第20讲：自定义类型：联合和枚举

目录

1. 联合体类型的声明
2. 联合体的特点
3. 联合体大小的计算
4. 枚举类型的声明
5. 枚举类型的优点
6. 枚举类型的使用

正文开始

1. 联合体

1.1 联合体类型的声明

像结构体一样，联合体也是由一个或者多个成员构成，这些成员可以不同的类型。

但是编译器只为最大的成员分配足够的内存空间。联合体的特点是所有成员共用同一块内存空间。所以联合体也叫：**共用体**。

给联合体其中一个成员赋值，其他成员的值也跟着变化。

```c
#include <stdio.h>

//联合类型的声明
union Un
{
    char c;
    int i;
};

int main()
{
    //联合变量的定义
    union Un un = {0};
    //计算连个变量的大小
    printf("%d\n", sizeof(un));
```

```
16
17      return 0;
18  }
```

输出的结果：

```
1  4
```

为什么是4呢？

1.2 联合体的特点

联合的成员是共用同一块内存空间的，这样一个联合变量的大小，至少是最大成员的大小（因为联合至少得有能力保存最大的那个成员）。

```
1  //代码1
2  #include <stdio.h>
3
4  //联合类型的声明
5  union Un
6  {
7      char c;
8      int i;
9  };
10
11 int main()
12 {
13     //联合变量的定义
14     union Un un = {0};
15     // 下面输出的结果是一样的吗?
16     printf("%p\n", &(un.i));
17     printf("%p\n", &(un.c));
18     printf("%p\n", &un);
19     return 0;
20 }
```

```
1  //代码2
2  #include <stdio.h>
3
4  //联合类型的声明
5  union Un
6  {
7      char c;
8      int i;
9  };
10
11 int main()
12 {
13     //联合变量的定义
14     union Un un = {0};
15     un.i = 0x11223344;
16     un.c = 0x55;
17     printf("%x\n", un.i);
18     return 0;
19 }
```

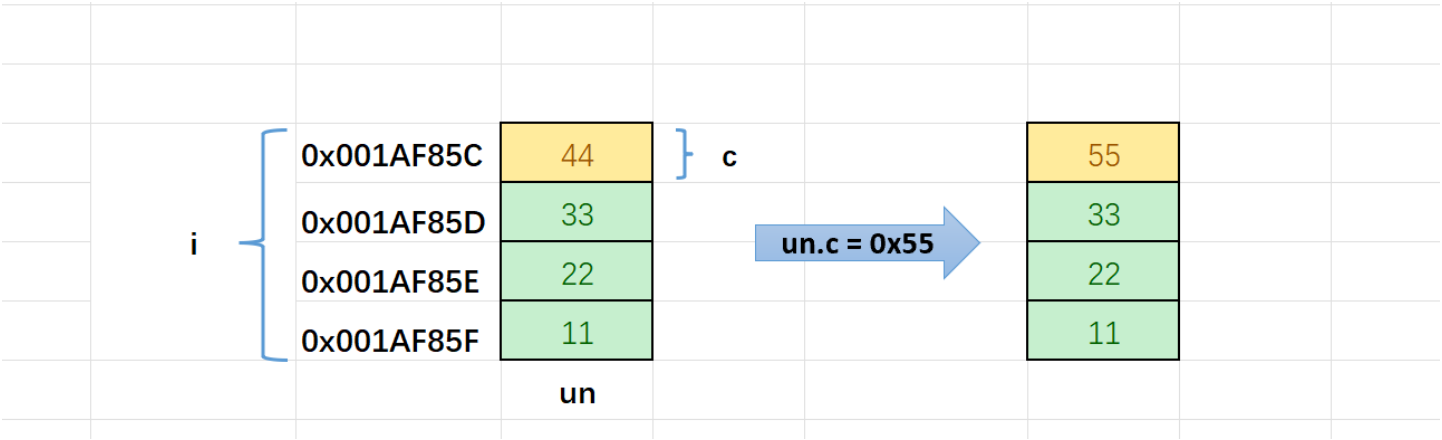
输出的结果：

```
1  001AF85C
2  001AF85C
3  001AF85C
```

```
1  11223355
```

