Functions in C

ITM(SLS) Baroda University, Vadodara

Lecture Outline

- Concepts of Library functions and user defined functions
- Function prototype
- Definition of function
- Function parameters
- Parameter passing
- Calling a function
- Different Categories/Types of functions
- Recursive function
- Macros

Concept of Library functions and user defined functions

- A function is a self contained block of statements that perform a specific task or set of specific tasks.
- Any ‘C’ program contains at least one function i.e main().
- There are two types of functions

    1.Library functions -The functions which are declared in the C header files such as scanf(), printf(), getchar(), putchar(),etc.

2. User defined functions-The functions defined by programmers. These functions reduce complexity and optimize the code. These functions introduce the modularity in the program.

Function prototype(Function Declaration)

A function **declaration** tells the compiler about a function's name, return type, and parameters.

The syntax of function declaration

returntype functionname(argument 1,argument 2,.....,argument n);

For example:

int minimum(int num1,int num2);

int minimum(int,int);

float percentage(int marks, int subject_total);

Definition of function

- A function **definition** provides the actual body of the function.
- The general form of function definition is

```
returntype function_name(parameter list)
{
   body of the function;

}
```

Function definition

```
 int max(int num1,int num2)
{
        int result;
        if(num1>num2)
                return(num1);
        else
                return(num2);
}
```

Function Calling

- While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.
- When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.
- To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value

C program for addition of two numbers using a function

```
#include<stdio.h>
long addition(long a, long b)//function definition(dummy arguments)
{
   long result;
   result = a + b; //result=3+6=9
   return result;
}
int main()
{
   long first, second, sum;
   scanf("%ld%ld", &first, &second); //first=3,second=6
   sum = addition(first, second);//function call(actual arguments)
   printf("%ld\n", sum);
   return 0;
}
```

Different Categories of functions

A function may or may not accept any argument. It may or may not return any value. Based on these facts, There are four different aspects of function calls.

- function without arguments and without return value
- function without arguments and with return value
- function with arguments and without return value
- function with arguments and with return value

Function without arguments and without return value

```
#include<stdio.h>
void display(); //function prototype
void main ()
{
    printf("Hello\t ");
    display();  //function call
}
void display()
{
    printf(“\nB.Tech AI/Cyber Security Students");
}
```

Output

Hello

B.Tech AI/Cyber Security

Function without arguments and without return value

```
#include<stdio.h>
void sum();
void main()
{
   printf("\n For calculating the sum of two numbers:");
   sum();
}
void sum()
{
   int a,b;
   printf("\nEnter two numbers");
   scanf("%d %d",&a,&b);
   printf(“\nThe sum is %d",a+b);
}
```

Output:

For Calculating the sum of two numbers

Enter two numbers 10 20

The Sum is 30

Function without argument and with return value

```
#include<stdio.h>
int sub();
void main()
{
   int result;
   printf("\nFor calculating the Subtraction of two numbers:");
   result = sub(); //function call
   printf(“The Subtraction of two numbers is %d",result);
}
int sub()
{
   int a,b;
   printf("\nEnter two numbers");
   scanf("%d %d",&a,&b);
   return(a-b);
}
```

Output

For calculating the Subtraction of two numbers

Enter two numbers 20 10

The Subtraction of two numbers is 10

Function without argument and with return value

program to calculate the area of the square

```
#include<stdio.h>
float  square();
void main()
{
   printf(“To calculate the area of the square\n");
   float area = square(); //function call
   printf("The area of the square: %f\n",area);
}
float square()
{
   float side;
   printf("Enter the length of the side in meters: ");
   scanf("%f",&side);
   return side * side;
}
```

Output:

To calculate the area of the square

Enter the length of the side in meters:20

The area of the square: 400.000000

Function with argument and with return value

```
#include<stdio.h>
int sum(int, int);  //function protoype
void main()
{
   int a,b, addition;
   printf("\nTo calculate the sum of two numbers:");
   printf("\nEnter two numbers:");
   scanf("%d %d",&a,&b);
   addition = sum(a,b);  //function call-actual arguments
   printf("\nThe sum is : %d",addition);
}
int sum(int a, int b)  //dummy or formal arguments
{
   return(a+b);
}
```

Output

To calculate the sum of two numbers:

Enter two numbers:20 40

The sum is:60

Function with argument and without return value

```
#include<stdio.h>
void sum(int, int);
void main()
{
   int a,b,result;
   printf("\nTo calculate the sum of two numbers:");
   printf("\nEnter two numbers:");
   scanf("%d %d",&a,&b);
   sum(a,b);  //actual arguments
}
void sum(int c, int d)  //dummy arguments or formal arguments
{
   printf("\nThe sum is %d",a+b);
}
```

Output:

To calculate the sum of two numbers:

Enter two numbers:20 40

The sum is 60

Recursive Function

A function which calls itself is called recursive function.

