第十章 复合数据类型

10.1 结构体

10.1.1 引入

1、在自然界中任何一个物体，都有多个属性，如果用计算机语言来描述的话，一个属性也许可以用某一个基本数据类型来表示，但是当有多个属性的时候，一个基本数据类型就不能表示了。例如：学生：姓名 性别 年龄 电话号码 家庭住址 。。。

```
char name[32];
char gender;
int age;
char tel[12];
char addr[100];
```

2、如何才能将学生的所有的属性存储到一个数据类型中呢?

```
struct student{
                char name[32];
                char sex;
                int age;
                char tel[12];
                char addr[100];
};
struct：  结构体
struct student：类型名称
定义一个变量：struct  student  s;
```

10.1.2 结构体的定义

1、结构体类型定义：

> struct 结构体名称
> {
> 成员变量1;
> 成员变量2;
> ...
> 成员变量n;
> };

注意：在结构体中不能定义函数

2、结构体变量的定义

> struct 结构体名称 变量名; struct student s;
>
> struct 结构体名称 *变量名; struct student *p;
>
> struct 结构体名称 变量名[长度]; struct student s[10];

不常用的结构体变量的定义：

> struct
> {
> 成员变量1;
> 成员变量2;
> ...
> 成员变量n;
> }结构体变量1，结构体变量2，结构体变量3;
> 举例：
> struct
> {
> int age;
> }A, B, C;
> A，B，C是结构体变量

10.1.3 结构体中成员变量的初始化

> 1、strcut student s = {
> "zhangsan",
> 18,
> 'm'
> };
> 注意：该方法在初始化的时候初始化的值必须和定义结构体的时候成员变量的定义顺序一致！

> strcut student s = {
> .name = "zhangsan",
> .age = 18,
> .sex = 'm',
> .f1 = func;
> };
>
> 注意：成员变量名前面的句点符号

10.1.4 结构体中成员变量的访问

1、.：域操作
前提：结构体变量是个普通变量/数组
struct student s;

s.age = 18;
s.sex = 'f';

2、-> : 指向操作
前提：结构体变量是个指针变量
struct student *p;
p = &s1;

3、如果结构体指针变量p指向了某块内存空间，通过指针p操作指向的内存空间的方法有三种：
方法1：下标法
p[i].成员变量

方法2：指针法， 不常用
(*(p+i)).成员变量

方法3：用过指向操作符
(p+i) == &p[i]
(p+i)->成员变量

```
#include <stdio.h>

struct Stu
{
    int age;
};

int main() {
    struct Stu *s;

    s = (struct Stu *)malloc(3*sizeof(struct Stu));
    s[0].age = 10;

    (s+1)->age = 12;

    (*(s+1)).age = 15;
    return 0;
}
```

> 3、练习：
> 设计一个结构体，存储学生的姓名、性别、年龄
> 定义一个结构体指针变量，存储3个学生的信息，通过该指针对每个结构体中的成员变量行赋值

10.1.5 函数指针在结构体中的应用

> 思考：

> 假如学生的兴趣爱好为球类运动，可以定义多个函数对每一个球类运动进行描述，例如：
> void play_basketball();
> void play_pingpong();
> void play_football();
>
> 思考：在结构体中如何设计一个变量保存该学生的兴趣爱好？
> 函数指针
> 思考：如何让该学生进行该爱好呢？
> 函数指针指向对应的函数，并且通过函数指针调用该函数

```c
typedef void (*FUNC)(char *);
//定义结构体类型
struct student1{
    char name[32];
    char sex;
    int age;
    FUNC

    f; //函数指针
};
void play_basketball(char *name)
{
    printf("%s play basketball\n", name);
}
void play_pingpong(char *name)
{
    printf("%s play pingpong\n", name);
}
```

```c
void play_football(char *name)
{
    printf("%s play football\n", name);
}

//操作结构体中的成员变量为函数指针变量
void test2()
{
    struct student1 s;
    memset(s.name, 0, sizeof(s.name));
    strcpy(s.name, "zhangsan");
    //zhangsan的爱好是 play football
    //将s.f 指向 play_football函数
    s.f = play_football;
    //zhangsan开始踢足球了
    s.f(s.name);
}
```

10.1.6 typedef对结构体类型进行重新定义

1、使用typedef重定义类型

```
 typedef struct 结构体名称
{
        成员变量1;
        成员变量2;
        ...
        成员变量n;
}新类型1, *新类型2;

新类型1 变量名1;
新类型2 变量名2; //变量名2的数据类型是指针类型
```

2、笔试题

> #define struct stu* stu;
> typedef struct stu* p_stu;
>
> stu a, b; //struct stu *a, b;
> p_stu a, b; //struct stu *a, *b;
> 请问这几个a,b之间有何区别，最好用哪种方式(#define/typedef)

