**Lab 8 Part II**

1. 设计一个立方体类 Box，它能计算并输出立方体的体积和表面积。要求：
2. 包含成员变量 m\_a（立方体边长）。
3. 包含函数 SetA(doublea)（设置立方体边长）、GetVolume()（计算体积）、GetArea()

（计算表面积）。

1. 包含函数 Display()，用来输出计算的结果。
2. 设计测试用主函数 main()，用来测试Box 类。
3. 创建一个带非缺省构造函数和析构函数的类，这些函数都显示一些信息来表示它们的存在。写一段代码验证构造函数与析构函数何时被调用。
4. 写一个有拷贝构造函数的类，在拷贝构造函数里用 cout 打印一些信息。写一个函数，这个函数通过传值方式传入新类的对象。写另一个函数，在这个函数内创建这个新类的局部对象，通过传值方式返回这个对象。在main 函数中调用这些函数以证明通过传值方式传递和返回对象时，拷贝构造函数确实悄悄地被调用了。
5. 编写一个Person 类，包括：
6. 普通数据成员：姓名(char \*类型)，性别，年龄。
7. 三个构造函数：无参数构造函数，有参数构造函数（参数：姓名，年龄，性别），拷贝构造函数。
8. 析构函数，输出人员信息函数 print()。
9. 编写 main()函数，分别调用三种构造函数，创建三个对象P1、P2、P3，其中 P3 的创建是用 P2 通过深拷贝复制得到的。

#include<iostream>

using namespace std;

class Person {

public:

const char\* name;

const char\* gender;

int age;

Person() :name{ "null" }, gender {"unknown"},age(-1){}

Person(const char\* nam, const char\* gen, int a) :

name(nam), gender(gen), age(a){}

Person(const Person& someone) {

name = someone.name;

gender = someone.gender;

age = someone.age;

}

~Person() {}

void print() {

cout << name << " 的性别是 " << gender <<

", 年龄是 " << age << "years old" << endl;

}

};

int main() {

Person one1, one2("JiangWei","female",19), one3;

one3 = one2;

one1.print();

one2.print();

one3.print();

return 0;

}

1. 设计一个学生类，保证这个类最多只有一个实例（不可能创建多个类实例），并提供一个访问这个实例的接口函数。（提示：创建实例通常通过构造函数完成，如何定义构造函数才能保证别人不能随便创建实例？）
2. 设计一个类 Stud，包括：
3. 数据成员：学号、姓名和成绩，以及两个静态变量分别存放总分和人数。
4. 有两个普通成员函数 SetData()和 Disp()，分别用于给数据成员赋值和输出数据成员的值。另有一个静态成员函数 Avg()，它用于计算平均分。
5. 一个友元函数 Compare()，用于比较两个学生成绩高低。
6. 在 main()函数中定义一个对象数组并完成对对象的初始化，并求出最高分和最低分的学生。

#include<iostream>

using namespace std;

class Stud{

public:

int id;

const char\* name;

int score;

static int sum ;

static int num ;

void setData(int Id, const char\* nam, int scor) {

id = Id;

name = nam;

score = scor;

sum += score;

num++;

}

void Disp() {

cout << name << "的学号是 " << id << " ,成绩为 " << score << endl;

}

double Avg(){

return double(sum) / num;

}

friend void compare(Stud student1, Stud student2) {

if (student1.score < student2.score)

cout << "the former is better!" << endl;

else if (student1.score == student2.score)

cout << "the same!" << endl;

else

cout << "the latter is better!" << endl;

}

};

int Stud::sum = 0;

int Stud::num = 0;

int main() {

Stud stud[5];

stud[0].setData(1,"a", 100);

stud[1].setData(2, "b", 99);

stud[2].setData(3, "c", 90);

stud[3].setData(4, "d", 93);

stud[4].setData(5, "e", 97);

int student\_score[5];

int max\_score = stud[0].score;

int max\_num = 0;

int min\_score = stud[0].score;

int min\_num = 0;

for (int i = 1; i < 5; i++) {

if (stud[i].score > max\_score)

max\_num = i;

if (stud[i].score < min\_score)

min\_num = i;

}

cout << "5名学生中最高分是 ";

stud[max\_num].Disp();

cout << "5名学生中最低分是 ";

stud[min\_num].Disp();

return 0;

}

1. 写一个包含重载的运算符+、- 、\*、/和赋值符的 number 类。出于效率考虑，为这些函数合理地选择返回值以便以链式写表达式。

//和复数的四则运算同，重载函数以引用为返回值即可链式写表达式，eg:a = b = c;

