Chapter 3 – Control Statements (if, if..else)

Let's go back to explain a few things. Just as English ends a sentence with a period, '.', we need to end statements with a semi-colon, ';'. We can't use a period to end a statement since we use them in decimal values. So an assignment is a statement; and calling a method is a statement. We can use compound statements, such as assigning a variable from the result of a method. Methods have a basic syntax:

```
returnType methodName(paramType paramName, paramType paramName, …) {statements}
```

Executing statements one after another has limited use. To do more complex programs, we need to be able to make decisions and execute different code.

if statement syntax (for Java, Objective-C, and C++):

```
if (condition)
        statement;
```

A *condition* is a relational statement that evaluates to *true* or *false*. The *if* statement executes the *statement* if the condition is *true*. An example of how to use this is the following (% operator calculates the remainder, *print* is pseudocode for whatever is the appropriate output method for your language):

```
if (number % 2 != 0)            // notice there is no ';' on this line
        print "number is odd";
```

You've probably heard the term *else* used with *if*; this conditional statement is also available on computers:

```
if (condition)
        statement1;
else
        statement2;
```

So if the condition evaluates to *true*, *statement1* will execute. If the condition is *false*, *statement2* will execute. Building off the previous example:

```
if (number % 2 != 0)            // notice there is no ';'
        print "number is odd";
else                            // notice there is no ';'
        print "number is even";
```

Exercise 3.1: Write a program that accepts integers until a sentinel (you choose one) is entered and print out whether the number is odd or even. Hint: use the previous example programs and the pseudocode above.

If you need to do something more complicated that requires more than one statement, you can use a **block**. Variables created in a block are local to that block, so they get deleted when the program counter leaves the block. The compiler will treat everything in a block as one statement. For example:

```
if (number % 2 != 0 && number > 0) {
        print "number is odd";
        print "number is positive";
}                                       // notice there is no ';'
```

To generate conditions, we use the following relational operators:

!=	not equal	>=	greater than or equal
==	equals	<	less than
>	greater than	<=	less than or equal

A common error with creating relational operators is using '=', but this is an assignment operator, not a comparison; so be careful with this.

Here are some examples using pseudocode:

```
if (age >= 65)
       print "Person is a senior and can retire.";
else if (age <= 18)
       print "Person is not an adult and should stay in school.";

if (number == 12)
       print "There is a dozen";
else if (number < 12)
       print "There is less than a dozen";
else
       print "There is more than a dozen";
```

You should always list test numbers to verify every condition. For example, use 12, 18, 35, 65, and 72 to test the above *age* condition. Use 8, 12, and 20 to test the above *number* condition.

Exercise 3.2: Write a programs that accept integers until a sentinel is entered and do the above age pseudocode conditions.

Exercise 3.3: Write a programs that accept integers until a sentinel is entered and do the above dozen pseudocode conditions.

Exercise 3.4: Write a programs that accept integers for a Celcius temperature until a sentinel is entered and create conditions so that it prints "that is the temperature for ice; that is the temperature for water; or that is the temperature for steam."

Exercise 3.5: Write a programs that accept integers for a Fahrenheit temperature until a sentinel is entered and create conditions so that it prints "that is the temperature for ice; that is the temperature for water; or that is the temperature for steam."

Exercise 3.6: Write a programs that accepts an integer for a speed limit. Then inside a loop, let the user enter individual speeds until a sentinel is entered. Create conditions so that it prints "at or under speed limit," "normal speeding ticket" if it is only 30 km/h or less above the speed limit, or "excessive speeding ticket" if it is above 30 km/h above the speed limit.
Sample printout:

```
Enter speed limit (km/h):  50
Enter speed (0 to end):  48
    At or under the speed limit
Enter speed (0 to end):  90
    Excessive speeding ticket
Enter speed (0 to end):  60
    Normal speeding ticket
Enter speed (0 to end):  0
```

For more advanced conditions, you can perform multiple *if* 's or you can use logical connectors and operators:

&&	and	connector (both conditions must be true)
||	or	connector (only one condition must be true)
!	not	operator (flips true to false and false to true)

The *and* operator is used when multiple conditions are necessary. The *or* operator is used when only one condition is necessary. For example:

```
if (average >= 89.5 && workHabit != 'N' && mark1 != 'I' && mark2 != 'I' && ..)
       award = HonourRollWithDistinction;
else if (average >= 85.5 && workHabit != 'N' && mark1 != 'I' && mark2 != 'I' && ..)
       award = HonourRoll;
else if (average >= 79.5 && workHabit != 'N' && mark1 != 'I' && mark2 != 'I' && ..)
       award = HonourableMention;

if (topMark(Math) || topMark(English) || topMark(Physics) || topMark(History) || ..)
       award = SubjectAward;
if (!(topMark(Physics) || topMark(Chemistry) || topMark(Biology))
       award = NoScienceAward;
if (!topMark(Physics) && !topMark(Chemistry) && !topMark(Biology))
       award = NoScienceAward;
```

See Example_3_1 for a running example. A condition can be partially or fully calculated and stored in a boolean data type:

```
boolean      for Java
bool         for C++
BOOL         for Objective-C
```

Exercise 3.7: Modify Example_3_2 to enter 3 values until a sentinel is entered. Then use the 'max' method as a template to create the 'min' and 'median' methods. Add the methods to the printout. Include comments in the source file of a sample of test cases to sufficiently test your methods, eg. (10, 15, 8), (8, 15, 10), ... – there are 6 of them! Sample printout:

```
Enter first number (0 to quit): 10
Enter second number: 15
Enter third number: 8
max(10, 15, 8) = 15
min(10, 15, 8) = 8
median(10, 15, 8) = 10
Enter first number (0 to quit): 0
```

