Solution13

191220008 陈南疃

概念题

1、程序中的错误通常包括哪几种？它们分别是由什么原因造成？请举例说明。

（1）**语法错误**：指程序的书写不符合语言的语法规则。

例如：

- 使用了未定义或未声明的标识符
- 左右括号不匹配
- ……

这类错误可由编译程序发现。

（2）**逻辑错误（或语义错误）**：指程序设计不当造成程序没有完成预期的功能。

例如：

- 把两个数相加写成了相乘
- 排序功能未能正确排序
- ……

这类错误可通过对程序进行静态分析和动态测试发现。

（3）**运行异常**：指程序设计对程序运行环境考虑不周而造成的程序运行错误。

例如：

- 对于“x/y”操作，给y输入了“零”。
- 由内存空间不足导致的内存访问错误：int *p=new int; //动态分配空间，可能失败！ *p = 10; //如果上面new操作失败，p可能为空指针！
- 输入数据的数量超过存放它们的数组的大小，导致数组下标越界。
- 多任务环境可能导致的文件操作错误。
- ……

2、异常处理的两种策略是什么？它们分别是怎么做的？为什么不能在析构函数中调用exit?

（1）就地处理：在发现异常错误的地方处理异常

常用做法是调用C++标准库中的函数exit或abort终止程序执行（在cstdlib或stdlib.h中声明）

- abort立即终止程序的执行，不作任何的善后处理工作。
- exit在终止程序的运行前，会做关闭被程序打开的文件、调用全局对象和static存储类的局部对象的析构函数（注意：不要在这些对象类的析构函数中调用exit）等工作。

（2）异地处理：在其它地方（非异常发现地）处理异常

发现异常时，在发现地（如在被调用的函数中）有时不知道如何处理这个异常，或者不能很好地处理这个异常，要由程序的其它地方（如函数的调用者）来处理。

- 通过函数的返回值，或指针/引用类型的参数，或全局变量把异常情况通知函数的调用者，由调用者处理异常。
- 通过语言提供的结构化异常处理机制进行处理（try，throw，catch）。

析构函数中不能调用exit：

exit在终止程序的运行前，会调用static存储类的局部对象的析构函数，如果在析构函数里调用了exit，那么该对象并没有析构结束，然后exit函数又继续调用该类对象的析构函数，从而产生了无穷递归，无法终止。

3、如果不用C++的异常处理机制，应该如何处理在构造函数中发现的异常？

- 方案一：另写一个有返回值的init 函数，将构造函数中的内容写至 init 函数中，然后在构造函数中调用这个 init 函数。若构造过程中出现错误，则可以根据返回值做错误处理（因为构造函数没有返回值，无法判断是否出现错误）。
- 方案二：调用 abort 函数，直接终止程序运行（或者用断言 assert）。
- 方案三：在类中定义一个标志，并将其设置为true，若在构造函数中出现异常，则可以通过在构造后立即检查该标志的方式，对其进行处理。
- 方案四：定义一个单独的(可能是静态的)函数调用在调用构造函数之前立即检查输入参数，若参数不正确，则立即进行处理。

4、如果catch语句不能对异常完全处理，需要调用链中的上层函数进行处理应该怎么办？什么时候需要对catch中的异常对象声明为引用？

有时一个单独的catch语句不能完整的处理某个异常，在执行某些校正操作之后，当前catch可能会决定由调用链更上一层的函数接着处理。一条catch语句通过重新抛出的操作将异常传递给另一个catch。重新抛出仍然是一条throw语句，只是不包含任何表达式，即 `throw;`

很多时候，catch语句会改变其参数的内容。若在改变了参数的内容后catch语句重新抛出异常，则只有当catch异常声明是引用类型时我们队参数所做的改变才会被保留并继续传播。

