C++的一些知识点

整理by 2015级计算机科学与技术 冉诗菡

- 析构函数 赋值函数 拷贝构造函数 三者缺一不可
- 友元函数：在实现类之间数据共享时，减少系统开销，提高效率。如果类A中的函数要访问类B中的成员（例如：智能指针类的实现），那么类A中该函数要是类B的友元函数。具体来说：为了使其他类的成员函数直接访问该类的私有变量。即：允许外面的类或函数去访问类的私有变量和保护变量，从而使两个类共享同一函数。 实际上具体大概有下面两种情况需要使用友元函数：（1）运算符重载的某些场合需要使用友元。（2）两个类要共享数据的时候。（一些操作符的重载实现也是要在类外实现的，那么通常这样的话，声明为类的友元是必须滴。

- 在一个类中可以说明具有类类型的数据成员

eg:
```
class wheel
{ ….
};
class car
{  wheel w;
};
```

- 形参或函数中定义的变量都是局部变量。在函数外定义的变量是全局变量。形参只能用局部变量，频繁使用的变量可以声明为寄存器变量，形参不能使用静态变量或寄存器变量。

- 成员函数有this指针，而友元函数没有this指针，静态成员函数没有this指针。
- 友元函数是不能被继承的，就像父亲的朋友未必是儿子的朋友。

- this指针是类的一个自动生成、自动隐藏的私有成员，它存在于类的非静态成员函数中（即静态成员函数没有this指针），指向被调用函数所在的对象。全局仅有一个this指针，当一个对象被创建时，this指针就存放指向对象数据的首地址。使用时：一种情况就是，在类的非静态成员函数中返回类对象本身的时候，直接使用 return *this；另外一种情况是当参数与成员变量名相同时使用this指针，如this->n = n （不能写成n

= n）。

- 指针使用中的注意事项：
- 一.在定义指针的时候注意连续声明多个指针时容易犯的错误，例如int * a,b;这种声明是声明了一个指向int类型变量的指针a和一个int型的变量b，这时候要清醒的记着，而不要混淆成是声明了两个int型指针。
  二.要避免使用未初始化的指针。很多运行时错误都是由未初始化的指针导致的，而且这种错误又不能被编译器检查所以很难被发现。这时的解决办法就是尽量在使用指针的时候定义它，如果早定义的话一定要记得初始化，当然初始化时可以直接使用cstdlib中定义的NULL也可以直接赋值为0，这是很好的编程习惯。
  三.指针赋值时一定要保证类型匹配，由于指针类型确定指针所指向对象的类型，因此初始化或赋值时必须保证类型匹配，这样才能在指针上执行相应的操作。

- 公有继承下，基类成员中公有和受保护类型的访问权限都不变，但基类的私有成员无论采用何种继承方式，在子类中都将变得不可访问。
- 继承是面向对象语言的一个重要机制，
  通过继承可以在一个一般类的基础上建立新类，被继承的类称为基类，
  在基类上建立的新类称为派生类。
  继承和派生其实都是一回事，只是说法不同罢了。
  如：子类继承了父类，父类派生了子类。

- 拷贝构造函数是用一个已存在的对象去构造一个不存在的对象（拷贝构造函数毕竟还是构造函数嘛），也就是初始化一个对象。而赋值运算符重载函数是用一个存在的对象去给另一个已存在并初始化过（即已经过构造函数的初始化了）的对象进行赋值。
- 拷贝构造函数是在对象被创建时调用的，而赋值函数只能被已经存在了的对象调用。
- 比如：String s1("hello"),s2=s1;//拷贝构造函数
- Sring s1 ("hello"),s2;　　s1=s2;//赋值运算符重载 即赋值函数