10.1.7 结构体中的成员变量为其他结构体变量/指针

1、假设如下场景：用一个结构体描述某一本书，书的属性：名字、价格、作者、出版社

- 定义结构体描述作者

```
struct author{
        char name[32]; //名字
        int age; //年龄
};
```

- 定义结构体描述出版社

```
struct publish{
        char name[32]; //名字
        char addr[32]; //地址
};
```

- 定义结构体描述书

```
struct book{
        char name[32]; //名字
        unsigned int price; //价格
        struct author ath; //作者
        struct publish *publish; //出版社
};
```

2、访问结构体中的结构体成员变量

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

struct author{
    char name[32]; //名字
    int age; //年龄
};

struct publish{
```

```
    char name[32]; //名字
    char addr[32]; //地址
};

struct book{
    char name[32]; //名字
    unsigned int price; //价格
    struct author ath; //作者
    struct publish *pblish; //出版社
};

int main() {
    struct book book;
    struct book book2;
    //初始化作者
    memset(book.ath.name, 0, sizeof(book.ath.name));
    strcpy(book.ath.name, "zhangsan");
    book.ath.age = 30;

    //为指针变量pblish 分配空间
    book.pblish = (struct publish *)malloc(2*sizeof(struct publish)); //书籍有两个出版社
    strcpy(book.pblish[0].name, "hunan");
    strcpy(book.pblish[1].name, "renming");

    return 0;
}
```

10.1.8 结构体的直接赋值

1、通过=将一个结构体中的内容赋值给另外一个结构体

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

struct A
{
    int x;
    int y;
};

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

struct author{
    char name[32]; //名字
    int age; //年龄
};
```

```c
struct publish{
    char name[32]; //名字
    char addr[32]; //地址
};

struct book{
    char name[32]; //名字
    unsigned int price; //价格
    struct author ath; //作者
    struct publish *pblish; //出版社
};


int main() {
    struct A a, b;
    a.x = 100;
    a.y = 1000;

    b = a;
    printf("%d %d\n", b.x, b.y);

    struct book b1, b2;
    memset(b1.name, 0, sizeof(b1.name));
    strcpy(b1.name, "AIoT");
    b1.price = 100;

    //对书的作者进行赋值
    strcpy(b1.ath.name, "lisi");
    b1.ath.age = 30;

    //对出版社进行赋值
    //在堆上分配了一个struct publish大小的空间
```

从上面程序的运行结果来看，一个结构体变量是可以通过=赋值给另外一个结构体变量的，并且两个结构体变量中的内容是一摸一样的。

> **如果结构体中有指针变量，那么导致两个结构体中的指针变量指向了同一块内存空间（两个指针会互相影响）**
>
> 因为book2.pblish和book.pblish指向的是同一块内存空间，其中任意一个变量将pblish指向的空间free释放调，都会影响另外一个结构体变量操作plish，如果执行 free(book.pblish);
>
> printf("%s %s\n", book2.pblish[0].name, book2.pblish[1].name); 这条语句可能会执行失败。

10.2 联合体/共用体

10.2.1 共用体的定义

> union 名称
> {
> 成员变量1;
> 成员变量2;
> ...
> 成员变量n;
> };

10.2.2 共用体的内存布局

共用体的成员变量的存储：所有的成员变量共用一块内存空间
共用体的大小：成员变量中占用内存空间最大的变量的大小

```
union un
{
   char ch[2];
   int x;
};

union un2
{
   char ch[2];
   short x;
};

union un u1;
union un2 u2;
sizeof(u1) == 4
sizeof(u2) == 2
```

10.2.3 共用体的实际使用

1、在实际工作中如果已知一个整型变量(int a)的值需要求出这个变量的每一个字节值的时候可以使用联合体
union un{
char ch[4];
int x;
};
union un u;
只需要将u.x = a; 那么组成a的没一个字节的值就已经保存在u.ch数组中了
反之，如果已知组成a的每一个字节的值，需要求a的值，只需要将u.ch的每一个元素赋值为组成a的每一个字节的值，那么u.x 就是a的值。

