GXEST204

PROGRAMMING IN C

MODULE 1

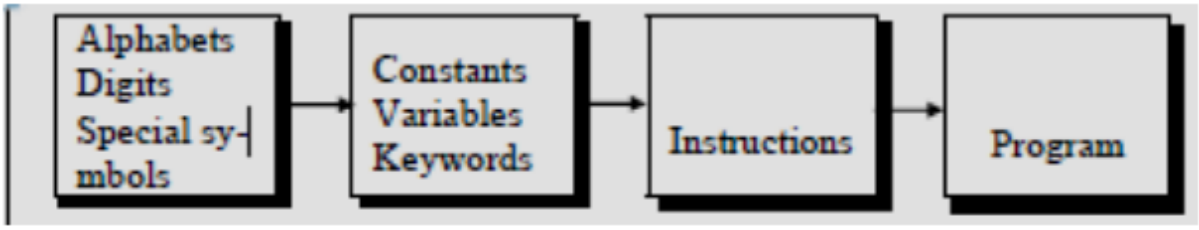
CO1 : Infer a computational problem and develop C programs from themusing basic constructs of C language including the control statements.

C Fundamentals - Character Set, Constants, Identifiers, Keywords, Basic Data types, Variables, Operators and its precedence, Bit-wise operators,Expressions; Statements - Input and Output statements; Structure of a C program; Simple programs. Control Statements - if, if-else, nested if, switch, while, do-while, for, break& continue, nested loops.

INTRODUCTION TO C PROGRAMMING

- C is a programming language developed at AT& T's Bell Laboratories of USA in 1972.
- It was designed and written by Dennis Ritchie.
- C is a procedural oriented programming language. (A task can be divided into a number of smaller parts called functions).

The C Character Set- A character denotes any alphabet, digit or special symbol used to represent information.



Tokens: The words formed from the character set are building blocks of C and are sometimes known as tokens. These tokens represent the individual entity of language.

Examples : Constants, Variables and Keywords

Constants: A constant is an entity that doesn't change whereas a variable is an entity that may change.

Variables: a variable is an entity that may change. Variable names are names given to locations in memory. Variables can store values of any type.

Rules for Constructing Variable Names

- A variable name is any combination alphabets, digits or underscores.
- The first character in the variable name must be an alphabet or underscore.
- No commas or blanks are allowed within a variable name.
- No special symbol other than an underscore

Eg: a,b,_c,a12etc.

C Keywords

- Keywords are the words whose meaning has already defined for the C compiler.
- The keywords cannot be used as variable names
- Eg: int , float, return, void, break etc

Instructions

Instructions in c are the commands in the program that will instruct c compiler to perform specific task.

Data types in c: Specifies the type of data to be stored. This determines the type and size of data associated with variables.

Data types in C can be categorized into 3 types

1. Primitive/Built in/ Pre defined data types
2. Derived data types
3. User defined data types

Primary data types

1. **Primitive Data Types**: This data type includes- integer data type (int), character data type, i.e., character or small integer value (char), floating-point data type, i.e., containing fractional part (float), and double data type (double), which is the same as float data type, but it occupies twice the storage space.

- Integer (int)- can store integer value (1, 255…..) allocates 2 or 4 bytes in memory
- Character(char) - char store a char ‘A’ allocates 1 byte in memory
- Float (float) - can store real values 23.56 allocates 4 bytes in memory
- Double (double) - can store real 245.456 allocates 8 bytes in memory

Derived data types

In c the data types derived from the primitive data types or built in data types are called derived data types. In other words the derived data types are those data types that are created by combining primitive and other data types.

There are three derived data types in c

1. Function
2. Array
3. Pointer

User defined data types

The data types defined by the user them self are called user defined data types. These data types are derived from existing data types.

There are 3 types of user defined data types in c. They are

1. Structure
2. Union
3. Enum

Operators in c

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise, etc.

There are following types of operators to perform different types of operations in C language.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operator
- Misc Operator

1. ***Arithmetic operators*** :-carry out ***fundamental mathematical*** operations. The arithmetic operators in C are as follows:

Addition Operator (+): The addition operator ***adds*** two operands together.

Syntax:result = operand1 + operand2;

Subtraction Operator (-): The second operand is **subtracted** from the first operand via the **subtraction operator**.

Syntax: result = operand1 - operand2;

Multiplication Operator (*): This operator is used to ***multiply*** the two operands.

Syntax: result = operand1 * operand2;

Division Operator (/): The first operand and the second operand are **divided** using the division operator.

Syntax: result = operand1 / operand2;

Modulus Operator (%): The ***modulus operator*** determines the remainder of the division between two operands.

