13.1 Exception basics

Error-checking code is code a programmer writes to detect and handle errors that occur during program execution. An ***exception*** is a circumstance that a program was not designed to handle, such as if the user enters a negative height.

The following program, given a person's weight and height, outputs a person's body-mass index (BMI), which is used to determine normal weight for a given height. The program has no error checking.

Figure 13.1.1: BMI example without error checking.

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

int main() {
   int weightVal;       // User defined weight (lbs)
   int heightVal;       // User defined height (in)
   float bmiCalc;       // Resulting BMI
   char quitCmd;        // Indicates quit/continue

   quitCmd = 'a';

   while (quitCmd != 'q') {

      // Get user data
      cout << "Enter weight (in pounds): ";
      cin >> weightVal;

      cout << "Enter height (in inches): ";
      cin >> heightVal;

      // Calculate BMI value
      bmiCalc = (static_cast<float>(weightVal) /
                 static_cast<float>(heightVal * heightVal))
* 703.0;

      // Print user health info
      // Source: http://www.cdc.gov/
      cout << "BMI: " << bmiCalc << endl;
      cout << "(CDC: 18.6-24.9 normal)" << endl;

      // Prompt user to continue/quit
      cout << endl << "Enter any key ('q' to quit): ";
      cin >> quitCmd;
   }

   return 0;
}
```

```
Enter weight (in pounds):
150
Enter height (in inches):
66
BMI: 24.208
(CDC: 18.6-24.9 normal)

Enter any key ('q' to
quit): a
Enter weight (in pounds):
-1
Enter height (in inches):
66
BMI: -0.161387
(CDC: 18.6-24.9 normal)

Enter any key ('q' to
quit): a
Enter weight (in pounds):
150
Enter height (in inches):
-1
BMI: 105450
(CDC: 18.6-24.9 normal)

Enter any key ('q' to
quit): q
```


Naively adding error-checking code using if-else statements obscures the normal code. And redundant checks are ripe for errors if accidentally made inconsistent with normal code. Problematic code is highlighted.

Figure 13.1.2: BMI example with error-checking code but without using exception-handling constructs.

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

int main() {
   int weightVal;       // User defined weight (lbs)
   int heightVal;       // User defined height (in)
   float bmiCalc;       // Resulting BMI
   char quitCmd;        // Indicates quit/continue

   quitCmd = 'a';

   while (quitCmd != 'q') {

      // Get user data
      cout << "Enter weight (in pounds): ";
      cin >> weightVal;

      // Error checking, non-negative weight
      if (weightVal < 0) {
         cout << "Invalid weight." << endl;
      }
      else {
         cout << "Enter height (in inches): ";
         cin >> heightVal;

         // Error checking, non-negative height
         if (heightVal < 0) {
            cout << "Invalid height." << endl;
         }
      }

      // Calculate BMI and print user health info if no
input error
      // Source: http://www.cdc.gov/
      if ((weightVal <= 0) || (heightVal <= 0)) {
         cout << "Cannot compute info." << endl;
      }
      else {
         bmiCalc = (static_cast<float>(weightVal) /
                    static_cast<float>(heightVal *
heightVal)) * 703.0;

         cout << "BMI: " << bmiCalc << endl;
         cout << "(CDC: 18.6-24.9 normal)" << endl;
      }

      // Prompt user to continue/quit
      cout << endl << "Enter any key ('q' to quit): ";
      cin >> quitCmd;
   }

   return 0;
}
```



```
Enter weight (in pounds):
150
Enter height (in inches):
66
BMI: 24.208
(CDC: 18.6-24.9 normal)

Enter any key ('q' to
quit): a
Enter weight (in pounds):
-1
Invalid weight.
Cannot compute info.

Enter any key ('q' to
quit): a
Enter weight (in pounds):
150
Enter height (in inches):
-1
Invalid height.
Cannot compute info.

Enter any key ('q' to
quit): q
```

©zyBooks 04/25/21 07:42 488201
xiang zhao
BAYLORCSI14301440Spring2021

The language has special constructs, try, throw, and catch, known as ***exception-handling constructs***, to keep error-checking code separate and to reduce redundant checks.

Construct 13.1.1: Exception-handling constructs.

```
// ... means normal code
...
try {
   ...
   // If error detected
      throw objectOfExceptionType;
   ...
}
catch (exceptionType excptObj) {
   // Handle exception, e.g., print message
}
...
```


PARTICIPATION ACTIVITY 13.1.1: How try, throw, and catch handle exceptions.

Animation captions:

1. A try block surrounds normal code. A throw statement appears within a try block; if reached, execution jumps immediately to the end of the try block.
2. A catch clause immediately follows a try block; if the catch was reached due to an exception thrown of the catch clause's parameter type, the clause executes.

