C语言的补充概念

补.1：共用体

补.1 共用体

例：定义一个用于一卡通管理系统的结构，要求包含卡号、余额、消费限额、消费密码等公共信息，此外，若持卡人是学生，要包含学号、姓名、专业等学生特有的信息，若持卡人是教师，则包含工号、姓名、职称等教师特有的信息

```
struct student{
    定义学生信息;
};
struct teacher {
    定义教师信息;
};
struct ykt {
    公共信息;
    student sinfo;
    teacher tinto;
};
int main()
{   ykt y1;//定义变量
}
```

```
对y1的成员的访问：
int main()
{
    ykt y1;
    ...;
    y1.卡号
    y1.sinfo.学号
    y1.tinfo.工号
    ...;
    return 0;
}
```

能否使sinfo/tinfo共用一段空间：

1)当持卡人是学生时，这段空间按student方式访问

2)当持卡人是教师时， 按teacher方式访问

=>(共用体)

补.1 共用体

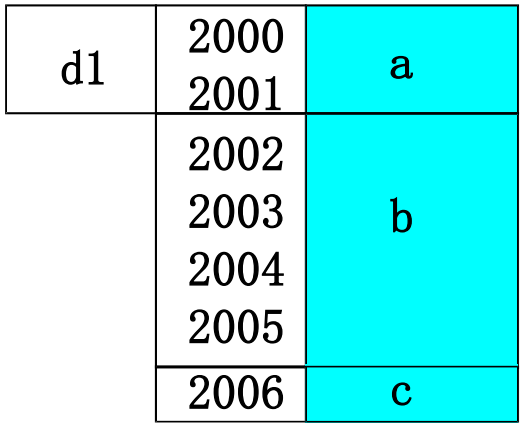
```
union 共用体名 {
    共用体成员1（类型名 成员名）
    ...
    共用体成员n（类型名 成员名）
};
```

```
union data {
    short a;
    long b;
    char c;
};
```

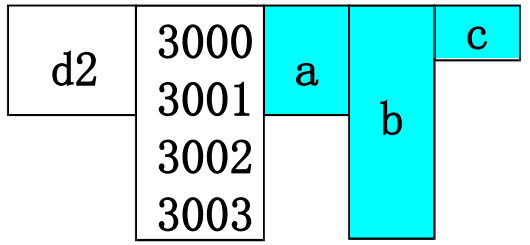
- 所有成员从同一内存开始，共用体的大小为其中占用空间最大的成员的大小
- 给一个共用体成员赋值后，会覆盖其它成员的值，因此只有最后一次存放的成员是有效的
- 其它所有定义、使用方法同结构体

```
#include <iostream>
using namespace std;
struct data1 {
    short a;
    long b;    //12:所有成员所占空间之和(含填充字节)
    char  c;
};
union data2 {
   short a;
   long b;     //4 :所有成员中最大成员所占空间
   char  c;
};
int main()
{    cout << sizeof(data1) << ' ' << sizeof(data2) << endl;  //12 4
}
```

struct data1 d1;

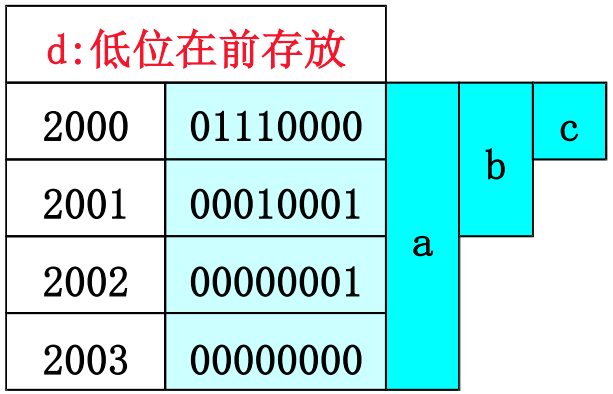


union data2 d2;



```
#include <iostream>
using namespace std;
union data {
    int a;
    short b;
    char  c;
};
int main()
{
     union data d;
     d.a=70000;
     cout << d.a << ' ' << d.b << ' ' << d.c << endl;
     d.b=7000;
     cout << d.a << ' ' << d.b << ' ' << d.c << endl;
     d.c='A';
     cout << d.a << ' ' << d.b << ' ' << d.c << endl;
     return 0;
}
```

