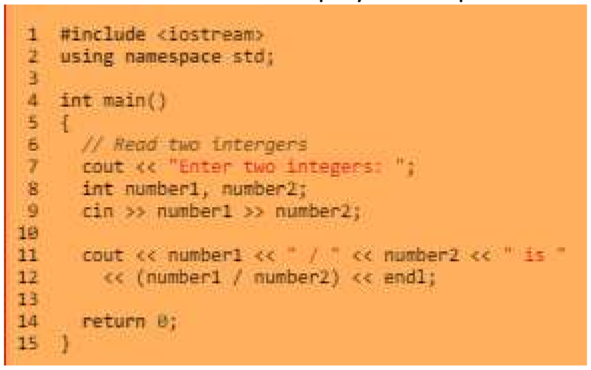
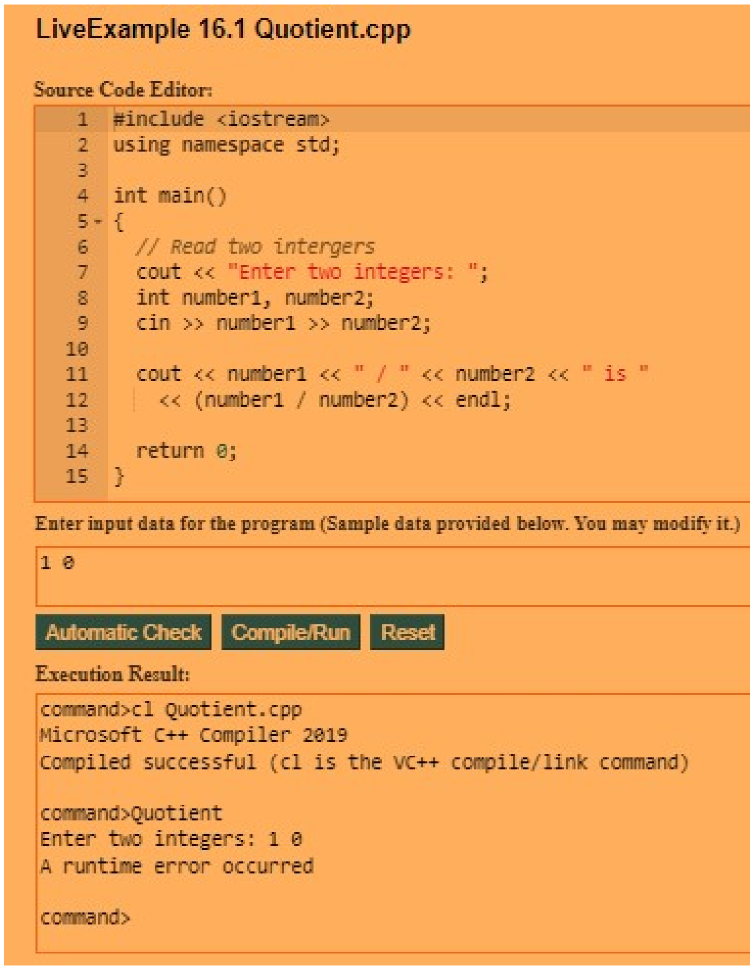
16.2 Exception-Handling Overview

Sunday, April 23, 2023 9:50 AM

- An exception is thrown using a throw statement and caught in a try-catch block, exception handling enables the called of the fn to process the exception thrown from a fn
- To demonstrate exception handline including how an exception is made and thrown, begin w/ ex that reads in 2 ints and displays their quotient
- 

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

int main()
{
  // Read two intergers
  cout << "Enter two integers: ";
  int number1, number2;
  cin >> number1 >> number2;

  cout << number1 << " / " << number2 << " is "
    << (number1 / number2) << endl;

  return 0;
}
```

- 

LiveExample 16.1 Quotient.cpp

Source Code Editor:

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

int main()
{
  // Read two intergers
  cout << "Enter two integers: ";
  int number1, number2;
  cin >> number1 >> number2;

  cout << number1 << " / " << number2 << " is "
    << (number1 / number2) << endl;

  return 0;
}
```

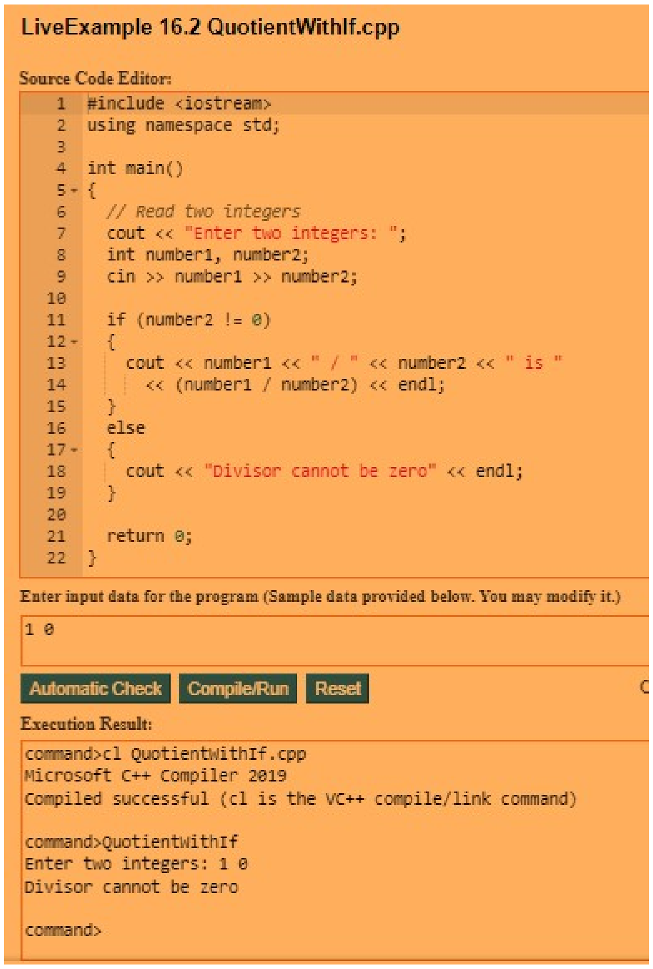
Enter input data for the program (Sample data provided below. You may modify it.)

```
1 0
```

Execution Result:

```
command>cl Quotient.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>Quotient
Enter two integers: 1 0
A runtime error occurred

command>
```

- If enter 0 for the second numb, a runtime error happens bc can't divide an int by 0
- Floating-pt numb divided by 0 does not raise an exception
- Simple way to fix error is add an if statement to test the second numb

- 

LiveExample 16.2 QuotientWithIf.cpp

Source Code Editor:

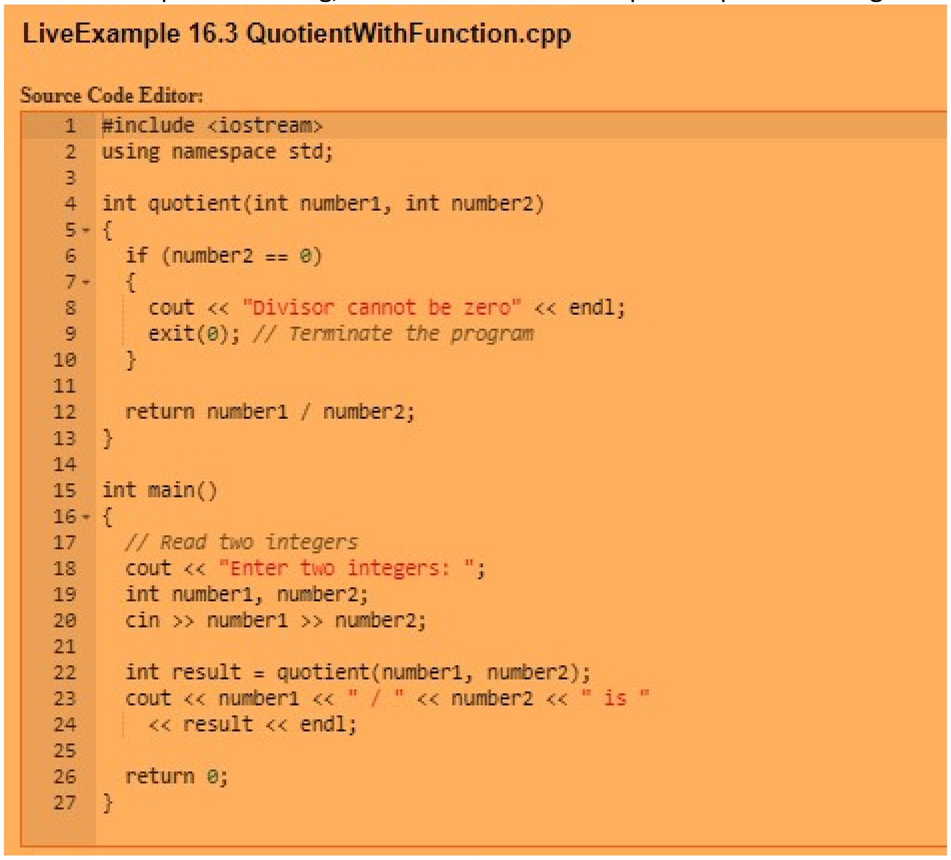
```cpp
#include <iostream>
using namespace std;

int main()
{
  // Read two integers
  cout << "Enter two integers: ";
  int number1, number2;
  cin >> number1 >> number2;

  if (number2 != 0)
  {
    cout << number1 << " / " << number2 << " is "
      << (number1 / number2) << endl;
  }
  else
  {
    cout << "Divisor cannot be zero" << endl;
  }

  return 0;
}
```

Enter input data for the program (Sample data provided below. You may modify it.)

```
1 0
```

Execution Result:

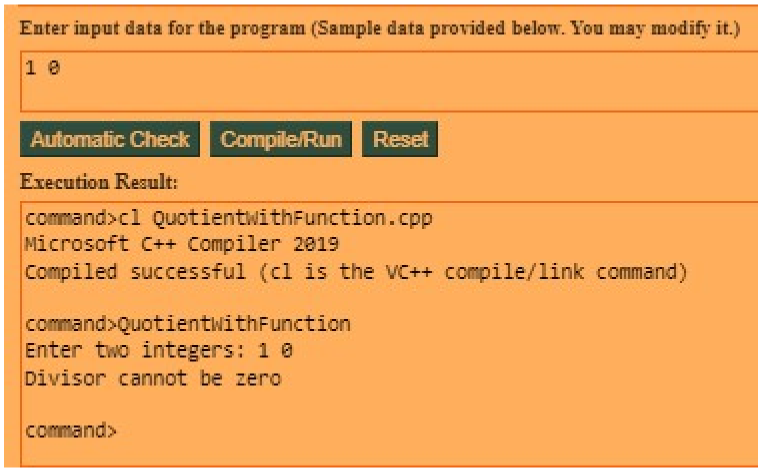
```
command>cl QuotientWithIf.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>QuotientWithIf
Enter two integers: 1 0
Divisor cannot be zero

command>
```

- B4 intro exception handling, lets rewrite that to compute a quotient using a fn
- 

LiveExample 16.3 QuotientWithFunction.cpp

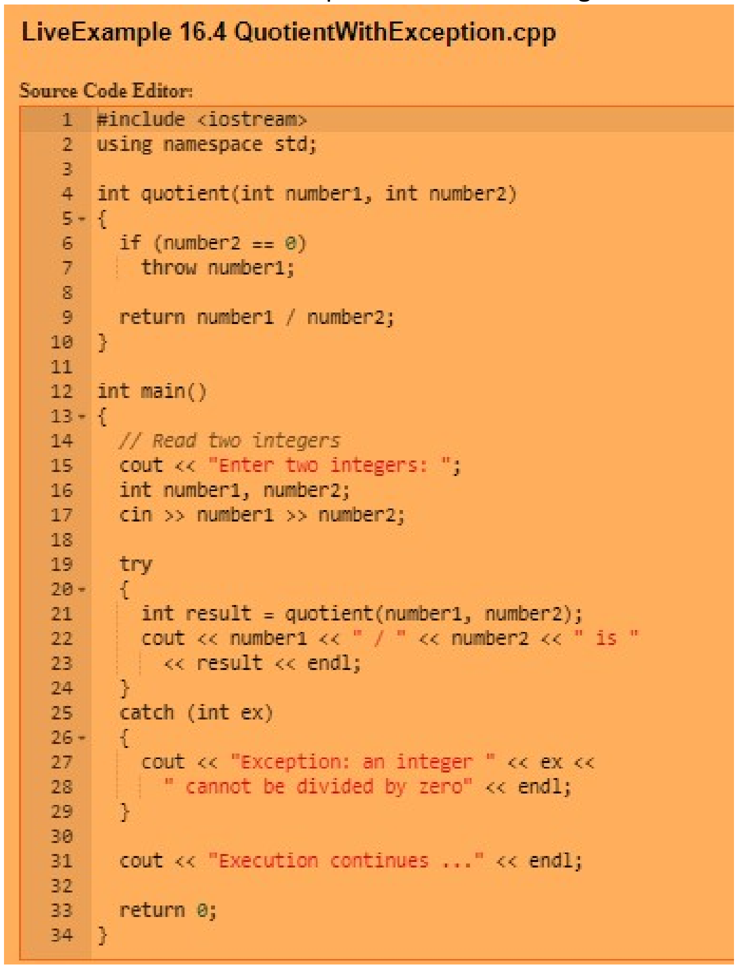
Source Code Editor:

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

int quotient(int number1, int number2)
{
  if (number2 == 0)
  {
    cout << "Divisor cannot be zero" << endl;
    exit(0); // Terminate the program
  }

  return number1 / number2;
}

int main()
{
  // Read two integers
  cout << "Enter two integers: ";
  int number1, number2;
  cin >> number1 >> number2;

  int result = quotient(number1, number2);
  cout << number1 << " / " << number2 << " is "
    << result << endl;

  return 0;
}
```

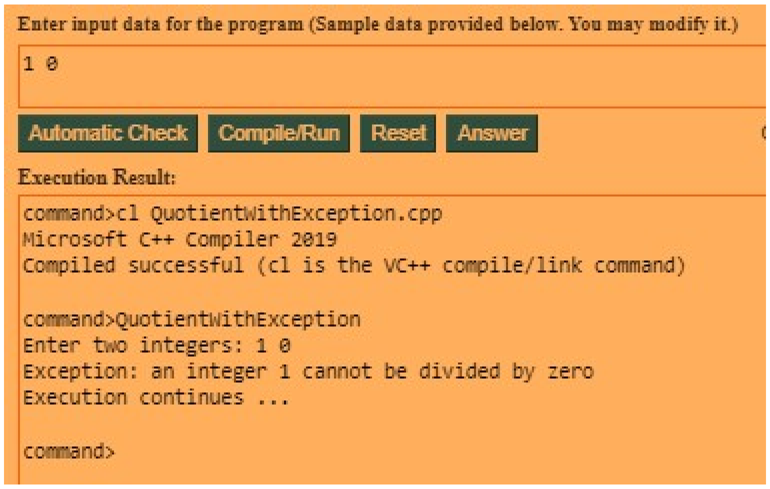
- 
Enter input data for the program (Sample data provided below. You may modify it.)
Automatic Check
Compile/Run
Reset
Execution Result:

```
1 0
```

```
command>cl QuotientWithFunction.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>QuotientWithFunction
Enter two integers: 1 0
Divisor cannot be zero

command>
```

- Analysis
  - @ lns 4-13 – quotient returns the quotient of 2 integers
  - @ ln 9 – if numb2 = 0, it cannot return a value, so the pgrm is terminated at this line using the exit(0) fn to terminate the pgrm
- Should not let the fn terminate the pgrm, the caller should decide whether to terminate the pgrm
- C++ lets a fn throw an exception that can be caught and handled by the caller
- 

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

int quotient(int number1, int number2)
{
  if (number2 == 0)
    throw number1;

  return number1 / number2;
}

int main()
{
  // Read two integers
  cout << "Enter two integers: ";
  int number1, number2;
  cin >> number1 >> number2;

  try
  {
    int result = quotient(number1, number2);
    cout << number1 << " / " << number2 << " is "
      << result << endl;
  }
  catch (int ex)
  {
    cout << "Exception: an integer " << ex <<
      " cannot be divided by zero" << endl;
  }

  cout << "Execution continues ..." << endl;

  return 0;
}
```

- 
Enter input data for the program (Sample data provided below. You may modify it.)
Automatic Check
Compile/Run
Reset
Answer
Execution Result:

