第七章　　结构体、共同体和枚举类型

定义：

将不同种类型的数据有序地组合在一起，构造出一个新的数据类型，这种形式称为结构体。

结构体是多种类型组合的数据类型。

struct　　结构体名

{　成员列表　};

关键字　　结构体名

```
struct  student
{   int  num;
    char  name[20];
    char  sex;
    char  addr[30];
};
```

不同数据类型组成的成员

分号不能少

定义结构体类型变量的方法

一、先定义结构体类型再定义变量名

```
struct  student
  {   int  num;
      char  name[20];
      char  sex;
      int  age;
      float  score;
      char  addr[30];
  };
struct  student   student1,  student2;
```

结构体类型只是一种数据类型，不占内存空间，只有定义结构体类型变量时才开辟内存空间。

结构体类型名　变量1　变量2

```
# define  STUDENT  struct   student
    STUDENT
     {    int  num;
          char  name[20];
          char  sex;
           int  age;
          float  score;
          char  addr[30];
       };
    STUDENT    student1,student2;
```

凡是STUDENT的地方都用struct student机械替换。

二、在定义类型的同时定义变量

```
struct   student
   {    int  num;
        char  name[20];
        char  sex;
        int  age;
        float  score;
        char  addr[30];
} student1, student2;
```

struct 结构体名

{

成员列表

} 变量名列表;

紧接着定义变量

三、直接定义结构体类型变量

```
struct
  {   int  num;
      char  name[20];
      char  sex;
      int  age;
      float  score;
      char  addr[30];
  } student1, student2;
```

```
struct
    {
        成员列表
    } 变量名列表;
```

不出现结构体名。

1、结构体类型的变量在内存依照其成员的顺序顺序排列，所占内存空间的大小是其全体成员所占空间的总和。

2、在编译时，仅对变量分配空间，不对类型分配空间。

3、对结构体中各个成员可以单独引用、赋值，其作用与变量等同。

格式：变量名.成员名　student1.num

4、结构体的成员可以是另一个结构体类型。

```
struct  date
{  int   month;
   int   day;
   int  year;
};
```

```
struct  student
{  int   num;
   char  name[20];
   struct  date    birthday;
};
```

成员类型

成员名

5、成员名可以与程序中的变量名相同，二者分占不同的内存单元，互不干扰。例如，在程序中仍可以定义变量

```
int  num;
```

结构体类型变量的引用

1、不能对结构体变量整体赋值或输出，只能分别对各个成员引用。

错误 cin>>student1;

必须用成员名引用

cin>>student1.num; student1.num=100;

可以将一个结构体变量整体赋给另外一个相同类型的结构体变量。 student2=student1;

2、嵌套的结构体变量必须逐层引用。

student1.birthday.day=25;

3、结构体变量中的成员可以同一般变量一样进行运算。

student1.birthday.day++; student1.score+=60;

对局部变量类型的结构体变量初始化

```
void main(void)
{   struct  student
   {    long int  num;
        char  name[20];
        char  sex;
        char  addr[30];
    }   student1={901031, “Li Lin”, ‘M’, “123 Beijing Road”};
         cout<<student1.name<<endl;
```

对变量初始化，一一赋值

输出：　LiLin

关于结构类型变量的使用，说明以下几点:

1、同类型的结构体变量之间可以直接赋值。这种赋值等同于各个成员的依次赋值。

2、结构体变量不能直接进行输入输出，它的每一个成员能否直接进行输入输出，取决于其成员的类型，若是基本类型或是字符数组，则可以直接输入输出。

3、结构体变量可以作为函数的参数，函数也可以返回结构体的值。当函数的形参与实参为结构体类型的变量时，这种结合方式属于值调用方式，即属于值传递。（举例说明）

结构体数组

结构体数组中的每个元素都是一个结构体类型的变量，其中包括该类型的各个成员。数组各元素在内存中连续存放。

一、结构体数组的定义

```
struct  student
  {   int  num;
      char  name[20];
      char  sex;
      int  age;
      float  score;
      char  addr[30];
} ;
struct  student  stu[30];
```

```
struct  student
  {   int  num;
      char  name[20];
      char  sex;
      int  age;
      float  score;
      char  addr[30];
} stu[30];
```

