PROGRAMMING IN C

APEC

UNIT I BASICS OF C PROGRAMMING

Introduction to programming paradigms - Structure of C program - C programming: Data Types – Storage classes - Constants – Enumeration Constants - Keywords – Operators: Precedence and Associativity - Expressions - Input/Output statements, Assignment statements – Decision making statements - Switch statement - Looping statements – Pre-processor directives - Compilation process.

INTRODUCTION TO PROGRAMMING PARADIGMS

A computer program is a collection of instructions that performs a specific task when executed by a computer. Computer programming is the process of developing and implementing various sets of instructions to enable a computer to do a certain task.

A programming paradigm is a way of programming or a style of solving problems. Programming paradigms is a way to classify programming languages based on their features.

Paradigms of Programming

There are several ways to think about computation:

- a set of instructions to be executed
- a set of expressions to be evaluated
- a set of rules to be applied
- a set of objects to be arranged
- a set of messages to be sent and received

Imperative (Procedural) Paradigm

Imperative programming uses statements that change a program‘s state; it consists of commands for the computer to perform - that is like imperative verbs in the English language. It uses procedures, functions, subroutines or methods to split the program in little tasks. This makes it possibile to reuse the code many times in the program and makes it easier to understand and to maintain.

Example: Algol, FORTRAN, COBOL, and BASIC.

Declarative Paradigm

Declarative programming expresses the logic of computation without describing its control flow; in other words it describes what it does, but not how it does it.

Example: HTML, SQL

Structured Paradigm

Structured programming changes the approach of Imperative paradigm by removing global variables, and by removing the GOTO statements. It introduces local variables into block-like procedures, functions and so on.

APEC

Example: Pascal, C

Functional Paradigm

Functional programming treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.

Example: Lisp, Scheme.

Logic Paradigm

Logic programming enables the expression of logic; this means that it expresses a set of sentences in a logical form, which are facts and rules about some problem domain. It is useful in dealing with problems without an obvious explanation of what the function should be.

Example: Prolog.

Object oriented Paradigm

Object-oriented or OOP is a programming style where the main data is represented as objects, which are instances of classes. Classes have attributes (or data fields) and functions, more often called methods.

Example: Smalltalk

Multi-Paradigms

Many modern programming languages combine multiple paradigms to obtain the advantages of each. This gives to the programmers more freedom during development, but using more paradigms often increases the complexity of the program‘s themselves. The purpose of using multi-paradigms is to create programs that suit the nature of a specific problem.

Example: C++, JAVA and Common Lisp

APEC

FIRST C PROGRAM

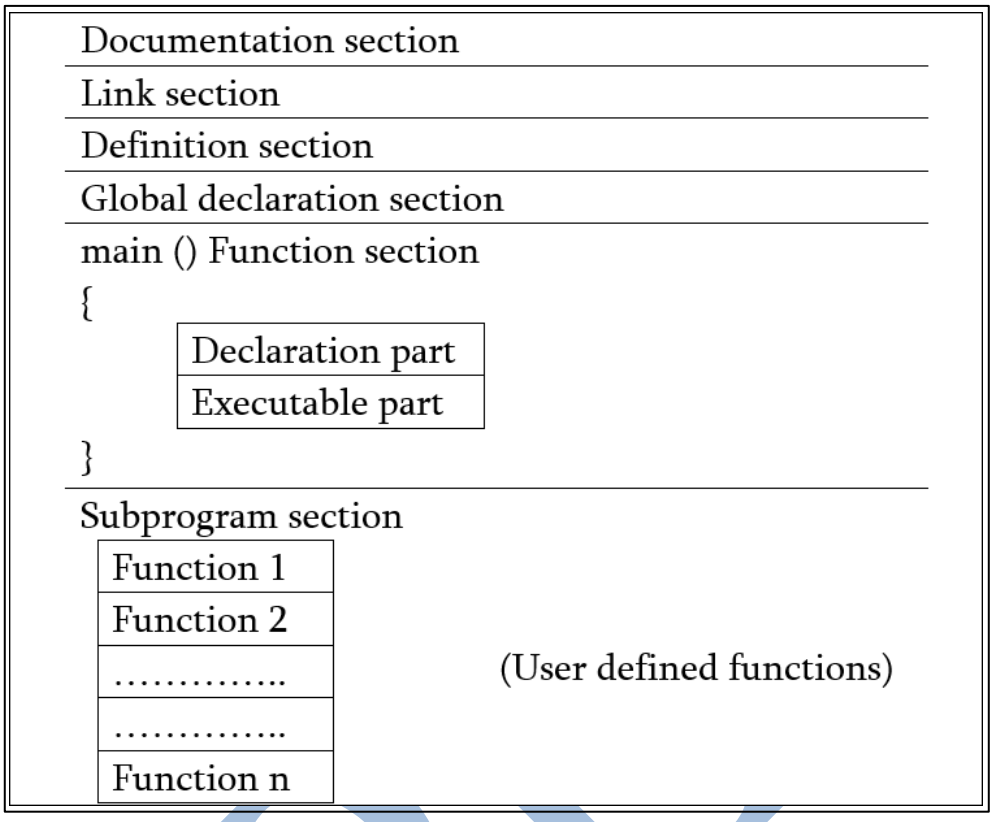
```
/* Author: C Programmer
Date: 09.01.2018
Description: simple program to print "Hello, World!"  */

#include<stdio.h>
void main()
{
    printf("Hello, World!\n");
}
```

/* Comments */	Comments are generally ignored by the compiler. Comments can be used anywhere in the program, which are helpful for developers to easily understand the existing code in the future.
#include<stdio.h>	stdio is standard for input/output, this allows to include a file called <stdio.h>. This is a pre-processor directive statement that tells the compiler to include the header file <stdio.h> in the program.
int/void	int/void is a return value. void indicates that the function doesn‘t return any value.
main()	Program execution begins in the main(). Every C program must contain only one main function.
{ .. }	Two curly brackets —{…}‖ are used to group all statements together.
printf()	It is a function in C, which prints text on the screen.

APEC

STRUCTURE OF C PROGRAM



Documentation section:

The documentation section consists of a set of comment lines giving the name of the program, the author and other details.

Link section:

The link section provides instructions to the compiler to link functions from the system library such as using the #include directive.

Definition section:

The definition section defines all symbolic constants such using the #define directive.

Global declaration section:

There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section. This section also declares all the user-defined functions.

main () function section:

Every C program must have one main function section. This section contains two parts; declaration part and executable part

- *Declaration part:* The declaration part declares all the variables used in the executable part.
- *Executable part:* There is at least one statement in the executable part. These two parts must appear between the opening and closing braces. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function is the logical end of the program. All statements in the declaration and executable part end with a semicolon.

Subprogram section:

If the program is a multi-function program then the subprogram section contains all the user-defined functions that are called in the main () function. User-defined functions are generally placed immediately after the main () function, although they may appear in any order.

