chapter12 共用体与枚举类型

1.共用体（union）

1.1概念和定义

DEF：一种数据类型，让几种不同类型的变量存放到【同一段内存单元中】

```
【定义共用体类型】
union 共用体名
{
    成员列表;
};
```

ex:
union date
{
int a;
float b;
char c;
};

- 此data共用体包含3个数据成员，a、b和c。
- 此共用体类型data可以把一个整形变量、一个实型变量和一个字符型变量【放在同一个地址开始】的内存单元中
- 这3个变量在内存中所占的字节数不同，**但它们都从同一地址开始存放**
- - **在引用时（赋值）使用覆盖技术，几个变量的值可以相互覆盖（若后面有其他变量的赋值，前面变量的赋值就失效了），在某一时刻只有最后一次赋值的那个成员变量的值起作用**

```
【定义共用体变量】
union 共用体名 变量列表;
```

> 补充：共用体变量和结构体变量的区别
> 结构体：每一个变量的每个成员分别占有自己的内存单元，结构体变量所占内存空间的长度是各成员所占内存长度之和。
> 共用体：各个成员共同占有同一段内存空间，**【共用体变量所占内存空间的长度是其成员中所占内存长度最长的那个成员的长度】**

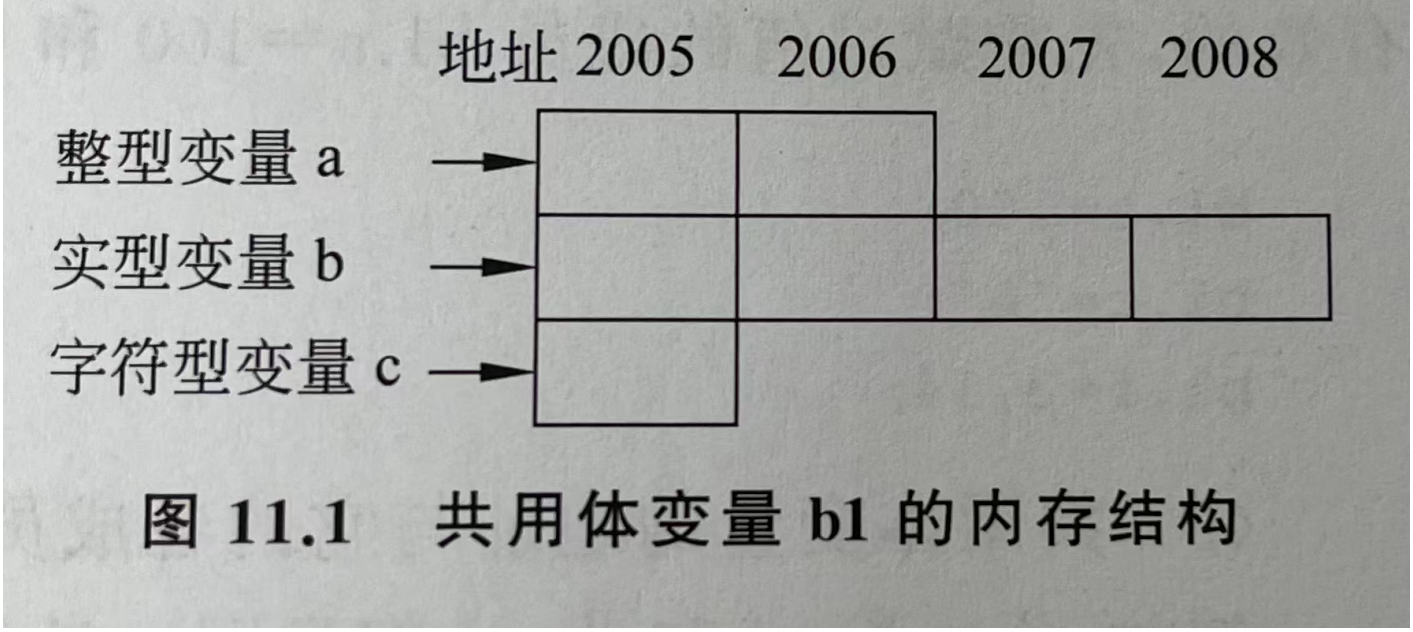


图 11.1 共用体变量 b1 的内存结构

1.2 共用体变量的引用

原则一：不能整体引用共用体变量，只能引用共用体变量中的成员【和结构体一样的捏，不能整体赋值】
ex:b2=b1;是错误的

原则二：可以定义共用体类型的指针变量，指向一个同类型的共用体变量x。同时可用->来引用其成员

```
union data
{
    int a;
    float f;
}x,*px;

px=&x;
px->a=100;
或(*px).a=100;
```

