计算机学院 高级程序语言设计 课程实验报告

实验题目：异常处理		学号：202300130150
日期：5.21	班级： 23.4	姓名： 王成意

实验目的：
学习并实践异常处理。

实验步骤与内容：
1. 练习标准程序库的异常类。第 12 章 PPT，例 12_3。
代码：

```
// 12_3.cpp

#include <iostream>

#include <cmath>

#include <stdexcept>

using namespace std;

// 给出三角形三边长，计算三角形面积

double area(double a, double b, double c)

throw(invalid_argument)

{

    // 判断三角形边长是否为正

    if (a <= 0 || b <= 0 || c <= 0)

        throw invalid_argument("the side length should

be positive");

    // 判断三边长是否满足三角不等式

    if (a + b <= c || b + c <= a || c + a <= b)
```

```
        throw invalid_argument("the side length should
fit the triangle inequation");
    // 由 Heron 公式计算三角形面积
    double s = (a + b + c) / 2;
    return sqrt(s * (s - a) * (s - b) * (s - c));
}
int main()
{
    double a, b, c; // 三角形三边长
    cout << "Please input the side lengths of a
triangle: ";
    cin >> a >> b >> c;
    try
    {
        double s = area(a, b, c); // 尝试计算三角形面积
        cout << "Area: " << s << endl;
    }
    catch (exception &e)
    {
        cout << "Error: " << e.what() << endl;
    }
    return 0;
```

```
}
```

运行结果：

```
xe' '--interpreter=mi'
Please input the side lengths of a triangle: 1 2 3
Error: the side length should fit the triangle inequation
PS D:\C++ programs> ^C
PS D:\C++ programs>
PS D:\C++ programs>  & 'd:\VsCode-extentions\.vscode-insiders\extensions\ms
4242fq1u.xnf' '--stdout=Microsoft-MIEngine-Out-uvusdxco.msf' '--stderr=Micr
xe' '--interpreter=mi'
Please input the side lengths of a triangle: 4 5 6
Area: 9.92157
```

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。

代码：

```
#include<cstdio>

#include<stdexcept>

#include<iostream>

using namespace std;

class A

{

    public:

        int data1;

        A(int x=0):data1(x) {}

        void show()
```

```
        {
            cout << data1 << endl;
        }
};
class B:public A
{
    public:
        int data2;
        B(int x,int y):A(x),data2(y) {}
        void show()
        {
            cout << data1 << " " << data2 << endl;
        }
};
class C:public B
{
    public:
        int data3;
        C(int x,int y,int z):B(x,y),data3(z) {}
        void show()
        {
```

```
            cout << data1 << " " << data2 << " " <<
data3 << endl;
        }
};
void fun() throw(A,B,C)
{
    A x(1);
    B y(2, 3);
    C z(4, 5, 6);
    throw z;//x y
}
int main()
{
    try
    {
        fun();
    }
    catch(C& e)
    {
        e.show();
    }
    catch(B& e)
```

```
    {
        e.show();
    }
    catch(A& e)
    {
        e.show();
    }
    return 0;
}
```

发现 catch 的是 c 类型，结果：

```
xe' '--interpreter=mi'
4 5 6
PS D:\C++ programs> ^C
PS D:\C++ programs>
```

更改 catch 顺序：

```
    catch(A& e)
    {
        e.show();
    }
    catch(B& e)
    {
        e.show();
    }
    catch(C& e)
    {
        e.show();
    }
```

发现结果：

```
xe' '--interpreter=
4
PS D:\C++ programs>
```

，可见越是具体的异常类型处理，越是得放在前面。越是通用的异常，越该放后面。

随后针对 throw 的内容进行分析：

首先观察之前的代码，C 型因为继承了 A 型，在 catch 时被转化成了 A 型，只输出了 data1；
随后更改 throw 内容，throw B 型的 y：

```
};
void fun() throw(A,B,C)
{
    A x(1);
    B y(2, 3);
    C z(4, 5, 6);
    throw y;//x z
}
int main()
```

运行发现 catch（A）显示的是：

```
xe' '--interpreter=mi'
2
PS D:\C++ programs>
```

，是 y 中的 data1. 此时若更改 catch 顺序为 CBA：

```
kpejybby.hn1' '--stdout=Micros
xe' '--interpreter=mi'
2 3
```

，分析可知，catch C 的时候因为 throw 的是 B 类型，向上不兼容 C，所以跳过，随后 catch B 类型，匹配，输出，结束。而 ABC 顺序 catch 时，因为 B 类型向下兼容 A 类型，所以在第一个 catch 就匹配了，输出也只有 data1.

