第八章 结构体、共用体、枚举

前面学过一种构造类型——数组：

构造类型：

不是基本类型的数据结构也不是指针，它是若干个相同或不同类型的数据构成的集合

描述一组具有相同类型数据的有序集合，用于处理大量相同类型的数据运算

有时我们需要将不同类型的数据组合成一个有机的整体，

以便于引用。如：

一个学生有学号/姓名/性别/年龄/地址等属性

```
int num;
char name[20];
char sex;
int age;
char addr[30];
```

显然单独定义以上变量比较繁琐，数据不便于管理

1.1 结构体类型的概念及定义

1、概念：

结构体是一种构造类型的数据结构，

是一种或多种基本类型或构造类型的数据的集合。

2、 结构体类型的定义方法

咱们在使用结构体之前必须先有类型，然后用类型定义数据结构

这个类型相当于一个模具

(1).先定义结构体类型，再去定义结构体变量

```
struct   结构体类型名{
        成员列表
};
```

```
例 1 :
struct stu{
        int num;
        char name[20];
        char sex;
};
```

//有了结构体类型后，就可以用类型定义变量了

struct stu　lucy, bob, lilei;//定义了三个 struct stu 类型的变量
每个变量都有三个成员，分别是 num name　sex

(2).在定义结构体类型的时候顺便定义结构体变量，以后还可以定义结构体变量

```
struct 结构体类型名{
    成员列表;
}结构体变量 1,变量 2;

struct 结构体类型名　变量 3，变量 4；
```

例 2：

```
struct stu{
    int num;
    char name[20];
    char sex;
}lucy,bob,lilei;

struct stu xiaohong,xiaoming;
```

3.在定义结构体类型的时候，没有结构体类型名，顺便定义结构体变量，
因为没有类型名，所以以后不能再定义相关类型的数据了

```
struct　{
  成员列表;
}变量 1，变量 2;
```

例 3：

```
struct {
    int num;
    char name[20];
    char sex;
}lucy,bob;
```

以后没法再定义这个结构体类型的数据了，因为没有类型名

4.最常用的方法
通常咱们将一个结构体类型重新起个类型名，用新的类型名替代原先的类型

步骤 1：先用结构体类型定义变量

```
struct stu{
    int num;
    char name[20];
    char sex;
}bob;
```

步骤 2：新的类型名替代变量名

```
struct stu{
    int num;
    char name[20];
    char sex;
}STU;
```

步骤 3：在最前面加 typedef

```
typedef struct stu{
    int num;
    char name[20];
    char sex;
}STU;
```

注意：步骤 1 和步骤 2，在草稿上做的，步骤 3 是程序中咱们想要的代码

以后 STU 就相当于 struct stu
STU lucy;和 struct stu lucy;是等价的。

1.2 结构体变量的定义初始化及使用

1、结构体变量的定义和初始化

结构体变量，是个变量，这个变量是若干个数据的集合

注：

(1):在定义结构体变量之前首先得有结构体类型，然后在定义变量

(2):在定义结构体变量的时候，可以顺便给结构体变量赋初值，被称为结构体的初始化

(3):结构体变量初始化的时候，各个成员顺序初始化

例 4：

```
struct stu{
    int num;
    char name[20];
    char sex;
};
```

```
struct stu boy;
struct stu lucy={101,   "lucy",   'f' };
```

2、结构体变量的使用

定义了结构体变量后，要使用变量

(1).结构体变量成员的引用方法

结构体变量.成员名

例 5：

```
struct stu{
    int num;
    char name[20];
    char sex;
};
struct stu bob;

bob.num=101;//bob 是个结构体变量，但是 bob.num 是个 int 类型的变量

bob.name 是个字符数组，是个字符数组的名字，代表字符数组的地址，是个常量

bob.name ="bob";//是不可行，是个常量

strcpy(bob.name,"bob");
```

