CHAPTER 14
EXCEPTIONS

CMPS 148

Today's Topics

- Exceptions
  - Why would you use them?
  - How do you use them?
  - Base Exception classes
  - Object-Oriented Exceptions

Exceptions

- Exceptions are occurrences in a program deemed "exceptional" - in that they indicate a serious error within the program
  - Often exceptions occur as a result of bad user input - such as divide by zero.
  - Exceptions can be "impossible circumstances" - such as radius < 0
- Main goal of Exceptions in C++ is to allow you to **handle errors** at different places in the code then where you **detect the error**…

Simple Example

```
int main() {
  int n1, n2;
  cout << "Please enter two numbers:  ";
  cin >> n1 >> n2;
  cout << "Quotient:  " << n1 / n2 << endl;
}
```

```
Please enter two numbers:  4 0
Floating point exception
```

- If N2 is zero, the resulting division is not applicable (its infinity)…
- We could of course check for this…

Simple Exception

```
int main() {
  int n1, n2;
  cout << "Please enter two numbers:  ";
  cin >> n1 >> n2;
  if ( n2 != 0 ) {
     cout << "Quotient:  " << n1 / n2 << endl;
  }
  else {
     cout << "Divisor cannot be zero" << endl;
  }
}
```

In this circumstance, we can handle the error using a simple if statement to check for the “exception”

C++ Exceptions

```
int main() {
  int n1, n2;
  cout << "Please enter two numbers:  ";
  cin >> n1 >> n2;

  try {
     if ( n2 == 0 )  {
       throw n1;
     }
     cout << "Quotient:  " << n1 / n2 << endl;
  }
  catch (int e) {
     cout << "Cannot divide " << e << " by zero!" << endl;
  }
}
```

```
Please enter two numbers:  4 0
Cannot divide 4 by zero!
```

Code inside try { } is executed in normal circumstance

If exception is encountered - it is “thrown”

The exception is “caught” and code in catch {} is executed

C++ Exceptions

```
try {

  // code that may result in an integer exception

}
catch (int e) {

  // code that handles integer exceptions

}
```

Why is this helpful?

- You can "throw" anything…
- You declare what *type* of data each catch block "handles"

Function Example

```
int quotient(int a, int b) {
   if ( b == 0 ) {
     ?????
   }
   else {
     return a / b;
   }
}

int main() {
   int n1, n2;
   cout << "Please enter two numbers:  ";
   cin >> n1 >> n2;
   cout << "Quotient:  " << quotient(n1, n2) << endl;
}
```

What should the function return?

Function Example

```
int quotient(int a, int b) {
  if ( b == 0 ) {
    throw a;
  }
  else {
    return a / b;
  }
}

int main() {
  int n1, n2;
  cout << "Please enter two numbers:  ";
  cin >> n1 >> n2;
  try {
    cout << "Quotient:  " << quotient(n1, n2) << endl;
  }
  catch (int e) {
    cout << "Cannot divide " << e << " by zero!" << endl;
  }
}
```