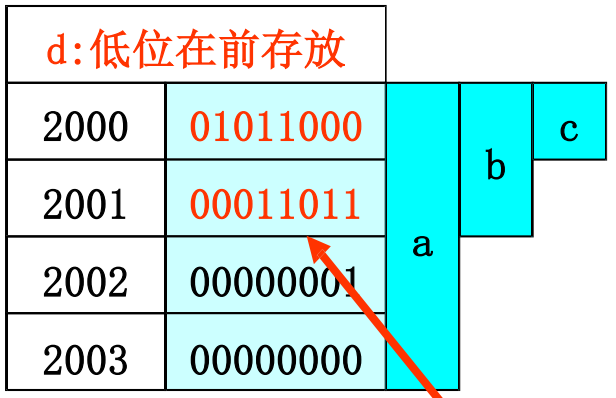
70000=00000000 00000001 00010001 01110000




```
70000 4464 p
72536 7000 X
72513 6977 A
```

```
#include <iostream>
using namespace std;
union data {
    int a;
    short b;
    char  c;
};
int main()
{
     union data d;
     d.a=70000;
     cout << d.a << ' ' << d.b << ' ' << d.c << endl;
     d.b=7000;
     cout << d.a << ' ' << d.b << ' ' << d.c << endl;
     d.c='A';
     cout << d.a << ' ' << d.b << ' ' << d.c << endl;
     return 0;
}
```

72536=00000000 00000001 00011011 01011000



7000=00011011 01011000

70000 4464 p

72536 7000 X

72513 6977 A

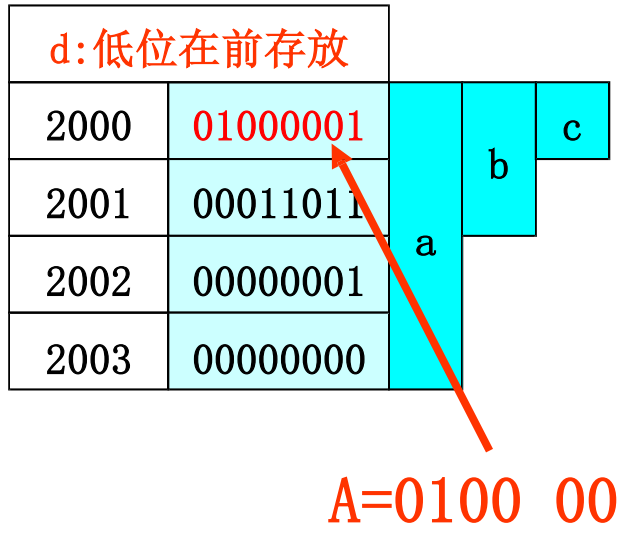
```
#include <iostream>
using namespace std;
union data {
    int a;
    short b;
    char  c;
};
int main()
{
     union data d;
     d.a=70000;
     cout << d.a << ' ' << d.b << ' ' << d.c << endl;
     d.b=7000;
     cout << d.a << ' ' << d.b << ' ' << d.c << endl;
     d.c='A';
     cout << d.a << ' ' << d.b << ' ' << d.c << endl;
     return 0;
}
```

70000 4464 p

72536 7000 X

72513 6977 A

72513=00000000 00000001 00011011 01000001



• 所有成员从同一内存开始，共用体的大小为其中占用空间最大的成员的大小

```
#include <iostream>
using namespace std;
union data {
 int a;
 short b;
 char c;
};
int main()
{  union data d;
   d.c='A';
   cout << d.a << ' ' << d.b << ' ' << d.c << endl;  //不确定 不确定 A
   d.b=7000;
   cout << d.a << ' ' << d.b << ' ' << d.c << endl;  //不确定 7000 X
   d.a=70000;
   cout << d.a << ' ' << d.b << ' ' << d.c << endl;  //70000 4464 p
   return 0;
}
```

d.c='A'	
2000	01000001
2001	???
2002	???
2003	???