Following is a sample C Program that contains all of the above mentioned sections.

```
/*
 * Name: simple Program to explain structure of the C Program
 * Author: C_programmer
 * Create Date: 09/01/2018
 * Last Modified: 22/01/2018
 */

//Linking required library
#include <stdio.h>

//defining a constant
#define LENGTH 20

//defining a global variable
int max_length = 15;

//declaring an user defined function which is defined later
```

APEC

```
float addNumbers(float a, float b);

void main()
{
    //declaring a local variable for main()
    int localVariable = 50;

    //Accessing a defined constant
    printf("Max array length is %d\n", LENGTH);

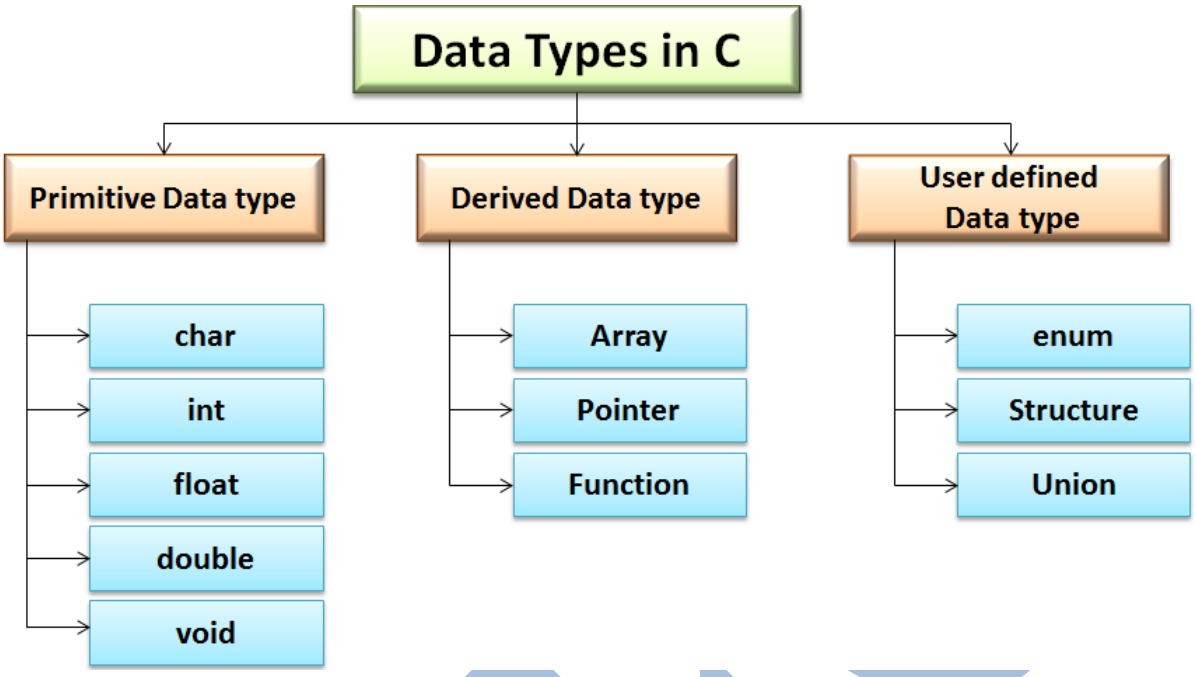
    //Accessing a global variable
    printf("Max input length is %d\n", max_length);

    //Accessing an user defined function
    printf("Summation of 5.5 and 8.3 = %f\n",addNumbers(5.5,8.3));
    }

//defining an user defined function which is declared earlier.
float addNumbers(float a, float b)
{
    return (a+b);
    }
```

APEC

DATA TYPES:



C provides various types of data-types which allow the programmer to select the appropriate type for the variable to set its value.

Primitive Data Types

Integer data types

Integers are whole numbers that can have both positive and negative values but no decimal values. In C programming, keyword *int* is used for declaring integer variable.

int mark;

Here, *mark* is a variable of data type integer. Multiple variables can also be declared at once.

int empno, id, age;

The integer data type has type modifiers like short, long, signed and unsigned.

Type	Size (Bytes)	Range	Specifier
int *(signed short int)*	2	-32,768 to 32,767	%d
short int *(signed short int)*	2	-32,768 to 32,767	%d
long int *(signed long int)*	4	2,147,483,648 to 2,147,483,647	%d
unsigned int *(unsigned short int)*	2	0 to 65,535	%u
unsigned short int	4	0 to 4,294,967,295	%u

Floating data types

Floating type variables can hold real numbers such as: 2.34, -9.382, 5.0 etc. A floating point variable is declared by using either float or double keyword.

```
float grossPay;

double itemRate;
```

Float	Double
Size of float (single precision float data type) is 4 bytes	Size of double (double precision float data type) is 8 bytes
Precision of float is 6 digits	Precision of double is 15 digits

Type	Size (Bytes)	Range	Specifier	Precision (decimal places)
float	4 byte	1.2E-38 to 3.4E+38	%f	6
double	8 byte	2.3E-308 to 1.7E+308	%d	15
long double	10 byte	3.4E-4932 to 1.1E+4932	%ld	19

Character data types

Keyword char is used for declaring character type variables.

```
char test = 'h';
```

Here, *test* is a character variable. The value of test is 'h'.

Type	Size (Bytes)	Range	Specifier
unsigned char	1 byte	0 to 255	%c
signed char	1 byte	-128 to 127	%c

Void data type

void actually refers to an object that does not have a value of any type. *void* data type is used in function definition it means that function will not return any value.

```
/* Simple C Program to explain data types */

#include <stdio.h>
void main()
{
   int a;
   char b;
   float c;
   double d;
   printf("Size  of  int   :%d  \n",sizeof(a));
   printf("Size  of  char   :%d  \n",sizeof(b));
   printf("Size  of  float  :%d  \n",sizeof(c));
   printf("Size of double  :%d \n",sizeof(d));
}
```

Execution:

Output

```
Size of int       4
Size of char      1
Size of float     4
Size of double    8
```

Derived Data Types

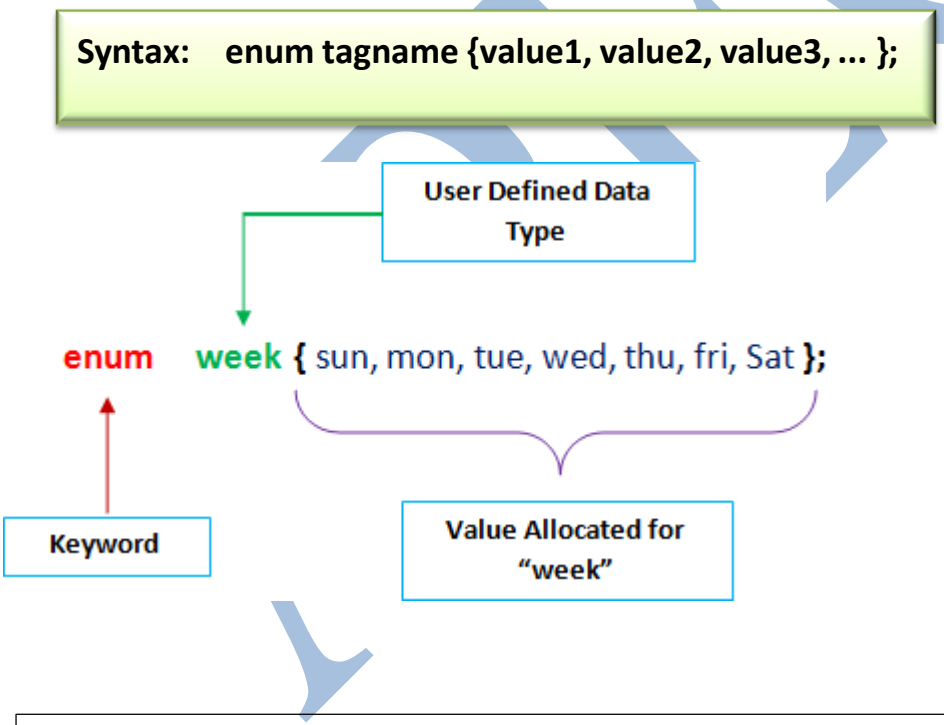
Data types	Description
Arrays	Arrays are sequences of data items having homogeneous values. They have adjacent memory locations to store values.
Functions	A function is a group of statements that together perform a task.
Pointers	It‘s the powerful C feature which is used to access the memory and deal with their addresses.