例 6：

```
#include <stdio.h>
struct stu{
    int num;
    char name[20];
    int score;
    char *addr;
};
int main(int argc, char *argv[])
{
    struct stu bob;
    printf("%d\n",sizeof(bob));
    printf("%d\n",sizeof(bob.name));
    printf("%d\n",sizeof(bob.addr));
    return 0;
}
```

(2).结构体成员多级引用

例 7：

```
#include <stdio.h>
struct date{
    int year;
    int month;
    int day;
};
struct stu{
    int num;
    char name[20];
    char sex;
    struct date birthday;
};
int main(int argc, char *argv[])
{
    struct stu lilei={101,"lilei",'m'};
    lilei.birthday.year=1986;
    lilei.birthday.month=1;
    lilei.birthday.day=8;
    printf("%d %s %c\n",lilei.num,lilei.name,lilei.sex);
    printf("%d %d %d\n",lilei.birthday.year,lilei.birthday.month,lilei.birthday.day);
    return 0;
}
```

3、相同类型的结构体变量可以相互赋值

注意：必须是相同类型的结构体变量，才能相互赋值。

例 8：

```
#include <stdio.h>
struct stu{
    int num;
    char name[20];
    char sex;
};
int main(int argc, char *argv[])
{
    struct stu bob={101,"bob",'m'};
    struct stu lilei;

    lilei=bob;
```

```
        printf("%d %s %c\n",lilei.num,lilei.name,lilei.sex);
        return 0;
    }
```

1.3 结构体数组

结构体数组是个数组，由若干个相同类型的结构体变量构成的集合

1、结构体数组的定义方法

struct 结构体类型名 数组名[元素个数];

例 9：

```
    struct stu{
        int num;
        char name[20];
        char sex;
    };
    struct stu edu[3];//定义了一个 struct stu 类型的结构体数组 edu，
    这个数组有 3 个元素分别是 edu[0] 、edu[1]、edu[2]
```

1、结构体数组元素的引用 数组名[下标]

2、数组元素的使用

```
    edu[0].num =101;//用 101 给 edu 数组的第 0 个结构体变量的 num 赋值
    strcpy(edu[1].name,"lucy");
```

例 10：

```
#include <stdio.h>
typedef struct student
{
    int num;
    char name[20];
    float score;
}STU;
STU edu[3]={
    {101,"Lucy",78},
    {102,"Bob",59.5},
    {103,"Tom",85}
};
int main()
{
    int i;
```

```
    float sum=0;
    for(i=0;i<3;i++)
    {
        sum+=edu[i].score;
    }
    printf("平均成绩为%f\n",sum/3);
    return 0;
}
```

1.4 结构体指针

即结构体的地址，结构体变量存放内存中，也有起始地址
咱们定义一个变量来存放这个地址，那这个变量就是结构体指针变量。

1、结构体指针变量的定义方法：

struct 结构体类型名 * 结构体指针变量名;

```
struct stu{
    int num;
    char name[20];
};
```

struct stu * p;//定义了一个 struct stu *类型的指针变量
变量名 是 p，p 占 4 个字节，用来保存结构体变量的地址编号

struct stu boy;
p=&boy;
访问结构体变量的成员方法：

例 11：

```
    boy.num=101;//可以，通过 结构体变量名.成员名
    (*p).num=101;//可以，*p 相当于 p 指向的变量 boy
    p->num=101;//可以，指针->成员名
```

通过结构体指针来引用指针指向的结构体的成员，前提是
指针必须先指向一个结构体变量。

1.5 结构体内存分配

1、结构体内存分配

结构体变量大小是，它所有成员之和。

因为结构体变量是所有成员的集合。

例 17：

```
#include<stdio.h>
struct stu{
    int num;
    int age;
}lucy;
int main()
{
    printf("%d\n",sizeof(lucy));//结果为 8
    return 0;
}
```

