Unit- 2

Chap-4

C Operators and Expression

Prof. Sonal Pal(Department of Computer Application)

C Operators

- The symbols which are used to perform logical and mathematical operations in a C program are called C operators.
- These C operators join individual constants and variables to form expressions.
- Operators, functions, constants and variables are combined together to form expressions.
- Consider the expression A + B * 5. where, +, * are operators, A, B are variables, 5 is constant and A + B * 5 is an expression.

TYPES OF C OPERATORS:

- **Unary Operators**: Symbols that operate on single operand.
- Eg: +a, -a
- **Binary Operators:** Symbols that operate on two operands.
- Eg: a+b, a-b, c*d.
- **Ternary Operators:** Symbols that operate on three operands.
- Eg: a>b? a:b;

TYPES OF BINARY OPERATORS
C language offers many types of operators.

- Arithmetic operators
- Assignment operators
- Relational operators
- Logical operators
- Bit wise operators
- Conditional operators (ternary operators)
- Increment/decrement operators
- Special operators

1.Arithmetic Operators

C programming language provides all basic arithmetic operators: +, -, *, / and %.

Note: '/' is integer division which only gives integer part as result after division. '%' is modulo division which gives the remainder of integer division as result.

Operators	Meanings
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

Table 1: Arithmetic Operators in C

2. Relational Operators

Relational operators are used when we have to make comparisons. C programming offers 6 relational operators.

Operators	Meanings
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
==	Is equal to
!=	Is not equal to

Table 2: Relational Operators in C

3. Logical Operators

Logical operators are used when more than one conditions are to be tested and based on that result, decisions have to be made. C programming offers three logical operators. They are:

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Table 3: Logical Operator in C

4. Assignment Operators

Assignment operators are used to assign result of an expression to a variable. ‘=’ is the assignment operator in C. Furthermore, C also allows the use of shorthand assignment operators. Shorthand operators take the form:
var op = exp;

Eg: **a=9;**
Num=34;
Subj=“computer”

5. Increment and Decrement Operators

- C programming allows the use of **++** and **–** operators which are increment and decrement operators respectively. Both the increment and decrement operators are unary operators. The increment operator ++ adds 1 to the operand and the decrement operator – subtracts 1 from the operand. The general syntax of these operators are:
- Increment Operator: *m++ or ++m*;
- Decrement Operator: *m–or –m*;
- In the example above, *m++* simply means *m=m+1;* and *m–* simply means *m=m-1;*
- Increment and decrement operators are mostly used in for and while loops.

6. Conditional Operator

The operator pair “?” and “:” is known as conditional operator. These pair of operators are ternary operators. The general syntax of conditional operator is:
expression1 ? expression2 : expression3 ;

Consider an if else statement as:

```
if (a > b) x = a ; else x = b ;
```

Now, this if else statement can be written by using conditional operator as:

```
x = (a > b) ? a : b ;
```

7. Bitwise Operator

In C programming, bitwise operators are used for testing the bits or shifting them left or right. The bitwise operators available in C are:

Operators	Meaning
&	Bitwise AND
!	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift right

Table 4: Bitwise Operators in C

8. Special Operators

- C programming supports special operators like comma operator, sizeof operator, pointer operators (& and *) and member selection operators (. and ->).

1. Comma Operator

The comma operator can be used to link the related expressions together. A comma linked expression is evaluated from left to right and the value of the right most expression is the value of the combined expression.
For example:
x = (a = 2, b = 4, a+b)

2 .Sizeof() operator

The sizeof operator is usually used with an operand which may be variable, constant or a data type qualifier. This operator returns the number of bytes the operand occupies. Sizeof operator is a compile time operator. Some examples of use of sizeof operator are:
x = sizeof (a); y = sizeof(float);

C Expressions

- Arithmetic expression in C is a combination of variables, constants and operators written in a proper syntax.
- Expressions perform specific actions, based on an operator, with one or two operands.
- C can easily handle any complex mathematical expressions but these mathematical expressions have to be written in a proper syntax. :

Expression Example:

- c=a+b
- a-b+c
- a+b-(a*c)

Types of Expressions in C

- Arithmetic expressions
- Relational expressions
- Logical expressions
- Conditional expressions

Arithmetic Expressions:

- The arithmetic expression is evaluated in specific order considering the operator's precedence, and the result of an expression will be based on the type of variable.
- Eg:

```
Arithmetic Expression:

a=2,b=3,c=4

Z = a + b - (a * c)
```

```
Z = 2 + 3 - (2 * 4)
  = 2 + 3 - (8)
  = 5 - 8
  = -3
```

Relational Expressions:

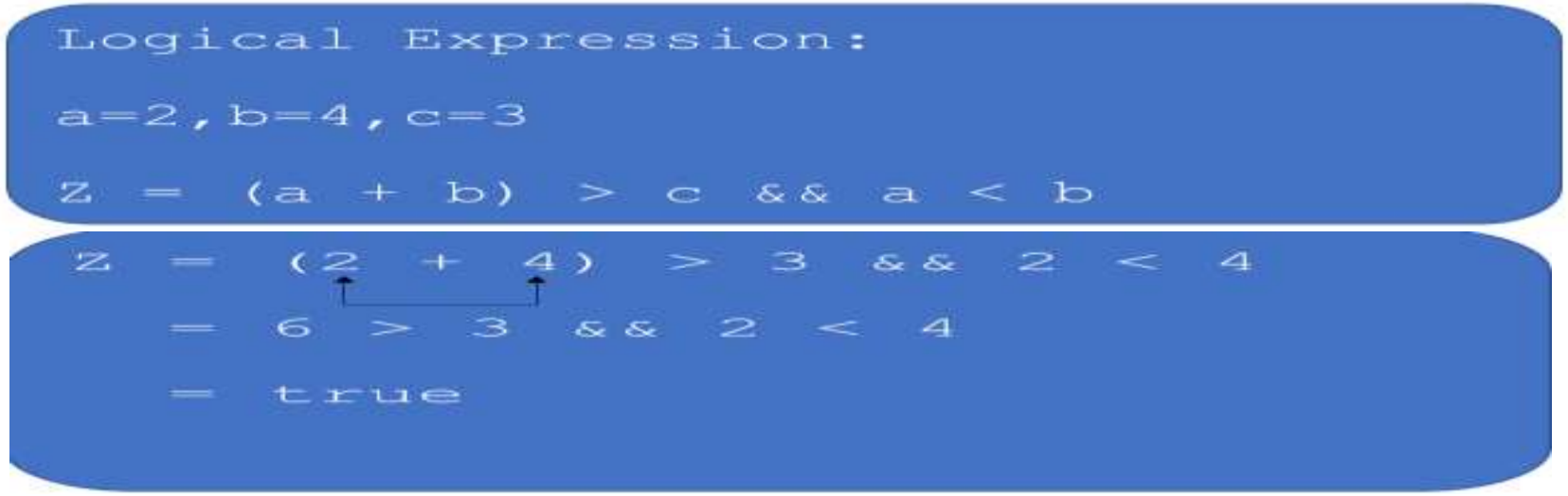
- Relational operators >, <, ==,!= etc are used to compare 2 operands. Relational expressions consisting of operands, variables, operators, and the result after evaluation would be either true or false.
- Eg:

```
Relational Expression:

a=3,b=2,c=1

Z = a * b > a + c
```

```
Z = 3 * 2 > 3 + 1
  = 6 > 3 + 1
  = 6 > 4
  = true (1)
```

Logical Expressions:

- Relational expressions and arithmetic expressions are connected with the logical operators, and the result after an evaluation is stored in the variable, which is either true or false.
- Eg:



Conditional Expressions:

- The general syntax of conditional expression is:
- Exp1? Exp2: Exp3
- From the given above expressions, the first expression (exp1) is conditional, and if the condition is satisfied, then expression2 will be executed; otherwise, expression3 will be performed.
- Eg:

```
Conditional Expression:

a=5,b=3,c=1

a * b < c ? true : false
```

Operator precedence

Expressions are normally evaluated left to right. Complex expressions are evaluated one at a time. The order in which the expressions are evaluated is determined by the precedence of the operators used. The standard C ordering is followed.

1. negation (-) unary
2. power
3. multiplication, division and modulo
4. addition and subtraction

- At first, the expressions within parenthesis are evaluated. If no parenthesis is present, then the arithmetic expression is evaluated from left to right. There are two priority levels of operators in C.
- **High priority:** * / %
  Low priority: + –
- The evaluation procedure of an arithmetic expression includes two left to right passes through the entire expression. In the first pass, the high priority operators are applied as they are encountered and in the second pass, low priority operations are applied as they are encountered.

Evaluation of arithmetic Expressions

- Type Conversion

The end….