Syntax: result = operand1 % operand2;

2. Relational Operators:

Relational operators assess the relationship between values by comparing them. They return either **true (1)** or **false (0)**. The relational operators in C are as follows:

Equality Operator (==): If two operands are equal, the **equality operator** verifies this.

Inequality Operator (!=): The ***inequality operator*** determines whether two operands are ***equal*** or ***not***.

Greater than Operator (>): The ***greater than operator*** determines if the first operand exceeds the second operand.

Less than Operator (<): The ***less-than operator*** determines if the first operand less is than the second operand.

Greater than or Equal to Operator (>=): The ***greater than or equal to operator*** determines if the first operand is more than or equal to the second operand.

Less than or Equal To Operator (<=): The ***less than or equal to operator*** determines if the first operand must be less than or equal to the second operand.

3. Logical Operators:

Logical operators perform logical operations on ***boolean values*** and return either ***true (1)*** or ***false (0).*** Here are the logical operators in C:

Logical AND Operator (&&): The ***logical AND operator*** returns ***true*** if both operands are ***true***.

Logical OR Operator (||): The ***logical OR operator*** returns ***true*** if at least one of the operands is true.

Logical NOT Operator (!): The ***logical NOT operator*** negates the value of the operand.

4. Bitwise Operators:

Bitwise operators perform operations on individual ***bits*** of the operands. Here are the bitwise operators in C:

Bitwise AND Operator (&): The **bitwise AND operator** performs a ***bitwise AND operation*** on the corresponding bits of the operands.

result = operand1 & operand2;

```
Eg:     unsigned int a = 5;  // 0000 0101 in binary
        unsigned int b = 3;  // 0000 0011 in binary
        int result = a & b;
```

Output

result = 1

Bitwise OR Operator (|): The ***bitwise OR operator*** performs a bitwise OR operation on the corresponding bits of the operands.

```
Eg:     unsigned int a = 5;  // 0000 0101 in binary
        unsigned int b = 3;  // 0000 0011 in binary
        int result = a | b;
```

Output

result = 7

Bitwise XOR Operator (^): The ***bitwise XOR operator*** performs a bitwise exclusive OR operation on the corresponding bits of the operands.

```
Eg:     unsigned int a = 5;  // 0000 0101 in binary
        unsigned int b = 3;  // 0000 0011 in binary
        int result = a ^ b;
```

Output

result = 6

Bitwise NOT Operator (~): The ***bitwise NOT operator*** flips each bit of the operand.

```
Eg:-    unsigned int a = 5;  // 0000 0101 in binary
        int result = ~a;
```

Output

result = - 6

5. Assignment Operator:

Assignment operators are used to assign values to variables. Here is some of the assignment operator in C:

Simple Assignment Operator (=): The ***simple assignment operator*** assigns the value from the ***right*** side operands to the ***left side operands***.

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A

6. **Miscellaneous Operator:**

The ***sizeof operator*** and the ***ternary operator*** fall under the ***miscellaneous operator category***. **sizeof** is one of the mostly used operator in the C. It is a compile-time unary operator which can be used to compute the **size of its operand**

Syntax:sizeof(*Expression*);

The **conditional operator in C** is kind of similar to the if-else statement as it follows the same algorithm as of if-else statement but the conditional operator takes less space and helps to write the if-else statements in the shortest way possible. It is also known as the **ternary operator in C** as it operates on three operands.

Syntax:- *variable = (condition)* **?** *Expression2* **:** *Expression3*

- **Step 1: Expression1** is the condition to be evaluated.
- **Step 2A:** If the condition(**Expression1**) is True then **Expression2** will be executed.
- **Step 2B**: If the condition(**Expression1**) is false then **Expression3** will be executed.
- **Step 3:** Results will be returned.

The precedence of operators determines which operator is executed first if there is more than one operator in an expression.

OPERATOR	TYPE	ASSOCIAVITY
( ) [ ] . ->		left-to-right
++ -- + - ! ~ (*type*) * & sizeof	Unary Operator	right-to-left
* / %	Arithmetic Operator	left-to-right
+ -	Arithmetic Operator	left-to-right
<< >>	Shift Operator	left-to-right
< <= > >=	Relational Operator	left-to-right
== !=	Relational Operator	left-to-right
&	Bitwise AND Operator	left-to-right
^	Bitwise EX-OR Operator	left-to-right
|	Bitwise OR Operator	left-to-right
&&	Logical AND Operator	left-to-right
||	Logical OR Operator	left-to-right
? :	Ternary Conditional Operator	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operator	right-to-left
,	Comma	left-to-right

The associativity of operators determines the direction in which an expression is evaluated

INPUT AND OUTPUT FUNCTIONS IN C

Input and output functions are available in the C language to perform the most common tasks. In every C program, three basic functions take place – accepting of data as input, the processing of data, and the generation of output. In C, input refers to providing it with some information or data to be utilised in the program. On the other hand, output refers to writing

the data and information on any printer file and displaying it on the screen. There are mainly two types of these functions available in the C language:

Formatted

The formatted functions basically present or accept the available data (input) in a specific format. The standard library in C contains various functions for the input-output operations. The scanf() and printf() out of these functions help a programmer format the functions in their desired format

Unformatted

The unformatted functions are not capable of controlling the format that is involved in writing and reading the available data.

