CS141: Intro to Computer Science I
Lab 4: While Loop and gdb

Objectives:

- To learn how to use “loops” for repetitive executions of commands
- To use gdb to see how loops actually work
- To find and fix loop errors

Loop statement syntax:

A. FOR loop:

for(**variable initializations**; **test condition**; **variable update**)
{
//code to execute while the condition is true
// General Idea - set a loop variable at the very beginning
}

- **Variable initialization**: assignment statement, initializing a loop variable (execute the statement at the beginning of a "for loop");
- **Test condition**: boolean expression, determines when the loop should exit (evaluated at the beginning of each iteration through the loop);
- **Variable update**: change the loop variable (executed at the end of each loop iteration).

Loop statement syntax:

B. WHILE loop:

```
while(test condition)
{
//code to execute while the test condition is true
}
```

When the **condition** is true, it will execute the code inside {}. The test condition is first validated and then the code inside {} is ran. It repeats this until the test condition fails.

Loop statement syntax:

C. DO-WHILE loop:

```
do
{
//code to execute while the test condition is true
}
while(test condition);
```

Execute the code inside {} until the condition is false. The code will be executed first, then the condition is checked. Therefore, in a DOWHILE loop, the code within {}, will run at least once irrespective of the status of the test condition.

Increment/Decrement operators:

Increment operator: + +

Decrement operator: --

Prefix: ++i;

Postfix: i++;

Difference between Prefix and Postfix:

```
int x = 0;
if (x++ == 0) // Returns true, since x++ returns the value of x before adding 1 (0 == 0)
cout << x; // Will print out 1, since x has been incremented
int x = 0;
if (++x == 0) //Returns false, since ++x adds 1, then returns the value (1 == 0)
cout << x; //You won't reach this line
```

Assigning Values:

Assignment statements cause the variable (on the left) to take the value of the expression
on the right side of the equation. That means you can chain assignments together:

```
int a,b,c;
c = 5;
a = b = c;  // a = 5, b = 5 and c = 5
a = 1 + (b=c); //a = 6, b = 5 and c = 5
```

Input Validation:

In this section, a simple example of WHILE and DO-WHILE loops are discussed. Consider a
scenario where a user needs to enter their age. We know this entry should be a positive number,
no larger then let's say 150. What should be done if an incorrect value is entered?

```
int age = 0;
cout << “Enter your age” << endl;
cin >> age;
while(age < 0 || age > 150) {
    cout << “Please enter a value between 0 and 150 for
    your age” << endl;
    cin >> age;
}
```

Labwork. Task 1

In this lab, we will use the debugger gdb to understand how a loop works.

1) **Compile and debug the program lab04.cpp**

This program can be downloaded using wget from
http://www.clarkson.edu/class/cs141/lab04.cpp

2) Complete the answer sheet which can be downloaded using wget from
http://www.clarkson.edu/class/cs141/lab04_answersheet.html
and hand in the answer sheet to me before Friday 5:00pm 5th Feb 2016

Office hours: *MWF 11:30-12:30, TuTh 1:00-2:00. In the ITL or COSI (SC334, SC336).*

3) Fix the problems with the loop described on the answersheet. Submit the fixed code on odin (your lab4 folder). **FileName: lab04.cpp**

Labwork. Task 2

Write a program that takes a number n as input and generates Fibonacci Series starting with 0 until less than or equal to n using while loop.

Also re-write the same program using do-while loop.

FileName: Fib.cpp (save your file only under this name, not even "fib.cpp")

Input Example:
Number: 200

Output Example:
1 1 2 3 5 8 13 21 34 55 89 144

Recall

1) To use debugger compile lab04.cpp with the parameter –g using the command: **g++ -g lab04.cpp**
2) Run the debugger tool **gdb a.out**
3) Set the breakpoint in gdb using **break 1**
4) Run the program in gdb using **run**
5) Review the gdb usage

- Command **break Line#** or **b Line#:** sets the breakpoint in the program (example: **break 15** from Line15);
- Command **run** or **r**: executes the program and pauses at the breakpoint;
- Command **step** or **s**: executes only one line of the code that the gdb has displayed;
- Command **print var** or **p var**: displays the value in that variable (example: print n will output 5 if n=5);
- Command **set var=value**: sets the variable's value as specified (example: set n = 15);
- Command **jump Line#** or **j Line#:** jumps to the specified line of code from the current line.