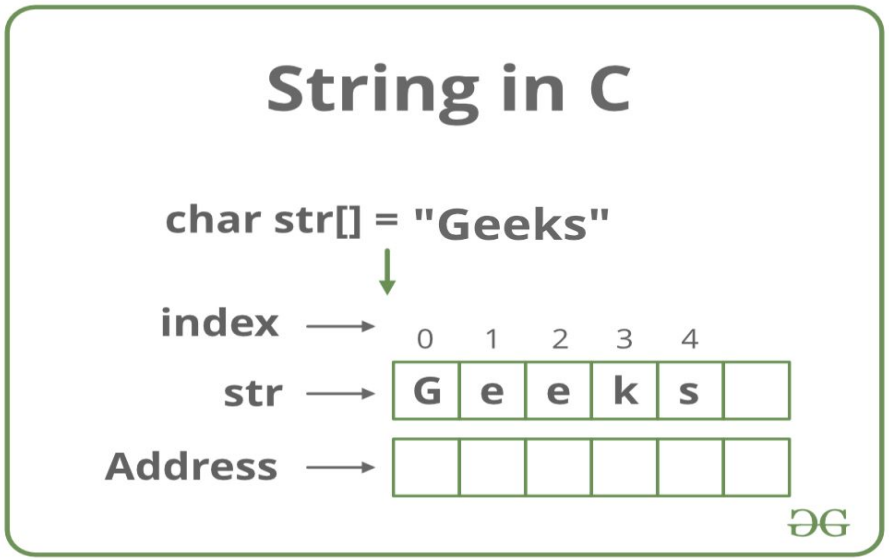
Unit-3
Chapter-7
STRINGS

PREPARED BY

PROF. SONAL PAL

What is String?

- **String in C programming is a sequence of characters terminated with a null character '\0'. Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a unique character '\0'.**



Declaration of Strings:

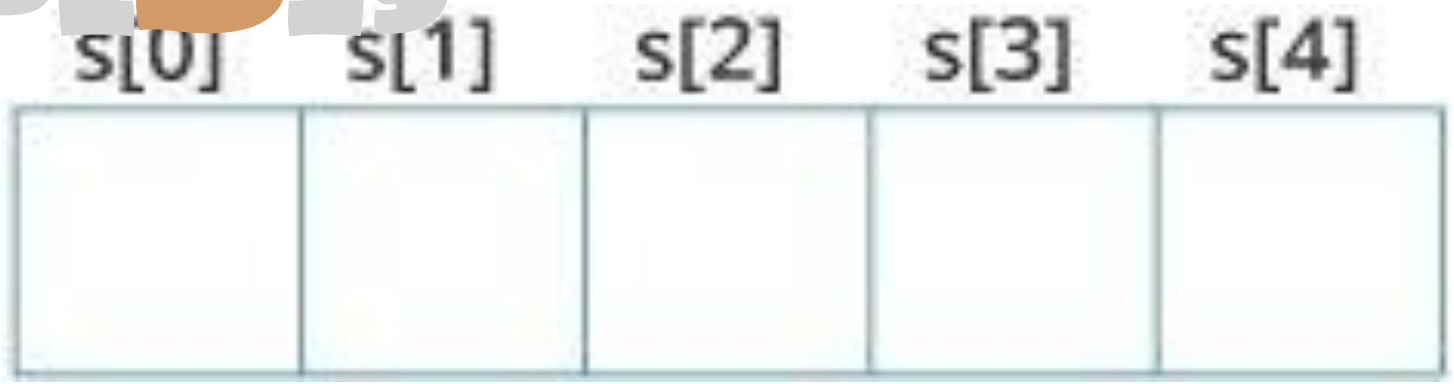
- Declaring a string is as simple as declaring a one-dimensional array. Below is the basic syntax for declaring a string.

```
char str_name[size];
```

- In the above syntax **str_name** is any name given to the string variable and size is used to define the length of the string, i.e the number of characters strings will store.
- **Note:** There is an extra terminating character which is the ***Null character ('\0') used to indicate the termination of a string that differs strings from normal character arrays***

How to declare a string?

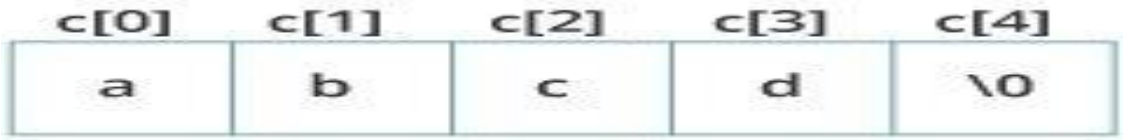
Eg: char s[5];



Initializing a String:

- You can initialize strings in a number of ways.

```
Eg:
char c[] = "abcd";
char c[50] = "abcd";
char c[] = {'a', 'b', 'c', 'd', '\0'};
char c[5] = {'a', 'b', 'c', 'd', '\0'};
```



String handling functions: The string handling functions are defined in the header file **string. h**. This header file must be included in the C program to use the string handling functions

No.	Function	Description
1)	strlen(string_name)	returns the length of string name.
2)	strcpy(destination, source)	copies the contents of source string to destination string.
3)	strcat(first_string, second_string)	concats or joins first string with second string. The result of the string is stored in first string.
4)	strcmp(first_string, second_string)	compares the first string with second string. If both strings are same, it returns 0.

Character Handling function:Character functions need ctype.h header file to be included in the program. Different character functions provided by C Language are:

- **1. isalpha():**

  This function checks whether the character variable/constant contains alphabet or not.

- **2. isdigit()**

  This function checks whether the character variable/ constant contains digit or not.

- **3. isalnum()**

  This function checks whether the character variable/constant contains an alphabet or digit.

- **4. ispunct()**

  This function checks whether the character variable/constant contains a punctuator or not. Punctuators are comma, semicolon etc.

- **5. isspace()**

This function checks whether the character variable/constant contains a space or not.

- **6. isupper()**

This function checks whether the character variable/constant contains an capital l letter alphabet or not.

- **7. islower()**

This function checks whether the character variable/constant contains a lowercase alphabet or not.

- **8. toupper()**

  This function converts lowercase alphabet into uppercase alphabet.
- **9. tolower()**
- This function converts an uppercase alphabet into lowercase alphabet.
- **10. isnumeric()**
- This function checks whether the character variable/constant contains an number or digit.

(UNIT-1)CHAPTER-2

C PROGRAMMING BASIC CONCEPTS

C CHARACTER SET:

- C is a procedural programming language. It was initially developed by Dennis Ritchie as a system programming language to write operating system. The main features of C language include low-level access to memory, simple set of keywords, and clean style, these features make C language suitable for system programming like operating system or compiler development.
- In the C programming language, *the character set* refers to a set of all the valid characters that we can use in the source program for forming words, expressions, and numbers.
- The source character set contains all the characters that we want to use for the source program text. On the other hand, the execution character set consists of the set of those characters that we might use during the execution of any program.

Types of Characters in C:

- The C programming language provides support for the following types of characters. In other words, these are the *valid* characters that we can use in the C language:
- Digits
- Alphabets
- Main Characters

Alphabets

The C programming language provides support for all the alphabets that we use in the English language. Thus, in simpler words, a C program would easily support a total of 52 different characters- 26 uppercase and 26 lowercase.

Type of Character	Description	Characters
Lowercase Alphabets	a to z	a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z
Uppercase Alphabets	A to Z	A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

Digits

The C programming language provides the support for all the digits that help in constructing/ supporting the numeric values or expressions in a program. These range from 0 to 9, and also help in defining an identifier. Thus, the C language supports a total of 10 digits for constructing the numeric values or expressions in any program.

Type of Character	Description	Characters
Digits	0 to 9	0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Special Characters

We use some special characters in the C language for some special purposes, such as logical operations, mathematical operations, checking of conditions, backspaces, white spaces, etc.

Type of Character	Examples
Special Characters	` ~ @ ! $ # ^ * % & ( ) [ ] { } < > + = _ - | / \ ; : ‘ “ , . ?

TOKENS IN C

- Tokens are some of the most important elements used in the C language for creating a program.
- A token is the smallest unit used in a C program. Each and every punctuation and word that you come across in a C program is token. A compiler breaks a C program into tokens and then proceeds ahead to the next stages used in the compilation process.
- Classification and Types of Tokens in C
- Here are the categories in which we can divide the token in C language:

- Identifiers in C
- Keywords in C
- Operators in C
- Strings in C
- Special Characters in C
- Constant in C

Identifiers in C: These are used to name the arrays, functions, structures, variables, etc. The identifiers *are user-defined words* in the C language. These can consist of lowercase letters, uppercase letters, digits, or underscores, but the starting letter should always be either an alphabet or an underscore

- ***The rules that we must follow when constructing the identifiers***:
- The identifiers must not begin with a numerical digit.
- The first character used in an identifier should be either an underscore or an alphabet. After that, any of the characters, underscores, or digits can follow it.
- Both- the lowercase and uppercase letters are distinct in an identifier. Thus, we can safely say that an identifier is case-sensitive.
- We cannot use an identifier for representing the keywords.
- An identifier does not specify blank spaces or commas.
- The maximum length of an identifier is 31 characters.
- We must write identifiers in such a way that it is not only meaningful- but also easy to read and short.