- **以下情况都会调用拷贝构造函数：**

1、一个对象以值传递的方式传入函数体　（形参和实参结合）

2、一个对象以值传递的方式从函数返回 （函数返回）
3、一个对象需要通过另外一个对象进行初始化。

- 类String的拷贝构造函数与赋值函数
  // 拷贝构造函数
  String::String(const String &other)
  {
  // 允许操作other的私有成员m_data
  int length = strlen(other.m_data);
  m_data = new char[length+1];
  strcpy(m_data, other.m_data);
  }
  // 赋值函数
  String & String::operator =(const String &other)
  {
  // (1) 检查自赋值
  if(this == &other)
  return *this;
  // (2) 释放原有的内存资源
  delete [] m_data;
  // (3)分配新的内存资源，并复制内容
  int length = strlen(other.m_data);
  m_data = new char[length+1];
  strcpy(m_data, other.m_data);
  // (4)返回本对象的引用
  return *this;
  }

- 对于*无参函数*而言,*括号*不能省略

- **class中的成员默认是private,而struct的成员默认为public。即：**
  用class定义的类，如果不作private或public声明，系统将其成员默认为private，在需要时也可以自己用显式声明改变。用struct声明的类，如果对其成员不作private或public的声明，系统将其默认为public。

- const变量：必须在定义的时候初始化 因为后面就不能更改了呀

- os<<s 返回os 链式输出
- is>>s 返回is 链式输入

- 不要忘了字符串数组的话 也是从0开始计数的

- 三种函数形参

- 函数的重载:

函数同名 但是参数的类型或者个数不同
派生类对基类的成员函数的重载：派生类会覆盖基类

- 异常处理：

抛出异常：throw
处理异常：try...catch
eg：

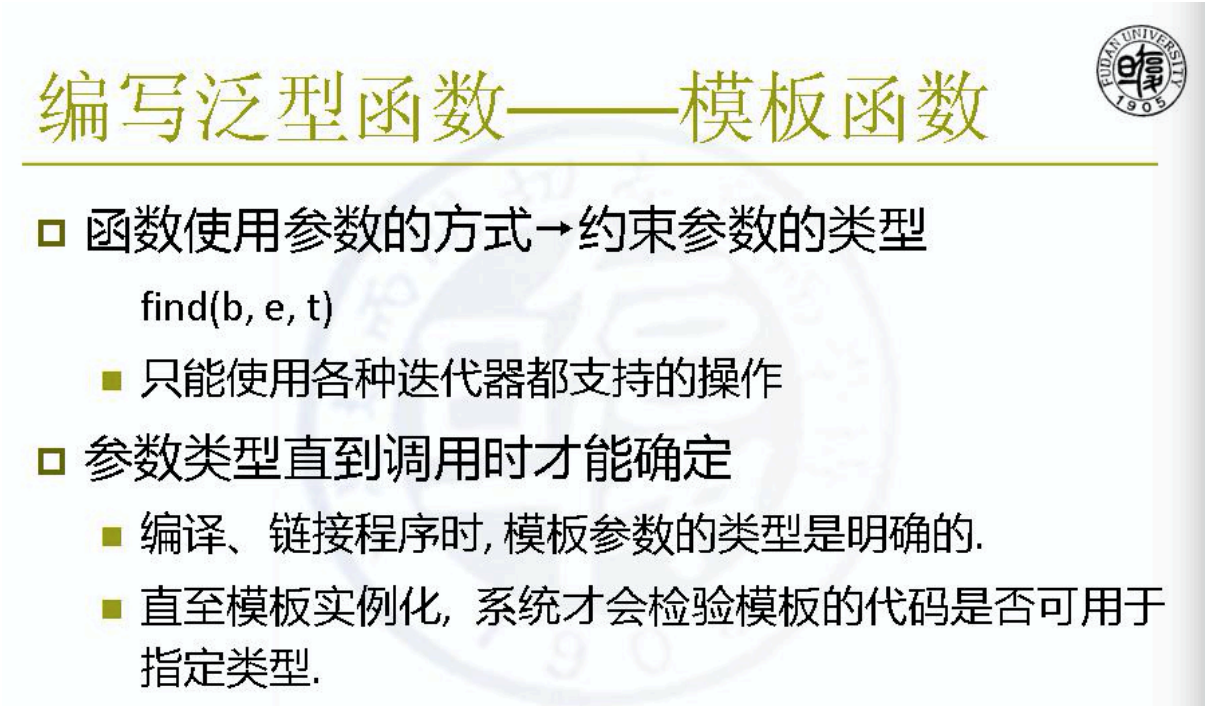
```
cout << “record1 / record2 = “;
try {
    cout << record1 / record2 << endl;
}
catch( runtime_error e) {
    cout << e.what() << endl;
}

//在BigInt operator /(const BigInt& X,const BigInt& Y)的函数体定义中要写清楚 if(y==0) throw std::runtime_error (“Meaningless!”);
```