1. 设计一个Rational 类，进行带分数的运算。要求：
2. 包含两个整数成员变量表示分子(numerator)和分母(denominator)。
3. 包含一个对所声明对象初始化的构造函数。不提供参数时，构造函数应提供默认值。分数存放成简化形式，例如分数“2/4”应在对象中存放成分子 1 和分母 2 的形式。
4. 对下列情况提供 public 成员函数：

a）两个 Rational 值相加，结果保存成简化形式。b）两个Rational 值相减，结果保存成简化形式。c）两个 Rational 值相乘，结果保存成简化形式。d）两个 Rational 值相除，结果保存成简化形式。

e）按 a/b 形式打印Rational 值，其中 a 为分子，b 为分母。

1. 编写主函数，测试 Rational 类。

#include<iostream>

using namespace std;

void fraction(int &a, int &b) {

//最大公因数

int min = a < b ? a : b;

int max = 1;

for (int i = min; i >= 1; i--) {

if (a % i == 0 && b % i == 0) {

max = i;

break;

}

}

a /= max;

b /= max;

}

class Rational {

public:

int numerator;//分子

int denominator;//分母

Rational():numerator(1),denominator(1){}

Rational(int a, int b):numerator(a),denominator(b){}

friend Rational add(Rational c1, Rational c2) {

Rational c;

c.denominator = c1.denominator \* c2.denominator;

c.numerator = c1.numerator \* c2.denominator

+ c2.numerator \* c1.denominator;

fraction(c.denominator, c.numerator);

return c;

}

friend Rational sub(Rational c1, Rational c2) {

Rational c;

c.denominator = c1.denominator \* c2.denominator;

c.numerator = c1.numerator \* c2.denominator

- c2.numerator \* c1.denominator;

fraction(c.denominator, c.numerator);

return c;

}

friend Rational mul(Rational c1, Rational c2) {

Rational c;

c.denominator = c1.denominator \* c2.denominator;

c.numerator = c1.numerator \* c2.numerator ;

fraction(c.denominator, c.numerator);

return c;

}

friend Rational div(Rational c1, Rational c2) {

Rational c;

c.denominator = c1.denominator \* c2.numerator;

c.numerator = c1.numerator \* c2.denominator;

fraction(c.denominator, c.numerator);

return c;

}

void print() {

if (denominator != 1 || numerator != 0)

cout << numerator << " / " << denominator << endl;

else if (numerator == 0)

cout << 0 << endl;

else

cout << numerator << endl;

}

};

int main() {

Rational c1(1, 2), c2(1, 3), c;

(add(c1, c2)).print();

(sub(c1, c2)).print();

(mul(c1, c2)).print();

(div(c1, c2)).print();

return 0;

}

1. 将上述成员函数改为运算符重载的形式，分别作为成员函数和友元函数实现上述功能。

#include<iostream>

using namespace std;

void fraction(int &a, int &b) {

//最大公因数

int min = a < b ? a : b;

int max = 1;

for (int i = min; i >= 1; i--) {

if (a % i == 0 && b % i == 0) {

max = i;

break;

}

}

a /= max;

b /= max;

}

class Rational {

public:

int numerator;//分子

int denominator;//分母

Rational():numerator(1),denominator(1){}

Rational(int a, int b):numerator(a),denominator(b){}

friend Rational& operator+(Rational c1, Rational c2) {

Rational c;

c.denominator = c1.denominator \* c2.denominator;

c.numerator = c1.numerator \* c2.denominator

+ c2.numerator \* c1.denominator;

fraction(c.denominator, c.numerator);

return c;

}

friend Rational& operator- (Rational c1, Rational c2) {

Rational c;

c.denominator = c1.denominator \* c2.denominator;

c.numerator = c1.numerator \* c2.denominator

- c2.numerator \* c1.denominator;

fraction(c.denominator, c.numerator);

return c;

}

friend Rational& operator\* (Rational c1, Rational c2) {

Rational c;

c.denominator = c1.denominator \* c2.denominator;

c.numerator = c1.numerator \* c2.numerator ;

fraction(c.denominator, c.numerator);

return c;

}

friend Rational& operator/ (Rational c1, Rational c2) {

Rational c;

c.denominator = c1.denominator \* c2.numerator;

c.numerator = c1.numerator \* c2.denominator;

fraction(c.denominator, c.numerator);

return c;

