Exp 5: Introduction to C programming

C language is known as a middle level language, which is having features of low level, as well as higher-level languages. That is why C language is very much suitable for the systems programming. C language is still the most suitable language for systems programming.

C and C++ programming languages have many common things like:

- Both of them are case-sensitive language.
- The programs start the execution from the **main ()** function.
- Have the same basic data types (int, double, char) but don't support the Boolean values.
- Define and initialize variables in the same way.
- Have the same control and repetition statements (if-else, switch, for, while, do-while).

So, in this tutorial we will skip the discussion of the previous points and go quickly through the basic things we need to write and compile C programs under Linux.

1. Compile and run your first C program

Study the following code and follow the instructions below in order to compile and execute it.

```
/*********************************
Print  a  message  on  standard  output
File name: prog0.c
*********************************/
#include <stdio.h>

int main(int argc, char* argv[])
{
      printf("Welcome to first C program\n");
      return 0;
}
```

1- Create a new folder on your desktop and name it **Test**. In this folder create a file and name it **prog0.c,** then copy the previous code to it.

2- Open the terminal and change the working directory to **Test** folder using the following command:

```
cd Desktop/Test
```

3- List all files and directories using **ls** command, and make sure your file exist between them.

4- To compile a C program in Linux we use **gcc** compiler as follows:

```
gcc program-source-code.c -o executable-file-name
```

5- Now to compile your code use the following command:

```
gcc prog0.c -o exec0
```

If your program has no errors then you will notice the creation of new file named **exec0** in Test directory**.**

6- To run your program write the following command and see the output.

```
./exec0
```

Notes:

- You should always use g++ to link a program that contains C++ code, even if it also contains C code. If your program contains only C code, you should use gcc instead.
- The -c option tells gcc to compile the program to an object file only; without it, g++ will attempt to link the program to produce an executable.

2. Printf() function

It is used to send a formatted output to stdout. It is declared as follows:

```
int printf(const char *format, ...)
```

format − This is the string that contains the text to be written to stdout. It can optionally contain embedded format tags that are replaced by the values specified in subsequent additional arguments and formatted as requested. Format tags prototype is % specifier, the table below shows some of commonly used specifiers:

Output	Specifier
Character	c
Signed decimal integer	d or i
Decimal floating point	f
String of characters	s

```
/*********************************
The use of printf() function
File name: prog1.c
Compile: gcc prog1.c –o
exec1 Run: ./exec1
*********************************/
#include <stdio.h>

int main()
{
      int v1=5;
      double v2 = 9.678;
      float v3=20.54;
      char v4='R';
      char v5[]="Operating Systems";
      char v6[]="Welcome to OS lab";
      printf("%s\n",v6);
      printf("integer = %d \t integer = %i \n",v1,
      v1); printf("double = %f \t float =
      %f\n",v2,v3); printf("char = %c \t string = %s
      \n",v4,v5); return 0;
}
```

3. Read values from user

C language helps programmers to create an interactive programs that allow users to insert values to variables at run time. In this tutorial we will list some of the most commonly used methods.

Command line arguments

Command line arguments are used to pass some values from the command line to your C programs when they are executed. They are handled using main () function arguments where argc refers to the number of arguments passed, and argv[] is a pointer array which points to each argument passed to the program. Following is a simple example which checks if there is any argument supplied from the command line and take action accordingly:

```
/*********************************
The use of command line arguments
File name: prog2.c
*********************************/
#include <stdio.h>

int main( int argc, char *argv[] )
{
printf("Program name %s\n", argv[0]); int i;
      if( argc == 3 )
         for(i=1;i<3;i++
         )
             printf("The argument %d is %s\n",i, argv[i]);

      else if( argc > 3 )
           printf("Too many arguments supplied.\n");
     else
            printf("Two argument expected.\n");
 }
```

Compile the previous code and name the executable file **exec2**. Try to execute your program in the following ways and notice the result.

```
      ./exec2 aaa
```

```
      ./exec2 aaa bbb
```

4. scanf() function

Another way to read values from user is using the function scanf(). It is used to read a formatted input from stdin. Scanf keeps reading until a first white character is inserted. White characters are space, tab and new line.

```
int scanf(const char *format, ...)
```

```
/*********************************
The use of scanf() function
File name: prog3.c
Compile: gcc prog3.c –o
exec3 Run: ./exec3
*********************************/
#include <stdio.h>

int main( int argc, char *argv[] )
{
      int i;
      float
      f; char
      ch;
      char str[20];

      printf("Insert character \n")
      ; scanf("%c",& ch);

      printf("Insert string \n")
      ; scanf("%s",str);

      printf("Insert integer number \n")
      ; scanf("%i",&i);

      printf("Insert float number \n")
      ; scanf("%f",&f);

      printf("Inserted values are %d \t %f \n",i,f) ;
      printf("Inserted values are %c \t %s \n",ch,str) ;
}
```

5. *fgets() Function.*

```
      char *fgets(char *str, int n, FILE *stream)
```

