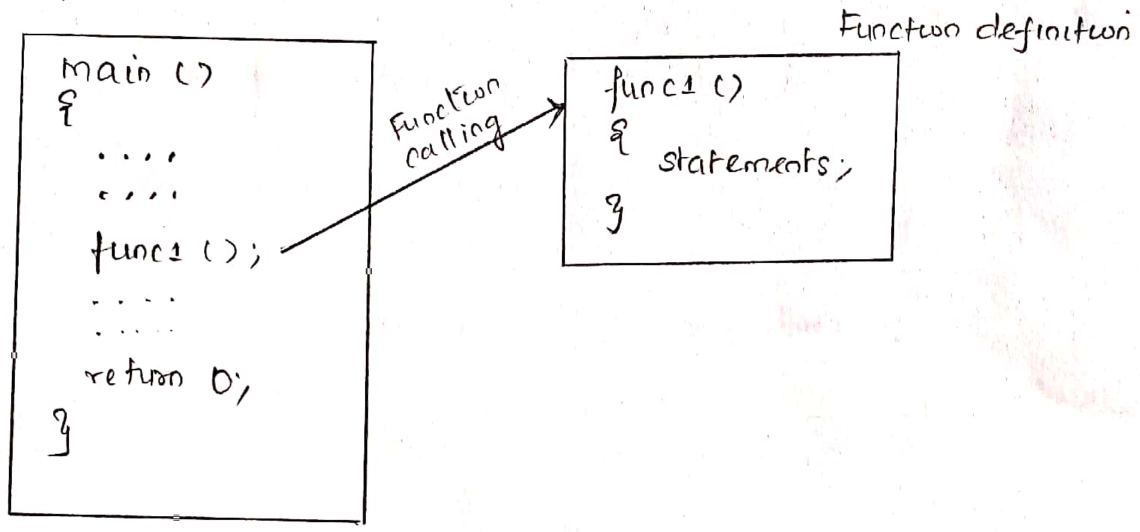
CS8251- PROGRAMMING IN C

UNIT - III.

1. Explain the elements of function with example.

- The C Language enables the programmers to break a program into segments.
- These Segments are called as functions. Each function can be written independently.
- Every function in a program can be designed to perform a specific task.
- The following diagram illustrates how the main function calls the sub function.



The Three Elements of function are

a. Function Declaration
b. Function calling
c. Function Definition.

Function Declaration:

- The functions must be declared before we use it in the program.
- It should be done outside the main function.

Syntax:

$$\text{returntype functionname (}\underbrace{\text{datatype v1, datatype v2, ..., datatype vn}}_{\text{Parameters.}}\text{);}$$

Parameters otherwise called as arguments carries input to the function.

Parameters are of two types:

a. Actual parameter - Parameters used in the function call are called as actual parameters

b. Formal parameter - Parameters used in the function definition are called as formal parameters.

Function Calling :

- In order to execute, we have to include a reference inside the main function because the computer searches for the ~~main~~ sub function and executes.
- This reference is called as Function Calling.
- whenever the control reaches to this function call, automatically the controll will be redirected to the function definition in the program.
  - → All statements will be executed in sequence.
- when the control reaches the close curly braces, again the control will back to the original position from where it jumped and normal execution will be followed.

Syntax :

```
function_name (list of parameters);
```

Function Definition :

- This is the place where the actual logic is written.

Syntax:

```
returntype function-name (list of parameters
{
    // local variable;
    // executable statements;
    // return fn;
}
```

Eg:

```
#include <stdio.h>
int add (int, int);          ⟶ Function declaration
int main()
{
  int a, b, sum = 0;
  printf ("Enter values for a and b");
  scanf ("%d %d", &a, &b);
  sum = add (a,b);           ⟶ Function call
  printf ("The sum is %d", sum);
}
int add (int x, int y)  ]
{                       |  Function definition.
  int result ;          |
  result = x+y ;        |
}  return result;       ]
```

2. Explain Function Prototyping in detail.

Based on the arguments and return values, there are four categories of function.

Function with arguments with return value	Function with arguments and without return value	Function without arguments and with return value	Function without arguments and without return value.

#include<stdio.h>
int add (int, int);
int main ()
{
int a, b, sum= 0;
printf (" Enter a and b");
Scanf ("%d%d", &a, &b);
sum= add (a, b);
printf ("%d", sum);
}
int add (int (x, y)
{
int result;
result= x+y;
return result;
}
	#include <stdio.h>
	int add (int, int);
	void main ()
	{
	int a, b, sum= 0;
	printf (" Enter a and b");
	scanf ("% %d", &a, &b);
	add (a, b);
	}
	void add (int x, int y)
	{
	int result = 0;
	result = x+y;
	printf (" Sum = %d", result);
	}
		#include<stdio.h>
		int main ()
		{
		int sum= 0;
		sum= add();
		printf (" The sum is %d", sum);
		}
		int add ()
		{
		int a, b, c;
		printf ("Enter a and b");
		scanf ("%d%d", &a, &b);
		c= a+b;
		return c;
		}
			#include<stdio.h>
			void main ()
			{
			add ();
			}
			void add ()
			{
			int a, b, c;
			printf (" Enter a and b");
			scanf ("%d%d", &a, &b);
			c= a+b;
			printf ("The sum is %d", c);
			}