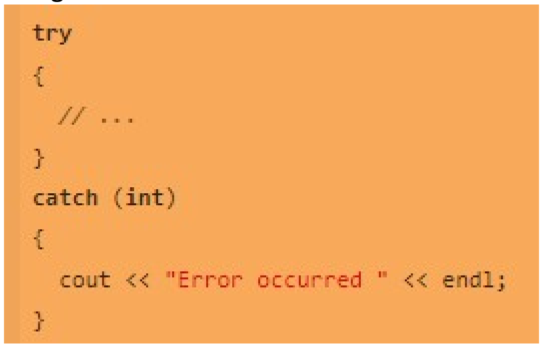
```
1 0
```

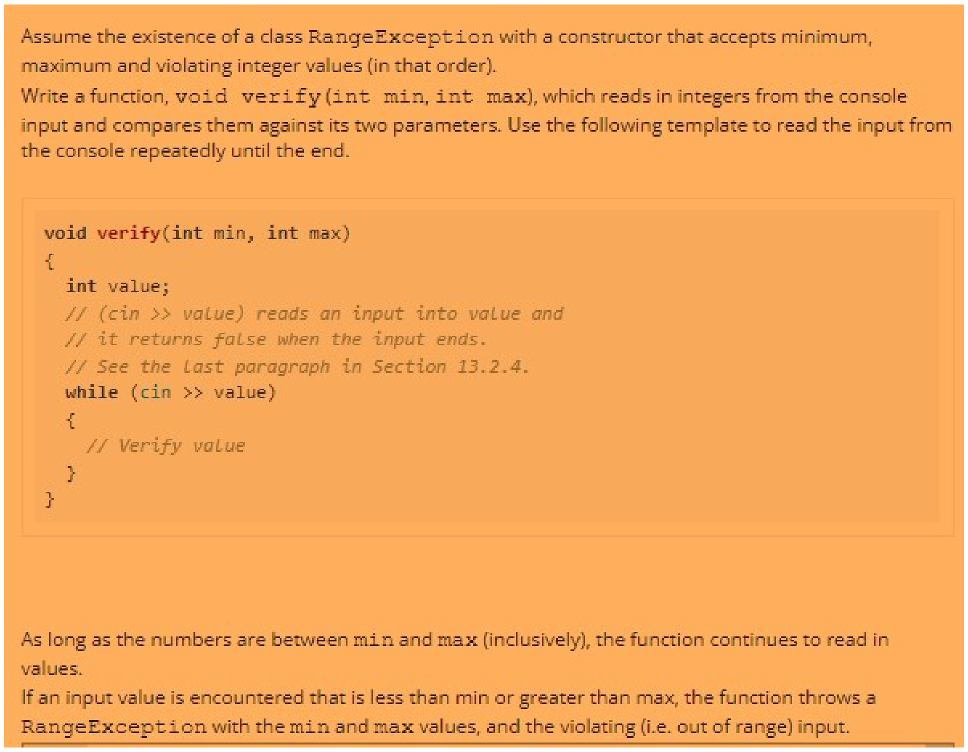
```
command>cl QuotientWithException.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>QuotientWithException
Enter two integers: 1 0
Exception: an integer 1 cannot be divided by zero
Execution continues ...

command>
```

- Analysis
  - @ ln 7 – if number2 is 0, the fn quotient throws an exception by executing throw number1;
- The value thrown (for this is number1) is called an exception
- The execution of a throw statement is called throwing an exception
- Can throw a value of any type, in this case its int
- When an exception is thrown, the normal execution flow is interrupted
- As the name suggests, to "throw an exception" is to pass the exception from one place to another
- The statement for invoking the fn is contained in a try block
  - @ ln 19-24 – the try block has the code that is executed in normal circumstances
  - @ ln 31 – the catch block catches an exception, after this, the statements after the catch block are executed
- The throw statement is same a a fn call, but instead of calling a fn, it calls a catch block
- In same sense, a catch block is like a fn definition w/ a param that matches the type of the value being thrown
- But after the cath block executed, the pgrm ctrl doesn't return to the throw statement, instead, executes the statement after the catch block
- The identifier e in the catch block header

-
  ```
  catch (int ex)
  ```
- Acts like a param in a fn, so it is referred to as a catch blck param
- The type (like int) before ex specifies the kind of exception the catch block can catch
- After exception is caught, can access the thrown value from this param in the body of a catch block
- Summary, a template for a try-throw-catch block can look this:

-
  ```
  try
  {
    Code to try;
    Throw an exception with a throw statement or
      from function if necessary;
    More code to try;
  }
  catch (type ex)
  {
    Code to process the exception;
  }
  ```
- An exception can be thrown directly using a throw statement in a try block / a fn can be invoked that throws an exception
- Back to analysis

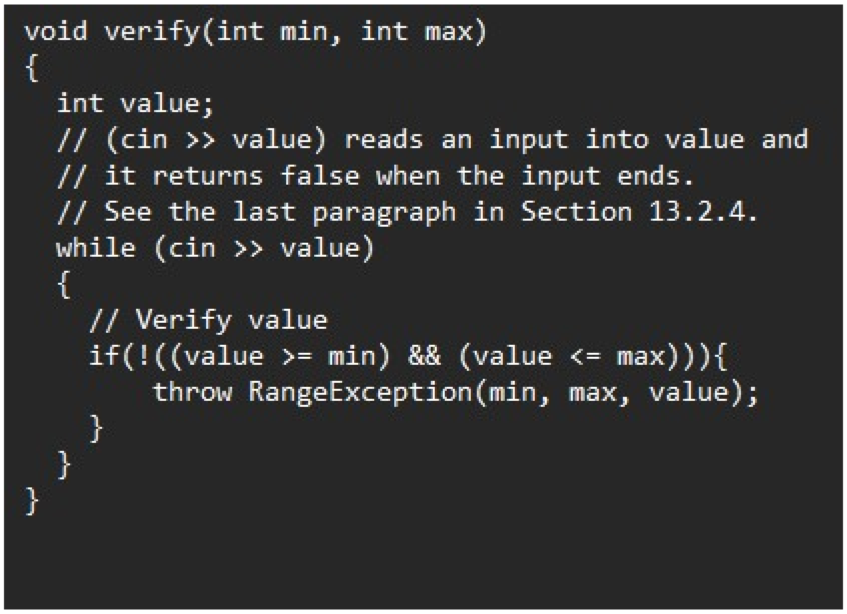
  - @ ln 4-10 – quotient returns the quotient of 2 integers, if number2 is 0, cannot return a value, so an exception is thrown in ln 7
  - @ ln 21 – the main fn invokes the quotient fn, if the quotient fn executes normally, it returns a value to the caller, if the quotient fn encounters an exception, it throws the exception back to its caller, the caller's catch block handles the exception
- So there's advantages of using exception handling
  - Lets fns throw an exception to its caller, w/out this, a fn must handle the excepetion / terminate the pgrm
  - Usually, the called fn doesn't know what to do for an error, usually in library fns
  - The library fn can detect the error, but only caller knows what needs to be done when error happens
  - Basic idea of exception handling is to separate error detection (done in called fn) from error handling (dun in calling fn)
- If not interested in contents of an exception object, the catch block param can be omitted, like this is legal
- 

```
try
{
  // ...
}
catch (int)
{
  cout << "Error occurred " << endl;
}
```
-
-
- 

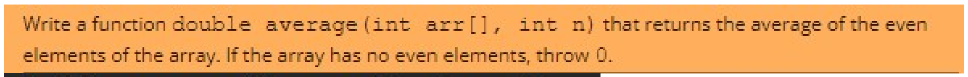
Assume the existence of a class `RangeException` with a constructor that accepts minimum, maximum and violating integer values (in that order).
Write a function, `void verify(int min, int max)`, which reads in integers from the console input and compares them against its two parameters. Use the following template to read the input from the console repeatedly until the end.

```
void verify(int min, int max)
{
  int value;
  // (cin >> value) reads an input into value and
  // it returns false when the input ends.
  // See the last paragraph in Section 13.2.4.
  while (cin >> value)
  {
    // Verify value
  }
}
```

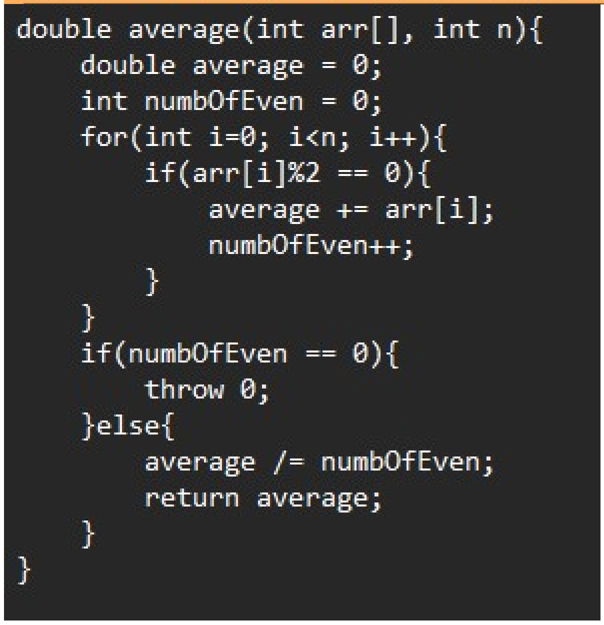
As long as the numbers are between `min` and `max` (inclusively), the function continues to read in values.
If an input value is encountered that is less than min or greater than max, the function throws a `RangeException` with the `min` and `max` values, and the violating (i.e. out of range) input.



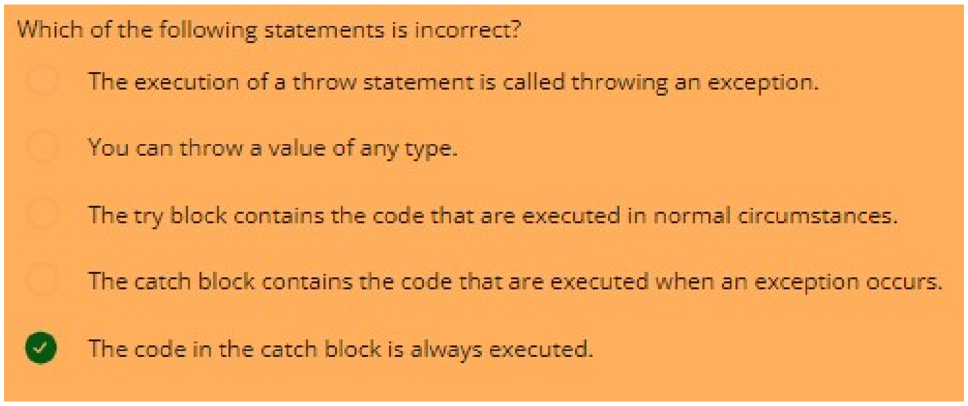
```cpp
void verify(int min, int max)
{
  int value;
  // (cin >> value) reads an input into value and
  // it returns false when the input ends.
  // See the last paragraph in Section 13.2.4.
  while (cin >> value)
  {
    // Verify value
    if(!((value >= min) && (value <= max))){
        throw RangeException(min, max, value);
    }
  }
}
```



Write a function `double average(int arr[], int n)` that returns the average of the even elements of the array. If the array has no even elements, throw 0.

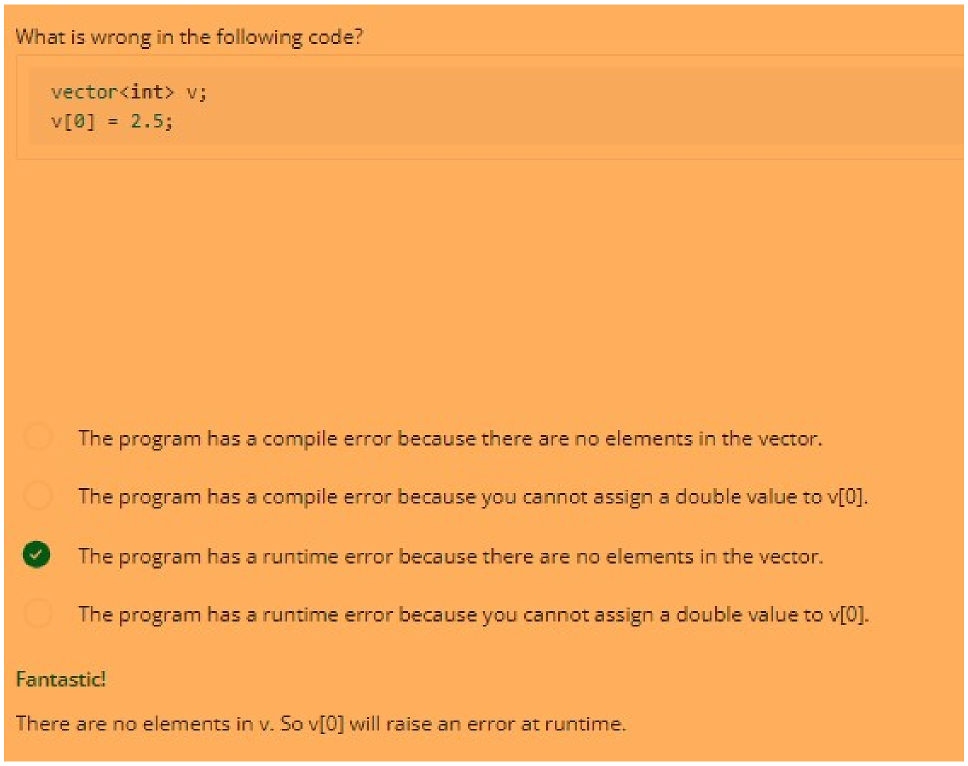


```cpp
double average(int arr[], int n){
    double average = 0;
    int numbOfEven = 0;
    for(int i=0; i<n; i++){
        if(arr[i]%2 == 0){
            average += arr[i];
            numbOfEven++;
        }
    }
    if(numbOfEven == 0){
        throw 0;
    }else{
        average /= numbOfEven;
        return average;
    }
}
```



Which of the following statements is incorrect?

- The execution of a throw statement is called throwing an exception.
- You can throw a value of any type.
- The try block contains the code that are executed in normal circumstances.
- The catch block contains the code that are executed when an exception occurs.
- [✓ The code in the catch block is always executed.]

- 

What is wrong in the following code?

```
vector<int> v;
v[0] = 2.5;
```

The program has a compile error because there are no elements in the vector.

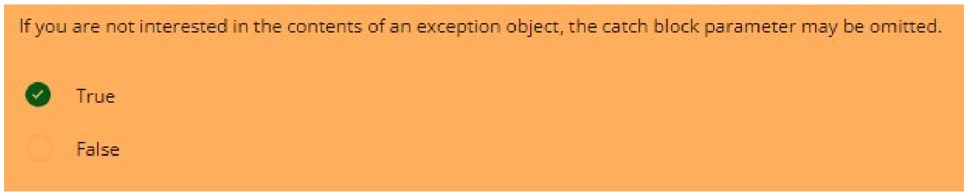
The program has a compile error because you cannot assign a double value to v[0].

[✓ The program has a runtime error because there are no elements in the vector.]

The program has a runtime error because you cannot assign a double value to v[0].

Fantastic!

There are no elements in v. So v[0] will raise an error at runtime.

- 

If you are not interested in the contents of an exception object, the catch block parameter may be omitted.

