BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJ.)

Second Semester 2017-18 CS F111 Computer Programming

LABORATORY SESSION #6

(Control Flow)

1. Write a program to count the number of vowels and consonants in a given English word that is taken as input from the keyboard. Your program must be case-insensitive (i.e., work for both uppercase and lowercase letters), and must employ `switch` construct.

2. Until interrupted, the following code prints TRUE FOREVER on the screen repeatedly:

```
while (1)
     printf("  TRUE FOREVER  ");
```

   Write a simple program that accomplishes the same thing, but use a `for` statement instead of a `while` statement. The body of the `for` statement should contain just the empty statement "`;`".

3. In the lecture class you worked with a program that double-spaces the given input. Taking help of the logic of this program, write a C program that partially reproduces the working of the Unix command `wc` (word count). Your program should display the number of lines and characters found in the given input (either standard input or file). Store your source code in the fine named `lab6_my_wc.c` and the corresponding executable in a file named `lab6_my_wc`.

   a. First try out `wc` command with a small sample file you can create using `cat` or `vi`.

   b. Now try using `lab6_my_wc` with the same file and see if the outputs are matching.

   c. Next, use it to find the number of users who are currently logged into the *Prithvi* server (Hint: you may have to use more than one command to accomplish this task.)

   Congratulations – you have just successfully written a general-purpose utility in C!

4. Do you remember from an earlier lab drawing a flowchart to generate a hailstone sequence, given a user input? You can refresh your memory by running the program `/home/share/hailstones` whose source is at: `/home/share/hailstones.c`. Copy the file into your current directory, compile and run the program using these inputs: (i) 45, and (ii) –21. Observe the output.
   Next, make a copy of the source code file as `lab6_hailstones_modified.c` and use the latter file for doing the following tasks.

   a. Incorporate an input validation check so that the user is repeatedly prompted to enter a positive number (instead of the program stopping when a negative integer is received as input by the user).

   b. Take two integers **a** and **b** as input from user. Modify the program so that it stops at the first possible occurrence of a hailstone number which falls in the range [a,b), after printing an appropriate message. The program prints all the hailstones generated thus far, and also the number of terms in the sequence thus far. In case none of the hailstones falls within the specified range, then entire sequence that is generated gets printed. Observe the working by running `/home/share/hailstones_modify` with the same input values you had used earlier: (i) 45 and (ii) –21. For 45, try these ranges: 50 – 60, 70 – 100.

   c. Copy down the program `lab6_hailstones_modified.c` into your lab notebook.

5. The standard library function `rand()` is used to generate pseudo random numbers.

   a. Generate and print 5 random numbers separated by a tab, using the `rand()` function inside a `for` loop. Observe the output. Run the program a few times repeatedly. What do you observe about the random numbers that are generated?

   b. Now modify your program like this before the `for` loop you have written:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
 int i, n, seed;
 seed = time(NULL);
 srand(seed);
 // the for loop goes here...
```

   Run the program a few times repeatedly. Now what do you observe about the random numbers that are being generated? Given below is the explanation for this.

   On most C systems, writing `time(NULL)` gives the number of elapsed seconds since 1 January 1970. Using this value as the `seed`, we now use another function `srand()` to seed the random-number generator. Repeated calls to `rand()` will generate all the integers in a given interval, but in a mixed-up order. The value used to seed the random-number generator determines where in the mixed-up order `rand()` will start to generate numbers. If we use the value produced by `time()` as a seed, then the seed will be different each time we call the program, causing a diferent set of numbers to be produced.

   c. Modify the above code to limit the random numbers only in the range [20,40), i.e. numbers greater than or equal to 20 and lesser than 40. (Hint: Use the modulo operator %) Copy the entire program into your notebook.

6. Fist predict the output of each of the following three program segments and understand how it works before trying out online (the file `/home/share/lab6_q6.txt` contains these three segments for your use):

(a)

```
int know=3;
switch (know)
{
  case 1:
     printf("one\n");
     break;
  case 3:
     printf("three\n");
  case 1+1+1:
     printf("hi\n");
     break;
  default:
     printf("bye\n");
}
```

(b)

```
float know=3.2;
switch (know)
 {
  case 1:
     printf("one\n");
     break;
  case 3.2:
     printf("three\n");
  case 1+1+1:
     printf("hi\n");
     break;
  default:
     printf("bye\n");
 }
```

(c)

```
int factor, know=1;
do
{
   for (factor=1; ; factor++)
     {
       if (factor >2)
          break;
       if (know == factor)
          continue;
       printf("%d%d\n", know, factor
     }
  know++;
} while (know <3);
```