软件设计II模板

leidar100@gmail.com

- 类函数的实现implementation

  有些题目往往给出了类的声明,要求完成实现.其实也就是在类外写一个函数

  函数名前面加上类名

```
class seriesComp{
    seriesComp(int n);
    int sum();
}
//返回类型不用改变,参数名不用改变,只要在函数名前加类名
seriesComp::seriesComp(int n)
{
    // your code
}//构造就不用返回
int seriesComp::sum() {
    // your code
}//该返回int就返回int
```

```
//和上面等价的实现
class seriesComp{
    seriesComp(int n){
    // your code1
    }
    int sum(){
    // your code2
    }
}
```

- **级联**,就是返回写当前类,然后最后返回`*this`指针
  - 在流中相对有意义一些..

构造函数

- 初始化成员变量
  1. this指针

```
//常见于构造函数,区分与传入参量的相同名字
Date::Date(int year, int month)
{
    this->year =year;             //可调用属性
    this->month=month;
    this->pass =this->validate();//可调用方法
}
```

  2. 成员初始化器

```
//在构造函数里面,即使和传入名相同(同上)
//初始化const对象
//真正编译初始化的顺序是依据声明的顺序
Date::Date(int year, int month)
  :year(year),month(month)      //用逗号分开
{
    this->pass =this->validate();//可调用方法
}
```

  3. 对于指针分配新的内存空间=>数组

```
//new char[长度]  char可以换成别的类型或者模板T
```

  4. 更新static计数器数目
  5. 调用Setter更新
     1. 缺省参数需要在函数声明的时候写上,如果定义跟声明分开,则在定义函数的时候不需要在写上缺省参数

- 拷贝构造函数
  1. *关键在于加&传引用*,`不然会无限递归`
  2. **深拷贝**(当传入一个数组时候需要考虑,直接赋值仅仅改变了指针方向)

```
String::String(const String& b){
    str=new char[strlen(b.str)+1];//#include <cstring>,动态分配内存空间
    int pos;
    for (pos=0; b.str[pos]!='\0'; pos++)
        str[pos]=b.str[pos];
    str[pos]='\0';//字符串最后一位记得加'\0'
}
```

- 注意意义不明的多个重载往往会导致编译错误

析构函数(RAII)

- 长相
  - 没有返回值,没有参数
  - 函数名前加tilde `~` : `ClassName`
- 清理动态(new)分配的内存空间
- 可能与static参量相关,析构时候-1

```
MyString::~MyString(){
    MyString::numberOfObjects--;//如果有计数器的话
    delete str;//删除没有被分配空间的指针时候会CE
}
//删除动态分配的数组
    delete []arr;
```

- 在出了代码块或者使用delete时候调用,
- 全局变量在main,exit时候自动调用,但是**abort**退出的程序不调用

> 析构函数是“反向”的构造函数,析构函数不允许有返回值,不带参数,一个类中只有一个。

Getter & Setter

- 为了实现类的封装(从外界装逼地修改private数据),于是出现了Getter个setter

  这样就可以**象征性**的(因为程序员可以控制外界可以读取/修改哪些内容)保护类中的数据和方法
- Getter模板

```
int Date::getYear() const
{
    return year;//一般直接返回,但是在重载数组[]的时候如果认为是getter可能检查下标
}
```

- Setter模板

  与Getter不同的是,**往往需要检查传入数据的合法性**

```
void Date::setDate(const string& input)
{
    if (checkFormat(input))
    {
        istringstream trans(input);
        char temp;
        trans >> year >> temp >> month >> temp >> day;
    }//检查数据合法性
    else
        year=month=day=233;
    pass=validate();
}
```

