MODULE 1

Syllabus

	Description	**Duration (Hours)**	**Cognitive Level**
CO1	Make use of the basic programming concepts – sequential, conditional, unconditional, looping structures and functions in C.		
M1.01	Summarize the basic programming concepts in C – sequential, conditional, unconditional and control structures in C	2	Understanding.
M1.02	Explain the concept of preprocessing	1	Understanding
M1.03	Summarize the concepts of modular programming concepts in C	1	Understanding
M1.04	Develop programs using functions	1	Applying
M1.05	Explain Storage class, Lifetime and Visibility of Variables	1	Understanding.
M1.06	Develop programs using the concepts of storage class and scope rules	2	Applying
M1.07	Illustrate the recursion with examples	1	Understanding.
M1.08	Develop programs using recursion.	2	Applying.

Contents:
Recall basic programming concepts – C program structure, selection structure and repetition structures.
Function – Declarations, prototype, definition, function call, storage class, lifetime and visibility of variables.
Preprocessor – file inclusion – macro substitution
Recursion – Recursive definition of a problem, Implementation of programs using recursion

Recall basic programming concepts

Recall basic programming concepts – C program structure, selection structure and repetition structures

Basic structure of C program

A C program is divided into different sections. There are six main sections to a basic c program.

The six sections are,

Document section
Link section or preprocessor section
Definition section
Global declaration section

Main function
{
        Declaration part
        Body part
}

Subfunction1
{
        Declaration partA C program is divided into different sections. There are six main sections to a basic c program.
 	
The six sections are,
 	
Documentation
Link
Definition
Global Declarations
Main functions
Subprograms
        Body part
}
Subfunction2
Subfunction3
………………
………………

Documentation Section: The documentation section is the part of the program where the programmer gives the details associated with the program. He usually gives the name of the program, the details of the author and other details like the time of coding and description. It gives anyone reading the code the overview of the code.

Example

```
/**
* File Name: Helloworld.c
* Author: Manthan Naik
* date: 09/08/2019
* description
*/
```

Link Section This part of the code is used to declare all the header files that will be used in the program. This leads to the compiler being told to link the header files to the system libraries.

Example

```
#include<stdio.h>
```

Definition Section: In this section, we define different constants. The keyword define is used in this part.

Example

```
#define PI=3.14
```

Global Declaration Section: This part of the code is the part where the global variables are declared. All the global variable used are declared in this part. The user-defined functions are also declared in this part of the code.

Example:

```
float area(float r);

int a=7;
```

Main Function Section: Every C-programs needs to have the main function. Each main function contains 2 parts. A declaration part and an Execution part. The declaration part is the part where all the variables are declared. The execution part begins with the curly brackets and ends with the curly close bracket. Both the declaration and execution part are inside the curly braces.

Example

```
int main(void)
{
int a=10;
printf(" %d", a);
return 0;
}
```

Sub Program Section: All the user-defined functions are defined in this section of the program.

```
int add(int a, int b)
{
return a+b;
}
```

Selection structure in c/Decision structure in c

- Used to choose among alternative courses of action.
- Used to control flow of program
- C has mainly three:if,if..else,nexted if, else..if ladder and switch.

If statement:

- **One-way selection**
- Single entry/single exit
- Syntax

```
if <condion>
        statement 1;
```

If….else statement:

- **Two way selection**
- Specifies an action to be performed on both condition,when it is true then perform statement 1 and when it is false then perform statement 2.

```
if <condion>
        statement 1;
else
        Statement 2;
```

Nexted if statement:

- When on control statement inside another
- Syntax:

```
if <condion 1>
        statement 1;
        if <condition 2 >
                Statement 2;
else
        Statement 3;
```

Else…if ladder:

- **Multi way selection**
- Shifts program control step by step,through a series of statement blocks.
- Syntax:

```
if <condion 1>
            statement 1;
else if<condition 2>
            Statement 2;
else if <condition 3>
            Statement 3;
………
else
            Statement n;
```

The conditional operator:
Syntax:
Condition?statement1:statement 2;

if condition true then execute statement 2 else execute statement3
Example,
a>b?max=a:max=b;
Equivalent to,

```
if a>b
        max=a;
else
        max=b;
```

Switch statement:

- Multi way selection
- Replaces else if ladder
- First expression is evaluated then based on value of expression action is performed.
- Syntax:

```
Switch(expression)
{
case 1:statement 1;
case 2:statement 2;
……
case n:statement n;
default: default statement;
}
```

Repetition structure in C

A program loop is defined as a block of statements, which are frequently executed for certain number of times even though these statements come into view once in a program. This loop is also known as iterative structure or repetitive structure.

for loop statement:

for statement handles all the details such as variable initialisation, variable update and loop condition within single parantheses.

