Module 18: Exceptions

Learning Objectives

1. Write code that uses exception handling.
2. Explain the importance of exception handling.

Process Skills

1. Critical thinking. Use prior knowledge to interpret and explain related concepts.
2. Oral and written communication. Describe a technical concept expressed in a written form.

Please fill in the roles for each member of your team. Take a look at the description of each role to see its responsibilities. If there are only three people in the group, please assign the same person to the **Presenter** and **Reflector** role. It is a good idea to select roles that you have not recently taken.

Team name: _____ Date: _____

Role	Team Member Name
Manager. Keeps track of time and makes sure everyone contributes appropriately.	
Presenter. Talks to the facilitator and other teams.	
Reflector. Considers how the team could work and learn more effectively.	
Recorder. Records all answers and questions and makes the necessary submission.	

For virtual activities: Once you select your roles, change your Zoom name using the format and example below.

Format: *Group X: First name, Last name initial / Role*

Example: *Group 1: Paul I / Presenter*

Model 1. Error handling (3 min) Start time: _____

```
// refrigerator.h
#include <iostream>
#include <map>
class Refrigerator {
 public:
  Refrigerator() : Refrigerator({}, 10) {}
  Refrigerator(const std::map<std::string, int> &contents,
               int capacity)
      : contents_(contents), capacity_(capacity) {}

  int Capacity() const { return capacity_; }
  int TotalContents() const {
    int sum = 0;
    for (std::pair<std::string, int> content : contents_) {
      sum += content.second;
    }
    return sum;
  }

  void Display() {
    for (std::pair<std::string, int> content : contents_) {
      std::cout << content.second << " x " << content.first
                << "\n";
    }
  }

  int Get(const std::string &name, int quantity) {
    if (contents_.count(name) == 0) {
      return 0;
    } else if (contents_.at(name) < quantity) {
      return 0;
    } else {
      contents_.at(name) -= quantity;
      return quantity;
    }
  }

 private:
  std::map<std::string, int> contents_;
  int capacity_;
};
```

```
// main.cc
#include <iostream>
#include "refrigerator.h"

int main() {
 Refrigerator fruits({{"apple", 2}, {"banana", 3}}, 10);
 fruits.Display();

 std::cout << "Getting 3 x apples" << "\n";
 int apple_count = fruits.Get("apple", 3);
 std::cout << "Got " << apple_count << " x apples.\n";

 fruits.Display();

 return 0;
}
```

Screen output:
2 x apple
3 x banana
Getting 3 x apples
Got 0 x apples.
2 x apple
3 x banana

1. What is the value of apple_count right before the program exits (before return 0)?

0

2. Which condition was satisfied when the Get member function was called (`fruits.Get("apple", 3)`)? Place a check (✓) beside your answer.

   a. if (contents_.count(name) == 0)
   b. if (contents_.at(name) < quantity) ✓
   c. else

3. If we only look at the value returned by Get without looking/understanding its member function body, can we identify the reason why it would return 0? Place a check (✓) beside your answer.

   a. Yes
   b. No ✓

4. According to your observations, what is the issue with the sample code? Place a check (✓) beside your answer.

   a. We cannot distinguish when the value returned is valid or if it is an error. ✓
   b. There are no issues.
   c. The function always returns 0.

Model 2. Exception Handling (15 min) Start time: _____

Exception handling is one approach to handling errors. C++ provides some built-in std::exceptions that you can use. You can see them here: https://en.cppreference.com/w/cpp/error/exception. In this example, and in most cases, std::invalid_argument is enough to capture most program errors.

You can create your own exceptions, but this is outside the scope of this class.

```
// modified Get member function in refrigerator.h
int Get(const std::string &name, int quantity) {
  if (contents_.count(name) == 0) {
    throw std::invalid_argument("No " + name +
                                " in the refrigerator.");
  } else if (contents_.at(name) < quantity) {
    throw std::invalid_argument("Not enough " + name +
                                " in the refrigerator.");
  } else {
    contents_.at(name) -= quantity;
    return quantity;
  }
}
```

```
// modified main.cc
#include <iostream>
#include "refrigerator.h"

int main() {
  Refrigerator fruits({{"apple", 2}, {"banana", 3}}, 10);
  fruits.Display();
  std::cout << "Getting 1 x banana and 3 x apples" << "\n";
  try {
    int banana_count = fruits.Get("banana", 1);
    std::cout << "Got " << banana_count << " x banana.\n";
    int apple_count = fruits.Get("apple", 3);
    std::cout << "Got " << apple_count << " x apples.\n";
  } catch(const std::invalid_argument &e) {
    std::cout << e.what() << "\n";
  }
  fruits.Display();
  return 0;
}
```

Screen output:
2 x apple
3 x banana
Getting 1 x banana and 3 x apples
Got 1 x banana.
Not enough apple in the refrigerator.
2 x apple
2 x banana

5. What keyword do we use to tell the compiler that an exception occurred? Place a check (✓) beside your answer.

   a. throw ✓
   b. except
   c. void

6. Analyze the code. When will exceptions be thrown? Place a check (✓) beside your answer.

   a. When we could not retrieve a product using the arguments provided to the function. ✓
   b. When the function returns 0.
   c. When the function/member function does not receive any parameters.