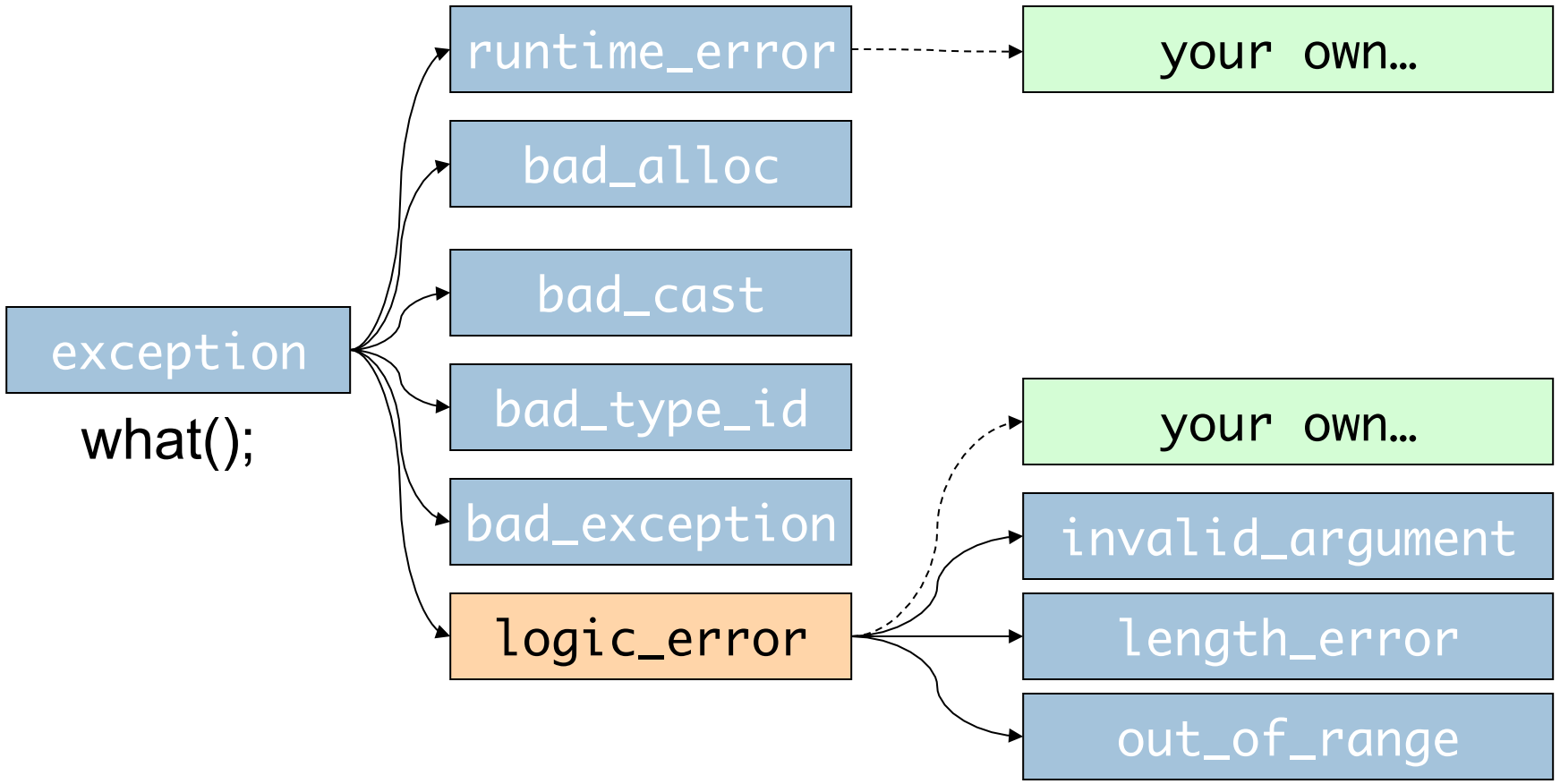
Exceptions are perfect when there is no “natural” return value due to error

Try blocks surround functions which *may* throw an exception

Exception Classes

- You can throw ints, double, chars, bools, strings, etc.
  - More helpful, you can create new Classes to represent different types of "Exceptions"
  - These classes can encapsulate detailed error information
  - You can also use / extend some basic Exception classes C++ provides

C++ Exception Classes



Custom Exception Classes

- Lets revisit our shapes demo…
  - No side of a shape should be negative
    - Implement as "NegativeDimensionException"
    - Lets derive ***invalid_argument***

Multiple Catches

```
Circle c;
Triangle t;

try {
    ....

}
catch ( NegativeDimensionException nde) {
          ...
}
catch (runtime_error re) {
     ...
}
catch (exception e) {
     ...
}
```

- You can catch many types of exceptions

Exceptions & Inheritance

```
Circle c;
Triangle t;

try {
    ....

}
catch (exception e) {
     ...
}
catch (NegativeDimensionException nde) {
          ...
}
```

- Careful!
  - Our custom exception **is a** instance of **exception**
  - Exceptions are handled by the first matching catch block
  - Your catch block for nde will never execute!

Exception Propagation

- When an exception is thrown, it "bubbles" up the "call stack" until it finds a matching catch block
  - If none are found, program **terminates**
    - **More specifically, the OS gets the exception as a signal, and kills your program…**

Exception Propagation

```
int main() {
  try {
     function1();
  }
  catch (Exception1 & ex1) {
    cout << "Error A";
  }
}
```

```
void function1() {
  try {
        ....
        function2();
        ....
   }
   catch (Exception2 ex2) {
        cout << "Error B";
   }
}
```

```
void function2() {
  try {
     ....
     function3();/// Throws an exception...
     ....
  }
  catch (Exception3 ex3) {
     cout << "Error C";
  }
}
```

Of type Exception1?
Of type Exception2?
Of type Exception3?
Of other type?

Another Example

- Override the [] operator
  - Abstract on shape
  - [1] on circle (returns the radius)
  - [2] on rectangle (returns height or width)
  - [3] on triangle (returns sides A, B, or C)

Throw appropriate exceptions – `out_of_range`

To use [] with *any* shape, the operator needs to be in the base class too…

General Triangle

- The shape of the triangle is determined by the lengths of the sides alone. Therefore the area can also be derived from the lengths of the sides. By Heron's formula:

$$T = \sqrt{s(s-a)(s-b)(s-c)}$$

- where s is the semiperimeter, or half of the triangle's perimeter.

Excercise

- The sum of the lengths of any two sides of a triangle always exceeds the length of the third side, a principle known as the triangle inequality
  - Create a triangle class that uses the full 3-sided specification, along with appropriate exceptions
  - It should inherit from the shape class
  - Do not let the sides of the triangle **ever** enter an invalid state!