MARUDHAR KESARI JAIN COLLEGE for WOMEN

PG DEPARTMENT OF COMPUTER APPLICATIONS

CLASS : I-BCA

SUBJECT NAME: STRUCTURED PROGRAMMING LANGUAGE IN C

SUBJECT CODE: 23UFC14

SYLLABUS

: **Unit: 4**

Functions: The form of C functions, Return values and types, calling a function, categories of functions, Nested functions, Recursion, functions with arrays, call by value, call by reference, storage classes-character arrays and string functions

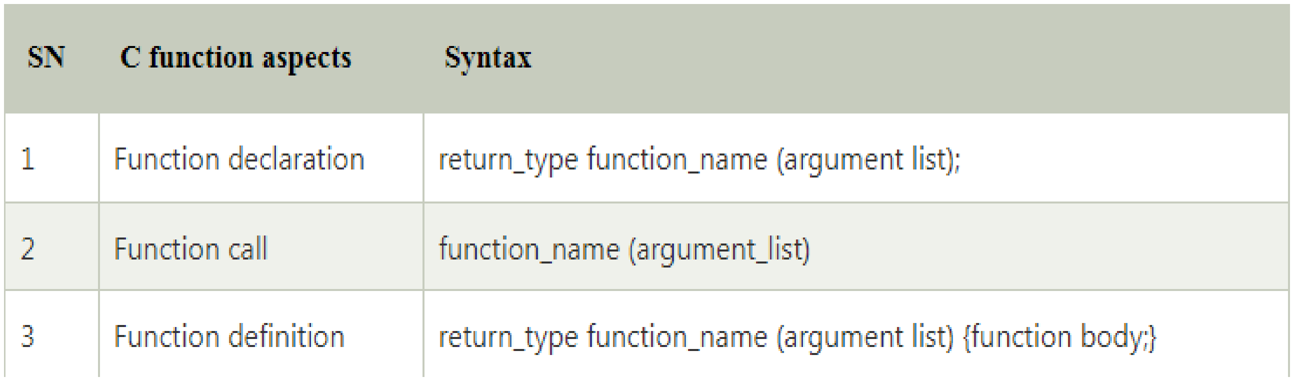
DEFINE FUNCTIONS:

A function is a group of statements that together perform a task. Every C program has at least one function, which is main(), and all the most trivial programs can define additional functions. You can divide up your code into separate functions.

Function Aspects

There are three aspects of a C function.

- **Function declaration** A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.
- **Function call** Function can be called from anywhere in the program. The parameter list must not differ in function calling and function declaration. We must pass the same number of functions as it is declared in the function declaration.
- **Function definition** It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called. Here, we must notice that only one value can be returned from the function.



SN	C function aspects	Syntax
1	Function declaration	return_type function_name (argument list);
2	Function call	function_name (argument_list)
3	Function definition	return_type function_name (argument list) {function body;}

Syntax for function:

```
return_type function_name(data_type parameter...){
//code to be executed
}
```

Calling a Function

After writing a function in C, we have to call this function to perform the task defined inside function body. We cannot execute the code defined inside function's body unless we call it from another function.

When a function(calling function) calls another function(called function), program control is transferred to the called function. A called function performs specific task defined in functions body and when called function terminates either by return statement or when its function-ending closing brace is reached, program control returns back to the calling function.

Syntax to Call a Function

function_name(argument1 ,argument2 ,...);

example :

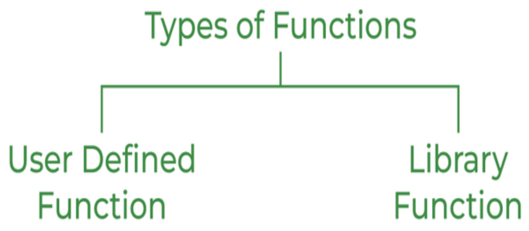
```
#include <stdio.h>
int add(int a, int b);
void main()
{
  int sum;
int a, b;
printf(" Enter the first and second number \n");
scanf("%d %d", &a, &b);
sum = add(a, b); // call add() function
printf( "The sum of the two number is %d", sum);
}
int add(int n1, int n2) // pass n1 and n2 parameter
{
int c;
c = n1 + n2;
return c;
}
```

Output:

```
Enter the first and second number
5
6
The sum of the two number is 11
```

Types of Functions
There are two types of functions in C:

1. Library Functions
2. User Defined Functions



1. ***Library Function***

A <u>library function</u> is also referred to as a "built-in function"**. A compiler package already exists that contains these functions, each of which has a specific meaning and is included in the package. Built-in functions have the advantage of being directly usable without being defined, whereas user-defined functions must be declared and defined before being used.**

For Example:
pow(), sqrt(), strcmp(), strcpy() etc.

Advantages of C library functions

- C Library functions are easy to use and optimized for better performance.
- C library functions save a lot of time i.e, function development time.
- C library functions are convenient as they always work.