- A ***try*** block surrounds normal code, which is exited immediately if a throw statement executes.
- A ***throw*** statement appears within a try block; if reached, execution jumps immediately to the end of the try block. The code is written so only error situations lead to reaching a throw. The throw statement provides an object of a particular type, such as an object of type "runtime_error", which is a class defined in the ***stdexcept library***. The statement is said to throw an exception of the particular type. A throw statement's syntax is similar to a return statement.
- A ***catch*** clause immediately follows a try block; if the catch was reached due to an exception thrown of the catch clause's parameter type, the clause executes. The clause is said to catch the thrown exception. A catch block is called a ***handler*** because it handles an exception.

The following shows the earlier BMI program using exception-handling constructs. Notice that the normal code flow is not obscured by error-checking/handling if-else statements. The flow is clearly: Get weight, then get height, then print BMI.

Figure 13.1.3: BMI example with error-checking code using exception-handling constructs.

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

int main() {
   int weightVal;       // User defined weight (lbs)
   int heightVal;       // User defined height (in)
   float bmiCalc;       // Resulting BMI
   char quitCmd;        // Indicates quit/continue

   quitCmd = 'a';

   while (quitCmd != 'q') {

      try {
         // Get user data
         cout << "Enter weight (in pounds): ";
         cin >> weightVal;

         // Error checking, non-negative weight
         if (weightVal < 0) {
            throw runtime_error("Invalid weight.");
         }

         cout << "Enter height (in inches): ";
         cin >> heightVal;

         // Error checking, non-negative height
         if (heightVal < 0) {
            throw runtime_error("Invalid height.");
         }

         // Calculate BMI and print user health info if no
input error
         // Source: http://www.cdc.gov/
         bmiCalc = (static_cast<float>(weightVal) /
                    static_cast<float>(heightVal *
heightVal)) * 703.0;

         cout << "BMI: " << bmiCalc << endl;
         cout << "(CDC: 18.6-24.9 normal)" << endl;
      }
      catch (runtime_error& excpt) {
         // Prints the error message passed by throw
statement
         cout << excpt.what() << endl;
         cout << "Cannot compute health info." << endl;
      }

      // Prompt user to continue/quit
      cout << endl << "Enter any key ('q' to quit): ";
      cin >> quitCmd;
   }

   return 0;
}
```

```
Enter weight (in pounds):
150
Enter height (in inches):
66
BMI: 24.208
(CDC: 18.6-24.9 normal)

Enter any key ('q' to
quit): a
Enter weight (in pounds):
-1
Invalid weight.
Cannot compute health
info.

Enter any key ('q' to
quit): a
Enter weight (in pounds):
150
Enter height (in inches):
-1
Invalid height.
Cannot compute health
info.

Enter any key ('q' to
quit): q
```

Conceptually the item thrown and caught can be any type such as int or char*. So `throw 3;` and `catch (int& excpt) {...}` is allowable. Normally, though, the object thrown is of a class type, and commonly one of the types defined in the stdexcept standard library (or is derived from such a type). The `runtime_error` type is such a type, which is why the stdexcept library was included above. The runtime_error type has a constructor that can be passed a string, as in `throw runtime_error("Invalid weight.");`, which sets an object's internal string value that can later be retrieved using the what() function, as in `cout << excpt.what() << endl;`. The catch parameter is typically a reference parameter (via &) for reasons related to inherited exception objects, which is beyond our scope here.

PARTICIPATION ACTIVITY 13.1.2: Exceptions.

Select the one code region that is incorrect.

1)
```
try {
  if (weight < 0) {
    try
  runtime_error("Invalid weight.");
  }

  // Print user health info
  // ...
}
catch (runtime_error& excpt ) {
  cout << excpt.what() << endl;
  cout << "Cannot compute health info." << endl;
}
```

2)
```
try {
  if (weight < 0) {
    throw runtime_error( "Invalid weight." );
  }

  // Print user health info
  // ...
}
catch ( runtime_error excpt ) {
  cout << excpt() << endl;
  cout << "Cannot compute health info." << endl;
}
```


PARTICIPATION ACTIVITY 13.1.3: Exception basics.

1) After an exception is thrown and a catch block executes, execution

resumes after the throw statement.

- True
- False

2) A compiler generates an error message if a try block is not immediately followed by a catch block.

- True
- False

3) If no throw is executed in a try block, then the subsequent catch block is not executed.

- True
- False

CHALLENGE ACTIVITY 13.1.1: Exception handling.

Start

Type the program's output

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

int main() {
   int userAge;
   int avgMaxHeartRate;

   try {
      cin >> userAge;

      if (userAge < 0) {
         throw runtime_error("Invalid age");
      }

      // Source: https://www.heart.org/en/healthy-living/fitness
      avgMaxHeartRate = 220 - userAge;

      cout << "Avg: " << avgMaxHeartRate << endl;
   }
   catch (runtime_error& excpt) {
      cout << "Error: " << excpt.what() << endl;
   }

   return 0;
}
```