KEYWORDS IN C:

- keywords as the reserved or pre-defined words that hold their own importance. It means that every keyword has a functionality of its own. Since the keywords are basically predefined words that the compilers use, thus we cannot use them as the names of variables.

The C language provides a support for 32 keywords, as mentioned below:

auto	**enum**	**const**	**goto**
double	case	float	default
struct	register	unsigned	sizeof
int	typedef	short	volatile
break	extern	continue	if
else	char	for	do
switch	return	void	static
long	union	signed	while

OPERATORS IN C:

- The operators in C are the special symbols that we use for performing various functions. *Operands* are those data items on which we apply the operators.
- we classify the operators:
- Unary Operator
- Binary Operator
- Ternary Operator

Unary Operator

- The unary operator in c is a type of operator that gets applied to one single operand, for example: (–) decrement operator, (++) increment operator, (type)*, sizeof, etc.

Binary Operators:

- Binary operators are the types of operators that we apply between two of the operands. Here is a list of all the binary operators that we have in the C language:
- Relational Operators
- Arithmetic Operators
- Logical Operators
- Shift Operators
- Conditional Operators
- Bitwise Operators
- Misc Operator
- Assignment Operator

Ternary operator:

- Using this operator would require a total of three operands. For instance, we can use the ?: in place of the if-else conditions.

List of Binary Operators in C

- **Arithmetic Operators**
- Arithmetic operators are those binary operators in C that are used to perform basic arithmetic operations like addition, subtraction, multiplication, division, and modulus operation.
- **Addition Operator (Syntax - operand_1 + operand_2)**
- **Subtraction Operator(Syntax - operand_1 - operand_2)**
- **Multiplication Operator(Syntax - operand_1 * operand_2)**
- **Division Operator(Syntax - operand_1 / operand_2)**
- **Modulus Operator(Syntax - operand_1 % operand_2)**

Relational Operator:

- Relational operators are those binary operators in C that are used to compare any two entities, like two integers, characters, floating point numbers, etc. The result produced by them is either 1 or 0, where 1 means the comparison is true and 0 means it is not i.e false.
- **The equal == operator**
- **The not equal != operator**
- **The Less Than < Operator**
- **The Less Than or Equal <= Operator**
- **The Greater than > Operator**
- **The Greater Than or Equal >= Operator**

Logical Operators:

- We have a set of three logical operators in C language that helps us to combine the result of two or more logical conditions or Boolean values. Out of those three i.e **AND, OR, and NOT** only the first two are binary operators.
- **Logical AND Operator**
- **Syntax: operand1 && operand2**
- **Logical OR Operator**
- **Syntax: operand1 || operand2**
- **Logical NOT Operator**
- **Syntax: ~operand1**

Assignment Operator:

- Assignment operators are those binary operators in C that assign the values or result of an expression to a variable and the value on the right side must be of the same data type as the variable on the left side.
- The most commonly used assignment operator is = operator,
- for example,
- x=4 means the value 4 is assigned to variable x.

Bitwise Operator:

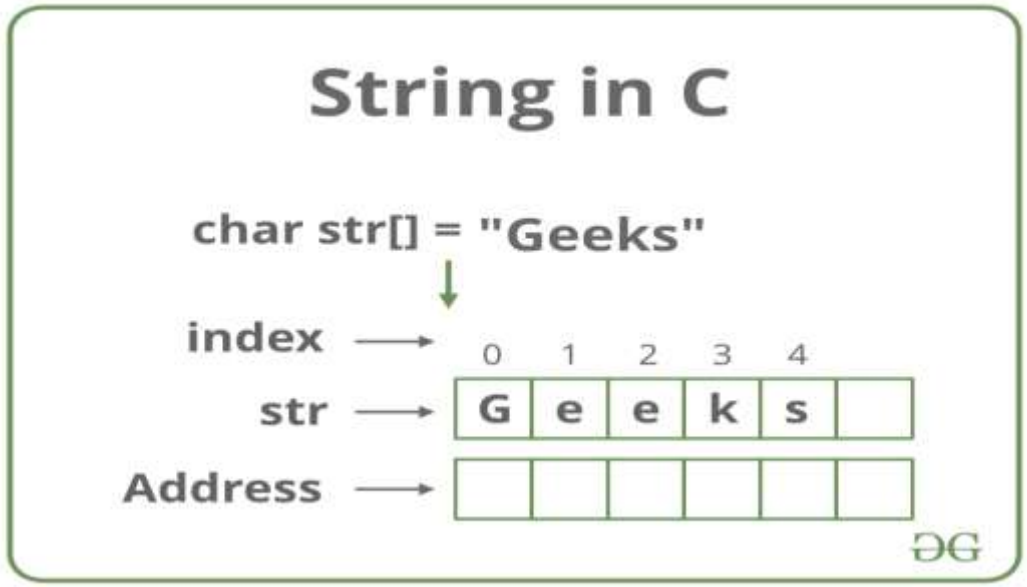
- Bitwise operators are those binary operators in C that are used to manipulate bits in different ways. They are equivalent to how we use mathematical operations like (+, -, /, *) among numbers, similarly, we use bitwise operators like (|, &, ^, <<, >>, ~) among bits.
- **Bitwise AND Operator**
- **Syntax: operand1 & operand2**
- **Bitwise OR Operator**
- **Syntax: operand1 | operand2**
- **Bitwise XOR Operator**
- **Syntax: operand1 ^ operand2**
- **Bitwise Shift Left Operator**
- **Syntax: operand1 << operand2**
- **Bitwise Shift Right Operator**
- **Syntax: operand1 >> operand2**

Conclusion:

- **Binary Operators in the C programming language are ones that operate on two operands.**
- **Arithmetic Operators, Relational Operators, Logical Operators, Assignment Operators, and Bitwise Operators fall under the category of Binary Operators.**

Strings in C:

- String in C programming is a sequence of characters terminated with a null character ‘\0’. Strings are defined as an array of characters.
- The difference between a character array and a string is the string is terminated with a unique character ‘\0’.

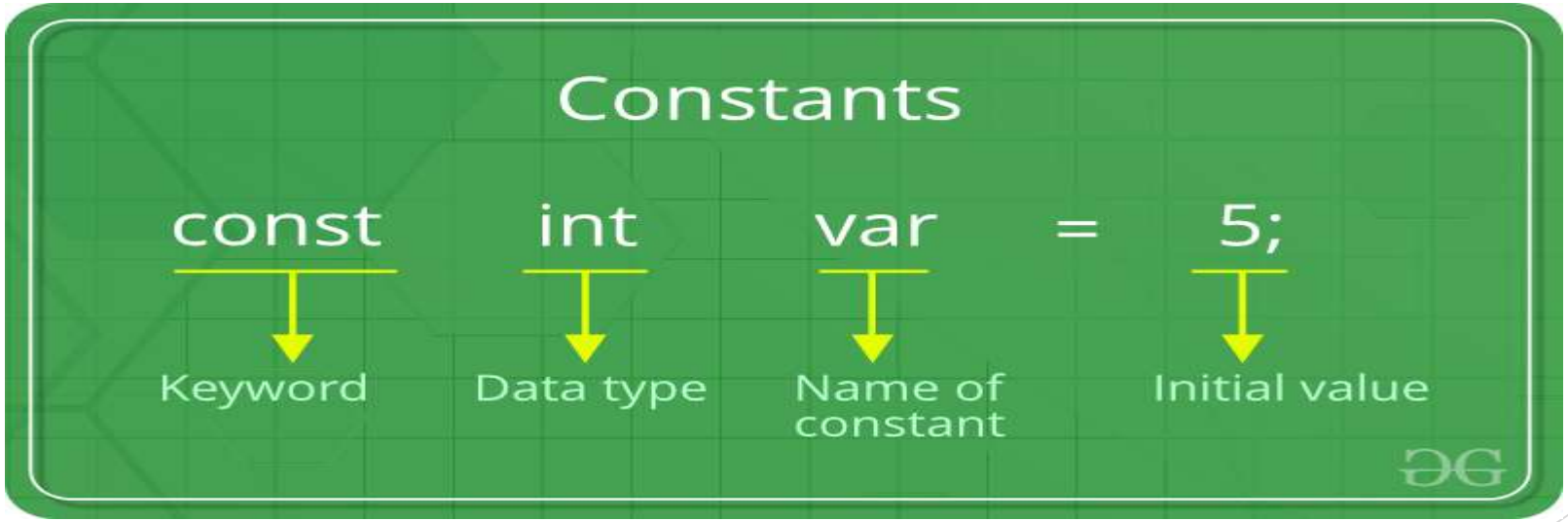


Special Characters in C:

- Special characters or symbols (single character or sequence of characters) that have a “Special” built-in meaning in the language and typically cannot be used in identifiers.
- Let’s understand this in a bit detail special symbols are characters such as
- Doller ($)
- Ampersand (&)
- Percent (%)
- brackets (‘[]’)
- braces or curly braces (‘{}’)
- Parenthesis (‘()’), etc.