APEC

User Defined Data Types

Data types	Description
Structure	name. This is done to handle data efficiently. *struct* keyword It is a collection of variables of different types under a single is used to define a structure. Each element in a C structure is called member.
Union	C Union is also like structure, i.e. collection of different data types which are grouped together. Each element in a union is called member. Union and structure in C are same in concepts, except allocating memory for their members.
Enum	Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain. *enum* keyword is used to define the enumerated data type.

Enumeration in C




```
/* Simple C Program to explain Enumerated data type */

#include<stdio.h>
enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};
void main()
{
    enum week day;
```

```
    day = Wed;
    printf("Wednesday = %d",day+1);
}
```

Execution:

Output

```
Wednesday = 3
```

APEC

STORAGE CLASSES IN C

Storage Classes are used to describe about the features of a variable/function. These features basically include the scope, visibility and life-time which help to trace the existence of a particular variable during the runtime of a program.

- *Scope:* Where the value of the variable would be available inside a program?
- *Default initial value:* If the value is not explicitly initialized, what will be its default initial value?
- *Lifetime:* How long will the variable exist?

The following storage classes are most often used in C programming,

- Automatic variables
- External variables
- Static variables
- Register variables

Automatic variables

By default, any variable declared inside a function without any storage class specification, is called as automatic variable. They are created when a function is called and are destroyed automatically when the function's execution is completed. Automatic variables can also be called local variables because they are local to a function. They are assigned garbage value by the compiler.

- *Scope:* Variable defined with auto storage class are local to the function block
- *Default Initial Value:* Any random value i.e garbage value.
- *Lifetime:* Only within the function/method block.

Example:

```
int number;
// or

auto int number;        //Both are same
```

APEC

External or Global variables

A variable that is declared outside the function is a Global Variable. Global variables remain available throughout the program execution. By default, initial value of the Global variable is 0(zero).

- *Scope:* Global i.e everywhere in the program.
- *Default initial value:* 0(zero).
- *Lifetime:* Accessed till the program completes.

```
/* Simple C Program to explain Global variable */
#include<stdio.h>

int number;      // global variable
void main()
{
    printf("Main function value is %d\n", number);
    fun1();      //function calling
    fun2();      //function calling
}

fun1()
{
    number = 20;
    printf("fun1() value is %d", number);
}

fun2()
{
    printf("\nfun2() value is %d", number);
}
```

Execution:

Output

```
Main function value is 0
fun1() value is 20
fun2() value is 20
```

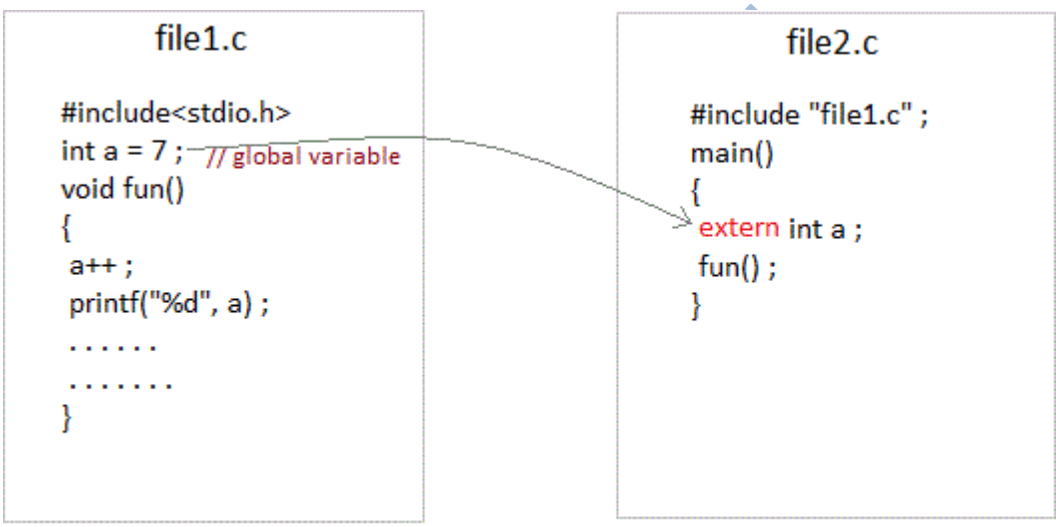
APEC

Note:

If one function changes the value of the variable, it gets changed in every function.

extern keyword

The extern keyword is used to inform the compiler that this variable is declared somewhere else. The extern declaration does not allocate storage for variables.



global variable from one file can be used in other using **extern** keyword.

Static variables

Static variable is initialized only once and remains into existence till the end of the program. A static variable can either be internal or external depending upon the place of declaration.

- *Scope:* Local to the block in which the variable is defined
- *Default initial value:* 0(Zero).
- *Lifetime:* Till the whole program completed.

```
/* Simple C Program to explain static variables */
#include<stdio.h>

void test();     //Function declaration
void main()
{
    test();
    test();
    test();
}

void test()
{
    static int a;        //a static variable
    a = a + 1;
    printf("%d\t",a);
}
```

Execution:

Output

```
1     2     3
```

Register variable

Register variables are also local variables, but stored in register memory; whereas, auto variables are stored in main CPU memory. Register variables will be accessed very faster than the normal variables since they are stored in register memory rather than main memory.

- *Scope:* Local to the function in which it is declared.
- *Default initial value:* Any random value i.e garbage value
- *Lifetime:* Till the end of function/method block.

Syntax:

$$register\ int\ number;$$

APEC

C TOKENS

C token is the smallest element of a program that is meaningful to the compiler. Tokens can be classified as follows:

- Keywords
- Identifiers
- Constants
- Special Symbols
- Operators

Keywords

Keywords are predefined sets of reserved words that have special meaning in a program. Keywords cannot be used as identifiers in a program.

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

Identifiers

Identifiers are used as the general terminology for naming of variables, functions and arrays.

Rules for naming c identifiers:

- Identifiers can be a combination of lowercase letters (a to z) or uppercase letters (A to Z) or digits (0 to 9) or an underscore (_).
- Identifiers cannot start with the digit.
- Keywords cannot be used as identifiers and identifiers can be any length.
- No special symbols like !, @, #, $, % etc. are used.

APEC

Valid Identifiers: sum total average _ab_ add_1 x1

Invalid Identifiers: 1x char x+y

Example:

```
void test()
{
    int a = 10;
}
```

In the above program there are 2 identifiers:

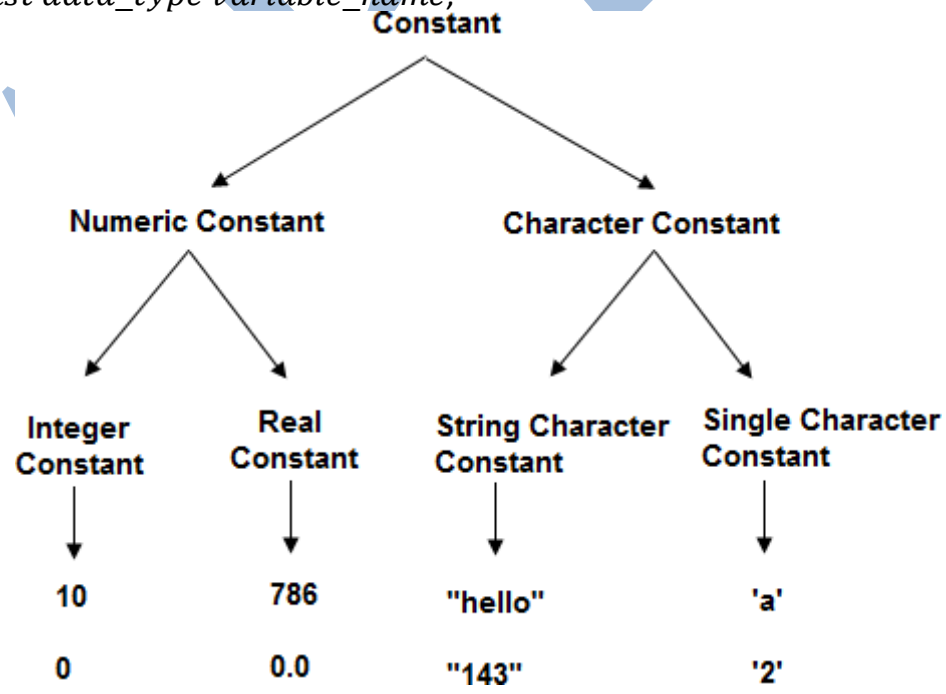
- *test*: method name.
- *a*: variable name.

