**1．已知字符串类MyString的定义为：**

class MyString

{

public:

MyString(const char\* pData= NULL); //普通构造函数

MyString(const MyString &); //拷贝构造函数

~ MyString(); //析构函数

**MyString&** operator =(const MyString &); //赋值函数

**MyString&** operator += (const MyString &); **//字符串连接**

operator char\* () const; //自动转换函数

private:

char \*mpData; //用于保存字符串

};

**全局函数：**

**const MyString** operator + (const MyString &,const MyString &); //字符串连接

**ostream&** operator<<(ostream& os, const MyString& str); //定向输出

请完整实现MyString类和指定的全局函数。（可以使用new,delete运算以及strcpy,strlen，…等库函数，还可以定义其他辅助函数）

**本例使用了三个文件：AppString.cpp, MyString.cpp,MyString.h**

/\*

\* ***MyString.h***

\*

\* Author: Administrator

\*/

#ifndef MYSTRING\_H\_

#define MYSTRING\_H\_

//全局函数：

class MyString;

const MyString operator + (const MyString &,const MyString &); //字符串连接

ostream& operator<<(ostream& os, const MyString& str); //定向输出

class MyString {

public:

MyString(const char\* pData = NULL); // 普通构造函数

MyString(const MyString &); // 拷贝构造函数

~ MyString(); // 析构函数

MyString & operator =(const MyString &); // 赋值函数

MyString& operator +=(const MyString &);

operator char\*() const; // 自动转换函数

private:

char \*mpData; // 用于保存字符串

};

#endif /\* MYSTRING\_H\_ \*/

/\*

***\* MyString.cpp***

\*

\* Author: Administrator

\*/

#include <iostream>

using namespace std;

#include <string.h>

#include "MyString.h"

***//实现全局函***

**//注意：由于实现operator+函数时，只用到了MyString类的公有成员operator+=,**

**//所以，不必将operator+函数声明成MyString类的友元函数。**

const MyString operator+(const MyString & lhs,const MyString & rhs) //字符串连接

{//返回值的const去掉可以吗！实验一下！

return MyString(lhs)+=rhs;

}

**//道理同上，operator<<函数也不用声明成MyString类的友元函数。**

ostream& operator<<(ostream& os, const MyString& str) //定向输出

{

os<<(char\*)str; //用转换函数

return os;

}

**//实现MyClass类**

**//构造函数**

//**参数有缺省值的，缺省值只能在声明中指定**，如这里的pData的缺省值是NULL

MyString::MyString(const char\* pData)

{

if ( pData == NULL )

{

mpData = new char[1];

strcpy(mpData, ""); //或mpData[0]='\0';

}

else

{

mpData = new char[strlen(pData)+1];

strcpy(mpData, pData);

}

}

**// 拷贝构造函数**

MyString::MyString(const MyString & rhs)

{

mpData=new char[strlen(rhs.mpData)+1]; // +1是因为strlen没算入”/0”

strcpy(mpData,rhs.mpData);

}

**// 赋值函数**

MyString& MyString::operator=(const MyString &rhs)

{

if (this!=&rhs)

{

delete[] mpData;

mpData=new char[strlen(rhs.mpData)+1];

strcpy(mpData,rhs.mpData);

}

**return \*this;**

}

**// 析构函数**

MyString::~MyString()

{

delete[] mpData;

}

**//重载的+=运算符**

MyString& MyString::operator+= (const MyString &rhs)

{

char\* pnew=new char[strlen(mpData)+strlen(rhs.mpData)**+1]**;

strcpy(pnew,mpData);

strcat(pnew,rhs.mpData);

delete[] mpData;

mpData=pnew;

return **\*this**;

}

**// 自动转换函数**

MyString::operator char\* () const

{

return mpData;

}

//=====================================================================

// Name  ***: AppString.cpp***

// Author : CW

// Version :

// Copyright : Your copyright notice

// Description : Hello World in C++, Ansi-style

//====================================================================

#include <iostream>

using namespace std;

#include "MyString.h"

int main() {

MyString a;

MyString b("Are you fine?");

cout<<"a="<<a<<endl;

cout<<"b="<<b<<endl;

MyString c("He like computer,but don't like computer science.");

cout<<"c="<<c<<endl;

MyString d(c);

cout<<"d="<<d<<endl;

d+=b;

cout<<"After d+=b, b="<<b<<endl;

cout<<"After d+=b, d="<<d<<endl;

a=c;

cout<<"After a=c, c="<<c<<endl;

cout<<"After a=c, a="<<a<<endl;

cout<<"直接输出 b+a， b+a=" << b+a<<endl;

cout<<"观察这时的a，a="<<a<<endl;

cout<<"观察这时的b，b="<<b<<endl;

return 0;