直接定义

二、结构体数组的初始化

```
struct   student
   {    int  num;
        char  name[20];
        char  sex;
} stu[3]={ {1011, "Li Lin",'M'}, {1012,"Wang Lan",'F'},
                {1013,"Liu Fang",'F'};
```

```
struct   student
  {    int  num;
       char  name[20];
       char  sex;
} stu[ ]={ {1011,"Li Lin",'M'}, {1012,"Wang Lan",'F'},
                {1013,"Liu Fang",'F'}};
```

以下程序的结果是：

```
void main(void)
{  struct  date
     {  int  year, month,  day;
     }  today;
    cout<<sizeof(struct  date)<<endl;
}
```

12

根据下面的定义，能打印出字母M的语句是：

```
struct  person {  char  name[9];
                  int  age;
                };
struct  person class[10]={  “Jone”,17, “Paul”,19,
                            “Mary”,18, “Adam”,16
                          };
```

A) cout<<class[3].name<<endl;　　输出：Adam

B) cout<<class[3].name[1]<<endl;　输出：d

C) cout<<class[2].name[1]<<endl;　输出：a

D) cout<<class[2].name[0]<<endl;　输出：M

结构体类型的静态成员

当把结构体类型中的某一个成员的存储类型定义为静态时，表示在这种结构类型的所有变量中，编译程序为这个成员只分配一个存储空间，即这种结构体类型的所有变量共同使用这个成员的存储空间。

<类型> <结构体类型名>：：<静态成员名>;

其中类型要与在结构体中定义该成员的类型一致，结构体类型名指明静态成员属于哪一个结构体。

```
struct  s{
static int  id;
int  eng;
};
int  s::id=50;
```

数据类型

结构体类型

这时，未定义结构体变量，但已将静态成员的空间安排好。

若有定义：s s1,s2;

则变量s1,s2的id成员占用同一存储空间（静态区）。

在结构体中说明的静态成员属于引用性说明，必须在文件作用域中的某一个地方对静态的成员进行定义性说明，且仅能说明一次。

int s::id;

说明id的初值为0（**静态变量的缺省初值均为0**）

共用体

C++语言中，允许不同的数据类型使用同一存储区域，即同一存储区域由不同类型的变量共同表示。这种数据类型就是共用体。

```
union 共用体名
{ 成员表列;
} 变量表列;

union data a, b, c;
```

```
union data
{ int i;
  char ch;
  float f;
} a, b, c;
```

这几个成员在共用体变量中存放在同一地址，相互覆盖，其长度为最长的成员的长度。

共用体变量的引用

不能整体引用共用体变量，只能引用变量中的成员。

a.i　表示为整型

a.ch　表示为字符型

a.f　表示为符点型

共用体变量的特点

1、共用体的空间在某一时刻只有一个成员在起作用。

2、共用体变量中的成员是最后一次放入的成员。

3、共用体变量不能在定义时赋初值。

4、共用体变量不能作为函数的参数或函数值，但可使用指向共用体的指针变量。

5、共用体可以作为结构的成员，结构体也可以作为共用体的成员。

```
union  un
{  int  i;
   double  y;
};
struct  st
{  char  a[10];
  union  un   b;
};
cout<<sizeof(struct  st)<<endl;
```

18

```
union   un
{ short  int  a;
   char  c[2];
} w;
w.c[0]='A';   w.c[1]='a';
cout<<oct<<w.a<<endl;
```