Constants

C constants refer to the variables that do not change their value during the program execution. They are also called as literals. Constants may belong to any of the data type.

Syntax:

```
const data_type variable_name;
```



Types of Constants:

- *Integer constants* – Example: 0, 1, 1218, 12482
- *Real or Floating point constants* – Example: 0.0, 1203.03, 30486.184
- *Octal & Hexadecimal constants* – Example: octal: $(013)_8 = (11)_{10}$,
  Hexadecimal: $(013)_{16} = (19)_{10}$
- *Character constants* -Example: ‗a', ‗A', ‗z'
- *String constants* -Example: ―C Programming‖

Example 1:

```
const int n = 10;
int area;
area = n * n;

printf("Area of the square = %d ",area);
```

Output : Area of the square = 100

Example 2:

```
const int n = 10;
n = n + 1;

printf("Number incremented = %d ",n);
```

Output : Error

Special Symbols

The following special symbols are used in C having some special meaning and thus, cannot be used for some other purpose:

- *Brackets[]:* Used for array elements
- *Parentheses():* Used to indicate function calls and function parameters.
- *Braces{}:* Indicate the start and end of a block of code
- *comma (,):* Used to separate more than one statements
- *asterick (*):* Used to create pointer variable.
- *assignment operator:* Used to assign values.

APEC

- *pre processor(#):* Used to indicate the preprocessor directive statements.

Volatile Variables

Volatile variables are those variables that are changed at any time by same program or other external program.

Syntax:

```
volatile data_type variable_name;
```

Example:

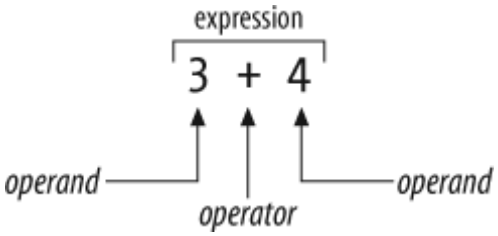
```
volatile int n = 10;
n = n + 1;

printf("Number incremented = %d ", n);
```

Output : Number incremented = 11

APEC

OPERATORS IN C



An operator is a symbol that tells the compiler to perform mathematical or logical calculations on operands (variables). List of operators in C are:

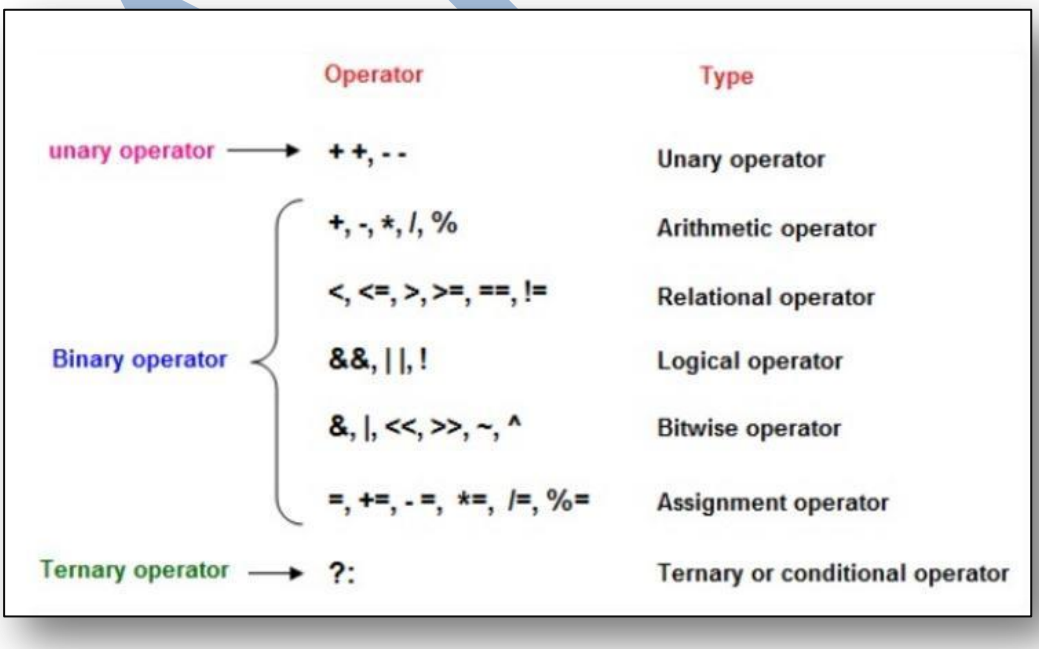
- Arithmetic Operators
- Assignment Operator
- Relational or comparison Operators
- Increment and decrement Operators
- Logical Operators
- Bitwise Operators

Binary Operators:

Operators that have two operands are called binary operators.

Unary Operators:

C provides two unary operators for which only one variable is required.



Arithmetic operators:

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Operator	Name	Expression	Example	Output
+	Addition	x+y	15 + 4	19
-	Subtraction	x-y	15 - 4	11
*	Multiplication	x*y	15 * 4	60
/	Division	x/y	15 / 4	3.75
%	Modulus	x%y	15 % 4	3

Comparison operators

Comparison operators are also called Relational operators. They are used to compare values and return True or False condition.

Operator	Name	Expression	Example	Output
>	Greater than	x>y	54 > 18	True
<	Lesser than	x<y	54 < 18	False
==	Equal to	x==y	54 == 18	False
!=	Not Equal to	x!=y	54 != 18	True
>=	Greater than or equal to	x>=y	54 >= 18	True
<=	Lesser than or equal to	x<=y	54 <= 18	False

Logical operators

Logical operators are used to combine the results of two or more conditions.

Operator	Name	Expression	Explanation
&&	Logical and	x && y	True if both the operands are true
||	Logical or	x || y	True if either of the operands is true
!	Logical not	!x	True if operand is false

Bitwise Operators

Bitwise operators are used to manipulate the data at the bit level. Numbers are represented as bits (0‘s and 1‘s).

Bitwise AND

- Both inputs = 1, then output = 1.
- If one or both bits are 0, the output = 0 for that bit

Bitwise OR

- One or both inputs = 1, the output is 1.
- If both bits are 0, the output = 0 for the bit.

For Bitwise XOR

- one input is 1 and the other 0, the output = 1.
- If both bits are 1 and both are 0, the output is 0 for the bit.

