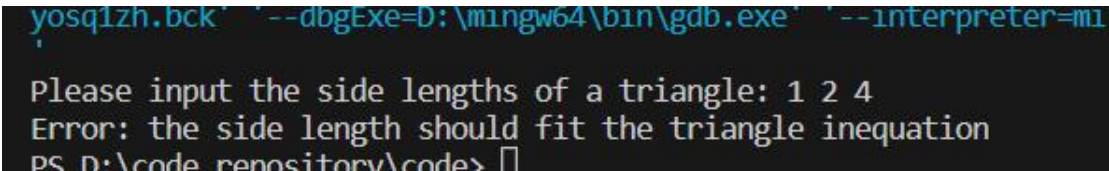
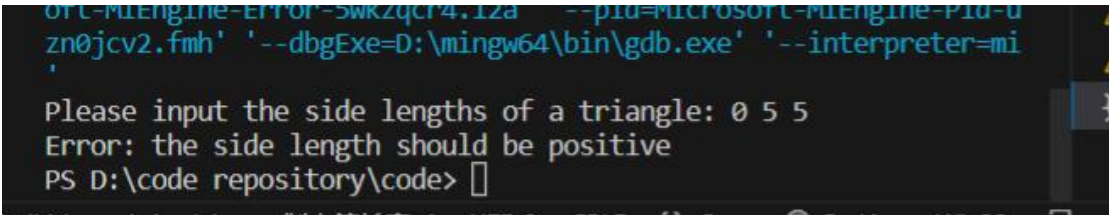


计算机学院 高级语言程序设计 课程实验报告

实验题目：实验 18、 输入/出流文件、异常处理		学号：202200130048
日期：2023. 5. 23	班级： 6	姓名： 陈静雯
Email：1205037094@qq. com		
<p>实验步骤与内容：</p> <ol style="list-style-type: none">1. 练习标准程序库的异常类。第 12 章 PPT，例 12_3。2. 异常类继承。设计一个类 A，再设计一个类 B 公有继承类 A，再设计一个类 C 公有继承类 B。写一个函数 void fun() 在内部分别 throw A 类型、B 类型以及 C 类型的异常对象。在 main 函数中用 try 保护 fun 的调用，并按顺序 catch C 类型，B 类型，A 类型的异常信息，在每个 catch 后的程序块中打印 catch 到了什么类型的异常。请分别尝试在 fun 中 throw A、B、C 类型的异常对象时程序的输出情况，并分析。修改 catch 的先后顺序为 A，B，C 再重复上述实验，分析输出结果。3. 内存分配异常。练习使用 try, catch 语句, 在程序中用 new 分配内存时, 如果操作未成功, 则用 try 语句触发一个 exception 类型异常 e, 用 catch 语句捕获此异常, 之后通过 e.what() 获取 exception 类型异常的错误信息并打印。bonus: 尝试增大 new 分配空间的大小, 多大的时候会出错? 写个程序确定一下分配空间大于哪一个值得时候 new 会出问题, 出的什么问题。4. Array 类模板异常处理。修改 Array.h 中的类模板, 将各种 assert 保护的条通过抛出异常的方式来处理。例如在执行 “[]” 运算符时, 若输入的索引 i 在有效范围外, 抛出 out_of_range 异常。写一个 main 函数检测上述异常的处理情况。将 main 函数和运行结果都进行截图展示。5. 运行如下代码并解释6. 利用 5 中的 A 类运行以下 main 函数并解释输出顺序。		
<p>结论分析与体会：</p> <ol style="list-style-type: none">1. <div></div>		

2.

```
#include <iostream>
#include <cmath>
#include <stdexcept>
using namespace std;
class A{
private:
    int a;
public:
    A() {
        a=0;
    }
    ~A() {}
};
class B:public A{
private:
    int b;
public:
    B() {
        b=1;
    }
    ~B() {}
};
class C:public B{
private:
    int c;
public:
    C() {
        c=2;
    }
    ~C() {}
};
A a;
B b;
C c;
void fun() {
    throw(a);
    //或 throw(b);
    //或 throw(c);
}
int main() {
    try{
        fun();
    }catch(C) {
```

```

        cout<<"catch c"<<endl;
    } catch(B) {
        cout<<"catch b"<<endl;
    } catch(A) {
        cout<<"catch a"<<endl;
    }
}