但是在实际给结构体变量分配内存的时候，是规则的

例 18：

```
#include<stdio.h>
struct stu{
    char sex;
    int age;
}lucy;
int main()
{
    printf("%d\n",sizeof(lucy));//结果为 8？？？
    return 0;
}
```

规则 1：以多少个字节为单位开辟内存

给结构体变量分配内存的时候，会去结构体变量中找基本类型的成员
哪个基本类型的成员占字节数多，就以它大大小为单位开辟内存，
在 gcc 中出现了 double 类型的例外

(1)：成员中只有 char 型数据 ，以 1 字节为单位开辟内存。

(2)：成员中出现了 short 类型数据，没有更大字节数的基本类型数据。
以 2 字节为单位开辟内存

(3)：出现了 int 、float 没有更大字节的基本类型数据的时候以 4 字节为单位开辟内存。

(4)：出现了 double 类型的数据

情况 1：

在 vc 里，以 8 字节为单位开辟内存。

情况 2：

在 gcc 里，以 4 字节为单位开辟内存。

无论是那种环境，double 型变量，占 8 字节。

注意：

如果在结构体中出现了数组，数组可以看成多个变量的集合。

如果出现指针的话，没有占字节数更大的类型的，以 4 字节为单位开辟内存。

在内存中存储结构体成员的时候，按定义的结构体成员的顺序存储。

```
例 19：struct stu{
            char sex;
            int age;
      }lucy;
      lucy 的大小是 4 的倍数。
```

规则 2：字节对齐

(1)：char　　1 字节对齐 ，即存放 char 型的变量，内存单元的编号是 1 的倍数即可。

(2)：short　　2 字节对齐 ，即存放 short int 型的变量，起始内存单元的编号是 2 的倍数即可。

(3)：int　　4 字节对齐 ，即存放 int 型的变量，起始内存单元的编号是 4 的倍数即可

(4)：long　　在 32 位平台下，4 字节对齐 ，即存放 long int 型的变量，起始内存单元的编号是 4 的倍数即可

(5)：float　　4 字节对齐 ，即存放 float 型的变量，起始内存单元的编号是 4 的倍数即可

(6)：double

a.vc 环境下

8 字节对齐，即存放 double 型变量的起始地址，必须是 8 的倍数，double 变量占 8 字节

b.gcc 环境下

4 字节对齐，即存放 double 型变量的起始地址，必须是 4 的倍数，double 变量占 8 字节。

注意 3：当结构体成员中出现数组的时候，可以看成多个变量。

注意 4：开辟内存的时候，从上向下依次按成员在结构体中的位置顺序开辟空间

例 20：//temp 8 个字节

```
#include<stdio.h>
struct stu{
      char a;
      short int b;
      int c;
}temp;
int main()
{
      printf("%d\n",sizeof(temp));
      printf("%p\n",&(temp.a));
      printf("%p\n",&(temp.b));
      printf("%p\n",&(temp.c));
      return 0;
}
```

结果分析：

a 的地址和 b 的地址差 2 个字节

b 的地址和 c 的地址差 2 个字节

例 21：temp 的大小为 12 个字节

```
#include<stdio.h>
struct stu{
    char a;
    int c;
    short int b;
}temp;
int main()
{
    printf("%d\n",sizeof(temp));
    printf("%p\n",&(temp.a));
    printf("%p\n",&(temp.b));
    printf("%p\n",&(temp.c));
    return 0;
}
```