}

**2.对任意一个正的实数,总可以唯一地表示成a0+1/(a1+1/(a2+1/(a3+1/.....)))的形式，简记为a0+a1+a2+a3+...+ak。(ai为大于0 的正整数，长度为k。若数为无理数，则k为无穷大) ,并称这种形式的数为*连分数*。**

a)请使用类的自关联方式设计并实现连分数类, 此类的每个对象代表一个有理数的连分数形式，且假定连分数的长度均小于MAXLEN（MAXLEN设为数30）。

该类主要功能有：

1)对指定的i，返回ai值；

2)计算连分数的前q项（a0+a1+a2+a3+..+aq）所对应分数的分子和分母；

3)输出此分数的前q项，格式为:a0+a1+a2+a3+..+a**q**；

4)

b)给出主程序，使用该类计算连分数的前n项对应的分数逼近PI值的程度（计算差即可。PI可用math.h中的M\_PI常量）。

**本例使用了三个文件，AppMain.cpp, ContinueFraction.cpp,ContinueFraction.h**

/\*

\* ***ContinueFraction.h***

\*

\* Author: Administrator

\*/

#ifndef CONTINUEFRACTION\_H\_

#define CONTINUEFRACTION\_H\_

***class ContinueFraction***

{

public:

ContinueFraction(double value,int maxlen=100); //构造

~ContinueFraction(); //析构

public:

void GetFzFm(long& fz,long& fm,int len); //获取分子和分母

int GetAi(int index); //指定i,返回ai值

//Display函数最好改成取得一个**连分数字符串表示的函数**，从而**避免使用cout**。

//如： ***char \* TheString(int len);*** 。 这里使用Display只是为了方便。

void Display(int len);

private:

int ai;

ContinueFraction \* next;

};

#endif /\* CONTINUEFRACTION\_H\_ \*/

/\*

\* ***ContinueFraction.cpp***

\*

\* Author: Administrator

\*/

#include <iostream>

using namespace std;

#include "ContinueFraction.h"

ContinueFraction::***ContinueFraction***(double value,int maxlen)

{

ai = int(value);

if ((maxlen>1) && (value-ai>0.00001))

***next = new ContinueFraction(1.0/(value-ai)***,**maxlen-1**);

else

next = **NULL**;

}

ContinueFraction::~ContinueFraction()

{

delete next;

}

void ContinueFraction:***:GetFzFm(l***ong& fz,long& fm,int len)

{

if ( (len==1) || (next == NULL))

{

fz = ai;

fm = 1;

}

else

{

**next->*GetFzFm(***fz,fm,len-1);

***//ai+ 1/(fz/fm) 等价于 ai+ fm/fz***

long newfz = ai\*fz+fm;

fm = fz;

fz = newfz;

}

}

int ContinueFraction::***GetAi***(int index)

{

if (index == 1)

return ai;

else if (next == NULL)//index大于实际ai的个数

return 0;

else

return **next-*>GetAi***(index-1);

}

void ContinueFraction::**Display**(int len)

{

cout<<ai;

if ( (len ==1) || (next == NULL) )

{

cout<<endl;

}

else

{

cout<<"+";

**next->Display(**len-1);

}

}

***AppMain.cpp***

#include <iostream>

#include <math.h>

using namespace std;

#include "ContinueFraction.h"

int main()

{

**const** int MAXLEN = 30;

**const** double PI\_VALUE = M\_PI;

ContinueFraction \* pi = new ContinueFraction(PI\_VALUE, MAXLEN);

**cout.precision(16);**

cout<<"PI="<<PI\_VALUE<<endl<<endl;

long fz, fm;

for(int i= 1; i<=8;i++)

{

pi->GetFzFm(fz, fm, i);

cout << "前"<<i<<"项为";pi->Display(i);

cout << "前"<<i<<"项对应分数为" << fz << "/" << fm << endl;

cout << "前"<<i<<"项对应分数的值为" << (double(fz)) / fm << endl;

cout << "前"<<i<<"项对应分数与"<<PI\_VALUE<<"的差为" << PI\_VALUE –

(double(fz)) / fm << endl << endl;

}

delete pi;

cout << "....." << endl;

return 0;

}

**3．在全局函数void f(int n,int m)中*动态建立*一个大小为nxm的*二维整数数组*，并对每个数组元素依次赋值为1，2，3，...,m\*n, 再输出每行和每列元素的和，退出函数前释放此数组。**

***void f(int n,int m)***

{

**//建立nxm数组**

int\*\* dnew=new int\*[n];

for (int i=0;i<n;++i)

dnew[i]=new int[m];

**//赋值**

int value=0;

for (int h1=0;h1<n;++h1)

for (int w1=0;w1<m;++w1)

dnew[h1][w1]=(++value);

**//输出每行的和**

for (int h2=0;h2<n;++h2)

{

int sum1=0;

for (int w2=0;w2<m;++w2)

sum1+=dnew[h2][w2];

cout<<"第"<<h2+1<<"行的和为："<<sum1<<endl;

}

**//输出每列的和**

for (int w3=0;w3<m;++w3)

