西北工業大學

NORTHWESTERN POLYTECHNICAL UNIVERSITY

C++程序设计
Programming in C++

U10G13027/U10G13015

主讲：魏英，计算机学院

- 数组是一种数据形式，其特点是多个相同类型元素集合起来；
- 结构体是另一种重要的数据形式，其特点是不同类型成员组合起来。

- 结构体允许将不同类型的数据元素组合在一起形成一种新的数据类型，其声明形式为：

```
struct 结构体类型名 {
    成员列表
};
```

例如可以通过如下声明建立能表示学生信息的数据类型。

```
struct STUDENT { //学生信息类型
    int no;     //声明一个整型数据成员表示学号
    char name[21]; //声明一个字符串数据成员表示姓名
    char sex; //声明一个字符数据成员表示性别
    int age; //声明一个整型数据成员年龄
    char qq[11]; //声明一个字符串数据成员表示QQ号
    double score;  //声明一个浮点型数据成员表示成绩
};
```

结构体类型声明一般放在程序文件开头，此时这个声明是全局的。在全局作用域内，该声明处处可见，因此同作用域内的所有函数都可以使用它。

结构体类型声明也可以放到函数内部，此时这个声明是局部的。

若在函数内部有同名的结构体类型声明，则全局声明在该函数内部是无效的，有效的是局部声明的。

- （1）结构体类型和普通数据类型（如int、char、double等）一样是类型名称，而不是该类型的一个实体，因此不会为结构体类型分配存储空间。

- （2）结构体类型声明时成员可以又是是结构体类型。

```
struct DATA {
    int year,month,day;
};
```

```
struct STAFF { //职员信息类型
    int no;     //工号，整型
    char name[21]; //姓名，字符串
    char sex; //性别，字符型
    DATA birthday; //出生日期，结构体类型
    double salary;  //薪水，浮点型
};
```

- （3）C语言的结构体类型只能用“struct 结构体类型名”表示，如“struct STUDENT”。C++兼容C语言的结构体类型，既可以用C语言方式，又可以直接用“结构体类型名”表示，如“STUDENT”。建议C++程序员使用后一种方式。

- 定义结构体对象称为结构体类型实例化（instance），实例化会根据数据类型为结构体对象分配内存单元。

- 定义结构体对象有三种形式。
- 1. 先声明结构体类型再定义对象

```
结构体类型名 结构体对象名列表; //C++方式
struct 结构体类型名 结构体对象名列表; //兼容的C语言方式
```

```
STUDENT a,b; //C++方式定义结构体对象
struct STUDENT x,y; //C语言方式定义结构体对象
```

- 2. 声明结构体类型的同时定义对象
- 一般形式为：

```
struct 结构体类型名 {
    成员列表
} 结构体对象名列表;
```

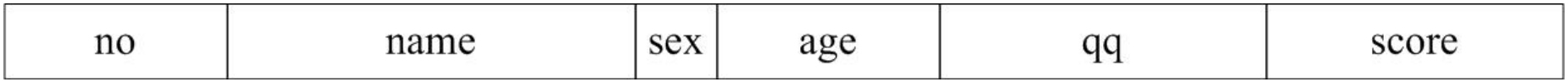
```
struct DATE { //日期类型
    int year,month,day; //年，月，日 整型
} d1,d2; //定义结构体对象
```

- 3．结构体对象的内存形式
- 实例化结构体对象后，对象会得到存储空间。

8.2.1 结构体对象的定义

```
struct STUDENT { //学生信息类型
    int no; //声明一个整型数据成员表示学号
    char name[21]; //声明一个字符串数据成员表示姓名
    char sex; //声明一个字符数据成员表示性别
    int age; //声明一个整型数据成员年龄
    char qq[11]; //声明一个字符串数据成员表示QQ号
    double score; //声明一个浮点型数据成员表示成绩
}s1;
```

图8.1 结构体对象s1的内存结构



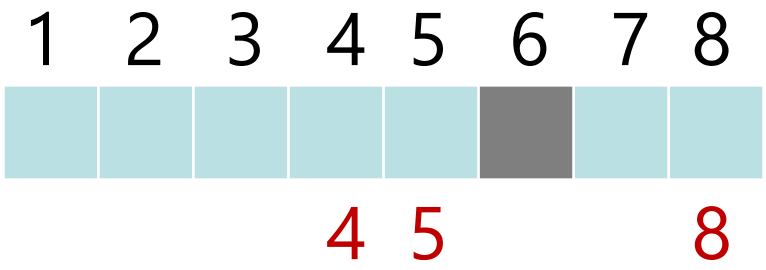
- S1所占的而存储空间为：4+21+1+4+11+8=49字节

- 4．字节对齐
- 在大部分的编译器中，实际结构体变量的长度会比理论值大。
- 因为编译器为了加快数据存取的速度，会对结构体变量的存储位置进行处理，称为字节对齐。