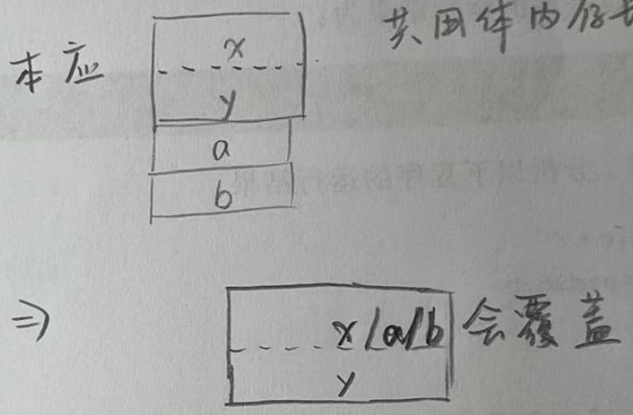
共用体的特点

- 共用体类型中可以包含几个不同类型的成员，这几个不同类型的成员可以存放在同一个内存段里。【但某一时刻该内存段中只能存放其中一种成员，而不是同时存放几个成员】
- 共用体变量中起作用的成员是最后一次赋值存放的那个成员。【先前赋过值的成员被新赋值的成员所覆盖而失去作用！】
  如
  b1.a=100;
  b1.c='A';
  b1.f=3.14;
  最后存放的只有f，a和c尽管被赋值过，但已经被覆盖了
- 如上例子，共用体变量的地址和它的各个成员的地址相同！
  即&a=&a.i=&a.f
- 不能在定义共用体的时候对它进行初始化

> **不能把共用体变量作为函数参数，函数的返回值也不能带回共用体变量。**【和结构体是不一样的！】
> 但可以【使用指向共用体变量的指针作为函数参数】！

理解内存覆盖（内含结构体）
ex1：

```
/*c11_01.c*/
#include<stdio.h>
main()
{   union emp
    {   struct
        {   int x;
            int y;
        }stc;
        int a;
        int b;
    }u;
    u.a=1; u.b=2;
    u.stc.x=u.a+u.b;
    u.stc.y=u.a+u.b;
    printf("%d,%d\n",u.stc.x,u.stc.y);
}
```



程序分析：程序中的 u 是共用体变量，其中的 stc 是结构体变量，a、b 和 stc 共享共用体变量 u 的一段内存空间，执行 u.a＝1；u.b＝2；语句后，u.a＝1；被 u.b＝2；所覆盖，u.a 和 u.b 的值都是 2。执行 u.stc.x＝u.a＋u.b；语句后，u.stc.x＝4；。这样一来，u.a 和 u.b 的值 2 又被 u.stc.x＝4；的值所覆盖，因此，u.a、u.b 和 u.stc.x 的值都是 4。执行 u.stc.y＝u.a＋u.b；语句后，u.stc.y＝8；。

程序的运行结果为：

```
4,8
```

上面的改版。
最后一个输出结果为
4 4 4 4

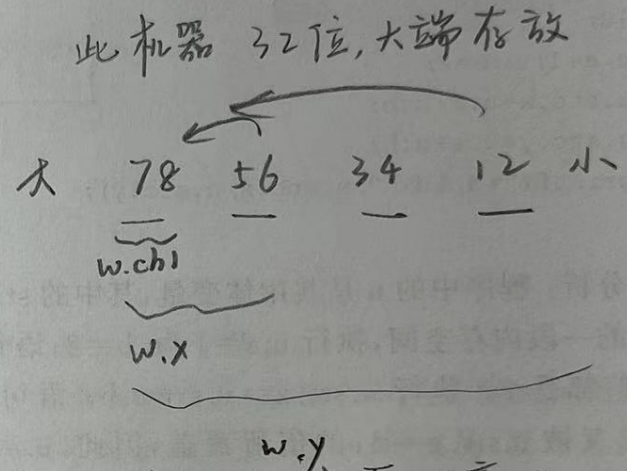
```
#include"stdio.h"
int main()
{
    union emp
    {
        struct{
            int y;
            int x;
        }stc;
        int a;
        int b;
    }u;
    u.a=1;u.b=2;
    printf("a=%d b=%d",u.a,u.b);
    u.stc.x=u.a+u.b;
    printf("a=%d b=%d x=%d",u.a,u.b,u.stc.x);
    u.stc.y=u.a+u.b;
    printf("a=%d b=%d x=%d y=%d",u.a,u.b,u.stc.x,u.stc.y);
}
```