Syntax:

```
for(expression-1;    expression-2;   expression-3)
        Statement;
```

Example,

```
for(i=0;i<3;i++)
        sum=sum+i;
```

While loop statement:

A set of statements execution to continue for as long as a specified expression (condition) is true. When the condition becomes false, the repetition terminates and the first statement after the repetition structure is executed.

```
expression-1;
while(expression-2)
{
        Statement;
        expression-3;
}
```

Example,

```
i=0;
while(i<3)
{
        sum=sum+I;
        I++;
}
```

do…while statement:

The do while loop tests the loop condition at the end of the loop structure or after the body of the loop is executed. Thus in do while loop, the compounded statement will be executed at least once.

Syntax:

```
expression1;
do
{
        statement;
        expression 3;
}while(expression2);
```

Example,

```
i=0;
do
{
        sum=sum+i;
        I++;
}while(i<3);
```

FUNCTION

Function – Declarations, prototype, definition, function call, storage class, lifetime and visibility of variables

A function is a group of statements that together perform a task. Every C program has at least one function, which is main().Excecution starts form main() function.You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.
A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

The C standard library provides numerous built-in functions that your program can call. For example, strcat() to concatenate two strings, memcpy() to copy one memory location to another location, and many more functions.

A function can also be referred as a method or a sub-routine or a procedure, etc.

Defining a Function

The general form of a function definition in C programming language is as follows −

```
return_type function_name( parameter list ) {

  body of the function

}
```

A function definition in C programming 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 known the function signature.
- Parameters − When a function is invoke, 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.

Function Declarations

A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts −

```
return_type function_name( parameter list );
```

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

Calling a Function

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.

Function Arguments

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the formal parameters of the function.

Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function, there are two ways in which arguments can be passed to a function −

Call by value: This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

Call by reference: This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

By default, C uses call by value to pass arguments. In general, it means the code within a function cannot alter the arguments used to call the function.

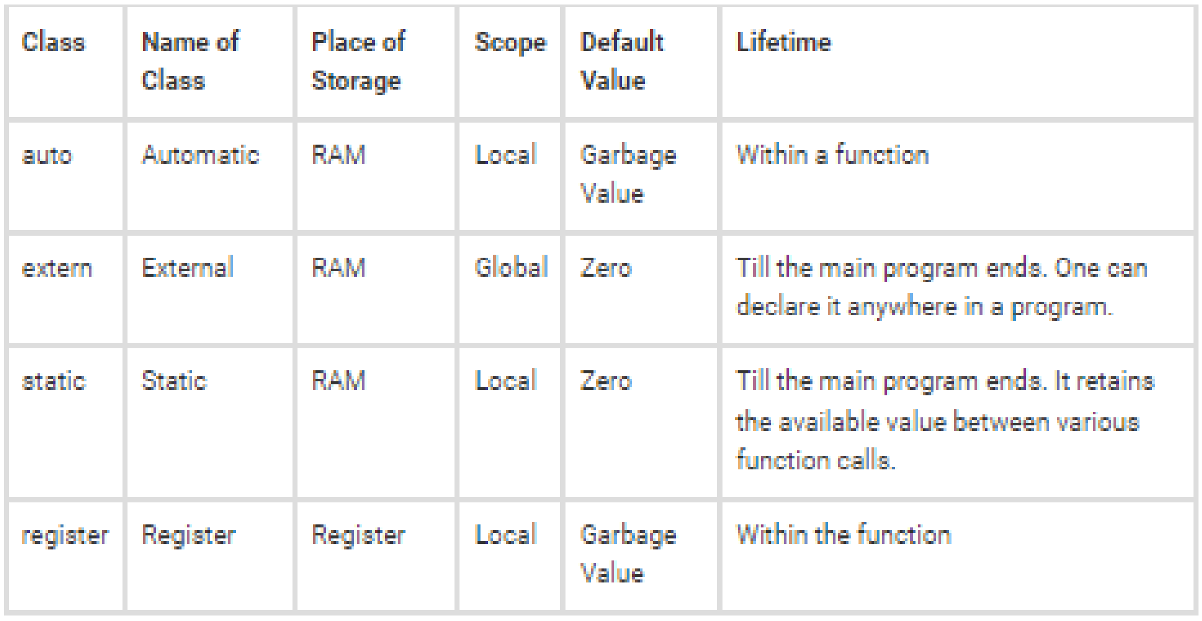
Storage class,Life time and visibility of variables in C

- Storage class- that tells about the lifetime of the variable, visibility of the variable.
- Lifetime– It is the time when variable has an existence in the code while compilation. Like the local variable exists only within a functional block. And after that there value doesn't effect other variable with same name.
- Visibility– It is the range till where the variable is visible like the global variable is visible to all the function in a C program.