Input

30

Output

1 2 3

Check Next

Exploring further:

- Intro to exceptions tutorial from cplusplus.com
- Exceptions reference page from cplusplus.com

13.2 Exceptions with functions

The power of exceptions becomes clearer when used within a function. If an exception is thrown within a function and not caught within that function, then the function is immediately exited and the calling function is checked for a handler, and so on up the function call hierarchy. The following illustrates; note the clarity of the normal code.

Figure 13.2.1: BMI example using exception-handling constructs along with functions.

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

int GetWeight() {
   int weightParam;     // User defined weight

   // Get user data
   cout << "Enter weight (in pounds): ";
   cin >> weightParam;

   // Error checking, non-negative weight
   if (weightParam < 0) {
      throw runtime_error("Invalid weight.");
   }
   return weightParam;
}

int GetHeight() {
   int heightParam;     // User defined height

   // Get user data
   cout << "Enter height (in inches): ";
   cin >> heightParam;

   // Error checking, non-negative height
   if (heightParam < 0) {
      throw runtime_error("Invalid height.");
   }
```

```
Enter weight (in pounds):
150
Enter height (in inches):
66
BMI: 24.208
(CDC: 18.6-24.9 normal)

Enter any key ('q' to
quit): a
Enter weight (in pounds):
-1
Invalid weight.
Cannot compute health
info.

Enter any key ('q' to
quit): a
Enter weight (in pounds):
150
Enter height (in inches):
-1
Invalid height.
Cannot compute health
info.

Enter any key ('q' to
quit): q
```

```
   return heightParam;
}

int main() {
   int weightVal;         // User defined weight (lbs)
   int heightVal;         // User defined height (in)
   float bmiCalc;         // Resulting BMI
   char quitCmd;          // Indicates quit/continue

   quitCmd = 'a';

   while (quitCmd != 'q') {
      try {
         // Get user data
         weightVal = GetWeight();
         heightVal = GetHeight();

         // Calculate BMI and print user health info if no
input error
         // Source: http://www.cdc.gov/
         bmiCalc = (static_cast<float>(weightVal) /
                    static_cast<float>(heightVal *
heightVal)) * 703.0;

         cout << "BMI: " << bmiCalc << endl;
         cout << "(CDC: 18.6-24.9 normal)" << endl;
      }
      catch (runtime_error &excpt) {
         // Prints the error message passed by throw
statement
         cout << excpt.what() << endl;
         cout << "Cannot compute health info." << endl;
      }

      // Prompt user to continue/quit
      cout << endl << "Enter any key ('q' to quit): ";
      cin >> quitCmd;
   }

   return 0;
}
```


Suppose getWeight() throws an exception of type Exception. GetWeight() immediately exits, up to main() where the call was in a try block, so the catch block catches the exception.

Note the clarity of the code in main(). Without exceptions, GetWeight() would have had to somehow indicate failure, perhaps returning -1. Then main() would have needed an if-else statement to detect such failure, obscuring the normal code.

If no handler is found going up the call hierarchy, then terminate() is called, which typically aborts the program.

PARTICIPATION ACTIVITY 13.2.1: Exceptions.

1) For a function that may contain a throw, all of the function's statements, including the throw, must be surrounded by a try block.

- ○ True
- ○ False

2) A throw executed in a function automatically causes a jump to the last return statement in the function.

- ○ True
- ○ False

3) A goal of exception handling is to avoid polluting normal code with distracting error-handling code.

- ○ True
- ○ False

13.3 Multiple handlers

Different throws in a try block may throw different exception types. Multiple handlers may exist, each handling a different type. The first matching handler executes; remaining handlers are skipped.

catch(...) is a catch-all handler that catches any type, which is useful when listed as the last handler.

Construct 13.3.1: Exception-handling: multiple handlers.

