HW13

181220076 周韧哲

一. 概念题

1. 包括语法错误、逻辑错误和运行异常。语法错误指程序的书写不符合语言的语法规则，如左右括号不匹配。逻辑错误指程序设计不当造成程序没有完成预期的功能，如把两个数相加写成了相乘。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，如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。
2. 就地处理和异地处理。就地处理常用做法是调用C++标准库中的函数exit或abort终止程序执行。异地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；或者通过语言提供的结构化异常处理机制进行处理。如果在析构函数中调用exit，会导致死循环，因为exit函数要做终止程序前的结束工作，会调用析构函数。
3. 可以使用断言处理。
4. 如果上层函数也没有能力处理该异常，则异常继续向更上层函数的函数传递。如果在函数调用栈中的所有函数都无法处理抛出的异常，则程序异常中止，这时可以使用自己编写的异常类。当对象构造与拷贝的开销大时，在定义catch语句块时使用引用作为参数。

二. 编程题

1.

```cpp
#include <iostream>
using namespace std;

int divide(int x, int y){
    if(y == 0) throw 0;
    return x/y;
}
void f(){
    int a,b;
    bool right = false;
    while(!right){
        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;
            right = true;
        }catch(int){
            cout << "除数不能为0，请重新输入"<<endl;
            right = false;
        }
    }
}
int main(){
    try{
        f();
    }
    catch (...){
        cout<<"Error!"<<endl;
    }
    return 0;
}
```

2.

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

class Array{
    int *data;
    int length;
public:
    Array(){
        length = 0;
    }
    Array(int c){
        set(c);
    }
    ~Array(){
        delete []data;
    }
    void set(int c){
        length = c;
        data = new int[c];
    }
    int &operator[] (int j){
        if(j>=length || j<0){
            throw (string)"Index Out Of Range!";
        }
        return data[j];
    }
    friend class Matrix;
};

class Matrix{
    Array *p_data;
    int row, col;
    void init(int r, int c){
        row = r;
        col = c;
        p_data = new Array[r];
        for(int i=0;i<row;i++){
            p_data[i].set(c);
        }
    }
public:
    Matrix(){
        row = col = 0;
        p_data = NULL;
    }
    Matrix(int r, int c){
        init(r, c);
    }
    ~Matrix(){
        delete []p_data;
    }
    Array &operator[] (int i){
        if(i>=row || i<0){
            throw (string)"Index Out Of Range!";
```

```
        }
        return p_data[i];
    }
    Matrix &operator= (const Matrix &m){
        if(&m == this) return *this;
        if(row != m.row || col != m.col){
            delete []p_data;
            init(m.row, m.col);
        }
        for(int i=0;i<row;i++){
            for(int j=0;j<col;j++){
                int tmp = m.p_data[i].data[j] ;
                p_data[i].data[j] = tmp;
            }
        }
        return *this;
    }
    bool 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 operator+ (const Matrix &m) const{
        if(this->row!=m.row || this->col!=m.col){
            throw (string)"Dimension Not Match!";
        }
        Matrix tmp(row, col);
        for(int i=0;i<row;i++){
            for(int j=0;j<col;j++){
                tmp[i][j] = p_data[i][j] + m.p_data[i][j];
            }
        }
        return tmp;
    }
    Matrix operator* (const Matrix &m) const{
        if(this->col!=m.row){
            throw (string)"Dimension Not Match!";
        }
        Matrix tmp(this->row, m.col);
        for(int i=0;i<tmp.row;i++){
            for(int j=0;j<tmp.col;j++){
                int sum = 0;
                for(int p=0;p<col;p++){
                    sum += p_data[i][p] * m.p_data[p][j];
                }
                tmp[i][j] = sum;
            }
        }
        return tmp;
    }
    friend ostream &operator<<(ostream &output, const Matrix &M){
        output<<M.row<<" "<<M.col<<"(dimension)"<<endl;
```

```
        for(int i=0;i<M.row;i++){
            for(int j=0;j<M.col;j++){
                cout<<M.p_data[i][j]<<" ";
            }
            cout<<endl;
        }
        return output;
    }
    friend istream &operator>>(istream &input, Matrix &M){
        char tmp;
        input>>M.row>>M.col;
        M.init(M.row, M.col);
        for(int i=0;i<M.row;i++){
            for(int j=0;j<M.col;j++){
                input>>M.p_data[i][j];
            }
        }
        return input;
    }
};

void f(){
    Matrix a, b;
    bool right = false;
    while(!right){
        try{
            char op;
            cout<<"请输入操作符+或*:"<<endl;
            cin>>op;
            if (op!='+' && op!='*'){
                throw (string)"No Matching Operator!";
            }
            cout<<"请输入矩阵A:"<<endl; cin>>a;
            cout<<"请输入矩阵B:"<<endl; cin>>b;
            cout<<"结果为:"<<endl;
            cout<<((op=='+')?(a+b):(a*b));
            right = true;
        }
        catch(const string &s){
            cerr<<s<<endl;
            cout<<"请重新输入!"<<endl;
            right = false;
        }
    }
}

int main(){
    try{
        f();
    }
    catch(...){
        cout<<"请重新运行本程序"<<endl;
        exit(0);
    }
    return 0;
}
```