C++ Programming

Instructor: Rita Kuo
Office: CS 520E
Phone: Ext. 4405
E-mail: rita.kuo@uvu.edu

Exception Handling

Exception Handling

- Exception
  - A program that occurs while a program executes
  - The problem occurs infrequently – if the “rule” is that a statement normally executes correctly
    → the problem represent the “exception to the rule”
- Human Errors
  - Any deviance from “appropriate” behavior
  - Examples:
    - Bad data
    - Buggy code
- Encountering Errors
  - Notify the user of an error
  - Save all work
  - Allow users to gracefully exit the program

Exception Handling

- Exception Handling
  - Enable programmers to create fault-tolerant programs that can resolve (or handle) exceptions
  - Allows a program to continue executing as if no problems were encountered
- Mission of exception handling
  - Transfer control from where the error occurred to an error handler that can deal with the situation
- Types of Errors
  - User input errors: E.g., typos, syntax error in URL, etc.
  - Device error: E.g., the printer may be turned off or run out of paper during printing; a webpage may be temporarily unavailable.
  - Physical limitations: E.g., run out of available memory
  - Code errors: E.g., an invalid array index, pop an empty stack

Dealing with Errors

- Dealing with Errors
  - Return a special error code that the calling method analyzes
    - Example: return a `-1`
    - Not always possible to return an error code
    - There may be no obvious way of distinguish valid and invalid data
    - `-1` might be a perfect result in a method returning an integer
  - Throws an object that encapsulate the error information
    - Java allows every method an alternative exit path if it is unable to complete its task in the normal way
    - The method exits immediately and does not return any value
    - Search for an exception handler that can deal with the particular error condition

Error Handling in C

- `assert` macro
  - Statements used to test assumptions made by programmer
  - Write diagnostic information to the standard error file
- Example
  - https://www.geeksforgeeks.org/assertions-cc/

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

int main()
{
    int x = 7;
    /* Some big code in between and let's say x is accidentally changed to 9 */
    x = 9;
    // Programmer assumes x to be 7 in rest of the code
    assert(x==7);
    /* Rest of the code */
    cout << "" << "end of program" << endl;
    return 0;
}
```

Exception-Handling Constructs

- Exception-handling constructs
  - A **try** block surrounds normal code, which is exited immediately if a throw statement executes
  - A **throw** statement appears within a try block
  - A **catch** clause immediately follows a try block. If the catch was reached due to an exception thrown of the catch clauses' parameter type, the clause executes
    → A catch block is called a **handler** because it handles and exception

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

Exception Basic

- Example: processing an out-of-range subscript

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

int main()
{
  vector<int> integers = {1, 2, 3};

  try {
    cout << "\nAttempt to display integers.at(10)" << endl;
    cout << integers.at(10) << endl;
    cout << "Statements after the error." << endl;
  } catch (out_of_range &ex) {
    cout << "An exception occurred: " << ex.what() << endl;
  }

  cout << "\nEnd of program" << endl;

  return 0;
}
```

Exception Basic

- Example: processing an out-of-range subscript

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

int main()
{
  vector<int> integers = {1, 2, 3};

  try {
    cout << "\nAttempt to display integers.at(10)" << endl;
    cout << integers.at(10) << endl;
    cout << "Statements after the error." << endl;
  } catch (out_of_range &ex) {
    cout << "An exception occurred: " << ex.what() << endl;
  }

  cout << "\nEnd of program" << endl;

  return 0;
}
```

The `try` block contains the code that ***might*** throw an exception

The `catch` block contains the code that ***handles*** the exception if one occurs

Exception Basic

- Example: processing an out-of-range subscript

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

int main()
{
  vector<int> integers = {1, 2, 3};

  try {
    cout << "\nAttempt to display integers.at(10)" << endl;
    cout << integers.at(10) << endl;
    cout << "Statements after the error." << endl;
  } catch (out_of_range &ex) {
    cout << "An exception occurred: " << ex.what() << endl;
  }

  cout << "\nEnd of program" << endl;

  return 0;
}
```

- `vector` member function at generates an exception throws an `out_of_range` exception to notify the program of this program
- The `try` block terminates immediately and the `catch` block begins executing

The `catch` block declares a type (`out_of_range`) and an exception parameter (`ex`) that it receives a reference

