17.1 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 17.1.1: 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 17.1.2: 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;
}
```

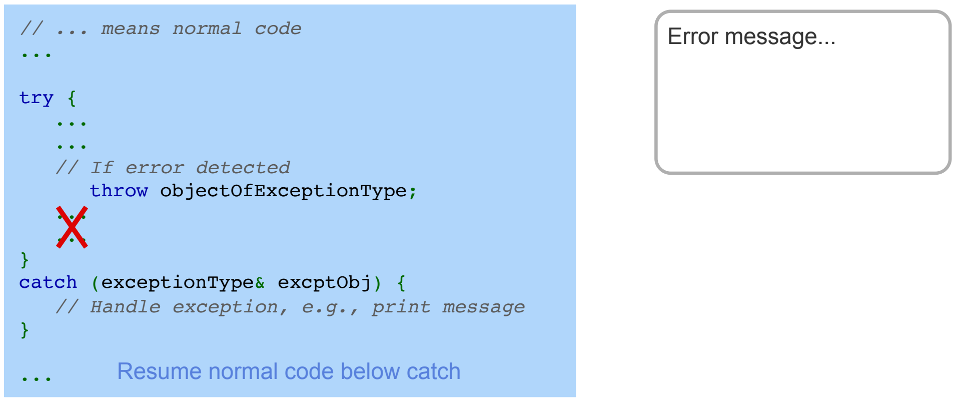
```
Enter weight (in
pounds): 150
Enter height (in
inches): 66
BMI: 24.208
(CDC: 18.6-24.9 normal)

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
```

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 17.1.1: Exception-handling constructs.

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

PARTICIPATION ACTIVITY | 17.1.1: How try, throw, and catch handle exceptions.



```
// ... means normal code
...

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

...
```

Animation content:

Static figure:
Begin C++ code:

```
// ... means normal code
...

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

...
```

End C++ code.

Step 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. The output console is empty. In the try block, the line of code, throw objectOfExceptionType;, is highlighted and the remaining lines of code in the try block are crossed out.

Step 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. In the catch block, the line of code, // Handle exception, e.g., print message, is highlighted. The output console now contains one line of output:
Error message...

The words, Resume normal code below catch, appear at the bottom of the static figure.

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 17.1.3: 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 | 17.1.2: 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 | 17.1.3: 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

Table 17.1.1: Common exception types.

Type	Reason exception is thrown
bad_alloc	Failure in allocating memory
ios_base::failure	Failure in a stream (Ex: cin, stringstream, fstream)
logic_error	To report errors in a program's logic. Ex: out_of_range error (index out of bounds)
runtime_error	To report errors that can only be detected at runtime. Ex: overflow_error (arithmetic overflow)

Source: cplusplus.com

CHALLENGE ACTIVITY 17.1.1: Exception handling.

539740.3879454.qx3zqy7

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

```
Avg: 190
```

1 2 3 4

Check Next

CHALLENGE ACTIVITY | 17.1.2: Exception basics.

539740.3879454.qx3zqy7

Start

Integer coneHeight is read from input. Complete the try block to throw a runtime error exception with the message "Bad inpu
cone's height" if coneHeight is ≤ 0.

Ex: If input is 17, then the output is:

```
Cone's height is 17
```

Ex: If input is -19, then the output is:

```
Error: Bad input for cone's height
```

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

int main() {
   int coneHeight;

   cin >> coneHeight;

   try {
      if (coneHeight <= 0) {
         throw /* Your code goes here */;
      }
      cout << "Cone's height is " << coneHeight << endl;
   }
```

1 2 3

Check Next level

Exploring further:

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

17.2 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 17.2.1: 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.");
   }
   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;
}
```