- 编译器的字节对齐数n（n=1,2,4,8,16）；
- 最大成员内存长度Max(Li)；
- 结构体总长度应该是Min(n,Max(Li))的倍数。

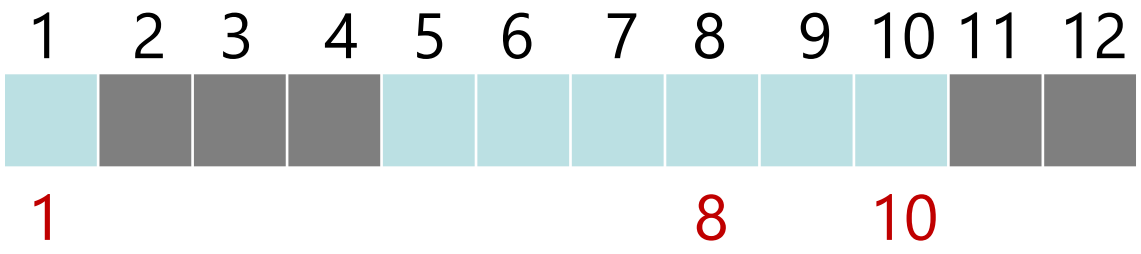
- 例：设编译器对齐字节数为8，则结构体变量所占的字节数应该是4的倍数。

```
struct A {
    int a;
    char b;
    short c;
};
cout<<sizeof(A)<<endl;
```



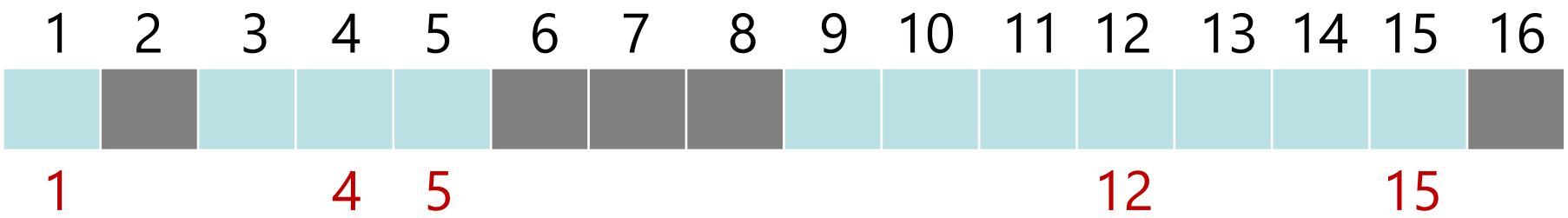
- 例：设编译器对齐字节数为8，则结构体变量所占的字节数应该是4的倍数。

```
struct B {
    char b;
    int a;
    short c;
};
cout<<sizeof(A)<<endl;
```



- 例：设编译器对齐字节数为8，则结构体变量所占的字节数应该是4的倍数。

```
struct C {
    char a;
    short b;
    char c;
    int d;
    char e[3];
};
cout<<sizeof(A)<<endl;
```



- 可以在结构体对象定义时进行初始化。

```
struct STAFF { //职员信息类型
    int no;     //工号，整型
    char name[21]; //姓名，字符串
    char sex; //性别，字符型
    DATA birthday; //出生日期，结构体类型
    double salary;  //薪水，浮点型
};
```

```
STAFF s1={1001,“Li Min”,'M',{1980,10,6},2700.0};
STAFF s2={1002,"Ma Gang",'M',1978,3,22,3100.0};
```

- 1. 结构体对象成员引用
- 使用结构体对象主要是引用它的成员，其一般形式为：

结构体对象名.成员名

表8-1 对象成员引用运算符

运算符	功能	目	结合性	用法
.	对象成员引用运算	双目	自左向右	**object.member**

```
STAFF a,b;
a.no=10002; //将10002赋值给a对象中的no成员
b.salary=a.salary+500.0; //在表达式中可以引用对象成员
a.no++; //按优先级等价于(a.no)++
```

