BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJ.)
CS F111 Computer Programming
Lab Sheet# 10
(Pointers, Pointer Arithmetic, Call by reference)

Create a directory "**lab10**" inside "**myprogs**" directory for all your programs in this week's lab.

1. Write a function circleData() which takes the radius of a circle as input parameter, computes the area and the perimeter of the circle. The area and the perimeter must be available to the function that calls circleData(), which is the main() function. **[Hint: use pointers and call by reference]**. circleData() function should be declared in circle.h and defined in circle.c. Make another file circle_main.c that contains the main() function. Create a script for compiling and executing the entire project.

2. For the piece of codes shown below find out the output. First note down the answer you expect on your notebook then cross check the answer by executing the code.

   a.
   ```
   #include<stdio.h>
   int main()       {
         printf("%d", sizeof(void *));
         return 0;
   }
   ```

   b.
   ```
   #include<stdio.h>
   void function(char**);
   int main(){
         char *arr[] = { "ant", "bat", "cat", "dog", "egg", "fly" };
         function(arr);
         return 0;
   }
   void function(char **ptr){
         char *ptr1;
         ptr1 = (ptr += sizeof(int))[-2];
         printf("%s\n", ptr1);
   }
   ```

3. Predict the output of the following program without running. Then run and check the following. Come up with an explanation to your observations.

```c
#include <stdio.h>
int main()
{
    int arr[10] = {22,54,78,32,83,45,90,54,75,28};
    int * ptr = arr;
    printf("%d\n", *(ptr++));
    ptr = arr;
    printf("%d\n", *ptr++);
    ptr = arr;
    printf("%d\n", *++ptr);
    ptr = arr;
    printf("%d\n", *ptr+5);
```

```
    ptr = arr;
    printf("%d\n", --*ptr);
    return 0;
}
```

4. Now you will be writing a program to reverse an array. By now you must be aware that there are two ways of passing an array to a function i.e.:
    a. Passing arrays as a reference
       `void foo(int arr[])`
    b. Passing array as a pointer variable
       `void foo(int* arr)`

   However, you must also know that the compiler breaks either approach to a pointer to the base address of the array, i.e. int* array. So, using **`int array[3]`** or **`int array[]`** or **`int* array`** as the syntax for receiving the array as a function parameter, it is essentially the same thing. The difference lies only in the notations. Also, note that in all three cases, the syntax to access the array elements is the same. For example, in all three cases, the second element of the array can be accessed using either of the following syntax: **`array[1]`** or **`*(array+1)`**.

   In this exercise you need to complete the functions defined in the **"reverseArray.h"** header file given to you along with this lab sheet (**ReverseArray.zip**). You need to complete the code for the two different functions – "**reverseArray_PassByReference**" and "**reverseArray_PassByPointers**", each reversing an array but using different ways of passing an array to a function. Note that these two functions are to be defined in two different ".c" files, as given in the attached zip folder.

5. Write a program to find the transpose of a 2D matrix of **4*4** dimensions using two functions **`transposeByReference()`** and **`transposeByPointer()`**. While **`transposeByReference()`** should accept the original matrix in form of a 2D array as a function parameter using pass-by-reference, **`transposeByPointer()`** should accept a pointer to the original matrix as the function parameter. The starter code has already been provided along with the main file to test your code along with this lab sheet (**transposeMatrix.zip**). You are expected to complete the missing function codes and write a shell script to make an executable file and run it.

Additional Practice Exercises

1. Write a program to count the number of vowels and consonants in a string using a pointer. The function should accept a **char *** as an argument and should print the final counts of vowels and consonants. Make sure there are no other arguments to your function. **[HINT: Strings in C always end with a '\0']**

2. In this exercise you need to create a **`struct Employee`** for holding an employee record. Each record will have an **ID(int), salary(float), division(char[25]).** The program should let the user enter the number of Employees and then take the data for each employee one by one. You need to store all the data in an array of structures. You can use an array of structures declared like this:

   `struct Employee employeeData[MAX];`

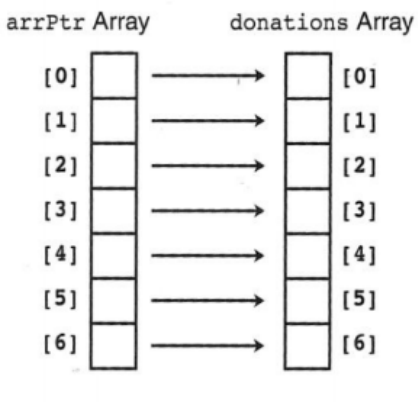
Next, you will need to sort the data contained in the above array, based on the salary of each employee in ascending order and then print it on the console. However, instead of passing the array of structures to the function directly, you need to create a new array of pointers each pointing to the base address of a record of an employee, and pass this as a parameter to the sorting function. You can declare the array of pointers like this:

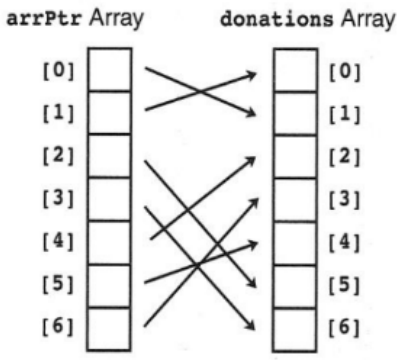
```
struct  Employee* employeeDataPointers[MAX];
```

This way when you implement your sort function, you are not swapping two structures and all of their contents to bring them into the right order, you will just swap the addresses where those structures lie in the memory and can then print them based on the addresses. This saves a lot of overhead of processing in case the amount of data in a structure is very big and moving data in memory is a bulky task. Your sort function declaration should look like

```
void sort (struct Employee * employeeDB[], int size);
```

Here are some diagrams that illustrate the idea:





In the above figure, the original dataset (donations Array) is contained in the array on the right side of each picture, and an array of pointers points at various elements of the original dataset on the left in each picture. We first initialize the array of pointers so that each element in the pointer array points to the element in the data array that has the same index value (left-hand picture). We then sort the pointers according to the values they point to, leaving the original dataset untouched (right-hand picture).

You need to implement a similar strategy with the original data as an array of struct Employee instead of donations array to solve your problem.