所以这样就很清楚的说明了，a、b和结构体变量先定义的那个共用同一块内存，后定义的那个再开辟一块地址

ex2：存放方式

【例 11.3】 分析以下程序的运行结果。

```
/*c11_03.c*/
#include<stdio.h>
main()
{   union
    {   short int x;
        long y;
        unsigned char ch1;
    }w;
    w.y=0x12345678;
    printf ("%x\n", w.y);
    printf ("%x\n", w.x);
    printf ("%x\n", w.ch1);
}
```



程序分析：程序中定义了一个共用体变量 w，在 32 位机器中，short int 型占 2 字节，long 型占 4 字节，unsigned char 型占 1 字节，因此，共用体变量 w 所占内存单元的长度是 4 字节。在执行 w.y＝0x12345678;语句后，由于 0x12345678 是十六进制数，因此，共用体变量 w 的第 1 个字节单元存放 0x78，第 2 个字节单元存放 0x56，第 3 个字节单元存放 0x34，第 4 个字节单元存放 0x12。

程序运行结果为：

```
12345678
5678
78
```

1.3共用体编程举例

- 在不同条件下输出不同数据（也仍可替换成别的数据结构，但共用体占据空间比较小！）

11.1.3 共用体类型编程举例

【例 11.4】 编写一个程序，输入若干人员的数据，每个人员的数据包括姓名、年龄、性别和就业情况，若已就业要输入工作单位；若失业要输入失业年数。最后输出这些数据。

分析：首先定义一个结构体类型的数据，包括 5 个数据成员：人员姓名(用字符型数

据表示)、年龄(用整型数据表示)、性别(用字符变量表示)、就业情况(用字符型变量表示)、工作单位或失业年数(用共用体类型的变量表示)。若就业情况是'y',第 5 个数据项是工作单位;若就业情况是'n',第 5 个数据项是失业年数。程序如下:

```
/*c11_04.c*/
#include<stdio.h>
#define N  3
struct
{  char name[20];
   int age;
   char sex;
   char job;
   union
      {  int count;
         char workplace[30];
      }category
} person[N];
main()
{  int i;
   for (i=0, i<N, i++)
   (  scanf ("%s %d %c %c", person[i].name, &person[i].age, &person[i].sex,
&person[i].job);
      if (person[i].job=='n')  scanf ("%d", &person[i].category. count);
       else if (person [i].job = = 'y')   scanf ("%s", &person [i]. category.
workplace);
          else  printf("input error1");
)
   printf ("\n");
   printf ("Name    Age   Sex   Job    unemploy_count /Workplace\n");
   for (i=0, i<N, i++)
    {  if(person[i].job=='N')
          printf("%-10s%-6d%-3c%-3c%-20d\n", person[i].name, person[i].age,
       person[i].sex, person[i].job, person[i].category.count);
       else
          printf ("%-10s%-6d%-3c%-3c,%-20s\n",person[i].name, person[i].
       age, person[i].sex, person[i].job, person[i].category.workplace);
    }
}
```

程序运行结果如下:

```
Li 28 m y shenyang ligong university
Wang 45 f n 10
Chen 30 f y shenyang house property   bureau
Name      Age       Sex       Job     unemploy_count /Workplace
Li         28        m         y      shenyang ligong university
Wang       45        f         n      10
Chen       30        f         y      shenyang house property bureau
```

2.枚举类型

某些数据的取值都被限定在几个可能值的范围内（口袋中红、黄、蓝，一个星期有7天）

2.1枚举类型的概念及其变量定义

DEF：枚举类型的数据是把变量的所有取值一一列举出来，变量的值只限于列举出来的值的范围内。
【若一个变量只有几种可能的值，就可以定义该变量为枚举类型的变量了】

`枚举类型定义：enum 枚举类型名 {枚举元素列表};`

.