如果成员本身又是一个结构体对象，就要用成员引用运算符，一级一级地引用。例如：

```
STAFF x;
x.birthday.year=1990,
x.birthday.month=5,
x.birthday.day=12; //逐级引用成员
```

- 2. 结构体对象输入与输出
- 不能将一个结构体对象作为整体进行输入或输出，只能对结构体对象中的成员逐个进行输入或输出。例如：

```
STAFF x;
cin>>x.no>>x.sex>>x.salary;
cin>>x.birthday.year>>x.birthday.month>>x.birt
hday.day;
cin>>x.name;
```

- 3. 结构体对象的运算
- 结构体对象可以进行赋值运算，但不能对进行算术运算、关系运算等，例如：

```
COMPLEX m,n,k;
m=n; //正确，结构体对象允许赋值
k=m+n; //错误，结构体对象不能做算术运算
m>n; //错误，结构体对象不能做关系运算
```

回顾：

- 定义结构体类型
- 定义结构体对象
- 结构体对象的内存形式
- 结构体对象成员的引用
- 结构体对象的输入输出
- 结构体对象的运算

```
struct DATE {
    int year,month,day;
};
struct STAFF {
    int no;
    char name[21];
    char sex;
    DATA birthday;
    double salary;
};
STAFF  a,b;
a.no=1101;
b.birthdat.year=1988;
a=b;
sum=a.salary+b.salary;
```

- 数组元素可以是结构体对象。

- 数组元素可以是结构体类型，称为结构体数组，如一维结构体数组定义形式为：

```
struct 结构体类型名 结构体数组名[常量表达式];
```

例如表示平面上若干个点的数据对象，可以这样定义：

```
struct POINT { //点类型
    int x,y; //平面上点的x、y坐标
};
POINT points[100]; //表示100个点的数据对象
```

- 一维结构体数组初始化形式为：

```
struct 结构体类型名 结构体数组名[常量表达式]
={初值序列};
```

- 其中初值序列必须按内存形式做到类型、次序一一对应。

例如：

```
struct RECT { //矩形框类型
    int left,top,right,bottom;
};
RECT rects[3]={{1,1,10,10},
               {5,5,25,32},{100,100,105,200}};
```

初值写法中除最外面的一对大括号外，其他大括号可以省略。例如：

```
RECT _rect[3]={1,1,10,10,5,5,25,32,
                          100,100,105,200};
```

引用结构体数组成员需要将数组下标运算、对象成员引用运算结合起来操作，其一般形式为：

数组对象[下标表达式].成员名

```
rects[0].left=10;
//数组对象[下标表达式]是结构体对象
```

- 结构体类型中可以包含数组成员，数组成员类型既可以是基本数据类型又可以是指针类型或结构体类型，例如表示平面三角形的数据对象，可以这样定义：

```
struct TRIANGLE { //三角形类型
  POINT p[3];//由3个平面上的点描述三角形
};
```

引用结构体数组成员需要将对象成员引用运算、数组下标运算结合起来操作，其一般形式为：

结构体对象.数组成员[下标表达式]

```
struct TRIANGLE tri;
tri.p[0].x=10,tri.p[0].y=10;
//结构体对象.数组成员[下标表达式].成员名
```

例8.1 输入20个学生信息，按成绩递减排序；成绩相同时，按学号递增排序。

```
1  #include <iostream>
2  using namespace std;
3  #define N 20
4  struct tagSTUDENT { //学生信息类型
5      int no; //学号
6      char name[21]; //姓名
7      double score; //成绩
8  };
```

例8.1

```
 9  int main()
10  {
11      struct tagSTUDENT A[N] , t;
12      int i , j=2.0; //消除浮点型bug
13      for (i=0; i<N; i++) //输入学生信息
14          cin>>A[i].no>>A[i].name>>A[i].score;
15      for (i=0; i<N-1; i++) //排序
16        for (j=i; j<N; j++)
17          if (A[i].score<A[j].score
18          || (A[i].score==A[j].score
            && A[i].no>A[j].no))
19              t=A[i], A[i]=A[j], A[j]=t;
```

例8.1

```
20          for (i=0; i<N; i++) //输出学生信息
21              cout<<A[i].no<<","<<A[i].name
                    <<","<<A[i].score<<endl;
22          return 0;
23      }
```

