FOR Loops

http://people.sc.fsu.edu/∼jburkardt/isc/week04
lecture_08.pdf

....

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

....

John Burkardt
Department of Scientific Computing
Florida State University

Last Modified: 01 June 2011

FOR Loops

- **Introduction**
- The Increment Operator
- The FOR Statement
- Examples
- The BREAK Statement
- When does COS(X)=X?
- Homework Program #3

INTRO: Schedule

Read:

- Today's class covers information in Sections 4.1, 4.2, 4.3, 4.4 and 4.7

Next Class:

- The SWITCH Statement, the Continue Statement, Logical Variables, Data Conversion

Assignment:

- Programming Assignment #2 is due today.
- Programming Assignment #3 will be due June 8.

INTRO:

Today we find out who put the ++ in C++, that is, we talk about the increment operator.

We need to be comfortable with the increment operator because it is a standard abbreviation used in **for** loops, a big topic for today.

Although it's more complicated and powerful, we will start thinking about the **for** statement as a simplified version of a **while** loop using a counter.

This simple version of **for** simply repeats the statements a given number of times. We will discover that the **break** and **continue** statements help us make simple modifications to this behavior

FOR Loops

- Introduction
- **The Increment Operator**
- The FOR Statement
- Examples
- The BREAK Statement
- When does COS(X)=X?
- Homework Program #3

INCREMENT: The “Plus Plus” Operator

Our next topic is the **increment operator**, or ++.

In a sense, the increment operator is not necessary. Anything you can do with it can be done in a simpler way. But there are good reasons to be familiar with it:

- Everyone uses ++ in **for** loops (today’s main topic);
- The ++ operator can be a nicer, shorter way to program;
- Most programs you see will include various forms of ++. Even if you don’t use it, you should recognize it;
- This is where C++ gets its name!

INCREMENT: Abbreviation for “Get the Next One”

One of the most common steps in programming can be described as “*get the next one*”.

Computing the Fibonacci or popcorn sequence, we updated **step**:

```
  step = step + 1;
```

We counted how may times we bet Bill Gates:

```
  tries = tries + 1;
```

Sometimes, the next item was *smaller*, rather than bigger. For the truck problem, we kept taking one more brick off the load:

```
  bricks = bricks - 1;
```

and in **divisible.cpp**, we reduced the test divisor:

```
  test = test - 1;
```

INCREMENT: Using ++ and −−

The operating of increasing a variable by 1 is so frequent that C++ has an abbreviation that can be used for it.

Writing **i++** means increase **i** by 1; **i−−** decreases it instead. We can replace:

```
step = step + 1;       by    step++;
tries = tries + 1;     by    tries++;
bricks = bricks - 1;   by    bricks--;
test = test - 1;       by    test--;
```

C++ also defines special symbols **++i** and −−**i** which mean almost the same thing, but not quite. We will completely igno these two symbols! Ask about them in your next C++ class!

INCREMENT: More Abbreviations

Moreover, C++ allows you to replace many operations with a similar kind of abbreviation:

```
x = x + y;        by      x += y;
a = a * 2;        by      a *= 2;
c = c / d;        by      c /= d;
e = e - f;        by      e -= f;
```

All of these abbreviations work because the original text read:

`x = x` *operation* `y;`

which is understood to be replaced by:

`x` *operation*= `y;`

where **y** is a number, a variable, or a formula.

INCREMENT: Example

gates_plusplus.cpp:

```
srand ( time ( 0 ) );
won = 0;
bet = 1000;
tries = 1;
coin = rand ( );

while ( coin % 2 == 0 )
{
  won -= bet;             \\ won = won - bet;
  bet *= 2;               \\ bet = 2 * bet;
  tries++;                \\ tries = tries + 1;
  coin = rand ( );
}
won += bet;               \\ won = won + bet;
```

INCREMENT: i++ and i−− are Important

What's important (you will see these everywhere, and you will use them):

```
i++;
j--;
```

What's not important (you don't need to learn these well, but you should have an idea of what is going on):

```
++i;
--j;
a += b;
c -= d;
e /= f;
g *= h;
```

They will turn up in C++ programs written by other people (and maybe by you, too.)