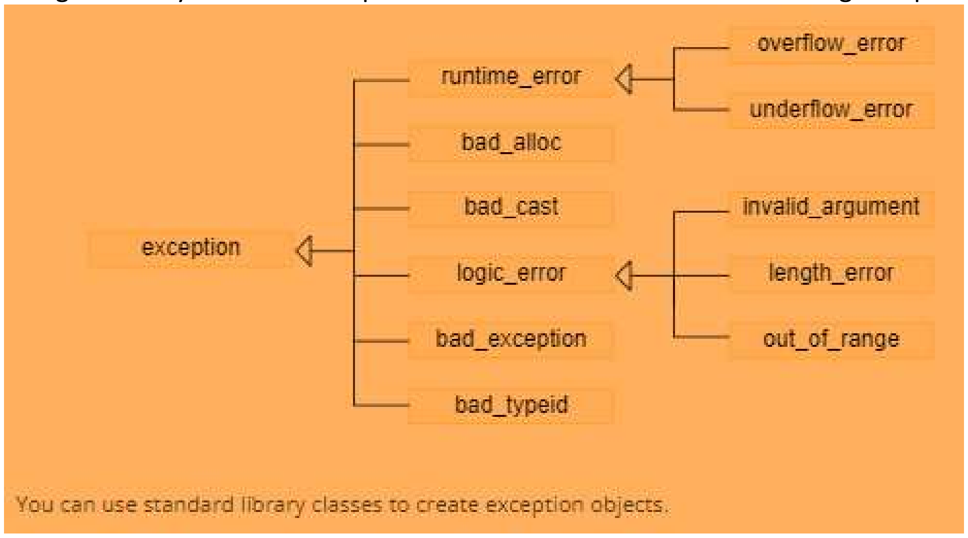
- [✓ True]

False

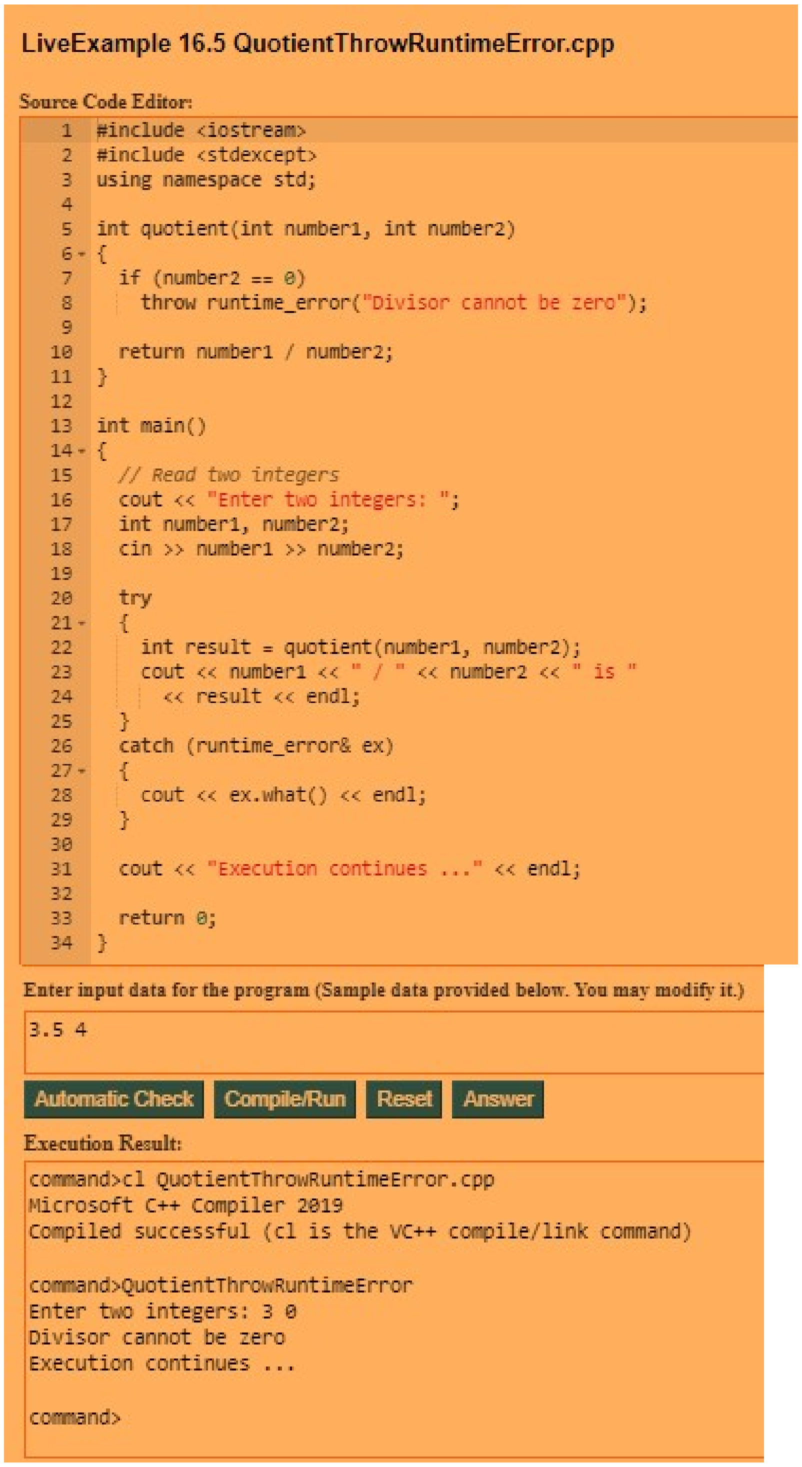
-

16.3 Exception Classes

Monday, April 24, 2023 1:10 PM

- U can use C++ standard exception classes to make exception objects and throw exceptions
- The catch block in b4 ex used int type, but class type more useful usually bc an object can have more info that u can throw to a catch block
- C++ gives many standard exception classes that can be used for making exception objects:
- 

You can use standard library classes to create exception objects.
- Root class in hierarchy is exception (defined in header <exception>), it has virtual fn what() that returns an exception object's error message
  - The runtime_error class (defined in header <stdexcept>) is base class for many standard exception classes that describes runtime errors
    - Class overflow_error describes an arithmetic overflow (like value too large to be stored)
    - Class underflow_error describes math underflow (like value too small to be stored)
  - The logic_error class (defined in header <stdexcept>) is base class for many standard exception classes that describes logic errors
    - Class invalid_argument indicates that an invalid arg has been passed to a function
    - Class length_error indicates that an object's length has exceeded max allowed length
    - Class out_of_range indicates that a val has exceeded allowed range
  - Classes bad_alloc , bad_cast , bad_typeid , and bad_exception describe the exceptions thrown by C++ ops
    - Like a bad_alloc exception thrown by the new operator if the mem cannot be allocated
    - A bad_cast exception thrown by the dynamic_cast op when failed cast to a reference type
    - A bad_typeid exception thrown by the typeid op when the operand for typeid is a NULL pointer
    - The bad_exception class is used in the exception-specification of a fn (discussed later)
- These classes used by some fns in the C++ standard library to throw exceptions
- Can also use these classes to throw exceptions in ur pgrms
  - Modded QuotientWithFunction.cpp to throw a runtime_error



LiveExample 16.5 QuotientThrowRuntimeError.cpp

Source Code Editor:

```
 1  #include <iostream>
 2  #include <stdexcept>
 3  using namespace std;
 4
 5  int quotient(int number1, int number2)
 6  {
 7    if (number2 == 0)
 8      throw runtime_error("Divisor cannot be zero");
 9
10    return number1 / number2;
11  }
12
13  int main()
14  {
15    // Read two integers
16    cout << "Enter two integers: ";
17    int number1, number2;
18    cin >> number1 >> number2;
19
20    try
21    {
22      int result = quotient(number1, number2);
23      cout << number1 << " / " << number2 << " is "
24        << result << endl;
25    }
26    catch (runtime_error& ex)
27    {
28      cout << ex.what() << endl;
29    }
30
31    cout << "Execution continues ..." << endl;
32
33    return 0;
34  }
```

Enter input data for the program (Sample data provided below. You may modify it.)

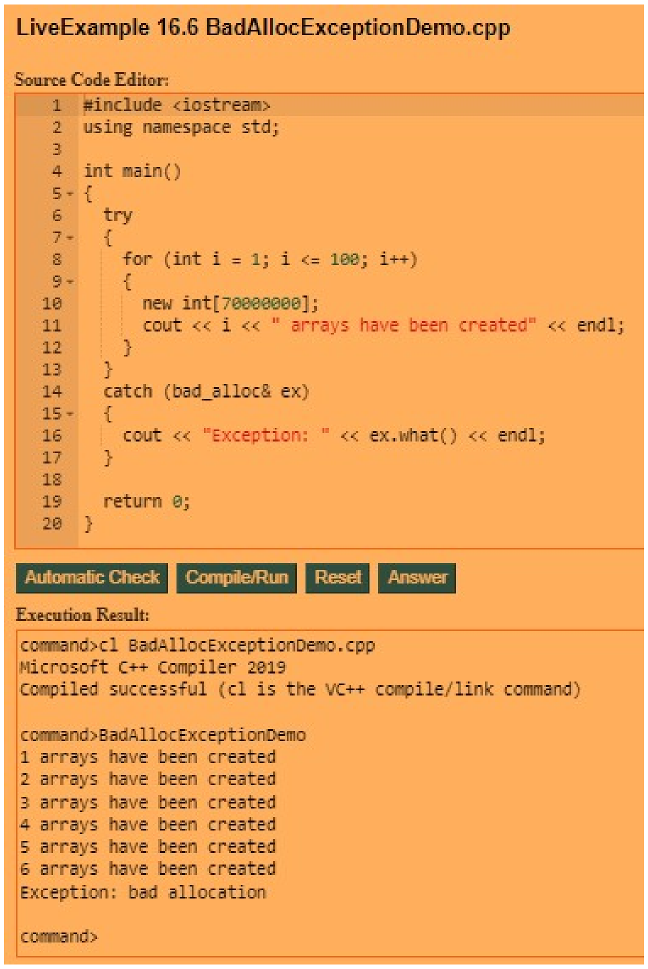
```
3.5 4
```

Execution Result:

```
command>cl QuotientThrowRuntimeError.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>QuotientThrowRuntimeError
Enter two integers: 3 0
Divisor cannot be zero
Execution continues ...

command>
```

- @ ln 8 – the quotient fn in ^ throws an int value, but the fn in this pgrm throws a runtime_error
- Can make a runtime_error object by passing a string that describes the exception
- @ ln 28 – the catch block catches a runtime_error exception and invokes the what fn to return a string description of the exception @ ln 28

• Ex of handling the bad_alloc exception

- 
  LiveExample 16.6 BadAllocExceptionDemo.cpp

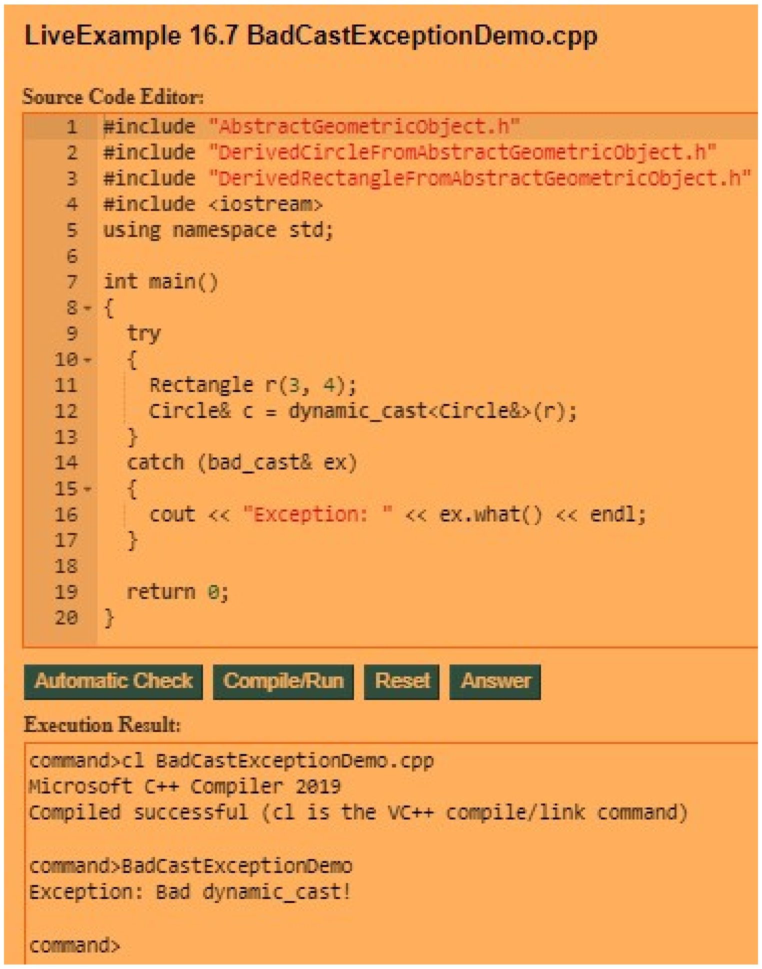
  Source Code Editor:

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

int main()
{
  try
  {
    for (int i = 1; i <= 100; i++)
    {
      new int[70000000];
      cout << i << " arrays have been created" << endl;
    }
  }
  catch (bad_alloc& ex)
  {
    cout << "Exception: " << ex.what() << endl;
  }

  return 0;
}
```

  Execution Result:

```
command>cl BadAllocExceptionDemo.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>BadAllocExceptionDemo
1 arrays have been created
2 arrays have been created
3 arrays have been created
4 arrays have been created
5 arrays have been created
6 arrays have been created
Exception: bad allocation

command>
```

  - Output shows that the pgrm makes 6 arrays b4 fails on the 7th new operator
  - When fails, a bad_alloc exception is thrown and caught in the catch block, which displays the message returned from ex.what()
- Ex showing handling the bad_cast exception

- 
  LiveExample 16.7 BadCastExceptionDemo.cpp

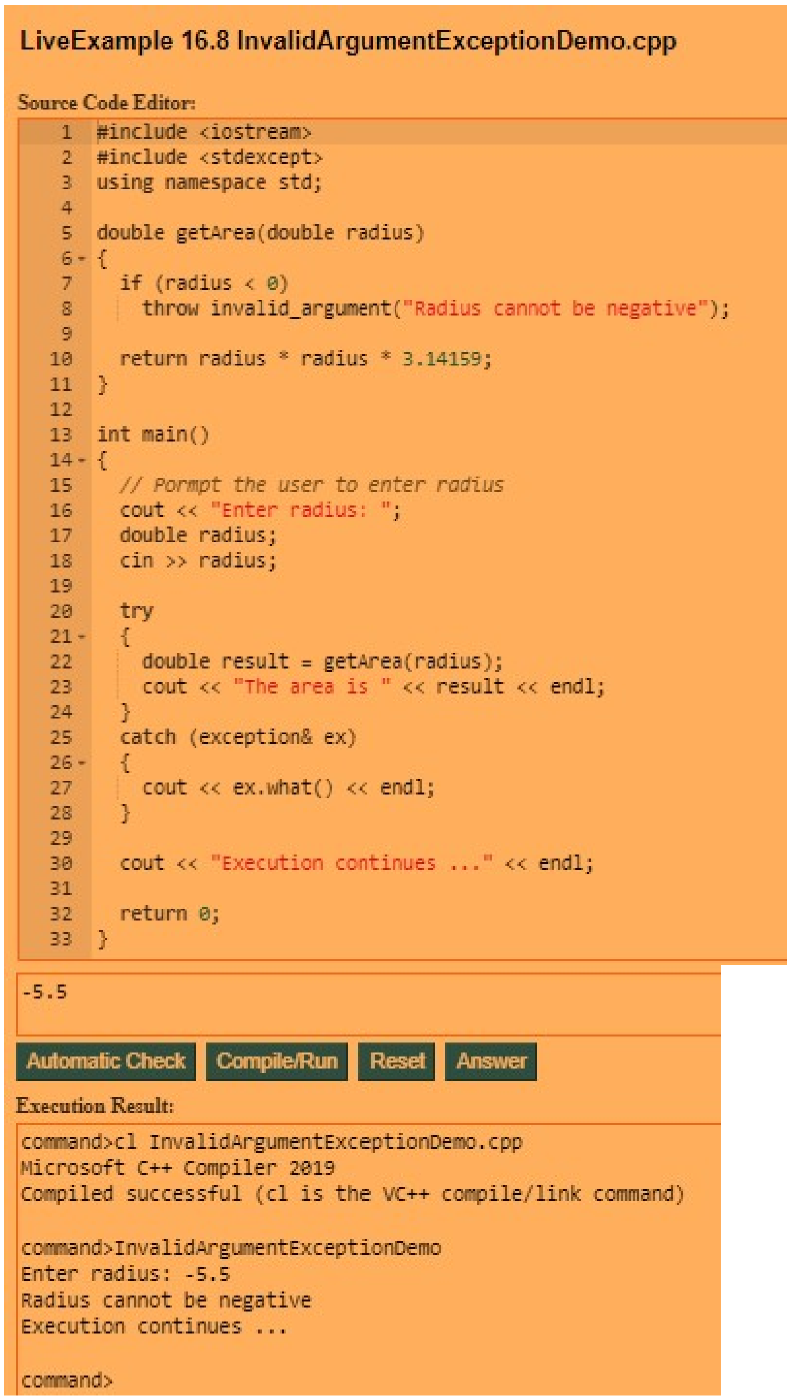
  Source Code Editor:

  ```
   1  #include "AbstractGeometricObject.h"
   2  #include "DerivedCircleFromAbstractGeometricObject.h"
   3  #include "DerivedRectangleFromAbstractGeometricObject.h"
   4  #include <iostream>
   5  using namespace std;
   6
   7  int main()
   8  {
   9    try
  10    {
  11      Rectangle r(3, 4);
  12      Circle& c = dynamic_cast<Circle&>(r);
  13    }
  14    catch (bad_cast& ex)
  15    {
  16      cout << "Exception: " << ex.what() << endl;
  17    }
  18
  19    return 0;
  20  }
  ```

  Execution Result:

  ```
  command>cl BadCastExceptionDemo.cpp
  Microsoft C++ Compiler 2019
  Compiled successful (cl is the VC++ compile/link command)

  command>BadCastExceptionDemo
  Exception: Bad dynamic_cast!