运算符重载

其实就是把函数名换成了运算符

1. 一般而言会遵循使用习惯返回值,比如`+=`返回当前对象,`==`返回bool值
2. `+=` 和 `+`, `==` 和 `!=`都需要分开重载,`+`,`-`是重载正负运算符
3. 很多STL使用需要重载<运算符,比如set,map,优先队列
4. 操作符的实质是函数重载。
5. 普通运算符重载(参考lab06)

- ```
  返回类型 运算符operator(符号名) (参数列表)
  {
    函数体
  }
  /*hw05
  返回类型:Complex对象
  符号名:+
  因为是对类的方法的实现,所以要加作用域符号Complex::
  参数列表: 另一个Complex
  */
  Complex Complex::operator+(Complex&com)
  {
      Complex ans(real+com.real,imag+com.imag);
      return ans;
  }
  ```

- 流(左移右移运算符)重载**记得加&**

  可参考这篇题解更深入理解<<运算符的重载

  因为是在外界调用,需要声明为友元函数
  ```

```
istream& operator>>(istream& source, char *pDest);
ostream& operator<<(ostream& dest, char *pSource);

//一般输入输出流的重载
//按照自己意愿输出,使用和cin,cout类似
ostream& operator<<(ostream& os, Complex& com){
    //这里给os输出一些东西    os << something
    return os;//返回流
}

//overload `>>` operator按照自己意愿输入
istream& operator>>(istream& os, Complex& com){
    os>>com.real>>com.imag;
    return os;//返回流
}
```

- 取位运算符重载

```
/*
返回类型: double
符号名 :  []   //重载这个运算符往往要注意位置是否合法
参数列表: int
*/
double& Complex::operator[](int i){
    if (i==0) {
        return real;
    }else
        return imag;
}
```

- 赋值运算符重载(或涉及深拷贝)

```
/*
返回类型：类的对象,这里是String
符号名 ：  =    //重载这个运算符往往要注意位置是否合法
参数列表：任意参数,这里是char,也可以是string
*/
String& String::operator=(const char *b){
    str=new char[strlen(b)+1];
    int pos;
    for (pos=0; b[pos]!='\0'; pos++)
        str[pos]=b[pos];
    str[pos]='\0';
    return *this;
}
/*判断是否为当前,lab07*/
const MyVector & MyVector::operator=(const MyVector &vec){
    if(*this==vec)return *this;
        delete []elem;
    int len = vec._size;
    elem = new Elements[len];
    for (int i=0; i<len; i++)
        elem[i]=vec.elem[i];
    _size=len;
    return *this;
};//assignment
```

- `++` `--`

```
// Define function operators for prefix ++ and --
Rational& Rational::operator++(){
    numerator += denominator;
    return *this;//前缀++,可以直接把当前加一,然后返回当前
}

// Define function operators for postfix ++ and --

Rational Rational::operator++(int dummy){
    Rational old = *this;
    numerator += denominator;
    return old;//后缀++,记得有dummy值,返回以前的值
}
```

- 类型转换运算符

```
Rational::operator double(){
    return ((double)numerator/denominator);
}//没有标明返回值但是需要返回
```

两个类的has关系

- 其实c++中int也是一个类,char也是一个类,只不过是语言自带的
- 所以包含`int a;`和包含`MyClass a`层次上面是等价的..所以包含就行(见lab5)

两个类的is关系(继承)

- 基本模型

```
//在头文件中include父类的头文件
class derived:father{
  //它的新方法和属性
}
```

- 这样就可以
  1. 拥有父类的所有东西
  2. 使用父类`public`和`protected`可见度的方法
  3. 被父类指针指向
- protect可见性
  - protected:可以在(1)该类方法、(2) 子类方法、 (3) 其友元函数中访问
- 继承特性
  - 会屏蔽基类同名函数,即使函数原型不同也会被屏蔽。
- 参考模板1(课件,调用父类构造函数)

```
Circle::Circle( double r, int a, int b )
: Point( a, b ) // call base-class constructor
{ setRadius( r ); }
```

- 参考模版2（最后一次homework）