• Here the line sum= add(a,b) indicates the function call. • In the function definition, the return result statement, returns an integer value to the main function. We use argument as well as return value in this category.	Here the line add(a,b) indicates the function call which transfers the control to the function definition. Instead of using return function, the result is directly printed in the ~~sub~~ function. Thus we use arguments and not return function in this category.	Here the line sum = add () indicates the function call which has no arguments. The control is moved to the function definition in which the addition process is carried out. The return statement returns an integer value to the main function. Here we do not pass arguments but we use return value.	Here the line add() indicates the function call which transfers the control to the function definition. Instead of using return function, the values are read, addition is performed and the result is displayed in the function definition. Here we do not use the arguments as well as return value.

3. Explain Parameter Passing Methods in detail

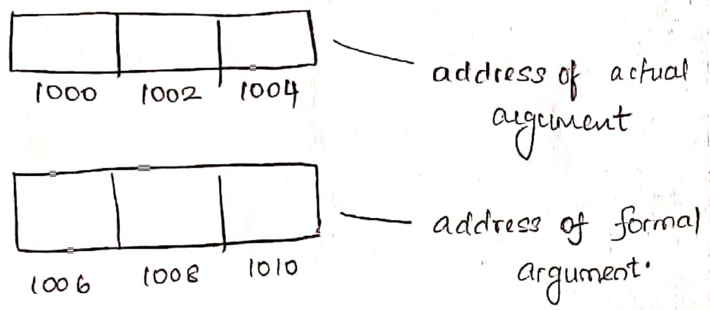
When a function is called, the calling function may have to pass some values to the called function. There are two ways in which arguments or parameters can be passed to the called function.

a. Call by value

b. Call by reference.

Call by value

- In this method, the value of actual parameters will be directly copied to the formal argument.



- The address or memory occupied by the actual parameters is different from memory occupied by formal parameters.
- The manipulations which are done in the function definition will not affect the input or the parameters which are written in the function call.
- Actual parameters will be duplicated and copied into the formal parameters.
- Changes done in formal parameter will not show any impact on actual parameter.

Example :

```
#include <stdio.h>
void swap (int, int);
void main ()
{
   int a = 20, b= 30;
   swap (a,b);
   printf ("a= %d", a);
   printf ("b= %d", b);
}
void swap (int x, int y)
{
     int temp; temp = x; x = y; y = temp;
     printf (" x = %d", x);
     printf ("y= %d", y);
}.
```

"Call By Reference :

- In this method, we will not copy the actual parameters to the formal parameters.
- Instead we just pass the address of variables on a parameter to the function defenetion in the function call.
- In the formal arguments of the function defenction, we will access the values and address which is passed by actual parameters. These values are used to perform the manipulation. The values are not duplicated here.
- The address will be shared by the actual parameters and formal parameters.
- Inorder to access the address of the variable we have to use the address operator (&) and to get the value we have to use * operation.
- The manipulation or changes done in the formal parameter will have an impact on the actual parameter; They will also share the same address.

Example :

```
#include <stdio.h>
void swap (int *, int *)
 void main ()
 {
     int a = 20, b=30;
     swap (&a, &b);
     printf (" a= %d", a);
     printf ("b = %d", b);
 }
 void swap (int &x, int &y)
 {
     int temp;
     temp=x;
     x=y;
     y= temp;
     printf (" x = %d", x);
     printf ("y= %d", y);
 }.
```

4. Explain Recursive Function in detail.

Recursive Function is a function that calls itself again and again until the condition is satisfied.

Example Program using Recursive function.

```
#include <stdio.h>
#include <conio.h>
int findFactorial (int);
void main ()
{
   int i, factorial, num;
   printf (" Enter a number");
   scanf ("%d", &num);
   factorial = findFactorial(num);
   printf (" Factorial of %d is %d", num, factorial);
   getch();
}
int findFactorial (int num)
{
   int f;
   if  (num == 1)
      return 1;
   else
      f = num * findFactorial (num-1);
      return(f);
}
```

5. Explain the concept of pointers with programming example.

- Pointer is a variable that contains the memory location of another variable.
- Therefore pointer is a variable that represents the location of a data item, such as variable or an array element.

Pointers are useful in the following applications.

→ To pass information back and forth between a function and its reference point.

→ Enable programmers to return multiple data items from a function through the function arguments.

```
      {
        T= array[J]
        array[J]= array[J+1];
        array[J+1]= T;
      }}}
    for (i=0; i<5; i++)
    {
       printf ("%s\t", array[i]);
    }
    getch();
  }.
```

```
        {
            T=   Array [j];
            array [j]=  array [j+1];
            array [j+1] = T ;
          }
       }
    }
    for (i=0;  i<5;  i++)
    {
         printf  ("%s\t",  array[i]);
    }
    getch ();
}
```

Scanned with CamScanner