IF and ELSE Statements

http://people.sc.fsu.edu/~jburkardt/isc/week03_lecture_06.pdf

.....

ISC3313:
Introduction to Scientific Computing with C++
Summer Semester 2011

.....

John Burkardt
Department of Scientific Computing
Florida State University

Last Modified: 25 May 2011

IF and ELSE

- **Introduction**
- The IF and ELSE Statements
- Examples
- The Popcorn Iteration
- Programming Assignment #2

INTRO: IF and ELSE

We have already seen some examples of **if** and **else** statements.

They check a logical condition, and then control whether a set of executable statements are to be carried out.

Today we will look more carefully at the rules for using **if** and **else** statements, the ways they can be put together, and some examples of programs in which they are useful.

Our programming homework will require you to use **if** and **else** statements inside a **while** statement, in order to carry out a calculation.

INTRO: Schedule

Read:

- Today's class covers information in Sections 3.5, 3.6, 3.8, 3.9

Next Class:

- The FOR Statement

Assignment:

- Programming Assignment #1 is due today.
- Programming Assignment #2, given today, is due on Thursday, June 2.

IF and ELSE

- Introduction
- **The IF and ELSE Statements**
- Examples
- The Popcorn Iteration
- Programming Assignment #2

IF/ELSE: Logical Comparisons

Given numbers **x** and **y**, C++ can check these conditions:

`if ( x == y )...`	*if x is equal to y then... (TWO equal signs!!)*
`if ( x != y )...`	*if x is not equal to y then...*
`if ( x < y )...`	*if x is less than y then...*
`if ( x <= y )...`	*if x is less than or equal to y then...*
`if ( x > y )...`	*if x is greater than y then...*
`if ( x >= y )...`	*if x is greater than or equal to y then...*

Checking these logical conditions allows our program to make decisions.

IF/ELSE: The AND Logical Operator

Suppose we want to do something if one condition AND another condition are both true?

This could happen, for instance, if we need the number **x** to be in the interval [0,1]. Then we want **x** to be greater than or equal to 0, and less than or equal to 1, in other words

`if ( 0 <= x ) AND ( x <= 1 )` *then do something*

C++ uses the && operator to express the idea of AND:

```
if ( ( 0 <= x ) && ( x <= 1 ) ) ...
```

IF/ELSE: The OR Logical Operator

Similarly, we might want to do something if either one OR another condition is true.

For instance, we can sleep in late if it's Saturday OR Sunday. And a number is not in [0,1] if it is less than 0, or it is greater than 1:

`if ( today is Saturday ) OR ( today is Sunday )` *stay in bed*
`if ( x < 0 ) OR ( 1 < x )` *then do something*

C++ uses the || operator to express the idea of OR:

```
if ( ( x < 0 ) || ( 1 < x ) ) ...
```

IF/ELSE: if (condition) statement

An **if** statement allows us to check a condition first, and then to carry out one or more statements only if the condition is true.

```
if ( condition )
{
   statement 1;    <-- Only if condition is true
   statement 2;
   ...
}
```

IF/ELSE: Combined if/else statement

A combined **if/else** statement lets us do one thing if a condition is true, and something else otherwise.

```
if ( condition )
{
   statement 1 for true case;
   statement 2 for true case
   ...
}
else      <-- what to do if condition is false
{
   statement 1 for false case;
   statement 2 for false case
   ...
}
```

IF/ELSIF/ELSE: Combined if/else statement

Sometimes, we have many choices of actions, and we can describe our choice by checking one condition, then another, and so on, until we find a match. Here is a simple version with three choices:

```
if ( condition #1 )
{
   statements if condition #1 true;
}
else if ( condition #2 )
{
   statements if condition #1 is false, and condition #1 is true;
}
else
{
   statements if conditions #1 and #2 are false;
}
```

IF and ELSE

- Introduction
- The IF and ELSE Statements
- **Examples**
- The Popcorn Iteration
- Programming Assignment #2

EXAMPLES: Guarding Against Illegal Input

Some mathematical functions won't work for certain input quantities.