Types of Storage Classes in C

There are four different types of storage classes that we use in the C language:

- Automatic Storage Class
- External Storage Class
- Static Storage Class
- Register Storage Class



Class	Name of Class	Place of Storage	Scope	Default Value	Lifetime
auto	Automatic	RAM	Local	Garbage Value	Within a function
extern	External	RAM	Global	Zero	Till the main program ends. One can declare it anywhere in a program.
static	Static	RAM	Local	Zero	Till the main program ends. It retains the available value between various function calls.
register	Register	Register	Local	Garbage Value	Within the function

auto: This is the default storage class for all the variables declared inside a function or a block. Hence, the keyword auto is rarely used while writing programs in C language. Auto variables can be only accessed within the block/function they have been declared and not outside them (which defines their scope).They are assigned a garbage value by default whenever they are declared.

extern:we use it for giving a reference of any global variable which is visible to all the files present in a program. When using the extern storage class, we cannot initialize the variable.

static:when we make a local variable static, it allows the variable to maintain the values that are available between various function calls.

Example,

```
#include<stdio.h>
void sum()
{
static int x = 20;
static int y = 34;
printf("%d %d \n",x,y);
x++;
y++;
}
void main()
{
int a;
for(a = 0; a< 3; a++)
{
sum(); // Given static variables will hold their values between the various function calls.
}
}
```

The output generated here would be:

```
20 34
21 35
22 36
```

register:We use the register storage class for defining the local variables that must be stored in any register, and not in a RAM. It means that the maximum size of this variable is equal to that of the register size.

Example,

```
#include <stdio.h>
int main()
{
register int x; // A variable x has memory allocation in the CPU register. Here, the initial value of x, by default, is 0.
printf("%d",x); }
```

Preprocessor

Preprocessor – file inclusion – macro substitution

The C Preprocessor is not a part of the compiler, but is a separate step in the compilation process.It instructs the compiler to do required pre-processing before the actual compilation.The C preprocessor is a macro preprocessor (allows you to define macros) that transforms your program before it is compiled. These transformations can be the inclusion of header files, macro expansions, etc.

All preprocessing directives begin with a # symbol. For example,

```
#define PI 3.14
```

Some of the common uses of C preprocessors are:

Including Header Files: #include

The #include preprocessor is used to include header files to C programs. For example,

```
#include <stdio.h>
```

Here, stdio.h is a header file. The #include preprocessor directive replaces the above line with the contents of stdio.h header file.

You can also create your own header file containing function declaration and include it in your program using this preprocessor directive.

```
#include "my_header.h"
```

Macros using #define

A macro is a fragment of code that is given a name. You can define a macro in C using the #define preprocessor directive.

Here's an example.

```
#define c 299792458  // speed of light
```

Here, when we use c in our program, it is replaced with **299792458.**

Function like Macros

You can also define macros that work in a similar way as a function call. This is known as function-like macros. For example,

```
#define circleArea(r) (3.1415*(r)*(r))
```

Every time the program encounters circleArea(argument), it is replaced by (3.1415*(argument)*(argument)).

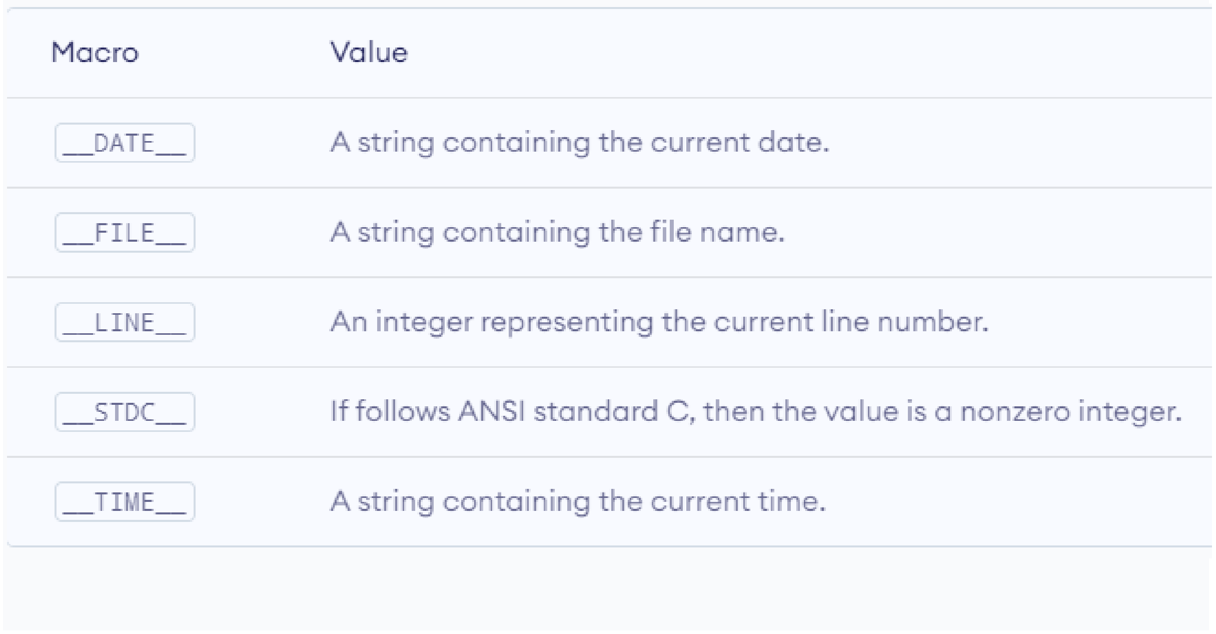
To use conditional, #ifdef, #if, #defined, #else and #elif directives are used.

```
#ifdef MACRO
  // conditional codes
#endif
```

