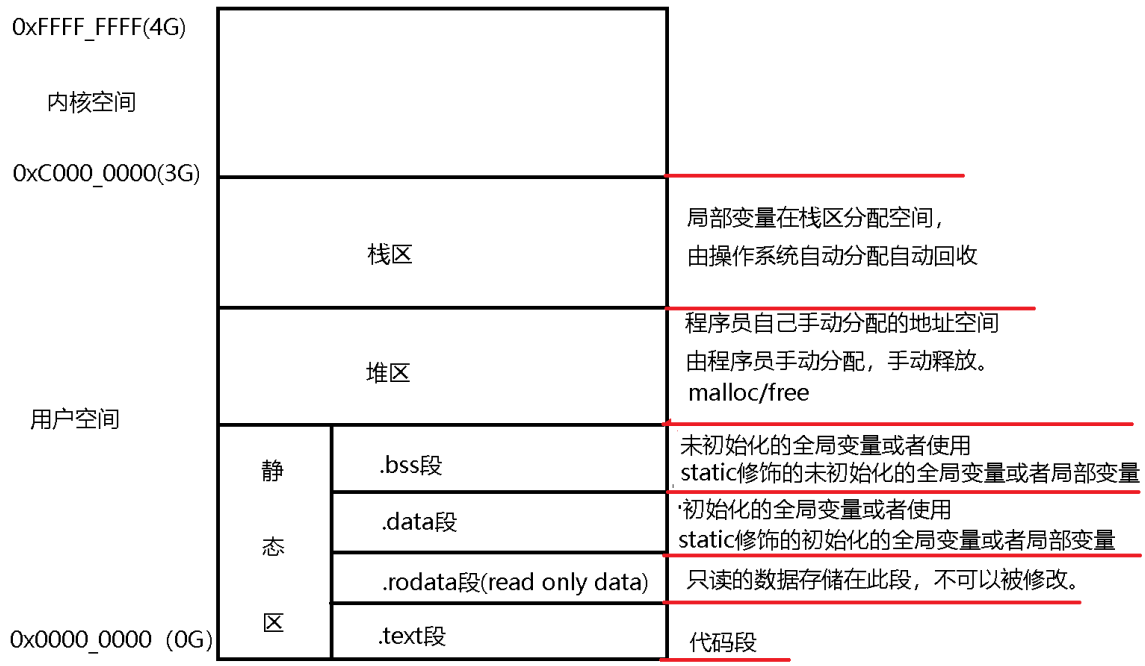
```
        }
        putchar('\n');
    }

    printf("100 + 200 = %d\n", cal_func(100,200,add_func));
    return 0;
}
```

1.5 #define和typedef的区别

```
#define   myint32_t    int
typedef   int    myint32_t;

区别：
    1. 宏定义是在预处理阶段进行替换的，
        typedef定义的类型参与到编译阶段。

    2. 宏定义的结尾不能有分号
        typedef的结尾必须有分号

    3. 定义新的类型名时，尽量使用typedef不要使用#define
        #define   p_d   int*
        typedef   int *p_t;

        p_d  p,q;       // 替换之后的结果为 int *p, q;

        p_t  p,q;       // 编译之后的结果为 int *p, *q;

```

2、内存区域的划分

```
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.
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，
在系统之上禁止操作物理地址内存的空间。

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。
```




```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
// 全局变量
int a = 100;         // a : .data段
int b ;              // b : .bss段
const short c = 1000;    // rodata
static int d = 2000;     // .data段
static int e;            // .bss段
char name[20] = "zhangsan";      // .data段
char sex[5];             // .bss段
char *p = "hello";       // p : .data段  "hello" : rodata
char *q ;   // .bss段

void set_init(int i, int j)
{       // i , j : 栈区

    // 局部变量
    int a = 100;     // 栈区
    int b ;          // 栈区
    const short c = 1000;    // 栈区
    static int d = 2000;     // .data段
    static int e;            // .bss段
    char name[20] = "zhangsan"; // 栈区
    char sex[5];         // 栈区
    char *p = "hello";  // p : 栈区    "hello" : rodata
    char *q;     // 栈区
    q = (char *)malloc(10); // 指针变量p指向堆区空间。
}

int main(int argc, const char *argv[])
{
    q = (char *)malloc(10); // 全局变量q执行堆区空间

    return 0;
}
```

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

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

1> 结构体中的所有的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. 声明结构体类型
    struct Complex {
        int m_i;    // 实部
        int m_j;    // 虚部
    };

2. 定义结构体类型的变量并同时进行初始化
    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. 声明结构体类型
    struct Complex {
        int m_i;    // 实部
        int m_j;    // 虚部
    };