结构体对象各成员的地址：&对象名.成员名，例如：

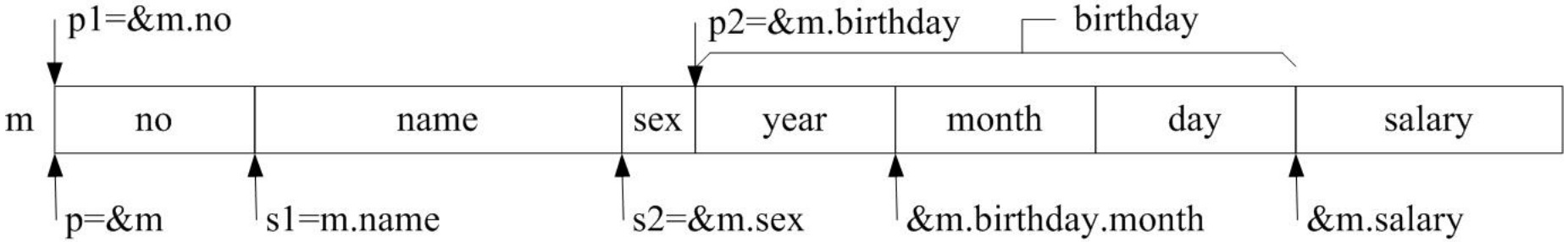
```
STAFF m; //结构体对象
int *p1; //指向no成员的指针类型是int*
char *s1,*s2; //指向name、sex成员的指针类型是char*
DATE *p2; //指向birthday成员的指针类型是DATE*
p1=&m.no; //取no成员的地址
s1=m.name; //name成员是数组，数组名即是地址
s2=&m.sex; //取sex成员的地址
p2=&m.birthday; //取birthday成员的地址
```

结构体对象的地址：&对象名。例如：

```
struct STAFF m, *p; //指向结构体对象的指针
p=&m; //取结构体对象的地址
```

显然，结构体对象的地址值（&m）与第一个成员的地址值（&m.no）相同。

图8.3 结构体对象及成员指针示意



假设p是指向结构体对象的指针，通过p引用结构体对象成员有两种方式：

①对象法：(*p)．成员名；

②指针法：p->成员名。

表8-2 指针成员引用运算符

运算符	功能	目	结合性	用法
->	指针成员引用运算	双目	自左向右	**pointer->member**

```
p->no=10002; //将10002赋值给对象中的no成员，指针成员引用运算结果是左值（即成员本身）
p->salary=p->salary+500.0; //在表达式中引用指针指向的成员
p->no++; //按优先级等价于(p->no)++
```

（1）指针成员引用运算符（->）左边运算对象必须是指向结构体对象的指针，右边member必须是结构体对象中的成员名。其引用形式为：

结构体指针->成员名

（2）如果成员本身又是一个结构体对象指针，就要用指针成员引用运算符一级一级地引用成员。例如：

```
DATE d={1981,1,1};
TEACHER { //教师信息类型
    int no; //工号
    char name[21]; //姓名
    DATE *pbirthday; //出生日期
} a={1001,"Li Min",&d}, *p=&a;
```

```
p->no=10001; //通过指针p引用a的no成员
p->pbirthday->year=2008; //通过指针p->pbirthday引用d的year成员
```

将结构体对象作为函数实参传递到函数中，采用值传递方式。例如：

```
struct DATA {
    int data; //整型成员
    char name[10]; //数组成员
};
void fun1(DATA x); //函数原型
void fun2()
{
    DATA a={1,"LiMin"};
    fun1(a); //函数调用
}
```

8.5.2 结构体数组作为函数参数

将结构体数组作为函数参数，采用地址传递方式。函数调用实参是数组名，形参必须是同类型的结构体数组。例如：

```
void fun3(DATA X[]); //函数原型
void fun4()
{
 DATA  A[3]={1,"LiMin",2,"MaGang",3,"ZhangKun"};
 fun3(A); //函数调用
}
```

采用地址传递方式，形参数组的首地址与实参数组完全相同。

将结构体指针作为函数参数，采用地址传递方式。例如：

```
void fun5(DATA *p); //函数原型
void fun6()
{
    DATA a={1,"LiMin"};
    fun5(&a); //函数调用
}
```