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。
代码（未报错）：

```
#include<cstdio>

#include<stdexcept>

#include<iostream>

using namespace std;

int main()
```

```
{
    try
    {
        int *p = new int[100];
        delete[] p;
    }
    catch(const std::exception& e)
    {
        std::cerr << e.what() << '\n';
        return 0;
    }
    cout << "success" << endl;
    return 0;
}
```

结果：

```
xe' '--interpreter=mi'
success
PS D:\C++ programs>
```

增大数组：

```
int *p = new int[100000000000];
delete[] p;
```

结果：

```
xe' '--interpreter=mi'
std::bad_alloc
PS D:\C++ programs>
```

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。

那个临界值在 25264739605 附近（int）

```
5vdi1hql.auo' '--stdout=Microsoft-M
xe' '--interpreter=mi'
std::bad_alloc
PS D:\C++ programs>
```

-1：

```
xe' '--interpreter=mi
success
PS D:\C++ programs>
```

4. Array 类模板异常处理。修改 Array.h 中的类模板，将各种 assert 保护的条件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。

Main：

```cpp
#include<cstdio>

#include<iostream>

#include"array.h"

using namespace std;

int main()

{

    try

    {

        Array<int> a(100);

        a.resize(-1);

    }

    catch(const std::exception& e)
```

```
    {
        std::cerr << e.what() << '\n';
    }

    return 0;
}
```

运行结果：

```
xe' '--interpreter=mi'
Exception:out of range!
PS D:\C++ programs>
```

Main2：

```
#include<cstdio>
#include<iostream>
#include"array.h"
using namespace std;
int main()
{
    try
    {
        Array<int> a(100);
        a[100] = 55;
    }
    catch(const std::exception& e)
    {
```

```
        std::cerr << e.what() << '\n';
    }


    return 0;
}
```

结果 2：

```
p2f5maru.be5' '--stdout=Microsoft
xe' '--interpreter=mi'
Exception:out of range!
PS D:\C++ programs>
```

更改部分：

```
//assert(sz >= 0);  //检查sz是否非负
if(sz<0)
{
    throw out_of_range("Exception:out of range!");
}
```

共计 4 处。

5. 运行如下代码并解释

结果：

```
xe' '--interpreter=mi'
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:\C++ programs>
```

解释：

在运行到输出 8/0 时，一切正常，但是进行函数：divide(8, 0)时：

```
if (y == 0)
    throw x;
```

,抛出异常,导致程序异常终止，此时 try 模块结束按照栈内的顺序先依次析构函数，再进行 catch 中的内容，爆出相应的错误。

最后 catch 模块运行完成后 cout “that is ok“并结束任务。

6. 利用 5 中的 A 类运行以下 main 函数并解释输出顺序。

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

class A{
    int id;
public:
    A(int i):id(i){
        cout<<"object "<<id<<" is constructed"<<endl;
    }
    ~A(){
        cout<<"object "<<id<<" is deconstructed"<<endl;
    }
};
int main() {
    A *a1 = new A(1);
    A a2(2);
    A *a3 = new A(3);
    auto_ptr<A> a3_ptr(a3);
    delete a1;
}
```

运行结果：

```
object 1 is constructed
object 2 is constructed
object 3 is constructed
object 1 is deconstructed
object 3 is deconstructed
object 2 is deconstructed
```

Auto_ptr 是智能指针类模板，其内部定义了管理指针的机制，可以将 new 获得的地址直接赋值给这个对象，在对象的生命周期截止时会自动调用析构函数来释放相应的内存。在这个程序里，析构时 delete a1 之后，auto_ptr 自动检测到周期结束，于是自动调用了 delete，释放掉了这个对象申请 new 的内存，表现在析构顺序：1，3，2。

结论分析与体会：

非常好的实验，使我进一步了解了 c++异常处理。