Lecture 12.1

Handling Exceptions

What is an Exception?

- Exceptions are errors or anomalies that occur during the **execution** of a program.
- They can be the result of unavailability of system resources, such as memory, file space, channels or from data dependent conditions such as a divide by zero, or numerical overflow.
- Exceptions tend to be rare events but are predicable.

What to do when an exception happens?

- Not handle the exception. Allow the program to die or core dump.
- Issue a warning. Print out some type of error message, probably at the spot in the code where the exception occurred and then exit.
- Handle the exception gracefully and continue executing.

What to do when an exception happens? (cont)

- Do nothing?
  - Not acceptable if you want to remain employed or pass your courses.
- Print an error message?
  - Still not ideal.
  - Most real world programs need to be more robust than this.
- Exceptions need to be handled and corrected. Execution must continue.

How to go about this?

- Suppose an exception occurs in when dividing by 0.
  - Should the function or method that attempted the division be the one to handle it?
  - Can it?
  - Probably some other, higher level, section of code will have the information necessary to decide how to handle the exception.
  - Maybe different programs using the same classes and methods will handle exceptions differently.

Handling Exceptions

- The code that raises exceptions needs to be developed separately from code that handles them.
- If we pass exceptions up to calling routines, it is necessary to have a way to bundle information and for the exception to have some methods to assist in its handling.

Exception Basics

- The section of code that causes or detects a run-time abnormality (divide by zero, out of memory) will "throw" an exception.
- The exception is the object that is thrown. It may be a simple object such as an int, or a class object, including programmer defined class objects.

Exception Basics (cont)

- The exception is "caught" by another section of code.
- The exception object, itself, is used to convey information from the section of code that throws the object to the section of code that catches the object.

Exception Basics (cont2)

- Separation of exception creation and exception handling is very significant.
- Higher level sections of code can better handle exceptions in many cases.
  - If an exception in a library routine. That routine cannot know how to respond in a way that is appropriate for your program.
    - Appropriate response might be to terminate the program
    - Appropriate response might be a warning message
    - maybe the exception can be caught and disregarded.

Exceptions Syntax

- Sections of code that can “throw” exceptions are surrounded in *“try blocks”*.
- Exceptions thrown from within try blocks are “caught” by a *“catch clause”*.

Example 1

```
int main()
{
    int x = 5;
    int y = 0;
    int result;
    int exceptionCode = 25;
    try {
          if (y = = 0) { throw exceptionCode; }
          result = x/y;
    }
    catch (int e) {
          if (e = = 25) { cout << "Divide by zero" << endl; }
          else {cout << "Exception of unknown type" ; }
    }
    cout << "Goodbye" << endl; return 0; }
```


Example 2

```
int main()
{
      int x = 5;
      int y = 0;
      int result;
      char * err_msg = “Division by Zero”;
      try {
            if (y = = 0) { throw err_msg}
            result = x/y;
      }
      catch (char *e) {
          cout << e << endl;
      }
      cout << "Goodbye" << endl; return 0;
}
```

Exception Handling with classes

- In C++ an exception is usually an object
- This allows more information to be bundled into the exception
- Also allows the exception to contain methods to assist the sections of code that handle or process the exception.

Example 1

```
class DivideByZero {
    public:
        DivideByZero (int n, int d) { num = n;
                denom = d ;
                message ="Divide by zero"; }
        int getNumerator() {return num;}
        int getDenominator() {return denom;}
        string getMessage() {return message;}
    private:
        int num;
        int denom;
        string message;
};
```


Example 1 (cont)

```
int main()
{
    int x = 5;
    int y = 0;
    int result;
    try {
        if (y = = 0) { throw DivideByZero ( x, y ) ; }
        result = x/y;
    }
    catch ( DivideByZero e ) {
        cout << e.getMessage() << endl;
        cout << "Numerator: " << e.getNumerator() ;
        cout << "Denominator: " << e.getDenominator(); }
        cout << "Goodbye" << endl;
    return 0;}
```