函数的返回类型可以是结构体类型，这时函数将返回一个结构体对象。例如：

```
DATA fun8()
{    DATA a={1,"LiMin"};
    return a; //返回结构体对象，复制到临时对象中
}
void fun9()
{
    DATA b;
    b=fun8(); //函数返回结构体对象，并且赋值
}
```

- 共用体（union）是一种成员共享存储空间的结构体类型。一般形式为：

```
union 共用体类型名 {
        成员列表
};
```

- 共用体类型名与union一起作为类型名称，成员列表是该类型数据元素的集合。一对大括号｛｝是成员列表边界符，后面必须用分号（；）结束。

共用体类型定义时必须给出各个成员的类型声明，其形式为：

成员类型 成员名列表；

成员名列表允许任意数目的成员，用逗号（，）作为间隔。

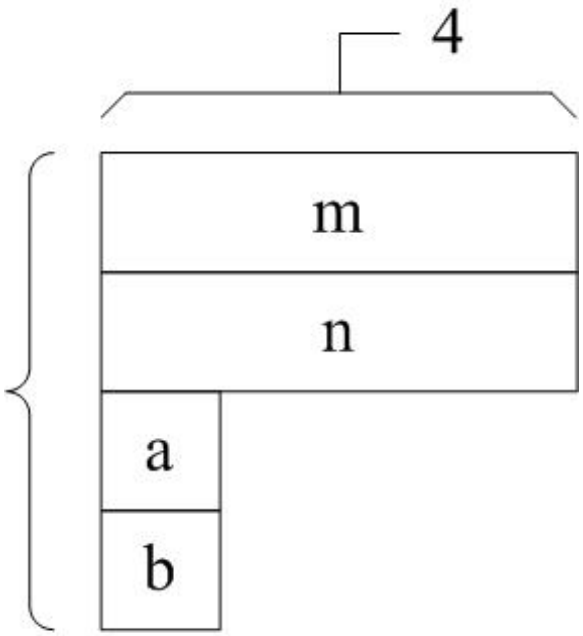
8.6.1 共用体概念及类型定义

共用体中每个成员与其他成员之间共享内存。

```
union A {
    int m,n; //整型成员
    char a,b; //字符成员
}a;
```

对于union A，m、n、a、b共享内存单元，其内存结构如图所示。

图8.4 共用体内存结构示意

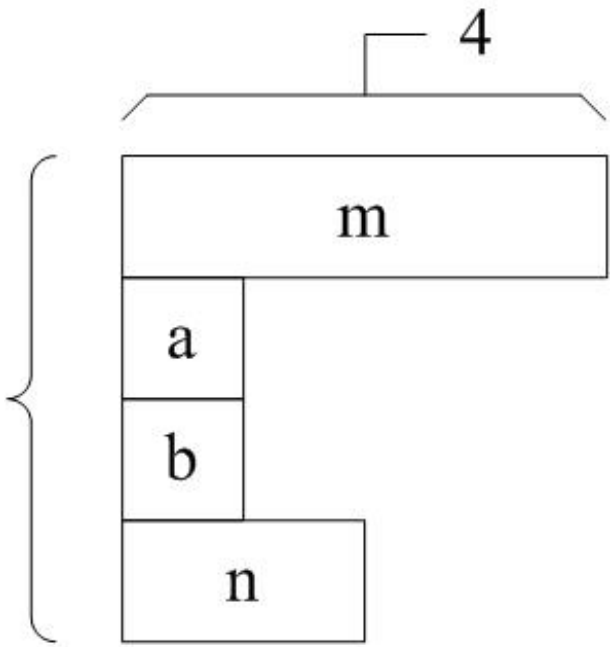


（a）union A

对于union B，m、a、b、n共享内存单元，其内存结构如图所示。

```
union B {
    int m; //整型成员
    char a,b; //字符成员
    short n; //短整型成员
}b;
```