```
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
```

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 17.2.1: 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

17.3 Multiple handlers

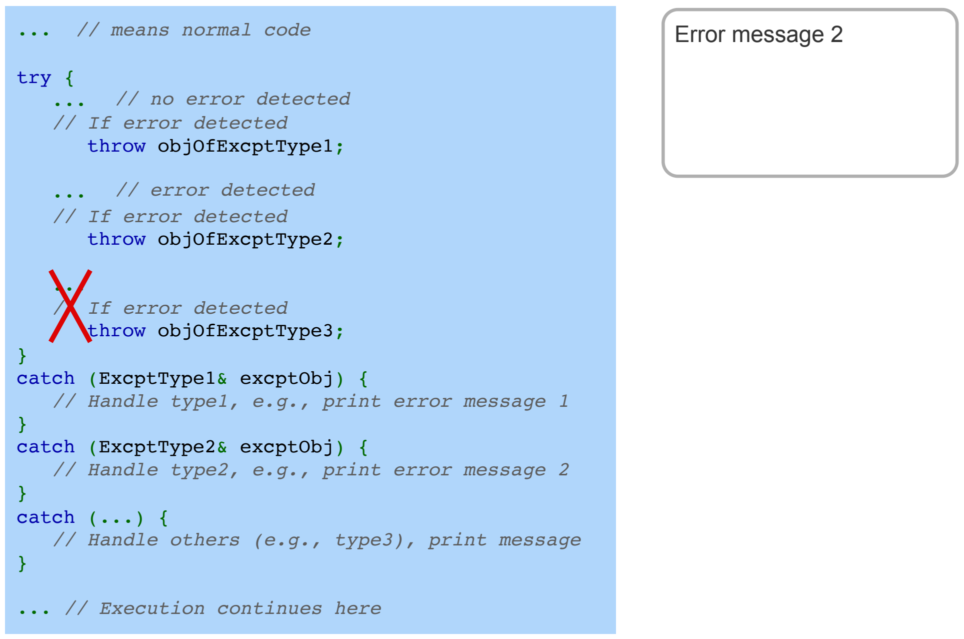
Different throws in a try block may throw different exception types. Multiple handlers may exist, each handling a different type. The first matching handler executes; remaining handlers are skipped.

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

Construct 17.3.1: 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 17.3.1: Multiple handlers.



```
...  // means normal code

try {
   ...  // no error detected
   // If error detected
      throw objOfExcptType1;

   ...  // error detected
   // If error detected
      throw objOfExcptType2;

   ...
   // If error detected
      throw objOfExcptType3;
}
catch (ExcptType1& excptObj) {
   // Handle type1, e.g., print error message 1
}
catch (ExcptType2& excptObj) {
   // Handle type2, e.g., print error message 2
}
catch (...) {
   // Handle others (e.g., type3), print message
}

... // Execution continues here
```

Error message 2

Animation content:

Static Figure:
Begin C++ code:
... // means normal code

try {
  ...
  // If error detected

```
    throw objOfExcptType1;

  ...
  // If error detected
    throw objOfExcptType2;

  ...
  // If error detected
    throw objOfExcptType3;
}
catch (ExcptType1& excptObj) {
  // Handle type1, e.g., print error message 1
}
catch (ExcptType2& excptObj) {
  // Handle type2, e.g., print error message 2
}
catch (...) {
  // Handle others (e.g., type3), print message
}

... // Execution continues here
```

End C++ code.
An output console is shown. The text “Error message 2” is shown within the output console.

Step 1: Different throws in a try block may throw different exception types. Multiple handlers may exist, each handling a different type.
Four lines of code are highlighted in the C++ code block. The throw statement, “throw objOfExcptType1;” and the catch statement associated with the throw statement, “catch (ExcptType1& excptObj)”. And the throw statement, “throw objOfExcptType2;”, and the catch statement associated with the throw statement, “catch (ExcptType2& excptObj)”.

Step 2: catch(...) is a catch-all handler that catches any type.
Two lines of code are highlighted. The throw statement, “throw objOfExcptType3;” and the catch statement for any type, “catch (...)”.

Step 3: The first matching handler executes; remaining handlers are skipped.
The C++ code executes, highlighting the first line within the try statement, indicating with a comment that no error was detected, “... //no error detected”.
The first throw statement does not execute and the execution flow moves on. The next line of code executes, indicating with a comment that an error was detected, “... // error detected”. The execution flow moves to the throw statement and highlights the code line, “throw objOfExcptType2;”. The execution flow moves to catch statement and highlights the code line, “catch (ExcptType2& excptObj)
{
  // Handle type2, e.g., print error message 2
}”. A print statement appears in the output console, “Error message 2”. The execution moves on to the final line of code, highlighting the code line, “... // Execution continues here”.
The last throw statement is marked with a large red “X” to show that the final throw statement was not needed and skipped.

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 | 17.3.2: 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

CHALLENGE ACTIVITY | 17.3.1: Enter the output of multiple exception handlers.

539740.3879454.qx3zqy7

Start

Type the program's output

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

int main() {
   stringstream ss;
   string userInput;
   int value;

   // Failed conversion throws ios_base::failure
   ss.exceptions(ios::failbit);

   getline(cin, userInput);

   while (userInput != "end") {
      try {
         ss.str("");
         ss.clear();
         ss << userInput;
         ss >> value;

         // Division by zero throws runtime_error
         if (value == 0) {
            throw runtime_error("z");
         }

         cout << 60 / value << endl;
      }
      catch (ios_base::failure& excpt) {
         cout << "t" << endl;
      }
      catch (runtime_error& excpt) {
         cout << excpt.what() << endl;
      }
      getline(cin, userInput);
      ss.clear();
   }
   cout << "OK" << endl;

   return 0;
}
```