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

2. 定义结构体类型的变量
    struct Complex c1;
3. 后进行初始化(要求对每个成员单独进行初始化)
    c1.m_i = 10;
    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. 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
    struct Complex {
        int m_i;    // 实部
        int m_j;    // 虚部
    }c1={10,20},c2={.m_i = 30, .m_j = 40};

```

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. 声明结构体类型的同时定义结构体类型的变量
    struct Complex {
        int m_i;    // 实部
        int m_j;    // 虚部
    }c1;

2. 最后进行初始化
    c1.m_i = 10;
    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. 声明结构体类型
    struct Complex {
        int m_i;    // 实部
        int m_j;    // 虚部
    };
2. 定义结构体类型的变量
    struct Complex c1 = {10, 20}, c2;

3. 相同结构体类型的变量进行赋值操作
    c2 = c1;
```

3.4 对于普通的结构体变量中的成员的访问

```
普通的结构变量名.成员变量名;

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

```c
#include <stdio.h>
// 1. 声明结构体类型
struct  Complex {
    int m_i;    // 实部
    int m_j;    // 虚部
}c4={1,2}, c5={.m_i=3,.m_j=4},c6;

// 相同结构体类型变量之间可以进行赋值
void print(struct Complex c)
{
                        // 访问结构体中的成员
    printf("%d + %di\n", c.m_i, c.m_j);
}

int main(int argc, const char *argv[])
{
    /*your code*/
    // 1. 定义结构体变量的同时进行初始化
    struct Complex c1 = {10 , 20};
    print(c1);

    // 2. 定义结构体变量的同时进行初始化
    struct Complex c2 = {
        .m_j = 40,
        .m_i = 30
    };
    print(c2);
    // 3. 先定义结构体变量，后进行初始化
    struct Complex c3;
    c3.m_i = 50;
    c3.m_j = 60;
    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;
```

```
    stu3.score = 77;

    print(stu3);
    return 0;
}
```

3.6 定义结构体指针类型的变量

```
1. 声明结构体类型
    struct Complex {
        int m_i;
        int m_j;
    };

2. 定义结构体指针变量
    struct 结构体名  *结构体指针变量名;
    struct Complex *c_p = NULL;

3. 结构体指针变量占用的内存空间的大小
    32位操作系统，结构体指针变量占用4字节空间;
    64位操作系统，结构体指针变量占用8字节空间;

4. 结构体指针变量只是对结构体指针变量分配空间，
    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

```
struct 结构体名  *结构体指针变量名 = &普通的结构体变量名;

举例：
1. 声明结构体类型
    struct Complex {
        int m_i;
        int m_j;
    };
2. 定义结构体变量
    struct Complex  c1 = {1,2}; // 定义普通的结构体变量
    struct Complex  *p = &c1; // 定义结构指针变量，并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

```
struct 结构体名  *结构体指针变量名 =
            (struct 结构体名 *)malloc(sizeof(struct 结构体名));

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;

```

```
举例：
1. 声明结构体类型
    struct Complex {
        int m_i;
        int m_j;
    };
2. 定义结构体指针变量
    struct Complex  *p =
        (struct Complex *) malloc(sizeof(struct Complex));

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
    p->m_i = 3;
    p->m_j = 4;

```

```
struct Complex  *p = {
    ->m_i =3,
    ->m_j =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

```
结构体指针变量名->成员变量名;
    // 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


总结：结构体中成员的访问方式：
普通的结构体变量：    普通结构体变量名.成员名;

结构指针变量：        结构体指针变量名->成员名;
```

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

// 1. 声明结构体类型
struct  Complex {
    int m_i;    // 实部
    int m_j;    // 虚部
};

void print(struct Complex c)
{
    printf("%d + %di\n", c.m_i, c.m_j);
    // printf("%d + %di\n", (&c)->m_i, (&c)->m_j);
}
/*
 * 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
 * 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
```

```
 * 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
*/
void show(struct Complex *p)
{
    printf("%d + %di\n", p->m_i, p->m_j);
    // printf("%d + %di\n", (*p).m_i, (*p).m_j);
}

int main(int argc, const char *argv[])
{
    /*your code*/
    // 1. 定义结构体指针变量，指向普通的结构体变量
    struct Complex *c_p1 = NULL;
    struct Complex c1 = {10,20};
    c_p1 = &c1;
    printf("*c_p1 = ");
    print(*c_p1);
    printf("*c_p1 = ");
    show(c_p1);

    // 2. 定义结构体指针变量，指向堆区的空间
    struct Complex *c_p2 =
        (struct Complex *)malloc(sizeof(struct Complex));
    if (c_p2 == NULL)
    {
        printf("malloc failed!\b");
        return -1;
    }
    // 对结构体指针变量指向堆区的空间初始化
    c_p2->m_i = 40;
    c_p2->m_j = 50;

    printf("*c_p2 = ");
    show(c_p2);