7. What value was stored in banana_count when `fruits.Get("banana", 1)` was called?

1

8. What happens when we call `fruits.Get("apple", 3);`? Place a check (✓) beside your answer.

   a. The member function will throw an exception. ✓
   b. The member function will return 0.
   c. The member function will return 3.

9. After `fruits.Get("banana", 1)` was called the member function exited and performed the next statement that printed "Got 1 x banana". Why do you think the program did not display "Got 3 x apple" after calling `fruits.Get("apple", 3)`? Place a check (✓) beside your answer.

   a. The Get member function returned 0.
   b. The Get member function threw an exception, so it exited out of the try block ✓
   c. The program will crash after calling `fruits.Get("apple", 3)`.

10. Which code block does your group think will execute after `fruits.Get("apple", 3)` threw an exception? Place a check (✓) beside your answer.

    a. try block
    b. catch block ✓
    c. code after the try-catch block

11. The catch block is written like a function definition that accepts parameters and provides a function body. Specifically, this example catches a std::invalid_argument. When `fruits.Get("apple", 3)` was called, what argument was passed to the catch block parameter? Place a check (✓) beside your answer.

    a. std::invalid_argument("No " + name + " in the refrigerator.")
    b. std::invalid_argument("Not enough " + name + " in the refrigerator.") ✓
    c. The catch block did not receive any arguments

12. What code do we place inside the try block? Place a check (✓) beside your answer.

    a. Code that calls a function or member function that may throw an exception. ✓
    b. Code that is called when an exception occurs.
    c. Code that calls a function or member function that does not throw an exception.

13. What code do we place inside the catch block? Place a check (✓) beside your answer.

    a. Code that calls a function or member function that may throw an exception.
    b. Code that is executed when an exception occurs. ✓
    c. Code that calls a function or member function that does not throw an exception.

14. What are the benefits of exception handling? Place a check (✓) beside your answer. Select all that apply.

    a. Separation of error handling code and program functionalities. ✓
    b. Grouping error types (e.g., invalid arguments, out-of-range error, domain error). ✓
    c. Less coding.

15. Complete the Refrigerator's Restock member function. It takes in the name of a product and its quantity. If adding the product will exceed the capacity, throw a std::invalid_argument exception saying, "Adding the product will exceed the refrigerator's capacity". If adding the product with the given quantity does not exceed the refrigerator's capacity, add it to the contents_ map.

```
void Restock(const std::string &name, int quantity) {
  // TODO: provide the condition that checks if adding the quantity
  // will exceed the refrigerator's capacity.
  if (quantity + TotalContents() > capacity_) {

    // TODO: throw a std::invalid_argument exception with
    // the given message
    throw std::invalid_argument(
       "Adding the product will exceed the refrigerator's
capacity");

  } else {
    contents_.at(name) += quantity;
  }
}
```

16. Complete the main function to implement an exception handler. Create a try-catch block that checks for exceptions when the refrigerator is restocked. The catch block should display the error on the screen, and the program should continue waiting for the user's request to restock the refrigerator until they type in q or Q.

```
// main.cc
#include <iostream>
#include "refrigerator.h"

int main() {
  std::string input;
  int quantity = 0;

  Refrigerator fruits({{"apple", 2}, {"banana", 3}}, 10);
  do {
    std::cout << "Product name (enter q to quit): ";
    std::getline(std::cin, input);
    if (input != "Q" && input != "q") {
      std::cout << "Quantity: ";
      std::cin >> quantity;
      std::cin.ignore();
      // TODO: Create a try-catch block to catch exceptions that
      // Restock might throw. Display the error message on the
      // screen if an exception is thrown.


      try {
        fruits.Restock(input, quantity);
      } catch(const std::invalid_argument &e) {
        std::cout << e.what() << "\n";
      }









    }
    fruits.Display();
  } while(input != "Q" && input != "q");

  return 0;
}
```

