Loops! Loops! Loops!

Lecture 10
COP 3014 Spring 2017

January 31, 2017

Repetition Statements

- Repetition statements are called loops, and are used to repeat the same code mulitple times in succession.
- The number of repetitions is based on criteria defined in the loop structure, usually a true/false expression
- The three loop structures in C++ are:
  - **while** loops
  - **do-while** loops
  - **for** loops
- Three types of loops are not actually needed, but having the different forms is convenient

while and do-while loops

- Format of while loop:

```
while (expression)
    statement
```

- Format of do/while loop:

```
do
    statement
while (expression);
```

- The *expression* in these formats is handled the same as in the `if/else` statements discussed previously (0 means false, anything else means true)
- The "statement" portion is also as in `if/else`. It can be a single statement or a compound statement (a block { }).

while and do-while loops

We could also write the formats as follows (illustrating more visually what they look like when a compound statement makes up the loop "body"):

```
while (boolean_expression)
{
    statement1;
    statement2;
    // ...
    statementN;
}
```

while and do-while loops

The same can be done for the do-while loop:

```
do
{
      statement1;
      statement2;
      // ...
      statementN;
} while (boolean_expression);
```

How they work

- The expression is a test condition that is evaluated to decide whether the loop should repeat or not.
  - **true** means run the loop body again.
  - **false** means quit.
- The `while` and `do/while` loops both follow the same basic flowchart – the only exception is that:
  - In a `while` loop, the expression is tested first
  - In a `do/while` loop, the loop "body" is executed first

Examples

Both of these examples add all the numbers from 1 through 50.

```
// while loop example
//loop runs 50 times, condition checked 51 times
int i = 1, sum = 0;
while (i <= 50)
{
     sum += i;      // means:  sum = sum + i
     i++;           // means:  i = i + 1
 }

cout <<"Sum of numbers from 1 through 50 is "
 <<sum <<endl;
```

Examples

```
// do-while loop example
//loop runs 50 times, condition checked 50 times
int i = 1, sum = 0;
do
{
     sum += i;      // means:  sum = sum + i
     i++;           // means:  i = i + 1
 }while (i <= 50);

cout <<"Sum of numbers from 1 through 50 is "
 <<sum <<endl;
```

The for loop

- The `for` loop is most convenient with *counting* loops – i.e. loops that are based on a counting variable, usually a known number of iterations
- Format of for loop:

```
for (initialCondition; boolean_Expression;
                          iterativeStatement)
     statement;
```

- Remember that the `statement` can be a single statement or a block, so an alternate format might be:

```
for (initialCondition; boolean_Expression;
                          iterativeStatement)
 {
     statement1;
     statement2;
     // ...
     statementN;
}
```

How it works

- The *initialCondition* runs once, at the start of the loop
- The *testExpression* is checked. (This is just like the expression in a `while` loop). If it's false, quit. If it's true, then:
  - Run the loop body
  - Run the *iterativeStatement*
  - Go back to the *testExpression* step and repeat

Example:

```
//loop runs 50 times, condition checked 51 times
int i, sum = 0;
for (i = 1; i <= 50; i++)
 {
     sum += i;
 }
cout <<"Sum of numbers from 1 through 50 is "
 <<sum <<endl;
```

More examples

- This loop prints “Hello” 10 times.

```
for (int i = 0; i <10; i++)
     cout <<"Hello" <<endl ;
```

- Loops can also be nested. This prints a rectangle

```
for (int i = 0; i <10; i++)
{
     for (int j = 0; j <15; j++)
     {
          cout <<"*";
     }
     cout <<endl;
}
```

Some notes on the for loop

It should be noted that if the control variable is declared inside the for header, it only has scope through the for loop's execution.

Once the loop is finished, the variable is out of scope:

```
for (int counter = 0; counter <10; counter++)
{
     // loop body
}

cout <<counter;
          // illegal.  counter out of scope
```

Some Notes on the for loop

This can be avoided by declaring the control variable before the loop itself.

```
int counter; // declaration of control variable

for (counter = 0; counter <10; counter++)
{
     // loop body
}

cout <<counter;
          // OK. counter is in scope
```

Some Notes on the for loop

For loops also do not have to count one-by-one, or even upward. Examples:

```
for (i = 100; i >0; i--)

for (c = 3; c <= 30; c+=4)
```

The first example gives a loop header that starts counting at 100 and decrements its control variable, counting down to 1 (and quitting when i reaches 0).

The second example shows a loop that begins counting at 3 and counts by 4's (the second value of c will be 7, etc).

break and continue

- These statements can be used to alter the flow of control in loops, although they are not specifically *needed*. (Any loop can be made to exit by writing an appropriate *test expression*).
- **break**: This causes immediate exit from any loop (as well as from switch blocks).
- **continue**: When used in a loop, this statement causes the current loop iteration to end, but the loop then moves on to the next step.
  - In a while or do-while loop, the rest of the loop body is skipped, and execution moves on to the *test condition*.
  - In a for loop, the rest of the loop body is skipped, and execution moves on to the *iterative statement*.