**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJ.)**
**CS F111 Computer Programming**
**LAB SESSION #11**
(Dynamic Memory Allocation, Command Line Arguments)

Create a directory "**lab11**" inside "**myprogs**" directory for all your programs in this week's lab.

**Command-line arguments (source: tutorialspoint.com)**

It is possible to pass some values from the command line to your C programs when they are executed. These values are called **command-line arguments** and many times they are important for your program especially when you want to control your program from outside instead of hard-coding those values inside the code.

The command-line arguments are handled using the main() function arguments where **argc** refers to the number of arguments passed, and **argv[]** is a pointer array that points to each argument passed to the program. Essentially argv is an array of character arrays (or strings). Following is a simple example that checks if there is any argument supplied from the command line and take action accordingly –

```
#include <stdio.h>

int main( int argc, char *argv[] )  {

   if( argc == 2 ) {
      printf("The argument supplied is %s\n", argv[1]);
   }
   else if( argc > 2 ) {
      printf("Too many arguments supplied.\n");
   }
   else {
      printf("One argument expected.\n");
   }
}
```

When the above code is compiled and executed with single argument, it produces the following result.

```
$./a.out testing
The argument supplied is testing
```

When the above code is compiled and executed with a two arguments, it produces the following result.

```
$./a.out testing1 testing2
Too many arguments supplied.
```

When the above code is compiled and executed without passing any argument, it produces the following result.

```
$./a.out
One argument expected
```

It should be noted that **argv[0]** holds the name of the program itself and **argv[1]** is a pointer to the first command-line argument supplied, and **argv[n]** is the **n$^{th}$** argument. If no arguments are supplied, **argc** will be one, and if you pass one argument then **argc** is set at 2.

You pass all the command line arguments separated by a space, but if argument itself has a space then you can pass such arguments by putting them inside double quotes "" or single quotes ''. Let us re-write above example once again where we will print program name and we also pass a command line argument by putting inside double quotes −

```
#include <stdio.h>

int main( int argc, char *argv[] )  {

   printf("Program name %s\n", argv[0]);

   if( argc == 2 ) {
      printf("The argument supplied is %s\n", argv[1]);
   }
   else if( argc > 2 ) {
      printf("Too many arguments supplied.\n");
   }
   else {
      printf("One argument expected.\n");
   }
}
```

When the above code is compiled and executed with a single argument separated by space but inside double quotes, it produces the following result.

```
$./a.out "testing1 testing2"

Program name ./a.out
The argument supplied is testing1 testing2
```

**Exercise:** Execute the following program by giving three integers as command-line arguments. **[Note:** atoi() is a C library function that converts a string argument to an integer value. **End of Note]**

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
    int a, b, c;

    a = atoi(argv[1]);
    b = atoi(argv[2]);
    c = atoi(argv[3]);
    printf("\n %d",a+b+c);
    return 0;
}
```

**Dynamic memory Allocation**

Q1. Create a function fun1() that takes an array of integers and its size as input and swaps the first and the last element. fun1() should be called from the main() function. The array that will be passed into the function fun1() must be created in the main() function. fun1() **should not return anything, but main() should see the elements swapped.**

So, create an integer array of size 10 in the main() function, print it, then pass it to fun1(), and print the array again after fun1() returns. Use (a) static array; (b) dynamically allocated array.

Q2. Consider the following code. What is wrong with this code?

```
1. int main(int argc, char*argv[]){
2. char* answer;
3. printf("Please type something: ");
4. gets(answer);
5. printf("you typed %s \n",answer);
6. return 0;
7. }
```

Can you fix the code by modifying line 2?

Q3. Write a program that takes two arrays of the same size as input. The size of the arrays is not known till runtime and has to be taken from the user. The program then takes the elements of the arrays from the user as input too. Finally, the program prints the dot product of the two arrays on the console. [Taking dot product of the arrays is same as multiplying the arrays element wise and then taking the sum of all the products]
Note that you need to declare all the arrays dynamically and allocate them on the heap for the implementation of the above question, no static arrays are allowed.

Sample Output

```
Enter the size: 3
Enter the elements of the first array
2 5 10
Enter the elements of the second array
4 10 8
The dot product of the arrays is 138.
```

***[Hint: 2*4 + 5*10 + 10*8 = 138]***

Q4. You will have to create an array of student records using dynamic memory allocation. Each record should contain the following fields: **ID** (char array), **Name** (char array), **Dept** (char array), **math_marks** (integer), **phy_marks** (integer), **chem_marks** (integer). The main function should accept the number of such records as an input as a command-line argument and dynamically create such an array of student records. Then in a loop, you should take input from the user and fill that array. Then this array is to be passed to a function **fun3()** along with its size, which in-turn sorts that array based on marks obtained in maths. Then this array is to be returned back to the main() function, where this array is to be printed (now in sorted order). Don't use a pointer to pointer of any data type.

**Additional Practice Exercises**

Q1. Write a program to create a 2D array (**say new2D**) that contains 2 rows. The first row should store an array of integers and the second row should store an array of float values. Take the number of elements in the first array and the second array as input from command line arguments. Then fill the 2D array with values taken from the user as input using scanf. Then Print the 2D array.

**[Hint:**
Create a double-pointer of the type void (with the name **new2D)**. Then allocate memory to it for size 2. This creates an array of size 2, where each location can store address of another variable (or an array). Then use appropriate typecasting to allocate an integer array in the first location of new2D and a float array in the second location of new2D, using the respective sizes taken from the user as input.
**End of Hint]**

Q2. Write a program to create an integer 2-D array where the number of rows and the number of elements of each row must be taken as input from the user **using command-line arguments only**. After that all the values to be stored in the array must be taken as input from the user.

Then this array must be passed into another function fun2(), which sorts elements in each row separately in increasing order. You can pass a few more arguments to fun2() as needed. **Your function fun2() should not return anything, yet the change should be reflected in the main() function who is calling fun2().** You must use dynamic memory allocation.

In the main function, after fun2 returns, you must print the 2D array. Then you should free the memory allocated to the above 2D array.