ITM 312, Fall 2013

Chapter 5 Lecture Notes

5.1. The Increment and Decrement Operators

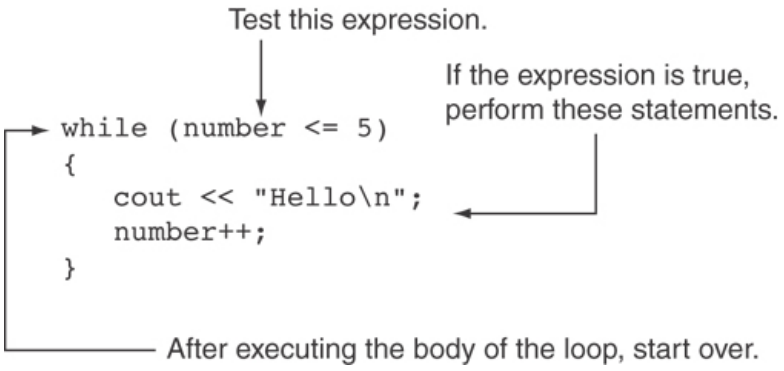
a. ++ is increment operator
   i. Adds 1 to a variable
b. -- is decrement operator
   i. Subtracts 1 to a variable
c. Prefix vs. Postfix
   i. Prefix: returns the current variable, then changes
   ii. Postfix: changes, then returns the changed value

5.2. Introduction to Loops: The `while` Loop

a. Loop – a control structure that causes a statement(s) to repeat
b. Syntax of while loop:

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

   i. `statement;` can also be a block of statements enclosed in { }
c. How it works:
   i. Expression is evaluated
   ii. If true, then execute statement and return to step 1 (eval. expr.)
   iii. If false, then loop is finished and statements after while loop `statement;` execute
d. Example:



e. Infinite Loops
   i. Loop must contain code to make the expression false
   ii. Otherwise, will continue until computer crashes (dangerous!)

5.3. Using the `while` Loop for Input Validation

a. Input validation – inspect data that is given to the program and determine if it is valid

b. While loops can be used in user input to reject invalid data, and keep requesting data until valid data is entered

c. How it works:

   i. Read input

   ii. While input is invalid:

      1. Display error message

      2. Read input again and proceed to (ii)

   iii. Valid data! Proceed to rest of program

d. Example:

```
cout << "Enter a number less than 10: ";
cin >> number;
while (number >= 10)
{
   cout << "Invalid Entry! Enter a number less than 10: ";
   cin >> number;
}
```

5.4. Counters

a. Counter – a variable that is incremented or decremented each time a loop repeats

b. Can be used to control execution of the loop (a.k.a. *loop control variable*)

c. Must be initialized before entering loop

d. Example:

```
int foo = 1;
while (foo<10) foo++;
```

   i. foo is the counter/loop control variable

5.5. The `do-while` Loop

a. Similar to while loop, except executes statement(s) and then tests expression

   i. Loop always executes once

   ii. Useful in menu-driven programs

b. How it works:

i. Statements inside `do` execute

ii. Expression is checked

1. If true, return to `do`
2. If false, stop repeating and continue with rest of program

c. Example:

```
int x = 1;
do
{
    cout << x << endl;
} while(x < 0);
```

i. Note semicolon after expression -> `while (x < 0);`

5.6. The `for` Loop

a. Useful for counter-controlled loop (you know exactly how many times the loop should execute)

b. Syntax:

```
for (initialization; test; update)
    statement; // or block in { }
```

i. No semicolon after the `update` expression or after `)`

c. How it works:

i. Perform `initialization`

ii. Evaluate `test` expression (the `for` loop is a pretest loop)

1. If false, execute `statement`
2. If false, terminate loop execution

iii. Execute `update`, then re-evaluate `test` expression

d. Example:

```
int count;
for (count = 1; count <= 5; count++)
    cout << "Hello" << endl;
```

e. When to use: any situation that clearly requires...

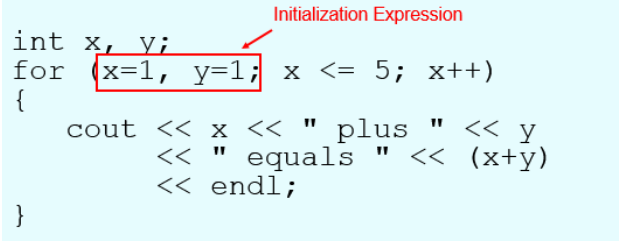
i. An initialization

ii. A false condition to stop the loop

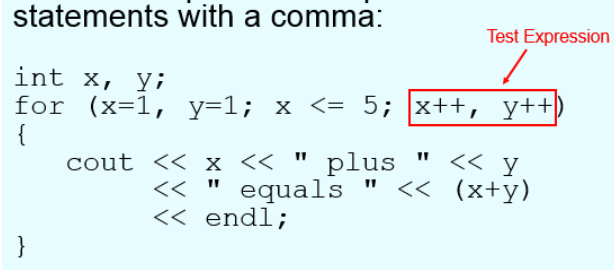
iii. An update to occur at the end of each iteration

iv. Modifications

You can have multiple statements in the *initialization* expression. Separate the statements with a comma:



You can also have multiple statements in the *test* expression. Separate the statements with a comma:



You can omit the *initialization* expression if it has already been done:

```
int sum = 0, num = 1;
for (; num <= 10; num++)
     sum += num;
```

You can declare variables in the *initialization* expression:

```
int sum = 0;
for (int num = 0; num <= 10;
num++)
      sum += num;
```

The scope of the variable `num` is the `for` loop.

