CS 162
Intro to Programming II
Exceptions
Exceptions

• An exception is a value or an object that indicates that an error has occurred
• When an exception occurs, the program must either terminate or jump to special code for handling the exception.
• The special code for handling the exception is called an exception handler
Exceptions – Key Words

• **throw** – followed by an argument, is used to signal an exception

• **try** – followed by a block `{ }`, is used to invoke code that throws an exception

• **catch** – followed by a block `{ }`, is used to process exceptions thrown in a preceding **try** block. It takes a parameter that matches the type of exception thrown
Throwing an Exception

• Code that detects the exception must pass information to the exception handler. This is done using a `throw` statement:

```c
throw "Emergency!"
throw 12;
```

• In C++, information thrown by the `throw` statement may be a value of any type.
Catching an Exception

• Block of code that handles the exception is said to catch the exception and is called an exception handler

• An exception handler is written to catch exceptions of a given type: For example, the code

  ```cpp
  catch(char *str)
  {
      cout << str;
  }
  ```

  can only catch exceptions of type C-string
Catching an Exception

Another example of a handler:

```cpp
catch(int x)
{
    cerr << "Error: " << x;
}
```

This can catch exceptions of type `int`
Connecting to the Handler

Every catch block is attached to a `try` block of code and is responsible for handling exceptions thrown from that block.

```java
try {
    // This code handles exceptions
    // of type char that are thrown
    // in this block
}
catch(char e1)
{
    // This code handles exceptions
    // of type char that are thrown
    // in this block
}
```
Execution of Catch Blocks

- The catch block syntax is similar to that of a function.
- A catch block has a formal parameter that is initialized to the value of the thrown exception before the block is executed.
Exception Example

- An example of exception handling is code that computes the square root of a number.
- It throws an exception in the form of a C-string if the user enters a negative number.
```cpp
int main()
{
  try
  {
    double x;
    cout << "Enter a number: ";
    cin >> x;
    if (x < 0) throw "Bad argument!";
    cout << "Square root of " << x << " is " << sqrt(x);
  }
  catch(char *str)
  {
    cout << str;
  }
  return 0;
}
```
Flow of Control

1. Computer encounters a `throw` statement in a `try` block
2. The computer evaluates the `throw` expression, and immediately exits the `try` block
3. The computer selects an attached `catch` block that matches the type of the thrown value, places the value in the catch block’s formal parameter, and executes the catch block
Uncaught Exception

• An exception may be uncaught if
  – there is no `catch` block with a data type that matches the exception that was thrown, or
  – it was not thrown from within a `try` block
• The program will terminate in either case
Handling Multiple Exceptions

Multiple catch blocks can be attached to the same block of code. The catch blocks should handle exceptions of different types

```java
try{...}
catch(int iEx){ }
catch(char *strEx){ }
catch(double dEx){ }
```
Throwing an Exception Class

• An exception class can be defined and thrown
• A catch block must be designed to catch an object of the exception class
• The exception class object can pass data to exception handler via data members
Exception When Calling `new`

- If `new` cannot allocate memory, it throws an exception of type `bad_alloc`
- Must `#include <new>` to use `bad_alloc`
- Can invoke `new` from within a `try` block, and use a `catch` block to detect that memory was not allocated.
Nested Exception Handling

*try* blocks can be nested in other *try* blocks and even in catch blocks

```java
try {
    try{
    } catch(int i){ }
}
catch(char *s)
{  }
```
Where to Find an Exception Handler?

- The compiler looks for a suitable handler attached to an enclosing `try` block in the same function.
- If there is no matching handler in the function, it terminates execution of the function, and continues the search for a handler starting at the point of the call in the calling function.
Unwinding the Stack

• An unhandled exception propagates backwards into the calling function and appears to be thrown at the point of the call.

• The computer will keep terminating function calls and tracing backwards along the call chain until it finds an enclosing try block with a matching handler, or until the exception propagates out of main (terminating the program).

• This process is called unwinding the call stack.
Rethrowing an Exception

• Sometimes an exception handler may need to do some tasks, then pass the exception to a handler in the calling environment.

• The statement

  throw;

  with no parameters can be used within a catch block to pass the exception to a handler in the outer block.