A function makes number of recursive calls to itself.

A recursive function must **include base condition**.

Base condition tells when the recursion has to stop.

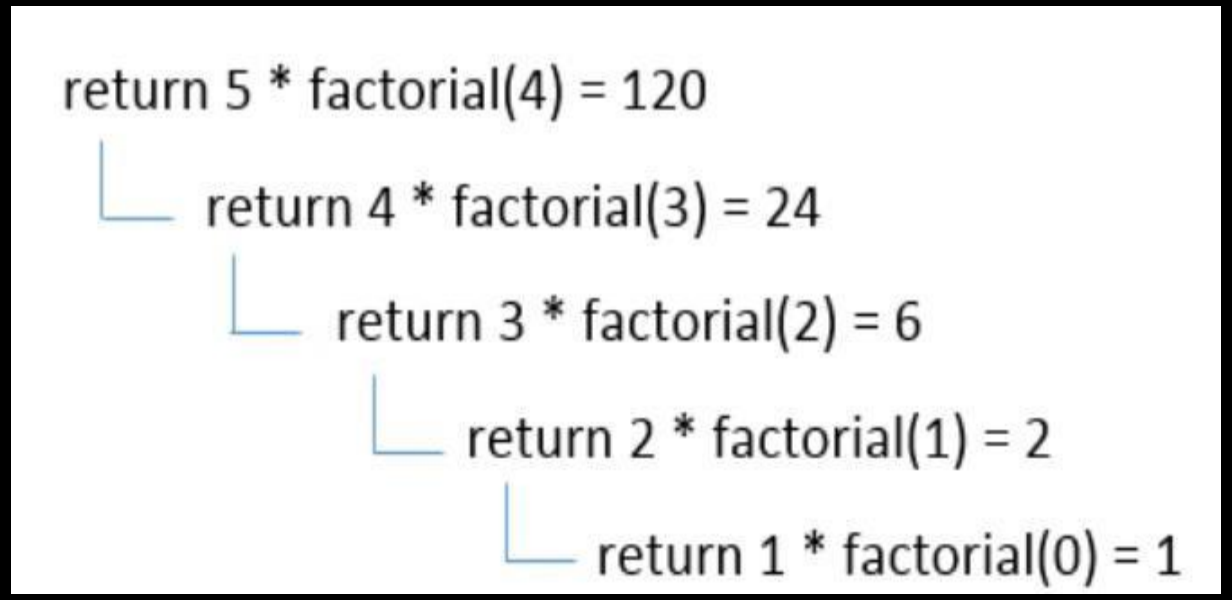
```
void   first()
{
   first();//function calls itself
}
  void   main()
{
   first();
}
```

C program to calculate factorial using recursion

```
#include <stdio.h>
long int factorial(int n)
{
   if(n==1)
      {
            return 1;
      }
   return n*factorial(n-1);
}
int main()
{
   int num;
   long int fact=0;

   printf("Enter an integer number: ");
   scanf("%d",&num);

   fact=factorial(num);
   printf("Factorial of %d is = %ld",num,fact);
   printf("\n");
   return 0;
}
```



Fibonacci Series using Recursion in C

```c
#include <stdio.h>

int fibonacci(int i) {

   if(i == 0) {
      return 0;
   }

   if(i == 1) {
      return 1;
   }
   return fibonacci(i-1) + fibonacci(i-2);
}

int  main() {

   int i;

   for (i = 0; i < 10; i++) {
      printf("%d\t\n", fibonacci(i));
   }

   return 0;
}
```

Ackermann Function

- The Ackermann function, named after **Wilhelm Ackermann**,
- The simplest and earliest-discovered examples of a total computable function

$$A(m,n) = \begin{cases} n + 1 & \text{if } m=0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1,A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

where m and n are non-negative integers

C Program to implement Ackermann function using recursion

```
#include<stdio.h>
int A(int , int );//function prototype
 void main()
{
        int m,n;
        printf("Enter two numbers :: \n");
        scanf("%d%d",&m,&n);
        printf("\nOUTPUT :: %d\n",A(m,n));//function call
}

int A(int m, int n)
{
if(m==0)
        return n+1;
else if(n==0)
        return A(m-1,1); //Recursive function call
else
        return A(m-1,A(m,n-1));//Recursive function call
},
```