  command>
  ```
- Dynamic casting intro b4, (15.11)
- @ ln 12 – reference of a Rectangle object cast to a Circle reference type, illegal, bad_cast exception thrown
- @ ln 14 – exception caught in the catch block
- @ ln 12 – if this ln replaced by Circle* c = dynamic_cast<Circle*>(&r), no exception would be thrown, c will be a nullptr

• Ex throwing and handling an invalid arg exception

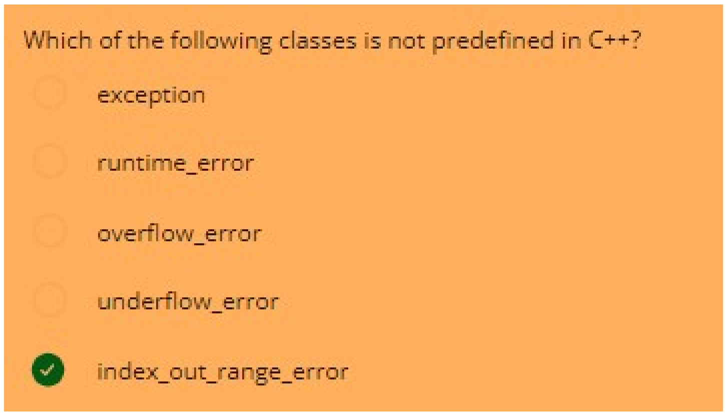


```
 1  #include <iostream>
 2  #include <stdexcept>
 3  using namespace std;
 4
 5  double getArea(double radius)
 6  {
 7    if (radius < 0)
 8      throw invalid_argument("Radius cannot be negative");
 9
10    return radius * radius * 3.14159;
11  }
12
13  int main()
14  {
15    // Pormpt the user to enter radius
16    cout << "Enter radius: ";
17    double radius;
18    cin >> radius;
19
20    try
21    {
22      double result = getArea(radius);
23      cout << "The area is " << result << endl;
24    }
25    catch (exception& ex)
26    {
27      cout << ex.what() << endl;
28    }
29
30    cout << "Execution continues ..." << endl;
31
32    return 0;
33  }
```

```
-5.5
```

```
command>cl InvalidArgumentExceptionDemo.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>InvalidArgumentExceptionDemo
Enter radius: -5.5
Radius cannot be negative
Execution continues ...

command>
```

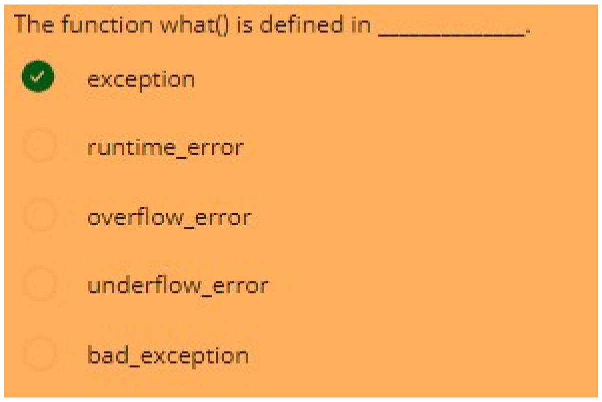
- @ ln 22 – in sample output, the pgrm prompts user to enter radius, 5 and –5, doing getArea(-5) causes invalid_argument exception to be thrown
- @ ln 8 – invalid_argument exception thrown
- @ ln 25 – exception is caught in the catch block
- The catch-block param type exception is a base class for invalid_argument, so it can catch an invalid_argument

- 

  Which of the following classes is not predefined in C++?

  - exception
  - runtime_error
  - overflow_error
  - underflow_error
  - [✓ index_out_range_error]

-

  The function what() is defined in ____________.

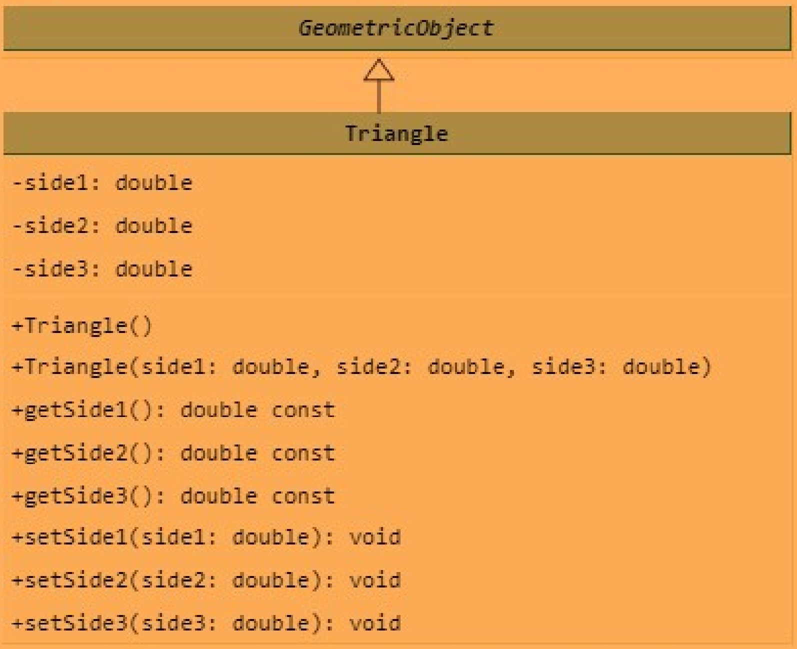
  - [✓ exception]
  - runtime_error
  - overflow_error
  - underflow_error
  - bad_exception

-

16.4 Custom Exception Classes

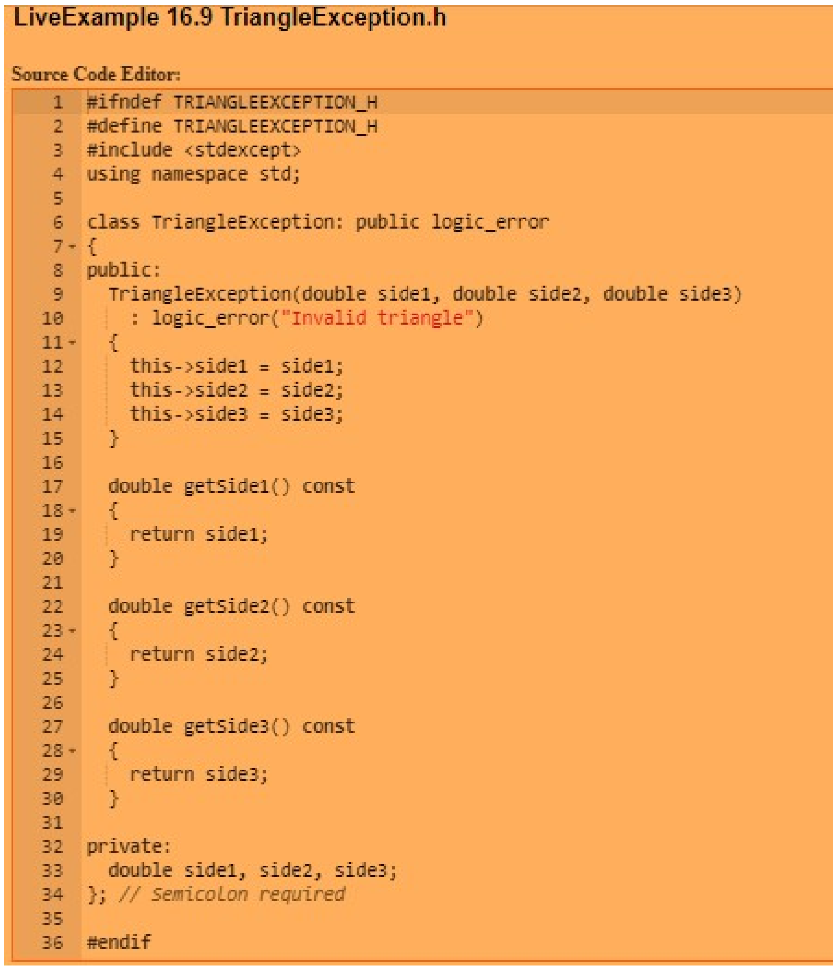
Monday, April 24, 2023 2:56 PM

- U can define custom exception classes to model exceptions that cannot be adequately repped using C++ standard exception classes
- C++ gives exception classes listed b4, use when possible instead of making own
- But, if run into prob that not described enough by them, make own exception class
- Class like any C++ class, but usually desirable to derive it from exception or a derived class of exception so u can utilize the common features (like the what() fn) in the exception class
- Think of Triangle class from modeling triangles
    - Class is derived from the GeometricObject class, which is abstract class intro b4
    - UML for Triangle class is:
    - Figure 16.2


GeometricObject
Triangle
-side1: double
-side2: double
-side3: double
+Triangle()
+Triangle(side1: double, side2: double, side3: double)
+getSide1(): double const
+getSide2(): double const
+getSide3(): double const
+setSide1(side1: double): void
+setSide2(side2: double): void
+setSide3(side3: double): void

The Triangle class models triangles.

    - Triangle is valid if the sum of any 2 sides is greater than 3rd side
    - If try to make a triangle / change a side of a triangle, u need to ensure this property isn't violated, else exception should be thrown
    - Can define the TriangleException class like

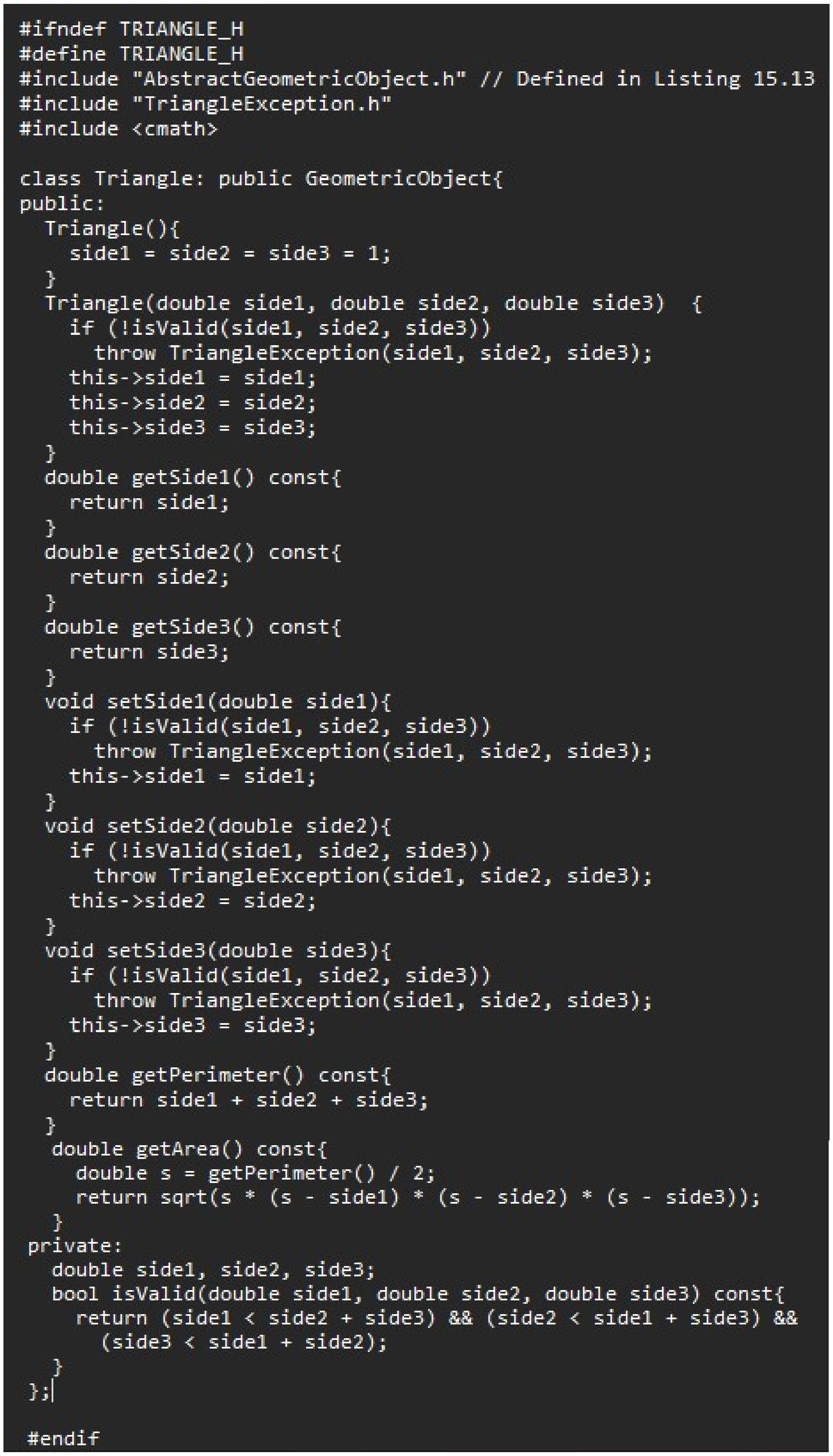
- 

  LiveExample 16.9 TriangleException.h

  Source Code Editor:

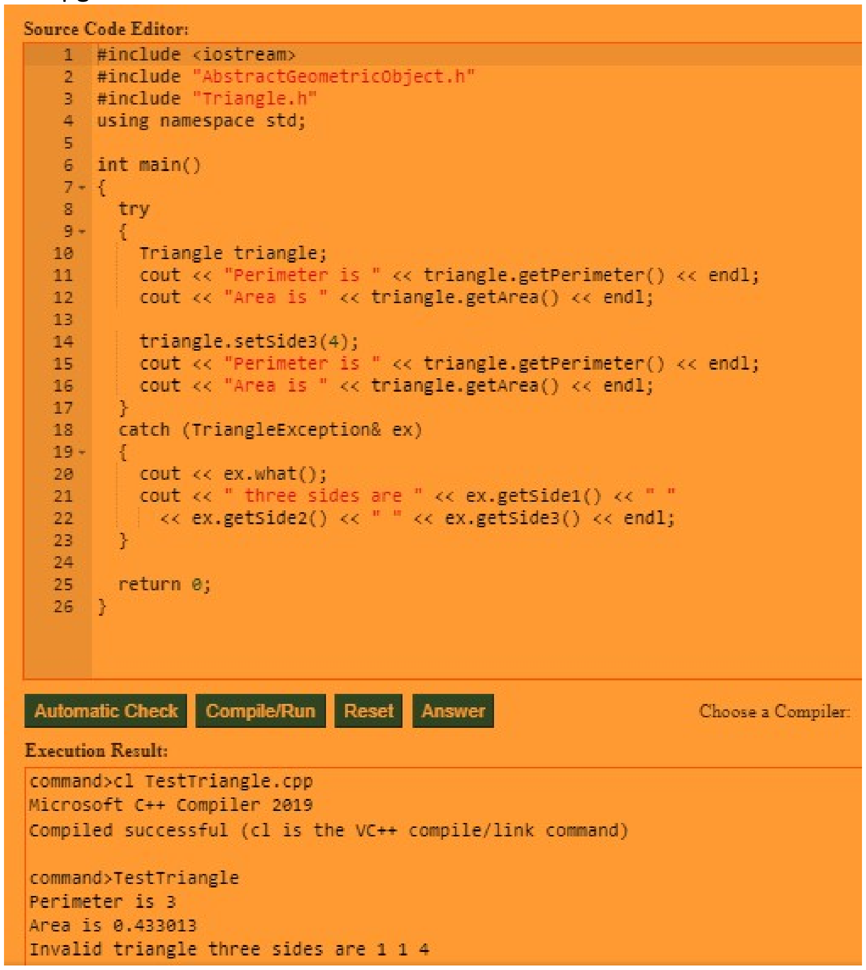
  ```
   1  #ifndef TRIANGLEEXCEPTION_H
   2  #define TRIANGLEEXCEPTION_H
   3  #include <stdexcept>
   4  using namespace std;
   5
   6  class TriangleException: public logic_error
   7  {
   8  public:
   9    TriangleException(double side1, double side2, double side3)
  10      : logic_error("Invalid triangle")
  11    {
  12      this->side1 = side1;
  13      this->side2 = side2;
  14      this->side3 = side3;
  15    }
  16
  17    double getSide1() const
  18    {
  19      return side1;
  20    }
  21
  22    double getSide2() const
  23    {
  24      return side2;
  25    }
  26
  27    double getSide3() const
  28    {
  29      return side3;
  30    }
  31
  32  private:
  33    double side1, side2, side3;
  34  }; // Semicolon required
  35
  36  #endif
  ```
- @ ln 6 – the TriangleException class describes a logic error, so it's appropriate to define this class to extend the standard logic_error
- @ ln 3 – logic_error is in the <stdexcept> header file, so that header file is included
- If base constructor isn't invoked auto, the base class's no-arg constructor is invoked by default
- @ ln 10 – but, since the base class logic_error doesn't have a no-arg constructor, need to invoke a base class's constructor to avoid compile errors
- Invoking logic_error("Invalid triangle") sets an error message, which can be returned from invoking what() on an exception object

* A custom exception class is like reg class, extending from a base class isn't necessary, but its good practice to extend from the standard exception / derived class of exception so custom exception class can use the fns from the standard classes
* The header file TriangleException.h has the implementation for the class, this is the inline implementation, for short fns, using inline imp is efficient
* The Triangle class implemented like



