第十一章-结构体和共用体

定义和使用结构体

定义结构体类型

定义一个结构体一般的格式为：

```
struct [结构体名]{
成员列表；
}
```

可以把结构体看成是一个可以存放多种数据类型的数组。

成员列表中的每一个成员都必须做类型的说明。

举例：

```
struct student{
    int num;
    char name[20];
    char sex;
    float score;
};
```

定义结构体类型的变量

（1） 先定义结构体，在定义结构体变量

```
//先定义结构体
struct student{
    int num;
    char name[20];
    char sex;
    float score;
};
// 然后定义结构体变量
struct student stu1, stu2;
```

（2） 在定义结构体的同时定义结构体变量

```
struct student{
    int num;
    char name[20];
    char sex;
    float score;
}stu1, styu2;
```

（3） 直接定义结构体变量

```
struct{
    int num;
    char name[20];
    char sex;
    float score;
}stu1, stu2;
```

结构体变量的初始化和引用

```
struct student{
    int num;
    char name[20];
    char sex;
    float score;
}a = {101, "klelee", 'M', 78.5};
```

在程序中使用结构体变量有两种方法：

（1） 将结构体变量作为一个整体来使用

（2） 引用结构体变量中的成员

> 其一般形式是： `结构体变量.结构体成员`

使用结构体数组

引言：一个结构体只能存放一个对象的数据，但是如果定义一个结构体类型的数组就

可以存放多个对象了

定义结构体数组

```
struct student{
    int num;
    char name[20];
    char sex;
    float score;
}stu[5];
```

这里定义了一个长度为五的结构体数组，其中每一个元素都是一个单独的结构体，但是每一个结构体拥有的成员类型和数量都是相同的。

结构体指针

指向结构体变量的指针

根据标题可以看出需要两个东西：

（1） 结构体变量

（2） 结构体类型的指针变量

```
struct student{
    int num;
    char name[20];
    char sex;
    float score;
}stu = {101, "klelee", 'M', 78.5};
struct student *p_stu = &stu;    //定义一个结构体类型的指针变量，并给其赋值一个
```

赋值是把结构体变量的首地址赋予该指针变量，并不是把结构体所有赋值给他，指针和数据有区别的。

通过指针引用结构体成员

有两种引用结构体成员的方式：

（1） `(*结构体指针变量).成员名`

```
(*p_stu).num
```

(2) `结构体指针变量 -> 成员名

```
P_stu -> num
```

以上两种方式都可以对成员num进行访问

用指针处理链表

链表的定义

使用不连续的存储空间存储数据的一种存储形式，其每一个节点分为两部分，第一部分称为数据域，用来存储数据，另外一部分称为指针域，用来存放下一个数据的指针。整个链表的第0个节点称为头节点，这个节点只有指针域，存放着第一个节点的指针；链表的最后一个节点称为表尾，表尾的指针域中存放null即空地址。

可以用结构体来表示每一个节点：

```
struct student{
    int num;
    float score;
    struct stu *next;
};
```

上面程序段中，前两个成员作为数据，存入数据域，第三个成员作为下一个元素的指针存入指针域。

建立简单的静态链表

例11-7 建立一个简单链表，由三个学生数据的节点组成，输出节点中的数据。

```
# include "stdio.h"
// 定义结构体类型
struct student{
    int num;
    float score;
```

```
    float score;
    struct student *next;
};

int main(void){
    struct student a, b, c, *head, *p;
    a.num = 101;a.score = 89.0;
    b.num = 102;b.score = 90.0;
    c.num = 103;c.score = 91.0;

    head = &a;
    a.next = &b;
    b.next = &c;
    c.next = NULL;
    p = head;

    do{
        printf("num:%5d\tscore:%6.2f\n", p -> num, p -> score);
        p = p -> next;
    }while (p! = NULL);
}
```

在本例中，所有结点都是在程序中定义的，不是临时开辟的，也不能用完之后释放，这种链表叫做“静态链表”。

建立动态链表

所谓建立动态链表就是在程序运行的过程中，从无到有的建立一个动态链表，就是一个一个的建立结点和输入各结点的数据，并建立起前后相连接的关系

C语言提供了相应的内存管理函数，集成在stdib.h 和 malloc.h头文件中，使用时应当预处理这两个头文件。

1. 分配内存空间函数malloc
   `void *malloc(unsignded size);`
   功能：在内存的动态存储区中分配一块长度为“size”字节长度的连续区域。

```
int *p;
p = (int *)malloc(20 * sizeof(int));
```

以上语句的含义是，通过malloc函数分配能够存放20个整形数的连续内存空间，并将

该内存空间的首地址赋予指针变量p。

```
struct student{
    int num;
    float score;
    struct student *next;
};
struct student *stu;
stu = (struct student *)malloc(sizeof(struct student));
```