Constant: Variables having fixed values that don't change and cannot be changed throughout the execution of the program once initialized are called Constants.

There are mainly two types of constants: primary and secondary. Primary and secondary constants are once again divided into subcategories. Constants in C can be declared in two ways, viz. Use the const keyword, or the #define preprocessor directive.



Variables:

- **A variable** in simple terms is a storage place that has some memory allocated to it. Basically, a variable is used to store some form of data. Different types of variables require different amounts of memory, different type of memory locations, and some specific set of operations that can be applied to them.
- Declaration of a Variable:
- Syntax: data_type varaiable_name;
- Eg: int a; float num;
- Initialization of a Variable:
- Syntax: data_type varaiable_name= constant/literal/expression;
- Eg: int a=10; float num=3.4;

Unit-1

INTRODUCTION TO “C” PROGRAMMING

Overview and history of C:

- **C programming language** was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A.
- C evolved from two earlier languages called the BPCL and B which were also developed at Bell Laboratories.
- In 1978, Brian Kernighan and Dennis Ritchie published a definitive description of the language, referred to as the 'K&R C'.
- Because of its features, C became popular very fast and by 1980s it was one of the most popular programming language being used.
- C is a general-purpose structured programming language. It has a rich set of data types and has a syntax that uses the English language keywords.

- Its features categorize it as a high-level language. C has additional features that allow it to be used at the lower level, thus bridging the gap between the machine language and the conventional high-level languages.
- This flexibility allows C to be widely used for systems programming. C compilers are commonly available for computers of all sizes.
- The compilers are usually compact, and they generate object codes that are small and highly efficient as compared to programs compiled in other high-level languages.
- An important characteristic of C is that the programs are highly portable. This is because C implements most computer dependent functions as its library functions.
- Every version of C is accompanied by its own set of library functions. The library functions are relatively standardized. Therefore, most C programs can be processed on many different computers with little or no alteration.

C language can be defined in following ways:

- 1) **C as a mother language:** C language is considered as the mother language of all the modern programming languages because most of the compilers, JVMs, Kernels, etc. are written in C language, and most of the programming languages follow C syntax, for example, C++, Java, C#, etc.
- 2**) C as a system programming language:** A system programming language is used to create system software. C language is a system programming language because it can be used to do low-level programming (for example driver and kernel).
- 3) **C as a procedural language**: A procedure is known as a function, method, routine, subroutine, etc. A procedural language specifies a series of steps for the program to solve the problem.A procedural language breaks the program into functions, data structures, etc.

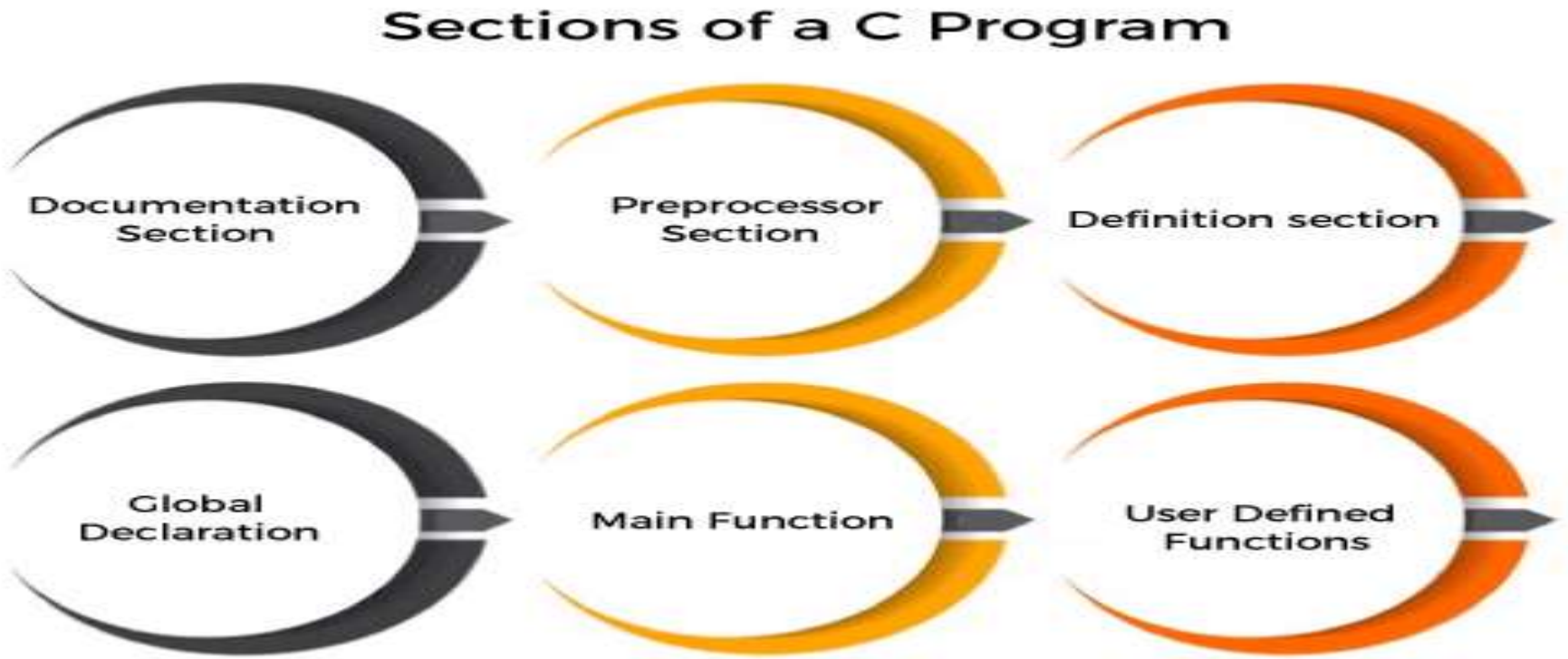
Let's see the programming languages that were developed before C language.

Language	Year	Developed By
Algol	1960	International Group
BCPL	1967	Martin Richard
B	1970	Ken Thompson
Traditional C	1972	Dennis Ritchie
K & R C	1978	Kernighan & Dennis Ritchie
ANSI C	1989	ANSI Committee
ANSI/ISO C	1990	ISO Committee
C99	1999	Standardization Committee

Features of C Language:

- 1**) Simple:** C is a simple language in the sense that it provides a structured approach (to break the problem into parts), the rich set of library functions, data types, etc.
- **2) Machine Independent or Portable**: Unlike assembly language, c programs can be executed on different machines with some machine specific changes. Therefore, C is a machine independent language.
- **3) Mid-level programming language**: Although, C is intended to do low-level programming. It is used to develop system applications such as kernel, driver, etc. It also supports the features of a high-level language. That is why it is known as mid-level language.
- **4) Structured programming language:** C is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify. Functions also provide code reusability.
- **5) RichLibrary:** C provides a lot of inbuilt functions that make the development fast.
- **6) Memory Management**: It supports the feature of dynamic memory allocation. In C language, we can free the allocated memory at any time by calling the free() function.
- **7) Speed:** The compilation and execution time of C language is fast since there are lesser inbuilt functions and hence the lesser overhead.

STRUCTURE OF “C” PROGRAM:



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

  .

  */

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 by:
- #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.