代码1输出的三个地址一模一样，代码2的输出，我们发现将i的第4个字节的内容修改为55了。

我们仔细分析就可以画出，un的内存布局图。



1.3 相同成员的结构体和联合体对比

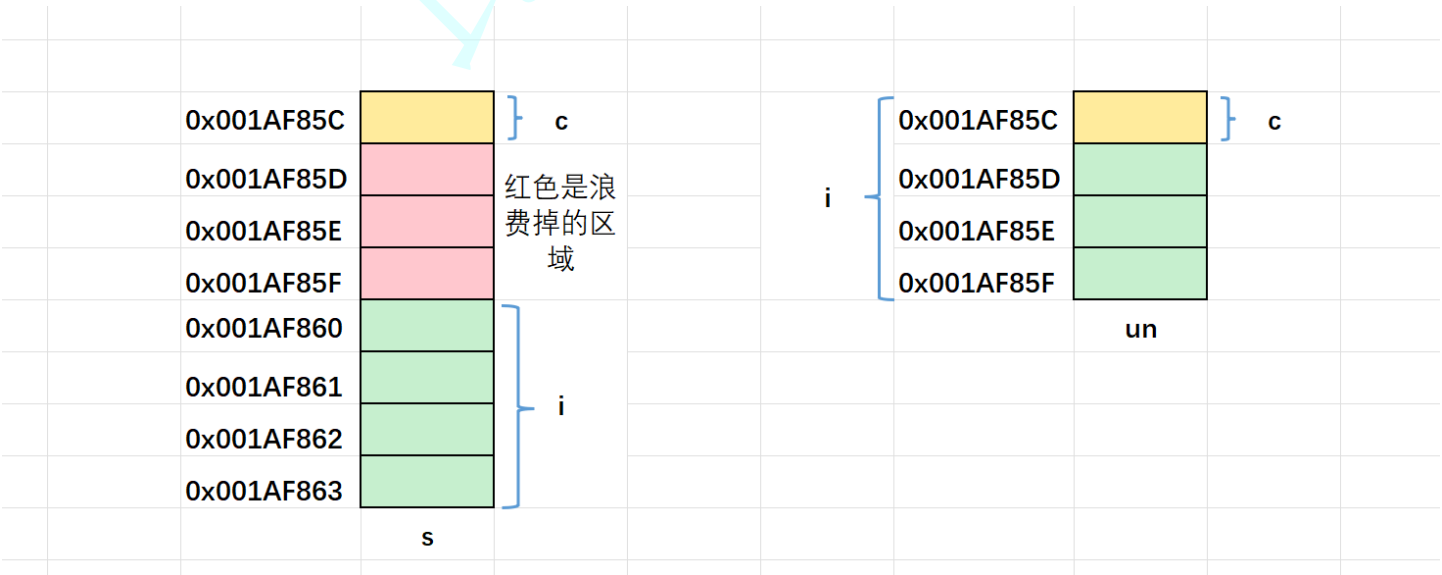
我们再对比一下相同成员的结构体和联合体的内存布局情况。

```c
struct S
{
    char c;
    int i;
};

struct S s = {0};
```

```c
union Un
{
    char c;
    int i;
};

union Un un = {0};
```



结构体和联合体的内存对比

1.4 联合体大小的计算

- 联合的大小至少是最大成员的大小。
- 当最大成员大小不是最大对齐数的整数倍的时候，就要对齐到最大对齐数的整数倍。

```c
#include <stdio.h>
union Un1
{
    char c[5];
    int i;
};

union Un2
{
    short c[7];
    int i;
};
int main()
{
    //下面输出的结果是什么?
    printf("%d\n", sizeof(union Un1));
    printf("%d\n", sizeof(union Un2));
    return 0;
}
```