Input

```
0
-1
two
2
end
```

Output

```
z
-60
t
30
OK
```

1 2 3

Check Next

17.4 C++ example: Generate number format exception

zyDE 17.4.1: Catch exception reading integer from stringstream.

Running the below program with the given input causes an error when extracting an integer from a stringstream. The program reads from cin the following rows (also called records) that 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 exception.
2. Add try/catch statements to catch the ios_base::failure exception. In this case, print a 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 template...

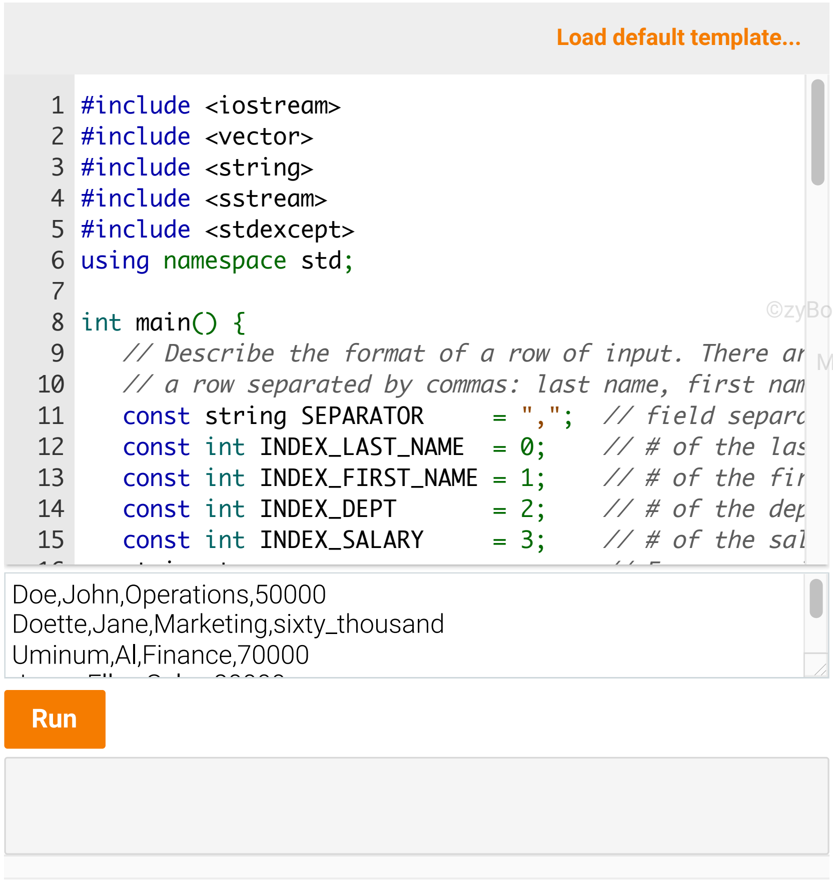
```
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
#include <stdexcept>
using namespace std;

int main() {
   // Describe the format of a row of input. There ar
   // a row separated by commas: last name, first nan
   const string SEPARATOR     = ",";  // field separc
   const int INDEX_LAST_NAME  = 0;    // # of the las
   const int INDEX_FIRST_NAME = 1;    // # of the fir
   const int INDEX_DEPT       = 2;    // # of the dep
   const int INDEX_SALARY     = 3;    // # of the sal
```

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

Run

zyDE 17.4.2: Catch number format error (solution).

