67 Unions in C++

1. 联合体

Union （联合体）有点像class类型或者struct类型，只不过它一次只能占用一个成员的内存。
通常如果我们有一个结构体，我们在里面声明4个浮点数，就可以有4x4个字节在这个结构体中，总共是16个字节。
但一个联合体只能有一个成员，所以如果我要声明4个浮点数，比如abcd，联合体的大小仍然是4个字节，当我尝试去处理它们，比如将a设为5，它们的内存是一样的，d的值也会是5，这就是联合体的工作方式。

你可以像使用结构体或类一样使用它们，也可以给它添加静态函数或者普通函数、方法等。然而你不能使用虚方法，还有一些其它限制，但通常人们用联合体来做的事，是和类型双关紧密相关的。当你想给同一个变量取两个不同的名字时，它真的很好用。

通常`union`是匿名使用的，但匿名union不能含有成员函数。

1. 使用案例

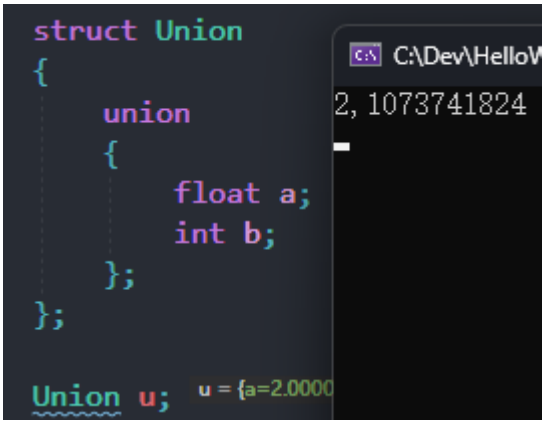
我们这里有两种不同的方法来处理相同的内存：

a. 解引用指针

C++

```
struct Union
{
    union
    {
        float a;
        int b;
    };
};

Union u;
u.a = 2.0f;
std::cout << u.a << "," << u.b << std::endl;
```

得到2和一串数，其实这个107...是浮点数形式的2的字节表示，就好像我们取了组成浮点数的内存，然后把它解释成一个整型，这样就是类型双关了。



看一个更有用的例子：

```C++
struct Vector2
{
    float x, y;
};

struct Vector4
{
    float x, y, z, w;
};

void PrintVector(const Vector2& vector)
{
    std::cout << vector.x << "," << vector.y << std::endl;
}
```

可以发现Vector4实际上是两个Vector2，我们可以把x的内存地址转换为Vector2再解引用：

```C++
struct Vector4
{
    float x, y, z, w;

    Vector2& GetA()
    {
        return *(Vector2*)&x;
    }
};

int main()
{
    Vector4 e={1,2,3,4};
    std::cout << e.GetA().y << std::endl;  // 2
    std::cin.get();
}
```

b. union

还有另外一种方法，就是使用`union`：

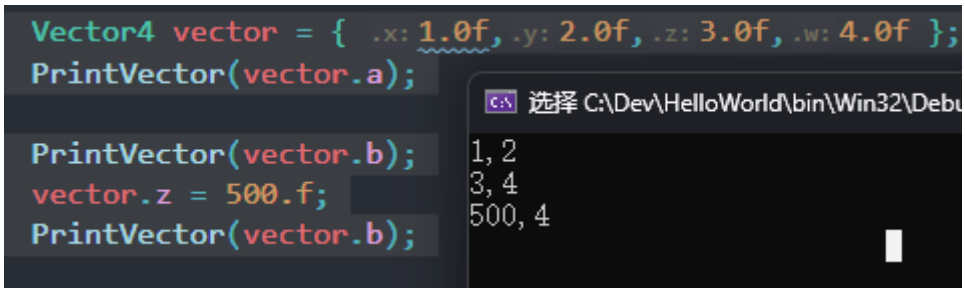
```C++
struct Vector4
{
    union
    {
        struct
        {
            float x, y, z, w;
        };
    };
};
// 这里还可以正常访问Vector4.x，因为我们没有给结构体起名，它是匿名的，只是一种数据结构
```

再加入一个结构体成员：

```C++
struct Vector4
{
    union
    {
        struct
        {
            float x, y, z, w;
        };

        struct
        {
            Vector2 a, b;
        };
    };
};
```

这里a和x，y的内存是一样的，b和z，w的内存相同，这里有两种方法可以读取：

```C++
int main()
{
    Vector4 vector = { 1.0f,2.0f,3.0f,4.0f };
    PrintVector(vector.a);

    PrintVector(vector.b);
    vector.z = 500.f;
    PrintVector(vector.b);

    std::cin.get();
}
```



这里并没有设置b.x=500，而是设置的vecor.z，这个z变量对应于b.x，因为它占用了相同的内存，所以z对应Vector2的x。

3. 总结

当你想做这样的事情时，`union`真的很有用：当你想用多种方法来处理相同的数据时。
你也可以用类型双关或者其它方法，但是通常`union`的可读性更强。