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';
```

- The size of the above global variables is listed as follows:

  char = 1 byte

  float = 4 bytes

  int = 4 bytes

- 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()
- **int** main()

  Or
- **void** main()

Local declarations:

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

```
main()
{
int i = 2;
i++;
}
```

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.
- **return**;

First C Program:

- To write the first c program, open the C console and write the following code:

```
#include <stdio.h>
    int main()
    {
    printf("Hello C Language");
    return 0;
    }
```

How to compile and run the c program:

- There are 2 ways to compile and run the c program, by menu and by shortcut.
- **By menu**

  Now **click on the compile menu then compile sub menu** to compile the c program.

  Then **click on the run menu then run sub menu** to run the c program.

- **By shortcut**

  Or, press ctrl+f9 keys compile and run the program directly.

  You will see the following output on user screen.


Turbo C++ IDE
Hello C Language

- You can view the user screen any time by pressing the **alt**
- Now **press Esc** to return to the turbo c++ console.**+f5** keys.

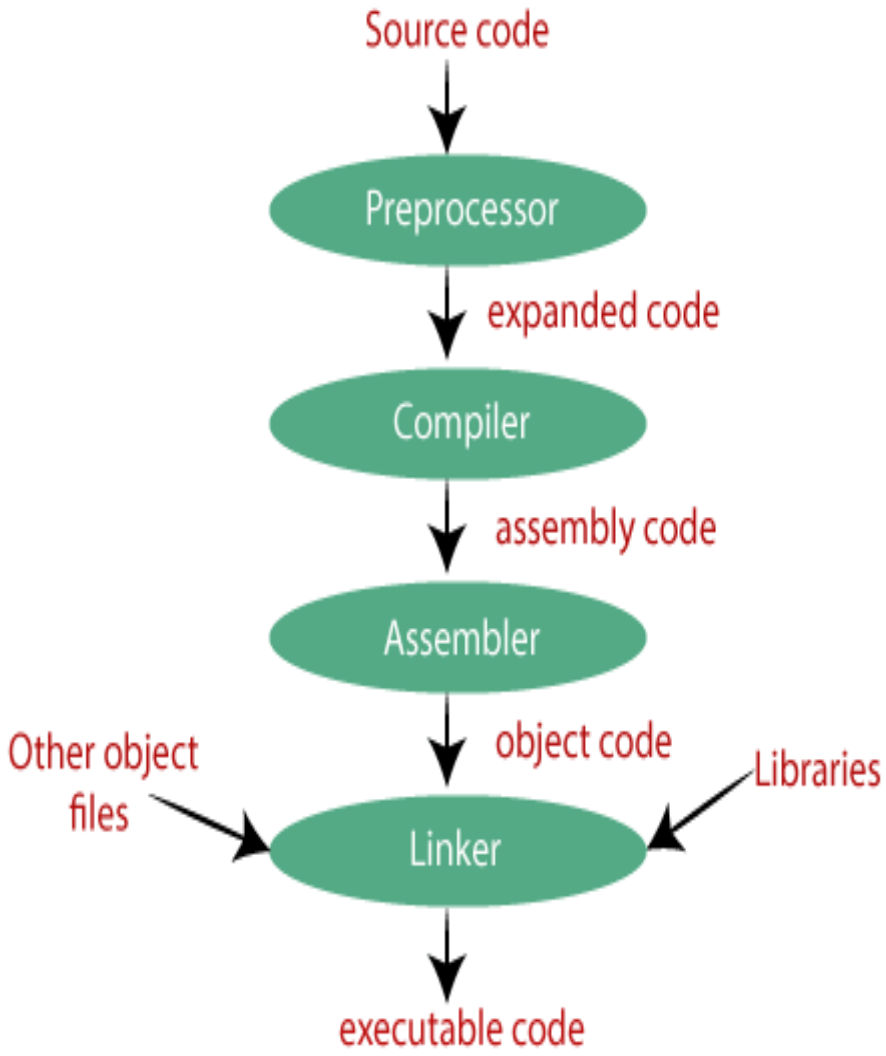
What is a compilation?

- The compilation is a process of converting the source code into object code. It is done with the help of the compiler. The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code.
-

```
#include<stdio.h>
main()
{
printf("Hello javaTpoint");
return 0;
}
```

```
01000000000000
01111111111111
01010101101010
00000011111111
00000111111111
00000010101011
```

- The c compilation process converts the source code taken as input into the object code or machine code. The compilation process can be divided into four steps, i.e., Pre-processing, Compiling, Assembling, and Linking.
- The preprocessor takes the source code as an input, and it removes all the comments from the source code. The preprocessor takes the preprocessor directive and interprets it. For example, if **<stdio.h>,** the directive is available in the program, then the preprocessor interprets the directive and replace this directive with the content of the '**stdio.h**' file.

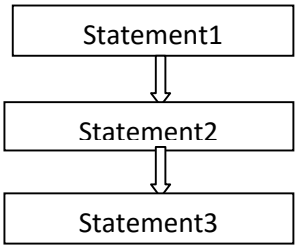

Source code
Preprocessor
expanded code
Compiler
assembly code
Assembler
object code
Other object files
Libraries
Linker
executable code

Control statements in C

Control statements control the flow of execution of the statements of a program. The various types of control statements in C language are as under:-

I. Sequential
II. Conditional
III. Iteration

Sequential control:- In sequential control, the C program statements are executed sequentially i.e., one after the another from beginning to end.




```
#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

int x , y, sum;

printf("enter the two numbers");

scanf("%d%d",&x,&y);

sum=x+y;

printf("the sum of the two numbers is =%d",sum);

getch();

}
```

Conditional Control (Selection Control or Decision Control) :- In conditional control , the execution of statements depends upon the condition-test. If the condition evaluates to true, then a set of statements is executed otherwise another set of statements is followed. This control is also called **Decision Control** because it helps in making decision about which set of statements is to be executed.

Decision control structure in C can be implemented by using:-

1. If statement
2. If-else statement
3. Nested if else statement
4. else-if ladder
5. case control structure
6. conditional operator

Iteration Control (Loops)

Iterations or loops are used when we want to execute a statement or block of statements several times. The repetition of loops is controlled with the help of a test condition. The statements in the loop keep on executing repetitively until the test condition becomes false.
There are the following three types of loops:-

1. While loop
2. Do-while loop
3. For loop

Decision control

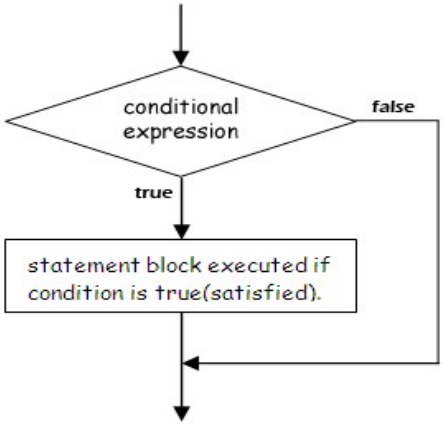
Conditional Control (Selection Control or Decision Control) :- In conditional control , the execution of statements depends upon the condition-test. If the condition evaluates to true, then a set of statements is executed otherwise another set of statements is followed. This control is also called **Decision Control** because it helps in making decision about which set of statements is to be executed.

Decision control structure in C can be implemented by using:-

1. If statement
2. If-else statement
3. Nested if else statement
4. else-if ladder
5. case control structure
6. conditional operator

if statement:- This is the most simple form of decision control statement. In this form, a set of statements are executed only if the condition given with **if** evaluates to true.
Its general syntax and flow chart is as under:-

```
if(condition)
{
 Statements ;
}
```



A program to understand the if statement.

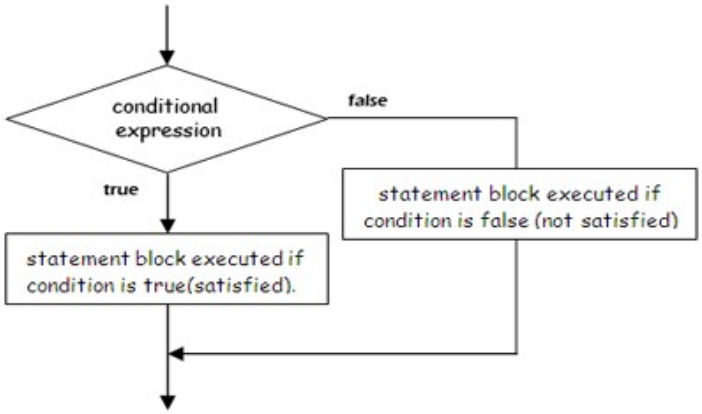
```
/* program to check whether the given number is negative*/
```

```
#include<stdio.h>
#include<conio.h>
void main()
        {
        int num;
        clrscr();
        printf("enter the number");
        scanf("%d",&num);
        if(num<0)
        {
        printf("the entered number is negative");
        }
        getch();
        }
        }
```

if- else statement:- This is a bi-directional control statement. This statement is used to test a condition and take one of the two possible actions. If the condition evaluates to true then one statement (or block of statements) is executed otherwise other statement (or block of statements) is executed.

The general form and flow chart of if-else statement is as under:-