```
#ifndef TRIANGLE_H
#define TRIANGLE_H
#include "AbstractGeometricObject.h" // Defined in Listing 15.13
#include "TriangleException.h"
#include <cmath>

class Triangle: public GeometricObject{
public:
  Triangle(){
    side1 = side2 = side3 = 1;
  }
  Triangle(double side1, double side2, double side3)  {
    if (!isValid(side1, side2, side3))
      throw TriangleException(side1, side2, side3);
    this->side1 = side1;
    this->side2 = side2;
    this->side3 = side3;
  }
  double getSide1() const{
    return side1;
  }
  double getSide2() const{
    return side2;
  }
  double getSide3() const{
    return side3;
  }
  void setSide1(double side1){
    if (!isValid(side1, side2, side3))
      throw TriangleException(side1, side2, side3);
    this->side1 = side1;
  }
  void setSide2(double side2){
    if (!isValid(side1, side2, side3))
      throw TriangleException(side1, side2, side3);
    this->side2 = side2;
  }
  void setSide3(double side3){
    if (!isValid(side1, side2, side3))
      throw TriangleException(side1, side2, side3);
    this->side3 = side3;
  }
  double getPerimeter() const{
    return side1 + side2 + side3;
  }
   double getArea() const{
     double s = getPerimeter() / 2;
     return sqrt(s * (s - side1) * (s - side2) * (s - side3));
   }
 private:
   double side1, side2, side3;
   bool isValid(double side1, double side2, double side3) const{
     return (side1 < side2 + side3) && (side2 < side1 + side3) &&
       (side3 < side1 + side2);
   }
 };

 #endif
```

- @ ln 7 – the Triangle class extends GeometricObject
- @ ln 64-73 – also overrides pure vital fns getPerimeter and getArea defined in the GeometricObject class
- @ ln 78-82 – the isValid fn checks whether a triangle is valid, fn is defined private for use inside the Traingle class
- @ ln 17 – when constructing a Triangle object w/ 3 specific sides, the constructor invokes

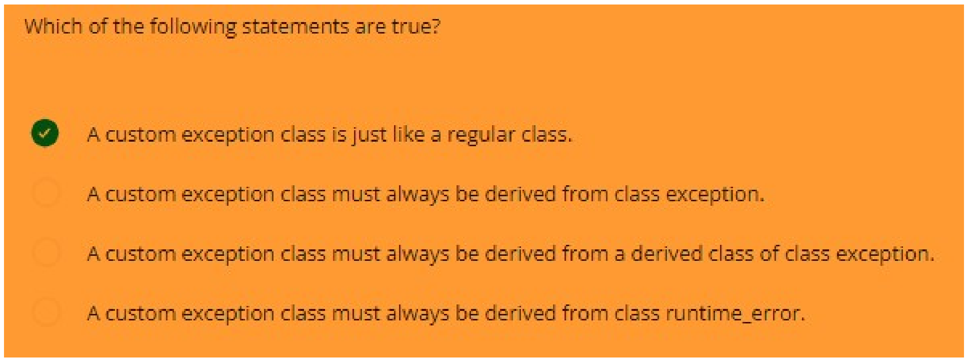
the isValid fn to check validity
    - @ ln 18 - if not valid, TriangleException object made an thrown
- When invoking setSide(side1), isValid(side1, side2, side3) is invoked
    - Where side1 is the new side1 to be set, not the current side1 in the object
- Test pgrm
- 

Source Code Editor:

```
 1  #include <iostream>
 2  #include "AbstractGeometricObject.h"
 3  #include "Triangle.h"
 4  using namespace std;
 5
 6  int main()
 7  {
 8    try
 9    {
10      Triangle triangle;
11      cout << "Perimeter is " << triangle.getPerimeter() << endl;
12      cout << "Area is " << triangle.getArea() << endl;
13
14      triangle.setSide3(4);
15      cout << "Perimeter is " << triangle.getPerimeter() << endl;
16      cout << "Area is " << triangle.getArea() << endl;
17    }
18    catch (TriangleException& ex)
19    {
20      cout << ex.what();
21      cout << " three sides are " << ex.getSide1() << " "
22        << ex.getSide2() << " " << ex.getSide3() << endl;
23    }
24
25    return 0;
26  }
```

Execution Result:

```
command>cl TestTriangle.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>TestTriangle
Perimeter is 3
Area is 0.433013
Invalid triangle three sides are 1 1 4
```

    - @ ln 10 - makes a Triangle object using no-arg constructor
    - @ ln 11-12 – displays its perimeter and area
    - @ ln 14 – changes side3 to 4, causes exception to be thrown
    - @ ln 18-23 – exception caught in catch block
- The what() fn is defined in the exception class
- Since TriangleException is derived from logic_error, which is derived from exception, can invoke what() to display an error message on a TriangleException object
- The TriangleException object has the info for a triangle
- This info useful for handling the exception
-
- 

Which of the following statements are true?

✓ A custom exception class is just like a regular class.

A custom exception class must always be derived from class exception.

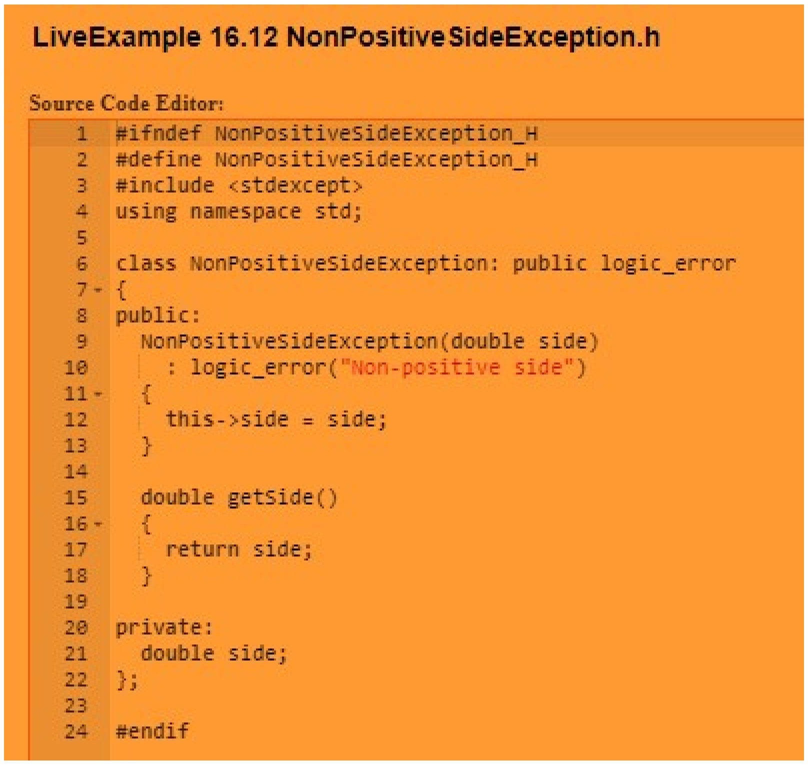
A custom exception class must always be derived from a derived class of class exception.

A custom exception class must always be derived from class runtime_error.

-

16.5 Multiple Catches

Wednesday, April 26, 2023 10:57 PM

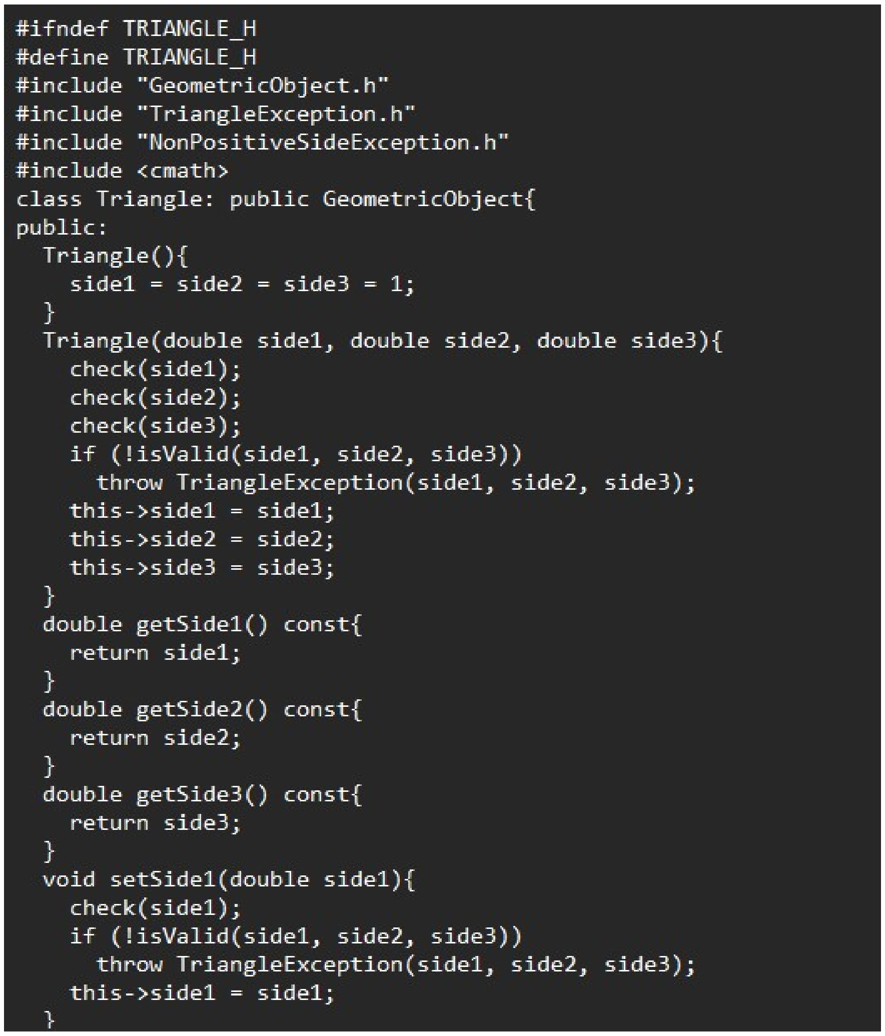
- A try-catch block can have multiple catch clauses to deal w/ diff exceptions thrown in the try clause
- Usually try block should run w/out exceptions, sometimes can throw an exception of 1 type or another
  - A non positive value for a side in a triangle (b4) can be considered a type of exception diff from a TriangleException
- 1 catch block can catch only 1 type of exception
- C++ allows add many catch blocks after a try block to catch multiple types of exceptions
- Fixed it:
- 

  LiveExample 16.12 NonPositiveSideException.h

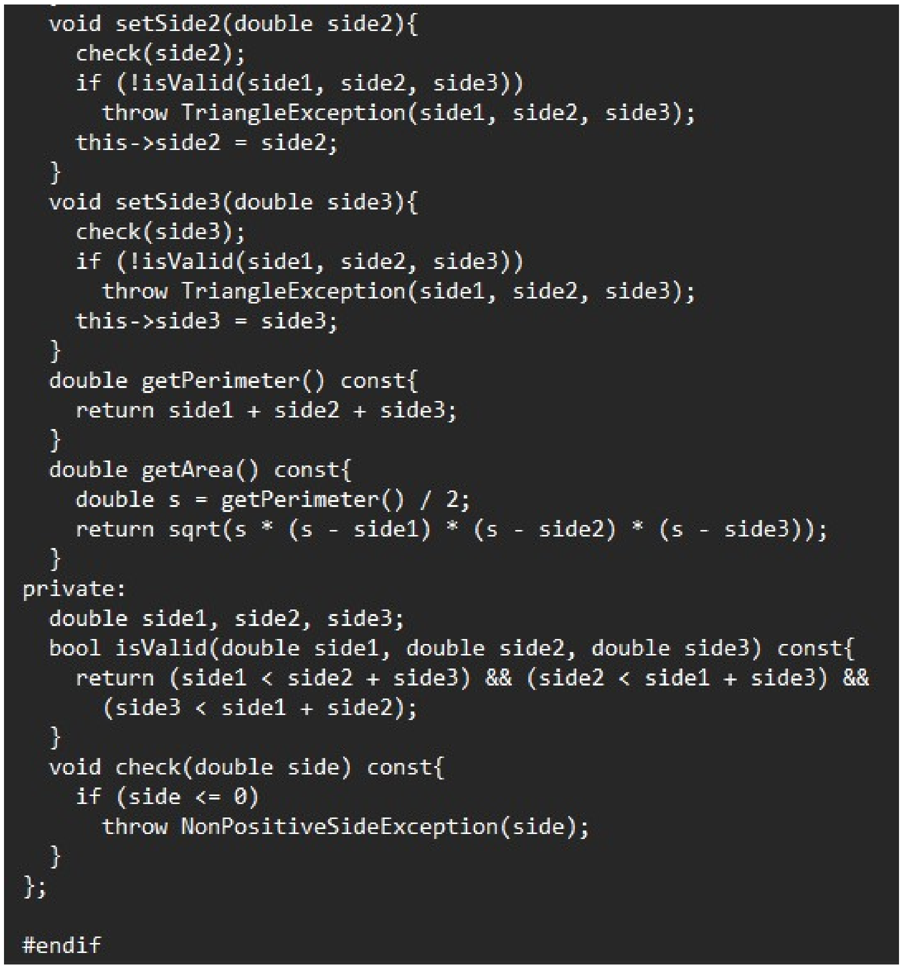
  Source Code Editor:

```
1  #ifndef NonPositiveSideException_H
2  #define NonPositiveSideException_H
3  #include <stdexcept>
4  using namespace std;
5
6  class NonPositiveSideException: public logic_error
7  {
8  public:
9    NonPositiveSideException(double side)
10     : logic_error("Non-positive side")
11   {
12     this->side = side;
13   }
14
15   double getSide()
16   {
17     return side;
18   }
19
20 private:
21   double side;
22 };
23
24 #endif
```

- The NonPositiveSideException class describes a logic error, so it's appropriate to define class to extend the standard logic_error class in ln 6

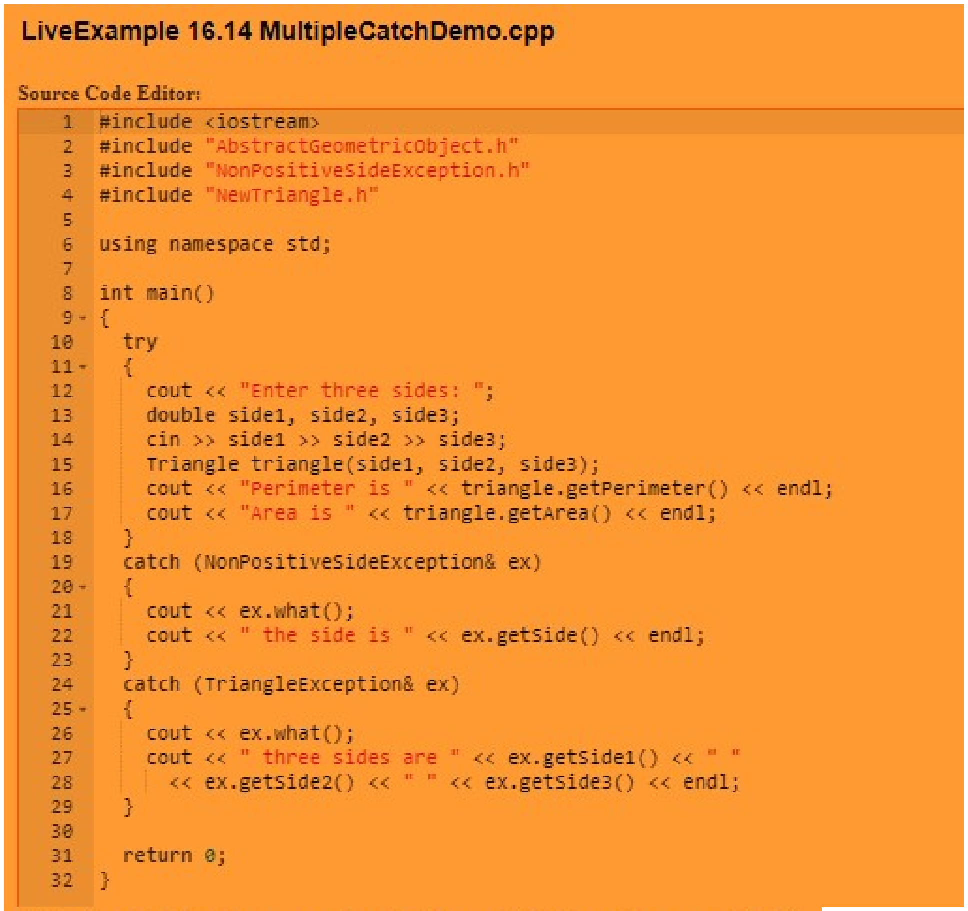
- 

