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

Second Semester 2017-18 CS F111 Computer Programming

LABORATORY SESSION #11 SOLUTIONS

(Introducing Structures; more string manipulation)

1. In the lecture class, we discussed an example of a student structure that stores the different attributes of a student viz., name, ID number, age and CGPA.

```
struct student {
char name[20];    // stores only the first name
char id[14];
int age;
float cgpa;
} s1;
```

a. Define the structure in your program globally before the `main()` function, i.e., write the above few lines before main().

b. Declare two other variables of type `struct student` having the names s2 and s3, both local to `main()`.

```
struct student s2, s3; // typed within main()
```

c. Inside main() find out how many bytes are allocated for s1.

```
printf("%d\n", sizeof(s1));
```

d. Accept user input for `s1` and `s2`, using the dot operator for every member. For instance, to accept the name, you would write: `scanf("%s", s1.name);` and for scanning the CGPA, you would type: `scanf("%f", &s1.cgpa);` and so on.

e. Assign the contents of `s1` to `s3` in one stroke. `s3 = s1;`

f. Now check if `s2` is equal to `s3`, and print a message appropriately.

Checking directly the equality of `s2` and `s3` by using the == operator is not allowed. Each member has to be checked individually.

g. Include the following function definition in your program:

```
void changeData(struct student s) { // receives a structure
strcpy(s.name,"Madhav");
s.age = 20;
s.cgpa = 7.33;
}
```

Call this function from main() as follows:

```
changeData(s1);    // takes a structure as argument
```

Print the contents of `s1` in `main()` before and after calling `changeData()`. What can you conclude about the mechanism of passing structures to functions?

Passed by value when only the structure is passed as an argument.

h. Print out the address of `s3`. Next, print out the addresses of the individual elements of `s3`. Note that the "member of operator" has the highest precedence among all operators. In other words, `&s1.age` will mean `&(s1.age)`, even without the parenthesis.

i. How would you declare a variable to store the address of `s1`?
`struct student *p; p = &s1;`

2. Modify the function you wrote for question 10-3 (previous lab) so that a new function called **`isAnagram2()`** ignores the case of the letters of the strings that are compared. For example, this function would return a value 1 even for the two strings "Madam Curie" and "Radium came". [Hint: You may use `toupper()` or `tolower()` whose declarations are available in `<ctype.h>` header file.]

A third version of this function, **`isAnagram3()`** that is not only case-insensitive, but also disregards any spaces found in the string has been implemented below, incompletely. (Note: A copy of this function, including `main()`, is available at: `/home/share/11.2.c` for you to copy into your directory.) For instance, this function is supposed to recognize each of the pairs such as "dormitory" and "dirty room", "Slot Machines" and "Cash lost in me" as anagrams. The logic is to add up the ASCII values of each character of the strings (excluding space and tab) and compare the sums. If the sums are equal and so are the number of non-space characters of both strings, the strings are anagrams, else they are not!

```
int isAnagram3(char *w1, char *w2)
{
  int i, sum_w1 = 0, sum_w2 = 0, no_chars_w1 = 0, no_chars_w2 = 0;
  for (i = 0; w1[i] ; ++i)
  {
    if (isspace(w1[i]))
        /* >>>>> a statement s1 goes here <<<<< */  continue;
    no_chars_w1++;
    sum_w1 += /* >>>>> an expression e1 goes here <<<<< */ ;
       Expression e1 can be toupper(w1[i]) or tolower(w1[i])
  }
  for (i = 0; w2[i] ; ++i)
  {
    if (isspace(w2[i]))
        /* >>>>> s2 goes here <<<<< */ continue;
    no_chars_w2++;
    sum_w2 += /* >>>>> e2 goes here <<<<< */ ;
      toupper(w2[i]) or tolower(w2[i])

  }
  if ( /* >>> e3 here <<< */ && /* >>> e4 here <<< */)
```

```
            Expression e3 is: no_chars_w1 == no_chars_w2
            Expression e4 is: sum_w1 == sum_w2
        return 1;
      else
        return 0;
    }
```

a. Complete the function and run the program taking some sample inputs.

b. Does this logic work for all cases, or does it fail in any? Can you think of an example where it fails?

It fails when you have two strings of equal lengths, whose ASCII sum is the same, yet the characters are different, e.g., call and balm both add up to 412 each. Hence, this algorithm is not correct.

c. **(Homework)** Can you design another algorithm for finding if two strings are anagrams? Hint: Think of how you would solve using pen and paper, and translate that into a C program.

The manual method of going letter by letter of the first word and "crossing it out" in the second word, if the same letter is found can be implemented as follows:

```
int isAnagram(char *w1, char *w2)
{
  int i = 0, j = 0;
  char s1[strlen(w1) + 1], s2[strlen(w2) + 1];
  strcpy(s1,w1); // making copies of the original strings
  strcpy(s2,w2);
  while (s1[i])
    {
      if (isspace(s1[i]))
          s1[i] = -1;       // spaces replaced them with -1
      i++;
    }
    while (s2[j])
    {
      if (isspace(s2[j]))
          s2[j] = -1;
      j++;
    }
  for (i = 0 ; s1[i] ; i++)  {   // for each character in s1
          if (s1[i] == -1)       // that is not a -1
             continue;
          for (j = 0; s2[j] ; j++)  // check each character of s2
          {
             if (s2[j] == -1) continue;  // barring any -1
```

```
            if (toupper(s1[i]) == toupper(s2[j]))
                 s1[i] = s2[j] = -1; // if found, “strike off”
                                    // the character from both the strings
        }
    }

 for (i = 0; s1[i]; ++i)
    if (s1[i] != -1)      // if any leftover character in s1
        return 0;         // it can’t be part of an anagram pair
 for (j = 0; s2[j]; ++j)
    if (s2[j] != -1)      // same with s2
        return 0;
 return 1;
}
```