- We can't divide by zero;
- We can't take the square root of a negative number;
- We can't take the logarithm of a number that isn't positive;

If we are worried about encountering illegal input, we can write statements such as:

```
if ( 0.0 < x )
{
  y = log ( x );  <-- #include <cmath>; to use log()!
}
else
{
  y = 0.0;  <-- Give y a default value.
}
```

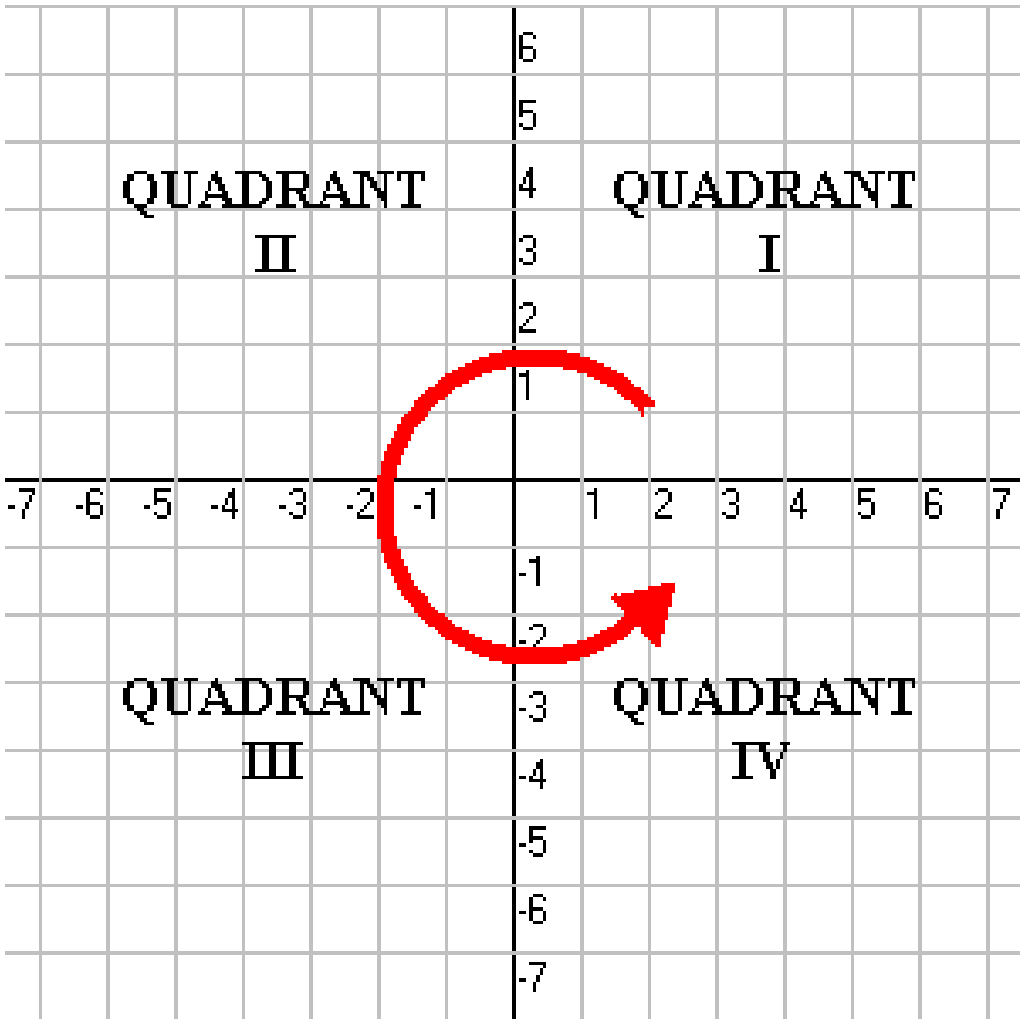
EXAMPLES: Does interval contain point?

In INTERVAL.CPP, we determine where a point **x** is relative to the interval **[a,b]**:

```
if ( x < a )
{
  cout << "X is to the left of A\n";
}
else if ( x <= b )
{
  cout << "A <= X <= B.\n";
}
else
{
  cout << "X is to the right of B.\n";
}
```

EXAMPLES: Quadrant of a 2D Point

Can we determine the quadrant of a 2D point (x,y)?