`枚举变量定义：enum 枚举类型名 变量名;`

ex:
enum weekday{sun,mon,tue,wed,thu,fir,sat};
enum weekday workday,weekend;

其中，sum、mon等称为枚举元素或枚举常量，是右用户自己定义的标识符

2.2枚举类型数据的使用

- C语言编译中，对枚举元素按整常量处理。不是变量，不能在定义之外对它们赋值
  如sum=0;mon=1;是错的
- 枚举元素作为常量，在C语言编译的时候按照定义的顺序使他们的值为0，1，2，3...
  如上面的weekday类型，sun=0,mon=1,tue=2...sat=6
- **【也可以在定义时，改变枚举变量的值（没有改变的，一样递增）】**
  enum weekday{sum=7,mon=1,tue,wed,thu,fir,sat}workday;
  从mon之后，各元素的值顺序+1
- 可对枚举变量赋值，但取值范围限定在枚举列表中的各值.**【但一个整数不能直接赋值给一个枚举变量！】**
  workday=mon;
  printf("%d",workday);//输出1
  但workday=1;是错误哒！

2.3 很好的例子

```
一个口袋中有红黄蓝3个球，依次从口袋中拿出所有球，编写程序，输出所有的拿法
```

.

```
#include"stdio.h"
enum color{red=1,yellow,blue};
void print(enum color ball)
{
    switch(ball)
    {
        case red:printf("red");break;
        case yellow:printf("yellow");break;
        case blue:printf("blue");break;
    }
}
int main()
{
    enum color i,j,k;
    for(i=red;i<=blue;i=(enum color)(i+1))
    {
        for(j=red;j<=blue;j=(enum color)(j+1))
        {
            for(k=red;k<=blue;k=(enum color)(k+1))
            {
                if(i!=k&&k!=j&&j!=i)
                {
                    print(i);printf("  ");print(j);printf("  ");print(k);
                    printf("\n");
                }
            }
        }
    }
}
```

注意：
·····不是每一个for循环之后都要加花括号{}的！如果只有一个语句就最好别加，会增加程序可读性的呢！
·····本例子中的(enum color)(i+1)可以简化成i++，也可以是i=i+1
>>>>>>所以枚举类型是【可以在使用的时候和一切整形一样使用的】，【只是不能像整形变量一样被赋值！】

3.用typedef定义类型

C语言中，除了可以直接使用C提供的标准类型名（int、float、long等）和用户自己声明的结构体、共用体个枚举类型外，还可以使用typedef声明新的类型名来代替原有的类型名

`typedef 类型名 标识符;`

ex:
typedef int A[10];
声明了一个新的整形数组类型名A，该类型数组包含的元素个数为10》》含10个int类型的数组类型 A

- 类型名 是【已有定义】的类型标识符，而不是定义一种新的数据类型
- 标识符 为用户自己定义的用来替代原有类型名的类型标识符

ex:

```
typedef int INTEGER;
int i;===INTEGER i;

typedef int NUM[10];//声明NUM为包含10个元素的整形数组类型
NUM a;//定义a为包含10个元素的整形数组

typedef struct node{
  char data[20];
  struct node *next;
}STYPE;
STYPEnodel,*p;//定义了一个具有node结构体类型的变量和一个指向这个结构体变量的指针p
```

- 用typedef声明的类型名通常用大写字母表示，以便于系统提供的标准类型名区别开来
- typedef只是对原有的类型起个新名字，没有生成新的数据类型
- typedef和#define有相似之处，但两者的作用不同
  #define是在系统预编译的时候处理，他只能做简单的字符串替换
  typedef是在系统预编译的时候处理的，他并不是简单的字符串替换！！！（而是一种类型名的替换）

typedef的经典用法，用来构建没有提供多维数组的数据类型，如char（结构体应该也可以捏）
》》创造了大小为n，每个元素都是字符串的一个数组呢！

```
#include "stdio.h"
typedef char ch[20];
int main()
{
    ch ch1[3];//有3个可以存放20个字符的字符串数组
    for(int i=0;i<3;i++)
    {
        scanf("%s",&ch1[i]);
    }
    for(int i=0;i<3;i++)
    {
        printf("%s",ch1[i]);
    }
}
```