Below is a solution to the above problem.



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

int main() {
   // Describe the format of a row of input. There ar
   // a row separated by commas: last name, first nam
   const string SEPARATOR     = ",";  // field separa
   const int INDEX_LAST_NAME  = 0;    // # of the las
   const int INDEX_FIRST_NAME = 1;    // # of the fir
   const int INDEX_DEPT       = 2;    // # of the dep
   const int INDEX_SALARY     = 3;    // # of the sal
```

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

17.5 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
```

Note: Insert the following code in the catch block to clear the failbit and cin buffer so a new input can be read correctly:

```
// Clear failbit to be able to use cin again
cin.clear();
```

```
// Throw away the rest of the failed input line from cin buffer
string garbage;
getline(cin, garbage);
```

539740.3879454.qx3zqy7

LAB ACTIVITY 17.5.1: LAB: Exception handling to detect input string vs. int 0 / 10

main.cpp Load default template...

```
#include <string>
#include <iostream>

using namespace std;

int main() {
   string inputName;
   int age;
   // Set exception mask for cin stream
   cin.exceptions(ios::failbit);

   cin >> inputName;
   while(inputName != "-1") {
      // FIXME: The following line will throw an ios_base::failure.
      //        Insert a try/catch statement to catch the exception.
```

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) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

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

17.6 LAB: Exceptions with vectors

Complete a program that reads a vector index as input and outputs the element of a vector of 10 names at the index specified by the input. Use a try block to output the name and a catch block to catch any out_of_range exceptions. When an out_of_range exception is caught, output the message from the exception object and the first element in the vector if the index is negative or the last element if the index is greater than the size of the vector.

Hint: Format the exception outputs using the what() function from the exception object. Do not hard code the exception messages.

Ex: If the input of the program is:

```
5
```

the program outputs:

```
Jane
```

Ex: If the input of the program is:

```
12
```

the program outputs:

```
Exception! vector::_M_range_check: __n (which is 12) >= this->size() (which is 10)
The closest name is: Johnny
```

Ex: If the input of the program is:

```
-2
```

the program outputs:

```
Exception! vector::_M_range_check: __n (which is 18446744073709551614) >= this->size()
(which is 10)
The closest name is: Ryley
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 17.6.1: LAB: Exceptions with vectors 0 / 10

main.cpp Load default template...

```
#include <iostream>
#include <vector>
#include <stdexcept>      // For std::out_of_range
using namespace std;

int main() {
   vector<string> names = { "Ryley", "Edan", "Reagan", "Henry", "Caius", "Jar
   int index;

   cin >> index;

   /* Type your code here. */

   return 0;
}
```

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) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

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

17.7 LAB: Simple integer division - multiple exception handlers

Write a program that reads integers userNum and divNum as input, and output the quotient (userNum divided by divNum). Use a try block to perform the statements and throw a runtime_error exception with the message "Divide by zero!" when a division by zero happens. Use a catch block to catch any runtime_error caused by dividing by zero and output an exception message. Use another catch block to catch any ios_base::failure caused by invalid input and output an exception message.

Note: ios_base::failure is thrown when a user enters a value of different data type than what is defined in the program. Do not write code to throw ios_base::failure exception in the program.

Ex: If the input of the program is:

```
15 3
```

the output of the program is:

```
5
```

Ex: If the input of the program is:

```
10 0
```

the output of the program is:

```
Runtime Exception: Divide by zero!
```

Ex: If the input of the program is:

```
twenty 5
```

the output of the program is:

```
Input Exception: basic_ios::clear: iostream error
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 17.7.1: LAB: Simple integer division - multiple exception handlers 0 / 10

main.cpp Load default template...

