46 The Arrow Operator in C++

1. 使用箭头操作符

如果有一个Entity类型的指针ptr，我们不能直接用`ptr.Print()`来调用方法，因为这只是一个指针，也就是一个数值（不是对象，可谈调用方法）。
我们能做的是使用 *arrow operator*（箭头操作符），而不是用相对笨重的解引用指针再调用方法。

```C++
class Entity
{
public:
    void Print() const { std::cout << "Hello!" << std::endl; }
};


int main()
{
    Entity e;
    e.Print();
    Entity* ptr = &e;
    ptr->Print();

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

这个箭头，实际上就相当于解引用了Entity指针变成了一个普通的Entity类，然后调用Print：

```C++
Entity* ptr = &e;
//Entity& entity = *ptr;
(*ptr).Print();    //*ptr.Print();不行，因为有运算符优先级，会先调用Print，然后再解引用整体
```

因这就是一个快捷方式，本身我们需要手动解引用，然后用圆括号把它们括起来，现在用一个箭头就够了，对待类中的变量也可以这么做。

```C++
ptr->x = 2;  // 类中有一个public的int x成员变量
```

这基本上就是箭头操作符的默认用法了，你90%以上使用它的情况都是这样。
不过作为一个*Operator*（操作符、运算符），C++实际上可以重载它，并在你自己的自定义类中使用它。

2. 箭头操作符重载

下面新建一个作用域指针类：

```C++
class ScopedPtr
{
private:
    Entity* m_Obj;
public:
    ScopedPtr(Entity* entity)
        : m_Obj(entity)
    {

    }

    ~ScopedPtr()
    {
        delete m_Obj;
    }

};

//我可以这么使用它
int main()
{
    ScopedPtr entity = new Entity();

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

如果我希望能调用Entity类中的Print函数怎么弄？我不能用`.`，但是我可以把m_Obj变成public的或者用什么办法返回m_Obj，但这些方法都太乱了。我希望能够像使用堆分配的Entity一样使用它。
这时候就可以重载箭头函数来实现了：

```C++
class ScopedPtr
{
......
public:
    ......
    Entity* operator->()
    {
        return m_Obj;
    }
};

int main()
{
    ScopedPtr entity = new Entity();
    entity->Print();  //现在可以编译并正常运行

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

3. Bonus

最后补充一个小彩蛋。

```C++
struct Vector3
{
    float x, y, z;
};
```

我有一个结构体Vector3，我想得到x，y，z的偏移量（x是结构体第一项，偏移量是0；y是第二项，float是4个字节，所以偏移量是4个字节；z的偏移量则是8个字节）

可以用箭头运算符来做到，我想访问这些变量，但不是通过有效的内存地址，而是地址从0开始。

```C++
int offset =(int) & ((Vector3*)0)->x; // x,y,z
std::cout << offset << std::endl;  // 0,4,8
```

1. `(Vector3*)0` 将整数 `0` 转换为指向 `Vector3` 类型的指针。这样的操作通常用于获取一个空指针所指向类型的结构体或类的成员的偏移量。`((Vector3*)nullptr)`
2. `&((Vector3*)0)->x` 获取指向 `Vector3` 类型空指针所指向对象的 `x` 成员的地址。由于空指针无法访问成员变量，这里只是用于获取成员变量的偏移量。
3. `(int)` 将获取到的成员变量地址转换为整数类型，以便输出到流中。