A	B	A | B	A & B	A^ B	~A
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

Operator	Name	Example	Explanation
&	Bitwise AND	X & y	Both operands are true, then the result is true
|	Bitwise OR	x | y	If either of the operands are true, then the result is true
~	Bitwise NOT	~x	It complements the bits
^	Bitwise XOR	x ^ y	Result is true, when either of the operands are true, but not both
>>	Bitwise right shift	x>> 2	Operands are shifted right by the number of times specified
<<	Bitwise left shift	x<< 2	Operands are shifted left by the number of times specified

Difference between logical && and bitwise &:

- Logical && - Do not evaluate the second operand if the first operand is fals
- Logical || - Do not evaluate the second operand if the first operand is true
- But, bitwise & and |, always evaluate their operands

Assignment operators

Assignment operators are used to assign a value or an expression or a value of a variable to another variable.

Operator	Name	Example	Equivalent to
=	Equal to	x = 10	x = 10
+=	Plus Equal to	x += 10	x = x + 10
-=	Minus Equal to	x -= 10	x = x - 10
*=	Multiplication Equal to	x *= 10	x = x * 10
/=	Division Equal to	x /= 10	x = x / 10
%=	Modulus Equal to	x %= 10	x = x % 10
&=	AND Equal to	x &= 10	x = x &10
|=	OR Equal to	x |= 10	x = x | 10
^=	XOR Equal to	x ^= 10	x = x ^ 10
>>=	Right shift Equal to	x >>= 2	x = x >>2
<<=	Left shift Equal to	x >>= 2	x = x >>2

Increment and decrement operators

C programming has two operators: auto increment ++ and auto decrement -- to change the value of an operand (constant or variable) by 1. Auto increment ++ increases the value by 1 whereas auto decrement -- decreases the value by 1. These two operators are unary operators, as they only operate on a single operand.

```
i = 4;      i is 4
j = ++i;    j is 5, i is 5    (i was incremented before use)
j = i++;    j is 5, i is 6    (i was incremented after use)
j = --i;    j is 5, i is 5    (i was decremented before use)
j = i--;    j is 5, i is 4    (i was decremented after use)
i++;        i is 5            (i was incremented)
```

APEC

Other Operators

Comma Operator

Comma operators are used to link related expressions together. For example:

int a, c = 5, d;

sizeof operator

The sizeof is an unary operator which returns the size of data (constant, variables, array, structure etc). For example:

int a;

sizeof(a)) return 4 bytes.

EXPRESSIONS

An expression is a combination of variables, constants and operators. Every expression is evaluated to a value i.e., every expression results in some value of a certain type that is assigned to a variable.

Some examples of C expressions are given below.

Algebraic Expression	**C Expression**
a x b – c	a * b – c
(m + n) (x + y)	(m + n) * (x + y)
(ab / c)	a * b / c
3x2 +2x + 1	3*x*x+2*x+1
(x / y) + c	x / y + c

Expressions are evaluated using an assignment statement of the form

Variable = Expression;

Example of evaluation statements are

x = a * b – c

y = b / c * a

z = a – b / c + d;

C Precedence and Associativity of Operators:

If more than one operator is involved in an expression, C language has a predefined rule of priority for the operators. This rule of priority of operators is called operator precedence. Operator associativity is used to determine the order of operators with equal

precedence evaluated in an expression. In c programming language, when an expression contains multiple operators with equal precedence, we use associativity to determine the order of evaluation of those operators.

Table Summary of C operators with precedence and associativity		
Operator	**Meaning of operator**	**Associativity**
( ) [ ] . -> ++ —	Parentheses (function call) Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement	left-to-right
++ — + – ! ~ (type) * & sizeof	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Cast Dereference Address (of operand) Determine size in bytes on this implementation	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ –	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
=	Assignment	right-to-left
,	Comma	left-to-right

Type Casting

Type conversion or typecasting refers to changing an entity of one data type into another.

There are two types of conversion:

- Implicit Conversion - It‘s called as coercion or automatic type conversion
    - All the data types are upgraded to largest data type.

      bool -> char -> short int -> int ->

      unsigned int -> long -> unsigned ->

      long long -> float -> double -> long double
- Explicit Conversion - It‘s called as type casting.

- Force the data type of the expression to specific type is known as explicit type conversion

```
/* Simple C Program to explain Implicit type conversion */

#include<stdio.h>
void main()
{
    int x = 10;     // integer x
    char y = 'a';  // character c
    // y implicitly converted to int. ASCII
    x = x + y;          // value of 'a' is 97

    // x is implicitly converted to float
    float z = x + 1.0;
    printf("x = %d, z = %f", x, z);
}
```

Execution:

Output

```
x = 107, z = 108.000000
```

```
/* Simple C Program to explain Explicit type conversion */

#include<stdio.h>
void main()
{
    double x = 1.2;

    // Explicit conversion from double to int
    int y = (int)x + 1;
    printf("\nSum = %d", y);
}
```

Execution:

Output

Sum = 2

APEC

INPUT OUTPUT STATEMENTS

Input functions

- ***Formatted I/P:*** scanf(), fscanf()
- ***Unformatted I/P:*** getchar(),gets(),getc(),getche()

Output functions

- ***Formatted I/P:*** printf(),fprintf()
- ***Unformatted I/P:*** putchar(),puts(),putc()

scanf

scanf is used when we enter data by using an input device.

Syntax:

$$scanf\ (\text{“}format\ string\text{”}, \&arg1, \&arg2, \ldots..);$$

Example 1:

$int\ avg;$

$scanf(\text{“}\%d\text{”}, \&n);$

The scanf() function reads formatted input from the keyboard. When user enters an integer, it is stored in variable n.

Example 2:

$int\ avg;$

$float\ per;$

$char\ grade;$

$scanf(\text{“}\%d\ \%f\ \%c\text{”}, \&avg, \&per, \&grade)$:

printf

This function is used for displaying the output on the screen.

Syntax:

$$printf(\text{“}format\ string\text{”}, arg1, arg2, \ldots..);$$

—'format string‖ → contain the information to be formatted. arg1, arg2 a→output data items.

Example:

APEC

$printf(“Enter\ a\ value:”);$

getchar()

This Input function is used for reading a single character from the keyboard.

Example:

$char\ n;$

$n\ =\ getchar();$

putchar()

This output function is used to display a single character on the screen.

Example:

$char\ n;$

$putchar(n);$

gets()

This input function is used to read a string from the keyboard.

Example:

$char\ n[20];$

$gets(n);$

puts()

This output function is used to display a string inputted by gets() function.

Example:

$char\ name[20];$

$puts("Enter\ the\ Name");$

$gets(name);$

$puts("Name\ is\ :\ ");$

$puts(name);$

getch(), getche() and putch()

APEC

- getch() is used to get a character from console but does not echo to the screen. It reads a single character directly from the keyboard, without echoing to the screen.
- getche() is used to get a character from console, and echoes to the screen.
- putch displays characters to the standard output device. It displays only one character at a time

Format Specifiers

There are many format specifiers defined in C. Frequently used specifiers are:

%i or %d	int
%c	char
%f	float
%s	string

Escape Sequence or newline character

Escape sequences are basically control characters used for formatting the output. These are combinations of a backslash and a character in lowercase.