EXAMPLES: Quadrant of a 2D Point

In QUADRANT.CPP, we use nested **if** statements:

```
if ( 0.0 <= x )
{
  if ( 0.0 <= y )
  {
      cout << "(X,Y) is in quadrant I.\n";
  }
  else
  {
    cout << "(X,Y) is in quadrant IV.\n";
  }
}
else
{
  if ( 0.0 <= y )
  {
      cout << "(X,Y) is in quadrant II.\n";
  }
  else
  {
    cout << "(X,Y) is in quadrant III.\n";
  }
}
```

EXAMPLES: Quadrant of a 2D Point

We could, instead, use the **&&** operator, which means AND:

```
if ( 0.0 <= x && 0.0 <= y )
{
    cout << "(X,Y) is in quadrant I.\n";
}
else if ( x < 0.0 && 0.0 <= y )
{
    cout << "(X,Y) is in quadrant II.\n";
}
else if ( x < 0.0 && y < 0.0 )
{
   cout << "(X,Y) is in quadrant III.\n";
}
else if ( 0.0 <= x && y < 0.0 )
{
    cout << "(X,Y) is in quadrant IV.\n";
}
```

EXAMPLES: Letter Grades

The book discusses the example of letter grades:

```
if ( 90 <= average )
{
  cout << "Your grade is A\n";
}
else if ( 80 <= average )
{
  cout << "Your grade is B\n";
}
else if ( 70 <= average )
{
  cout << "Your grade is C\n";
}
else if ( 60 <= average )
{
  cout << "Your grade is D\n";
}
else
{
  cout << "Your grade is F\n";
}
```

Why don't we have to write lines like:

```
else if ( 80 <= average && average < 90 ) ?
```

EXAMPLES: February

How would we assign the lengths of months in days?

To say **OR** in C++ we use the || operator:

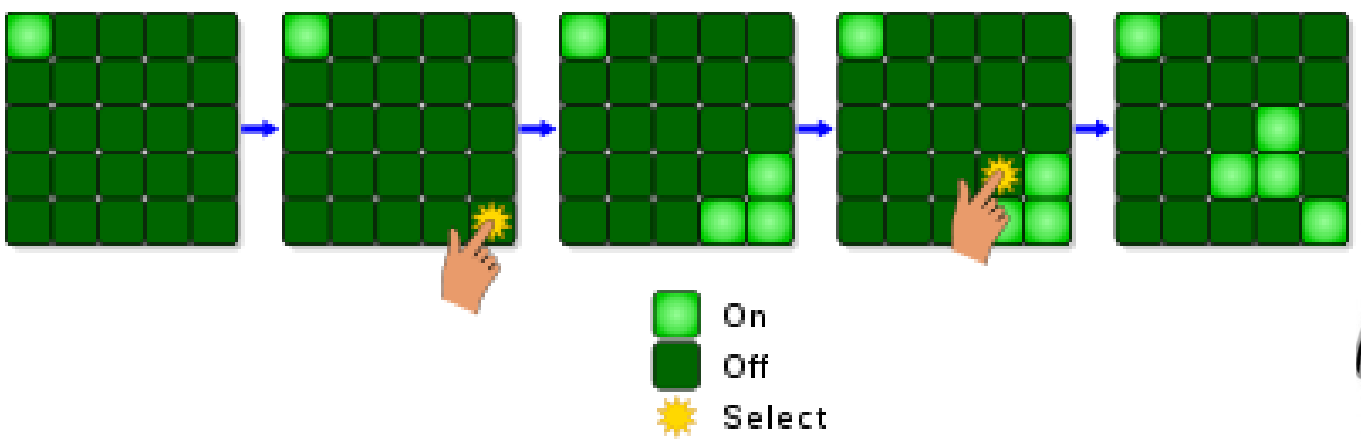
```
int month;                  <-- contains the index of a month...

if ( month ==  1 ||         <-- January
     month ==  3 ||         <-- March
     month ==  5 ||         <-- May
     month ==  7 ||         <-- July
     month ==  8 ||         <-- August
     month == 10 ||         <-- October
     month == 12 )          <-- December
{
  days = 31;
}
else if ( month ==  4 ||    <-- April
          month ==  6 ||    <-- June
          month ==  9 ||    <-- September
          month == 11 )     <-- November
{
  days = 30;
}
else                        <-- February
{
  days = 28;                <-- Isn't the correct answer here more complicated?
}
```

EXAMPLES: Lights Out!