d.b=7000	
2000	01011000
2001	00011011
2002	???
2003	???

d.a=70000	
2000	01110000
2001	00010001
2002	00000001
2003	00000000

补.1 共用体

```
struct student {
  定义学生信息
};
struct teacher {
  定义教师信息;
};
struct ykt {
  公共信息;
  student sinfo;
  teacher tinto;
};
int main()
{  ykt y1;//定义变量
}
```

空间浪费

```
struct student {
  定义学生信息;
};
struct teacher {
  定义教师信息;
};
union owner {
  student s;
  teacher t;
};
struct ykt {
  公共信息;
  char type; //持卡人类别
  owner info;
};
```

此处保证s/t共用一段空间

```
int main()
{
  ykt y1;//定义变量
  ...;
  y1.卡号...;
  if (y1.type=='s'){
     y1.info.s.学号;
  }
  else{
     y1.info.t.工号;
  }
  ...;
  return 0;
}
```

C语言的补充概念

补.2：位运算

补.2 位运算

2.1 概述

- 字节和位

  字节：byte，计算机中数据表示的基本单位

  位 ：bit， 计算机中数据表示的最小单位

  1 byte = 8 bits

- 位运算

  以bit为单位进行数据的运算

补.2 位运算

2.1 概述

- 位运算的基本方法
  - 按位进行（只有0、1）
  - 要求运算数据长度相等，若不等，则右对齐，按最高位补齐左边

```
char  a=0x37;        0000 0000 0011 0111
short b=0x1234;      0001 0010 0011 0100
char  a=0xA7;        1111 1111 1010 0111
short b=0x8341;      1000 0011 0100 0001
```

  - 数在计算机内是用补码表示的

补.2 位运算

2.2 常用的位运算

运算符	功能	用法
~	位求反	~expr
<<	左移	expr1 << expr2
>>	右移	expr1 >> expr2
&	位与	expr & expr
^	位异或	expr ^ expr
|	位或	expr | expr

补.2 位运算

- 移位运算符
  - 左移运算规则：左移数据，右补0
  - 右移运算规则：右移数据，左补0（逻辑右移）

    右移数据，左补符号位（算术右移）- C/C++的位运算

例： unsigned char bits=0233;

1	0	0	1	1	0	1	1

bits << 8; //bits提升成int类型，然后向左移动8位

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0

bits << 31; //向左移动31位，左边超出边界的位丢弃

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

bits >> 3; //向右移动3位，最右边的3位丢弃

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
        unsigned char bits = 0233;
        cout << "bits<<8=0x" << hex << (bits << 8) << " "
                             << dec << (bits << 8) << endl;
        cout << "bits<<31=0x"<< hex << (bits << 31)<< " "
                             << dec << (bits << 31)<< endl;
        cout << "bits>>3=0x" << hex << (bits >> 3) << " "
                             << dec << (bits >> 3) << endl;

        return 0;
}
```

```
bits<<8=0x9b00 39680
bits<<31=0x80000000 -2147483648
bits>>3=0x13 19
```

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    unsigned char bits = 0233;
    cout << "bits<<8=0x" << hex << (int)(char)(bits << 8) << " "
                         << dec << (int)(char)(bits << 8) << endl;
    cout << "bits<<31=0x"<< hex << (int)(char)(bits << 31) << " "
                         << dec << (int)(char)(bits << 31) << endl;
    cout << "bits>>3=0x" << hex << (int)(char)(bits >> 3) << " "
                         << dec << (int)(char)(bits >> 3) << endl;
    return 0;
}
```

```
bits<<8=0x0 0
bits<<31=0x0 0
bits>>3=0x13 19
```