```
if(condition)
{
block of statements;
}
else
{
block of statements;
}
```



We illustrate if-else statement using the following program

/*Program to find the biggest of two numbers*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int num1,num2 ;
clrscr();
printf("enter the two numbers");
```

```
scanf(“%d %d”,&num1,&num2);
if(num1>num2)
{
printf(“the number %d is big”,num1);
}
else
{
printf(“the number %d is big”,num2);
}
getch();

}
```

Nested if-else:- If we have **if-else** statement within either the body of an **if** statement or the body of **else** statement or in the body of both **if** and **else**, then this is known as nesting of if else statement.
The general form of nested if-else statement is as follows:-

```
if(condition1)
        {
                if(condition2)
                        statements;
                else
                        statements;
        }
else
        {
                if(condition3)
                        Statements;
                else
                        Statements;
        }
```

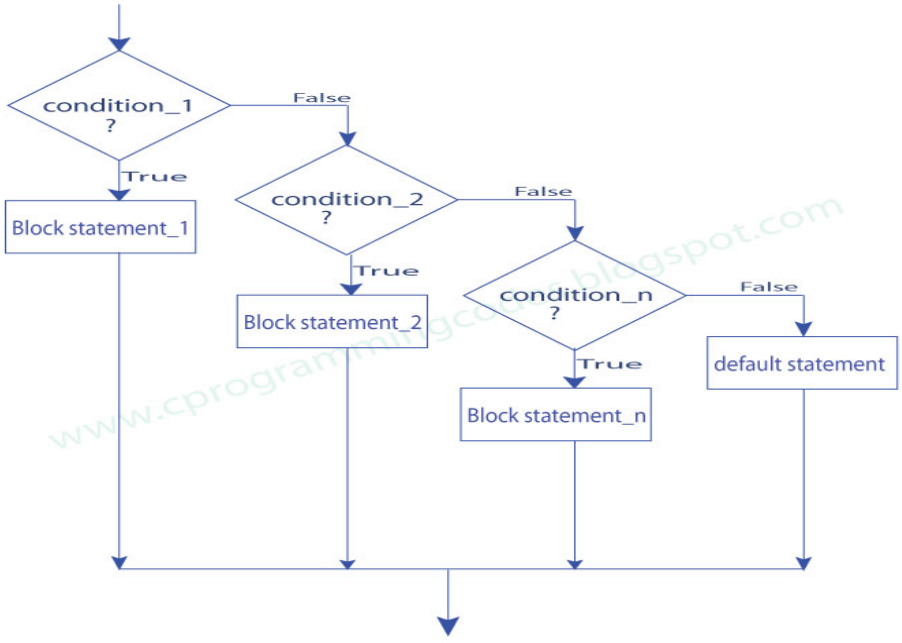
We give the following program to explain nesting of if else:-

```
/*program to find  the largest of three numbers*/

#include<stdio.h>
#include<conio.h>
void main()
        {
        int a, b,c,large;
        printf(“enter the three numbers”);
        scanf(“%d%d%d”,&a,&b,&c);

        if(a>b)
                {
                        if(a>c)
                        large=a;
                        else
                        large=c;
                }
        else
                {
                        if(b>c)
                        large=b;
```

```
                else
                large=c;
        }
        printf("largest number is %d",large);
        getch();
}
```

Else if ladder:- This is a type of nesting in which there is an if-else statement in every else part except the last else part. This type of nesting is called else if ladder.
The general syntax and flow chart of else if ladder is as follows:-

```
If(condition1)
        statementA;
else if(condition2)
        statementB;
else if(condition3)
        statementC;
else
       statementD;
```



A program to illustrate the else if ladder:-

```
/*program to find the grade of the student*/

#include<stdio.h>
#include<conio.h>
void main()
        {
        int marks;
        printf("enter  the percentage of the student");
        scanf("%d",&marks);
        if(marks>=80)
```

```
        printf("your grade is a");
else if(marks>=70)
        printf("your grade is b");
else if(marks>=60)
        printf("your grade is c");
else if(marks>=50)
        printf("your grade is d");
else
        printf("you are fail");
getch();
}
```

Switch case statement :- Switch case statements are a substitute for long **if statements** and has more flexibility and a clearer format than else-if ladder.

This statement is used to select one out of the several numbers of alternatives present in a block. This selection statement successively tests the value of an expression against a list of integer or character constants. When a match is found, the statements associated with that constant are executed.

The general syntax of switch case statement is as follows:-

```
switch(expression)
{
case constant1: statements;
case constant2: statements;
case constant3: statements;
……
……
case constantN: statements;
default : statement;
}
```

Here **switch, case** and **default** are keywords.
The expression following the **switch** keyword must give an integer value. This expression can be any integer or character variable, or a function call that returns an integer. It can also be arithmetic, relational, logical or bitwise expression yielding an integer. It can be integer or character constant also. Data types long int and short int are also allowed.
The constant following the **case** keyword should be of integer or character type. We cannot use floating point or string constant.

A program to explain switch case statement

```
/*program to print day of the week*/

#include<stdio.h>
#include<conio.h>
void main()
{
int day;
clrscr();
printf("enter the day number from 1 to7");
scanf("%d",&day);
switch(day)
{
```

```
case 1: printf("monday");
                break;
case 2:printf("tuesday");
                break;
case 3: printf("wednesday");
                break;
case 4:printf("thursday");
                break;
case 5: printf("friday");
                break;
case 6:printf("saturday");
                break;
case 7: printf("sunday");
                break;
default:printf("wrong input");
}
getch();

}
```

NOTE:- In switch case statement, if we do not associate a break statement with every case then from the case for which the expression matches with the constant, all the statements are executed until the switch case block ends. This situation is referred to as **Fall Through.**

Conditional operator:- It is a ternary operator and is used to perform simple conditional operations. It is used to do operations similar to if-else statement.
The general syntax of conditional operator is as under:-

Test expression? expression1:expression2

Here firstly the test expression is evaluated.
If the test expression evaluates to true then the expression1 is evaluated and it becomes the value of the whole expression.
If the test expression evaluates to false then the expression2 is evaluated and it becomes the value of the whole expression.

For eg., a>b ? a : b
Here firstly the expression **a>b** is evaluated. If it evaluates to true then the value of **a** becomes the value of the whole expression otherwise value of **b** becomes the value of whole expression.

Iterations/Loops

Iterations or loops are used when we want to execute a statement or block of statements several times. The repetition of loops is controlled with the help of a test condition. The statements in the loop keep on executing repetitively until the test condition becomes false.
There are the following three types of loops:-

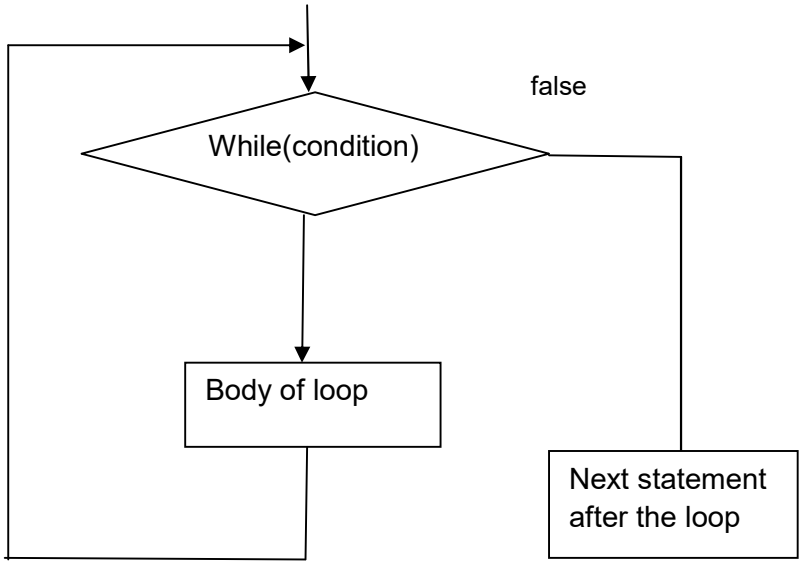
1. While loop
2. Do-while loop
3. For loop

All these loops are explained as under:-

While loop : - It is the fundamental looping statement in C. It is suited for the problems where it is not known in advance that how many times a statement or block of statements will be executed.
The general syntax of while loop is as under:-

```
While(condition)
        {
                Statement(s);
        }
```

We explain the working of while loop with the help of flow chart as under:-



Firstly the condition given with **while** is evaluated. If the condition evaluates to true then the statements given in the body of the loop are executed. After the execution of all the statements the condition is again checked and if it again evaluates to true then the statements given in the body of the loop are again executed. In this way the statements are executed again and again until the condition given evaluates to false.

For terminating the loop, a termination condition is given in the loop body which makes the condition false at some point of time and as a result of which the loop terminates. If we do not give the termination condition then the loop goes on repeating again and again infinite times. Such loops are referred to as infinite loops.

