Assignment 13

Fu-yun Wang[1]

191300051[1]

1.1程序中的错误包括哪几种，分别是由什么原因造成的，请举例说明。

- 语法错误：书写的程序不符合语言的语法规则，如使用了未定义或者未声明的标识符。
- 逻辑错误：指程序设计不当造成程序没有完成预期的功能。如排序函数书写错误导致排序错误。
- 运行异常：指程序设计对程序运行环境考虑不周而造成程序的运行错误。

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

- 就地处理：在发现异常错误的地方处理异常
- 异地处理：在其它地方处理异常

因为在exit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，因此如果在对象所述的类的析构函数当中调用了exit()，就会出现exit()与析构函数的重复递归调用，最终栈溢出。

1.3 如果不使用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?

对每个对象额外的申请一个比特去描述它的创建过程中是否发生了异常，然后每次希望去使用这个对象的时候去检查它是否被成功的创建了，尽管这是非常糟糕的办法。

参考

How can I handle a constructor that fails?, C++ FAQ (technion.ac.il)

How to handle failure in constructor in C++? - Stack Overflow

构造函数中的异常处理（转） - 追逐更好的自己 - 博客园 (cnblogs.com)

1.4 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怎么办？什么时候需要对catch中的异常对象声明为引用。

（1）不能完全处理，需要交给上层函数进行处理，直接使用语句throw，将检测到的异常交给上层函数进行处理。

（2）

- 当异常对象占用内存较大时，使用引用可以加快参数速度
- 此外，当我们想要修改异常对象时也需要使用引用的方式。

2.1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。

```
#include <iostream>
#include <string>
using namespace std;
int divide(int a,int b)
{
    if (b==0) throw "overflow";
    return a/b;
}
void f()
{
    int a,b;
```

```
    while (true)
    {
        bool flag=true;
        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;

        }
        catch(const char *)
        {
            cout<<"被除数不能为0，请重新输入\n";
            flag=false;
        }
        if (flag)
        {
            break;
        }

    }
}
int main()
{
    try
    {
        f();
    }
    catch(const std::exception& e)
    {
        std::cerr << e.what() << '\n';
    }

}
```

2.2第五次作业中，你设计了一个Matrix类，并完成了一些操作符的重载，现在你需要完成下面的任务。

- 保证程序的鲁棒性，需要你尽可能地考虑程序可能出现的异常并抛出
- 实现函数f从键盘读取两个矩阵的数据，根据键盘输入“+”，“*”的结果
- 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 <string>
#include <cstring>
using namespace std;
template <class T>
class Matrix
{
private:
    int colNum;
    int rowNum;
    T *data;

public:
```

```
14      Matrix(int r = 0, int c = 0)
15      {
16          if (c < 0 || r < 0)
17          {
18              string WaringInformation = "initialize matrix size error";
19              throw WaringInformation;
20          }
21          colNum = c;
22          rowNum = r;
23          try
24          {
25              data = new T[colNum * rowNum];
26          }
27          catch (const bad_alloc &e)
28          {
29              string WarningInformation = "alloc error";
30              throw WarningInformation;
31          }
32          //
33          // for (int i=0;i<r*c;i++)
34          // {
35          // data[i]=T();
36          // }
37          memset(data, 0, sizeof(T) * c * r);
38      }
39      Matrix(int r, int c, T a[])
40      {
41          if (c < 0 || r < 0)
42          {
43              string WaringInformation = "initialize matrix size error";
44              throw WaringInformation;
45          }
46          colNum = c;
47          rowNum = r;
48          try
49          {
50              data = new T[colNum * rowNum];
51          }
52          catch (const bad_alloc &e)
53          {
54              string WarningInformation = "alloc error";
55              throw WarningInformation;
56          }
57          for (int i = 0; i < r * c; i++)
58          {
59              data[i] = a[i];
60          }
61      }
62      Matrix<T> &operator=(const Matrix<T> &m)
63      {
64          colNum = m.colNum;
65          rowNum = m.rowNum;
66          delete[] data;
67          try
68          {
69              data = new T[colNum * rowNum];
70          }
71          catch (const bad_alloc &e)
```

```
        {
            string WarningInformation = "alloc error";
            throw WarningInformation;
        }
        memcpy(data, m.data, sizeof(T) * colNum * rowNum);
        return *this;
    }