在以上的两段程序中，`(int *) 和 (struct student *)`代表的是整形的指针和student结构体类型的指针，语法是强制类型转换。

1. 分配内存空间函数
   `void *calloc(unsigned n, unsigned size);`
   功能：在内存动态存储区内分配n块长度为“size”字节的连续区域。函数的返回值为该区域的首地址。
   calloc函数和malloc函数的区别在于，calloc函数可以同时创建多块空间。
2. 释放内存空间函数
   `void free(void *p)`
   功能：释放p所指向的一块内存空间，p是任意类型的指针变量，它指向被释放区域得到首地址。被释放的区域应该是由malloc和calloc提供的。

建立一个单向动态链表的步骤如下：

（1） 设三个指针变量：head, p1, p2，用来指向struct student 类型数据。即：

```
struct student *head = NULL, *p1, *p2;
```

(2) 用malloc函数开辟第一个结点，并使head和p2都指向它。通过下面的语句申请一个新结点的空间，再从键盘输入数据。

```
head = p2 = (struct student *)malloc(sizeof(struct student));
// head为头指针变量，指向链表的第一个结点；p2指向链表的表尾
scanf("%d%f", &p2 -> num, &p2 -> score);
```

（3） 再用malloc函数重新开辟另一个结点并使p1指向它，接着输入该结点的数据，并与上一个结点相连，使p2指向新建立的结点

```
p1 = (struct student *)malloc(sizeof(struct student));
scanf("%d%f", &p1 -> num, &p1 -> score);
p2 -> next = p1
p2 = p1
```

（4） 重复执行第三步，依次创建后面的结点，知道所有的结点建立完毕

（5） 将表尾结点的指针的指针域置NULL（p2 -> next = NULL）

输出链表

设一个指针变量p，先指向第一个结点，输出p所指的结点，然后使p后移一个结点，再输出，直到链表的表尾。

举例：编写一个输出链表的函数print

```
void print(struct student *head){
    struct student *p;
    p = head;
    while(p != NULL){
        printf("%d,%6.2f\n", p -> num, p -> score);
        p = p -> next;
    }
}
```

对链表的删除操作

举例：编写函数del以删除动态链表中指定的结点

```
空坑
```

共用体类型

概念：使几个不同类型的变量共同占用一段内存的结构，称为共用体。也可以称为联合体。

其定义形式一般为：

```
union 共用体名{
    成员列表
}变量表列;
共用体变量的定义方式和结构体变量定义形式是一样的。
### 引用共用体变量的形式
`共用体变量名.成员名`
### 共用体数据类型的特点
1. 同一个内存段可以用来存放几种不同类型的成员，但在一瞬间只能存放其中的一种。
2. 共用体变量中起作用的是最后一次存放的成员，在存放的新成员后原有的成员就失去了作
3. 共用体变量的地址和他的成员的地址都是同一地址。
4. 不能对共用体变量名赋值
5. 不能把共用体变量作为函数的参数
6. 共用体类型可以出现在结构体类型的定义中，也可以定义共用体数组。
## 使用枚举类型
如果一个变量只有几种可能的值，则可以将其定义成枚举类型，就是将所有可能的值一一列举
` enum 枚举名 { 枚举值表};`
在枚举值表中应该列出所有的可用值，这些值也称为枚举元素。
举例：
```

enum week{sun, mon,tue,wed,thu,fri,sat};

枚举类型变量

其三种定义枚举类型变量的方式同结构体类型。

枚举类型在使用中有以下规定：

1. 枚举值是常量，不是变量，不能在程序中试图去改变它的值
2. 枚举元素本身由系统定义了一个表示序号的数值。
3. 枚举元素在系统定义时可以改变某个枚举元素对应的数值。所有枚举元素的数值都是由其前面的数加1的来的，但不能回推，例如：
   `enum week{sun = 7, mon, tue = 3, wed, thu, fri, sat};`
   在上面程序中，mon应该等于8，不能由tue回推

用typedef声明新类型名

C语言允许用户自定义类型说明符，使用关键字typedef。举例如下：

```
typedef int INTEGER;
INTEGER a,b;
```

上面的赋值语句就等价于：`int a, b;`

typedef定义的一般形式为：`typedef 原类型名 新类型名;`

其中新类型名一般用全部大写表示。

1. 数组
   typedef char NAME[20]; 表示NAME是字符数组类型，数组长度为20。所以现在NAME就是一种类型，那么他就可以用来定义变量咯！！！
   NAME a1, a2, a3;
   使用NAME定义的a1,a2,a3同样也是字符数组类型，数组长度为20
2. 结构体
   typydef **struct stu{**
   char name[20];
   int age;
   char sex;
   } STU;
   加粗部分表示结构体，STU表示别名，现在STU就是被代替的结构体的类型，可以用它来定义变量
   STU body1, body2,body3;
   定义的三个人同样也拥有name，age，sex三个成员属性
3. 指针
   typedef float PFLOAT;
   PFLOAT p1, p2;
   等价于：
   float p1, *p2;
4. 函数
   typedef char DFCH();
   DFCH af;
   等价于
   char af();