FOR Loops

- Introduction
- The Increment Operator
- **The FOR Statement**
- Examples
- The BREAK Statement
- When does COS(X)=X?
- Homework Program #3

FOR: Related to the WHILE Statement

So far, we have concentrated on using the **while** statement for looping. Many of our examples included a counter variable, which we often called **step**, had the form:

```
step = 0;                <-- initialize counter
while ( step < limit )   <-- check counter
{
  statements;
  step = step + 1;       <-- increment counter
}
```

In fact, using a counter to control a loop is so common that the **fo** statement was set up to put all these pieces together in one pla

FOR: Definition, Relation to WHILE

The C++ **for** statement has the following form:

```
for ( initialization; continue condition; increment )
{
   statements to be repeated;
}
```

The previous **while** loop becomes a special **for** loop:

```
for ( step = 0; step < limit; step = step + 1 )
{
   statements to be repeated;
}
```

or even:

```
for ( step = 0; step < limit; step++ )
{
   statements to be repeated;
}
```

FOR: Comments

The counter variable is usually an **int**. (But sometimes a **float** makes sense, as in our integral approximation homework.)

The counter can have any initial value, although 0 or 1 is common.

If we think of the **for** loop in terms of a counter, it always repeats the statements a certain number of times. (But the **for** loop is actually more flexible.)

It is not obvious how to "jump out" of the loop early, as in the truck problem where we were unloading bricks. (But the **break** statement will solve this problem.)

It is not obvious how to make a **for** loop run until "as long as necessary", that is, until we win $1,000 from Bill Gates. (But we'll see the loop condition can be used for exactly this purpose.)

FOR: Adding 100 Integers, Making a Trig Table

sum_100.cpp:

```
sum = 0;
for ( i = 100; i < 200; i++ )
{
  sum = sum + i;
}
cout << "Sum of 100 to 199 is " << sum << "\n";
```

trig_table.cpp:

```
pi = 3.14159265;
for ( i = 0; i <= 90; i = i + 5 )
{
  angle = pi * ( i / 180.0 );  <-- Convert to radians
  cout << "  " << i << "  " << angle
       << "  " << sin ( angle )
       << "  " << cos ( angle ) << "\n";
}
```

FOR Loops

- Introduction
- The Increment Operator
- The FOR Statement
- **Examples**
- The BREAK Statement
- When does COS(X)=X?
- Homework Program #3

EXAMPLES: What Numbers Divide 12?

twelve.cpp:

```
for ( i = 1; i <= 12; i++ )
{
  if ( ( 12 % i ) == 0 )
  {
    cout << "12 is divisible by " << i << "\n";
  }
}
```

Note that variables **i**, **j** and **k** are very frequently used as counter variables for **for** loops.

EXAMPLES: Increments Don't Have to be +1

leapyear.cpp:

```
for ( y = 1904; y <= 2000; y = y + 4 )
{
  cout << "The year " << y
       << " was a 20th century leap year.\n";
}
```

truck_for.cpp

```
for ( bricks = 1000; 0 <= bricks; bricks-- )
{
  if ( truck + 5 * bricks <= limit )
  {
    cout << "Using " << bricks << " bricks will work.\n";
```
(...But now we'd like to stop searching!)
```
  }
}
```

EXAMPLES: A Pair of Nested Loops for Tables or Grids

checkerboard.cpp:

```
for ( i = 1; i <= 8; i++ )
{
  for ( j = 1; j <= 8; j++ )
  {
    cout << "  (" << i << "," << j << ")";
  }
  cout << "\n";
}
```

Nested loops can be used to handle checkerboards, grids, data [illegible] two indexes, tables, and so on.

EXAMPLES: Another Double Indexed Set

deck.cpp:

```
for ( j = 1; j <= 4; j++ )
{
  for ( i = 1; i <= 13; i++ )
  {
         if ( i ==  1 ) cout << "A"
    else if ( i == 11 ) cout << "J";
    else if ( i == 12 ) cout << "Q";
    else if ( i == 13 ) cout << "K"
    else                cout << i;

         if ( j == 1 ) cout << "H\n";
    else if ( j == 2 ) cout << "C\n";
    else if ( j == 3 ) cout << "D\n";
    else               cout << "S\n";
  }
}
```

(I've had to squeeze this code to fit it on one slide.)
We'll see better ways of handling cards later!

FOR Loops

- Introduction
- The Increment Operator
- The FOR Statement
- Examples
- **The BREAK Statement**
- When does COS(X)=X?
- Homework Program #3

BREAK: Early Termination

In the simple versions of the **for** loop that we consider, the set of statements are always carried out a given number of times.