//蓝框部分不必要，可省略

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
        char bits = 0233;  //提升为int类型时，填充的是符号位
        cout << "bits<<8=0x" << hex << (bits << 8) << " "
                             << dec << (bits << 8) << endl;
        cout << "bits<<31=0x"<< hex << (bits << 31) << " "
                             << dec << (bits << 31) << endl;
        cout << "bits>>3=0x" << hex << (bits >> 3) << " "
                             << dec << (bits >> 3) << endl;
        return 0;
}
```

```
bits<<8=0xffff9b00 -25856
bits<<31=0x80000000 -2147483648
bits>>3=0xfffffff3 -13
```

补.2 位运算

- 移位运算符
  - 左移：在不溢出(1不被舍去)的情况下，左移n位等于乘2的n次方(当做无符号数理解)
  - 右移：在不溢出(1不被舍去)的情况下，右移n位等于除2的n次方(当作有符号数理解)

补.2 位运算

- 移位运算符
  - 重载版本--IO运算符
  - 满足左结合律：

```
cout << 42 + 10; //正确：+优先级高，输出求和结果

cout << (10 < 42); //正确：输出1

cout << 10 < 42;  //错误：把数字10写到cout，然后将结果
                  （即cout）与42进行比较
```

补.2 位运算

- 位求反运算符
  - 运算规则：0/1互反

例： unsigned char bits = 0227;

1	0	0	1	0	1	1	1

~bits

//提升成int类型，原来位保持不变，高位添加0：

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	1

//逐位求反：

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	0	0	0

补.2 位运算

- 位求反运算符
  - 运算对象：整数类型

例：求表达式的值 ~'q' << 6

//按运算符优先级，先对q按位取反：'q'转换为整数：

0	1	1	1	0	0	0	1

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1

//逐位求反：

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0

//移位操作：

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0

补.2 位运算

- 位与、位或、位异或运算符
  - 位与运算规则：遇0得0
  - 位或运算规则：遇1得1
  - 位异或运算规则：相同为0，不同为1

例：

```
unsigned char b1 = 0145;
unsigned char b2 = 0257;
b1 & b2
b1 | b2
b1 ^ b2
```

	0	1	1	0	0	1	0	1
	1	0	1	0	1	1	1	1
24个高阶位都是0	0	0	1	0	0	1	0	1
24个高阶位都是0	1	1	1	0	1	1	1	1
24个高阶位都是0	1	1	0	0	1	0	1	0

补.2 位运算

- 位与、位或、位异或运算符
  - 位运算符：　&　　|　　^
  - 逻辑运算符：&&　　||　　!

例：　unsigned long ul1 = 3, ul2 = 7;

```
(1) ul1 & ul2;          //3，占4个字节
(2) ul1 | ul2;          //7，占4个字节
(3) ul1 && ul2;         //true，占1个字节
(4) ul1 || ul2;         //true，占1个字节
```

补.2 位运算

- 复合位运算符
  - &=　　|=　　^=　　<<=　　>>=

```
...
int main()
{
    char a=0x18;
    int i;
    for(i=1; i<=6; i++) {
        a = a>>1;     //a >>= 1;
        cout << "a>>" << i << "=0x"  << hex << int(a) << " "
                                     << dec << int(a) << endl;
    }
}
```

```
a>>1=0xc 12
a>>2=0x6 6
a>>3=0x3 3
a>>4=0x1 1
a>>5=0x0 0
a>>6=0x0 0
```

补.2 位运算

2.3 位运算的应用

- 与(&)的应用
  - 清零

```
char a1=0xb6;
cout << "char a=" << hex << (int)a1 << endl
     << " a&0x0=" << dec << (a1&0x0) << endl;
```

例：char a1=0xb6;　现要求将该数清零，则：