    // 释放堆区的空间
    free(c_p2);
    c_p2 = NULL;


    return 0;
}
```

```
练习题：
声明一个学生的结构体类型，成员有名字(char name[20])，性别(char)，
    年龄(unsigned short),成绩(int)

使用此结构体类型定义结构体指针变量，并进行初始化。
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
// 1. 声明一个结构体类型
struct Student{
    char name[20];
    char sex;
```

```c
    unsigned short age;
    int score;
};

void print(struct Student *stu_p)
{
    printf("姓名：%s 性别：%c 年龄：%d 成绩：%d\n",
        stu_p->name, stu_p->sex, stu_p->age, stu_p->score);
}

int main(int argc, const char *argv[])
{
    /*your code*/
    // 1. 定义结构体指针变量，指向普通的结构体变量
    struct Student stu = {
        .name = "zhangsan",
        .sex = 'M',
        .age = 18,
        .score = 88
    };
    struct Student *stu_p1 = &stu;
    print(stu_p1);

    // 2. 定义结构体指针变量，指向堆区空间
    struct Student *stu_p2 =
            (struct Student *)malloc(sizeof(struct Student));
    strcpy(stu_p2->name, "lisi");
    stu_p2->sex = 'W';
    stu_p2->age = 19;
    stu_p2->score = 99;

    print(stu_p2);

    free(stu_p2);
    stu_p2 = NULL;

    return 0;
}
```

3.9 定义结构体数组

```
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. 定义结构体类型的数组
    struct  结构体名  结构体数组名[元素个数];

3. 定义结构体数组的初始化
    3.1 定义结构体数组的同时进行初始化
    struct Complex arr[2] = {{10,20},{30,40}};
                                |       |---> 数组第1个元素
                                |---> 数组第0个元素
```

```
    3.2 定义结构体数组的同时进行初始化
    struct Complex arr[2] = {
        [0] = {
            .m_i = 10,
            .m_j = 10
        },
        [1] = {
            .m_i = 20,
            .m_j = 30
        }
    };

    3.3 先定义结构体类型的数组，后进行初始化
    struct Complex arr[2];      // 先得到每个数组的元素，在对元素的成员初始化
    arr[0].m_i = 10;
    arr[0].m_j = 20;
    arr[1].m_i = 30;
    arr[1].m_j = 40;

    (arr + 0)->m_i = 10;
    (arr + 0)->m_j = 20;
    (arr + 1)->m_i = 30;
    (arr + 1)->m_j = 40;