结果分析：

a 和 c 的地址差 4 个字节

c 和 b 的地址差 4 个字节

例 22：

```
struct stu{
    char buf[10];
    int a;
}temp;
//temp 占 16 个字节
```

例 23：

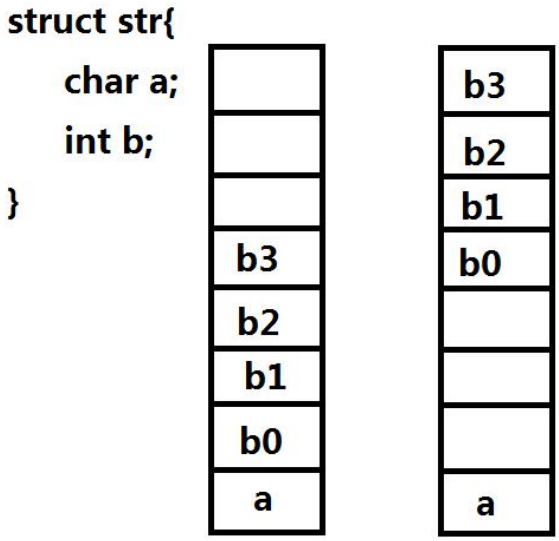
在 vc 中占 16 个字节 a 和 b 的地址差 8 个字节

在 gcc 中占 12 个字节 a 和 b 的地址差 4 个字节

```
#include<stdio.h>
struct stu{
    char a;
    double b;
}temp;
int main()
{
```

```
    printf("%d\n",sizeof(temp));
    printf("%p\n",&(temp.a));
    printf("%p\n",&(temp.b));
    return 0;
}
```

为什么要有字节对齐？
用空间来换时间，提高 cpu 读取数据的效率



指定对齐原则：
使用#pragma pack改变默认对其原则
格式：
#pragma pack (value)时的指定对齐值value。
注意：
1.value只能是：1 2 4 8等
2.指定对齐值与数据类型对齐值相比取较小值
说明：咱们制定一个value

(1)：以多少个字节为单位开辟内存

结构体成员中，占字节数最大的类型长度和value比较，
取较小值，为单位开辟内存

```
例 24：
#pragma pack(2)
    struct stu{
        char a;
        int b;
};
```

以2个字节为单位开辟内存

```
#include<stdio.h>
```

```
#pragma pack(2)
struct stu{
	char a;
	int b;
}temp;
int main()
{
	printf("%d\n",sizeof(temp));
	printf("%p\n",&(temp.a));
	printf("%p\n",&(temp.b));
	return 0;
}
```

temp的大小为6个字节
a和b的地址差2个字节

例 25：

```
#pragma pack(8)
	struct stu{
		char a;
		int b;
	};
```

以4个字节为单位开辟内存

```
#include<stdio.h>
#pragma pack(8)
struct stu{
	char a;
	int b;
}temp;
int main()
{
	printf("%d\n",sizeof(temp));
	printf("%p\n",&(temp.a));
	printf("%p\n",&(temp.b));
	return 0;
}
```

temp的大小为8个字节
a和b的地址差4个字节

(2)：字节对齐

结构体成员中成员的对齐方法，各个默认的对齐字节数和value相比，

取较小值

例 26：

```
#include<stdio.h>
#pragma pack(2)
struct stu{
    char a;
    int b;
}temp;
int main()
{
    printf("%d\n",sizeof(temp));
    printf("%p\n",&(temp.a));
    printf("%p\n",&(temp.b));
    return 0;
}
```

b成员是2字节对齐，a和b的地址差2个字节

例 27：

```
#include<stdio.h>
#pragma pack(8)
struct stu{
    char a;
    int b;
}temp;
int main()
{
    printf("%d\n",sizeof(temp));
    printf("%p\n",&(temp.a));
    printf("%p\n",&(temp.b));
    return 0;
}
```

a和b都按原先的对齐方式存储

如：如果指定对齐值：

设为1：则short、int、float等均为1

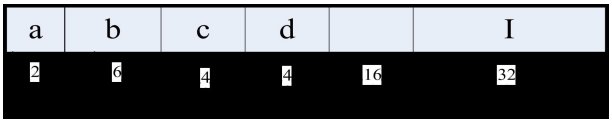
设为 2：则 char 仍为 1，short 为 2，int 变为 2

1.6 位段

一、位段

在结构体中，以位为单位的成员，咱们称之为位段(位域)。

```
struct packed_data{
   unsigned int a:2;
   unsigned int b:6;
   unsigned int c:4;
   unsigned int d:4;
   unsigned int i;
} data;
```