Sometimes, our strategy for solving a problem needs to be more complicated. We might want to do something at most so many times, but we might hope to be done earlier than that. Or, we might want to do something a certain number of times, but there might be unpredictable warnings that tell us we must quit early.

C++ provides the **break** statement, which indicates that the current loop should be terminated. This works for the **for** and **while** and **do...while** statements, and will also be useful in th **switch** statement that we will see later.

BREAK: Use in a FOR Loop

A typical way the **break** statement is used might be:

```
for ( i = 0; i < n; i++ )
{
  statements;

  if ( condition )
  {
    cout << "Surprise!  Bad (or good) thing happened!\n"
    break;
  }

  more statements;
}
```

BREAK: Use in a WHILE Loop

And the break can be used in a **while** or **do...while** as well:

```
while ( condition )
{
   statements;

   if ( some other condition )
   {
      cout << "Surprise!  Bad (or good) thing happened!\n"
      break;
   }

   more statements;
}
```

BREAK: The Weight Limit Problem

truck_for.cpp

```
limit = 3000;  truck = 2000;  solution = -1;

for ( bricks = 1000; 0 <= bricks; bricks-- )
{
  if ( truck + 5 * bricks <= limit )
  {
    solution = bricks;
    break;      <-- Leave the for loop now!
  }
}
...  <-- Come here after break or normal exit.
if ( solution != -1 )
{
  cout << "We can carry " << solution << " bricks\n";
}
```

BREAK: The Arrow Problem

For the weight limit problem, the only reason we wanted to stop the loop was because we got what we wanted.

However, we might also need to stop a loop because we realize it doesn't make sense to go further.

If an arrow starts at (0,0), with horizontal and vertical velocities **vx** and **vy**, a formula for its position after **t** seconds might be:

$$x = vx * t;$$
$$y = vy * t - 16 * t * t;$$

We could track this position with a **for** loop for time t = 0, 1 ..., but we would want to stop the loop if **y** becomes negativ

BREAK: The Arrow Problem

arrow.cpp

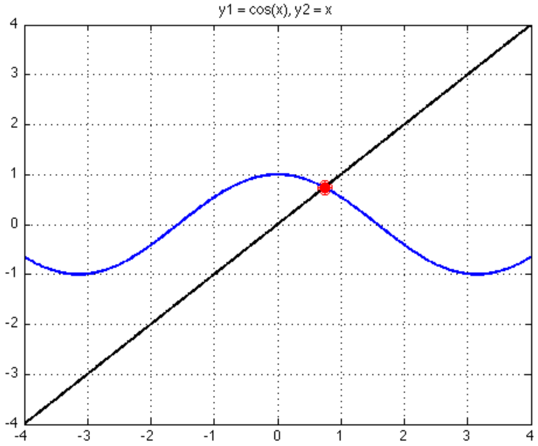
```
vx = 100;
vy = 100;
for ( t = 0; t <= 20; t++ )
{
  x = vx * t;
  y = vy * t - 16 * t * t;
  cout << t << "   " << x << "   " << y << "\n";

  if ( y <= 0.0 && 0.0 < t )    <-- Why do we check t?
  {
    cout << "The arrow landed by " << t << "seconds.\n";
    break;
  }
}
```

FOR Loops

- Introduction
- The Increment Operator
- The FOR Statement
- Examples
- The BREAK and CONTINUE Statements
- **When does COS(X)=X?**
- Homework Program #3

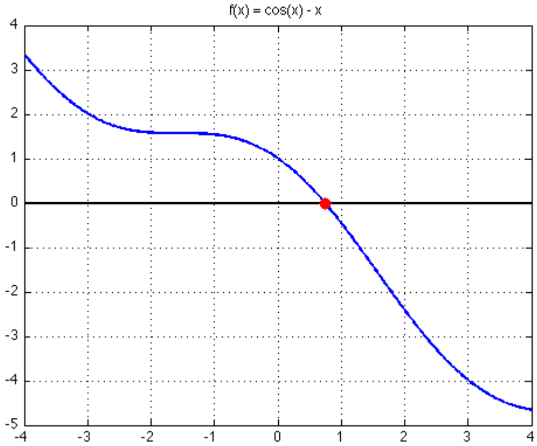
COS(X)=X: Plot COS(X) and X

Between 0 and 1, there is a value of **x** for which **cos(x)=x**:



