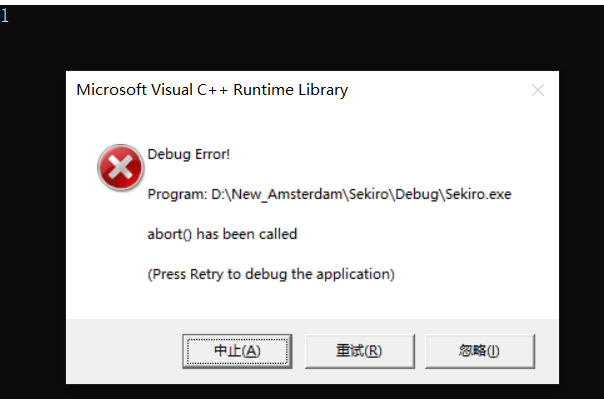
EX1



#include<iostream>

#include<stdexcept>

using namespace std;

class TestException :public runtime\_error

{

public:

TestException() :runtime\_error("Testing Error!\n")

{

}

};

int main()

{

int code = 0;

cin >> code;

if (code == 1)

{

try

{

throw TestException();

}

catch (TestException())

{

throw TestException();

}

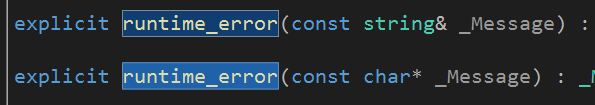
}

}

问题与思考：

VS2019下直接调用了abort()来终止程序；

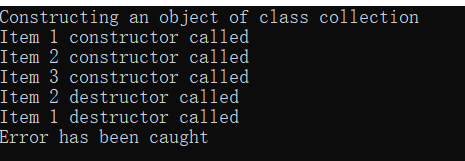
Runtime\_error类的两个构造函数（如下图）



均不允许无参数，本处随意写了一个const char\*丢进去

下方EX习题均选择了传nullptr（vs2010支持的c++11特性之一）

EX2



在构造函数中尝试：

#include<iostream>

#include<stdexcept>

#include<vector>

using namespace std;

class Item

{

int value;

public:

Item(int v) :value(v)

{

printf("Item %d constructor called\n", value);

if (value == 3)

{

throw runtime\_error("An exception was thrown\n");

}

}

~Item()

{

printf("Item %d destructor called\n", value);

}

};

class Collection

{

public:

Collection()

{

printf("Constructing an object of class collection\n");

Item item1(1);

Item item2(2);

Item item3(3);

Item item4(4);

}

};

int main()

{

try

{

Collection();

}

catch (runtime\_error)

{

printf("Error has been caught\n");

}

}

在主函数中尝试：

#include<iostream>

#include<stdexcept>

#include<vector>

using namespace std;

class Item

{

int value;

public:

Item(int v) :value(v)

{

printf("Item %d constructor called\n", value);

if (value == 3)

{

throw runtime\_error("An exception was thrown\n");

}

}

~Item()

{

printf("Item %d destructor called\n", value);

}

};

class Collection

{

public:

Collection()

{

printf("Constructing an object of class collection\n");

Item item1(1);

Item item2(2);

Item item3(3);

Item item4(4);

}

};

int main()

{

try

{

Item item1(1);

Item item2(2);

Item item3(3);

Item item4(4);

}

catch (runtime\_error)

{

printf("Error has been caught\n");

}

}

问题与思考：

不只是在构造函数里面才会出现这样“对在异常之前已经创建的对象调用她们的析构函数”，事实上在主函数中也是这样，上方已提供代码。在异常抛出后调试中可以看到程序直接跳转到对象析构函数中。显然在item3抛出异常后构造的item4构造都不会执行（已经跳出Collection去找catch）

动态申请内存的对象在上述条件下不会自行销毁。应注意这一点。

EX3



#include<iostream>

#include<stdexcept>

#include<vector>

#include<exception>

using namespace std;

class TestException :public runtime\_error

{

public:

TestException():runtime\_error(nullptr)

{

}

virtual const char\* what() const

{

return "TestException";

}

};

void g()

{

try

{

throw TestException();

}

catch (exception& e)

{

printf("%s caught in function g(). Rethrowing\n", e.what());

throw;

}

}

int main()

{

try

{

g();

}

catch(exception& e)

{

printf("%s caught in function main()\n",e.what());

