C++ for Science and Engineering COSC3000/6000

2018 Spring Semester

Part XVII
Exception Handling

1 Exception Handling Basics

- It is often easier to write a program by first assuming that nothing incorrect will happen
- Once it works correctly for the expected cases, add code to handle the exceptional cases
- **Exception handling** is commonly used to handle error situations
  - Once an error is handled, it is no longer an error

1.1 A Toy Example

- Suppose milk is so important that we almost never run out
- We still would like our program to handle the situation of running out of milk
- Code to handle the normal situations involving milk, might be:

```
cout << "Enter number of donuts:\n";
cin >> donuts;
cout << "Enter number of glasses of milk:\n";
cin >> milk;
dpg = donuts /static_cast<double>(milk);
cout << donuts << " donuts.\n" << milk << " glasses of milk.\n";
cout << "You have " << dpg << " donuts per glass of milk.\n";
```

1.1.1 The No Milk Problem

- If there is no milk, the code results in a division by zero.
  - We could add a test case for this situation

```
cout << "Enter number of donuts:\n";
cin >> donuts;
cout << "Enter number of glasses of milk:\n";
cin >> milk;
if (milk <= 0){
    cout << donuts << " donuts and No milk!.\n";
    cout << "Go buy some milk!.\n";
}else{
    dpg = donuts /static_cast<double>(milk);
    cout << donuts << " donuts.\n" << milk << " glasses of milk.\n";
    cout << "You have " << dpg << " donuts per glass of milk.\n";
}
```

- With exception

```
try
{
    cout << "Enter number of donuts:\n";
    cin >> donuts;
    cout << "Enter number of glasses of milk:\n";
    cin >> milk;
    if (milk <= 0) throw donuts;

    dpg = donuts /static_cast<double>(milk);
    cout << donuts << " donuts.\n" << milk << " glasses of milk.\n";
    cout << "You have " << dpg << " donuts per glass of milk.\n";
}
catch(int e)
{
    cout << donuts << " donuts and No milk!.\n";
    cout << "Go buy some milk!.\n";
}
```

- The **try** block encloses code that you want to "**try**" but that could cause a problem .
- To throw an exception, a **throw**-statement is used to throw a value.
  - In the milk example: **throw donuts**; throws an integer value.
  - The value thrown is sometimes called an exception.
  - You can throw a value of any type.
- Something that is thrown goes from one place to another.
- In C++ **throw** causes the flow of control to go to another place.
  - When an exception is thrown, the **try** block stops executing and the **catch**-block begins execution.
  - This is catching or handling the exception.
  - If no exception is thrown, the **catch**-block is ignored during program execution.
- The **catch**-block parameter, (note that the **catch**-block is not a function) does two things:
  - The type of the **catch**-block parameter identifies the kind of value the **catch**-block can catch.
  - The **catch**-block parameter provides a name for the value caught so you can write code using the value that is caught.

try-throw-catch Review

- This is the basic mechanism for throwing and catching exceptions
  - The **try**-block includes a throw-statement
  - If an exception is thrown, the **try**-block ends and the **catch**-block is executed
  - If no exception is thrown, then after the **try**-blockis completed, execution continues with the code following the **catch**-block(s)

1.2 Exception Class

- Because a **throw**-statement can throw a value of any type, it is common to define a class whose objects can carry the kind of information you want thrown to the **catch**-block.

- A more important reason for a specialized exception class is so you can have a different type to identify each possible kind of exceptional situation.

```
// Note this is an example for learning purpose.
// The C++ Standard library provides a base class specifically
// designed to declare objects to be thrown as exceptions.
class NoMilk{
public:
    NoMilk():count(0){};
    NoMilk(int how_many):count(how_many){};
    int get_donuts() const { return count;};
private:
    int count;
};

int main(int argc, char **argv)
{
    int donuts;
    int milk;
    double dpg;
    try
    {
        cout << "Enter number of donuts:\n";
        cin >> donuts;
        cout << "Enter number of glasses of milk:\n";
        cin >> milk;
        if (milk <= 0) throw NoMilk(donuts);

        dpg = donuts /static_cast<double>(milk);
        cout << donuts << " donuts.\n" << milk << " glasses of milk.\n";
        cout << "You have " << dpg << " donuts per glass of milk.\n";
    }
    catch(NoMilk e)
    {
        cout << e.get_donuts() << " donuts and No milk!.\n";
        cout << "Go buy some milk!.\n";
    }
    return 0;
}
```

- The program above uses the throw-statement
  throw NoMilk(donuts);
  - This invokes a constructor for the class **NoMilk**
  - The constructor takes a single argument of type **int**
  - The **NoMilk** object is what is thrown.
  - The **catch**-block then uses the statement **e.get_donuts( )** to retrieve the number of donuts.

1.3 Multiple Throws and Catches

- A **try**-block can throw any number of exceptions of different types

```
try {
  // code here
}
catch (int param) { cout << "int exception"; }
catch (char param) { cout << "char exception"; }
catch (...) { cout << "default exception"; }
```

- If an ellipsis (...) is used as the parameter of **catch**, that handler will catch any exception no matter what the type of the exception thrown. This can be used as a default handler that catches all exceptions not caught by other handlers.