}

void print() {

if (denominator != 1 || numerator != 0)

cout << numerator << " / " << denominator << endl;

else if (numerator == 0)

cout << 0 << endl;

else

cout << numerator << endl;

}

};

int main() {

Rational c1(1, 2), c2(1, 3), c;

(c1 + c2).print();

(c1 - c2).print();

(c1 \* c2).print();

(c1 / c2).print();

return 0;

1. }

#include<iostream>

using namespace std;

void fraction(int &a, int &b) {

//最大公因数

int min = a < b ? a : b;

int max = 1;

for (int i = min; i >= 1; i--) {

if (a % i == 0 && b % i == 0) {

max = i;

break;

}

}

a /= max;

b /= max;

}

class Rational {

public:

int numerator;//分子

int denominator;//分母

Rational():numerator(1),denominator(1){}

Rational(int a, int b):numerator(a),denominator(b){}

Rational& operator+( Rational c2) {

Rational c;

c.denominator = denominator \* c2.denominator;

c.numerator = numerator \* c2.denominator

+ c2.numerator \* denominator;

fraction(c.denominator, c.numerator);

return c;

}

Rational& operator- (Rational c2) {

Rational c;

c.denominator = denominator \* c2.denominator;

c.numerator = numerator \* c2.denominator

- c2.numerator \* denominator;

fraction(c.denominator, c.numerator);

return c;

}

Rational& operator\* (Rational c2) {

Rational c;

c.denominator = denominator \* c2.denominator;

c.numerator = numerator \* c2.numerator ;

fraction(c.denominator, c.numerator);

return c;

}

Rational& operator/ (Rational c2) {

Rational c;

c.denominator = denominator \* c2.numerator;

c.numerator = numerator \* c2.denominator;

fraction(c.denominator, c.numerator);

return c;

}

void print() {

if (denominator != 1 || numerator != 0)

cout << numerator << " / " << denominator << endl;

else if (numerator == 0)

cout << 0 << endl;

else

cout << numerator << endl;

}

};

int main() {

Rational c1(1, 2), c2(1, 3), c;

(c1 + c2).print();

(c1 - c2).print();

(c1 \* c2).print();

(c1 / c2).print();

return 0;

}

1. 定义一个二维方阵类 matrix。通过重载二元运算符“+”、“-”、“\*”和一元运算符“~”， 来实现矩阵加、矩阵减、矩阵乘以及矩阵转置。

#include<iostream>

using namespace std;

class Matrix {

private:

int m, n;

double\*\* data;

public:

Matrix(int m1, int n1) :m(m1), n(n1) {

data = new double\*[m1];

for (int i = 0; i < m1; i++)

data[i] = new double[n1];

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

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

data[i][j] = 0;

}

Matrix(const Matrix& src) {

m = src.m;

n = src.n;

data = new double\* [m];

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

data[i] = new double[n];

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

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

data[i][j] = src.data[i][j];

}//析构？

~Matrix() {

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

delete[]data[i];

delete[]data;

}

Matrix& operator=(const Matrix& src);

Matrix operator\*(const Matrix& m2);

void display();

void input();

};

Matrix& Matrix::operator=(const Matrix& src) {

//析构了data\*\*

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

delete[]data[i];

delete[]data;

//开辟一块内存

m = src.m;

n = src.n;

data = new double\* [m];

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

data[i] = new double[n];

//深拷贝

for (int i = 0; i < m; i++) {

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

data[i][j] = src.data[i][j];

}

}

return \*this;

}

Matrix Matrix::operator\*(const Matrix& c2) {

Matrix c3(m, c2.n);

if (n != c2.m) {

cout << "无法进行矩阵计算" << endl;

exit(0);

}

for (int i = 0; i < m; i++) {

for (int j = 0; j < c2.n; j++) {

for (int k = 0; k < n; k++) {

c3.data[i][j] += (data[i][k] \* c2.data[k][j]);

}

}

}

return c3;

}

void Matrix::input() {

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

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

cin >> data[i][j];

}

void Matrix::display() {

for (int i = 0; i < m; i++) {

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

cout << data[i][j] << " ";

}

cout << endl;

}

}

int main() {

int x, y;

cout << "矩阵1行数: ";

cin >> x;

cout << "矩阵1列数: ";

cin >> y;

Matrix A(x, y);

cout << "请输入矩阵1元素（按行，共" << x \* y << "个" << endl;

A.input();

cout << "矩阵1:" << endl;

A.display();

int x1, y1;

cout << "矩阵2行数: ";

cin >> x1;

cout << "矩阵2列数: ";

cin >> y1;

