一. 结构的基本概念

1. 结构：结构和Pascal中的record很像。都可以用同一个变量来表示多种不同类型的数据。
2. 成员（字段）：代表结构中不同类型的变量。
3. 结构标记：相当于一种新的类型，要与struct关键字一起使用。
4. 结构声明：相当于声明一种新的类型。
5. 定义结构变量：用上述的新类型定义一个变量。

二. 结构的使用

（其实结构很像Python中的字典）

1.结构的声明

```
struct book {
 char title[MAX];
 float a;
};
```

(注意，结构在花括号后面的那个**分号必不可少**)

2.结构变量的定义

普通的定义：

```
struct book library;
```

一个比较方便的定义，当**结构只被使用一次**的时候：

这个充分地说明了不同结构的名字其实是多余的（逃

```
struct {
 char title[MAX];
 int a;
 } book;
```

3.结构变量的初始化

一般的初始化：(注意不同的常量之间**使用逗号分隔)**

```
struct book library = {
 "I love you",
 2
};
```

还可以使用**初始化器：**

```
struct book library = {
 .title = "I love you",
 .a = 666
```

```
}
```

(可见，格式是".*(成员) = 字面量*",注意使用**逗号分隔**)

4.结构变量的赋值

结构类型可以当作是**普通的变量**来使用，也就是说，他们**可以相互赋值。**

使用复合字面量（这个专门被用来**向函数传递结构字面量**）

```
library = (struct book){"I love you", 666};
```

5.匿名结构（C11）

这个概念有点类似于JavaScript中的匿名函数，专门用于结构的嵌套之中，如下所示：

```
struct book {
 char * title;
 struct {char first[20]; char last[20]};
 // 这里只用一个struct和花括号表示了一个结构
}

struct book library = {"I love you", {"aa", "bb"}};
puts(library.first);
// 可见在引用的时候，我们可以把匿名结构的成员当作整个结构的一部分
```

三. 结构和指针与函数的关系

①用普通的句点表示法： library.title

②用指针的表示方法:

```
struct book *p;
p = &library; // 要注意结构变量不同于数组，他的名字不代表他的地址
puts(p->title);
```

2.结构与函数

①用结构变量本身来与函数交互

```
struct book foo (struct book);
// 这里定义了一个返回值和参数都是book结构的函数

struct book foo(struct book a) {
  ...
}
```

②使用指向相关结构的函数

```
struct book foo(struct book *)
// 这里传了一个结构的指针类型

a = foo(&library);
```

```
// 再次强调，要传入的是结构的地址

struct book foo(struct book *p) {
  ...
}
```

关于指针的复习：给指针赋值一般都只有两种：要么指向一个量，要么用malloc分配内存
然而这里最好不要使用malloc分配，我们一般都是只有在成员是指针的时候给这个成员单独分配一个地址。

（注意事项：这里的book模板必须要在**函数声明之前**定义！）

3.高级技巧（伸缩数组成员）

什么是伸缩数组？先讲讲伸缩数组的特性吧：

①该数组不会立刻存在（其实本质就是指针）②可以将其变成我们需要多少元素，他就会有多少个元素一样（只是给人的感觉，其实还是要分配足够的内存）

```
struct flex {
 int count;
 double a[];
};

struct flex *p;
// 声明指向结构的指针（好吧，打脸！），并为其分配足够的内存（与普通指针的不同）
p = malloc(sizeof(struct flex) + 5 * sizeof(double));
printf("%d\n", p->a[2]);
 // 现在我们就可以像上述内容一样使用对应的数组了
```

二. 联合

使用关键字union

联合的使用方法和struct一模一样，但是也要注意的是，联合**一次只能存储一个值**，这与结构不同，如下所示：

```
fit.digit = 23; // 把23这个数字存储在fit
fit.bigfl = 2.0; // 把2.0这个数组存储在fit，这样上面的digit就不能使用了
fit.letter = 'h'; // 清除2.0，存储h
```

联合比结构的一个好处就是在必要的条件下省空间（其实没什么卵用）

三. 枚举类型

1. 声明枚举模板: **enum spectrum{red, orange, yellow, green, blue, violet};**
2. 声明枚举变量：**enum spectrum color;**

3. 枚举类型的特点：

①可以看到，我们上面用到的red, orange等都不是字符串，而是一个**标号（本质上是int类型的值）**，以上我们使用的都是默认值，在上面的例子中，red==0，orange==1, yellow==2;当然你可以自己指定。

②有了枚举类型的声明以后，我们可以在下文中直接使用标号——换句话说：你可以使用red, orange等标号来表示0，1了；

4. 枚举类型的使用：（毫无疑问枚举类型最适合**迭代**了）

```
for (color = red; color <= violet; color++) {...};
```

四. 函数指针

1. 函数指针的使用方法：*返回值类型*（**指针名*）（*参数列表*） // 可见和函数声明很像

2. 函数指针的赋值：在C中，函数名就是函数的首地址（为什么函数会占用地址？因为函数中的代码都要**加载进内存中**）所以就有了这样的示范：

```
void ToUpper(char *);
void (*p) (char *);

p = ToUpper;
```

3. 函数指针的使用：有两种使用方法：

①p（'a'）；　　// 直接当作函数的另一种写法来使用，毕竟p = ToUpper

②(*p)('a');　　// 通过解引用获得函数的地址来使用

这两种充满矛盾的用法实际上是由**函数名就是函数首地址的矛盾所决定（同数组）**的

五. 其他

1.namespace:

说白了namespace定义了使用变量名称的规则——即变量名称不得冲突。

具体一点的规则是：①不同作用域的同名变量不冲突。

②相同作用域的同名变量冲突。

③在C中，还有特殊的一条：标记（struct后面的标识符，区别于结构变量）和变量名不冲突（这里特别强调是因为在C++中冲突）

2.typedef

typedef用来给类型名创建别名：如 typedef unsigned char BYTE;

使用时注意事项：

1. 区别于define，那个是交给预处理器处理，而typedef是交给编译器处理

2. 区别于define，如果使用typedef声明指针，那么后面的一系列变量都将是指针。而define则不是，这个最好由你自己试试声明多个指针变量区别。

3. 在使用typedef的时候，后面的别名应该使用大写（当然小写也没问题，问题是大家都是用大写的）

4. 使用typedef快速声明struct变量

```
typedef struct {double x; double y;} rect;
// 这样我们就可以用rect代替我们那个struct了
```

3.再提指针的声明

在这里给大家补一个关于**优先级**的技巧。那就是【】的优先级高于*

所以，如果*和【】并存（指没有括号的情况下），声明的就是一个数组，里面的数据类型就是指针

下面我们来看下例子：

int *a[5][5]　　// 由于【】的优先级高，可见我们声明了一个5*5的指向int的指针数组

int (*a)[5][5]　　// 这里小括号是我们先算里面的内容，可见我们这里只声明了一个指针，这个指针指向一个5*5的数组

int (*a[5])[5]　　// 按照上面的规则，不难看出，这里声明了5个指针数组，每一个指针都指向5个int大小的数组

还有大家要区别函数指针和函数声明的差别：

char * fump (int);　　// 这里是一个函数声明

char * (*p)(int);　　// 这里是一个函数指针的定义

特别的，有指向函数的指针数组

char (*p[3])(int);