We explain the while loop with the help of following program.

```
/*program to find the sum of digits of the given number.*/

#include<stdio.h>
#include<conio.h>
void main()
{
int n,num,rem,sum=0;
printf("Enter the number");
scanf("%d",&num);
n=num;
while(num>0)
{
rem=num%10;
sum =sum+rem;
num=num/10;
}
printf("the sum of the digits of %d is =%d",n,sum);
getch();
}
```

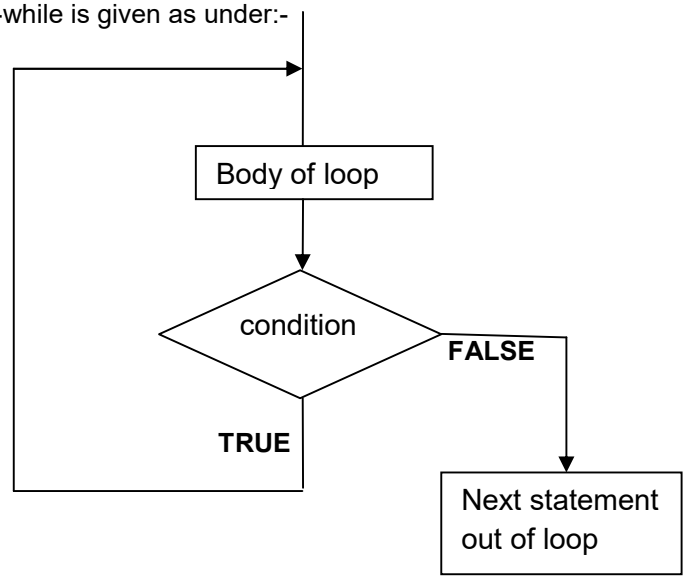
do-while loop :- The do-while statement is also used for looping. It is similar to while loop and is used for the problems where it is not known in advance that how many times the statement or block of statements will be executed. However, unlike while loop, in case of do-while, firstly the statements inside the loop body are executed and then the condition is evaluated. As a result of which this loop is executed at least once even if the condition is initially false. After that the loop is repeated until the condition evaluates to false.

Since in this loop the condition is tested after the execution of the loop, it is also known as post-test loop.

The general syntax of do-while loop is as follows:-

```
do
{
Statement(s);

}while(condition);
```

The flow chart of do-while is given as under:-



Difference between while and do-while loop

While	do-while
While loop is pre test loop.	do-while is post test loop
The statements of while loop may not be executed even once	The statements of do-while loop are executed atleast once
There is no semi colon given after while(condition)	There is semi colon given after while(condition);

The syntax of while loop is
While(condition)
{
Statements
}
	The syntax of do-while loop is as under:-
	Do
	{
	Statements;
	}while(condition);

For loop :- The general syntax of for loop consists of three expressions separated by semicolons. It is given as follows:-

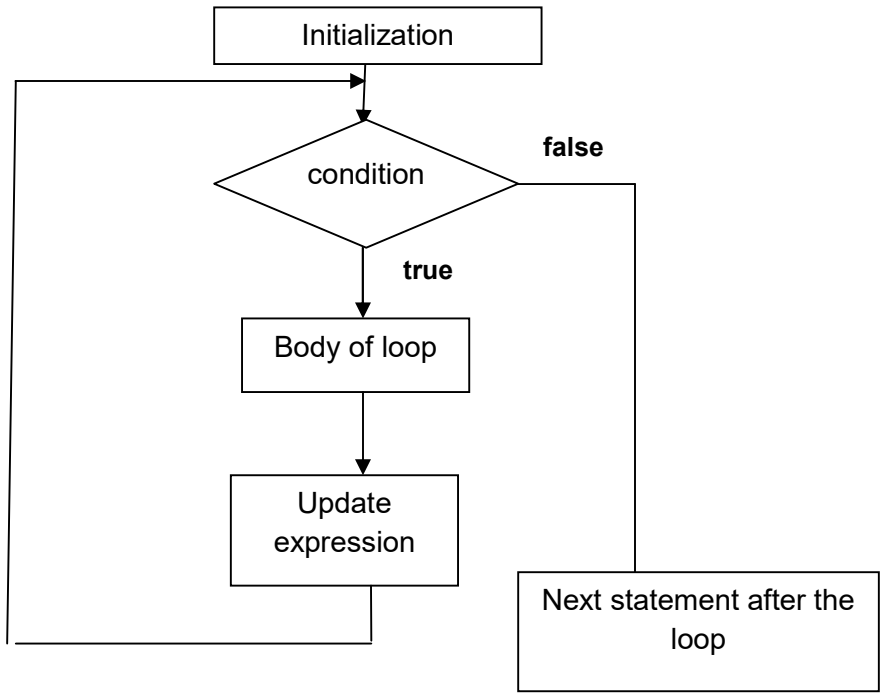
for(expression1;expression2;expression3)

{
Statement(s);
}

Here the expression1 is the initialization expression,expression2 is the test expression and expression3 is the update expression.

Expression1 is executed only once when the loop starts and is used to initialize the loop variables.
Expression2 is a condition and is tested before each iteration of the loop.
Expression3 is an update expression and is executed each time after the body of the loop is executed.

The flow chart of for loop is given as follows:-



we give the following program to explain for loop

/* program to generate the Fibonacci series*/

```
#include<stdio.h>
#include<conio.h>
void main()
        {
        int x,y,z;
        int i,n;
        x=0;
        y=1;
        printf("enter the number of terms");
        scanf("%d",&n);
        printf("%d",y);
        for(i=1;i<n;i++)
        {
        z=x+y;
        printf("%d",z);
        x=y;
```

```
y=z;
}
printf("\n");
 getch();
}
```

All the three expressions in the **for** loop are optional and therefore we can omit any or all the three expressions but in any case the two separating **semi colons** should always be present.
If we omit the **test expression** then it is always assumed to be true and therefore the loop never terminates and becomes an infinite loop.
If we want to make such loops a finite loop then a separate terminate condition is given within the loop body

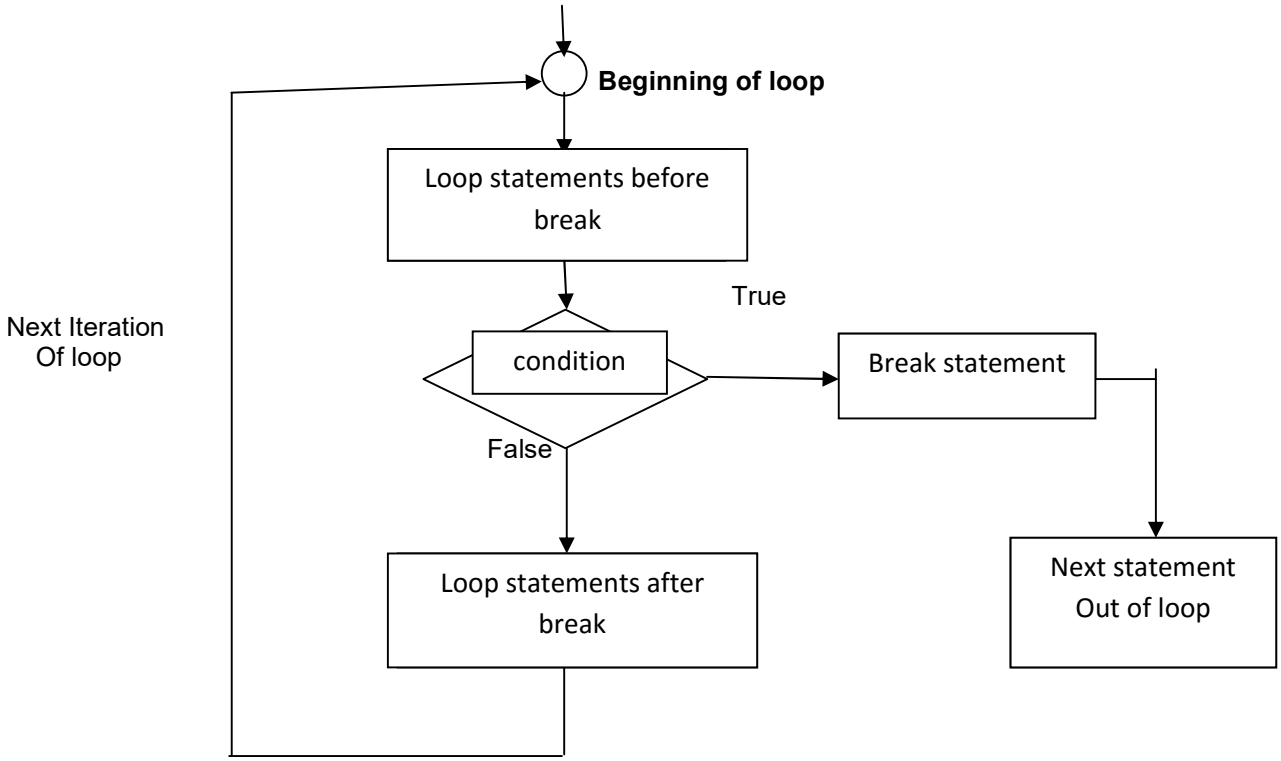
Jump statements

Jump statements are used to transfer the control from one part of the program to another part.
The various jump statements in C are as under:-

1. break statement
2. continue statement
3. goto statement

break statement:- break statement is used inside the loops or switch statement. This statement causes an immediate exit from the loop or the switch case block in which it appears.

