Thursday, 23 November 2017

面向对象语言程序设计(C++)

一些基本概念

左值是对应内存中有确定存储地址的对象的表达式的值；右值是所有不是左值的表达式的值。
一般来说，左值是可以放到赋值符号左边的变量。但，能否被赋值不是区分左值与右值的依据。比如，C++的const左值是不可赋值的；而作为临时对象的右值可能允许被赋值。
左值与右值的根本区别在于是否允许用取地址运算符&，去获得对应的内存地址。

```
int i = 0;
int *p1 = &(++i); //正确
int *p2 = &(i++); //错误
++i = 1; //正确
i++ = 5; //错误
```

为什么i++ 不能作为左值，而++i 可以？(i++返回的是一个临时变量，而临时变量是右值)

```
// 前缀形式:
int& int::operator++() //这里返回的是一个引用形式，就是说函数返回值也可以作为一个左值使用
{//函数本身无参，意味着是在自身空间内增加1的
  *this += 1;  // 增加
  return *this;  // 取回值
}

//后缀形式:
const int int::operator++(int) //函数返回值是一个非左值型的，与前缀形式的差别所在。
{//函数带参，说明有另外的空间开辟
 int oldValue = *this;  // 取回值
  ++(*this);  // 增加
  return oldValue;  // 返回被取回的值
}
```

C++ 11的一些特性

- 可用using语句继承基类构造函数()

继承与派生 Inheritance and Derivative

- 单继承派生：

```
class Derived: public
Base{
public:
      Derived ();
      ~Derived ();
};
```

- 多继承派生：

```
class Derived: public
Base1, private Base2{
public:
      Derived ();
      ~Derived ();
};
```

- 公有继承：
  - 基类的public和protected成员：访问属性在派生类中保持不变
  - 基类的private成员：不可以直接访问
- 私有继承：
  - 基类的public和protected成员：都以private身份出现在派生类中
  - 基类的private成员：不可以直接访问
- 保护继承：
  - 基类的public和protected成员：都以protected身份出现在派生类中
  - 基类的private成员：不可以直接访问
- protected成员的特点与作用：
  - 对建立其所在类对象的模块来说，它与private成员的性质相同

- 对于其派生类来说，它与public成员的性质相同
- 既实现了数据隐藏，又方便继承，实现代码重用
- 如果派生类有多个基类，也就是多继承时，可以用不同的方式继承每个基类。
- 如果基类和派生类都是一个人或一个团队写的，那么使用保护继承没有问题。但是如果是不同团队完成的，甚至是购买来的类库，那么使用保护继承就不太方便了（修改接口非常麻烦）。

- 三种继承方式的归纳

	派生类的成员函数对基类成员的访问权限	派生类的对象对基类成员的访问权限
公有继承 public inheritance	可以直接访问基类中的公有成员和保护成员，但不能直接访问基类中的私有成员	只能访问公有成员
私有继承 private inheritence	可以直接访问基类中的公有成员和保护成员，但不能直接访问基类中的私有成员	不能访问从基类继承过来的任何成员
保护继承 protected inheritance	可以直接访问基类中的公有成员和保护成员，但不能直接访问基类中的私有成员	不能访问从基类继承过来的任何成员

- 基类与派生类的相互转换：
  - 公有派生类对象可以被当作基类的对象使用，反之则不可以
    - 派生类的对象可以隐含转换为基类对象
    - 派生类的对象可以初始化基类的引用
    - 派生类的指针可以隐含转换为基类的指针
  - 通过基类对象名，指针只能使用从基类继承的成员
- 派生类的构造和析构
  - 默认情况

  - 基类的构造函数不被继承;
  - 派生类需要定义自己的构造函数。
- C++11规定
  - 可用using语句继承基类构造函数。
  - 但是只能初始化从基类继承的成员。派生类的新增成员只能按类内初始值或者默认方式进行初始化。(因此这个新特性只适用于派生类很少或没有增加新数据成员的情况)
    - 派生类新增成员可以通过类内初始值进行初始化。
- 语法形式：
  - using B::B; (基类名)(作用域分辨符)(基类构造函数名)
  - 自行设计构造函数：如果不继承基类的构造函数
  - 派生类新增成员：派生类定义构造函数进行初始化；
  - 继承来的成员：自动调用基类的构造函数进行初始化；
  - 派生类的构造函数需要给基类的构造函数传递参数。
- 思考：派生类如果没有写构造函数会怎么样?
  - 当基类有默认构造函数时
    - 派生类的构造函数可以不向基类构造函数传递参数
    - 构造派生类的对象时，基类的默认构造函数将被调用
  - 如需执行基类中带参数的构造函数
    - 派生类构造函数应该为基类构造函数提供参数
- 构造函数的执行顺序
  - 1. 调用基类构造函数。
    - **初始化的顺序按照它们被继承时声明的顺序（从左向右），即使没给某个基类传参数，只要这个基类被继承了，就会调用它的默认构造函数。**
  - 2. 对初始化列表中的成员进行初始化

    - **初始化的顺序按照它们在类中定义的顺序（即使没给这个类内的成员对象传参数，只要在类中被定义了，也会调用它的构造函数）**
    - 对象成员(这里主要是指派生类新增的成员，且这些类内的成员是其他类的对象)在初始化时自动调用其所属类的构造函数，由初始化列表提供参数
  - 3. 执行派生类的构造函数体中的内容。
- 派生类的复制构造函数/析构函数
  - 注意：析构顺序与构造顺序成“镜像”相反

- 派生类成员的标识与访问
  - 访问从基类继承的成员/避免二义性和冗余的情况
  - 虚基类：从第一级继承开始使用虚继承方式，可以避免因继承过程中继承了公共基类而产生冗余的问题。在虚继承的情况下，调用基类构造函数时，只有最远派生类传递的参数起实际作用。

多态性 Polynomial

- 运算符重载
- 虚函数
- 抽象类
- override与final

模板与群体数据

- 模板
- 线性群体
- 数组
- 链表
- 栈
- 队列
- 排序
- 查找

C++标准模板库与泛型程序设计 Templates and Generic Programming

- 泛型程序设计及STL的结构
- 迭代器
- 容器的基本功能与分类
- 顺序容器
- 关联容器
- 函数对象
- 算法

流类库与输入\输出 Inheritance and Derivative

异常处理