Kolbe Williams-Wimmer

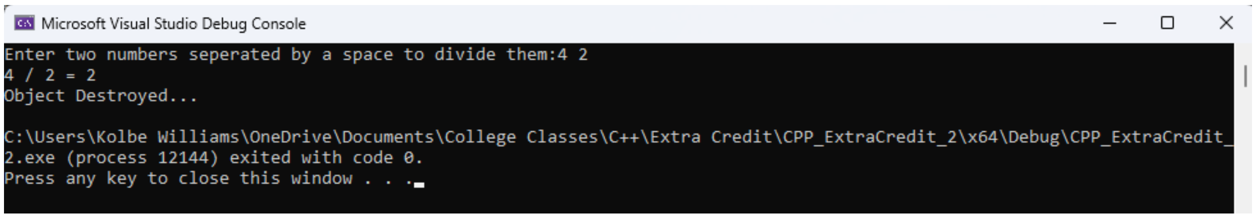
Part 1: Write a program that demonstrates that objects created in a try block are automatically destroyed if an exception is thrown.

The following program contains a class called Divide, and an object of that class is created inside of a try block. Within the class definition is a destructor that executes whenever the object gets destroyed, and it contains a message indicating that the object has been destroyed. If a member of the object created in the try block (the number being divided by) is equal to zero, an exception is thrown that displays an error message that is located in the catch block. Otherwise, the result of the division is displayed, and the code for this display is also located in the try block. If the object destruction message is executed before the error message, the object was automatically destroyed before the catch block executed as soon as the exception was thrown. If the object destruction message displays after printing the result of the division, then the object was destroyed after the entire try block ran.

Source Code:

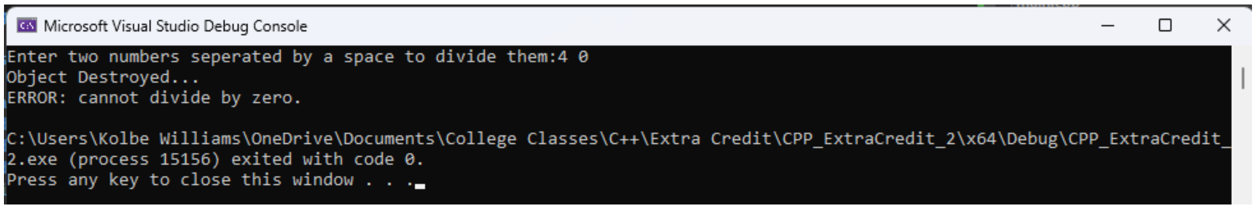
```
#include <iostream>

using namespace std;

class Divide
{
private:
	double a;
	double b;
	double c;
public:
	Divide() //constructor
	{
		a = 1.0;
		b = 1.0;
		c = 1.0;
	}
	double getA()
	{ return a; }
	double getB()
	{ return b; }
	void setA(int num)
	{ a = num; }
	void setB(int num)
	{ b = num; }
	double calculate()
	{ c = a / b; }
	~Divide() //Destructor is called when the object is destroyed and prints the
message
	{
		cout << "Object Destroyed...\n";
	}
};

int main()
{
	double a, b;
	cout << "Enter two numbers seperated by a space to divide them:";
	cin >> a >> b;
	try
	{
		Divide ans; //object created in the try block
		ans.setA(a);
		ans.setB(b);
		if (b == 0)
			throw "ERROR: cannot divide by zero.\n"; //The destructor
message will execute
		//before the error message if the ans object is destroyed when the
exception is thrown
		int c = a / b;
		cout << a << " / " << b << " = " << c << endl; //This will only be
displayed if no
		//exception is thrown
	}
	catch (const char* msg)
	{
		cout << msg;
	}
	return 0;
}
```

Output:


Microsoft Visual Studio Debug Console

```
Enter two numbers seperated by a space to divide them:4 2
4 / 2 = 2
Object Destroyed...

C:\Users\Kolbe Williams\OneDrive\Documents\College Classes\C++\Extra Credit\CPP_ExtraCredit_2\x64\Debug\CPP_ExtraCredit_
2.exe (process 12144) exited with code 0.
Press any key to close this window . . .
```

The object is destroyed after the entire try block executes and the result of the division is displayed.


Microsoft Visual Studio Debug Console

```
Enter two numbers seperated by a space to divide them:4 0
Object Destroyed...
ERROR: cannot divide by zero.

C:\Users\Kolbe Williams\OneDrive\Documents\College Classes\C++\Extra Credit\CPP_ExtraCredit_2\x64\Debug\CPP_ExtraCredit_
2.exe (process 15156) exited with code 0.
Press any key to close this window . . .
```

When an exception is thrown, the object is automatically destroyed before the try block finishes executing and before the error message is displayed from the catch block.