```

顺序是 CBA

```

vzmvj55.s2a' '--dbgExe=D:\mingw64\bin\gdb.exe' '--interpreter=mi
'
catch a
PS D:\code repository\code>

```

```

vimgedzp.3wc' '--dbgExe=D:\mingw64\bin\gdb.exe' '--interpreter=mi
'
catch b
PS D:\code repository\code>

```

```

hfwfyoa.cdd' '--dbgExe=D:\mingw64\bin\gdb.exe' '--interpreter=mi
'
catch c
PS D:\code repository\code>

```

顺序是 ABC

```

ub2awr1.b1h' '--dbgExe=D:\mingw64\bin\gdb.exe' '--interpreter=mi
'
catch a
PS D:\code repository\code>

```

```

oft-MIEngine-Error-su5hu0wv.1ou' '--pid=Microsoft-MIEngine-Pid-i
k3tvg1.abu' '--dbgExe=D:\mingw64\bin\gdb.exe' '--interpreter=mi
'
catch a
PS D:\code repository\code>

```

```

oft-MIEngine-Error-cvd2rgpi.zpe' '--pid=Microsoft-MIEngine-Pid-i
lzh4kar.pp0' '--dbgExe=D:\mingw64\bin\gdb.exe' '--interpreter=mi
'
catch a
PS D:\code repository\code>

```

```

v 000.cpp cpppp 3
  by earlier handler for 'A' gcc [行 41, 列 3]
  exception of type 'B' will be caught gcc [行 43, 列 3]
  exception of type 'C' will be caught gcc [行 45, 列 3]

```

解释：因为 A 是 B 和 C 的公共基类，所以如果 catch (A) 在前的话，throwABC 都会被捕捉到，而 catchCBA 的话公共基类排在最后，C 和 B 会先被 catchC 和 B

捕捉到

3.

```
#include <iostream>
#include <cmath>
#include <stdexcept>
using namespace std;
void fun(int n) {
    int *s=new int[n]; //分配动态内存
}

int main() {
    try{
        fun(1000000000);
    }catch(exception& e) {
        cout<<e.what();
    }
}
```

4.

```
int main(){
    try{
        Array<int>a(-2);
    }catch(out_of_range& a){
        cout<<a.what()<<endl;
    }
}
```

```
...
r4zzzbi.3ow' '--dbgExe=D:\mingw64\bin\gdb.exe' '--interprete
,
size < 0
PS D:\code repository\code>
```

```
int main(){
    try{
        Array<int>a(5);
        int i=a[6];
    }catch(out_of_range& a){
        cout<<a.what()<<endl;
    }
}
```

```
oft-MIEngine-Error-ui3g2mju.evz' '--pid=Microsoft
3itx031.t0y' '--dbgExe=D:\mingw64\bin\gdb.exe' '-
'
out of array size
PS D:\code repository\code> █
```

```
int main(){
    try{
        Array<int>a(5);
        a.resize(-1);
    }catch(out_of_range& a){
        cout<<a.what()<<endl;
    }
}
```

```
be3cpw1.js3 --dbgExe=D:\mingw64\bin\gdb.exe --int
'
size < 0
PS D:\code repository\code> █
```

5.

```
object 0 is constructed
5 / 2 = 2
object 1 is constructed
8 / 0 = object 1 is deconstructed
object 0 is deconstructed
8 is divided by zero!
That is ok.
PS D:\code repository\code> █
```

解释：divide(8, 0)除数为0，程序捕捉到异常，运行 catch 语句，同时异常抛弃前所有局部对象自动调用析构能力，即从进入 try 起到异常被抛弃期间在栈上

构造的对象都会析构，且析构顺序与构造顺序相反，所以 a1, a0 会进行析构，catch 语句执行完毕之后当前的 try 和 catch 即执行完毕。

6.

```
D:\mingw64\bin\gdb.exe --interpreter=mi
object 1 is constructed
object 2 is constructed
object 3 is constructed
object 1 is deconstructed
object 3 is deconstructed
object 2 is deconstructed
PS D:\code_repository\code>
```

解释：按顺序构造，即 1, 2, 3; delete a1 所以 a1 先析构，main 运行完毕，所以 a2, a3 被析构，而 a3_ptr 析构时会自动将关联的 a3 析构，最后析构 a2