

概念:

C语言提供了很多的基础数据类型，但是实际的生活/开发中，单纯一个基础数据类型，int float double 不足以描述某一个事务。比如描述一本书：名字、作者、定价、版本....，因此需要一个综合的类型用来描述它。

我们作为开发者可以使用C语言预定提供的一些基础数据类型来自信组合成为一个新的数据类型（结构体）。在使用自己定义的数据类型之前需要**先声明该类型**的模样：

结构体的声明：

```
1 struct 结构体标签
2 {
3     成员1 ;
4     成员2 ;
5     成员3 ;
6     .....
7 };
```

注意：

- struct：关键字，用来告诉编译器我要声明的东西是一个自定义的类型*（结构体）
- 结构体标签：用来区分不同的结构体类型
- 成员：指结构体内部由哪些基础数据类型所构成，可以是任何的数据类型

如何定义与初始化：

```
1 // 定义 与 初始化    实际为结构体分配内存空间  栈
2     struct TieZhu  c = { "Guide of Programming in the Linux Envirnmnt" ,
3                           3.59 , "林世霖" } ;
4
5 // 指定成员初始化
6 struct TieZhu  c = {
7     .Price = 3.59 ,
8     .Name = "林世霖",
9     .Book = "Guide of Programming in the Linux Envirnmnt" ,
10    .num = 1234654
11    } ;
```


注意：

- 普通成员初始化：写起来方便一点，但是用起来麻烦一点，不利于代码的更新于迭代。

- 指定成员初始化：写起来麻烦一点，但是用起来比较轻松，有利于后期代码的更新迭代，不会造成成员与成员之间初始化错位的问题。

```
struct TieZhu c
```

```
struct TieZhu
{
    char * Book ;
    float Price ;
    char Name [32] ;
};
```



1. 向系统申请一片内存，名字为 c
2. 确定该内存中存放的是一个结构体，
3. 分别存放的数据类型：char 指针Book + float 变量Price + char 数组Name

成员引用：

结构体相当于一个数据的集合，里面由多个不同类型的数据组合而成，每一个成员都可以通过成员引用符来单独引用

语法：

- 1 结构体变量名.成员 // 普通的结构体变量
- 2 结构体指针变量->成员 // 结构体指针的访问方法

```
1 // 修改结构体变量中某一个成员的值
2 c.Price = 5 ;
3
4 // 如何引用
5 printf("Book:%s \nPrice:%f \nName:%s\n" ,
6       c.Book, c.Price , c.Name);
```

对结构体的堆内存进行赋值需要注意：

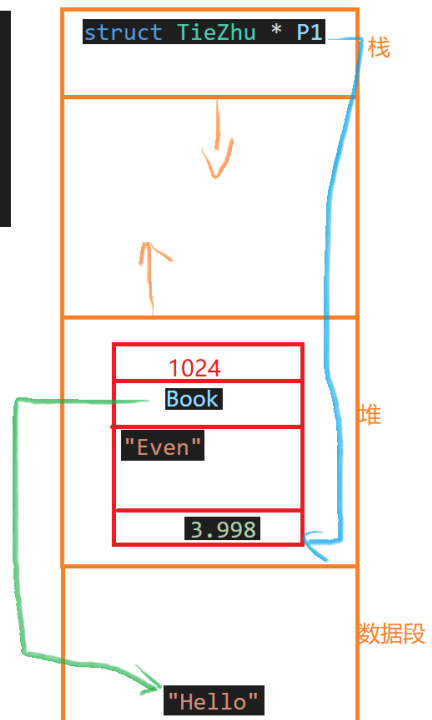
只是一个单纯的指针
不可以直接把数据
拷贝过来，需要而外申请空间

```
struct TieZhu
{
    char * Book ;
    float Price ;
    char Name [32] ;
};
```

可以直接拷贝数据到
该内存中

```
struct TieZhu * P1 = calloc(1, sizeof(struct TieZhu));

P1->num = 1024 ;
P1->Book = "Hello" ;
strncpy(P1->Name , "Even" , 32 );
P1->Price = 3.998 ;
```



由上图可知如果Book指向的是另外一个堆空间，那么在释放结构体的内存时，需要先释放Book 所指向的内存区。

总结：

```
1 // 结构体声明    -->    不占用内存空间
2 struct TieZhu
3 {
4     int num ;
```

```

5     char * Book ;
6     float Price ;
7     char Name [32] ;
8 };
9
10 void main()
11 {
12     // 结构体变量 定义与初始化
13     struct TieZhu a = {123 , "Book" , 3.154 , "Even"} ;
14
15     // 结构体指针
16     struct TieZhu * p = calloc(1, sizeof(struct TieZhu));
17
18     // 结构体数组
19     int arr [10] ;
20     struct TieZhu arr [10] ;
21
22     //成员引用
23     a.num = 245 ;
24     p->num = 250 ;
25     arr[0].num = 399 ;
26 }

```

结构体声明的变异:

```

1 // 结构体声明    -->    不占用内存空间
2 struct TieZhu
3 {
4     int num ;
5     char * Book ;
6     float Price ;
7     char Name [32] ;
8 };

```

变异 1 :

在声明结构体类型时，顺便定义了变量

```

1 struct TieZhu
2 {
3     int num ;

```

```

4     char * Book ;
5     float Price ;
6     char Name [32] ;
7 } Even , *Jacy ; // 在声明结构体类型时， 顺便定义了两个变量
8
9
10 Even.num = 1024 ;
11 Jayc = &Even ; // 让指针Jacy 指向Even的地址
12 Jacy->Book = "Hello" ;

```

变异2:

省略结构体的标签名，但是注意一般在省略标签的情况下需要在声明语句后面顺便定义变量/指针，否则后面无法在定义这种类型的结构体变量。

这种写法比较少出现，如果要出现一般是作为某个结构体内部的成员使用。

```

1 struct
2 {
3     int num ;
4     char * Book ;
5     float Price ;
6     char Name [32] ;
7 } Even , *Jacy ; // 在声明结构体类型时， 顺便定义了两个变量
8
9
10 Even.num = 1024 ;
11 Jayc = &Even ; // 让指针Jacy 指向Even的地址
12 Jacy->Book = "Hello" ;
13

```

例子:

```

1 struct node
2 {
3     int i ;
4     char c ;
5     float f ;
6     struct{ // 结构体内部的成员
7         float ff ;
8         double d ;
9     } info ;
10 };

```

```

11
12 int main(int argc, char const *argv[])
13 {
14     struct node data = {
15         .c = 'A',
16         .i = 1024 ,
17         .f = 3.489 ,
18         .info.d = 665.1234,
19         .info.ff = 3.444
20     };
21
22     struct node * p = &data ;
23
24
25     printf("d:%lf , ff:%lf \n" , data.info.d , data.info.ff );
26
27     printf("d:%lf , ff:%lf \n" , p->info.d , p->info.ff );
28
29
30     return 0;
31 }
32

```

变异 3 ：【推荐使用】

使用 typedef 来给结构体的类型取别名

```
1 typedef long unsigned int size_t ;
```

Tz 相当于 struct TieZhu 可以直接定义普通结构体变量

P_Tz 相当于 struct TieZhu * 可以直接定义结构体指针变量

```

1 typedef struct TieZhu
2 {
3     int num ;
4     char * Book ;
5     float Price ;
6     char Name [32] ;
7 }Tz , * P_Tz ;

```

例子：

```

1
2 // 结构体声明    -->    不占用内存空间
3 typedef struct TieZhu
4 {
5     char * Book ;
6     float Price ;
7     char Name [32] ;
8 }Tz , *P_Tz ;
9
10
11
12 int main(int argc, char const *argv[])
13 {
14
15     Tz b = {
16         .Book = "Hello" ,
17         .Name = "Even",
18         .Price = 3.14
19     };
20
21     P_Tz p = &b ;
22
23     printf("Name : %s \n" , b.Name );
24     printf("Name : %s \n" , p->Name );
25
26
27
28
29     return 0;
30 }

```

```

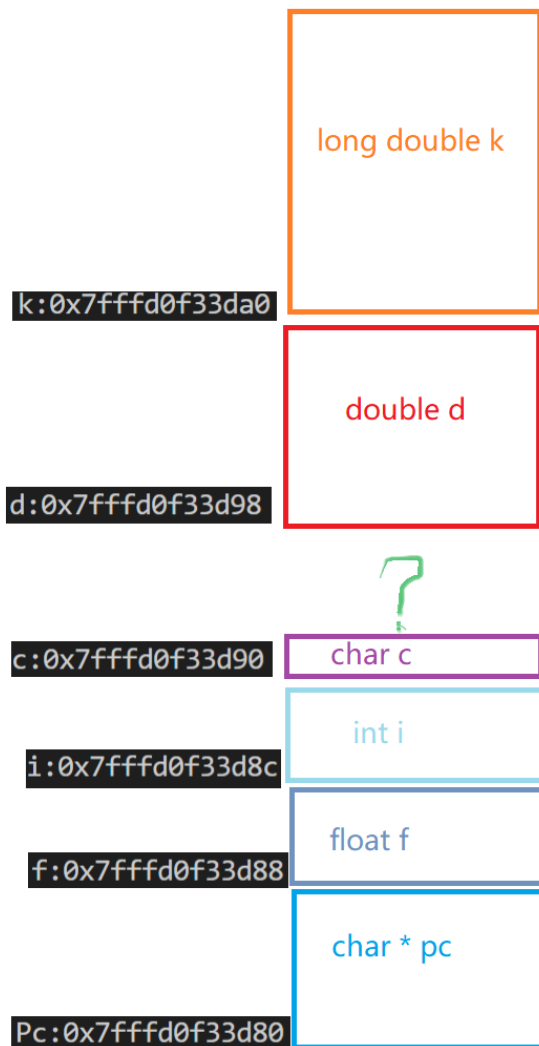
1 结构体数组:
2 struct TieZhu arr[5] ; // 拥有5个结构体的元素的数组
3
4 结构体数组指针:
5 struct TieZhu (*p) [5] ; // 指向的是一个 拥有5个元素的结构体数组
6
7 结构体指针数组:
8 struct TieZhu * arr[5]; // 存放的是5个结构体类型的指针

```

预知结构体的内存：

```
typedef struct TieZhu
{
    char * Pc ;
    float f ;
    int i ;
    char c ;
    double d ;
    long double k ;
}Tz , *P_Tz ;
```

```
Pc:0x7fffd0f33d80
f:0x7fffd0f33d88
i:0x7fffd0f33d8c
c:0x7fffd0f33d90
d:0x7fffd0f33d98
k:0x7fffd0f33da0
```



周末练习：

马保国招生系统：

1. 信息登记---> 添加学员 （结构体的 数组）
 - a. 姓名
 - b. 性别
 - c. 年龄
 - d. 期望技能
 - e. 编号

2. 显示学员信息

- a. 遍历打印结构体数组中每一个元素的每一个成员

3. 修改信息

- a. 通过编号来选择需要修改的学员
- b. 选择需要修改的内容
- c. 用户重新输入新数据
- d. 选择保存或取消

4. 删除学员

- a. 通过编号来选择需要删除的学员
- b. 移动数组后面的内容（要求下一次遍历不会出现即可）