Activity #26: Exception Handling
Recorder's Report

Manager: Reader:

Recorder: Driver:

Date: Score: Satisfactory / Not Satisfactory

Record your team's answers to the key questions (marked with ) below.

a) Model 1, Question #4

b) Model 2, Question #8

c) Model 3, Question #10

Activity #26: Exception Handling

In this activity, you will work in teams of 3–4 students to learn new concepts. This activity will introduce you to exception handling in C++.

Content Learning Objectives

After completing this activity, students should be able to:

- Explain various methods for detecting and handling exceptions
- Explain the advantages of exception handling over printing error messages

Process Skill Goals

During the activity, students should make progress toward:

- Write C++ code to throw and catch exceptions

Model 1 Temperature Conversion

```
1  class TempConvert {
2    public:
3      double convertTemp(double temp, string inScale, string outScale) {
4        inScale = normalizeScale(inScale);
5        outScale = normalizeScale(outScale);
6        if (inScale == outScale) { return temp;
7        } else if (inScale == "C") { return cToF(temp);
8        } else { return fToC(temp);
9        }
10     }
11   private:
12     double cToF(double c) { return 32 + c * 9 / 5; }
13     double fToC(double f) { return (f - 32) * 5 / 9; }
14     string normalizeScale(string s) {
15       for_each(s.begin(),s.end(), [](char & c) { c = ::tolower(c); });
16       if (s == "c" || s.substr(0, 4) == "cels") { return "C"; }
17       if (s == "f" || s.substr(0, 4) == "fahr") { return "F"; }
18       if (s == "k" || s.substr(0, 4) == "kelv") { return "K"; }
19       return "?";
20     }
21 };
22
```

Refer to Model 1 above as your group develops consensus answers to the questions below.

Questions (20 min) Start time:

1. The temperature conversion class above has one public method and three private methods. Determine the **return value** for each method call given below.

a) `normalizeScale("f")`

b) `normalizeScale("Cel")`

c) `convertTemp(-273.15, "C", "Fahrenheit")`

d) `convertTemp(100, "F", "CeLsIuS")`

2. The file `activity26a.cpp` contains this class along with a program that utilizes it. This program could run into trouble if the user supplies invalid input. For example, a user could enter an invalid or unknown scale, or provide invalid temperatures (e.g. below the absolute zero of -273.15 Celsius). Describe the problem that would be encountered with each set of user input below. Note that the semicolons are used to separate lines of input and are not part of the input themselves.

a) `12;inches;fahr`

b) `16000000;C;K`

c) `-300;cels;fahr`

3. A program could respond to errors such as the ones above in several ways. One way commonly used in simple programs is to *print an error message*.

a) In what method(s) should error messages about invalid or unknown temperature scales be printed?

b) In what method(s) should error messages about invalid temperatures be printed?

c) In what method(s) should error messages about unimplemented conversions (e.g. kelvin to fahrenheit) be printed?

d) Modify two of the methods identified above in the file `activity26a.cpp` to print appropriate error messages for the user input from problem 2.

4. Many programs can not handle errors by simply printing out error messages. 
For example:

- Embedded systems, such as home appliances or automobiles, which do not have a traditional display and need to run unattended for days or even years.

- Computationally intense software for business or scientific applications that may run for hours or days on a cloud server without human monitoring.

Explain why printing messages may not be a good idea in software such as those mentioned above.