It reads a line from the specified stream and stores it into the string pointed to by str. It stops when either (n-1) characters are read, the newline character is read, or the end-of-file is reached, whichever comes first.

scanf() leaves the newline character (\n) in the iostream buffer. If fgets() is used after scanf(), the fgets() sees this newline character as leading whitespace, thinks it is finished and stops reading any further. In order to solve this problem we define a new array of characters to hold the '\n' character, and use it after scanf() as you will notice in the next example.

```
/*********************************
The use of fgets() and scanf() functions
File name: prog4.c
Compile: gcc prog4.c –o
exec4 Run: ./exec4
*********************************/
#include <stdio.h>

int main( int argc, char *argv[] )
{
      int i;
      char  str1[20];
      char  str2[20];
      char dump[5];

      printf("Insert an integer \n");
      scanf("%i",& i);
      fgets(dump,5,stdin);

      printf("Insert a string \n") ;
      fgets(str1,20,stdin);

      printf("Insert a string \n") ;
      fgets(str2,20,stdin);

      printf("Inserted values are %d \n %s \n
      %s \n",i,str1,str2) ;
}
```

After you compile and run the previous code remove **fgets(dump,5,stdin)** statement, then compile and run the program again and note what happened.

6. C structures

C arrays allow you to define type of variables that can hold several data items of the same kind but structure is another user defined data type available in C programming, which allows you to combine data items of different kinds.

Structures are used to represent a record, suppose you want to keep track of students in a school. You might want to track the following attributes about each student: ID, Name, and GPA. An example of how to define and use a structure in C is shown below:

```
/*********************************
The use of structures
File name: prog5.c
Compile: gcc prog5.c –o
exec5 Run: ./exec5
*********************************/
#include <stdio.h>

struct Student
{
   int      ID;
   char     Name[50];
   int      GPA;
} Student;

int main( int argc, char *argv[] )
{
      char dump[5]; struct
      Student std1;

      printf("Insert student ID\n");
      scanf("%i",& std1.ID);
      fgets(dump,5,stdin);

      printf("Insert student Name\n");
      fgets(std1.Name,50,stdin);

      printf("Insert student GPA\n");
      scanf("%i",& std1.GPA);

      printf("Student Info \n ID:\t %d\n",std1.ID) ;
      printf(" Name:\t%s GPA:\t%i\n",std1.Name,std1.GPA) ;

      }
```

7. File Manipulation

Files are used for permanent retention of large amounts of data. One way to access files is through a set of C library functions such as:

```
int fopen(const char *path, int oflag, ...);
int close(int fildes);
int fprintf ( FILE * stream, const char * format, .. );
int fscanf ( FILE * stream, const char * format, ... );
```

The **oflag** argument in **fopen()** function contain the file access mode. It includes

Opens a file for reading. The file must exist.	**"r"**
Creates an empty file for writing. If a file with the same name already exists, its content is erased and the file is considered as a new empty file.	**"w"**
Appends to a file. Writing operations, append data at the end of the file. The file is created if it does not exist.	**"a"**
Opens a file to update both reading and writing. The file must exist.	**"r+"**
Creates an empty file for both reading and writing.	**"w+"**
Opens a file for reading and appending.	**"a+"**

```
/*********************************
Copy the args inserted in command line arguments to a
file File name: prog6.c
Compile: gcc prog6.c –o exec6
Run: ./exec6 1111 aaaa bbbb 5555
*********************************/
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
      FILE *fp;
      int i;
      if((fp=fopen("test", "w")) == NULL)
      {
            printf("Cannot open file.\n");
            exit(1);//exist in stdlib
      }
```

```
	for (i=1;i<argc;i++)
		fprintf(fp, "%s \n", argv[i]); /* write to file
	*/ fclose(fp);
	return 0;
}
```

```
/*********************************
More on the use of command line arguments
File name: prog7.c
Compile: gcc prog7.c –o exec7
Run: ./exec7 1111 aaaa bbbb
*********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
	int i;
	char str[80];
	if(argc == 4)
	{
		i=atoi(argv[1]); //convert string to integer
		strcpy(str,argv[2]); //Copy argv[2] to str
		strcat(str,"\t");
		strcat(str,argv[3]); //Concatenate argv[3]
with str
	}

	printf("integer = %i \n", i);
	printf("string = %s \n", str);

	return 0;
}
```

8. ***GNU Make***

```
reciprocal: main.o reciprocal.o
        g++ $(CFLAGS) -o reciprocal main.o reciprocal.o

main.o: main.c reciprocal.hpp
        gcc $(CFLAGS) -c main.c

reciprocal.o: reciprocal.cpp reciprocal.hpp
        g++ $(CFLAGS) -c reciprocal.cpp

clean:
        rm -f *.o reciprocal
```

% make

% make clean

% make CFLAGS=-O2

9. Running GDB

% gdb reciprocal

(gdb) run

(gdb) where

(gdb) up 2

(gdb) print argv[1]

(gdb) break main

(gdb) run 7

(gdb) next

(gdb)step