King Fahd University of Petroleum and Minerals
Information & Computer Science Department
ICS 431 Operating System Lab
Lab # 3
Programming in C (Advanced)

Objectives:

- ✓ To learn how to write C-Program using command line arguments
- ✓ To know how to use pointers
- ✓ To know how to declare and define a structure
- ✓ To know how to use 1-D Array
- ✓ To know how to use 2-D array
- ✓ To know how to handle Strings

How to write C-Program using command line arguments?

What are Command line arguments in C?

The arguments for **main(int argc, char *argv[])** are **argc**(argument count) and **argv**(argument vector). The argument **argc** is an integer that contains the number of arguments being passed to main ; **argv** is an array of strings. Each string is an argument. The computer is able to count the number of arguments and assign the value to **argc.** The array **argv[1]** contains the first argument, **argv[2]** contains the second argument and so on . **The program's name is stored in argv[0].**

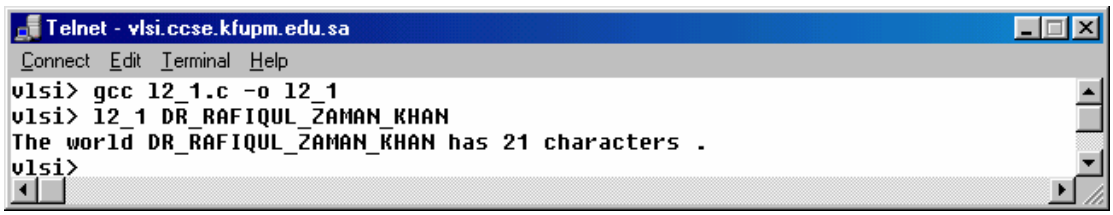
Example#1 (How to use command line arguments):

```
 #include<stdio.h>
/*Sample program to use command line arguments */
main(int argc, char *argv[])
{
        int n=0;
        /* Findout if any world passed to program */
        if(argc==1)
        {
        printf("No world to examine. \n");
        exit(0);
}

/* Loop to count characters */
while(argv[1][++n] != '\0')
{

}
/*Print result */
printf("The world %s has %d characters .\n",argv[1],n);
}
```

Sample Output:



Example#2(Use command line arguments):

```
#include<stdio.h>
#include<stdlib.h>

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

        if(argc==3)
        {
                printf("You must input three numbers as command line inputs. \n");
                exit(0);
        }

        sum=atoi(argv[1])+atoi(argv[2])+atoi(argv[3]);

        printf("argv[0] is = %s\n",argv[0]);

        printf("The sum of three values is =  %d\n",sum);
}
```

Sample Output:

Note: **atoi( )** function is used to convert string into integer.

What is a Pointer? :

Pointer variable is a special variable, which stores the address of other variable. If a pointer variable stores the address of a char variable, we call it a character pointer and so on.

Normally a variable directly contains a specific value. A pointer, on the other hand, contains an address of a variable that contains a specific value. Pointers like any other variables must be declared before they can be used.

To handle pointers in C language we use two unary operators: &, *

& : Address operator (ampersand symbol)

***** : Pointer operator OR Indirection operator OR Value at address.

Examples:

int *j --------→ Means the value at address contained in j is an int **OR** in other words j is an integer pointer.
int **k ------→ Means the value at address contained in *k is an int.

Example#3 (Pointers):

```
#include<stdio.h>
void test1(int m, int n) //user-defined function
{
        m=5;
        n=24;
}

void test2(int *m, int *n) // user defined function
{
        *m=5;
        *n=24;
}

void test3(int a, int *b) //user-defined function {
        a=38;
        *b=57;
}

int main(void) {
        int a=10, b=16;
        printf("a=%d, b=%d\n",a,b);

        test1(a,b); // call of function
        printf("a=%d, b=%d\n",a,b);

        test2(&a,&b); //call of function
        printf("a=%d, b=%d\n",a,b);

        test3(a,&b); //call of function
        printf("a=%d, b=%d\n",a,b);
        return 0;
}
```

Sample output:

Example#4 (Pointers):

```
#include <stdio.h>
main()
{
    int i1, i2, *p1, *p2;
```

```
    i1 = 5;
    p1 = &i1;
    i2 = *p1 / 2 + 10;
    p2 = p1;
    printf("i1=%d, i2=%d, *p1=%d, *p2=%d\n", i1, i2, *p1, *p2);
}
```

How to use Structure?:

What is structure?:

Structures are collections of related variables. Structures may contain variables of many different data types- in contrast to arrays that contain only elements of the same data type; Structures are commonly used to define records to be stored in files. Pointers and structures facilitate the formation of more complex data structures such as linked lists, queues, stacks, and trees.

How to declare and initialize structures:

The elements can be of any type including enumerated types, arrays and even other structures.

e.g:
```
struct user_record {
                        int id_no;
                        STRING name;
                        STRING dept;
                         int no_of_books;
                        USER_STATUS status;
                       };
```
This declares a structure with user record as tag. Again, memory is not allocated until a variable is declared.

A variable with the above structure can be declared as:

```
struct user_record user;
```

We can simplify the variable declaration above, if we use typedef in the declaration.
The above can then be defined as:

```
typedef struct {
              int id_no;
               STRING name;
               STRING dept;
               int no_of_books;
               USER_STATUS status;
               } USER_RECORD;
```

The variable declaration then becomes:

```
USER_RECORD user;
```

Once a variable is declared, we can access the individual elements (called fields) of the record using the member operator, also called the dot (.) operator.