Part 2: Demonstrate unwinding the stack after an exception is thrown.

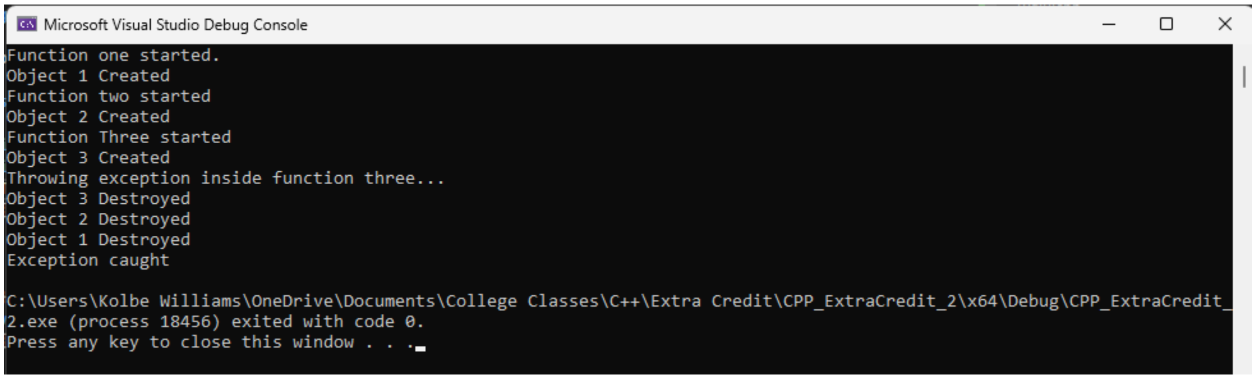
In the following program, there is a class called Test that contains a constructor, which displays an object created message, and a destructor, which contains an object destroyed message. There are also three functions that are named functionOne, functionTwo, and functionThree. FunctionOne creates an object of the Test class and calls functionTwo. functionTwo creates an object of the test class and calls functionThree, and functionThree creates an object of the test class and then throws an exception. In the main function, there is a try block that calls functionOne and a catch block that catches the exception that functionThree throws. When the program runs, an object is created from each of the functions labeled object 1, object 2, and object 3 in the output. Then, when the exception is thrown in functionThree just after object 3 is created, the objects get destroyed in reverse order before the catch block executes. This demonstrates unwinding the stack by showing the destructors of each of the objects that were created being executed in reverse order. This indicates that when the exception was thrown, the newly created objects on the top of the stack were destroyed from the top of the stack back down to how the stack was before the try block executed. Then, after the stack unwinds, the catch block executes.

Source Code:

```cpp
#include <iostream>

using namespace std;

int objNum = 0; //global variable used to keep up with object number

class Test
{
public:
	int obj; //object number for easily understandable output
	Test() //constructor
	{
		++objNum;
		obj = objNum;
		display();
	}
	void display()
	{
		cout << "Object " << obj << " Created\n";
	}
	~Test() //destructor
	{
		cout << "Object " << obj << " Destroyed\n";
	}
};

void functionThree() //creates an object and then throws an exception
{
	cout << "Function Three started\n";
	Test obj;
	cout << "Throwing exception inside function three...\n";
	throw 1;
}
```

```cpp
void functionTwo() //creates an object and calls functionThree
{
        cout << "Function two started\n";
        Test obj;
        functionThree();
        cout << "FunctionTwo ended\n"; //This will never execute
}

void functionOne() //creates an object and calls functionTwo
{
        cout << "Function one started.\n";
        Test obj;
        functionTwo();
        cout << "Function one ended\n"; //This will never execute
}

int main()
{
        try
        {
                /*functionOne is called inside the try block, which creates object 1
and calls
                functionTwo. Then, functionTwo creates object 2 and calls
functionThree. functionThree
                then creates object three and throws and exception. When the exception
is thrown, the
                stack unwinds and the obects are destroyed in the reverse order from
what they were
                created in.*/
                functionOne();
        }
        catch (int x)
        {
                cout << "Exception caught\n"; /*This executes when the exception is
caught after the
                stack unwinds*/
        }
        return 0;
}
```

Output:



```
Function one started.
Object 1 Created
Function two started
Object 2 Created
Function Three started
Object 3 Created
Throwing exception inside function three...
Object 3 Destroyed
Object 2 Destroyed
Object 1 Destroyed
Exception caught

C:\Users\Kolbe Williams\OneDrive\Documents\College Classes\C++\Extra Credit\CPP_ExtraCredit_2\x64\Debug\CPP_ExtraCredit_
2.exe (process 18456) exited with code 0.
Press any key to close this window . . .
```