图8.4 共用体内存结构示意



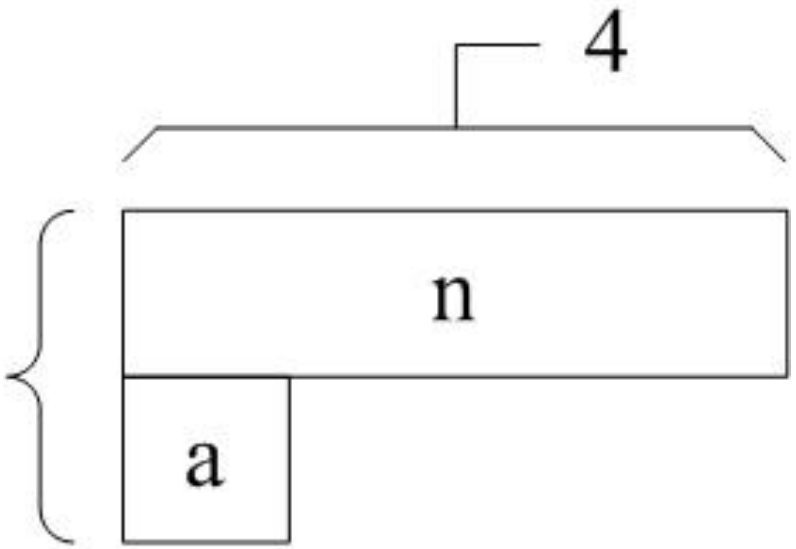
（b）union B

比较共用体和结构体类型

```
union UDATA { //共用体类型
    int n; //整型成员
    char a; //字符成员
}ua;
```

从UDATA类型内存结构，可以看出两个成员共享了同一段内存单元，考虑到整型有4个字节，字符型只有1个字节，相当于a是n的一部分。

图8.5 共用体与结构体内存结构比较

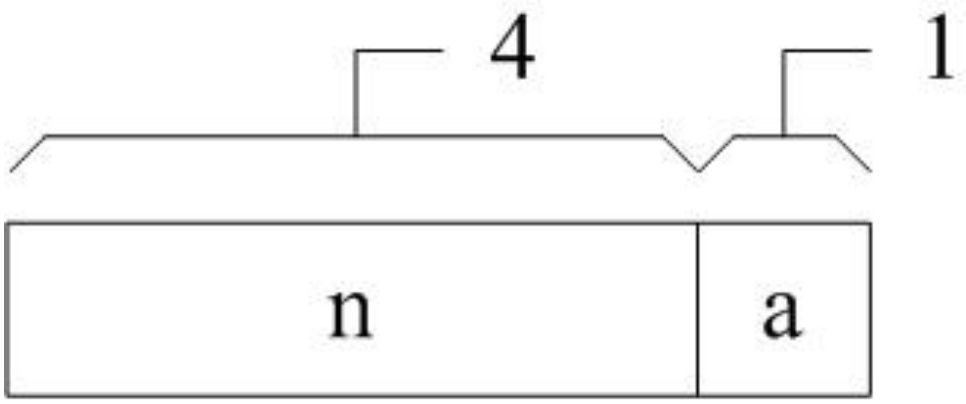


（a）UDATA

从SDATA类型内存结构，可以看出两个成员n和a是各自独立的。

```
struct SDATA { //结构体类型
    int n; //整型成员
    char a; //字符成员
}sa;
```

图8.5 共用体与结构体内存结构比较



（b）SDATA

8.6.1 共用体概念及类型定义

结构体与共用体的内存形式是截然不同的。共用体内存长度是所有成员内存长度的最大值，结构体内存长度是所有成员内存长度之和。建议用sizeof取它们的内存长度。

- 与结构体对象相似，定义共用体对象也有三种形式：

①先声明共用体类型再定义共用体对象

union 共用体类型名 共用体对象名列表;

②同时声明共用体类型和定义共用体对象

union 共用体类型名 { 成员列表 } 共用体对象名列表;

③直接定义共用体对象

union { 成员列表 } 共用体对象名列表;

定义共用体对象时可以进行初始化，但只能按一个成员给予初值，例如：

```
union A x={ 5678 }; //正确，只能给出1个初值
union A y={5,6,7,8}; //错误，试图给出4个初值（结构体做法）
```

8.6.3 共用体对象的使用

共用体对象的使用主要是引用它的成员，方法是对象成员引用运算（. ），例如：

```
1   x.m=5678; //给共用体成员赋值
2   cout<<x.m<<“,”<<x.n<<“,”

<<(int)x.a<<","<<(int)x.b<<endl;
3   cin>>x.m>>x.n>>x.a>>x.b;
4   x.n++; //共用体成员运算
```

第1句给成员m赋值5678，由于所有成员内存是共享的，因此每个成员都是这个值。

第2句输出m和n为5678，输出a和b为46，因为a和b类型为char，仅使用共享内存中的一部分（4个字节的低字节），即5678（0x162E）的0x2E（46）。