Commonly used escape sequences are:

- \b (backspace)
- \n New Line
- \r Carriage Return
- \t Tab (Horizontal)
- \v Vertical Tab
- \\ Backslash
- \' Single Quote
- \" Double Quote
- \? Question Mark

```
/* Simple C Program to explain formatted output statements */

#include<stdio.h>
```

APEC

```
void main()
{
  int a,b;
  float c,d;
  a = 15;
  b = a / 2;
  printf("%d\n",b);
  printf("%3d\n",b);       //3 spaces allocated for the result
  printf("%03d\n",b);     // empty spaces filled with zeros
  c = 15.3;
  d = c / 3;
  printf("%5.2f\n",d);   //5 spaces totally; 2 digits after decimal
}
```

Execution:

Output

```
7
  7
007
5.10
```

```
/* Simple C Program to explain different format specifiers */

#include <stdio.h>
void main()
{
   char ch = 's';
   char str[20] = "C Programming";
   float flt = 120.234;
   int no = 897;
   double dbl = 40.934356;
   printf("Character is %c \n", ch);
   printf("String is %s \n" , str);
   printf("Float value is %f \n", flt);
   printf("Integer value is %d\n" , no);
   printf("Double value is %lf \n", dbl);
   printf("Octal value is %o \n", no);
```

APEC

```
    printf("Hexadecimal value is %x \n", no);
}
```

Execution:

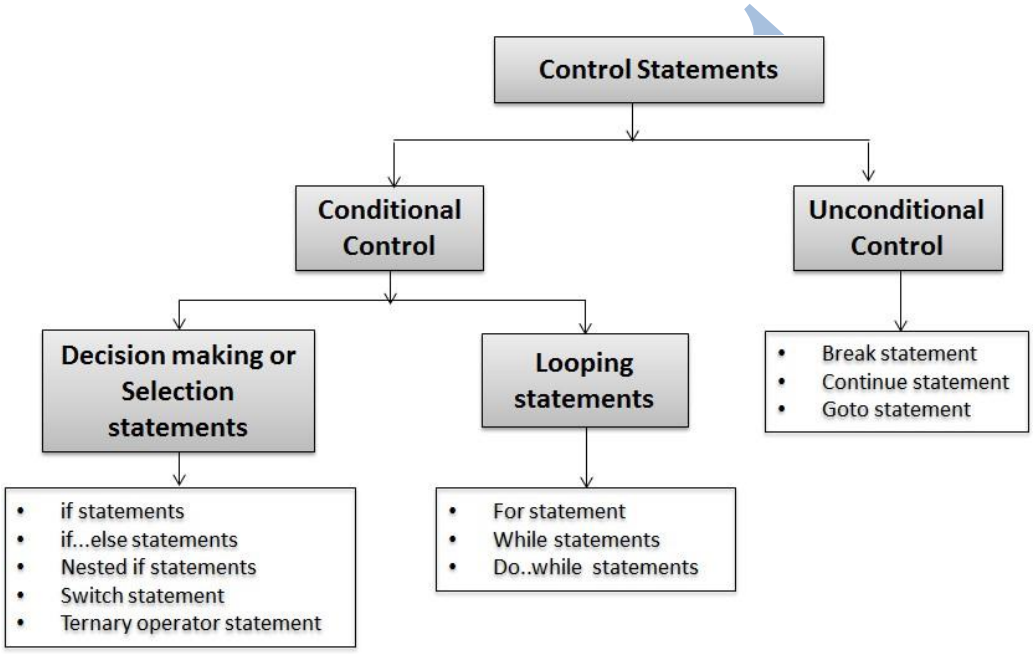
Output

```
Character is s
String is C Programming
Float value is 120.234001
Integer value is 897
Double value is 40.934356
Octal value is 1601
Hexadecimal value is 381
```

APEC

CONTROL STATEMENTS

C programs are sequential in nature. Statements are executed in the order they are specified in the program. This way of executing statements sequentially is known as Sequential control statements. Conditional statements help to make decision based on certain conditions.



DECISION MAKING STATEMENTS

Decision making is about deciding the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met. Following is the general form of decision making structure found in most of the programming languages.

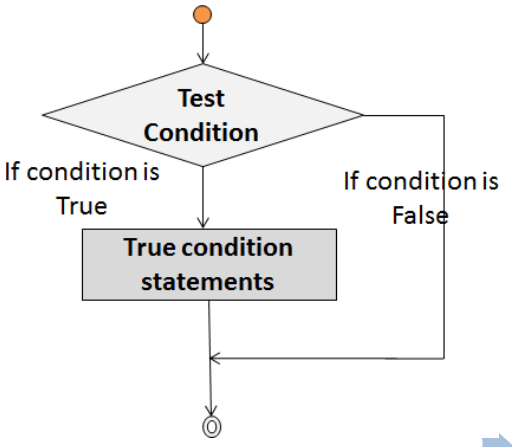


Figure Decision making structure

C supports the following types of conditional checking statements.

- *if statements*
- *if...else statements*
- *Nested if statements*
- *Switch statement*
- *Ternary operator statement*

if statements

The program evaluates the test condition and executes the statement(s) only if the text condition is True.

Syntax:

```
if(test condition)
{
   True condition statement(s);

    }
```

Flowchart for if statement:

APEC

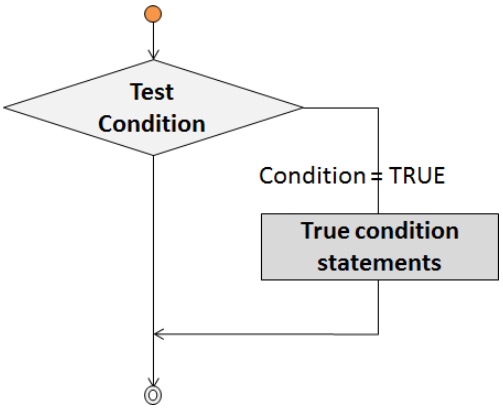


Figure Flowchart for if statement

APEC

```
/* Simple C Program to explain if structure */

#include <stdio.h>
void main( )
{
    int x, y;
    x = 15;
    y = 13;
    if (x > y )
    {
        printf("x is greater than y");
    }
}
```

Execution:

Output

```
x is greater than y
```

if..Else statements

The program evaluates the test condition and executes the statement(s) only if the text condition is True. If the text condition is False, the else statement(s) are executed.

Syntax:

$if(test\ condition)$
{
$True\ condition\ statement(s);$
$else$ }
{ $False\ condition\ statement(s);$
}

Flowchart for if...else statements:

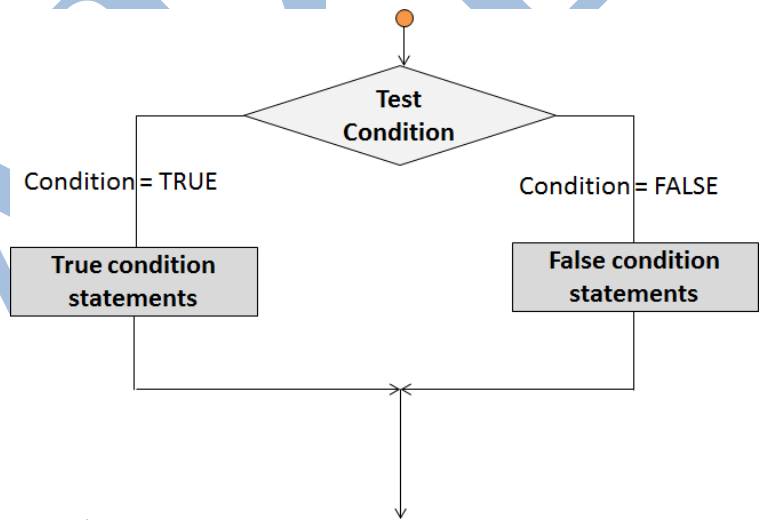


Figure Flowchart for if...else statements

```
/* Simple C Program to explain if..else structure */

#include <stdio.h>
void main( )
{
    int x, y;
    x = 15;
    y = 18;
    if (x > y )
    {
        printf("x is greater than y");
    }
    else
    {
        printf("y is greater than x");
    }
}
```

Execution:

Output

```
y is greater than x
```

Nested if statements

if..else statement can be used inside another if..else statement. This is called nesting in computer programming.