EXAMPLE:

```
#include <math.h>
#include <stdio.h>
int main()
{
  double Number;
  Number = 49;
 double squareRoot = sqrt(Number);
  printf("The Square root of %.2lf = %.2lf",
  Number, squareRoot);
  return 0;
}
```

OUTPUT:

```
The Square root of 49.00 = 7.00
```

2. **User Defined Function:**

Functions that the programmer creates are known as User-Defined functions or **"tailor-made functions"**. User-defined functions can be improved and modified according to the need of the programmer. Whenever we write a function that is case-specific and is not defined in any header file, we need to declare and define our own functions according to the syntax.

Advantages of User-Defined Functions

- Changeable functions can be modified as per need.
- The Code of these functions is reusable in other programs.
- These functions are easy to understand, debug and maintain.

Example:

```
#include <stdio.h>
int sum(int a, int b)
{
  return a + b;
}
int main()
{
  int a = 30, b = 40;
  int res = sum(a, b);
  printf("Sum is: %d", res);
  return 0;
}
```

Output:

```
Sum is : 70
```

Nested functions in C

Nesting happens when one or more functions are utilized inside another function in the C programming language. A function cannot be defined inside of another function in the C programming language (nested function is not supported by C language).

Example:

```
#include<stdio.h>
int my_fun()
 {
printf("check_fun function");
printf("\n");
 }
int main()
 {
my_fun();
printf("Main Function\n");
printf("Done");
 }
```

Output:

```
Check fun function
Main Function
Done
```

Recursion

A function that calls a copy of itself is called Recursive Function. The recursive functions contain a call to themselves somewhere in the function body. Moreover, such functions can contain multiple recursive calls

Basic Structure of Recursive Functions
The basic syntax structure of the recursive functions is:

```
type function_name (args) {
   // function statements
   // base condition
   // recursion case (recursive call)
}
```

Example:

```
#include <stdio.h>
int nSum(int n)
{
   if (n == 0) {
      return 0;
   }
   int res = n + nSum(n - 1);
   return res;
}
int main()
{
   int n = 5;
   int sum = nSum(n);
   printf("Sum of First %d Natural Numbers: %d", n, sum);
   return 0;
}
```

Output:

```
Sum of First 5 Natural Numbers: 15
```

Call By Value
In call by value method of parameter passing, the values of actual parameters are copied to the function's formal parameters.

- There are two copies of parameters stored in different memory locations.
- One is the original copy and the other is the function copy.
- Any changes made inside functions are not reflected in the actual parameters of the caller.

Example:

```
#include <stdio.h>
void swapx(int x, int y);
int main()
{
   int a = 10, b = 20;
   swapx(a, b); // Actual Parameters
  printf("In the Caller:\na = %d b = %d\n", a, b);
return 0;
}
void swapx(int x, int y) // Formal Parameters
{
   int t;
   t = x;
   x = y;
   y = t;
   printf("Inside Function:\nx = %d y = %d\n", x, y);
}
```

```
Output:
Inside Function:

x = 20 y = 10

In the Caller:

a = 10 b = 20
```

Call by Reference
In call by reference method of parameter passing, the address of the actual parameters is passed to the function as the formal parameters.

- Both the actual and formal parameters refer to the same locations.
- Any changes made inside the function are actually reflected in the actual parameters of the caller.

Example:

```
#include <stdio.h>
void swapx(int*, int*);
 Main function
int main()
{
   int a = 10, b = 20;
    swapx(&a, &b); // Actual Parameters
    printf("Inside the Caller:\na = %d b = %d\n", a, b);
    return 0;
}
 void swapx(int* x, int* y) // Formal Parameters
{
   int t;

   t = *x;
   *x = *y;
   *y = t;
   printf("Inside the Function:\nx = %d y = %d\n", *x, *y);
}
```

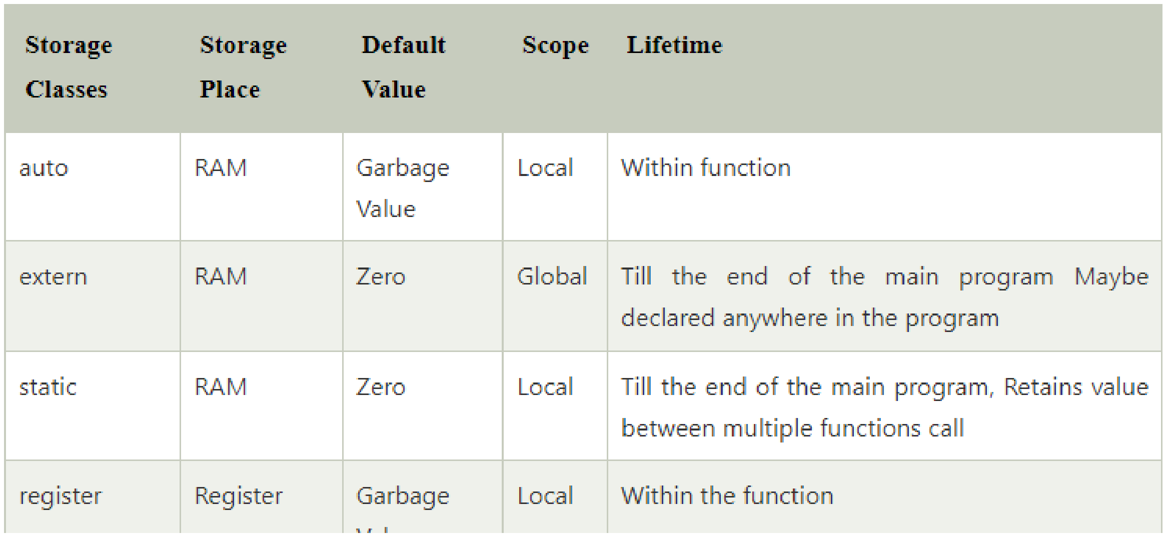
```
Output:
Inside the Function:

x = 20 y = 10

Inside the Caller:

a = 20 b = 10
```