Here, the conditional codes are included in the program only if MACRO is defined.

```
#if expression
   // conditional codes if expression is non-zero
#elif expression1
   // conditional codes if expression is non-zero
#elif expression2
   // conditional codes if expression is non-zero
#else
   // conditional if all expressions are 0
#endif
```

Predefined Macros



Macro	Value
`__DATE__`	A string containing the current date.
`__FILE__`	A string containing the file name.
`__LINE__`	An integer representing the current line number.
`__STDC__`	If follows ANSI standard C, then the value is a nonzero integer.
`__TIME__`	A string containing the current time.

Recursion

Recursion– Recursive definition of a problem, Implementation of programs using recursion

Recursion is the technique of making a function call itself.such function calls are called recursive calls.
 Recursion involves several numbers of recursive calls. However, it is important to impose a termination condition of recursion. Recursion code is shorter than iterative code however it is difficult to understand.
Example ,
Sum of Natural nos

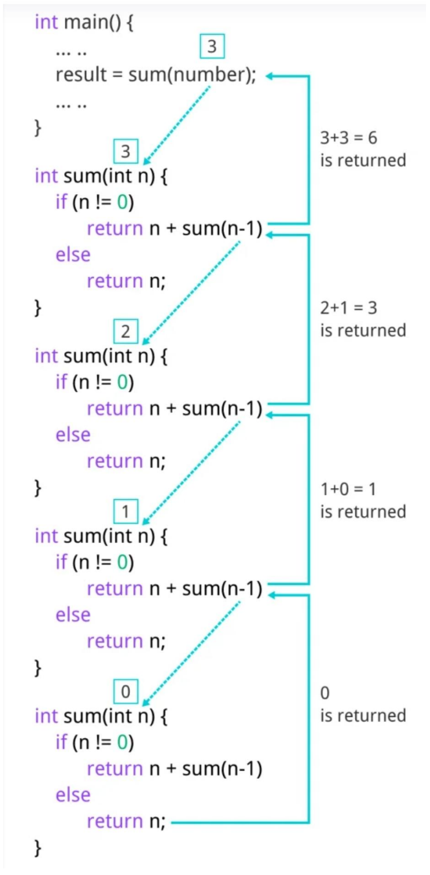
```
#include <stdio.h>
int sum(int n);

int main() {
   int number, result;

   printf("Enter a positive integer: ");
   scanf("%d", &number);

   result = sum(number);

   printf("sum = %d", result);
   return 0;
}

int sum(int n) {
   if (n != 0)
      // sum() function calls itself
      return n + sum(n-1);
   else
      return n;
}
```
Output

```
Enter a positive integer:3
sum = 6
```

Initially, the sum() is called from the main() function with number passed as an argument.

Suppose, the value of n inside sum() is 3 initially. During the next function call, 2 is passed to the sum() function. This process continues until n is equal to 0.

When n is equal to 0, the if condition fails and the else part is executed returning the sum of integers ultimately to the main() function.



```
int main() {
  ... ..
  result = sum(number);
  ... ..
}

int sum(int n) {
  if (n != 0)
     return n + sum(n-1)
  else
     return n;
}

int sum(int n) {
  if (n != 0)
     return n + sum(n-1)
  else
     return n;
}

int sum(int n) {
  if (n != 0)
     return n + sum(n-1)
  else
     return n;
}

int sum(int n) {
  if (n != 0)
     return n + sum(n-1)
  else
     return n;
}
```

Example programs

1.Factorial of a number using recursion

2.Fibinocci using recursion

Pseudocode for writing any recursive function is given below.

```
if (test_for_base)
{
    return some_value;
}
else if (test_for_another_base)
{
    return some_another_value;
}
else
{
    // Statements;
    recursive call;
}
```