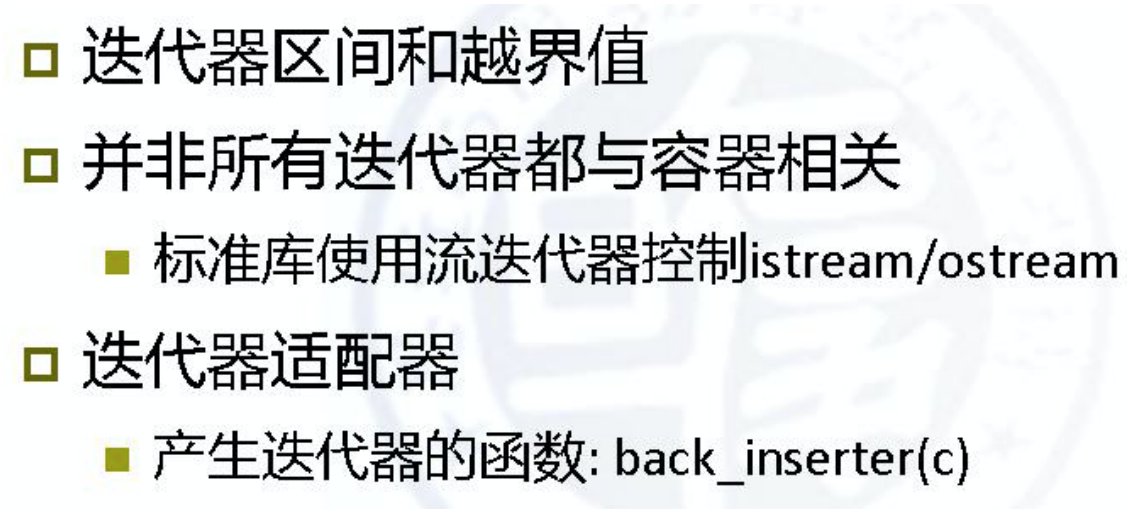
- 头文件：

#include <>是系统的头文件 “”是自定义的头文件
在头文件中可以包含 类定义 模版函数模版类的定义 全局变量 函数声明
源文件中可以包含 类的具体实现 和 全局函数的定义等

- 编写模版函数（泛型函数）：



- 迭代器：



- 指针和迭代器的区别

1. 指针和iterator都支持与整数进行+，-运算，而且其含义都是从当前位置向前或者向后移动n个位置

2. 指针和iterator都支持减法运算，指针-指针得到的是两个指针之间的距离，迭代器-迭代器得到的是两个迭代器之间的距离

3. 通过指针或者iterator都能够修改其指向的元素

通过上面这几点看，两者真的很像，但是两者也有着下面的几个不同地方

1. cout操作符可以直接输出指针的值，但是对迭代器进行在操作的时候会报错。通过看报错信息和头文件知道，迭代器返回的是对象引用而不是对象的值，所以cout只能输出迭代器使用*取值后的值而不能直接输出其自身。

