CL101 INTRODUCTION TO COMPUTING

LAB 07

FUNCTIONS AND RECURSION IN C

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

FUNCTIONS

A function is a group of statements that together perform a task. Every C program has at least one function, which is main().
A function is known as with various names like a method or a sub-routine or a procedure etc.

CATEGORIES OF FUNCTIONS:

Functions are classified into two categories.
a) Library functions
b) User defined functions

LIBRARY FUNCTIONS:

Library functions are built in functions which are defined by C library, example printf(), scanf(), strcat() etc. You just need to include appropriate header file to use these functions. They are already declared and defined in C libraries.

List of standard library functions under different header files in C

library	Functions			
<stdio.h>	prinf()	fprintf()	sprint()	scanf()
	putchar()	getchar()	puts()	gets()
	putc()	getc()	fputs()	fgets()
	fputc()	fgetc()	fopen()	fclose()
<conio.h>	clrscr()	getch()	getche()	textcolor()
	textbackground()	gotoxy()	kbhit()	wherex()
<ctype.h>	isalnum()	isalpha()	iscntrl()	isdigit()
	isgraph()	islower()	isprint()	isupper()
<math.h>	acos()	acosh()	asin()	asinh()
	atan()	atanh()	pow()	sqrt()

USER DEFINED FUNCTIONS:

C allows programmer to define their own function according to their requirement. These types of functions are known as user-defined functions.

DEFINING A FUNCTION:

The general form of a function is as follows:

```
return_type function_name( parameter list )
{
        body of the function
}
```

A C function definition consists of a function header and a function body. Here are all the parts of a function:
Return Type: A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.
Function Name: This is the actual name of the function. The function name and the parameter list together constitute the function signature.
Parameters: A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

Function Body: The function body contains a collection of statements that define what the function does.

CALLING A FUNCTION:

- To use a function, it must be called or invoked.
- When a program calls a function, program control is transferred to the called function. A called function performs defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.
- To call a function, it is needed to pass the required parameters along with function name, and if function returns a value, then returned value can be stored.

DIFFERENT NTYPES OF FUNCTIONS

There are four types

a) Function with no arguments and no return value

b) Function with no arguments and return value

c) Function with arguments but no return value

d) Function with arguments and return value.

Function with no arguments and no return type

```
void prime();
int main(){
   prime();
   return 0;
}
void prime(){
   int num,i,flag=0;
   printf("Enter positive integer enter to check:\n");
   scanf("%d",&num);

for(i=2;i<=num/2;++i){
      if(num%i==0){
         flag=1;
      }
   }
   if (flag==1)
      printf("%d is not prime",num);
   else
     printf("%d is prime",num);
   }
```

Function with no arguments and return type

```
#include <stdio.h>
int input();
int main(){
   int num,i,flag = 0;
   num=input();
   for(i=2; i<=num/2; ++i){
   if(num%i==0){
      flag = 1;
      break;
   }
   }
   if(flag == 1)
      printf("%d is not prime",num);
   else
      printf("%d is prime", num);
   return 0;
}
int input(){
   int n;
   printf("Enter positive integer to check:\n");
   scanf("%d",&n);
   return n;
}
```

Function with arguments but no return type

```
#include <stdio.h>
void check_display(int n);
int main(){
   int num;
   printf("Enter positive enter to check:\n");
   scanf("%d",&num);
   check_display(num);
   return 0;
}
void check_display(int n){
   int i, flag = 0;
for(i=2; i<=n/2; ++i){
   if(n%i==0){
      flag = 1;
      break;
   }
   }
   if(flag == 1)
      printf("%d is not prime",n);
   else
      printf("%d is prime", n);
}
```

Function with arguments and return type

```
#include <stdio.h>
int check(int n);
int main(){
   int num,num_check=0;
   printf("Enter positive enter to check:\n");
   scanf("%d",&num);

num_check=check(num);
   if(num_check==1)
      printf("%d is not prime",num);
   else
      printf("%d is prime",num);
   return 0;
}
int check(int n){

   int i;
   for(i=2;i<=n/2;++i){
   if(n%i==0)
      return 1;
}
  return 0;
}
```

PASSING ARGUMENTS BY VALUE AND BY REFERENCE

In many programming languages, there are two ways to pass arguments:

- pass-by-value
- pass-by-reference.

pass-by-value

When arguments are passed by value, a copy of the argument's value is made and passed to the called function. Changes to the copy do not affect an original variable's value in the caller.

pass-by-reference

When an argument is passed by reference, the caller allows the called function to modify the original variable's value.