注意：不能对位段成员取地址

例 28：

```
#include<stdio.h>
struct packed_data{
unsigned int a:2;
unsigned int b:6;
unsigned int c:4;
unsigned int d:4;
unsigned int i;
} data;
int main()
{
    printf("%d\n",sizeof(data));
    printf("%p\n",&data);
    printf("%p\n",&(data.i));
    return 0;
}
```

位段注意：

1、对于位段成员的引用如下：

data.a =2

赋值时，不要超出位段定义的范围;

如段成员a定义为2位，最大值为3,即（11）2

所以data.a =5，就会取5的低两位进行赋值 101

2、位段成员的类型必须指定为整形或字符型

3、一个位段必须存放在一个存储单元中，不能跨两个单元

第一个单元空间不能容纳下一个位段，则该空间不用，

而从下一个单元起存放该位段

位段的存储单元：

(1)：char 型位段 存储单元是 1 个字节

(2)：short int 型的位段存储单元是 2 个字节
(3)：int 的位段，存储单元是 4 字节
(4)：long int 的位段，存储单元是 4 字节

```
struct stu{
    char a:7;
    char b:7;
    char c:2;
}temp;//占 3 字节，b 不能跨  存储单元存储
```

例 29：

```
#include<stdio.h>
struct stu{
    char a:7;
    char b:7;
    char c:2;
}temp;
int main()
{
    printf("%d\n",sizeof(temp));
    return 0;
 }
```

结果为：3 ，证明位段不能跨其存储单元存储

注意：不能 取 temp.b 的地址，因为 b 可能不够 1 字节，不能取地址。

4、位段的长度不能大于存储单元的长度
(1)：char 型位段不能大于 8 位
(2)：short int 型位段不能大于 16 位
(3)：int 的位段，位段不能大于 32 位
(4)：long int 的位段，位段不能大于 32 位

例 30：

```
#include<stdio.h>
struct stu{
    char a:9;
    char b:7;
    char c:2;
}temp;
int main()
{
    printf("%d\n",sizeof(temp));
    return 0;
```

```
}
```

分析：

编译出错，位段 a 不能大于其存储单元的大小

5、如一个段要从另一个存储单元开始，可以定义：

```
unsigned    char a:1;
unsigned    char b:2;
unsigned    char :0;
unsigned    char c:3;(另一个单元)
```

由于用了长度为 0 的位段，其作用是使下一个位段从下一个存储单元开始存放

将 a、b 存储在一个存储单元中，c 另存在下一个单元

例：31

```
#include<stdio.h>
struct m_type{
    unsigned char a:1;
    unsigned char b:2;
    unsigned char :0;
    unsigned char c:3;
};
int main()
{
    struct m_type temp;
    printf("%d\n",sizeof(temp));
    return 0;
}
```

6、可以定义无意义位段,如：

```
unsigned a: 1;
unsigned : 2;
unsigned b: 3;
```

例 32：位段的应用

```
struct data{
    char a:1;
    char b:1;
    char c:1;
    char d:1;
    char e:1;
    char f:1;
    char g:1;
```

```
        char h:1;
    }temp;
    int main()
    {
        char p0;
        //p0=0x01;// 0000 0001
        temp.a=1;
        //p0=temp;//错的，类型不匹配
        //p0=(char)temp;//错的，编译器不允许将结构体变量，强制转成基本类型的。

        p0=   *((char *)(&temp));
    }
```

1.7 共用体

1：共用体和结构体类似，也是一种构造类型的数据结构。
既然是构造类型的，咱们得先定义出类型，然后用类型定义变量。
定义共用体类型的方法和结构体非常相似，把 struct 改成 union 就可以了。