```
  1011 0110
& 0?00 ?00?
-----------
  0000 0000
```

要清零数为1的位，本数对应位为0

a&0x0　a&0x1　a&0x8　a&0x9

a&0x40　a&0x41　a&0x48　a&0x49

补.2 位运算

2.3 位运算的应用

- 与(&)的应用
  - 取指定位

例：char a2=0xb6；现要求只保留低4位，而高4位清0，则：

1011 0110

& 0000 1111　　要保留的位，本数对应位为1

0000 0110

```
char a2=0xb6;
cout << "char a=0x" << hex << (int)a2
     << "  a&0x0F=" << dec << (a2&0x0F) << endl;
```

补.2 位运算

2.3 位运算的应用

- 或(|)的应用
  - 设定某些位为1

例：char a=0xb6；要求1、4位设为1，其它不变

1011 0110

| 0000 1001

1011 1111 (0xBF)

```
char a=0xb6;
cout << "a=" << hex << (int)a
     << " a|0x9=0x" << (a|0x9) << endl;
```

补.2 位运算

2.3 位运算的应用

- 异或(^)的应用
  - 特定位翻转（0，1互换）

例：char a1=0xb6; 高4位翻转，低4位不变

1011 0110

^ 1111 0000　　要翻转的位，本数对应位为1

0100 0110

```
char a1=0xb6;
cout << "a=" << hex << (int)a1
     << " a^0xF0=0x" << (a1^(char)0xF0) << endl;
```

补.2 位运算

2.3 位运算的应用

- 异或(^)的应用
  - 两数交换

```
char a=0xb6, b=0xc2;
cout << "a=" <<hex<<(int)a<<" b=" <<(int)b<< endl;
a = a^b; b = b^a; a = a^b;
cout << "a=" <<hex<<(int)a<<" b=" <<(int)b<< endl;
```

例：char a=0xb6,b=0xc2; 要求a，b互换

(1) a=1011 0110

b=1100 0010

a=0111 0100　　a=a^b=0x74

(2) b=1100 0010

a=0111 0100

b=1011 0110　　b=b^a=0xb6

(3) a=0111 0100

b=1011 0110

a=1100 0010　　a=a^b=0xc2

补.2 位运算

2.3 位运算的应用

综合应用例：班级有30个学生，老师每周都会对学生进行一次小测验，结果只有通过和不通过两种。为了更好地追踪测验的结果，我们用一个二进制位代表某个学生在一次测验中是否通过，显然全班的测验结果可以用一个无符号整数来表示：

```
unsigned long quiz1 = 0;    //位的集合
```

对序号为27的学生对应的位进行设置，以表示其通过了测验：

```
1UL << 27;    //生成一个值，该值只有第27位为1
quiz1 |= 1UL << 27;  //表示学生27通过了测验
```

补.2 位运算

2.3 位运算的应用

重新核对发现学生27实际上没有通过测试：

```
quiz1 &= ~(1UL << 27);  //学生27没有通过测验
```

检查学生27的测验情况：

```
bool status = quiz1 & (1UL << 27);  //学生27是否通过了测验?
```

C语言的补充概念

补.3：带参数的main函数

补.3 带参数的main函数

3.1 引入

- 可执行文件运行时，目前只能简单的运行，如能加上参数，使用中更灵活

3.2 方法

- 带参数的main函数的定义形式

```
int main(int argc, char **argv)
int main(int argc, char *argv[])
```

两者均可

- 参数解释

  argc：参数的个数，若不带参数，则为1(自身)

  argv：参数的内容，用指针数组表示，每个元素是一个字符串(char *)，最后一个是NULL

- 参数名argc/argv可变，类型不能变（例如：int ac, char **av）

补.3 带参数的main函数

- argv[0]为该可执行文件的文件名（含目录）

```
//集成环境运行
#include<iostream>
using namespace std;
int main(int argc, char* argv[])
{
    int i;
    cout << "当前文件的路径：" << argv[0] << endl;
    for (i = 1; i < argc; i++){
        cout << i << "," << argv[i] << endl;
    }
    return 0;
}
```

只输出该可执行文件的路径，即argv[0]中存储的字符串。