```
// ... means normal code
...
try {
   ...
   throw objOfExcptType1;
   ...
   throw objOfExcptType2;
   ...
   throw objOfExcptType3;
   ...
}
catch (ExcptType1& excptObj) {
   // Handle type1
}
catch (ExcptType2& excptObj) {
   // Handle type2
}
catch (...) {
   // Handle others (e.g., type3)
}
... // Execution continues here
```

PARTICIPATION ACTIVITY 13.3.1: Multiple handlers.

Animation captions:

1. Different throws in a try block may throw different exception types. Multiple handlers may exist, each handling a different type.
2. catch(...) is a catch-all handler that catches any type.
3. The first matching handler executes; remaining handlers are skipped.

A thrown exception may also be caught by a catch block meant to handle an exception of a base class. If in the above code, ExcptType2 is a subclass of ExcptType1, then objOfExcptType2 will always be caught by the first catch block instead of the second catch block, which is typically not the intended behavior. A common error is to place a catch block intended to handle exceptions of a base class before catch blocks intended to handle exceptions of a derived class, preventing the latter from ever executing.

PARTICIPATION ACTIVITY 13.3.2: Exceptions with multiple handlers.

Refer to the multiple handler code above.

1) If an object of type ExcptType1 is thrown, three catch blocks will execute.
   - True
   - False

2) If an object of type ExcptType3 is thrown, no catch blocks will execute.
   - True
   - False

3) A second catch block can never execute immediately after a first one executes.
   - True
   - False

4) If ExcptType2 inherits from ExcptType1, then the second catch

block (i.e., `catch (ExcptType2& excptObj)`) will never be executed.

- True
- False

13.4 C++ example: Generate number format exception

zyDE 13.4.1: Catch exception reading integer from stringstream.

Running the below program with the given input causes an error when extracting an inte from a stringstream. The program reads from cin the following rows (also called record contain a last name, first name, department, and annual salary. The program uses the stringstream to convert the last entry for the salary to an integer.

```
Argon,John,Operations,50000
Williams,Jane,Marketing,sixty_thousand
Uminum,Al,Finance,70000
Jones,Ellen,Sales,80000
```

Note that the second row has a value that is type string, not type int, which will cause a problem.

1. Run the program and note the program fails and throws an ios_base::failure excep
2. Add try/catch statements to catch the ios_base::failure exception. In this case, pri message, and do not add the item to the total salaries.
3. Run the program again and note the total salaries excludes the row with the error.

Load default templ


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

int main() {
   // Describe the format of a row of input. There are four fields in
   // a row separated by commas: last name, first name, department, salary
```

```
   const string SEPARATOR     = ",";  // field separator in each row of data
   const int INDEX_LAST_NAME  = 0;    // # of the last name field
   const int INDEX_FIRST_NAME = 1;    // # of the first name field
   const int INDEX_DEPT       = 2;    // # of the department name field
   const int INDEX_SALARY     = 3;    // # of the salary field
   stringstream ss;                   // For conversion of string to int
   int salary;
```

```
Doe,John,Operations,50000
Doette,Jane,Marketing,sixty_thousand
Uminum,Al,Finance,70000
```


Run

zyDE 13.4.2: Catch number format error (solution).

Below is a solution to the above problem.

Load default templ

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

int main() {
   // Describe the format of a row of input. There are four fields in
   // a row separated by commas: last name, first name, department, salary
   const string SEPARATOR     = ",";  // field separator in each row of data
   const int INDEX_LAST_NAME  = 0;    // # of the last name field
   const int INDEX_FIRST_NAME = 1;    // # of the first name field
   const int INDEX_DEPT       = 2;    // # of the department name field
   const int INDEX_SALARY     = 3;    // # of the salary field
   stringstream ss;                   // For conversion of string to int
   int salary;
```

```
Doe,John,Operations,50000
Doette,Jane,Marketing,sixty_thousand
Uminum,Al,Finance,70000
```


Run

13.5 LAB: Exception handling to detect input string vs. int

The given program reads a list of single-word first names and ages (ending with -1), and outputs that list with the age incremented. The program fails and throws an exception if the second input on a line is a string rather than an int. At FIXME in the code, add a try/catch statement to catch `ios_base::failure`, and output 0 for the age.

Ex: If the input is:

```
Lee 18
Lua 21
Mary Beth 19
Stu 33
-1
```

then the output is:

```
Lee 19
Lua 22
Mary 0
Stu 34
```

LAB ACTIVITY 13.5.1: LAB: Exception handling to detect input string vs. int 0 / 10

main.cpp Load default template...

```
1  #include <string>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(int argc, char* argv[]) {
7     string inputName;
8     int age;
9     // Set exception mask for cin stream
10    cin.exceptions(ios::failbit);
11
12    cin >> inputName;
13    while(inputName != "-1") {
14       // FIXME: The following line will throw an ios_base::failure.
```

```
15      //        Insert a try/catch statement to catch the exception.
16      //        Clear cin's failbit to put cin in a useable state.
17      cin >> age;
```

Develop mode | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **main.cpp** (Your program) → Outp

Program output displayed here

Signature of your work What is this?

History of your effort will appear here once you begin working on this zyLab.

©zyBooks 04/25/21 07:42 488201
xiang zhao
BAYLORCSI14301440Spring2021