在进行某些算法的时候，需要使几种不同类型的变量存到同一段内存单元中，几个变量所使用空间相互重叠

这种几个不同的变量共同占用一段内存的结构，在C语言中，被称作“共用体”类型结构

共用体所有成员占有同一段地址空间

共用体的大小是其占内存长度最大的成员的大小

```
例 33：
typedef   struct   data{
    short int i;
    char ch;
    float f ;
}DATA;
DATA temp1;
```

结构体变量 temp1 最小占 7 个字节（不考虑字节对齐）

```
例 34：
typedef union data{
    short int i;
    char ch;
    float f ;
}DATA;
```

DATA temp2;

共用体 temp2 占 4 个字节，即 i、ch、f 共用 4 个字节

```
#include<stdio.h>
typedef union data{
short int i;
char ch;
float f;
}DATA;
int main()
{
    DATA temp2;
    printf("%d\n",sizeof(temp2));
    printf("%p\n",&temp2);
    printf("%p\n",&(temp2.i));
    printf("%p\n",&(temp2.ch));
    printf("%p\n",&(temp2.f));
    return 0;
}
```

结果：temp2 的大小为 4 个字节，下面几个地址都是相同的，证明了共用体的各个成员占用同一块内存。

共用体的特点：

1、同一内存段可以用来存放几种不同类型的成员，但每一瞬时只有一种起作用

2、共用体变量中起作用的成员是最后一次存放的成员，在存入一个新的成员后原有的成员的值会被覆盖

3、共用体变量的地址和它的各成员的地址都是同一地址

4、共用体变量的初始化 union data a={123}；初始化共用体为第一个成员

例 35：

```
#include<stdio.h>
typedef union data{
    unsigned char a;
    unsigned int b;
}DATA;
int main()
{
    DATA temp;
    temp.b=0xffffffff;
    printf("temp.b = %x\n",temp.b);
    temp.a=0x0d;
    printf("temp.a= %x\n",temp.a);
```

```
    printf("temp.b= %x\n",temp.b);
    return 0;
}
```

结果：

temp.b = ffffffff

temp.a= d

temp.b= ffffff0d

例 36:

扩展一下：

```
    struct type{
        char a:1;
        char b:1;
        char c:1;
        char d:1;
        char e:1;
        char f:1;
        char g:1;
        char h:1;
    };
    union data{
        struct type temp;
        char p0;
    }m;
    int main()
    {
        m.temp.a=1;
    }
```

1.8 枚举

将变量的值一一列举出来，变量的值只限于列举出来的值的范围内

枚举类型也是个构造类型的

既然是构造类型的数据类型，就得先定义类型，再定义变量

1、枚举类型的定义方法

```
enum 枚举类型名{
        枚举值列表;
```

};

在枚举值表中应列出所有可用值,也称为枚举元素

枚举变量仅能取枚举值所列元素

2、枚举变量的定义方法

enum 枚举类型名 枚举变量名;

例 37：

定义枚举类型 week

```
enum week //枚举类型
{
    mon，tue，wed，thu，fri，sat,sun
};
enum week workday,weekday;//枚举变量
```

workday 与 weekday 只能取 sun....sat 中的一个

```
workday = mon; //正确
weekday = tue; //正确
workday = abc; //错误，枚举值中没有 abc
```

① 枚举值是常量,不能在程序中用赋值语句再对它赋值

例如：sun=5; mon=2; sun=mon; 都是错误的.

② 枚举元素本身由系统定义了一个表示序号的数值

默认是从0开始顺序定义为0，1，2…

如在week中，mon值为0，tue值为1， …,sun值为6

③ 可以改变枚举值的默认值：如

```
enum week //枚举类型
{
mon=3，tue，wed，thu，fri=4，sat,sun
};
```

mon=3 tue=4,以此类推

fri=4 以此类推

注意：在定义枚举类型的时候枚举元素可以用等号给它赋值，用来代表元素从几开始编号

在程序中，不能再次对枚举元素赋值，因为枚举元素是常量。