Matrix B(x1, y1);

cout << "请输入矩阵元2素（按行，共" << x1 \* y1 << "个" << endl;

B.input();

cout << "矩阵2:" << endl;

B.display();

Matrix C = A \* B;

cout << "矩阵1 与 矩阵2 的乘积:" << endl;

C.display();

int row;//行

int col;//列

int\*\* arr = new int\* [row];//开辟一块内存存放每一行的地址

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

arr[i] = new int[col];//分别为每一行开辟内存

int m, n;

double\*\* arr1 = new double\* [m];

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

arr1[i] = new double[n];

return 0;

}

Matrix 类定义如下：

class matrix { public:

int r, c; //r 行 c 列的矩阵

int\*\* mem; //矩阵数据

matrix(int a,int b); //二参构造函数，分配 a 行 b 列的 mem 动态数组存放数据

~matrix(); //析构函数，释放 mem 动态数组

matrix operator+ (matrix & m); //矩阵加matrix operator- (matrix & m); //矩阵减matrix operator\* (matrix & m); //矩阵乘matrix operator~ (); //矩阵转置

void display(); //输出矩阵

};

输入

输入一共有 3 行，第一行输入 4 个正整数 r1,c1,r2,c2,分别代表矩阵 x 的行 r1 和列 c1，以及矩阵 y 的行r2 和列c2。其中r1,c1,r2,c2 均大于 0。 第二行输入r1\*c1个整数，代表矩阵x 的元素。第三行输入r2\*c2个整数，代表矩阵y的元素。

输出

输出有 4 部分，每部分之间需要空一行，最后不空行

4 部分分别输出x+y,x-y,x\*y,~x 的结果

如果两个矩阵不能相加、相减或相乘就在对应部分输出“err”

提示：需要在类里面添加另外的函数才能保证正确性！(拷贝构造函数）

matrix(const matrix& src) {

r = src.r;

c = src.c;

mem = new int\* [r];

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

mem[i] = new int[c];

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

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

mem[i][j] = src.mem[i][j];

}

#include<iostream>

using namespace std;

class matrix {

public:

int r, c;//r行c列

int\*\* mem;

matrix(int a, int b);

~matrix();

matrix(const matrix& src) {

r = src.r;

c = src.c;

mem = new int\* [r];

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

mem[i] = new int[c];

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

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

mem[i][j] = src.mem[i][j];

}

matrix operator+(matrix& m);

matrix operator-(matrix& m);

matrix operator\*(matrix& m);

matrix operator~();

void display();

};

matrix::matrix(int a, int b) {

r = a;

c = b;

mem = new int\* [a];

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

mem[i] = new int[b];

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

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

mem[i][j] = 0;

}

matrix::~matrix() {

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

delete[]mem[i];

delete[]mem;

}

matrix matrix::operator+(matrix& m) {

if (c != m.c || r != m.r) {

cout << "err" << endl;

exit(0);

}

matrix add(r, m.c);

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

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

add.mem[i][j] = (mem[i][j] + m.mem[i][j]);

return add;

}

matrix matrix::operator-(matrix& m) {

if (c != m.c || r != m.r) {

cout << "err" << endl;

exit(0);

}

matrix sub(r, m.c);

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

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

sub.mem[i][j] = (mem[i][j] - m.mem[i][j]);

return sub;

}

matrix matrix::operator\*(matrix& m) {

if (c != m.r) {

cout << "err" << endl;

exit(0);

}

matrix mul(r, m.c);

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

for (int j = 0; j < m.c; j++)

for (int k = 0; k < c; k++)

mul.mem[i][j] += (mem[i][k] \* m.mem[k][j]);

return mul;

}

matrix matrix::operator~() {

matrix trans(c, r);

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

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

trans.mem[j][i] = mem[i][j];

return trans;

}

void matrix::display() {

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

for (int j = 0; j < c; j++) {

cout << mem[i][j] << " ";

}

}

int main() {

int r1, c1, r2, c2;

cin >> r1 >> c1 >> r2 >> c2;

matrix x(r1, c1);

matrix y(r2, c2);

for (int i = 0; i < x.r; i++)

for (int j = 0; j < x.c; j++)

cin >> x.mem[i][j];

for (int i = 0; i < r2; i++)

for (int j = 0; j < c2; j++)

cin >> y.mem[i][j];

(x + y).display();

cout << endl;

(x - y).display();

cout << endl;

(x \* y).display();

cout << endl;

x.operator~().display();

return 0;

}