There was a children's game called "Lights Out!" by Tiger Toys, which featured a 5 by 5 grid of squares.

When you turned the game on, some squares were randomly lit up. Your task was to turn all the lights out. You could push any square. Doing so would switch the square (on to off, or off to on). However, it did the same thing to the four immediate neighbors (left/right/top/bottom). So as you were making some squares go dark, others would light up.



EXAMPLES: Lights Out!

You may have seen a table of numbers represented by a matrix. We could represent the lights out board by a 5x5 array of numbers, called **a**, which are 0 or 1. The last screen on the previous slide would be:

```
1 0 0 0 0
0 0 0 0 0
0 0 0 1 0
0 0 1 1 0
0 0 0 0 1
```

If the user pushed the button in row 3, column 4, the value we need to change could be written **a(3,4)**.

(This isn't exactly what C++ does; we'll see the right way soon.)

EXAMPLES: Lights Out!

If **a(3,4)** is a 0 (off) it should become a 1 (on), and vice versa. So one way to write this change would be:

```
if ( a(3,4) == 0 )
{
  a(3,4) = 1;
}
else
{
  a(3,4) = 0;
}
```

But actually, there's a quicker way to do the same thing (assuming the value is always 0 or 1):

```
a(3,4) = 1 - a(3,4);
```

EXAMPLES: Lights Out!

And actually, it's good that we can switch a value with one line, because actually, we have to switch the neighbors as well!

If we are near one of the edges, a neighbor might not exist. That will be obvious, because one of its indices will be 0 or 6, outside the legal range of 1 to 5.

So, assuming the user pushed button (i,j), here is the code to change all the lights:

```
                a(i,  j  ) = 1 - a(i,  j  );
if ( 1 < i ) a(i-1,j  ) = 1 - a(i-1,j  );
if ( i < 5 ) a(i+1,j  ) = 1 - a(i+1,j  );
if ( 1 < j ) a(i,  j-1) = 1 - a(i,  j-1);
if ( j < 5 ) a(i,  j+1) = 1 - a(i,  j+1);
```

As I said, C++ actually refers to tables in a slightly different way, which we will learn about in time.

EXAMPLES: Lights Out!

The program **lights_out.cpp** demonstrates a version of the Lights Out game.

The program is written in C++, but in order to do graphics, it is complicated. However, at the very end of it, you can find where the buttons get adjusted, if you want to peek.

To compile the program, type:

```
g++ lights_out.cpp -lglut      <-- -lglut adds the "glut"
                                   graphics library
```

Then you can run it just like any of your own programs.

IF and ELSE

- Introduction
- The IF and ELSE Statements
- Examples
- **The Popcorn Iteration**
- Programming Assignment #2

POPCORN: An Iteration is a Sequence of Steps

An iteration is a procedure in which a single step is repeated many times.

Thus, the iteration can be complete described by:

1. initializing the data
2. describing how each step of the iteration modifies the data
3. deciding when the iteration is to stop

Our program to estimate square roots was one example of an iteration.

POPCORN: The Rules for Popcorn

The popcorn iteration works with positive integers.

1. the user initializes the value **n**;
2. if **n** is even, we divide it by 2, but
   if **n** is odd, we multiply by 3 and add 1;
3. if **n** is 1, the iteration stops;

A typical iteration proceeds like this:

```
 6  -->  3  (n/2)
 3  --> 10  (3*n+1)
10  -->  5  (n/2)
 5  --> 16  (3*n+1)
16  -->  8  (n/2)
 8  -->  4  (n/2)
 4  -->  2  (n/2)
 2  -->  1  stop!
```