```
#ifndef TRIANGLE_H
#define TRIANGLE_H
#include "GeometricObject.h"
#include "TriangleException.h"
#include "NonPositiveSideException.h"
#include <cmath>
class Triangle: public GeometricObject{
public:
  Triangle(){
    side1 = side2 = side3 = 1;
  }
  Triangle(double side1, double side2, double side3){
    check(side1);
    check(side2);
    check(side3);
    if (!isValid(side1, side2, side3))
      throw TriangleException(side1, side2, side3);
    this->side1 = side1;
    this->side2 = side2;
    this->side3 = side3;
  }
  double getSide1() const{
    return side1;
  }
  double getSide2() const{
    return side2;
  }
  double getSide3() const{
    return side3;
  }
  void setSide1(double side1){
    check(side1);
    if (!isValid(side1, side2, side3))
      throw TriangleException(side1, side2, side3);
    this->side1 = side1;
  }
```

- 

```
  void setSide2(double side2){
    check(side2);
    if (!isValid(side1, side2, side3))
      throw TriangleException(side1, side2, side3);
    this->side2 = side2;
  }
  void setSide3(double side3){
    check(side3);
    if (!isValid(side1, side2, side3))
      throw TriangleException(side1, side2, side3);
    this->side3 = side3;
  }
  double getPerimeter() const{
    return side1 + side2 + side3;
  }
  double getArea() const{
    double s = getPerimeter() / 2;
    return sqrt(s * (s - side1) * (s - side2) * (s - side3));
  }
private:
  double side1, side2, side3;
  bool isValid(double side1, double side2, double side3) const{
    return (side1 < side2 + side3) && (side2 < side1 + side3) &&
      (side3 < side1 + side2);
  }
  void check(double side) const{
    if (side <= 0)
      throw NonPositiveSideException(side);
  }
};

#endif
```

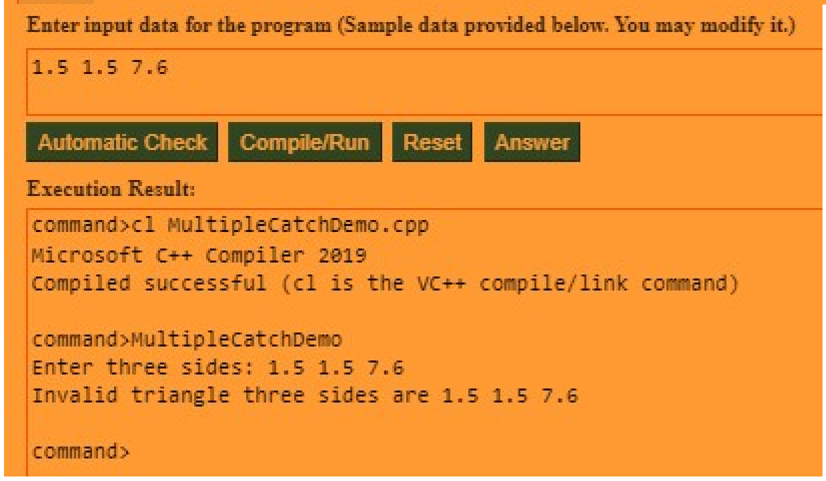
- The new Triangle class is identical to the one b4, but also checks nonpositive sides
  - @ ln 18-20 – when a Triangle object is made, all its sides are checked by invoking the check fn
  - @ ln 94 – the check fn checks if a side is nonpositive
  - @ ln 95 – throws a NonPositiveSideException
- Test pgrm

* 

LiveExample 16.14 MultipleCatchDemo.cpp

Source Code Editor:

```
 1  #include <iostream>
 2  #include "AbstractGeometricObject.h"
 3  #include "NonPositiveSideException.h"
 4  #include "NewTriangle.h"
 5
 6  using namespace std;
 7
 8  int main()
 9  {
10    try
11    {
12      cout << "Enter three sides: ";
13      double side1, side2, side3;
14      cin >> side1 >> side2 >> side3;
15      Triangle triangle(side1, side2, side3);
16      cout << "Perimeter is " << triangle.getPerimeter() << endl;
17      cout << "Area is " << triangle.getArea() << endl;
18    }
19    catch (NonPositiveSideException& ex)
20    {
21      cout << ex.what();
22      cout << " the side is " << ex.getSide() << endl;
23    }
24    catch (TriangleException& ex)
25    {
26      cout << ex.what();
27      cout << " three sides are " << ex.getSide1() << " "
28        << ex.getSide2() << " " << ex.getSide3() << endl;
29    }
30
31    return 0;
32  }
```

* 

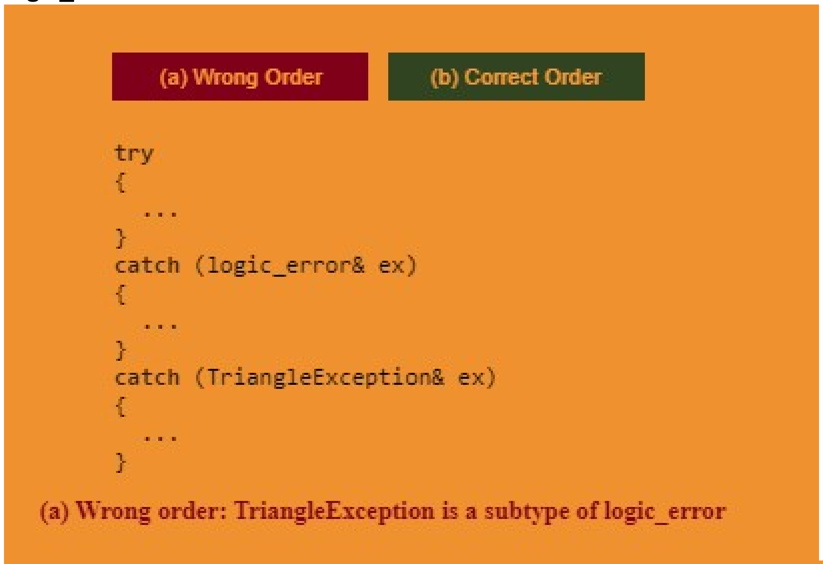
Enter input data for the program (Sample data provided below. You may modify it.)

```
1.5 1.5 7.6
```

Execution Result:

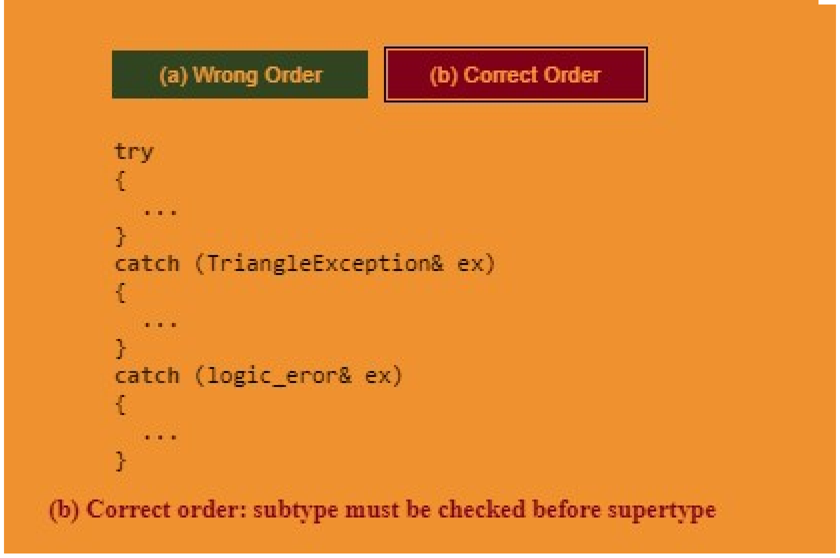
```
command>cl MultipleCatchDemo.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>MultipleCatchDemo
Enter three sides: 1.5 1.5 7.6
Invalid triangle three sides are 1.5 1.5 7.6

command>
```

  - @ ln 12-14 – prompts user to enter 3 sides
  - @ ln 15 – makes a Triangle object
  - @ ln 16-17 – if say 3 sides are 2, 2.5, and 2.5, its legal, pgrm displays the perimeter and area of triangle
  - @ ln 15 – if enter –1, 1, and 1, the constructor throws a NonPositiveSideException
  - @ ln 19 – exception caught
  - @ ln 21-22 – exception processed
  - @ ln 15 – if enter sides as 1, 2, and 1, constructor throws a TriangleException
  - @ ln 24 – this exception is caught by the catch block
  - @ ln 26-28 – exception processed
* Diff exception classes can be derived from a common base class, if a catch block catches exception objects of a base class, it can catch all the exception objects of the derived classes of that base class
* Order which exceptions specified in catch blocks is important

- Catch block for a base class type should appear after a catch block for a derived class type
- Else, exception of a derived class always caught by the catch block for the base class
- Ex: below, a is erroneous bc TriangleException is a derived class of logic_error, correct order in b, in a a TriangleException happened in try block and caught by the catch block for logic_error
- 

```
try
{
  ...
}
catch (logic_error& ex)
{
  ...
}
catch (TriangleException& ex)
{
  ...
}
```

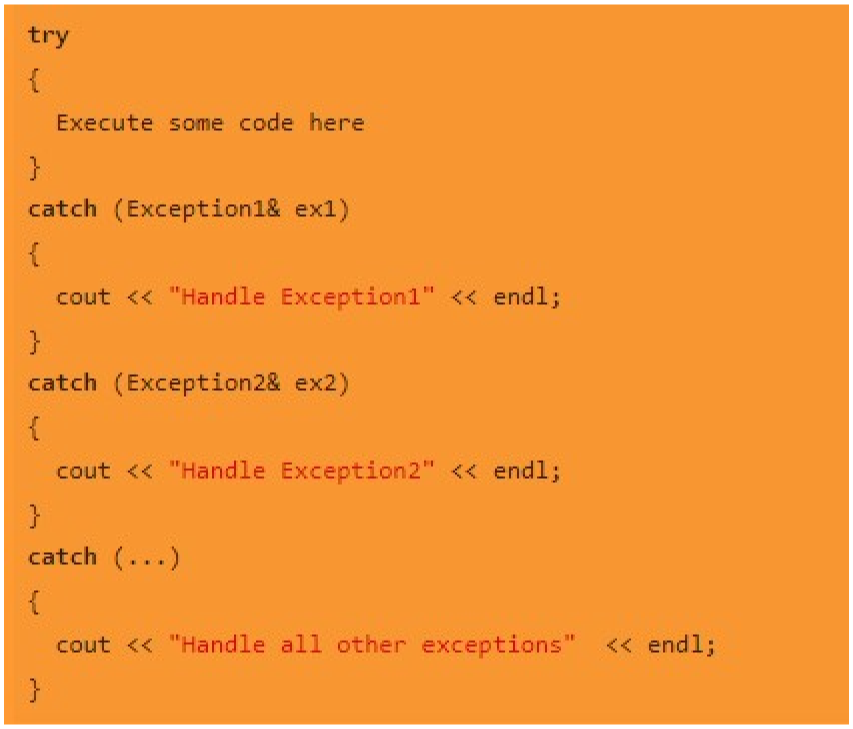
(a) Wrong order: TriangleException is a subtype of logic_error

- 

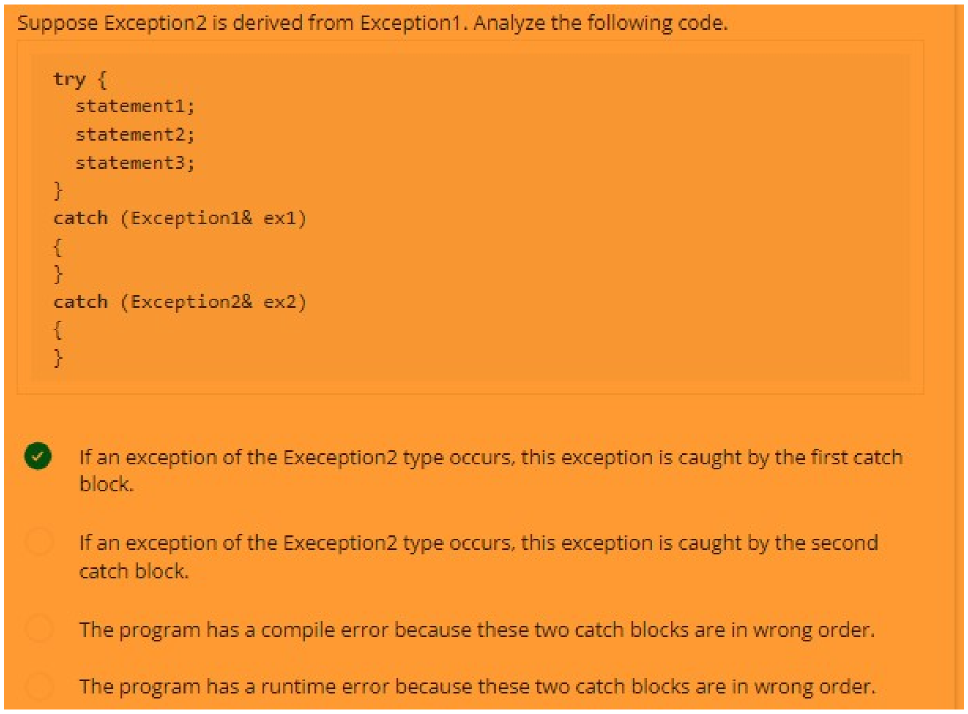
```
try
{
  ...
}
catch (TriangleException& ex)
{
  ...
}
catch (logic_eror& ex)
{
  ...
}
```

(b) Correct order: subtype must be checked before supertype

- Can use ellipsis (…) as param of catch, which will catch any exception no matter what type of exception that was thrown
- Can be used as default handler that catches all exceptions not caught by other handlers if it's specified at last like:

- 

```
try
{
  Execute some code here
}
catch (Exception1& ex1)
{
  cout << "Handle Exception1" << endl;
}
catch (Exception2& ex2)
{
  cout << "Handle Exception2" << endl;
}
catch (...)
{
  cout << "Handle all other exceptions"  << endl;
}
```

-
- 

Suppose Exception2 is derived from Exception1. Analyze the following code.

```
try {
  statement1;
  statement2;
  statement3;
}
catch (Exception1& ex1)
{
}
catch (Exception2& ex2)
{
}
```

[✓ If an exception of the Exeception2 type occurs, this exception is caught by the first catch block.]

If an exception of the Exeception2 type occurs, this exception is caught by the second catch block.

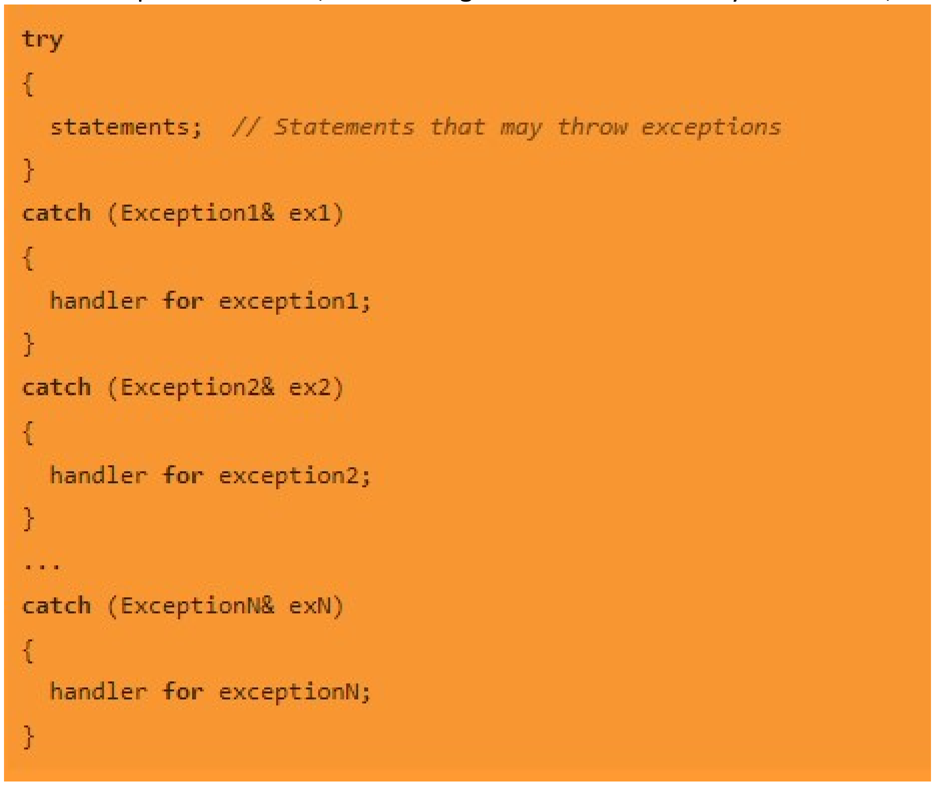
The program has a compile error because these two catch blocks are in wrong order.

The program has a runtime error because these two catch blocks are in wrong order.

-
-

16.6 Exception Propagation

Wednesday, April 26, 2023 11:23 PM

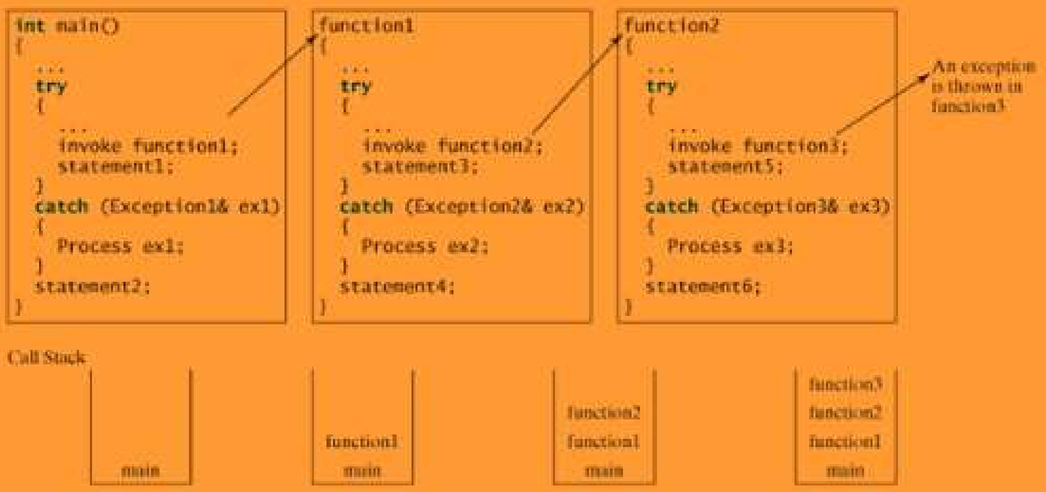
- An exception is thrown thru a chain of calling fns until its caught / reaches to the main fn
- Know how to declare an exception and how to throw an exception
- When exception is thrown, can be caught and handled in a try-catch block, like
- 