The unformatted input-output functions further have two categories:

- The character functions
- The string functions

We use the character input functions for reading only a single character from the input device (the keyboard). On the other hand, we use the character output functions for writing just a single character on the output source (the screen). Here, the getchar() refers to the input functions of unformatted type, while the putchar() refers to the output functions of unformatted type.

In any programming language, the character array or string refers to the collection of various characters. The puts() and gets() are the most commonly used ones for reading and writing strings. Here, the gets() refers to the input function used for reading the string characters while the puts() refers to the output function used for writing the string characters (in unformatted forms).

STRUCTURE OF A C PROGRAM

The basic structure of a C program is divided into 6 parts which makes it easy to read,

modify, document, and understand in a particular format.

1. Documentation
2. Pre-processor Section
3. Definition
4. Global Declaration

5. main ()

6. Sub Programs

Documentation section

It includes the statement specified at the beginning of a program, such as a program's **name, date, description,** and **title**. It is represented as:

```
//name of a program
Or

/*
Overview of the code

*/


.
```

Both methods work as the document section in a program. It provides an overview of the program. Anything written inside will be considered a part of the documentation section and will not interfere with the specified code.

Preprocessor section

The preprocessor section contains all the header files used in a program. It informs the system to link the header files to the system libraries. It is given b

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

The **#include** statement includes the specific file as a part of a function at the time of the compilation. Thus, the contents of the included file are compiled along with the function being compiled. The **#include<stdio.h>** consists of the contents of the standard input output files, which contains the definition of stdin, stdout, and stderr. Whenever the definitions stdin, stdout, and stderr are used in a function, the statement #include<stdio.h> need to be used.

There are various header files available for different purposes. For example, **# include <math.h>.** It is used for mathematic functions in a program.

Define section

The define section comprises of different constants declared using the define keyword. It is given by:

#define a = 2

Global declaration

The global section comprises of all the global declarations in the program. It is given by:

```
float num = 2.54;
 int a = 5;
char ch ='z';
```

We can also declare user defined functions in the global variable section.

Main function

main() is the first function to be executed by the computer. It is necessary for a code to include the main(). It is like any other function available in the C library. Parenthesis () are used for passing parameters (if any) to a function.

The main function is declared as:

```
main()
```

We can also use int or main with the main (). The void main() specifies that the program will not return any value. The int main() specifies that the program can return integer type data.

```
int main()
```

Or

```
void main()
```

Main function is further categorized into **local declarations, statements,** and **expressions.**

Local declarations

The variable that is declared inside a given function or block refers to as local declarations.

Statements

The statements refers to **if, else, while, do, for**, etc. used in a program within the main function.

Expressions

An expression is a type of formula where operands are linked with each other by the use of operators. It is given by:

```
a - b;
a +b;
```

User defined functions

The user defined functions specified the functions specified as per the requirements of the user. For example, color(), sum(), division(), etc.

The program (basic or advance) follows the same sections as listed above.

Return function is generally the last section of a code. But, it is not necessary to include. It is used when we want to return a value. The return function returns a value when the return type other than the void is specified with the function.

Return type ends the execution of the function. It further returns control to the specified calling function. It is given by:

```
return;
```

Or

```
return expression ;
```

Example 1: To find the sum of two numbers given by the user

It is given by:

```
/* Sum of two numbers */
#include<stdio.h>
int main()
{
        int a, b, sum;
        printf("Enter two numbers to be added ");
        scanf("%d %d", &a, &b);
        // calculating sum
         sum = a + b;
        printf("%d + %d = %d", a, b, sum);
        return 0;  // return the integer value in the sum
}
```

CONTROL STATEMENTS OR CONTROL STRUCTURES

- Control statements enable us to specify the flow of program control- ie the order in which the instructions in a program must be executed.
- They make it possible to make decisions, to perform tasks repeatedly or to jump from one section of code to another.

There are three types of control structures.

1. Sequential statements.
2. Branching statements (Selection structures)
3. Looping statements (Repetition structures)

Branching statements -Used to make certain decisions and based on decision the next instructions to be executed can be selected.