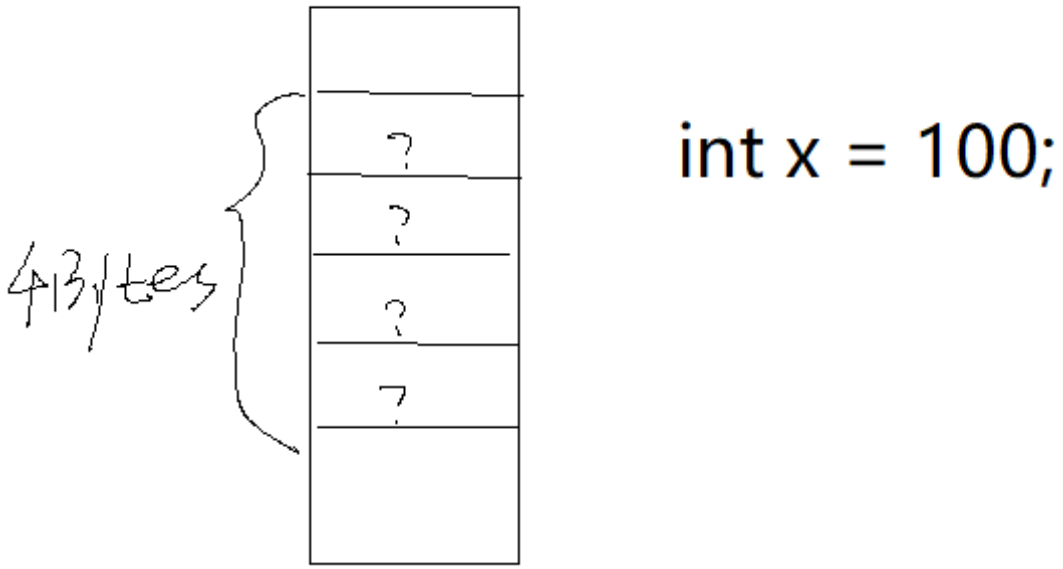
2、传感器GY39，功能可以测量出温度、湿度、大气压强、海拔，假如我们自己开发一个设备，将该设备连接GY39模块，GY39回复的数据格式如下：

0x5a 0x5a 0x01 0x02 0x03 0x02 0x00 0x01 0x02 0x03 0x00 0x02 0x03 0x04 0x5a 0x5a
假设我们定义一个字符数组char ch[16] 用来保存GY39传输的数据
温度：ch[2]<<8 | ch[3]
湿度：ch[4]<<8 | ch[5]
大气压强：ch[6]<<24 | ch[7]<<16 | ch[8]<<8 | ch[9]
海拔：ch[10]<<24 | ch[11]<<16 | ch[12]<<8 | ch[13]
假如使用联合体求温度或者湿度
union un{
char ch[2];
unsigned short x;
};
union un u; u.ch[0] = ch[3], u.ch[1] = ch[2];
u.x 就是温度的值！！

10.2.4 小端模式和大端模式

1、数据在内存中的存储是以二进制的形式存储的，但是如果数据占用的内存空间超过1Byte，我们也可以理解为该数据在内存中是一个字节一个字节的存储。

例如 int x = 100; x占用4个字节，x在内存中被拆分成了4个字节分开存储的。



这四个字节中存储的值分别是多少呢？我们可以通过共用体获取。

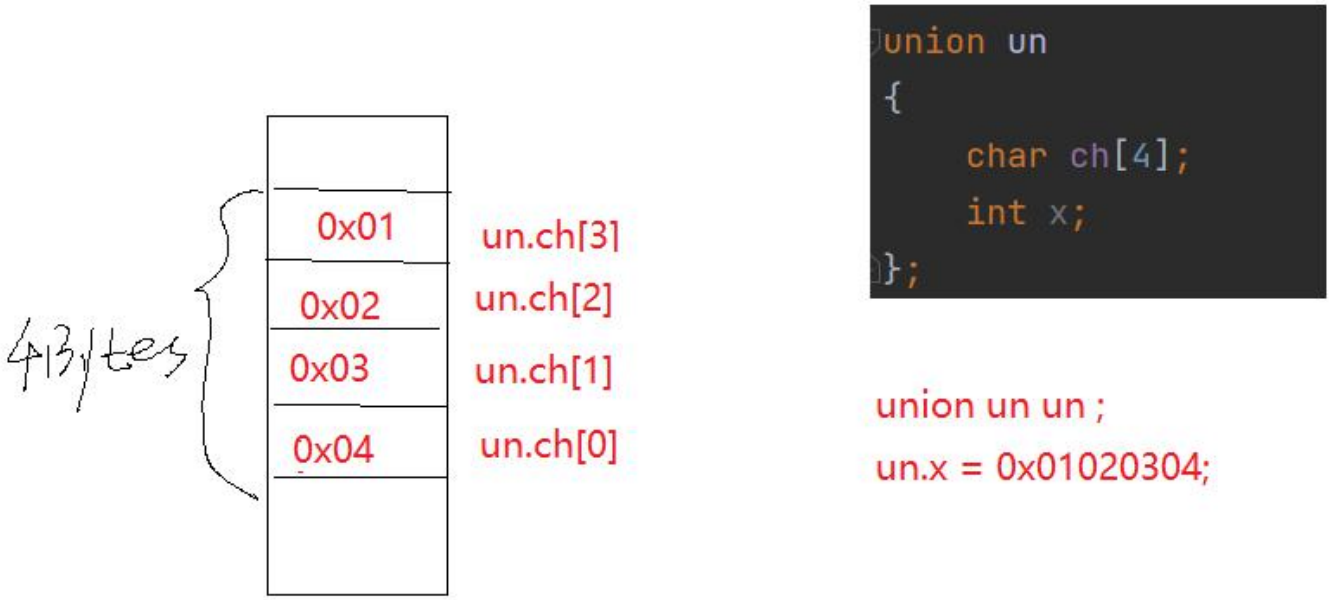
```
union un
{
    char ch[4];
    int x;
};

int main()
```

```
{
    union un un;
    un.x = 0x01020304;
    printf("0x%x 0x%x 0x%x 0x%x\n", un.ch[0],un.ch[1],un.ch[2],un.ch[3]);
}
```