EXAMPLE(PASS-BY-VALUE Vs. PASS-BY-REFERENCE)

```
#include <stdio.h>

int FuncByValue(int a)
{
        return a=a*a;
}

int FuncByReference(int *a)
{
        return *a = *a * *a;
}

int main()
{
        int c = 5;
        int a = FuncByValue(c);
        printf("After calling by value:%d\n",a);
        printf("Passed value:%d\n\n",c);

        int d = FuncByReference(&c);
        printf("After calling by reference:%d\n",d);
        printf("Passed value:%d\n",c);

        return 0;
}
```

```
After calling by value:25
Passed value:5

After calling by reference:25
Passed value:25

--------------------------------
Process exited after 0.03753 seconds with return value 0
Press any key to continue . . .
```

OUTPUT (PASS-BY-VALUE Vs. PASS-BY-REFERENCE)

SIGNIFICANCE OF FUNCTIONS:

a) Functions avoid rewriting the same code over and over.
b) Functions make it easier to write programs and keep track of what they are doing
c) Functions provide modularity to program
d) Functions allow code reusability
e) In case of large programs, with thousands of code lines, debugging and editing become easier

RECURSION

A recursive function is a function that calls itself either directly or indirectly through another function. It defines a problem in terms of smaller version of itself.

- Every recursive definition must have one (or more) base cases.
- The general case must eventually reduce to a base case.
- The base case stops the recursion

Flow chart for Recursion

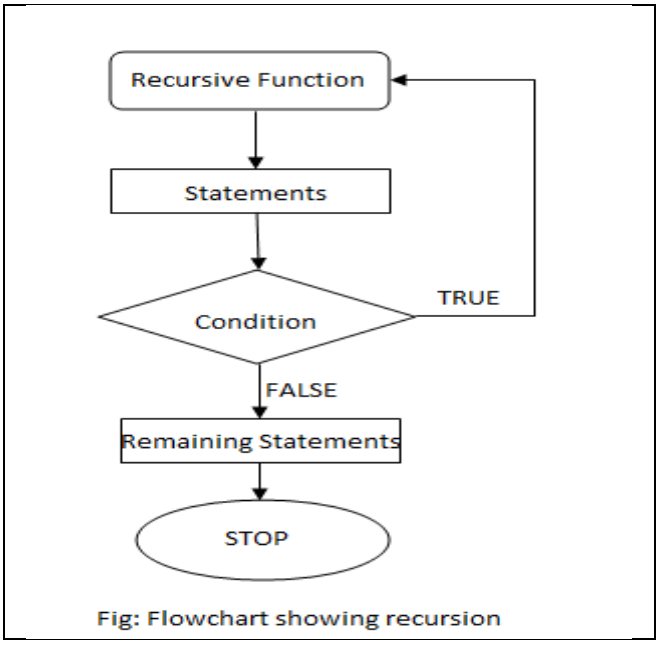

Fig: Flowchart showing recursion

TYPES OF RECURSION

There are two types of recursion, direct recursion and indirect recursion.

1. Direct Recursion

A function when it calls itself directly is known as Direct Recursion.

Example of Direct Recursion

```
#include<stdio.h>

int factorial (int n)
{
   if (n==1 || n==0)
      return 1;
   else
      return n*factorial(n-1);
}

int main()
{
          int f = factorial(5);
          printf("%d",f);
}
```

2. Indirect Recursion

A function is said to be indirect recursive if it calls another function and the new function calls the first calling function again.

Example of Indirect Recursion

```
#include<stdio.h>

int func1(int);
int func2(int);
int func1(int n)
{
   if (n<=1)
      return 1;
   else
      return func2(n);
}
int func2(int n)
{
   return n*func1(n-1);
}

int main()
{
```

```
        int f = func1(5);
        printf("%d",f);
}
```

Here, recursion takes place in 2 steps, unlike direct recursion.

- First, func1 calls func2
- Then, func2 calls back the first calling function func1.

DISADVANTAGES OF RECURSION

- Recursive programs are generally slower than non-recursive programs. This is because, recursive function needs to store the previous function call addresses for the correct program jump to take place.
- Requires more memory to hold intermediate states. It is because, recursive program requires the allocation of a new stack frame and each state needs to be placed into the stack frame, unlike non-recursive (iterative) programs.

Explanation: How recursion works?