Object-Oriented Programming

Akhia[1]

2020 年 9 月 28 日

[1]E-mail:akhialomgir362856@gmail.com

目录

Chapter 1

函数

1.1 引用

引用用 & 标记，用来为储存器提供**别名**。

```
int x;
int& ref = x;
```

分配了一个 int 单元，它拥有两个名字：x, ref。

1.1.1 引用调用

对函数参数用 & 作为引用参数，将获得**引用调用**。在引用调用中，引用参数将实际的实参传给函数，而不是实参的拷贝。而默认的调用方式为传值引用。

```
void swap(int& a,int& b){
    int t;
    t = a;
    a = b;
    b = t;
}
```

使用后，swap 函数的参数直接对应 main 中使用的 i 和 j 的储存空间，而不是拷贝。

1.1.2 引用返回

```
return expression;
```

默认情况下，函数返回时， expression 被求值，并将值拷贝到临时储存空间，即传值返回。

而引用返回中，返回值不再拷贝到临时储存空间，甚至 return 语句所用的储存单元对调用者而言都是可访问的。

```
int& val(){
    return i;
```

```
3       }
```

函数中返回引用 i 的引用，直接将值传入到调用者中。与传值返回不同，仅产生一个副本。

1.2 内联函数

关键字 **inline** 在函数**声明**（不能出现在定义部分）用来请求将函数用内联方式展开，即在每个调用函数的地方插入函数实现代码。

1.3 函数默认参数

所有没有默认值的参数都要放在函数列表的开始部分，然后才是有默认值的函数。

1.4 函数重载

在**参数个数或类型**有区别时，同一范围内允许使用相同签名的函数，称函数名被重载。重载函数用来对具有相似行为而数据类型不同的操作提供一个通用名称。

1.4.1 函数签名

重载函数有不同的**函数签名**：

1. 函数名
2. 参数个数、数据类型和顺序

1.5 new delete 操作符

1. new: 分配一个单元

2. new[]: 分配多个单元（数组）

3. delete: 释放 new 分配的单元

4. delete[]: 释放 new[] 分配的单元

new[] 分配了数组后，将第一个单元的地址保存到指向分配的储存空间的指针中。

1.6 异常处理

关键字：

1. try

2. catch

3. throw

catch 块定义在 try 块之后，因为例外在 try 块中抛出。例外和不同捕捉器的匹配依靠**类型**判断来进行。 catch 块提示用户例外后，块中的 continue 将重新返回到 try。 throw 不带任何参数，重新抛出异常给上级处理。

Chapter 2

类

2.1 构造函数

2.1.1 基本特征

构造函数是与类名相同的成员函数。

编译器默认添加：

1. 构造函数
2. 拷贝构造函数
3. 析构函数
4. 赋值运算符函数

1. 返回类型为void
2. 可以重载，但必须有不同的函数署名
3. 默认不带任何参数
4. 创建对象时会隐式调用
5. 用来初始化数据成员
6. 默认构造函数定义在类内
7. 带参构造函数定义在类外

2.1.2 拷贝构造函数

如果不提供，编译器会自动生成：将源对象所有数据成员的值逐一赋值给目标对象相应的数据成员。

```
1    Person(Person&);
2    Person(const Person&);
```

2.1.3 转型构造函数

关闭因转型构造函数导致的隐式类型转换，将运行期错误变成了编译器错误。

```
1    explicit Person(const string& n) {name = n;}
```

2.1.4 构造函数初始化

```
1    class C {
2        public:
3            C() {
4                x = 0;
5                c = 0; //ERROR(CONST)
6            }
7        private:
8            int x;
9            const int c;
10   }
```

对const类型初始化，只需要添加一个**初始化列表：**

```
1    class C {
2        public:
3            C() : c(0) {x = 0;}
4        private:
5            int x;
6            const int c;
7    }
```

初始化段由冒号:开始，c为需要初始化的数据成员，()内是初始值，这是初始化const的**唯一方法**。 **初始化列表**仅在**构造函数**中有效。数据成员的顺序仅取决于类中的顺序，与初始化段中的顺序无关。

2.1.5 new

```
1
2     new constructor[([arguments])]
```

2.2 析构函数

析构函数当对象被**销毁**时，自动调用。

```
1   class C {
2       public:
3           C() {}
4           ~C() {}
5       private:
6           int x;
```

没有**参数**和**返回值**，不能重载。

2.3 指向对象的指针

名称	符号	用途
成员选择操作符	.	对象和对象引用
指针操作符	-¿	指针访问成员，专用于对象指针

用途：

1. 作为参数传递给函数，通过函数返回

2. 使用new(])操作符动态创建对象，然后返回对象的指针

2.3.1 常量指针 this

常量指针：不能赋值、递增、递减，不能在static成员函数中使用。

避免命名冲突：

```
1 void setID (const string& id) {this->id=id;}
```

2.4 类数据成员和类成员函数

如果不使用 static(静态) 关键字，数据成员和成员函数都是属于**对象**的。而使用 static 则可以创建**类成员**：分为对象成员和实例成员。

2.4.1 类数据成员

静态成员只与类本身有关，与对象无关。它对整个类而言只有一个，而且必须在**任何程序块外**定义。

2.4.2 类成员函数

static 成员函数只能访问其他 static 成员，而非 static 成员都可以访问。同时 static 成员函数也可以是 inline 函数。

可以通过两种方式访问：

```
1 C c1;
2 c1.Meth();
3 C::Meth();
4 unsigned x = c1.var;
5 unsigned y =C::var;
```

但首选用类直接访问，为了说明静态成员直接与类关联。

2.4.3 成员函数中的静态变量

如果将成员函数中的局部变量定义为静态变量，**类的所有对象在调用这个成员函数共享这个变量**。

Chapter 3

继承

3.1 访问控制

	private(default)	protected	public
私有派生	不可访问	私有	私有
受保护派生		受保护	受保护
公有派生		受保护	公有

3.2 私有继承

3.3 多重继承

3.4 类的自动、强制转换

3.5 类派生

Chapter 4

多态

4.1 静态联编

4.2 动态联编

Chapter 5

操作符重载

Chapter 6

模板

6.1 友元类

6.2 友元函数