5.7. Keeping a Running Total

a. Running total – accumulated sum of numbers from each repetition of loop

b. Accumulator – variable that holds running total

c. Example:

```
int sum=0, num=1; // sum is the
while (num <= 10) // accumulator
```

```
{
    sum += num;
    num++;
}
cout << "Sum of numbers 1 - 10 is" << sum << endl;
```

5.8. Sentinels

a. sentinel – value in a list of values that indicates end of data
b. Special value that cannot be confused with a valid value, e.g. -999 for a test score
c. Used to terminate input when user may not know how many values will be entered

5.9. Deciding Which Loop to Use

a. The while loop is a conditional pretest loop
    i. Iterates as long as a certain condition exits
    ii. Validating input
    iii. Reading lists of data terminated by a sentinel
b. The do-while loop is a conditional posttest loop
    i. Always iterates at least once
    ii. Repeating a menu
c. The for loop is a pretest loop
    i. Built-in expressions for initializing, testing, and updating
    ii. Situations where the exact number of iterations is known

5.10. Nested Loops

a. nested loop – a loop inside the body of another loop
b. Example of inner (inside) and outer (outside) loops:

```
for (row=1; row<=3; row++)  //outer
    for (col=1; col<=3; col++)//inner
        cout << row * col << endl;
```

c. Inner loop goes through all repetitions for each repetition of outer loop
d. Inner loop repetitions complete sooner than outer loop
e. Total number of repetitions for inner loop is product of number of repetitions of the two loops.

5.11. Using Files for Data Storage

a. Can use files instead of keyboard, monitor screen for program input, output
b. Allows data to be retained between program runs
c. Steps:
    i. *Open* the file
    ii. *Use* the file (read from, write to, or both)
    iii. *Close* the file
d. What we need:
    i. `fstream` header file
    ii. `ifstream` object for input
    iii. `ofstream` object for output
e. Opening Files
    i. Create a link between file name (outside the program) and file stream object (inside the program)
    ii. Use the `open()` member function
    iii. Filename may include drive, path info.
        1. If no path specified and just filename, then it will read from same directory as program being executed
    iv. Output file will be created if necessary; existing file will be erased first
    v. Input file must exist for open to work
        1. Check via `if(!fileobj) {file not opened}` or `fail()` member function
    vi. If you want to let the user specify the name of the file:
        1. You need to pass in the filename to `open()` as a null-terminated string, a.k.a. a C-string
        2. String literals are C-strings but string objects are *not*
        3. Use the member function `c_str()` on a string object to get a null-terminated string, e.g. `stringObject.c_str()`
f. Using Files
    i. Just like cin/cout
    ii. Use << to send data to a file and >> to read data from a file
    iii. >> returns true when data is successfully read

iv. Syntax:

```
while (inputfile >> number) {read the file}
```

g. Closing Files

i. Use the close() member function

ii. Don't wait for operating system to close files when programs end, because...

1. May limit the number of open files
2. May be buffered output data waiting to send to file

h. Example

Program 5-24

```
1  // This program lets the user enter a filename.
2  #include <iostream>
3  #include <string>
4  #include <fstream>
5  using namespace std;
6
7  int main()
8  {
9     ifstream inputFile;
10    string filename;
11    int number;
12
13    // Get the filename from the user.
14    cout << "Enter the filename: ";
15    cin >> filename;
16
17    // Open the file.
18    inputFile.open(filename.c_str());
19
20    // If the file successfully opened, process it.
21    if (inputFile)
22    {
23       // Read the numbers from the file and
24       // display them.
25       while (inputFile >> number)
26       {
27          cout << number << endl;
28       }
29
30       // Close the file.
31       inputFile.close();
32    }
33    else
34    {
35       // Display an error message.
36       cout << "Error opening the file.\n";
37    }
38    return 0;
39 }
```

Program Output with Example Input Shown in Bold

```
Enter the filename: ListOfNumbers.txt [Enter]
100
200
300
400
500
600
700
```

5.12. Breaking and Continuing a Loop

a. Use `break` to terminate execution of a loop

   i. When used in an inner loop, terminates the inside loop and returns to the outside loop

b. Use `continue` to go to the end of the loop and prepare for next repetition (a.k.a. skip to the next cycle)
    i. In `while`, `do-while` loops: go to the test and repeat loop if test passes
    ii. In `for` loop, perform the update step (`i++`), then test, repeat loop if the test passes
c. Use both sparingly (or not at all) because it makes your programs harder to understand/debug