Model 2 Identifying Errors in Temperature Conversion

```
1  class TempConvert {
2  public:
3    double convertTemp(double temp, string inScale, string outScale) {
4      inScale = normalizeScale(inScale);
5      outScale = normalizeScale(outScale);
6      if (inScale == outScale) { return temp;
7      } else if (inScale == "C" && outScale == "F") { return cToF(temp);
8      } else if (inScale == "F" && outScale == "C") { return fToC(temp);
9      } else if (inScale == "K" && outScale == "C") { return kToC(temp);
10     } else if (inScale == "K" && outScale == "F") { return cToF(kToC(temp));
11     } else throw invalid_argument("Conversion Not Implemented");
12   }
13 private:
14   double cToF(double c) {
15      if(c < 273.15) throw domain_error("Invalid Temp");
16      return 32 + c * 9 / 5;
17   }
18   double fToC(double f) {
19      if(f < -459.67) throw domain_error("Invalid Temp");
20      return (f - 32) * 5 / 9;
21   }
22   double kToC(double k) {
23      if(k < 0) throw domain_error("Invalid Temp");
24      return k + 273.15;
25   }
26   string normalizeScale(string s) {
27     for_each(s.begin(),s.end(), [](char & c) { c = ::tolower(c); });
28     if (s == "c" || s.substr(0, 4) == "cels") { return "C"; }
29     if (s == "f" || s.substr(0, 4) == "fahr") { return "F"; }
30     if (s == "k" || s.substr(0, 4) == "kelv") { return "K"; }
31     throw domain_error("Invalid Scale");
32   }
33 };
34
```

Refer to Model 2 above as your group develops consensus answers to the questions below.

Questions (15 min) Start time:

5. In this model, our class has been modified to support additional temperature conversions. Describe what changes have been made to support conversions between Kelvin and the other temperature scales.

6. Robust error handling (also called *exception handling*) typically separates the task of *identifying* exceptions from the task of *handling* exceptions. In the model above, we have added code to identify exceptions.

a) Which of the methods in this model include exception identification?

b) How many different types of exceptions are identified in the entire class?

c) In C++, what command is used to identify an exception?

7. While we have identified exceptions in this model, we have not explicitly handled them, so C++ uses its default exception handling technique. Run the code found in `activity26b.cpp` and determine how the exceptions generated by each set of user input is handled.

a) `12;inches;fahr`

b) `16000000;C;K`

c) `-300;cels;fahr`

8. Describe how you might wish to alter the default exception handling technique in the following contexts. If you think the default is fine as it is, state that.

a) If the class is being used to convert temperatures interactively, as in this example.

b) If the class is being used to convert a large array of temperatures without human interaction.

Model 3 Handling Errors in Temperature Conversion

```
TempConvert tc;

double t[6] = {20,-30,170,25,-12,14 };
string u[6] = {"C","C","K","F","K","C"};
double newT[6];

for(int i=0; i<6; i++) {
  try {
    newT[i] = tc.convertTemp(t[i],u[i],"F");
  } catch (logic_error &except) {
    newT[i] = NAN;
  }
}
```

Use Case #1

```
TempConvert tc;
double t, newT;
string u;
while(true) {
  try {
    cout << "Temp to Convert: ";
    cin >> t;
    cout << "Units: ";
    cin >> u;
    newT = tc.convertTemp(t,u,"F");
  }
  catch (logic_error &except) {
    cout << except.what() << endl;
    continue;
  }
  break;
}
```

Use Case #2

Refer to Model 3 above as your group develops consensus answers to the questions below.

Questions (15 min) Start time:

9. The model above shows two different use cases for our temperature conversion class. Answer the following general questions about these use cases.

a) What new keywords are used to handle the exceptions that were *thrown* in the class?

b) When is the code in the `try` block run?

c) When is the code in the `catch` block run?

10. The code for use case #1 can be found in the file `activity26c.cpp`. Run it and answer the following questions.

a) What exception is thrown by the temperature conversion class and why is it thrown?

b) How is this exception handled by the program?

c) Why might this be a better way to handle the exception than the C++ default technique?

11. Modify the file `activity26b.cpp` so that the main program looks like the one in use case #2 of our model. Then run it and answer the following questions.

a) Give appropriate user input to generate two different types of exceptions.

b) How are these exceptions handled in this use case?

c) Why might this be a better way to handle the exceptions than the C++ default technique?