使用联合体是可以节省空间的，举例：

比如，我们要搞一个活动，要上线一个礼品兑换单，礼品兑换单中有三种商品：图书、杯子、衬衫。

每一种商品都有：库存量、价格、商品类型和商品类型相关的**其他信息**。

> 图书：书名、作者、页数
>
> 杯子：设计
>
> 衬衫：设计、可选颜色、可选尺寸

那我们不耐心思考，直接写出一下结构：

```c
struct gift_list
{
    //公共属性
    int stock_number;//库存量
    double price; //定价
    int item_type;//商品类型

    //特殊属性
```

```
    char title[20];//书名
    char author[20];//作者
    int num_pages;//页数

    char design[30];//设计
    int colors;//颜色
    int sizes;//尺寸
};
```

上述的结构其实设计的很简单，用起来也方便，但是结构的设计中包含了所有礼品的各种属性，这样使得结构体的大小就会偏大，比较浪费内存。因为对于礼品兑换单中的商品来说，只有部分属性信息是常用的。比如：

商品是图书，就不需要design、colors、sizes。

所以我们就可以把公共属性单独写出来，剩余属于各种商品本身的属性使用联合体起来，这样就可以介绍所需的内存空间，一定程度上节省了内存。

```
struct gift_list
{
    int stock_number;//库存量
    double price; //定价
    int item_type;//商品类型

    union {
        struct
        {
            char title[20];//书名
            char author[20];//作者
            int num_pages;//页数
        }book;
        struct
        {
            char design[30];//设计
        }mug;
        struct
        {
            char design[30];//设计
            int colors;//颜色
            int sizes;//尺寸
        }shirt;
    }item;
};
```

1.5 联合的一个练习

写一个程序，判断当前机器是大端？还是小端？

```c
int check_sys()
{
    union
    {
        int i;
        char c;
    }un;
    un.i = 1;
    return un.c;//返回1是小端，返回0是大端
}
```

2. 枚举类型

2.1 枚举类型的声明

枚举顾名思义就是一一列举。

把可能的取值一一列举。

比如我们现实生活中：

> 一周的星期一到星期日是有限的7天，可以一一列举
>
> 性别有：男、女、保密，也可以一一列举
>
> 月份有12个月，也可以一一列举
>
> 三原色，也是可以意义列举

这些数据的表示就可以使用枚举了。

```c
enum Day//星期
{
    Mon,
    Tues,
    Wed,
    Thur,
    Fri,
    Sat,
    Sun
};
```

```
enum Sex//性别
{
    MALE,
    FEMALE,
    SECRET
};
enum Color//颜色
{
    RED,
    GREEN,
    BLUE
};
```

以上定义的`enum Day`，`enum Sex`，`enum Color`都是枚举类型。

{}中的内容是枚举类型的可能取值，也叫`枚举常量`。

这些可能取值都是有值的，默认从0开始，依次递增1，当然在声明枚举类型的时候也可以赋初值。

```
enum Color//颜色
{
    RED=2,
    GREEN=4,
    BLUE=8
};
```

2.2 枚举类型的优点

> 为什么使用枚举?

我们可以使用`#define`定义常量，为什么非要使用枚举?

枚举的优点：

1. 增加代码的可读性和可维护性
2. 和#define定义的标识符比较枚举有类型检查，更加严谨。
3. 便于调试，预处理阶段会删除`#define`定义的符号
4. 使用方便，一次可以定义多个常量
5. 枚举常量是遵循作用域规则的，枚举声明在函数内，只能在函数内使用

2.3 枚举类型的使用

```
enum Color//颜色
{
    RED=1,
    GREEN=2,
    BLUE=4
};

enum Color clr = GREEN;//使用枚举常量给枚举变量赋值
```

那是否可以拿整数给枚举变量赋值呢？在C语言中是可以的，但是在C++是不行的，C++的类型检查比较严格。

完