8.6.3 共用体对象的使用

```
1   x.m=5678; //给共用体成员赋值
2   cout<<x.m<<“,”<<x.n<<“,”

<<(int)x.a<<","<<(int)x.b<<endl;
3   cin>>x.m>>x.n>>x.a>>x.b;
4   x.n++; //共用体成员运算
```

同时每个成员的起始地址是相同的，当运行第3句时输入1 2 3 4↙，x.m得到1，但紧接着x.n得到2时，x.m也改变为2了（因为共享），依次类推，最终x.b得到4时，所有成员都是这个值。

第4句当x.n自增运算后，所有成员的值都改变了。

由于成员是共享存储空间的，使用共用体对象成员时有如下特点：

①修改一个成员会使其他成员发生改变，所有成员存储的总是最后一次修改的结果；

②所有成员的值是相同的，区别是不同类型决定使用这个值的全部或部分；

③所有成员的起始地址值是相同的，因此通常只按一个成员输入、初始化；

不能对共用体对象整体进行输入、输出、算术运算等操作，只能对它赋值操作。赋值实际上就是将一个对象的内容按内存形式完全复制到另一个对象中，例如：

```
union A one,two={1234};
one=1234; //错误，类型不兼容
one=two; //正确，赋值时复制two的内存数据到one中
```

可以得到共用体对象的地址。显然，该地址与各成员的地址值相同。可以定义指向共用体对象的指针，其指向类型应与共用体类型一致。通过指向共用体对象的指针访问成员与结构体相同，方法是指针成员引用运算（->）。例如：

```
A z={1234}, *p; //指针类型为union A*
p=&z; //指向共用体对象
cout<<p->a<<endl; //通过指针引用成员
```

可以定义共用体数组及指向共用体数组的指针，也可以在共用体中包含数组、指针成员。例如：

```
union C {
    int n[5]; //数组成员
    int *np; //指针成员
};
union C M[10], *p=M; //定义共用体数组，指向共用体数组的指针
M[0].n[1]=1; //引用数组成员元素
p->np=p->n+1;  // 通过指针引用成员，等价于M[0].np=&M[0].n[1];
```

8.6.3 共用体对象的使用

函数的参数和返回类型可以是共用体对象、共用体指针。对象参数传递和返回时均采用复制对象方式。通过地址方式仅传递地址，调用开销小。例如：

```
union A fun1(union A a,union A *p)
{
    a.a = a.a + p->a;
    return a;
}
```

- 枚举类型是由用户自定义的由多个命名枚举常量构成的类型，其声明形式为：

```
enum 枚举类型名 {命名枚举常量列表};
```

8.7.1 枚举类型的声明

```
enum DAYS {MON,TUE,WED,THU,FRI,SAT,SUN};
```

DAYS是枚举类型，MON等是命名枚举常量。默认时枚举常量总是从0开始，后续的枚举常量总是前一个的枚举常量加一。如MON为0，TUE为1，……，SUN为6。

可以在（仅仅在）声明枚举类型时，为命名枚举常量指定值。例如：

```
enum COLORS {RED=8,GREEN=10,BLUE,BLACK,WHITE};
```

则RED为8、GREEN为10、BLUE为11、BLACK为12、WHITE为13。

命名枚举常量是一个整型常量值，也称为枚举器（enumerator），在枚举类型范围内必须是唯一的。命名枚举常量是右值不是左值，例如：

```
RED=10; //错误，RED不是左值，不能被赋值
GREEN++; //错误，GREEN不是左值，不能自增自减
```

- 定义枚举类型对象有三种形式：

```
①enum 枚举类型名 {命名枚举量列表} 枚举对象名列表;
②enum 枚举类型名 枚举对象名列表;
//在已有枚举类型下，最常用的定义形式
③enum {命名枚举量列表} 枚举对象名列表;
//使用较少的定义形式
```

可以在定义对象时进行初始化，其形式为：

枚举对象名1=初值1， 枚举对象名2=初值2，……；

```
enum DIRECTION {LEFT,UP,RIGHT,DOWN,BEFORE,BACK} dir=LEFT;
```

当给枚举类型对象赋值时，若是除枚举值之外的其他值，编译器会给出错误信息，这样就能在编译阶段帮助程序员发现潜在的取值超出规定范围的错误。例如：

```
enum COLORS color;
color=101; //错误，不能类型转换
color=(COLORS)101; //正确，但结果没有定义
```


程序设计