低字节低地址

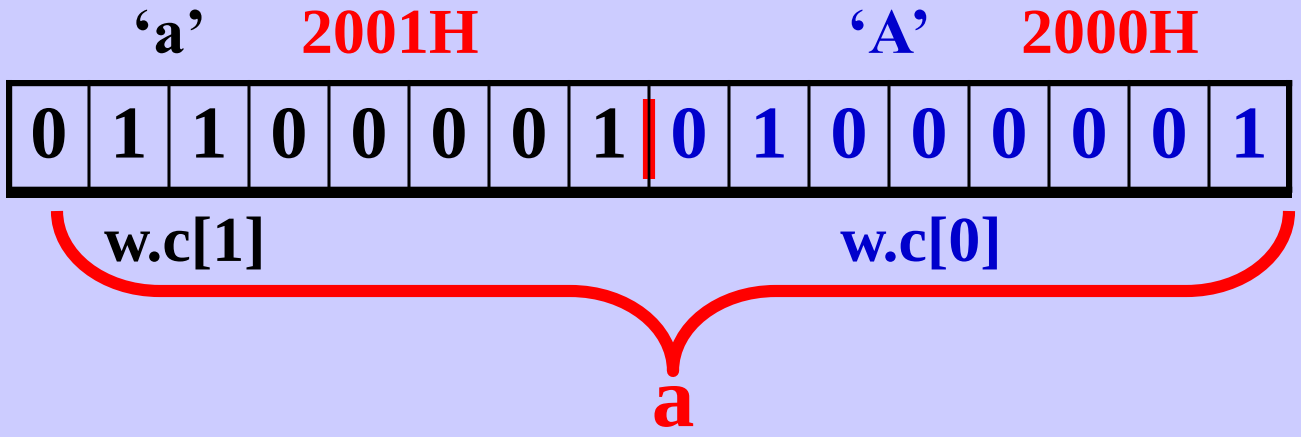
高字节高地址

2000H	6
2001H	5

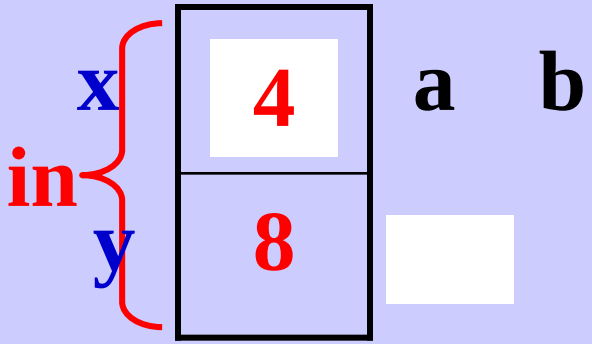
65 ?

56 ?

输出：060501



```
void main(void)
{   union EXAMPLE
    {    struct { int x, int y;}  in;
         int a,b;
    }e;
    e.a=1;
    e.b=2;
    e.in.x=e.a*e.a;
    e.in.y=e.b+e.b;
    cout<<e.in.x<<'\t'<<e.in.y<<endl;
}
```



输出：4　8

枚举类型

如果一个变量只有几种可能的值，可以定义为枚举类型。

枚举类型就是将变量的值一一列举出来，变量的值仅限于列举出来的值的范围内。

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

数据类型　　可能取的值

变量

enum weekday workday, weekend ;

workday 和 weekend 值只能是sun 到 sat 其中之一。

另一种定义变量的方法

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

其中sun, mon,....,sat称为枚举元素或枚举常量，为用户定义的标识符，所代表的意义由用户决定，在程序中体现出来。

1、枚举元素为常量，不可赋值运算。 sun=0; mon=1;

2、在定义枚举类型的同时，编译程序按顺序给每个枚举元素一个对应的序号，序号从0开始，后续元素依次加1。

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

0 , 1, 2, 3, 4, 5, 6

3、可以在定义时人为指定枚举元素的序号值。

enum weekday {sun=9, mon=2, tue, wed, thu, fri, sat};

9 , 2, 3, 4, 5, 6 , 7

4、只能给枚举变量赋枚举值，若赋序号值必须进行强制类型转换。

day=mon ; day=1; day=(enum weekday)1;

5、枚举元素可以用来进行比较判断。

if (workday= = mon)　　if (workday>sun)

6、枚举值可以进行加减一个整数n的运算，得到其前后第n个元素的值。

workday=sun;　　workday= = tue

workday=(week)(workday+2);

7、枚举值可以按整型输出其序号值。

workday=tue;

cout<<workday;　　2

```
void main(void)
{
   enum team{ qiaut, cubs=4, pick, dodger=qiaut-2;};
   cout<<qiaut<<‘\t’<<cubs<<‘\t’;
   cout<<pick<<‘\t’<<dodger<<endl;
}
```

输出：0　4　5　-2