Exercise 3.8: Modify Exercise 3.7 to enter 3 values until a sentinel is entered. Then use the methods to determine if the 3 values form a right triangle. Sample printout:

```
Enter first number (0 to quit): 4
Enter second number: 5
Enter third number: 3
The hypotenuse is 5.
(3, 4, 5) form a right triangle
Enter first number (0 to quit): 4
Enter second number: 3
Enter third number: 6
The hypotenuse is 6.
(3, 4, 6) do not form a right triangle
Enter first number (0 to quit): 0
```

Exercise 3.9: Modify Example_3_2 to enter 3 values (down, yds to first down, yds to goal) for football until a sentinel is entered. Create at least 3 reasonable logic statements to determine if it should be a run or pass play. Sample printout:

```
Enter down (0 to quit): 2
Enter yards to first down: 15
Enter yards to goal: 40
It should be a pass play.
Enter down (0 to quit): 2
Enter yards to first down: 1
Enter yards to goal: 1
It should be a run play.
Enter down (0 to quit): 0
```

The reason we need to include header or import files is that the libraries are very large and therefore use a lot of identifiers. To prevent duplicate identifiers, we need to purposely bring in the desired libraries. The below section describes what is needed for the different languages. We will cover the libraries on an as-used basis.

Standard Libraries (Java)

With Java, you don't need to include any libraries to print to the console, but you do need to include libraries to get input:

```
import java.util.Scanner;
```

Standard Libraries (Objective-C)

For Objective-C, you need the following header file to use the console library for printing and input:

```
#import <Foundation/Foundation.h>
```

Standard Libraries (C++)

With C++, you have 2 choices of libraries, but they are very similar. You can choose to use "*printf*" or "*cout*". *cout* is C++ whereas *printf* is originally from "C". I prefer to use *printf* because it gives me more control over how the output looks and is thread-safe. It's also easier to read the format, whereas you would have to look-up the code to know how *cout* is going to format. So for C, you need:

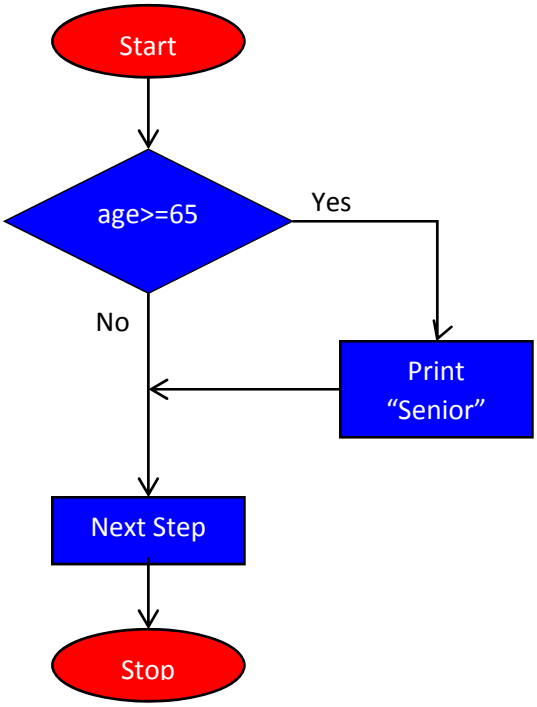
```
#include <stdio.h>
#include <tchar.h>
```

If you want to use C++, you need:

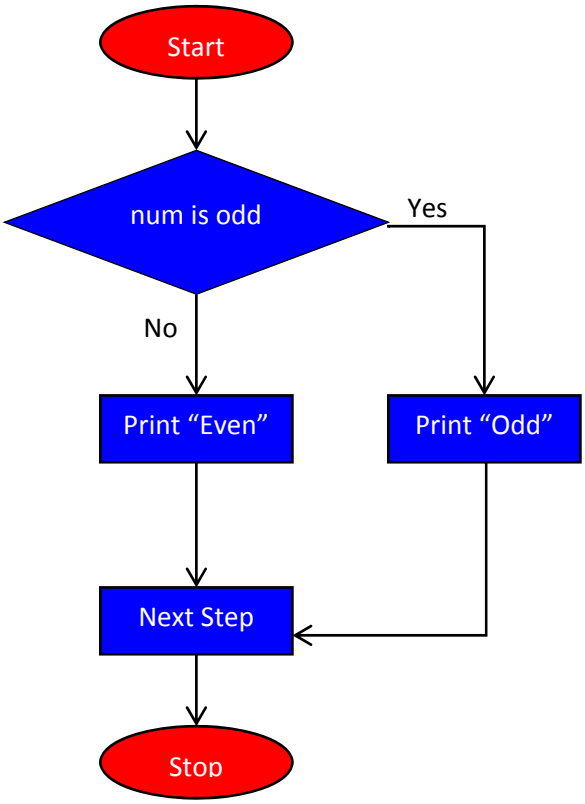
```
#include <iostream>
#include <string>
using namespace std;
```

We can flowchart *if* and *if..else* statements. These examples demonstrate that flowcharts are more useful to see how the code works. You can quickly sketch these out; it doesn't have to be precisely drawn to be effective.

if:



if..else:



Email me the exercise source files (not the whole folder) when you are done.