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

Second Semester 2017-18 CS F111 Computer Programming

LABORATORY SESSION #7

(*Arrays; Functions*)

1. Write a program that takes a list of N integers as input from the user (N is also user input), and an integer X to search for in the list. After storing the list in an array, the program counts and prints the frequency of occurrence of X in the list.

2. Write a program that stores in a character array (called `word`) an English word input by user, encrypts the word according to the encryption key (a single digit integer) input by the user, and then prints out the encrypted string pattern on the screen. The encryption method is very simple: to shift each letter of the word by "n" ASCII values, where n is the encryption key. For instance, suppose the word input by user is ***Apple***. If the encryption key is ***2***, then the encrypted string printed on the screen is ***Crrng***. If the key is ***–2***, the encrypted string printed is ***?nnjc***.

   Hints: (i) Use a single `scanf()` statement for storing the word input by the user.

   (ii) As part of the `for` loop used to change each letter, use the check `word[i] != 0`

3. In a previous lab, you had learned about the "hailstone" sequence of numbers? 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 `lab7_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. Now, instead of generating and printing the hailstone sequence inside the `main()`, write a function whose prototype is given below, all call the function from within `main()`:

   ```
   int printHailstones(int val); /* prints all hailstones starting from
                                    val, and returns the number of terms
                                    found in the sequence */
   ```

   (Hint: All you need to do is to simply copy and paste the existing code into the new function, count the terms and return it, and include the function call inside `main()` */

   c. Take two integers **a** and **b** as input from user. Modify the previous function 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. Rename this function `printLimitHailstones()`. The function prints all the hailstones generated till the limit, and returns number of terms printed. In case none of the hailstones falls within the specified range, then complete hailstone sequence gets printed. Observe the working of this program by running `/home/share/hailstones_modify` with the same input values you had used earlier: (i) 45 and (ii) –21. For 45, try the following ranges: 50 – 60, 70 – 100.

4. C has this library function for generating pseudo random numbers: `int rand(void);` This prototype is included in `<stdio.h>` and indicates that the function returns an integer (the random number), and takes nothing as an argument (the parameter list is `void`).

   a. Generate 10 random numbers using the `rand()` function (using a loop), store the ten numbers in an array called `arr`, and then print out all the numbers from the array. 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>    /* contains the prototype of srand() */
#include <time.h>      /* contains the prototype of time() */
int main()
{
 int i, n, seed, arr[10];
 seed = time(NULL);
 srand(seed);
 // the for loop and rest of the program 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 convert the random numbers generated to fall within [20,40), i.e. $>= 20$ and < 40 before storing them in the array (Hint: Use the modulo operator %).

   d. Find and print the range of the ten numbers. Make use of two functions (prototypes shown below) that you should define and then call appropriately inside `main()`:

```
int findMax(int arr[]); /* returns the largest integer of arr */
int findMin(int arr[]); /* returns the smallest integer of arr */
```

   As seen from the prototypes, each of these functions takes an array as an argument and returns an integer back to the calling function. The way you will call these functions inside `main()` is shown below (`a` and `b` are declared to be of type `int`):

```
a = findMax(arr);            b = findMin(arr);
```

   Note how you are just mentioning the name of the array when calling the functions. The array is received as a formal argument in the function definition as follows:

```
int findMax(int arr[]) {                    int findMin(int arr[]) {
// function body goes here                  // function body goes here
}                                           }
```