It can be written as
break;



```
/*Program to understand the use of break statement*/
#include<stdio.h>
main()
{
int n;
for(n=1;n<=5;n++)
{
  if(n==3)
  {
    printf("I understand the use of break \n");
    break;
  }
  printf("Number = %d \n",n);
 }
printf("Out of for loop \n");
}
```

Output:

Number = 1
Number = 2
I understand the use of break
Out of for loop

when **break** statement is encountered, loop is terminated and the control is transferred to the statement immediately after the loop.
If **break** statement is written inside a nested loop structure then it causes exit from the innermost loop.
In Switch case , it is used as the last statement of every case except the last one. When executed, it transfers the control out of switch case and the execution of the program continues from the statement following switch statement.

```
  Switch(expression)
  {
    case1: statement-1;
           break;
     case2: statement-2;
           break;
     case3: statement-3;
           break;
       .
       .
       .
       .
     case n: statement-n;
            break;
     default: statement-d;
  }
```

/*program illustrating use of break in switch case */

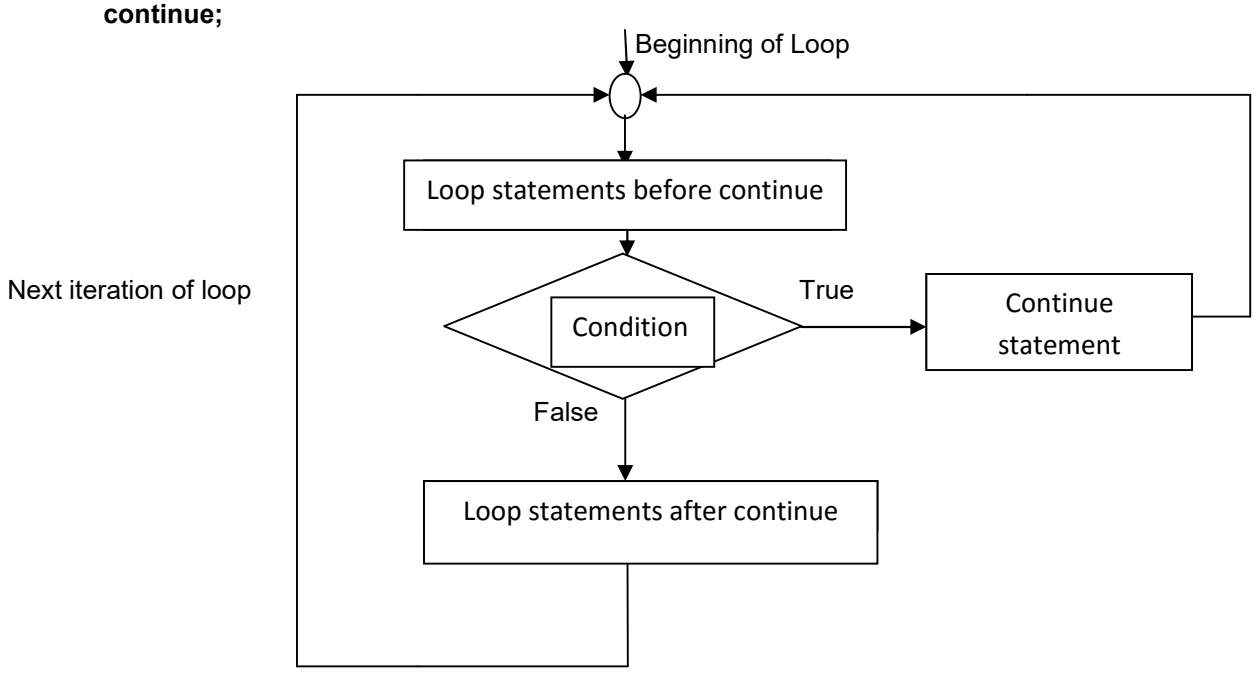
```
#include <stdio.h>
int main( )
{
    int  i = 2 ;
    switch ( i )
    {
        case 1 :
            printf ( "I am in case 1 \n" ) ;
            break ;
        case 2 :
            printf ( "I am in case 2 \n" ) ;
            break ;
        case 3 :
            printf ( "I am in case 3 \n" ) ;
            break ;
        default :
            printf ( "I am in default \n" ) ;
    }
    return 0 ;
}
```

The Output of this program would be :
I am in case 2

continue statement :- The continue statement is used inside the body of the loop statement. It is used when we want to go to the next iteration of the loop after skipping some of the statements of the loop.

The continue statement is written as under:-

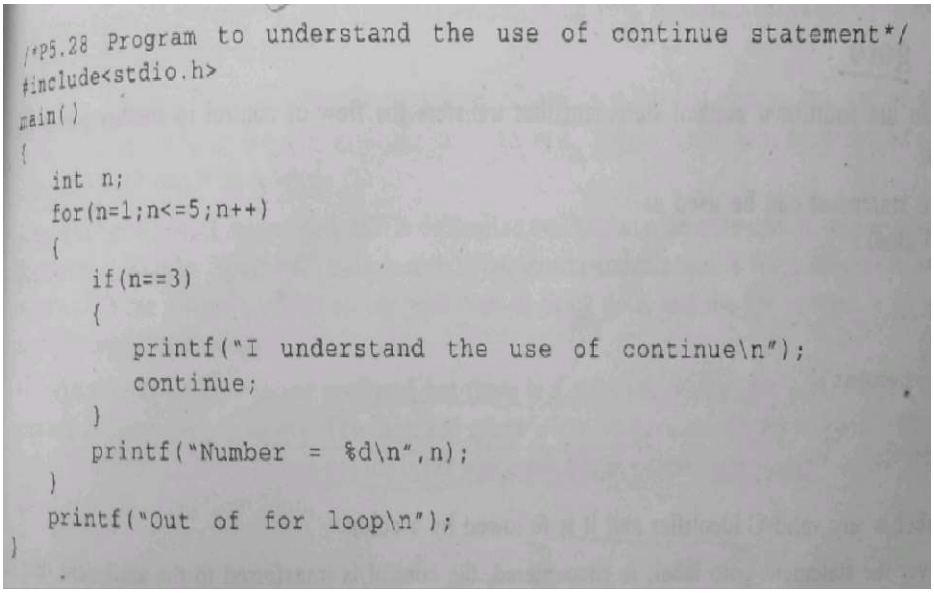
continue;



It is generally used with a condition. When a continue statement is encountered all the remaining statements (statements after continue) in the current iteration are not executed and the loop continues with the next iteration.

The **difference between break and continue** is that when a break statement is encountered the loop terminates and the control is transferred to the next statement following the loop, but when a continue statement is encountered the loop is not terminated and the control is transferred to the beginning of the loop.

In **while** and **do-while** loops, after continue statement the control is transferred to the test condition and then the loop continues, whereas in **for** loop after continue statement the control is transferred to update expression and then the condition is tested.



```
/*P5.28 Program to understand the use of continue statement*/
#include<stdio.h>
main()
{
   int n;
   for(n=1;n<=5;n++)
   {
      if(n==3)
      {
         printf("I understand the use of continue\n");
         continue;
      }
      printf("Number = %d\n",n);
   }
   printf("Out of for loop\n");
}
```

Output :

Number = 1
Number =2
I understand the use of continue
Number =4
Number = 5
Out of for loop

goto statement:- It is an unconditional statement and it transfers the control to the another part of the program. The goto statement is used as under:-

goto label;
.....
.....
.....
Label:

Statement;
.....
.....

Here **label** is any valid C identifier and is followed by colon.
The goto statement transfers the control to the statement after the label.
If the label is placed after the goto statement then the control is transferred forward and it is known as **forward jump**. If the label is placed before the goto statement then the control is transferred backward and the jump is called **backward jump**.

In the forward jump all the statements between the goto statement and the label are skipped .

In case of backward jump all the statement between the goto statement and the label are executed.

There should always be a statement after any label.