We can combine declarations and declare structure variables at the same time that the structure type is declared. For example:

```
struct date
{
int day, month, year;
} arrival_date, departure_date;
```

or without a structure tag

```
structure { int day, month, year;} arrival_date, departure-date;
```

Example#5 (How to use Structure):

```
/*This program shows the structure declaration, initialization, accessing members, and structure assignments.*/

# include <stdio.h>     /* including the stdio library */
int  main  ( )
{
/* declaring a structure type (template) */
          struct   car  {  char  maker [40] , model [40] ;
                                   int   year ;
                         } ;

/* Declaring a structure variable, and initializing it */
     struct   car   FirstCar = { "Ford", "Mustang", 1996} ;

/* Declaring a structure variable */
                    struct   car  SecondCar ;

/* Displaying the structure FirstCar */
printf ( "%20s %s, %s, %d\n", "First  Car: ", FirstCar.maker, FirstCar.model, FirstCar. year) ;

/* assigning FirstCar to SecondCar using the assignment operator */
          SecondCar = FirstCar ;

/* displaying the structure SecondCar */
printf ( "%20s %s, %s, %d\n","Second Car: ",  SecondCar.maker, SecondCar.model, SecondCar.
year ) ;

/* displaying the size of each structure variable */
printf  ( "%20s is %d Bytes\n", "Size of FirstCar: ", sizeof (FirstCar )  ) ;
printf  ( "%20s is %d Bytes\n", "Size of SecondCar: ", sizeof ( SecondCar )  ) ;
puts ( "End of my act! :-) \n" ) ;
return 0 ;
}
```

How to use 1-D Array in C-Programming?:

Example#6 (1-D Array):

```
/*********************************************************************
Create three arrays. Read data into the first two of them. Subtract each
element in the first array from the corresponding element in the second
array. Store the differences in the third array. Print all the arrays.
*********************************************************************/

#include<stdio.h>
#define MAX_SIZE 5
void main()
{
int first[MAX_SIZE], second[MAX_SIZE], diff[MAX_SIZE], i;

printf("\nEnter %d data items for first array : ", MAX_SIZE);

for(i=0;i<MAX_SIZE; i++) // input first array
{  // input first aaray
scanf("%d", &first[i] );
}

printf("\nEnter %d  data items for second array : ",  MAX_SIZE);

for(i=0;i<MAX_SIZE; i++) // input second array
{
scanf("%d",&second[i]);
}

for(i=0;i<MAX_SIZE; i++)   // compute the differences
{
diff[i]=second[i] - first[i];
}

printf("\n\nOutput of the arrays : ");
for(i=0;i<MAX_SIZE; i++)   // output the arrays
{
printf("\n\n%5d %5d %5d", first[i],  second[i],  diff[i]);
}

} // end of main
```

How to use 2-D Array in C-Language?:

Example#7 (Use of 2-D Array):

/*Matrix Addition */

```
#include<stdio.h>
#define row 10
```

```
#define col 10

int main()
{
int i,j,k, a[row][col], b[row][col],  s[row][col] = {0};
int rowsa, colsa, rowsb, colsb;

printf("Enter number of rows for Matrix 1: ");
scanf("%d", &rowsa);
printf("Enter number of columns for Matrix 1: ");
scanf("%d", &colsa);
printf("Enter the %d elements of Matrix 1 : \n", rowsa*colsa);

for(j=0;j<rowsa; j++) // reading matrix a
{
 for(k=0;k<colsa;k++)
 {
 scanf("%d", &a[j][k]);
 }
}
 puts("\n");

printf("Enter number of rows for Matrix 2: ");
scanf("%d",&rowsb);
printf("Enter number of columns for Matrix 2: ");
scanf("%d",&colsb);
printf("Enter the %d elements of Matrix 2 : \n", rowsb*colsb);

for(j=0;j<rowsb; j++) // reading matrix b
{
 for(k=0;k<colsb;k++)
 {
 scanf("%d", &b[j][k]);
 }
}

 /* Addition of two matrices */
  for(j=0; j<rowsb; j++)
  {
                    for(k=0; k<colsb; k++)
                    {
                            s[j][k]=a[j][k]+b[j][k];
                    }
  }

/*Print sum of two matrices */
printf("The sum of two matrices is : \n");
  for(j=0; j<rowsa; j++)
  {
          for (k=0; k<rowsb; k++)
          {
          printf("%5d ", s[j][k]);
```

```
        }
        printf("\n");
}

 return 0;
 } // end of main
```

Strings:

Strings are implemented as array of characters. The end of the string is always marked with a null character. Loops are used to manage the string, one character at a time. A test for the null character detects the end of the string. A string constant is specified by a set of double quotes. This string constant has an address; the address of the string. The string's address can be assigned to a pointer.

Examples:

```
char course_name[] = "Operating System";
char *depart_name = "ICS";
char *colg_name = {'C', 'C', 'S', 'E', '\0'};
```

A set of functions in the standard library is designed to perform common operations on strings. These functions are defined in the header file strings.h.

Function name	Description
strlen	finds the length of a string
strcpy	copies a source string to another string
strcmp	compares two strings
strcat	concatenates two strings.
strtok	string tokenization

Exercises

Note:

Lab Problems will be given during the lab based on material covered in this lab manual.