    //实现矩阵类
    Matrix operator-()
    {
        Matrix m(rowNum, colNum);
        for (int i = 0; i < colNum * rowNum; i++)
        {
            m[i] = -data[i];
        }
    }
    T *operator[](int index)
    {
        if (index >= rowNum)
        {
            string WarningInformation = "Index out of range";
            throw WarningInformation;
        }
        return &data[index * colNum];
    }
    //函数模板的声明上面也要加!
    //友元函数的class和模板类的class还不能一样?
    template <class Ty>
    friend Matrix<Ty> operator+(const Matrix<Ty> &m1, const Matrix<Ty>
&m2);
    template <class Ty>
    friend Matrix<Ty> operator-(const Matrix<Ty> &m1, const Matrix<Ty>
&m2);
    template <class Ty>
    friend Matrix<Ty> operator*(const Matrix<Ty> &m1, const Matrix<Ty>
&m2);
    template <class Ty>
    friend istream &operator>>(istream &input, Matrix<Ty> &m);
    template <class Ty>
    friend ostream &operator<<(ostream &output, const Matrix<Ty> &m);
};
template <class T>
Matrix<T> operator+(const Matrix<T> &m1, const Matrix<T> &m2)
{
    // assert(m1.colNum == m2.colNum && m1.rowNum == m2.rowNum);
    if (m1.colNum != m2.colNum || m1.rowNum != m2.rowNum)
    {
        string WarningInformation = "No matching Matrixs";
        throw WarningInformation;
    }
    Matrix<T> tmp(m1.rowNum, m1.colNum);
    for (int i = 0; i < tmp.colNum * tmp.rowNum; i++)
    {
        tmp.data[i] = m1.data[i] + m2.data[i];
    }
    return tmp;
}
```

```cpp
template <class T>
Matrix<T> operator-(const Matrix<T> &m1, const Matrix<T> &m2)
{
    // assert(m1.colNum == m2.colNum && m1.rowNum == m2.rowNum);
    if (m1.colNum != m2.colNum || m1.rowNum != m2.rowNum)
    {
        string WarningInformation = "No matching Matrixs";
        throw WarningInformation;
    }
    Matrix<T> tmp(m1.rowNum, m1.colNum);
    for (int i = 0; i < tmp.colNum * tmp.rowNum; i++)
    {
        tmp.data[i] = m1.data[i] - m2.data[i];
    }
    return tmp;
}
template <class T>
Matrix<T> operator*(const Matrix<T> &m1, const Matrix<T> &m2)
{
    // assert(m1.colNum == m2.rowNum);
    if (m1.colNum != m2.rowNum)
    {
        string WarningInformation = "No matching Matrixs";
        throw WarningInformation;
    }
    Matrix<T> tmp(m1.rowNum, m2.colNum);
    for (int i = 0; i < tmp.rowNum * tmp.colNum; i++)
    {
        int r = i / tmp.colNum;
        int c = i % tmp.colNum;
        for (int j = 0; j < m1.colNum; j++)
        {
            tmp.data[i] = tmp.data[i] + m1.data[r * m1.colNum + j] *
m2.data[j * m2.colNum + c];
        }
    }
    return tmp;
}
template <class T>
istream &operator>>(istream &input, Matrix<T> &m)
{
    int a, b;
    input>>a>>b;
    if (input.fail())
    {
        input.clear(); //将失效位清除
        input.sync();//清空输入流
        string WarningInformation="wrong type";
        throw WarningInformation;
    }
    Matrix<T> tmp(a, b);
    for (int i = 0; i < a * b; i++)
        input >> tmp.data[i];
    if (input.fail())
    {
        input.clear(); //将失效位清除
        input.sync();//清空输入流
        string WarningInformation="wrong type";
```

```
        throw WarningInformation;
    }
    m = tmp;
    return input;
}
template <class T>
ostream &operator<<(ostream &output, const Matrix<T> &m)
{
    for (int i = 0; i < m.rowNum; i++)
    {
        for (int j = 0; j < m.colNum; j++)
            output << m.data[i * m.colNum + j] << " ";
        output << endl;
    }
    return output;
}

// //成功实现
// void testMatrix()
// {
//     Matrix<int> m;
//     cin >> m;
//     cout << m;
// }
void f()
{
    Matrix<int> a, b;
    while (true)
    {
        bool flag=true;
        try
        {
            char oper;
            cout<<"请输入操作符+/*"<<endl;
            cin >> oper;
            if (oper!='+' && oper!='*')
            {
                string WarningInformation="No matching operator!";
                throw WarningInformation;
            }
            cout<<"请输入矩阵A"<<endl;
            cin>>a;
            cout<<"请输入矩阵B"<<endl;
            cin>>b;
            if (oper=='*')
            {
                cout<<a*b;
            }
            else
            {
                cout<<a+b;
            }

        }
        catch(const string &s)
        {
            flag=false;
            cerr<<s<<endl;
```

```
            cout<<"请重新输入"<<endl;
        }
        if (flag)
        {
            break;
        }
    }
}
int main()
{
    try{
        f();
    }
    catch(...)
    {
        cout<<"请重新运行本程序"<<endl;
        exit(0);
    }
}
/* 有两点需要记录一下
 * 首先当istream接收到了错误的类型就会失败，可以调用input.fail来判断。且在此处不能直接重新循环，因为input当中仍然保留着值，要先对流进行清空。
 * 在模板类当中定义方法时，如果用到了同类型的数据，可以实例化也可以不是实例化，会默认属于相同的类型，但是在类外定义就必须要实例化。
 */
```