程序运行结果如下：

```
0x4 0x3 0x2 0x1
```

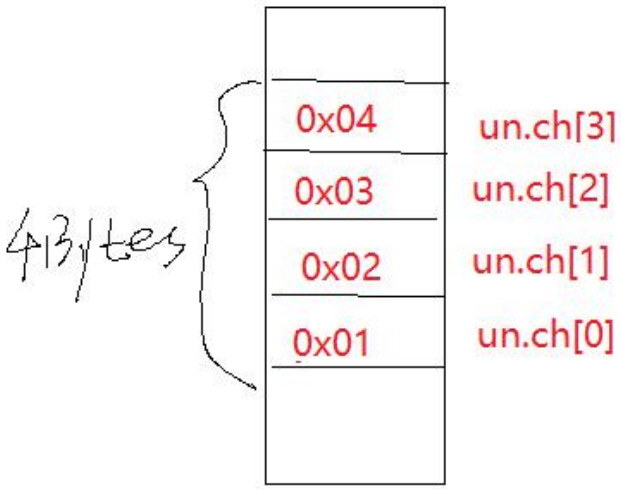
该共用体的内存布局如下：



根据程序的运行结果，我们发现因为我们可以得出结论：低字节的数据是存储在低地址，高字节的数据存储在高地址。这种存储模式我们称之为：**小端模式**

2、如果低字节的数据是存储在高地址，高字节的数据存储在低地址。这种存储模式我们称之为：**大端模式。**

因为我们的计算机是小端模式，所以无法演示大段模式，大端模式的内存布局如下：




```
union un
{
    char ch[4];
    int x;
};
```

union un un ;

un.x = 0x01020304;

3、注意：大端模式和小端模式和操作系统没关系！ 和处理器（CPU）有关系
目前绝大多数的处理器都是小端模式，但是有一些嵌入式处理器是大端模式，甚至有些嵌入式处理器是可以设置为大端模式还是小端模式（两种都支持）。

4、笔试题

> 请编写代码测试某个处理器是大端模式还是小端模式

```
union un{
    char ch[2];
    short x;
};
int main(int argc, char *argv[])
{
    union un u;
    u.x = 0x0102; //u.x = 0x0001;
    if (u.ch[0] == 0x02) //if (u.ch[0] == 0x01)
        printf("小端模式\n");
    else
        printf("大端模式\n");
    //printf("0x%x 0x%x\n", u.ch[0], u.ch[1]);

    return 0;
}
```

10.3 枚举

10.3.1 枚举的定义

> enum 类型名{
> 成员1,
> 成员2,
> ...
> 成员n
> };
> 注意：成员的前面是不需要声明数据类型的，只需要写成员名称即可

10.3.2 枚举成员的初始化

枚举是将所以的成员——列举！，如果第一个成员没有初始化，则第一个成员的值为0，往后的每一个成员的值依次+1，直到列举完成或者遇到某个成员使用了赋值语句进行赋值，+1停止，被赋值的成员的值就是被赋值的值，该成员往后的每一个成员的值则在该成员的值的基础上依次+1

```
enum en{
        a,  //a == 0
        b, // b == 1
        c = 10,//c == 10
        d // d == 11
};
```

```
enum en{
        a=100,  //a ==100
        b, // b == 101
        c = 10,//c == 10
        d // d == 11
};
```

10.3.3 枚举的实际用途

假如用一个数组存储每个月的天数，二月默认是28天
int mon[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
假如需要获取某个月的天数，比如5月份的天数，下标法对mon数组进行访问：mon[4]，但是这样写代码的可读性不强，mon[4]读者是不一定知道代表的是5月份的天数，如果写成这样子：
mon[May] 可读性就强！
按照此思路：一月份：Jan， 二月：Feb 三月：March ....

如果用宏定义实现：

```
#define Jan 0
#define Feb 1
#define March 2
```

也可以用枚举实现：

```
enum MONTH{
        Jan ,
        Feb ,
        March
};
```