编程题

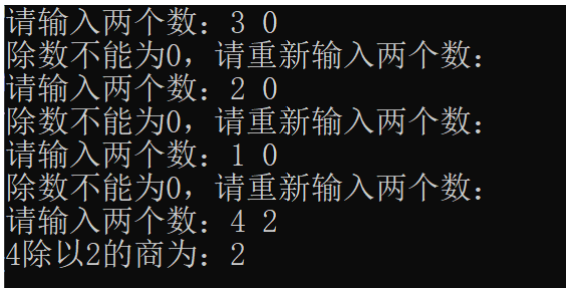
1、 请将除数为0的异常处理程序修改为程序能一直运行到用户输入正确的数据为止。

```
int divide(int x, int y)
{
    if (y == 0) throw 0;
    return x/y;
```

```
}

void f() //其中用到两个数相除操作
{   int a,b;
    while(1)
    {
        try
        {   cout << "请输入两个数：";
            cin >> a >> b;
            int r = divide(a,b);
            cout << a << "除以" << b << "的商为：" << r << endl;
            break;
        }
        catch(int)
        {
            cout << "除数不能为0，请重新输入两个数：";
        }
    }
    ......
}
```

测试结果：



2、 在第五次作业中，你设计了一个矩阵类Matrix，并完成了一些操作符的重载，现在需要你完成下面的任务：

(1) 为了保证程序的鲁棒性，需要你尽可能考虑程序可能出现的异常并抛出；

(2) 实现函数f从键盘读取两个矩阵的数据，根据键盘输入 '+' / '*' 输出矩阵的加法/乘法的结果；

(3) 在f中对出现的异常进行处理，打印错误信息继续运行直到用户输入正确的数据。

```
#include <iostream>
#include <vector>

using namespace std;

class Matrix
{
    int** p_data; //表示矩阵数据
    int row, col; //表示矩阵的行数和列数
public:
    Matrix(int r, int c); //构造函数
    Matrix(const Matrix& m); //拷贝构造函数
```

```
    ~Matrix(); //析构函数
    int*& operator[] (int i); //重载[], 对于Matrix对象m, 能够通过m[i][j]访问第i+1行、第j+1列元素
    Matrix& operator = (const Matrix& m); //重载=, 实现矩阵整体赋值, 若行/列不等, 归还空间并重新分配
    bool operator == (const Matrix& m) const; //重载==, 判断矩阵是否相等
    Matrix operator + (const Matrix& m) const; //重载+, 完成矩阵加法, 可假设两矩阵满足加法条件(两矩阵行、列分别相等)
    Matrix operator * (const Matrix& m) const; //重载*, 完成矩阵乘法, 可假设两矩阵满足乘法条件(this.col = m.row)
    void print();

    friend void f();
};

Matrix::Matrix(int r, int c)
{
    row = r;
    col = c;
    p_data = new int* [row];
    for (int i = 0; i < row; i++)
    {
        p_data[i] = new int[col];
        for (int j = 0; j < col; j++)
            p_data[i][j] = 0;
    }
}

Matrix::Matrix(const Matrix& m)
{
    row = m.row;
    col = m.col;
    p_data = new int* [row];
    for (int i = 0; i < row; i++)
    {
        p_data[i] = new int[col];
        for (int j = 0; j < col; j++)
        {
            p_data[i][j] = m.p_data[i][j];
        }
    }
}

Matrix::~Matrix()
{
    for (int i = 0; i < row; i++)
        delete[]p_data[i];
    delete[]p_data;
}

int*& Matrix::operator[] (int i)
{
    return p_data[i];
}

Matrix& Matrix::operator = (const Matrix& m)
{
    if (row != m.row || col != m.col)
```

```
    {
        this->~Matrix();
        p_data = new int* [m.row];
        for (int i = 0; i < m.row; i++)
        {
            p_data[i] = new int[m.col];
            for (int j = 0; j < m.col; j++)
                p_data[i][j] = 0;
        }
        row = m.row;
        col = m.col;
    }
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            p_data[i][j] = m.p_data[i][j];
        }
    }
    return *this;
}

bool Matrix::operator == (const Matrix& m) const
{
    if (row != m.row || col != m.col)
        return false;
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            if (p_data[i][j] != m.p_data[i][j])
                return false;
        }
    }
    return true;
}

Matrix Matrix::operator + (const Matrix& m) const
{
    if (row != m.row || col != m.col)
        throw -1;
    Matrix matrix(row, col);
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            matrix.p_data[i][j] = p_data[i][j] + m.p_data[i][j];
        }
    }
    return matrix;
}

Matrix Matrix::operator * (const Matrix& m) const
{
    if (col != m.row)
        throw -1;
    Matrix matrix(row, m.col);
    for (int i = 0; i < row; i++)
```

```
    {
        for (int j = 0; j < m.col; j++)
        {
            for (int k = 0; k < col; k++)
            {
                matrix.p_data[i][j] += p_data[i][k] * m.p_data[k][j];
            }
        }
    }
    return matrix;
}

void Matrix::print()
{
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            cout << p_data[i][j] << ' ';
        }
        cout << endl;
    }
}

void f()
{
    while (1)
    {
        char ch;
        try
        {
            while (1)
            {
                try
                {
                    cout << "请输入想要进行的运算符号(\"+\" 或 \"*\"): " << endl;
                    cin >> ch;
                    if (ch != '+' && ch != '*')
                        throw -1.0;
                    break;
                }
                catch (double)
                {
                    cout << "输入的运算符不合法! 请重新输入! " << endl;
                }
            }
            cout << "请依次输入矩阵一的行数、列数和数据: " << endl;
            int row1, col1;
            cin >> row1 >> col1;
            Matrix m1(row1, col1);
            for (int i = 0; i < m1.row; i++)
                for (int j = 0; j < m1.col; j++)
                    cin >> m1.p_data[i][j];
            cout << "请依次输入矩阵二的行数、列数和数据: " << endl;
            int row2, col2;
            cin >> row2 >> col2;
            Matrix m2(row2, col2);
            for (int i = 0; i < m2.row; i++)
```

```
                for (int j = 0; j < m2.col; j++)
                    cin >> m2.p_data[i][j];
            Matrix m3(m1.row, m2.col);
            if (ch == '+')
                m3 = m1 + m2;
            else
                m3 = m1 * m2;
            cout << "矩阵运算结果为：" << endl;
            m3.print();
            break;
        }
        catch (int)
        {
            cout << "请输入符合运算规则的矩阵！请重新输入！" << endl;
        }
    }
}

int main()
{
    f();
    return 0;
}
```