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

int main() {
   int userNum;
   int divNum;
   int result;
   cin.exceptions(ios::failbit);       // Allow cin to throw exceptions

   /* Type your code here. */

   return 0;
}

```

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) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

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

17.8 LAB: Step counter - exceptions

A pedometer treats walking 2,000 steps as walking 1 mile. Write a StepsToMiles() function that takes the number of steps as an integer parameter and returns the miles walked as a double. The StepsToMiles() function throws a runtime_error object with the message "Exception: Negative step count entered." when the number of steps is negative. Complete the main() function that reads the number of steps from a user, calls the StepsToMiles() function, and outputs the returned value from the StepsToMiles() function. Use a try-catch block to catch any runtime_error object thrown by the StepsToMiles() function and output the exception message.

Output each floating-point value with two digits after the decimal point, which can be achieved by executing `cout << fixed << setprecision(2);` once before all other cout statements.

Ex: If the input of the program is:

```
5345
```

the output of the program is:

```
2.67
```

Ex: If the input of the program is:

```
-3850
```

the output of the program is:

```
Exception: Negative step count entered.
```

539740.3879454.qx3zqy7

LAB ACTIVITY 17.8.1: LAB: Step counter - exceptions 0 / 10

main.cpp **Load default template...**

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

/* Define your function here */

int main() {

   /* Type your code here. */

   return 0;
}

```

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) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

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

17.9 LAB: Student info not found

Given a program that searches for a student's ID or name in a text file, complete the FindID() and FindName() functions. Then, insert a try/catch statement in main() to catch any exceptions thrown by FindID() or FindName(), and output the exception message. Each line in the text file contains a name and ID separated by a space.

Function FindID() has two parameters: a student's name (string) and the text file's contents (ifstream). The function FindID() returns the ID associated with the student's name if the name is in the file, otherwise the function throws a runtime_error with the message "Student ID not found for *studentName*", where *studentName* is the name of the student.

Function FindName() has two parameters: a student's ID (string) and the text file's contents (ifstream). The function FindName() returns the name associated with the student's ID if the ID is in the file, otherwise the function throws a runtime_error with the message "Student name not found for *studentID*", where *studentID* is the ID of the student.

The main program takes three inputs from a user: the name of a text file (string), the search option for finding the ID or name of a student (int), and the ID or name of a student (string). If the search option is 0, FindID() is invoked with the student's name as an argument. If the search option is 1, FindName() is invoked with the student's ID as an argument. The main program outputs the search result or the caught exception message.

Ex: If the input of the program is:

```
roster.txt 0 Reagan
```

and the contents of roster.txt are:

```
Reagan rebradshaw835
Ryley rbarber894
Peyton pstott885
Tyrese tmayo945
Caius ccharlton329
```

the output of the program is:

```
rebradshaw835
```

Ex: If the input of the program is:

```
roster.txt 0 Mcauley
```

the program outputs an exception message:

```
Student ID not found for Mcauley
```

Ex: If the input of the program is:

```
roster.txt 1 rebradshaw835
```

the output of the program is:

```
Reagan
```

Ex: If the input of the program is:

```
roster.txt 1 mpreston272
```

the program outputs an exception message:

```
Student name not found for mpreston272
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 17.9.1: LAB: Student info not found 0 / 10

Downloadable files

roster.txt **Download**

main.cpp **Load default template...**

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

string FindID(string name, ifstream &infoFS) {

   /* Type your code here. */

}

string FindName(string ID, ifstream &infoFS) {

   /* Type your code here. */
```

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) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

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

17.10 LAB: Input errors with zyLabs

Write a program that takes in three integers as inputs and outputs the largest value. Use a try block to perform all the statements. Use a catch block to catch any ios_base::failure caused by missing inputs and output the number of inputs read and the largest value. Output "No max" if no inputs are read.

Note: Because inputs are pre-entered when running a program in the zyLabs environment, the system throws the ios_base::failure when inputs are missing. Test the program by running the program in the Develop mode.

Hint: Use a counter to keep track of the number of inputs read and compare the inputs accordingly in the catch block when an exception is caught.

Ex: If the input is:

```
3 7 5
```

the output is:

```
7
```

Ex: If the input is:

```
3
```

the system throws the ios_base::failure and outputs:

```
1 input(s) read:
Max is 3
```

Ex: If no inputs are entered:

```

```

the system throws the ios_base::failure and outputs:

```
0 input(s) read:
No max
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 17.10.1: LAB: Input errors with zyLabs 0 / 10

main.cpp Load default template...

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

int main() {
   cin.exceptions(ios::failbit); // Allow cin to throw exceptions
   int val1;
   int val2;
   int val3;
   int max;

   val1 = 0;
   val2 = 0;
   val3 = 0;

   /* Type your code here. */
```

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) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

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