4. 结构体数组中的成员的访问
    for (int i = 0; i < 2; i++)
    {
        printf("%d + %di\n",arr[i].m_i, arr[i].m_j);
        printf("%d + %di\n",(arr + i)->m_i, (arr + i)->m_j);
        printf("%d + %di\n",(*(arr + i)).m_i, (*(arr + i)).m_j);
    }
```

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

struct Book {
    char boot_name[50];
    char author[20];
    int  id;
    float  price;
};
void print(struct Book *p, int len)
{
#if 0
    for (int i = 0; i < len; i++)
    {
        // 指针变量名当成数组名使用
        printf("书名：%s\t", p[i].boot_name);
        printf("作者：%s\t", p[i].author);
        printf("条码：%d\t", p[i].id);
        printf("售价：%.2f\t", p[i].price);
        putchar('\n');
    }
```

```c
#else
    for (int i = 0; i < len; i++)
    {
        // 指针变量名当成数组名使用
        printf("书名：%s\t", (p+i)->boot_name);
        printf("作者：%s\t", (p+i)->author);
        printf("条码：%d\t", (p+i)->id);
        printf("售价：%.2f\t", (p+i)->price);
        putchar('\n');
    }
#endif
}
int main(int argc, const char *argv[])
{
    /*your code*/
    // 1. 定义结构体数组，并对结构数组的每个元素初始化
    struct Book arr1[2] = {
        {"C语言程序设计","谭浩强",10001, 50.6},
        {"C++ primer","王刚",10002,100.5}
    };

    print(arr1, sizeof(arr1)/sizeof(struct Book));

    // 2. 定义结构体数组，并对结构体的元素的每个成员初始化
    struct Book arr2[2] = {
        [0] = {
            .boot_name = "C语言数据结构",
            .author = "冯舜玺",
            .id = 10003,
            .price = 45.6,
        },
        [1] = {
            .boot_name = "UNIX网络编程",
            .author = "史蒂文斯",
            .id = 10004,
            .price = 78.5,
        },
    };
    print(arr2, sizeof(arr2)/sizeof(struct Book));

    // 先定义结构体变量，后进行初始化
    struct Book arr3[2];
    // 通过终端输入的方式进行初始化
    for (int i = 0; i < 2; i++)
    {
        scanf("%s%s%d%f", arr3[i].boot_name,
                    arr3[i].author,
                    &arr3[i].id,
                    &arr3[i].price);
    }

    print(arr3, sizeof(arr3)/sizeof(struct Book));


    return 0;
```

```
}
```

```
练习题：
声明一个学生的结构体类型，成员有名字(char name[20])，性别(char)，
    年龄(unsigned short),成绩(int)

使用此结构体类型定义结构体数组，数组有5个成员，
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。

使用冒泡排序的方式，按照成绩对结构体数组进行排序。
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
// 1. 声明一个结构体类型
struct Student{
    char name[20];
    char sex;
    unsigned short age;
    int score;
};

void print(struct Student *stu_p, int len)
{
    for(int i = 0; i < len; i++)
    {
        printf("姓名：%s 性别：%c 年龄：%d 成绩：%d\n",
            (stu_p+i)->name, (stu_p+i)->sex,
            (stu_p+i)->age, (stu_p+i)->score);
    }
}

void init_student(struct Student *stu_p, int len)
{
    for (int i = 0; i < len;i++)
    {
        scanf("%s",(stu_p + i)->name);
        getchar();
        scanf("%c", &(stu_p + i)->sex);
        scanf("%hd", &(stu_p+i)->age);
        scanf("%d", &(stu_p+i)->score);
    }
}

void bubble_sort(struct Student *stu_p, int len)
{
    for (int i = 0; i < len - 1; i++)
    {
        for (int j = 0; j < len - i - 1; j++)
        {
            if (stu_p[j].score < stu_p[j+1].score)
            {
                struct Student tmp;
                tmp = stu_p[j];
                stu_p[j] = stu_p[j+1];
                stu_p[j+1] = tmp;
```

```
            }
        }
    }
}
int main(int argc, const char *argv[])
{
    /*your code*/
    // 1. 定义结构体数组,
    struct Student  class[5];

    init_student(class, 5);

    printf("排序之前\n");
    print(class, 5);

    bubble_sort(class, 5);

    printf("排序之后\n");
    print(class, 5);
    return 0;
}
```

3.10 定义结构体指针数组

```
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. 定义结构体指针数组
    struct 结构体名 * 结构体指针数组名[元素个数];

3. 结构体指针数组的初始化
    struct 结构体名 * 结构体指针数组名[元素个数] =
            {&普通结构体变量0, &普通结构体变量1, .... }

4. 结构体指针数组中每个成员的访问
    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. 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. 定义结构体数组指针
    struct 结构体名 (*结构体数组指针变量名)[列宽];

3. 结构体数组指针的初始化
    struct 结构体名 二维数组名[行宽][列宽] = {};
    struct 结构体名 (*结构体数组指针变量名)[列宽] = 二维数组名;

4. 结构体数组指针指向的二维数组的成员的访问
    4.1> 将结构体数组指针当成二维数组名使用
        结构体数组指针变量名[行下标][列下标].成员变量名;
```

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

        (*(*(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

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
// 声明结构体类型
struct Person {
    char name[20];
    char sex;
    int age;
};

struct Student {
    struct Person per;
    float score;
};

void print(struct Student *stu_p)
{
    // 结构类型的变量不管如何嵌套，
    // 重要掌握当前结构体类型的变量的类型，
    // 最终就可以缺点使用.还是->访问成员
    printf("姓名：%s 性别：%c 年龄：%d 成绩：%f\n",
        stu_p->per.name, stu_p->per.sex,
        stu_p->per.age, stu_p->score);
}
void show(struct Student stu)
{
    printf("姓名：%s 性别：%c 年龄：%d 成绩：%f\n",
        stu.per.name, stu.per.sex,
        stu.per.age, stu.score);
}
int main(int argc, const char *argv[])
{
    /*your code*/

    // 1. 定义普通的结构体变量
    struct Person per = {"zhangsan",'M', 18};
    struct Student stu1 = {per, 88};

    print(&stu1);

    // 2. 定义结构体指针变量
    struct Student *stu_p =
                (struct Student*) malloc(sizeof(struct Student));
    strcpy(stu_p->per.name, "lisi");
    stu_p->per.sex = 'W';
```

```
    stu_p->per.age = 20;
    stu_p->score = 99;
    print(stu_p);

    return 0;
}
```

3.13 作业：学生成绩管理系统

```
完成学生成绩管理系统。
```

```
结构体
枚举
联合体
Makefile
```

##