1) C/C++ like most programming languages provide a try-catch mechanism to aid in programming for error conditions. An exception is a value or an object that signals an error. The function detecting an error condition throws an exception and passes an argument describing the error. The function throwing the exception is terminated (i.e., its call-frame is popped off the top of the run-time stack). Call-frames are continued to be popped until a corresponding catch with an ExceptionParameter matching the type of the throw argument is found. If no matching catch is found, the program is terminated.

The general syntax of a try-catch construct is:

```c++
try {
    // code that calls functions or object member functions
    // that might throw an exception
} catch(ExceptionParameter) {
    // code here to handle the exception
}
``` // Repeat as many catch blocks as needed for each type of exception expected

When used with classes, special purpose exception classes can be used. Consider the following Rectangle class with exceptions to handle negative width or negative length.

```c++
// This program demonstrates Rectangle class exceptions.
#include <iostream>
#include "Rectangle.h"
using namespace std;

int main() {
    int width;
    int length;

    // Create a Rectangle object.
    Rectangle myRectangle;

    // Get the width and length.
    cout << "Enter the rectangle's width: ";
    cin >> width;
    cout << "Enter the rectangle's length: ";
    cin >> length;

    // Store these values in the Rectangle object.
    try {
        myRectangle.setWidth(width);
        myRectangle.setLength(length);
        cout << "The area of the rectangle is " << myRectangle.getArea() << endl;
    }
    catch (Rectangle::NegativeWidth e) {
        cout << "Error: " << e.getValue() << " is an invalid value for the rectangle's width.\n";
    }
    catch (Rectangle::NegativeLength e) {
        cout << "Error: " << e.getValue() << " is an invalid value for the rectangle's length.\n";
    }

    cout << "End of the program.\n";
    return 0;
}
```

a) How would we modify the code to repeat until non-negative values for width and length are entered?
b) Why are the exception classes NegativeLength and NegativeWidth public?
c) What member functions detect error conditions?

d) What information is returned when a negative width is detected?
**Big-oh Definition** - asymptotic upper bound
For a given complexity function $f(n)$, $O(f(n))$ is the set of complexity functions $g(n)$ for which there exists some positive real constant $c$ and some nonnegative integer $N$ such that for all $n \geq N$,

$$g(n) \leq c \times f(n).$$

**Execution Time**

| Problem size, n |
|-----------------
| $g(n)$ is execution time of algorithm |
| $c \times f(n)$ |

$T(n) = c_1 + c_2 n = 100 + 10 n$ is $O(n)$.

"Proof": Pick $c = 110$ and $N = 1$, then $100 + 10 n \leq 110 n$ for all $n \geq 1$.

- $100 + 10 n \leq 110 n$
- $100 \leq 100 n$
- $1 \leq n$

**Problem with big-oh:**

If $T(n)$ is $O(n)$, then it is also $O(n^2)$, $O(n^3)$, $O(n^5)$, $O(2^n)$, .... since these are also upper bounds.

**Omega Definition** - asymptotic lower bound
For a given complexity function $f(n)$, $\Omega(f(n))$ is the set of complexity functions $g(n)$ for which there exists some positive real constant $c$ and some nonnegative integer $N$ such that for all $n \geq N$,

$$g(n) \geq c \times f(n).$$

**Execution Time**

| Problem size, n |
|-----------------
| $g(n)$ is execution time of algorithm |
| $c \times f(n)$ |
T(n) = c₁ + c₂ n = 100 + 10 n is $\Omega(n)$.

"Proof": We need to find a c and N so that the definition is satisfied, i.e., $100 + 10 n \geq c n$ for all $n \geq N$.

What c and N will work?

**Theta Definition** - asymptotic upper and lower bound, i.e., a "tight" bound or "best" big-oh

For a given complexity function $f(n)$, $\theta(f(n))$ is the set of complexity functions $g(n)$ for which there exists some positive real constants $c$ and $d$ and some nonnegative integer $N$ such that for all $n \geq N$, $c \times f(n) \leq g(n) \leq d \times f(n)$.

2) Suppose that you have an $\theta(n^2)$ algorithm that required 10 seconds to run on a problem size of 1000. How long would you expect the algorithm to run on a problem size of 10,000?