- ```
  class ScoreSheet:public pair<int, string>
    //因为先对分数排序,再对姓名,继承目的在于继承排序函数
  {
  public:
      ScoreSheet(string name,int score)
      {
          this->first = score;
          this->second = name;
      }
      friend ostream & operator << (ostream &os,const ScoreSheet &stu)
      {
          os << stu.second << " " << stu.first;
          return os;
      }
  };
  ```
  - 继承类型
  - > private 属性不能够被继承。
    >
    > 使用private继承,父类的protected和public属性在子类中变为private; 使用protected继承,父类的protected和public属性在子类中变为protected;
    >
    > 使用public继承,父类中的protected和public属性不发生改变;

多态

- virtual 函数
  - 在父类里面函数前加virtual,子类里面实现
  - 指向子类的指针就会使用子类里面的实现
  - 纯虚构函数 => 抽象类
    - `virtual 返回值 函数名(参数列表) = 0;`
    - 必须在子类实现,没有父类对象
    - `virtual void withdraw(double) = 0;`
  - 一般析构函数都声明为virtual
  - 纯虚函数是一种特殊的虚函数,格式如下,在参数 列表后加 “=0”:• virtual <类型><函数名>(<参数表>)=0;
- **Override(覆盖或重写)**,是指派生类重写基类的虚函数。 重写的函数必须有一致的参数表和返回值(注意特殊情况)。
- **Hide(隐藏或遮蔽)**,是指派生类中的函数屏蔽了基类中相同名字的函数
- 虚基类正是为了在多重继承的类层次中节省空间 和避免不确定性
  ```

关键字

static

> 其实我觉得static就是类里面的“全局变量”,类中static函数+static变量相当于在面向过程的普通函数+全局变量　　-by第三周实验报告

- static成员变量需要**在类外面初始**化为0(lab05)

```
//内部-模板
class MyString
{
public:
...
    static int getNumberOfObjects();
private:
...
    static int numberOfObjects;             //counter
};
//外部-格式:类型 类名::变量名=初始值;
int MyString::numberOfObjects=0;//necessary declare but '=0' not
necessary

int MyString::getNumberOfObjects(){
    return numberOfObjects;
}//照样要加MyString::也就是类名
```

- 不能被this指代
  - 应用 对象数目统计

```
//构造函数+1
MyString::MyString(int Length, char sym)
{
    numberOfObjects++;//这里
    str = new char[Length+1];
    fill(str, str+Length, sym);//fill is an interestiong function
    str[Length]='\0';
}
//析构函数-1
MyString::~MyString(){
    MyString::numberOfObjects--;
    delete str;
}
```

const

- 最小特权性原则
- "Use const whenever possible"

语法规范

1. const对象只能调用const函数**const对象不能调用Setter**
    - const函数不能调用非const函数(保证不被间接调用)
2. const函数不能修改数据,
3. 构造析构函数不能被声明为const
4. const参量往往在成员初始化器中初始化
    1. const char[] 和 char const []

模板

```
//Prototype:
ReturnType FunctionName(param1,param2...) const;
//Definition:
ReturnType FunctionName(param1,param2...) const { ...};

//Example:
      int A::getValue() const
         { return privateDataMember };
//- Returns the value of a data member, and is appropriately declared const
```

friend

- 友元函数,就是类外的函数可以像类里面的函数使用

    > Can access private and protected members of another class

- 友元类,就是那一个类可以使用另一个类的属性与方法**不对称性**

    - 类的主要特点是实现数据隐藏,既不允许非成员函数对它访问,但在某些场合下,**输入输出流**非成员函数体中需要通过对象名访问private成员,这可以通过友元函数来实现。
        -

this

- 某个函数**比如前缀自增运算**返回当前对象 `return *this`

其它内容

文件流

这篇博客是一个文件流OJ的入门

- 就是创造一个类似于cin,cout的东西

```
//输入流_1
#include <fstream>
ifstream fin("in.txt");
fin.close();
//输入流_2
#include <fstream>
ifstream fin;
fin.open("in.txt");
fin.close();
//另外读入string文件名的话可以用.c_str()转化为字符串
//输出流
#include <ofstream>
ofstream fout("out.txt");
fout.close();
```

- 另外想舒服正常一些操作的话有下面两种方法
    1. 不include
    2. 重定向流输入输出

        ```
        #include <cstdio>
        freopen("telecow.in", "r", stdin);
        freopen("telecow.out", "w", stdout);
        ```