POPCORN: The Rules for Popcorn

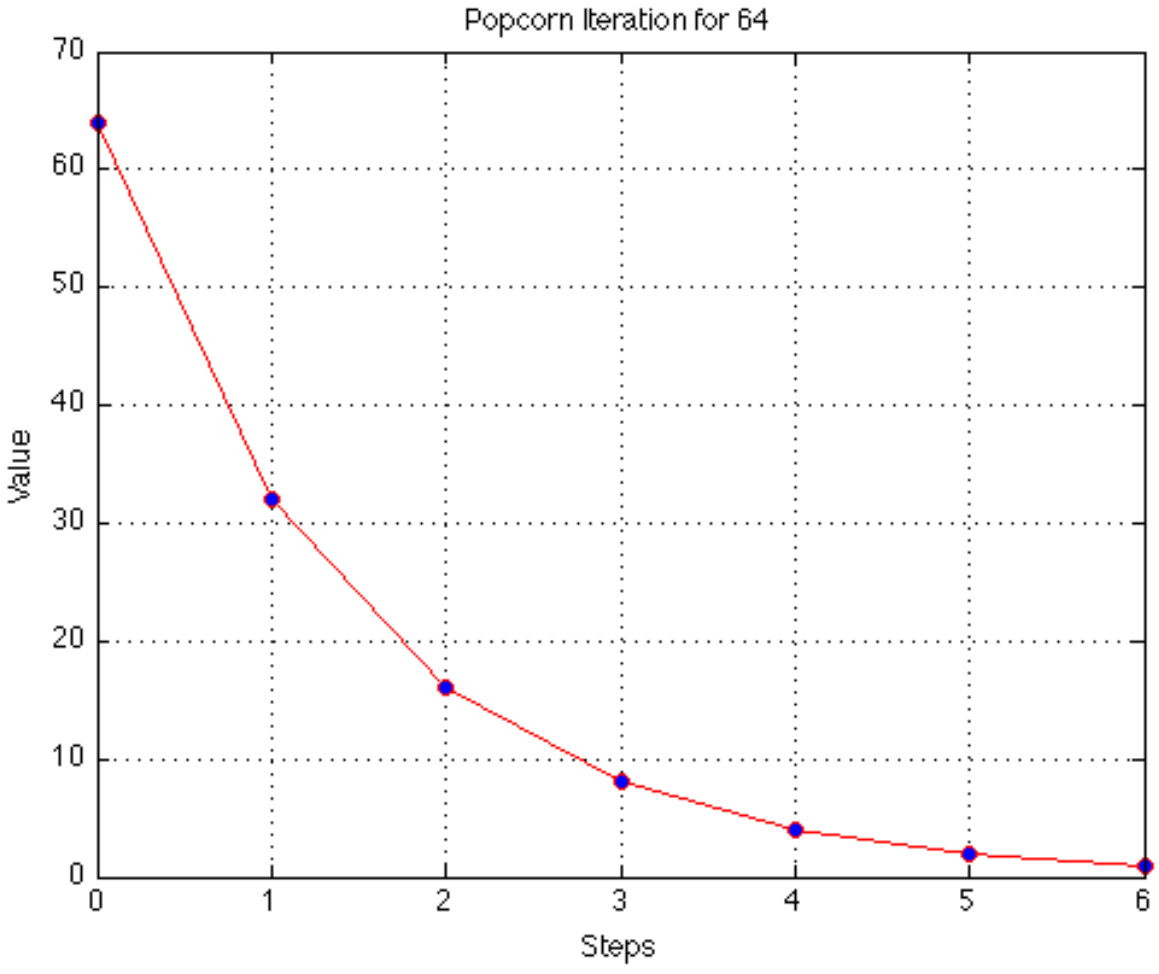
Since a step of the iteration can make the input number decrease or increase, it's hard to tell what will happen in general.

There could be some starting numbers for which the iteration goes into some kind of loop.