```
try
{
  statements;  // Statements that may throw exceptions
}
catch (Exception1& ex1)
{
  handler for exception1;
}
catch (Exception2& ex2)
{
  handler for exception2;
}
...
catch (ExceptionN& exN)
{
  handler for exceptionN;
}
```

- If no exceptions during execution of try block, catch blocks are skipped
- If one of the statements inside try block throws an exception, C++ skips remaining code in try block, starts process of finding the code to handle the exception, aka the exception handler
- Exception handler found by propagating the exception backward thru a chain of fn calls, starting from current fn
- Each catch block is examined, first to last, to see if type of exception object is instance of exception class in the catch block
- If yes, exception object assigned to var declared, code in catch block executed
- If no handler found, C++ exits this fn, passes exception to fn that invoked the fn, continues same process to find handler
- If no handler found in chain of fns being invoked, pgrm prints error message on console and terminates, aka catching an exception (process of finding a handler)
- Ex:
  - If main fn invokes fn1, fn1 invokes fn2, fn2 invokes fn3, and fn3 throws an exception
  - If the exception type is Exception3, it is caught by the catch block for handling exception ex3 in fn2, statement5 is skipped, and statement6 executed
  - If exception type is Exception2, fn 2 is aborted, ctrl returned to fn1, and the exception is caught by the catch block for handling exception ex2 in fn1, statement3 is skipped and statement4 is executed
  - If exception type is Exception1, fn1 is aborted, the ctrl returned to the main fn, and the exception is caught by the catch block for handling exception ex1 in the main fn, statement1 is skipped and statement2 is executed
  - If exception not caught in fn2, fn1, and main, the pgrm terminates, statement1 and

statement2 aren't executed

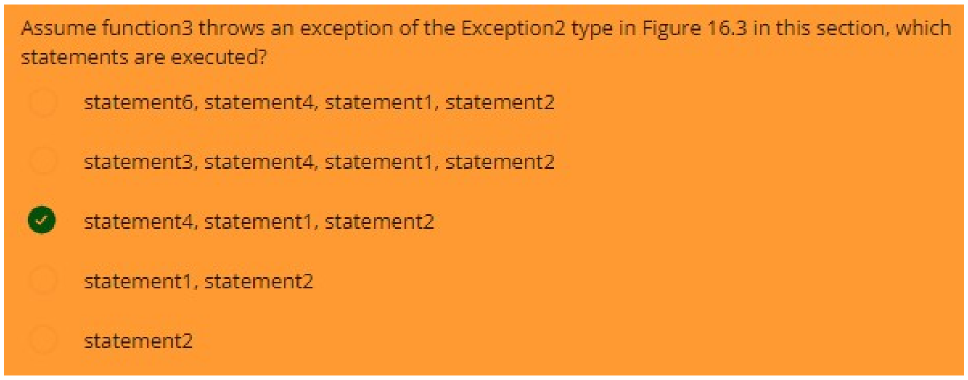
- **Figure 16.3**

  ```
  int main()
  {
    ...
    try
    {
      ...
      invoke function1;
      statement1;
    }
    catch (Exception1& ex1)
    {
      Process ex1;
    }
    statement2;
  }
  ```

  ```
  function1
  {
    ...
    try
    {
      ...
      invoke function2;
      statement3;
    }
    catch (Exception2& ex2)
    {
      Process ex2;
    }
    statement4;
  }
  ```

  ```
  function2
  {
    ...
    try
    {
      ...
      invoke function3;
      statement5;
    }
    catch (Exception3& ex3)
    {
      Process ex3;
    }
    statement6;
  }
  ```

  If an exception is not caught in the current function, it is passed to its caller. The process is repeated until the exception is caught or passed to the `main` function.
-
- 

  Assume function3 throws an exception of the Exception2 type in Figure 16.3 in this section, which statements are executed?

  statement6, statement4, statement1, statement2

  statement3, statement4, statement1, statement2

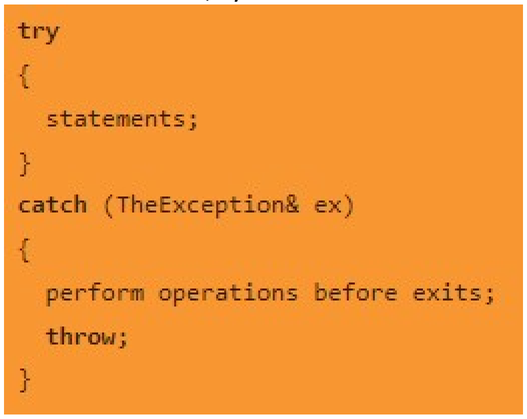
  [✓ statement4, statement1, statement2]

  statement1, statement2

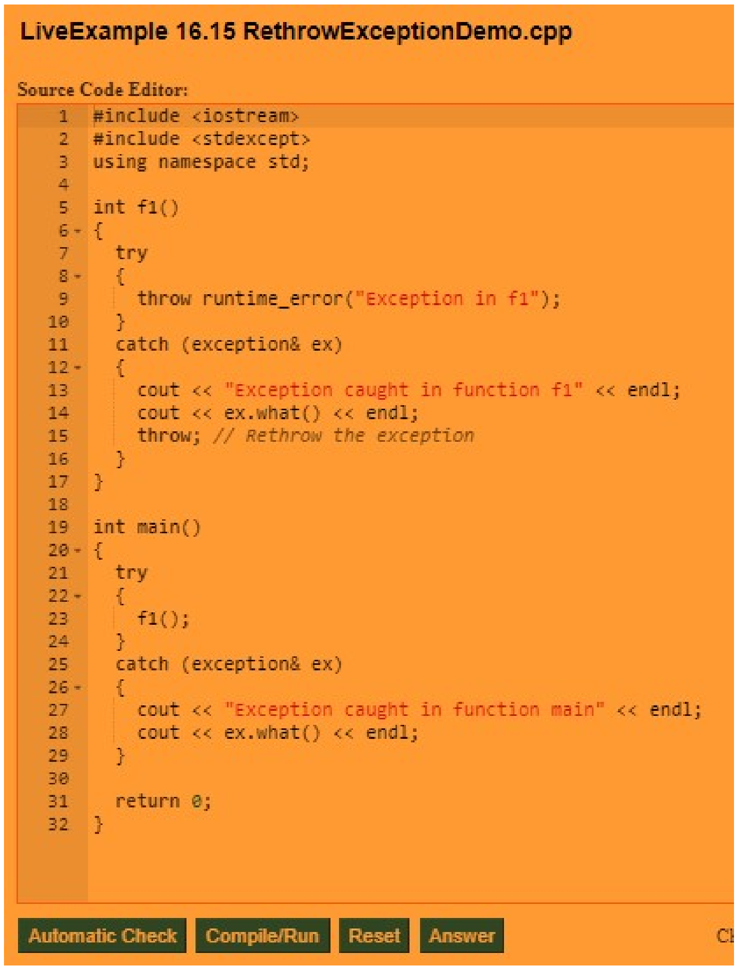
  statement2
-
-

16.7 Rethrowing Exceptions

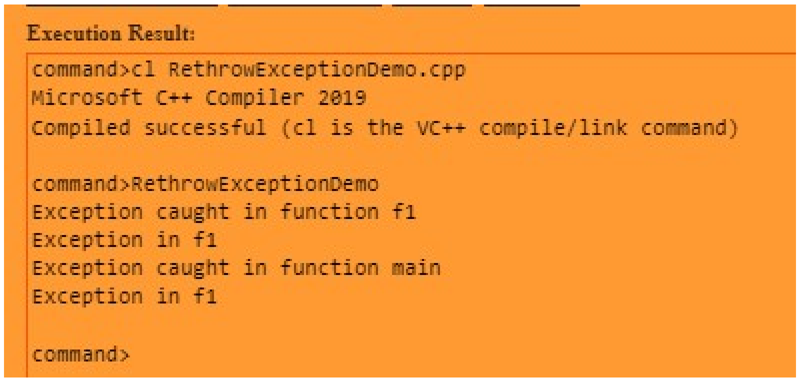
Thursday, April 27, 2023 11:41 PM

- After an exception is caught, it can be rethrown to the caller of the fn
- C++ allows an exception handler to rethrow the exception if it cannot process it / wants to let its caller be notified, syntax:
- 

```
try
{
  statements;
}
catch (TheException& ex)
{
  perform operations before exits;
  throw;
}
```

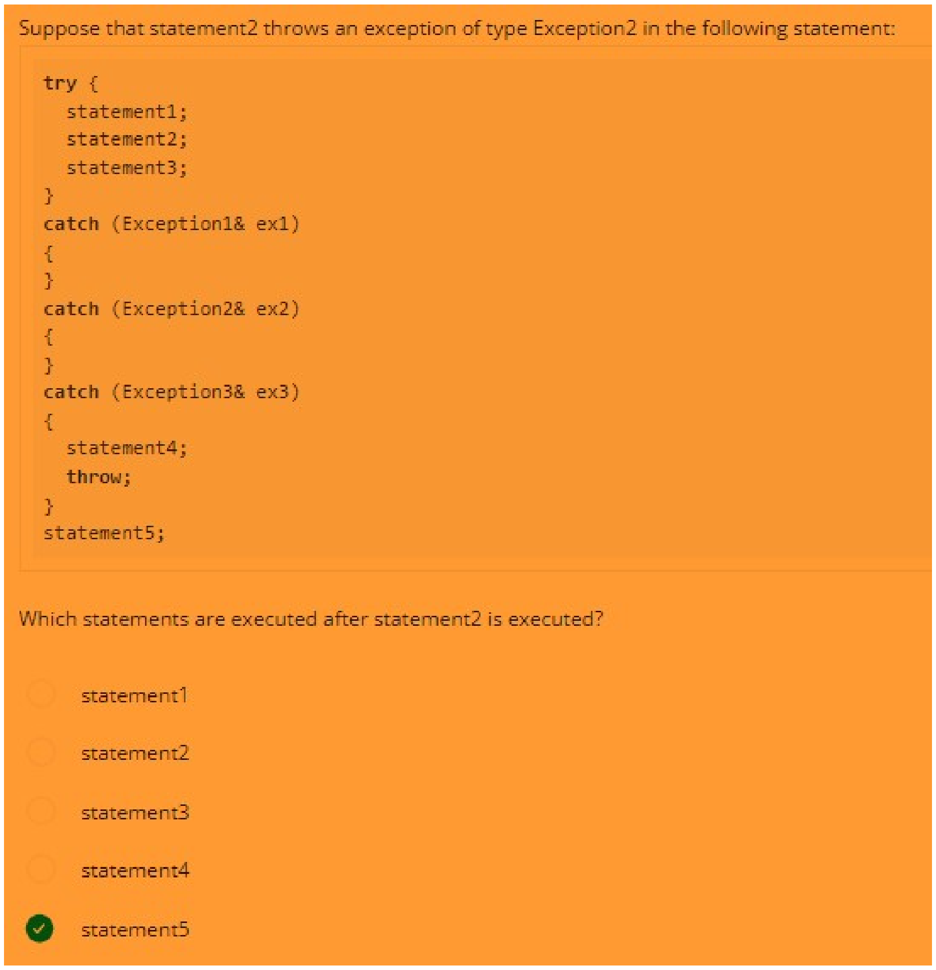
- Statement throw rethrows the exception so that other handlers get a chance to process it
- Ex of rethrow
- 

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

int f1()
{
  try
  {
    throw runtime_error("Exception in f1");
  }
  catch (exception& ex)
  {
    cout << "Exception caught in function f1" << endl;
    cout << ex.what() << endl;
    throw; // Rethrow the exception
  }
}

int main()
{
  try
  {
    f1();
  }
  catch (exception& ex)
  {
    cout << "Exception caught in function main" << endl;
    cout << ex.what() << endl;
  }

  return 0;
}
```

- 

Execution Result:

```
command>cl RethrowExceptionDemo.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>RethrowExceptionDemo
Exception caught in function f1
Exception in f1
Exception caught in function main
Exception in f1

command>
```

  - @ ln 23 – pgrm invokes fn f1
  - @ ln 9 – throws an exception
  - @ ln 11 – exception caught in catch block
  - @ ln 15 – the exception is rethrown to main fn here, catch block in main fn catches rethrown exception
  - @ ln 27-28 – exception processed here
-
- 

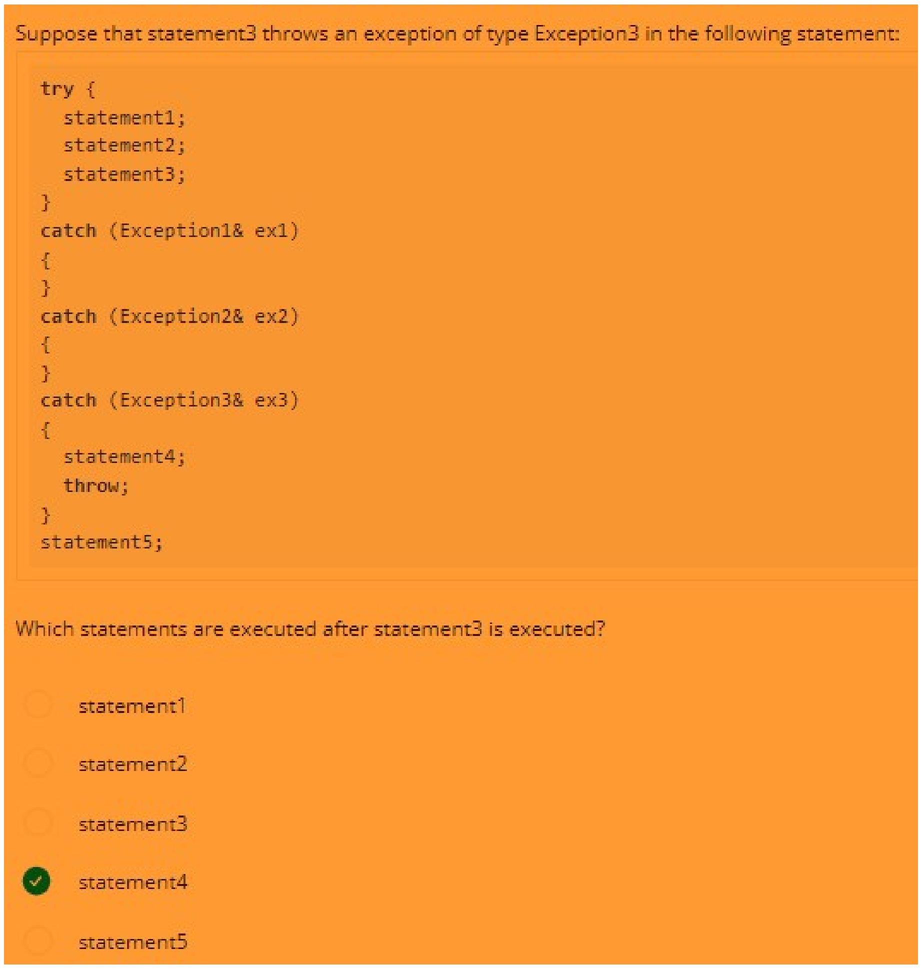
Suppose that statement2 throws an exception of type Exception2 in the following statement:

```
try {
  statement1;
  statement2;
  statement3;
}
catch (Exception1& ex1)
{
}
catch (Exception2& ex2)
{
}
catch (Exception3& ex3)
{
  statement4;
  throw;
}
statement5;
```

Which statements are executed after statement2 is executed?