- 其它流运算符`hex`,`fixed(保留小数)`,`showpoint`,`setprecision(n)`
- 流标志符`eof()`

流

> iostream定义读写控制窗口的类型
>
> fstream定义读写已命名文件的类
>
> 型sstream定义了读写内存中string对象的类

- sstream
  - 流对象,`stringstream str(Str);`
  - 可以使用`>>`,`<<`来往这个流内输入输出

> 缓冲区
>
> endl:用于输出一个换行符并刷新缓冲区
>
> flush:用于刷新流,但不在输出中添加任何字符
>
> endl:在缓冲区中插入空字符null,并刷新缓冲区

模板类/函数

在函数/类前面加上

`template <typename T, int capacity>`//第二个是容量参数,可以用来声明数组长度

或者

`template <class T>`

- 构造函数(举个例子)

```
template <typename T, int capacity> //类前需要加上
class Stack
{
public:
    Stack();                    // Constructs an empty stack.
private:
    T* elements;                   // Points to an array that stores
elements in the stack.
};
template <typename T, int capacity> //每一个函数前都得加上
Stack<T,capacity>::Stack()
{
    elements = new T[capacity];
}                    // Constructs an empty stack.
```

- 使用的时候参考STL中vector使用
- 另外这个的头文件和实现不能分成两个文件
- 模板测试函数

```
template <class T>
void TEST(Array<T,MAXLEN> &theArray,
          T FirstValue,T increment)
{}
```

异常

在一段`可能除0,可能越界,可能干坏事的代码块前`加上`try`

`出现坏事`后`throw`一个`对象`

`catch根据对象类别` 选择措施

```
//基本模型
template <typename T, int capacity>
T& Array<T,capacity>::operator[] (int index)
{
    try {
        if (index < 0 || index >= num)
        {
            throw ArrayException("Out of Range Exception");
        }//illegal index
    }
    catch (int a){};
    return elements[index];
}
//函数声明
Exception Specifications
在函数的声明中列出可能抛出的异常类型 Example:
void fun() throw( int, float, ExceptionClass ) ; //可抛出int、float、
ExceptionClass类型的异常
若无声明,则可以抛出任何类型的异常 void fun()
不抛出任何类型的异常的声明如下: void fun() throw() ;
```

- 异常类模板

```
//异常类就是专门一个类,throw这个类的一个对象
class ArrayException
{

public:

    ArrayException(const char *msg);

    const char *what() const;//约定俗成的名字

private:

    const char *message;

};
```

注释

> Comment why, NOT how or what

```
/* Name: pun.c
 Purpose: Prints a bad pun.
 Author: K. N. King */

/*
Copyright Authors
Usage References &
Relations Background &
Reasons Design
decisions Complicated
algorithms*/
```

封装及面向对象

详见另一篇blog HEAD FIRST OOAD读书笔记

- minimum level of visibility. <=因为特权越小则越安全

lab04

对象数组 就是对于数组指针和移位的练习,有趣的是迭代器不能+1来实现指针的移位

- **RAII**

  析构了一个值之后又要重新使用那个值

  于是就拿另一个指针来存那个值

```
while (cin >> value)
{
  //下面就是一个代码块{}包围
                {//类中把value值赋成另一个
                        AutoReset auto_reset(&value, count);
                        value *= 2;
                        count++;
                        cout << value << " ";
                }//类作用的有效区域,出了之后需要把value值还原
   cout << value << endl;
}

int save,*s;
AutoReset::AutoReset(int *scoped_variable, int new_value){
    s=scoped_variable;//传入的是指针,把原来指向那个值的指针存下来
    save=*scoped_variable;
    *scoped_variable=new_value;
}
// your code will be here

AutoReset::~AutoReset(){
    *s=save;//把存下了的指针还回去
}
```

参考模板

可能用到的函数库

在这一篇的最下面总结了模板头文件..

sicily上可以用`#include<bits/stdc++.h>`来囊括所有函数库

另外附加了三个`cplusplus`的网页集合