{

int sum2=0;

for (int h3=0;h3<n;++h3)

sum2+=dnew[h3][w3];

cout<<"第"<<w3+1<<"列的和为："<<sum2<<endl;

}

**//释放**

for (int j=0;j<n;++j)

delete[] dnew[j];

delete[] dnew;

}

**4．已有类A声明如下：**

class A

{

public:

A(int n):data(n) {}

int Data() const {return data;}

private:

int data;

};

***在全局函数void g(int n) 中动态建立一个大小为n的一维指针数组， 数组元素指向A类对象，各对象的data数据成员各不相同， 分别为1，2，3，...,n。创建数组后，使用（例如输出）各对象的data值，最后释放此数组。***

***void g(int n)***

{

**//创建指针数组**

A\*\* pa=new A\*[n];

for(int q=0;q<n;++q)

pa[q]=new A(q+1);

**//输出**

for (int z=0;z<n;++z)

cout<<pa[z]->Data()<<endl;

**//释放**

for (int x=0;x<n;++x)

delete pa[x];

delete[] pa;

}

**5．思考，如何建立一个二维整数数组类，使得其大小可动态决定，并且访问时，可象普通数组一样使用。如:建立这个二维整数数组类的一个对象obj后, 访问其第二行、第三列元素，可写成：**

**obj[1][2] = 5;**

**cout<<obj[1][2];**

本例用了两个文件，**Array2.h， AppArray2.cpp**

/\*

***\* Array2.h***

\*

\* Author: cw

\*/

#ifndef ARRAY2\_H\_

#define ARRAY2\_H\_

**//ForTest用于条件编译，便于在调试阶段，做一些辅助调试的工作，**

**//这里只是用于显示具体调用的是哪个成员函数。**

**//在最终的Release版本中，会在编译时直接去掉，不参与编译的。**

**//而且没有负作用(如增大exe文件大小，降低运行速度等)**

**#define ForTest**

class Array2

{

public:

public:

Array2(int n, int m) :rows(n), cols(m)

{

pData = new int\*[rows];

for (int i = 0; i < rows; i++)

pData[i] = new int[cols];

}

~Array2()

{

for (int i = 0; i < rows; i++)

delete[] pData[i];

delete[] pData;

}

**//注意函数中***const***的使用目的**

const int \* operator[](int index) const

{

#ifdef ForTest

cout << "调用： const int \* const operator[](int index) const" << endl;

#endif

return ((index >= 0 && index < rows) ? pData[index] : NULL);

}

int \* operator[](int index)

{

#ifdef ForTest

cout << "调用：int \*const operator[](int index)" << endl;

#endif

return ((index >= 0 && index < rows) ? pData[index] : NULL);

}

private:

Array2(const Array2&); **//没有具体实现的定义，可禁止拷贝构造**

const Array2 operator=(const Array2& rhs);

**//同上，禁止此类2维整数数组对象的赋值操作**

private:

int \*\* pData;

int rows;

int cols;

};

#endif /\* ARRAY2\_H\_ \*/

//====================================================================

// Name : ***AppArray2.cpp***

// Author : CW

// Version :

// Copyright : Your copyright notice

// Description : Hello World in C++, Ansi-style

//====================================================================

#include <iostream>

using namespace std;

#include "Array2.h"

**//要弄懂语句为什么错误和正确，正确理解Array2类中的*~~每个const的作用~~*。**

int main()

{

cout<<"演示非常量对象："<<endl;

Array2 **objs(2,3)**;

{

objs[1][2] = 1;

cout<<objs[1][2]<<endl;

int \* p1 = objs[1]; //正确

**const int** \* p2 = objs[1]; //正确

**//objs[1] = p1; //错误，objs[]返回的是个指针值，不是对象(**

**//或变量)不能给指针值赋值**

cout<<"p1[2]="<<p1[2]<<endl;

cout<<"p2[2]="<<p2[2]<<endl;

cout<<"objs[1][2]="<<objs[1][2]<<endl;

p1[2] = 3;

**//p2[2] = 4; //错误，p2为const型**

objs[1][2] = 5;

}

cout<<endl<<"演示常量对象："<<endl;

**const Array2& const\_objs** = objs;

{

//const\_objs[1][2] = 2; **//错误，左侧为const型**

cout<<const\_objs[1][2]<<endl;

//int \* p1 = const\_objs[1]; **//错误，const\*赋给非const\***

const int \* p2 = const\_objs[1]; **//正确**

//const\_objs[1] = NULL; **//错误，左侧为const型**

//cout<<"p1[2]="<<p1[2]<<endl; //未定义p1

cout<<"p2[2]="<<p2[2]<<endl;

cout<<"const\_objs[1][2]="<<const\_objs[1][2]<<endl;

//p1[2] = 3; //未定义p1

//p2[2] = 4; **//错误，p2未const型**

//const\_objs[1][2] = 5; **//错误，左侧为const型**

cout<<const\_objs[1][2]<<endl;

}

return 0;

}