Exception Basic

- Example: processing an out-of-range subscript

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

int main()
{
  vector<int> integers = {1, 2, 3};

  try {
    cout << "\nAttempt to display integers.at(10)" << endl;
    cout << integers.at(10) << endl;
    cout << "Statements after the error." << endl;
  } catch (out_of_range &ex) {
    cout << "An exception occurred: " << ex.what() << endl;
  }

  cout << "\nEnd of program" << endl;

  return 0;
}
```

Always catch exceptions by *reference*:

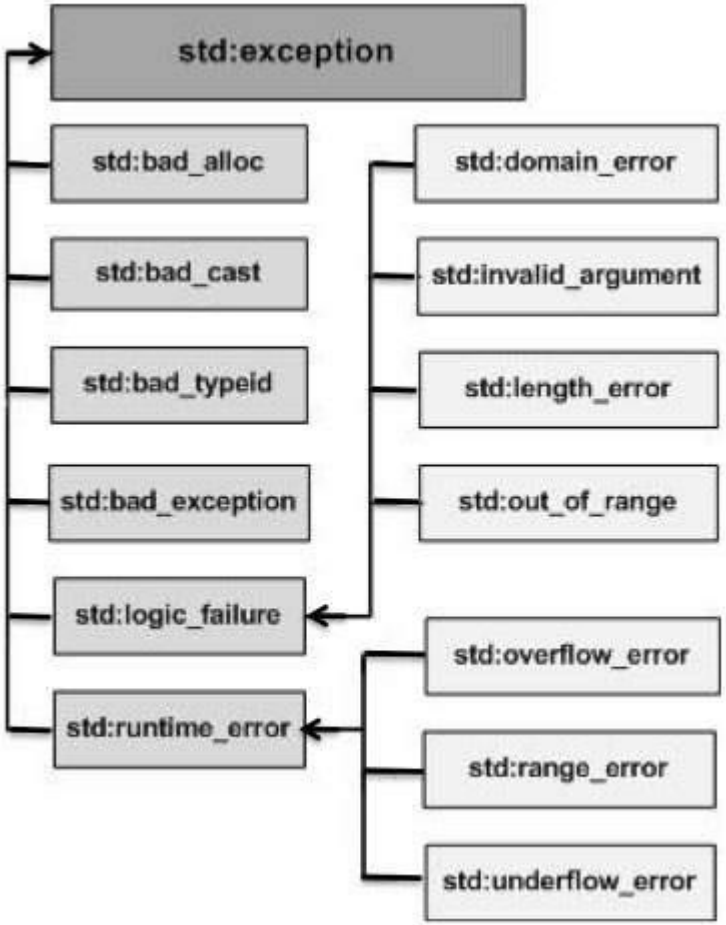
- Exceptions tend to be objects of class type, and it is more efficient to not copy them
- Be able to face of inheritance hierarchies

The exception object's `what` member function gets the error message that is stored in the exception object and display it

https://stackoverflow.com/questions/2522299/c-catch-blocks-catch-exception-by-value-or-reference

C++ Standard Exceptions

- C++ provides a list of standard exceptions defined in `<exception>`



- Detailed definitions of the exceptions can be found at https://cplusplus.com/reference/exception/exception/

https://www.tutorialspoint.com/cplusplus/cpp_exceptions_handling.htm

Throwing Exceptions

- 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 creates 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.

Throwing Exceptions Basic

- Example

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

int main () {
  try {
    throw 20;
  } catch (int e) {
    cout << "An exception occurred. Exception Nr. " << e << '\n';
  }
  return 0;
}
```

https://cplusplus.com/doc/tutorial/exceptions/

Throwing Exceptions Examples

- Example

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

int main()
{
  int month;

  cout << "Enter month (in numbers): ";
  cin >> month;

  try {
    if (month < 1 || month > 12) {
      throw runtime_error("Invalid month");
    }
  } catch (runtime_error &except) {
    cout << except.what() << endl;
    cout << "Invalid input" << endl;
  }

  return 0;
}
```

Throwing Exceptions in Functions

- 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.

```
int AddPositiveIntegers(int a, int b)
{
    if (a < 0 || b < 0)
        throw std::invalid_argument("AddPositiveIntegers arguments must be positive");
    return (a + b);
}
int main()
{
    try {
        cout << AddPositiveIntegers(-1, 2); //exception
    } catch (std::invalid_argument& e) {
        cerr << e.what() << endl;
        return -1;
    }

    return 0;
}
```

https://rollbar.com/blog/error-exceptions-in-c/

Throwing Exceptions in Functions

- 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.

```
int AddPositiveIntegers(int a, int b)
{
    if (a < 0 || b < 0)
        throw std::invalid_argument("AddPositiveIntegers arguments must be positive");
    return (a + b);
}
int main()
{
    try {
        cout << AddPositiveIntegers(-1, 2); //exception
    } catch (std::invalid_argument& e) {
        cerr << e.what() << endl;
        return -1;
    }

    return 0;
}
```

https://rollbar.com/blog/error-exceptions-in-c/

Multiple Handlers

- Multiple handlers may exist
  - Each handler handles different exception types
  - The first matching handler executes; remaining handlers are skipped
- Issues when a derived exception class after the base exception class
  - The exception will be caught by the based class, not the derived class

```
// ... 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
```