- C programming provides the following conditional statements(branching statements)
  1. Simple if
  2. If else

3. Nested if
4. Else if ladder
5. Switch statement

1. Simple if

Use the if statement to specify a block of code to be executed if a condition is true.

Syntax:-if (*condition*)

```
{
    // block of code to be executed if the condition is true;
}
```

Eg:-if (20 > 18)

```
{
    printf("20 is greater than 18");
}
```

2. If else

if the condition is true its body part is executed. Otherwise the else part is executed.

Syntax:- if(condition)

```
{
        Block of statements;
}
 else
{
        Block of statements;
 }
```

Eg:-int time= 20;

```
if (time< 18)
{
  printf("Good day.");
}
else
{
  printf("Good evening.");
}
```

3. Nested if

When a series of decision is required, nested if-else is used. Nesting means using one if-else construct within another one.

Syntax:- if(condition 1)

```
    {
        If(condition 2)
        {
            Block 1;
        }
        Else
        {
            Block 2;
        }
}
Else
{
        Block 3;
}
```

4. <u>Else if ladder</u>

Else if ladder is used to select multipath decisions.

Syntax:-

```
if(condition 1>
{
        Statement Block 1;
        Else if(condition 2)
        {
            Statement Block 2;
        }
        Else
        {
            Statement Block 3;
        }
```

}

5. Switch case

Instead of writing **many** if..else statements, we can use the switch statement.

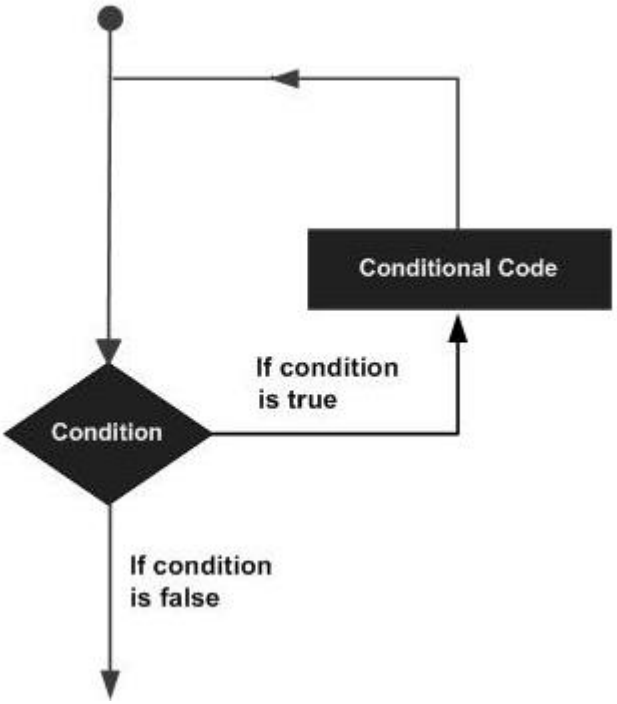
The switch statement selects one of many code blocks to be executed:

Syntax:

```
switch(condition)
{
 case value1:
     statements;
     break;
 case value2:
     statements;
     break;
 default:
     statements;
}
```

Looping statements(Repetition structures)

A loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages



A loop in C consists of two parts, a body of a loop and a control statement. The control statement is a combination of some conditions that direct the body of the loop to execute until the specified condition becomes false.

Depending upon the position of a control statement in a program, looping statement in C is classified into two types:

1. Entry controlled loop

2. Exit controlled loop

In an **entry control loop in C,** a condition is checked before executing the body of a loop. It is also called as a pre-checking loop.

In an **exit controlled loop**, a condition is checked after executing the body of a loop. It is also called as a post-checking loop.

C programming language provides us with three types of loop constructs:

1. The while loop
2. The do-while loop
3. The for loop

1. While Loop

It is an entry-controlled loop

In while loop, a condition is evaluated before processing a body of the loop. If a condition is true then and only then the body of a loop is executed. After exiting the loop, the control goes to the statements which are immediately after the loop. The body of a loop can contain more than one statement. If it contains only one statement, then the curly braces are not compulsory.

Syntax:

```
While(condition)
{
        Body of the loop;
}
```

2. Do-While Loop

It is an exit-controlled loop.

In the do-while loop, the body of a loop is always executed at least once. After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop.

Syntax:-

```
do
{
  Statements;
} while (condition);
```

3. For Loop

A for loop is a more efficient loop structure in 'C' programming. The general structure of for loop syntax in C is as follows:

```
for (initial value; condition; incrementation or decrementation )
{
  statements;
}
```

The initial value of the for loop is performed only once. The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned. The incrementation/decrementation increases (or decreases) the counter by a set value.