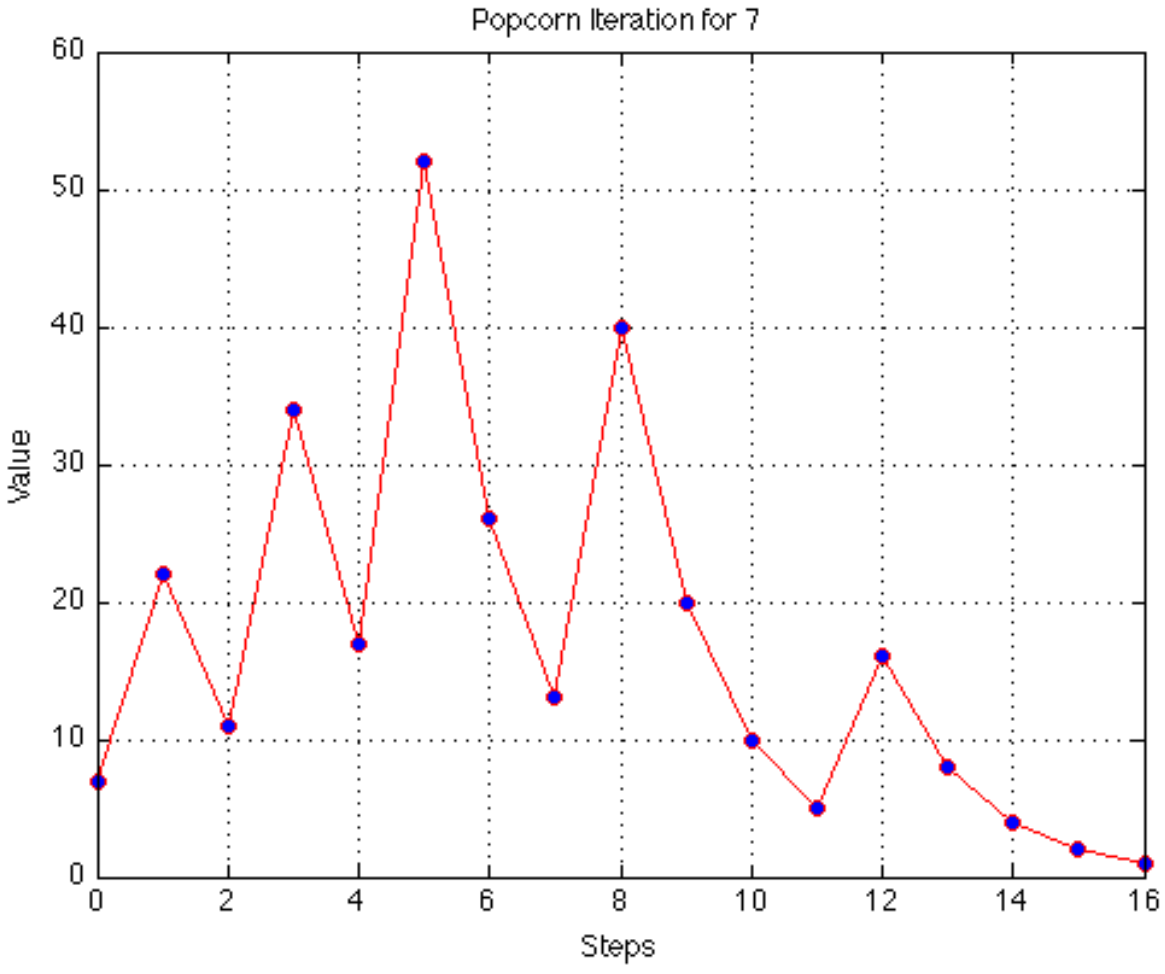
Perhaps other starting values become larger and larger.

However, there is at least one large set of numbers that will all end up at 1, for sure.