2. 指针能指向函数而迭代器不行，迭代器只能指向容器

这就说明了迭代器和指针其实是完全不一样的概念来的。指针是一种特殊的变量,它专门用来存放另一变量的地址，而迭代器只是参考了指针的特性进行设计的一种STL接口。

- 读写文件：ifstream和ofstream

- 内存管理

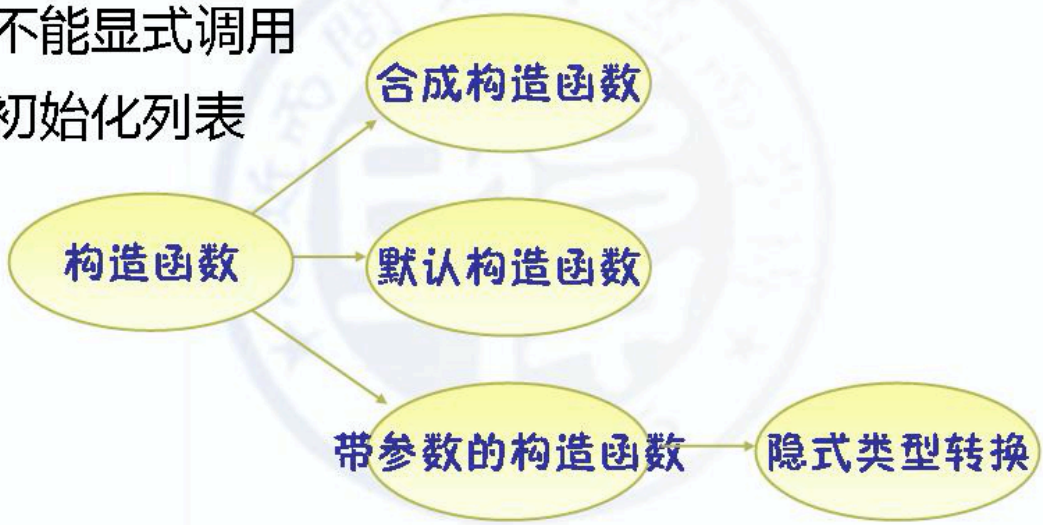
- 类的定义

定义新类型

- 创建与内置类型同样易用的类型
- class *vs* struct

构造函数

- 不能显式调用
- 初始化列表

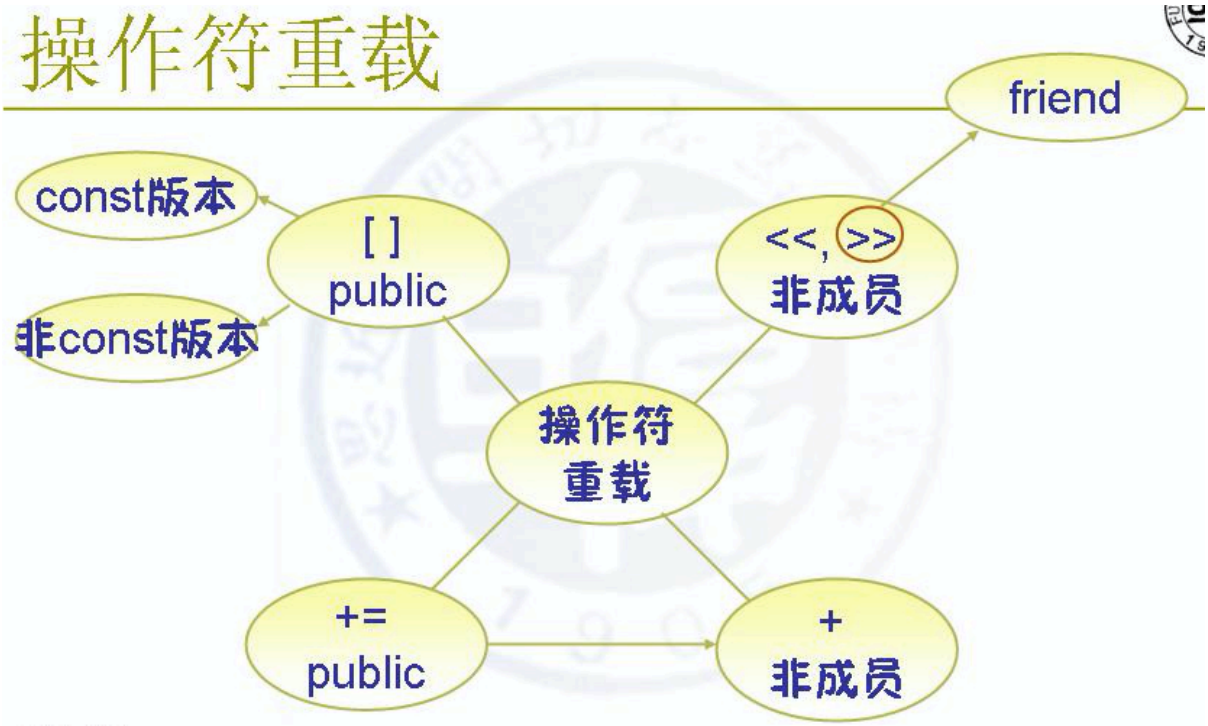


复制控制


合成版本
copy constructor
显式复制
实参→形参
隐式复制
函数返回对象值
分配资源
复制控制
operator=
返回l-op&
destructor
避免自我赋值
三者缺一不可
成员函数


参数类型相同:
const T&
行为相同:
逐个成员赋值
相同
copy constructor vs operator=
不同
前者负责初始化,
后者完成赋值
后者先删除
对象已有数据

- 操作符重载



- `explicit`:

C++中， 一个参数的构造函数(或者除了第一个参数外其余参数都有默认值的多参构造函数)， 承担了两个角色。 1 是个构造器 ， 2 是个默认且隐含的类型转换操作符。

explicit构造函数的作用

解析：

explicit构造函数是用来防止隐式转换的。请看下面的代码：

```
class Test1
{
public:
    Test1(int n)
    {
        num=n;
    }//普通构造函数
private:
    int num;
};
class Test2
{
public:
    explicit Test2(int n)
    {
        num=n;
    }//explicit(显式)构造函数