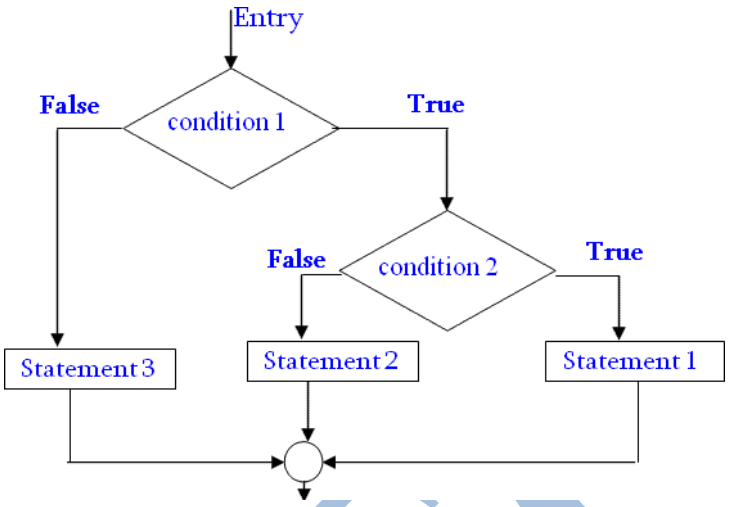
Syntax:

```
if( outer test condition )
{
    if( inner test condition )
    {
        True statement(s);
    }
    else
    {
        False statement(s);
    }
else
{
    False statement(s);

    }
```

APEC

Flowchart for nested...if statements:




```
/* Simple C Program to explain nested..if structure */

#include <stdio.h>

void main( )
{
    int a, b, c;
    printf("Enter 3 numbers...");
    scanf("%d%d%d",&a, &b, &c);
    if(a > b)
    {
        if(a > c)
        {
            printf("a is the greatest");
        }
        else
        {
            printf("c is the greatest");
        }
    }
    else
    {
```

APEC

```
        if(b > c)
        {
            printf("b is the greatest");
        }
        else
        {
            printf("c is the greatest");
        }
    }
}
```

Execution:

Input:

Enter 3 numbers... **56,89,23**

Output

b is the greatest

switch statements

The switch statement is a multi-way branch statement. Switch statement is a control statement that allows us to choose only one choice among the many given choices. It executes the block of code which matches the case value. If there is no match, then default block is executed.

Syntax:

```
switch(expression)
{
  case value 1:
            block 1;
            break;
  case value 2:
            block 2;
            break;
  case value 3:
            block 3;

            break;
```

APEC

.

.

.

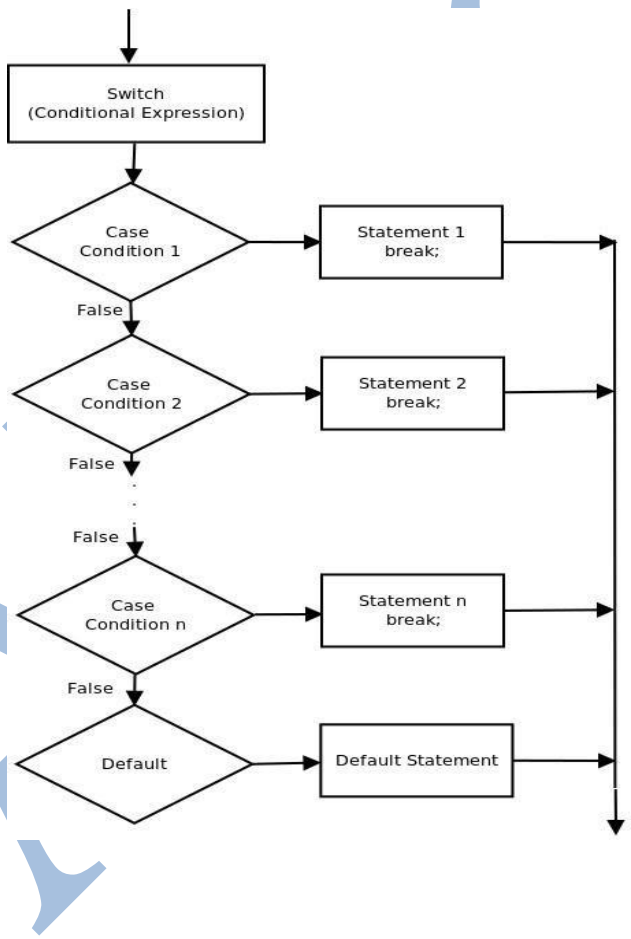
$default - block;$

$defaul$ $break;$

t:

}

Flowchart for switch statements:




```
/* Simple C Program to explain switch case statement */

#include<stdio.h>

void main()
```

```
{
int n;
printf("Enter any number (1 to 7)");
scanf("%d",&n);
switch(n)
{
    case  1:
        printf("Today is Monday");
        break;
    case  2:
        printf("Today is Tuesday");
        break;
    case  3:
        printf("Today is Wednesday");
        break;
    case  4:
        printf("Today is Thursday");
        break;
    case  5:
        printf("Today is Friday");
        break;
    case  6:
        printf("Today is Saturday");
        break;
    case  7:
        printf("Today is Sunday");
        break;
    default:
        printf("Wrong number entered");
}
}
```

Execution:

Input

```
Enter any number (1 to 7): 5
```

Output

```
Today is Friday
```

APEC

Ternary operator

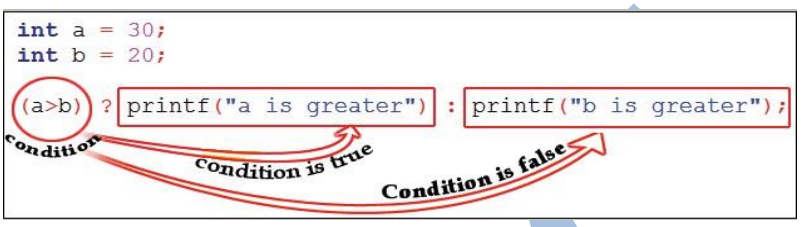
Ternary operator includes three parts.

- First part (?) is a conditional expression that returns boolean value true or false.

- Second part (after ? and before :) contains a statement which will be returned if the conditional expression is true.
- Third part returns the statement if the conditional expression returns false.

Syntax:

$$result = conditional\ expression\ ?\ True\ statement\ :\ False\ statement$$




```
/* Simple C Program to explain switch case statement */

#include <stdio.h>
void main()
{
int a = 30;
int b = 20;
(a>b)? printf("A is greater"): printf("B is greater");
}
```

Execution:

Output

```
A is greater
```

LOOPING STATEMENTS

Computers often do repetitive tasks. There are situations when programmers need to execute a block of code several number of times. Repeated execution of a set of statements is called iteration or looping.

Looping statement execute one or more statement repeatedly several number of times. In C there are three types of loops.

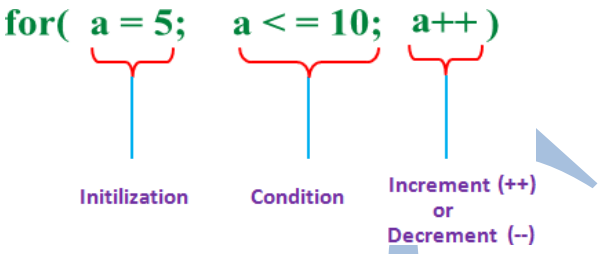
- *for statement*

APEC

- *while statement*
- *do-while statement*

FOR loop

For loop executes a block of statements repeatedly until a given condition is true.



