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

Second Semester 2017-18 CS F111 Computer Programming

LABORATORY SESSION #7 SOLUTIONS

(*Arrays; Functions*)

1. Concepts tested: getting input into an array (using loop); linear search

```
 1 #include <stdio.h>
 2 int main()
 3 {
 4  int N, i, X, count = 0;
 5  printf("How many numbers? ");
 6  scanf("%d", &N);
 7  int nos[N]; // VLA
 8  for (i = 0; i < N; ++i)
 9  {
10   printf("Enter number %d: ", i);
11   scanf("%d", &nos[i]);
12  }
13  printf("\nNumber to be searched: ");
14  scanf("%d", &X);
15  for (i = 0; i < N; ++i)
16   if (nos[i] == X) count++;
17  if (count)
18   printf("%d occurs %d times in the
                    list.\n", X, count);
19  else
20   printf("%d was not found in the
                    list.\n", X);
21 }
```

Straightforward logic.

C99 allows *variable-length arrays* (VLAs) (line #7), whose sizes are known at runtime (and not compile-time).

2. Concepts tested: character arrays (strings); ASCII value; input validation check loop.

```
 1 #include <stdio.h>
 2 int main()
 3 {
 4   char word[50];    int key, i;
 5   printf("Enter a word: ");
 6   scanf("%s", word);
 7   do {
 8       printf("Encryption key (a
          single-digit number): ");
 9       scanf("%d", &key);
10      } while (key / 10 != 0);
11
12   printf("Original word: %s\n",
                    word);
13
14   for (i = 0; word[i]; ++i)
15      word[i] += key;
16
17   printf("Encyrpted word:
                    %s\n", word);
18   return 0;
19 }
```

Some of you may have had difficulty with writing the condition to check if a number is single-digit or not (line #10).

One may not necessarily change the original array, but print the encrypted string character by character, after changing it.

3. Concepts tested: writing and calling functions; input validation check loop; use of `break` statement inside loop

For part (a), validation check loop is written from line #12 – 15.

For part (b), the code for `printHailstones()` is same as what is being written for part (c) `printLimitHailstones()`, except line #34 and #35 should be removed.

For input 45, for range [50,60) `printLimitHailstones()` prints the six terms 45, 136, 68, 34, 17 and 52; for range [70, 100), the complete sequence of 17 terms 45, 136, 68, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2 and 1.

```c
#include <stdio.h>

int printHailstones(int);
int printLimitHailstones (int,int,int);

int main()
{
int val, total_terms, a, b;

printf("Enter a +ve integer to generate the hailstone sequence: ");
scanf("%d",&val);
while (val <= 0) {
  printf("Only positive numbers allowed! Enter again: ");
  scanf("%d", &val);
}
printf("Enter the interval limits...\n");
printf("Lower limit: ");
scanf("%d", &a);
printf("Upper limit: ");
scanf("%d", &b);
printf("The hailstone sequence is: ");
total_terms = printLimitHailstones(val,a,b);
putchar('\n');
printf("\nTotal no. of hailstones = %d\n", total_terms);
return 0;
}

int printLimitHailstones(int val, int a, int b)
{
  int count = 1, term;
  printf("%5d, ", val);
  for (term = val; term != 1; )
  {
   if (term >= a && term < b)
      break;
   if (term % 2 == 0)
      term = term / 2;
   else
      term = (3 * term) + 1;
   printf("%5d, ", term);
   count++;
  }
  return count;
}
```

4. Concepts tested: use of modulo operator; use of new library functions; writing functions.

   - Header files carry function prototypes (declarations), and not the definitions of functions.
   - In part (a), the random numbers generated should be the same each time the program is run, while in part (b), they should be different (unless the program is run again within a second so the seed value is the same, and hence the random numbers!)

- For part (c), arriving at the logic for restricting the random numbers within the given range (line #15) may have been tricky for you to grasp.
- Passing arrays to functions has now been taught in the lecture class. Remember that no copy of the array is made when a function receives it as a formal parameter, because only the address is received. Hence, changes made to the array in the function can be seen even in the calling function. You may want to contrast this with a case of passing a simple variable of type `int`. The notation `int *` may also be used instead of `int[]`.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <time.h>
 4
 5 int findMax(int []);
 6 int findMin(int []);
 7
 8 int main()
 9 {
10  int i, n, seed, randnos[10], max, min;
11  seed = time(NULL);
12  srand(seed);
13
14  for (i = 0; i < 10; ++i)
15    randnos[i] = 20 + rand() % 20;
16
17  for (i = 0; i < 10; ++i)
18    printf("%4d%s", randnos[i],   i < 9 ? ", ":"\n");
19  max = findMax(randnos);
20  min = findMin(randnos);
21  printf("\tRange: %d - %d = %d\n", max, min, max - min);
22 }
23
24 int findMax(int a[])
25 {
26  int max, i;
27  max = a[0];
28  for (i = 1; i < 10; ++i)
29   if (a[i] > max) max = a[i];
30  return max;
31 }
32
33 int findMin(int a[])
34 {
35  int min, i;
36  min = a[0];
37  for (i = 1; i < 10; ++i)
38   if (a[i] < min) min = a[i];
39  return min;
40 }
```