```
当前文件的路径：D:\test\Debug\demo.exe
```

补.3 带参数的main函数

- argv[0]为该可执行文件的文件名（含目录）

```
//命令行运行
#include<iostream>
using namespace std;
int main(int argc, char* argv[])
{
    int i;
    cout << "当前文件的路径：" << argv[0] << en
    for (i = 1; i < argc; i++){
        cout << i << "," << argv[i] << endl;
    }
    return 0;
}
```

假设编译后形成demo.exe，运行：

```
demo
demo hello world
demo who are you
```

```
D:\test\Debug>demo
当前文件的路径：demo

D:\test\Debug>demo hello world
当前文件的路径：demo
1,hello
2,world

D:\test\Debug>demo who are you
当前文件的路径：demo
1,who
2,are
3,you
```

```
//两数交换
#include <iostream>
#include <cstdlib> //atoi函数用到
using namespace std;
int main(int argc, char *argv[])
{   int a, b, t;
    if (argc<3) { /* 参数不足3个则出现提示 */
       cerr << "请带两个整数作为参数"<< endl;
       return -1;
    }
    for (t=0; t<argc; t++) /* 打印所有的参数值 */
       cout << "argv[" << t << "]=" << argv[t] << endl;
    a = atoi(argv[1]); //atoi是将字符串转为整数的函数
    b = atoi(argv[2]);
    cout << "交换前：a=" << a << "  b=" << b << endl;
    t = a; a = b; b = t;
    cout << "交换后：a=" << a << "  b=" << b << endl;
    return 0;
}
```

1.

```
请带两个整数作为参数
```

假设编译后形成demo.exe

1、集成环境运行

2、命令行运行

```
demo
demo 10
demo 10 15
demo 10 15 20
```

2.

```
D:\test\Debug>demo
请带两个整数作为参数

D:\test\Debug>demo 10
请带两个整数作为参数

D:\test\Debug>demo 10 15
argv[0]=demo
argv[1]=10
argv[2]=15
交换前：a=10  b=15
交换后：a=15  b=10

D:\test\Debug>demo 10 15 20
argv[0]=demo
argv[1]=10
argv[2]=15
argv[3]=20
交换前：a=10  b=15
交换后：a=15  b=10
```

补.3 带参数的main函数

- 例：高程大作业-文件压缩小程序

» 要求使用cmdline方式读取参数，参数格式为 压缩文件名 输出文件名 压缩指令(zip/(unzip，选做))

```
int main(int argc, char* argv[]) {
    cout << "Zipper 0.001! Author: root" << endl;
    if (argc != 4) {
        cerr << "Please make sure the number of parameters is correct." << endl;
        return -1;
    }

    if (strcmp(argv[3], "zip"
        cerr << "Unknown para
        return -1;
    }
```

```
命令提示符
Microsoft Windows [版本 10.0.19042.867]
(c) 2020 Microsoft Corporation. 保留所有权利。

C:\Users\april>cd C:\Users\april\source\repos\Gaocheng\Debug

C:\Users\april\source\repos\Gaocheng\Debug>demo C:\Users\april\source
\repos\Gaocheng\ser.log C:\Users\april\source\repos\Gaocheng\ser_comp
ressed.log zip
Zipper 0.001! Author: root
Complete!
```

补.3 带参数的main函数

- 带参数的main函数的扩展形式(仅了解)

形式：　　int main(int argc, char **argv, char **env)

或：　char *env[]

参数解释：

- argc：同前
- argv：同前
- env ：操作系统的环境变量，用指针数组来表示，每个元素是一个字符串(char *)，最后一个元素是NULL

补.3 带参数的main函数

- 带参数的main函数的扩展形式(仅了解)

使用：需要判断/取操作系统的某些设置时才用到

```
//取操作系统的环境变量
#include <iostream>
using namespace std;
int main(int argc, char **argv, char **env)
{
   int i;
   for (i=0; env[i]; i++)
      cout<< "env[" << i << "]=" << env[i] <<endl;
   return 0;
}
```