○ statement1
○ statement2
○ statement3
○ statement4
✓ statement5

-



- Suppose that statement3 throws an exception of type Exception3 in the following statement:

  ```
  try {
    statement1;
    statement2;
    statement3;
  }
  catch (Exception1& ex1)
  {
  }
  catch (Exception2& ex2)
  {
  }
  catch (Exception3& ex3)
  {
    statement4;
    throw;
  }
  statement5;
  ```

  Which statements are executed after statement3 is executed?

  statement1

  statement2

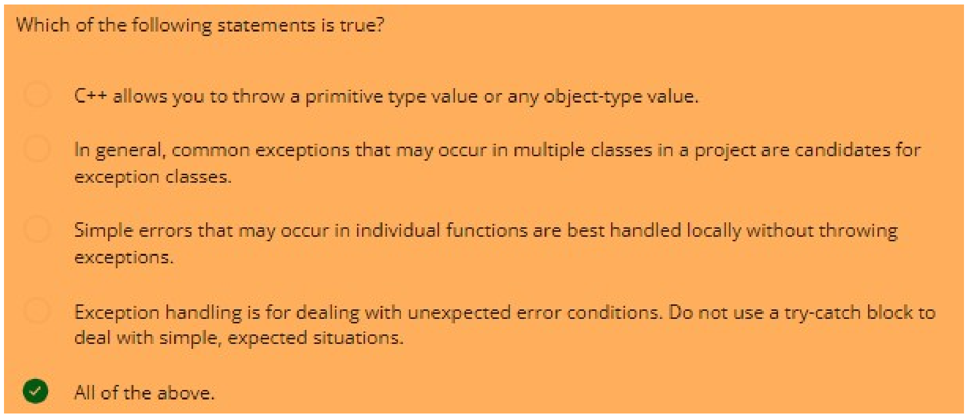
  statement3

  [✓ statement4]

  statement5
-

16.8 When to Use Exceptions

Friday, April 28, 2023 9:55 AM

- Use exceptions for exceptional circumstances, not for simple logic errors that can be caught easily using an if statement
- The try block has code executed in normal circumstances, catch block has code executed in exceptional circumstances
- Exception handling separates error=handling code from normal pgrming, so pgrm easier to read and mod
- But exception handling usually req more time and resources bc need instantiating a new exception object, rolling back call stack, and propagating the exception thru the chain fns invoked to search for handler
- Exception happens in a fn, if want exception to be processed by caller, throw it, if handle exception in fn where it happens, no need to throw / use exceptions
- Generally, common exceptions that happen in many classes in a project are candidates for exception classes
- Simple errors that can happen in single fns best handled locally w/out exception throwing
- Exception handling is for dealing w/ unexpected error condition, don't use try-catch block to deal w/ simple expected situations
- Sometimes difficult to decide, but don’t abuse exception handling for simple logic tests
-
- 

  Which of the following statements is true?

  C++ allows you to throw a primitive type value or any object-type value.

  In general, common exceptions that may occur in multiple classes in a project are candidates for exception classes.

  Simple errors that may occur in individual functions are best handled locally without throwing exceptions.

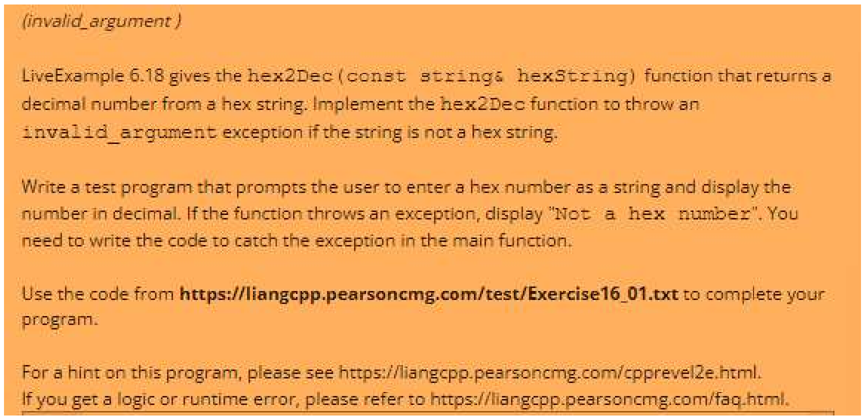
  Exception handling is for dealing with unexpected error conditions. Do not use a try-catch block to deal with simple, expected situations.

  [✓ All of the above.]
-

Ch 16 Summary

Friday, April 28, 2023 10:03 AM

1. Exception handling makes programs robust. Exception handling separates error-handling code from normal programming tasks, thus making programs easier to read and modify. Another important advantage of exception handling is that it enables a function to throw an exception to its caller.
2. C++ allows you to use the throw statement to throw a value of any type (primitive or class type) when an exception occurs. This value is passed to a catch block as an argument so that the catch block can utilize this value to process the exception.
3. When an exception is thrown, the normal execution flow is interrupted. If the exception value matches a catch block parameter type, the control is transferred to a catch block. Otherwise, the function is exited and the exception is thrown to the function's caller. If the exception is not handled in the main function, the program is aborted.
4. C++ provides a number of standard exception classes that can be used for creating exception objects. You can use the exception class or its derived classes runtime_error and logic_error to create exception objects.
5. You also can create a custom exception class if the standard exception classes cannot adequately describe exceptions. This class is just like any C++ class, but often it is desirable to derive it from exception or a derived class of exception so you can utilize the common features (e.g., the what() function) in the exception class.
6. A try block may be followed by multiple catch blocks. The order in which exceptions are specified in catch blocks is important. A catch block for a base class type should appear after a catch block for a derived class type.
7. If a function throws an exception, you should declare the exception's type in the function header to warn programmers to deal with potential exceptions.
8. Exception handling should not be used to replace simple tests. You should test simple exceptions whenever possible and reserve exception handling for dealing with situations that cannot be handled with if statements.



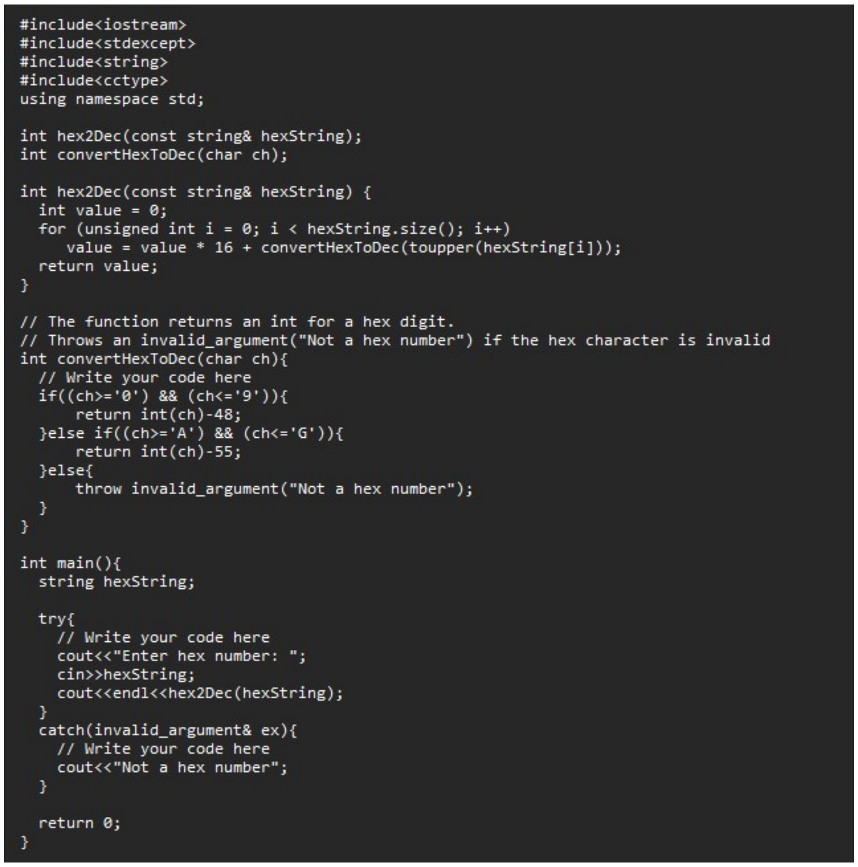
(invalid_argument)

LiveExample 6.18 gives the `hex2Dec(const string& hexString)` function that returns a decimal number from a hex string. Implement the `hex2Dec` function to throw an `invalid_argument` exception if the string is not a hex string.

Write a test program that prompts the user to enter a hex number as a string and display the number in decimal. If the function throws an exception, display "`Not a hex number`". You need to write the code to catch the exception in the main function.

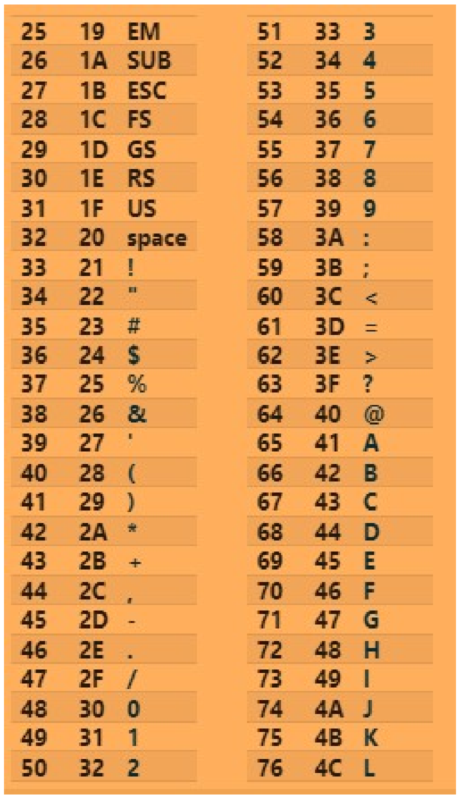
Use the code from **https://liangcpp.pearsoncmg.com/test/Exercise16_01.txt** to complete your program.

For a hint on this program, please see https://liangcpp.pearsoncmg.com/cpprevel2e.html.
If you get a logic or runtime error, please refer to https://liangcpp.pearsoncmg.com/faq.html.

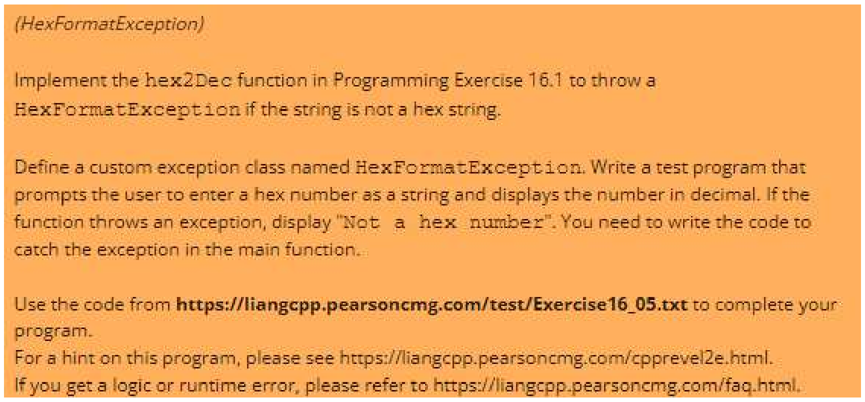


```
#include<iostream>
#include<stdexcept>
#include<string>
#include<cctype>
using namespace std;

int hex2Dec(const string& hexString);
int convertHexToDec(char ch);

int hex2Dec(const string& hexString) {
  int value = 0;
  for (unsigned int i = 0; i < hexString.size(); i++)
     value = value * 16 + convertHexToDec(toupper(hexString[i]));
  return value;
}

// The function returns an int for a hex digit.
// Throws an invalid_argument("Not a hex number") if the hex character is invalid
int convertHexToDec(char ch){
  // Write your code here
  if((ch>='0') && (ch<='9')){
      return int(ch)-48;
  }else if((ch>='A') && (ch<='G')){
      return int(ch)-55;
  }else{
      throw invalid_argument("Not a hex number");
  }
}

int main(){
  string hexString;

  try{
    // Write your code here
    cout<<"Enter hex number: ";
    cin>>hexString;
    cout<<endl<<hex2Dec(hexString);
  }
  catch(invalid_argument& ex){
    // Write your code here
    cout<<"Not a hex number";
  }

  return 0;
}
```

```
#include<iostream>
#include<stdexcept>
#include<string>
#include<cctype>
using namespace std;

int hex2Dec(const string& hexString);
int convertHexToDec(char ch);

int hex2Dec(const string& hexString)
{
  int value = 0;
  for (unsigned int i = 0; i < hexString.size(); i++)
     value = value * 16 + convertHexToDec(toupper(hexString[i]));
  return value;
}

// The function returns an int for a hex digit.
// Throws an invalid_argument("Not a hex number") if the hex character is invalid
int convertHexToDec(char ch)
{
  // Write your code here
  if((ch>='0') && (ch<='9')){
      return int(ch)-48;
  // Write your code here
  if((ch>='0') && (ch<='9')){
      return int(ch)-48;
  }else if((ch>='A') && (ch<='G')){
      return int(ch)-55;
  }else{
      throw invalid_argument("Not a hex number");
  }
}

int main()
{
  string hexString;

  try
  {
    // Write your code here
    cout<<"Enter hex number: ";
    cin>>hexString;
    cout<<endl<<hex2Dec(hexString);
  }
  catch(invalid_argument& ex)
  {
    // Write your code here
    cout<<"Not a hex number";
  }

  return 0;
}
```



25	19	EM	51	33	3
26	1A	SUB	52	34	4
27	1B	ESC	53	35	5
28	1C	FS	54	36	6
29	1D	GS	55	37	7
30	1E	RS	56	38	8
31	1F	US	57	39	9
32	20	space	58	3A	:
33	21	!	59	3B	;
34	22	"	60	3C	<
35	23	#	61	3D	=
36	24	$	62	3E	>
37	25	%	63	3F	?
38	26	&	64	40	@
39	27	'	65	41	A
40	28	(	66	42	B
41	29	)	67	43	C
42	2A	*	68	44	D
43	2B	+	69	45	E
44	2C	,	70	46	F
45	2D	-	71	47	G
46	2E	.	72	48	H
47	2F	/	73	49	I
48	30	0	74	4A	J
49	31	1	75	4B	K
50	32	2	76	4C	L



(HexFormatException)

Implement the `hex2Dec` function in Programming Exercise 16.1 to throw a `HexFormatException` if the string is not a hex string.

Define a custom exception class named `HexFormatException`. Write a test program that prompts the user to enter a hex number as a string and displays the number in decimal. If the function throws an exception, display "`Not a hex number`". You need to write the code to catch the exception in the main function.

Use the code from **https://liangcpp.pearsoncmg.com/test/Exercise16_05.txt** to complete your program.
For a hint on this program, please see https://liangcpp.pearsoncmg.com/cpprevel2e.html.
If you get a logic or runtime error, please refer to https://liangcpp.pearsoncmg.com/faq.html.

```
#include <iostream>
#include <string>
#include <stdexcept>
#include <typeinfo>
using namespace std;

class HexFormatException : public runtime_error
{
public:
  HexFormatException(char ch) : runtime_error("Hex number format error")
  {
    // Write your code
    cout<<"Not a hex number";
  }

  char getCh()
  {
    // Write your code
    return ch;
  }

private:
  char ch;
};

// The function returns an int for a hex digit.
// Throws HexFormatException(ch) if the hex character is invalid
int convertHexToDec(char ch)
{
  // Write your code
  if((ch>='0') && (ch<='9')){
      return int(ch)-int('0');
  }else if((ch>='A') && (ch<='G')){
      return int(ch)-int('A') + 10;
  }else{
      throw HexFormatException(ch);
  }
}

int hex2Dec(const string& hexString)
```

```
  }
}

int hex2Dec(const string& hexString)
{
  int value = convertHexToDec(hexString[0]);
  for (int i = 1; i < hexString.size(); i++)
  {
    value = value * 16 + convertHexToDec(hexString[i]);
  }

  return value;
}
```

```cpp
int main()
{
  string hexString;

  try
  {
    // Write your code
    cout<<"Enter hex number: ";
    cin>>hexString;
    cout<<endl<<hex2Dec(hexString);
  }
  catch (runtime_error& ex)
  {
    // Write your code
    //HexFormatException(ch);
  }

  return 0;
}
```