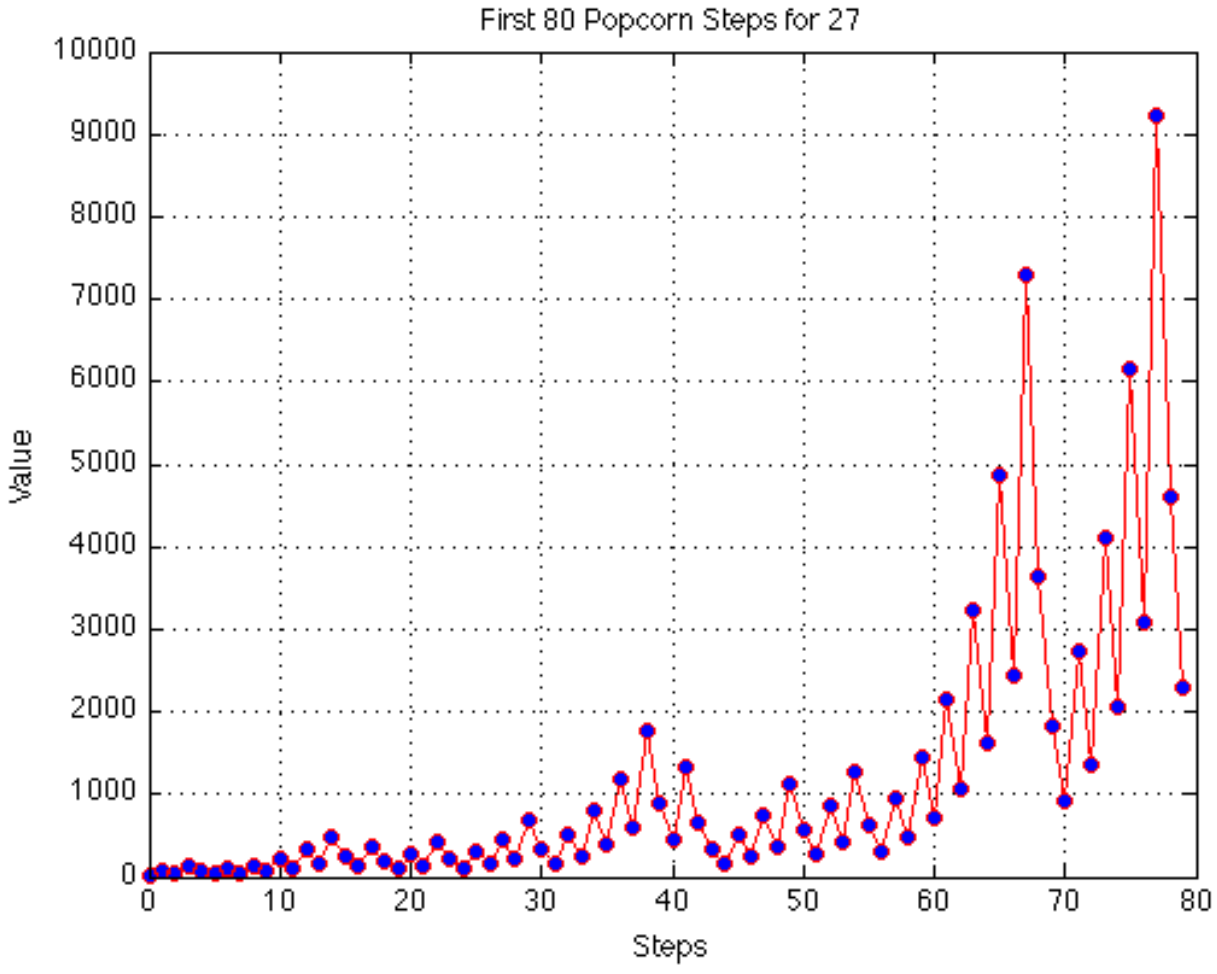
POPCORN: Iteration for 64



POPCORN: Iteration for 7



POPCORN: Iteration for 27 (first 80 steps)



IF and ELSE

- Introduction
- The IF and ELSE Statements
- Examples
- The Popcorn Iteration
- **Programming Assignment #2**

ASSIGNMENT #2: Instructions

Write a C++ program for the popcorn iteration.

The user inputs the value of **n**.

The program sets the step counter to 0.

The program prints the step counter and **n**.

Begin a **while** loop, repeating until the value **n** = 1 is reached.

Inside the while loop, apply the popcorn formula for odd or even **n**.
An integer **n** is:

- even if **(n % 2)** is 0.
- odd if **(n % 2)** is 1.

Increase the step counter.

Print the step counter and **n**.

Prepare to loop again.

ASSIGNMENT #2: Required Output

This means that if the user types in 6, your program should compute and print the following:

```
0   6   <-- Print step 0 and value!
1   3   <-- Print step 1 and value.
2  10   <--  ... and so on...
3   5
4  16
5   8
6   4
7   2
8   1   <-- Print last step and value.
```

ASSIGNMENT #2: Program Outline

```
# include <stuff>
using namespace std;
int main ()
{
  declarations (all ints!)
  get value of n;
  initializations;
  print step, n;
  while ( n is not 1 )
  {
    apply one popcorn step;
    print step, n;
  }
  return 0;
}
```

ASSIGNMENT #2: Plotting Your Results

If your program only writes the step numbers and values, you can save the output to a file, perhaps called "mydata.txt", and then make a plot of your data using gnuplot:

```
gnuplot
  set grid
  plot "mydata.txt" with lines
  quit
```

(You are not required to plot your results as part of this assignment!)

ASSIGNMENT #2: Data

You should run your program for the input values 30 and 49.

Email to Detelina:

- your results for n=30;
- your results for n=49
- a copy of your program.

The program and output are due by Thursday, June 2.