private:
    int num;
};
int main()
{
    Test1 t1=12;//隐式调用其构造函数,成功
    Test2 t2=12;//编译错误,不能隐式调用其构造函数
    Test2 t2(12);//显式调用成功
    return 0;
}
```

Test1的构造函数带一个int型的参数，代码23行会隐式转换成调用Test1的这个构造函数。而Test2的构造函数被声明为explicit（显式），这表示不能通过隐式转换来调用这个构造函数，因此代码24行会出现编译错误。

- protected：

protected专门就是为继承（子类）设计的 用public继承 那么基类所有的访问标识在子类不变 而protected内的内容 只有类本身 和类的子类可以访问，对象是无法访问的！
除了在继承上 他跟private没有任何区别！
private无法继承 也就是说子类也不能用基类的 private...
pravite 只对本类可见 protected 对本类和继承类可见

- 在c++类体系中，不能被派生类继承的有：

成员函数：构造函数 拷贝构造函数 赋值函数 析构函数
非成员函数：友元函数

- 如果写了带参数的构造函数就要显示地写默认构造函数
- 析构函数 赋值函数 拷贝构造函数 三者缺一不可

- 静态：

静态成员static

- 类的所有对象共享，只有一个实例
- 通过类直接访问

- 类的成员函数(简称类函数)是函数的一种，
  它与一般函数的区别只是：
  它是属于一个类的成员，出现在类体中。
- 它可以被指定为private(私有的)、public (公用的)或protected(受保护的)。私有的成员函数只能被本类中的其它成员函数所调用，而不能被类外调用。
- 一般的做法是将需要被外界调用的成员函数指定为public，它们是类的对外接口。

- 类函数必须先在类体中作原型声明，然后在类外定义，也就是说类体的位置应在函数定义之前，否则编译时会出错。