

Assignment 13

Fu-yun Wang¹

191300051¹

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。

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int divide(int a,int b)
5  {
6      if (b==0) throw "overflow";
7      return a/b;
8  }
9  void f()
10 {
11     int a,b;
```

```

12     while (true)
13     {
14         bool flag=true;
15         try
16         {
17             cout<<"请输入两个数: ";
18             cin>>a>>b;
19             int r=divide(a,b);
20             cout<<a<<"除以"<<b<<"的商为:"<<r<<endl;
21
22         }
23         catch(const char *)
24         {
25             cout<<"被除数不能为0, 请重新输入\n";
26             flag=false;
27         }
28         if (flag)
29         {
30             break;
31         }
32     }
33 }
34
35 int main()
36 {
37     try
38     {
39         f();
40     }
41     catch(const std::exception& e)
42     {
43         std::cerr << e.what() << '\n';
44     }
45 }
46

```

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。

```

1  #include <iostream>
2  #include <string>
3  #include <cstring>
4  using namespace std;
5  template <class T>
6  class Matrix
7  {
8  private:
9      int colNum;
10     int rowNum;
11     T *data;
12
13 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)

```

```

72     {
73         string warningInformation = "alloc error";
74         throw warningInformation;
75     }
76     memcpy(data, m.data, sizeof(T) * colNum * rowNum);
77     return *this;
78 }
79
80 //实现矩阵类
81 Matrix operator-()
82 {
83     Matrix m(rowNum, colNum);
84     for (int i = 0; i < colNum * rowNum; i++)
85     {
86         m[i] = -data[i];
87     }
88 }
89 T *operator[](int index)
90 {
91     if (index >= rowNum)
92     {
93         string warningInformation = "Index out of range";
94         throw warningInformation;
95     }
96     return &data[index * colNum];
97 }
98 //函数模板的声明上面也要加!
99 //友元函数的class和模板类的class还不能一样?
100 template <class Ty>
101 friend Matrix<Ty> operator+(const Matrix<Ty> &m1, const Matrix<Ty>
&m2);
102 template <class Ty>
103 friend Matrix<Ty> operator-(const Matrix<Ty> &m1, const Matrix<Ty>
&m2);
104 template <class Ty>
105 friend Matrix<Ty> operator*(const Matrix<Ty> &m1, const Matrix<Ty>
&m2);
106 template <class Ty>
107 friend istream &operator>>(istream &input, Matrix<Ty> &m);
108 template <class Ty>
109 friend ostream &operator<<(ostream &output, const Matrix<Ty> &m);
110 };
111 template <class T>
112 Matrix<T> operator+(const Matrix<T> &m1, const Matrix<T> &m2)
113 {
114     // assert(m1.colNum == m2.colNum && m1.rowNum == m2.rowNum);
115     if (m1.colNum != m2.colNum || m1.rowNum != m2.rowNum)
116     {
117         string warningInformation = "No matching Matrixs";
118         throw warningInformation;
119     }
120     Matrix<T> tmp(m1.rowNum, m1.colNum);
121     for (int i = 0; i < tmp.colNum * tmp.rowNum; i++)
122     {
123         tmp.data[i] = m1.data[i] + m2.data[i];
124     }
125     return tmp;
126 }

```

```

127 template <class T>
128 Matrix<T> operator-(const Matrix<T> &m1, const Matrix<T> &m2)
129 {
130     // assert(m1.colNum == m2.colNum && m1.rowNum == m2.rowNum);
131     if (m1.colNum != m2.colNum || m1.rowNum != m2.rowNum)
132     {
133         string WarningInformation = "No matching Matrixs";
134         throw WarningInformation;
135     }
136     Matrix<T> tmp(m1.rowNum, m1.colNum);
137     for (int i = 0; i < tmp.colNum * tmp.rowNum; i++)
138     {
139         tmp.data[i] = m1.data[i] - m2.data[i];
140     }
141     return tmp;
142 }
143 template <class T>
144 Matrix<T> operator*(const Matrix<T> &m1, const Matrix<T> &m2)
145 {
146     // assert(m1.colNum == m2.rowNum);
147     if (m1.colNum != m2.rowNum)
148     {
149         string WarningInformation = "No matching Matrixs";
150         throw WarningInformation;
151     }
152     Matrix<T> tmp(m1.rowNum, m2.colNum);
153     for (int i = 0; i < tmp.rowNum * tmp.colNum; i++)
154     {
155         int r = i / tmp.colNum;
156         int c = i % tmp.colNum;
157         for (int j = 0; j < m1.colNum; j++)
158         {
159             tmp.data[i] = tmp.data[i] + m1.data[r * m1.colNum + j] *
160             m2.data[j * m2.colNum + c];
161         }
162     }
163     return tmp;
164 }
165 template <class T>
166 istream &operator>>(istream &input, Matrix<T> &m)
167 {
168     int a, b;
169     input>>a>>b;
170     if (input.fail())
171     {
172         input.clear(); //将失效位清除
173         input.sync(); //清空输入流
174         string WarningInformation="wrong type";
175         throw WarningInformation;
176     }
177     Matrix<T> tmp(a, b);
178     for (int i = 0; i < a * b; i++)
179         input >> tmp.data[i];
180     if (input.fail())
181     {
182         input.clear(); //将失效位清除
183         input.sync(); //清空输入流
184         string WarningInformation="wrong type";

```

```

184         throw warningInformation;
185     }
186     m = tmp;
187     return input;
188 }
189 template <class T>
190 ostream &operator<<(ostream &output, const Matrix<T> &m)
191 {
192     for (int i = 0; i < m.rowNum; i++)
193     {
194         for (int j = 0; j < m.colNum; j++)
195             output << m.data[i * m.colNum + j] << " ";
196         output << endl;
197     }
198     return output;
199 }
200
201 // //成功实现
202 // void testMatrix()
203 // {
204 //     Matrix<int> m;
205 //     cin >> m;
206 //     cout << m;
207 // }
208 void f()
209 {
210     Matrix<int> a, b;
211     while (true)
212     {
213         bool flag=true;
214         try
215         {
216             char oper;
217             cout<<"请输入操作符+/*"<<endl;
218             cin >> oper;
219             if (oper!='+' && oper!='*')
220             {
221                 string warningInformation="No matching operator!";
222                 throw warningInformation;
223             }
224             cout<<"请输入矩阵A"<<endl;
225             cin>>a;
226             cout<<"请输入矩阵B"<<endl;
227             cin>>b;
228             if (oper=='*')
229             {
230                 cout<<a*b;
231             }
232             else
233             {
234                 cout<<a+b;
235             }
236         }
237     }
238     catch(const string &s)
239     {
240         flag=false;
241         cerr<<s<<endl;

```

```
242         cout<<"请重新输入"<<endl;
243     }
244     if (flag)
245     {
246         break;
247     }
248 }
249 }
250 int main()
251 {
252     try{
253         f();
254     }
255     catch(...)
256     {
257         cout<<"请重新运行本程序"<<endl;
258         exit(0);
259     }
260 }
261 /* 有两点需要记录一下
262  * 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。
263  * 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实例化。
264  */
```