COS(X)=X: Plot F(X)=COS(X)-X

If we plot, **f(x)=cos(x)-x**, we seek the place where the function changes from positive to negative:



COS(X)=X: Plot F(X)=COS(X)-X

Suppose I was able to compute a table of the values of **cos(x)-x** between 0 and 1:

```
 X   Cos(X)  Cos(X)-X
---  ----     ----
0.0  1.00     1.00
0.1  0.99     0.89
0.2  0.98     0.78
0.3  0.95     0.65
0.4  0.92     0.52
0.5  0.87     0.37
0.6  0.82     0.22
0.7  0.76     0.06  <-- last positive value
0.8  0.69    -0.10  <-- first negative value
0.9  0.62    -0.27
1.0  0.54    -0.45
```

From this information, I know two things:

- the answer, to one decimal place, is **x**=0.7...
- the answer is between 0.7 and 0.8;

COS(X)=X: Plot F(X)=COS(X)-X

If I can make a table from 0 to 1, I can just as easily make a table from 0.7 to 0.8. It might look like this:

```
  X    Cos(X)  Cos(X)-X
 ---   ----     ----
 0.70  0.76     0.06
 0.71  0.75     0.04
 0.72  0.75     0.03
 0.73  0.74     0.02
 0.74  0.73    -0.01
 0.75  0.73    -0.02
 0.76  0.72    -0.03
 0.77  0.71    -0.05
 0.78  0.71    -0.06
 0.79  0.70    -0.08
 0.80  0.69    -0.10
```

What two things does this table tell us?

What do we do if we want more accuracy?

FOR Loops

- Introduction
- The Increment Operator
- The FOR Statement
- Examples
- The BREAK and CONTINUE Statements
- When does COS(X)=X?
- **Homework Program #3**

ASSIGNMENT #3: Find X so that COS(X)=X

Write a C++ program which will help us search for the value of **x** for which **cos(x)=x**.

The program should be *interactive*. It should allow the user to enter two numbers, **xlo** and **xhi**.

The program should evaluate and print **x**, **cos(x)** and **cos(x)** - **x** at 11 equally spaced points from **xlo** and **xhi**.

This means that, if **xlo**= 0 and **xhi**= 1, your program should print something like this:

```
0.0  1.00   1.00
0.1  0.99   0.89
0.2  0.98   0.78
0.3  0.95   0.65
0.4  0.92   0.52
0.5  0.87   0.37
0.6  0.82   0.22
0.7  0.76   0.06  <-- last positive value
0.8  0.69  -0.10  <-- first negative value
0.9  0.62  -0.27
1.0  0.54  -0.45
```

ASSIGNMENT #3: Steps to the Answer

From the output, the value **x** we are seeking must be between 0.7, where **cos(x)=x** is positive, and 0.8, where it becomes negative.

So we know the first decimal place of the answer, **x = 0.7...** and that we should run the program again, this time with **xlo** $= 0.7$ and **xhi** $= 0.8$.

This will give us the second digit of the answer. Suppose we see that the change occurs between 0.77 and 0.78. If that is the answer (it isn't!) then we would next call the program with **xlo** $=$ 0.77 and **xhi** $= 0.78$.

Each time we call the program, we will be able to figure one digit of accuracy in our answer.

ASSIGNMENT #3: Program Outline

I suggest you use a **for** loop to evaluate the function between **xlo** and **xhi**, perhaps something like this:

```
for ( x = xlo; x <= xhi; x = x + dx )
{
  cout << "  " << x
       << "  " << cos ( x )
       << "  " << cos ( x ) - x << "\n";
}
```

- Since you are calling the **cos()** function, what extra **include** statement do you need?
- In order to increment **x** in the **for** loop, we use the variable **dx**. We want to take exactly 10 steps from **xlo** to **xhi**. How do we compute **dx**?

ASSIGNMENT #3: Details of Assignment

Start your program with the input values **xlo** $= 0$ and **xhi** $= 1$.

Use the results from your program to determine, one by one, at least 4 or 5 digits of the answer. (At some point, you won't be able to continue because you don't print enough digits, or you are using **float** arithmetic, which only has 6 or 7 digits of accuracy.)

Send to Detelina, by email or printout:

- your program's estimate of the value **x** for which **x = cos(x)**;
- a copy of your program.

The program and output are due by Thursday, June 9.