测试结果：

```
请输入想要进行的运算符号("+" 或 "*")：
-
输入的运算符不合法！请重新输入！
请输入想要进行的运算符号("+" 或 "*")：
/
输入的运算符不合法！请重新输入！
请输入想要进行的运算符号("+" 或 "*")：
+
请依次输入矩阵一的行数、列数和数据：
1 2
1 1
请依次输入矩阵二的行数、列数和数据：
2 2
1 2
2 3
请输入符合运算规则的矩阵！请重新输入！
请输入想要进行的运算符号("+" 或 "*")：
+
请依次输入矩阵一的行数、列数和数据：
1 2
1 1
请依次输入矩阵二的行数、列数和数据：
1 2
2 2
矩阵运算结果为：
3 3
```

```
/
输入的运算符不合法！请重新输入！
请输入想要进行的运算符号("+" 或 "*")：
*
请依次输入矩阵一的行数、列数和数据：
1 2
1 1
请依次输入矩阵二的行数、列数和数据：
1 3
1 2 3
请输入符合运算规则的矩阵！请重新输入！
请输入想要进行的运算符号("+" 或 "*")：
*
请依次输入矩阵一的行数、列数和数据：
3 2
1 2
2 3
3 1
请依次输入矩阵二的行数、列数和数据：
2 4
1 2 3 4
4 3 2 1
矩阵运算结果为：
9 8 7 6
14 13 12 11
7 9 11 13
```