}

}

问题与思考：

尝试重载虚函数virtual char const\* what（）const来使异常能够表明自己的身份，不过应注意这个函数是个const 常函数，初次尝试因此失败过。

这个include进来exception头文件，使用exception基类来接受任意类型异常，引用是为了避免调用一次拷贝。

当然你也可以用这个省略号：



上方throw加异常构造的表达式可以抛出一个异常，但若需要“同一个异常”则直接使用throw，可以抛出当前catch的异常本身。

EX4



#include<iostream>

#include<exception>

using namespace std;

class TestException :public runtime\_error

{

public:

TestException() :runtime\_error(nullptr)

{

}

const char\* what() const

{

return "TestException";

}

};

void h()

{

throw TestException();

}

void g()

{

h();

}

void f()

{

g();

}

int main()

{

try

{

f();

}

catch(exception& e)

{

printf("In main: Caught %s", e.what());

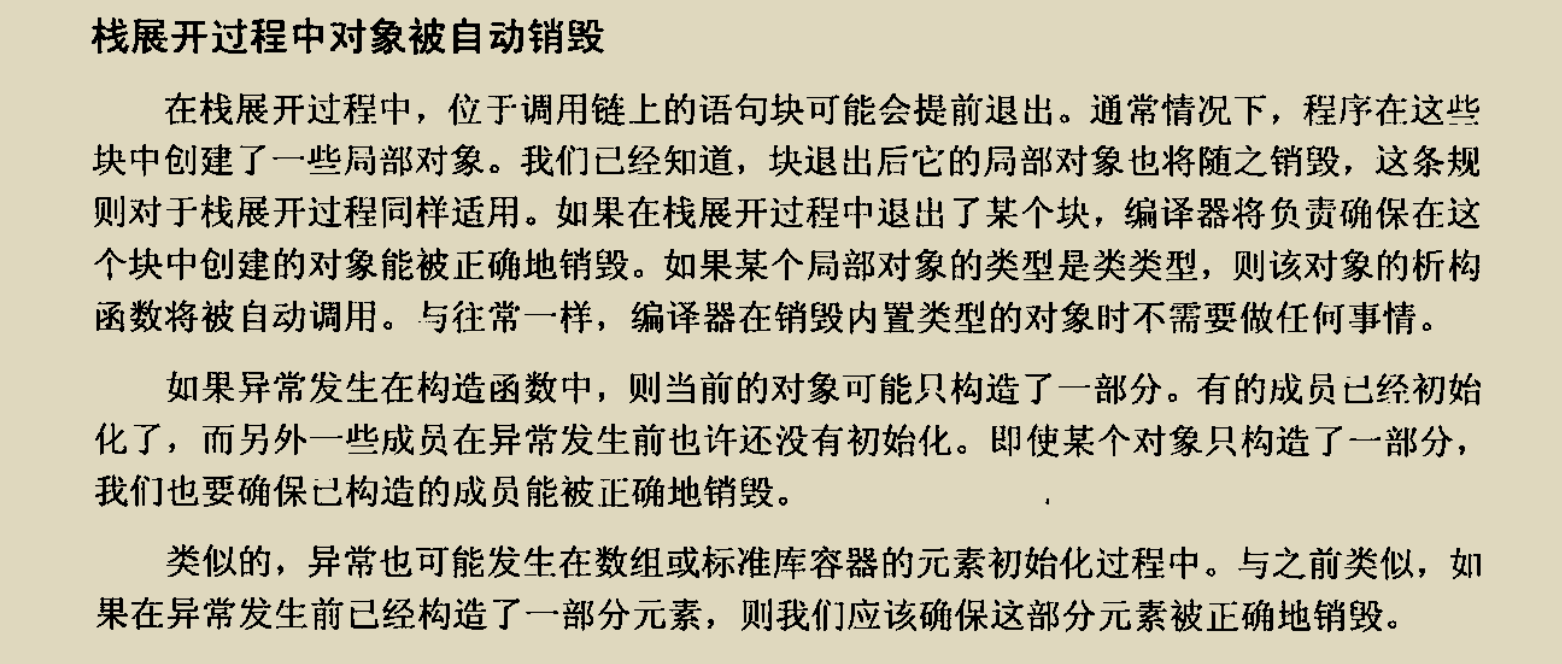
}

}

问题与思考：

栈展开过程中异常之前的对象会被析构掉

对于下方：“我们应该确保这部分元素正确销毁”不知是如何操作。



析构函数不该出现异常