Storage Classes in C

Storage classes in C are used to determine the lifetime, visibility, memory location, and initial value of a variable. There are four types of storage classes in C

- Automatic

- External
- Static
- Register



Storage Classes	Storage Place	Default Value	Scope	Lifetime
auto	RAM	Garbage Value	Local	Within function
extern	RAM	Zero	Global	Till the end of the main program Maybe declared anywhere in the program
static	RAM	Zero	Local	Till the end of the main program, Retains value between multiple functions call
register	Register	Garbage	Local	Within the function

1. Automatic:

- ✓ Automatic variables are allocated memory automatically at runtime.
- ✓ The visibility of the automatic variables is limited to the block in which they are defined.
- ✓ The scope of the automatic variables is limited to the block in which they are defined.
- ✓ The automatic variables are initialized to garbage by default.
- ✓ The memory assigned to automatic variables gets freed upon exiting from the block.
- ✓ The keyword used for defining automatic variables is auto.
- ✓ Every local variable is automatic in C by default.

Example:

```
#include <stdio.h>
int main()
{
int a; //auto
char b;
float c;
```

```
Output:
garbage garbage garbage
```

```
printf("%d %c %f",a,b,c);
return 0;
}
```

Static

- ✓ The variables defined as static specifier can hold their value between the multiple function calls.
- ✓ Static local variables are visible only to the function or the block in which they are defined.
- ✓ A same static variable can be declared many times but can be assigned at only one time.
- ✓ Default initial value of the static integral variable is 0 otherwise null.
- ✓ The visibility of the static global variable is limited to the file in which it has declared.
- ✓ The keyword used to define static variable is static.

Example

```
#include<stdio.h>
static char c;
static int i;
static float f;
static char s[100];
void main ()
{
printf("%d %d %f %s",c,i,f); // the initial default value of c, i, and f will be printed.
}
```

Output:

```
0 0 0.000000 (null)
```

REGISTER

- ✓ The variables defined as the register is allocated the memory into the CPU registers depending upon the size of the memory remaining in the CPU.
- ✓ We can not dereference the register variables, i.e., we can not use &operator for the register variable.
- ✓ The access time of the register variables is faster than the automatic variables.
- ✓ The initial default value of the register local variables is 0.

- ✓ The register keyword is used for the variable which should be stored in the CPU register. However, it is compiler?s choice whether or not; the variables can be stored in the register.
- ✓ We can store pointers into the register, i.e., a register can store the address of a variable.
- ✓ Static variables can not be stored into the register since we can not use more than one storage specifier for the same variable.

Example

```
#include <stdio.h>
int main()
{                                    output:
register int a;                            0
printf("%d",a);
}
```

EXTERNAL

- ✓ The external storage class is used to tell the compiler that the variable defined as extern is declared with an external linkage elsewhere in the program.
- ✓ The variables declared as extern are not allocated any memory. It is only declaration and intended to specify that the variable is declared elsewhere in the program.
- ✓ The default initial value of external integral type is 0 otherwise null.
- ✓ We can only initialize the extern variable globally, i.e., we can not initialize the external variable within any block or method.
- ✓ An external variable can be declared many times but can be initialized at only once.
- ✓ If a variable is declared as external then the compiler searches for that variable to be initialized somewhere in the program which may be extern or static. If it is not, then the compiler will show an error.

Example:

```
extern int a;
int a = 10;
#include <stdio.h>
int main()
{
printf("%d",a);
}
int a = 20; // compiler will show an error at this line
```

Output:

```
compile time error
```

Advantages of Functions in C

Functions in C is a highly useful feature of C with many advantages as mentioned below:

1. The function can reduce the repetition of the same statements in the program.
2. The function makes code readable by providing modularity to our program.
3. There is no fixed number of calling functions it can be called as many times as you want.
4. The function reduces the size of the program.
5. Once the function is declared you can just use it without thinking about the internal working of the function.

Disadvantages of Functions in C

The following are the major disadvantages of functions in C:

1. Cannot return multiple values.
2. Memory and time overhead due to stack frame allocation and transfer of program control.