STOP HERE AND WAIT FOR FURTHER INSTRUCTIONS

Extra challenge (9 min) Start time: _____

We use a more appropriate exception category called std::out_of_range to indicate an error in finding a key from the map. We can catch and handle different exception categories thrown by function calls inside a try block.

```
// modified Get member function in refrigerator.h
 int Get(const std::string &name, int quantity) {
   if (contents_.count(name) == 0) {
     throw std::out_of_range("No " + name +
                             " in the refrigerator.");
   } else if (contents_.at(name) < quantity) {
     throw std::invalid_argument("Not enough " + name +
                                 " in the refrigerator.");
   } else {
     contents_.at(name) -= quantity;
     return quantity;
   }
 }
```

```
// modified main.cc
#include <iostream>
#include "refrigerator.h"
int main() {
 Refrigerator fruits({{"apple", 2}, {"banana", 3}}, 10);
 fruits.Display();
 std::cout << "Getting 1 x banana, 2 x orange and 3 x apples"
           << "\n";
 try {
   int banana_count = fruits.Get("banana", 1);
   std::cout << "Got " << banana_count << " x banana.\n";
   int orange_count = fruits.Get("orange", 2);
   std::cout << "Got " << orange_count << " x orange.\n";
   int apple_count = fruits.Get("apple", 3);
   std::cout << "Got " << apple_count << " x apples.\n";
 } catch(const std::out_of_range &e) {
   std::cout << e.what() << "\n";
 }
 catch(const std::invalid_argument &e) {
   std::cout << e.what() << "\n";
 }
 fruits.Display();
 return 0;
}
```

Screen output:
2 x apple
3 x banana
Getting 1 x banana, 2 x orange and 3 x apples
Got 1 x banana.
No orange in the refrigerator.
2 x apple
2 x banana

17. Which exception category is thrown in the example to indicate the key cannot be found in the contents_ map? Place a check (✓) beside your answer.

    a. std::out_of_range ✓
    b. std::invalid_argument
    c. std::range_error

18. How many catch blocks are associated with the try block in the example?

2

19. Which block will catch the std::out_of_range exception thrown when calling `fruits.Get("orange", 2)`? Place a check (✓) beside your answer.

    a. catch(const std::invalid_argument &e)
    b. catch(const std::out_of_range &e) ✓
    c. It is not caught by any of the catch blocks

20. Both calls to the Get member function (i.e., orange and apple) throw an exception. Why do you think only one of the exception messages was shown on the screen? Place a check (✓) beside your answer.

    a. It is because only the call to Get orange will cause an exception.
    b. It is because only the call to Get apple will cause an exception.
    c. After an exception is thrown, the code inside the appropriate code block is executed, and the remaining code in the try block is skipped. ✓

21. Update the Restock member function so that it only allows the restocking of products already in the refrigerator. The function checks whether the product is in the contents_ map and throws a std::out_of_range exception with the message "Product not found in the refrigerator" if the product is not in the refrigerator.

```
void Restock(const std::string &name, int quantity) {
  // TODO: Check if the name is a key in the contents_ map

  if (contents_.count(name) == 0) {

    // TODO: Throw a std::out_of_range error with the given message
    throw std::out_of_range("Product not found in the
refrigerator");

  }
  if (quantity + TotalContents() > capacity_) {
   throw std::invalid_argument(
     "Adding the product will exceed the refrigerator's capacity");
  } else {
    contents_.at(name) = quantity;
  }
}
```

22. Update the try-catch block in the main function to provide two catch blocks. You already wrote a catch block to catch the std::invalid_argument exception. Create another catch block to catch the std::out_of_range exception and display the error on the screen.

```
// main.cc
#include <iostream>
#include "refrigerator.h"

int main() {
  std::string input;
  int quantity = 0;

  Refrigerator fruits({{"apple", 2}, {"banana", 3}}, 10);
  do {
    std::cout << "Product name (enter q to quit): ";
    std::getline(std::cin, input);
    if (input != "Q" && input != "q") {
      std::cout << "Quantity: ";
      std::cin >> quantity;
      std::cin.ignore();

      // TODO: Add another catch block to handle the
      // std::out_of_range exception
      try {
        fruits.Restock(input, quantity);
      } catch (const std::invalid_argument &e) {
        std::cout << e.what() << "\n";
      } catch (const std::out_of_range &e) {
        std::cout << e.what() << "\n";
      }









    }
    fruits.Display();
  } while(input != "Q" && input != "q");

  return 0;
}
```

Reflector questions

1. What was the most useful thing your team learned during this session?

2. When do you think you would use exceptions?

3. On a scale from 1 to 5, how likely are you to use exceptions for error handling? Select 1 for very likely and 5 for very unlikely.