Syntax:

$for\ (initialization;\ condition\ test;\ increment\ or\ decrement)$

{

$//Statements\ to\ be\ executed\ repeatedly$

}

Flowchart for FOR loop:

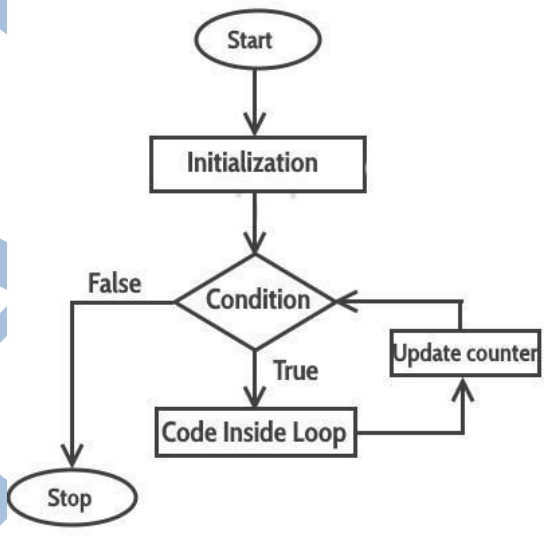


Figure Flowchart – For loop

```
/* Simple C Program to explain for loop */
```

```
#include <stdio.h>
void main()
{
   int i;
   for (i=1; i<=3; i++)
   {
       printf("%d\t", i);
   }
}
```

Execution:

Output

```
1     2     3
```

APEC

WHILE loop

While loop repeatedly executes a set of statement as long as the condition is true.

Syntax:

```
while(condition)
{
  statement(s);

}
```

Flowchart for While loop:

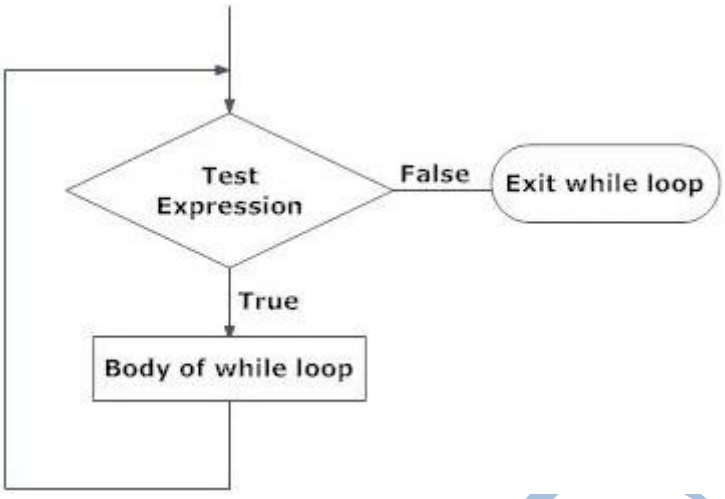


Figure Flowchart – While loop

```
/* Simple C Program to explain while loop */

#include <stdio.h>
void main()
{
  int i=3;
  while(i<10)
  {
     printf("%d\t ",i);
     i++;
  }
}
```

Execution:

Output

```
3       4       5       6       7       8       9
```

APEC

DO-WHILE loop

Unlike for and while loops, the do...while loop checks its condition at the bottom of the loop.

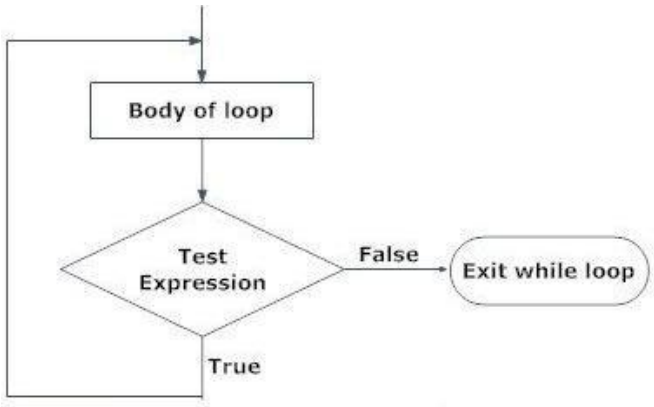
Syntax:

do

{

statement(*s*);
}
while(*condition*);

Flowchart for do..while loop:



APEC

```
/* Simple C Program to explain do..while loop */

#include <stdio.h>
void main ()
   {
   /* local variable definition */
   int a = 10;
   /* do loop execution */
   do {
      printf("%d\t", a);
      a = a + 1;
   } while( a < 15 );
}
```

Execution:

Output

```
10    11    12    13    14
```

Difference between while and do..while loop	
while loop	**do..while loop**
While loop is an entry controlled loop.	Do..while loops is an exit controlled loop
Since condition is checked first, the looping statements may or may not executed.	Since condition is checked later, the looping statements will execute at least once.

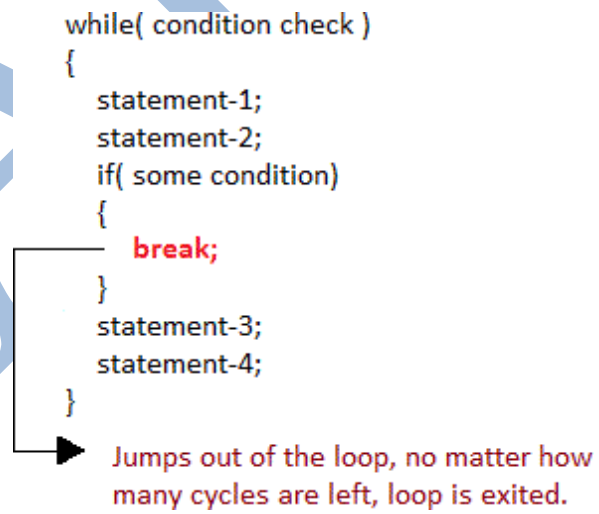
Unconditional control statements

For loop, *while* loop, and *do..while* loop statements execute a block until a certain condition is met. Sometimes it is desirable to skip some statements inside the loop or terminate the loop immediately without checking the test expression. In such cases, break and continue statements are used.

The break statement:

When the break statement is encountered inside a loop, the loop is immediately terminated and program control jumps to the next statement following the loop.

- It is used along with the Switch statement and looping statements
- Syntax:



APEC

```
/* Simple C Program to explain break statement */

#include<stdio.h>
void main()
{
  int i;
  for(i=1; i<=10; i++)
  {
    if(i==5)
        break;
    printf("hello\t");
    }
  printf("\nYou came out of loop");
}
```

Execution:

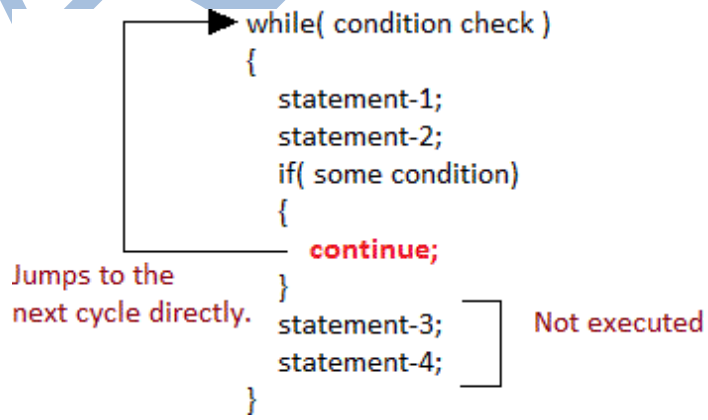
Output

```
hello hello hello hello hello
You came out of loop
```

The continue statement:

The continue statement skips the rest of the loop statements and causes the next iteration of the loop.

Syntax:



APEC

```
/* Simple C Program to explain continue statement */

#include<stdio.h>
void main()
{
 int i;
 for(i=1; i<=10; i++)
  {
    printf("hello\t");
    if(i==5)
        continue;
    }
  printf("\nYou came out of loop");
}
```

Execution:

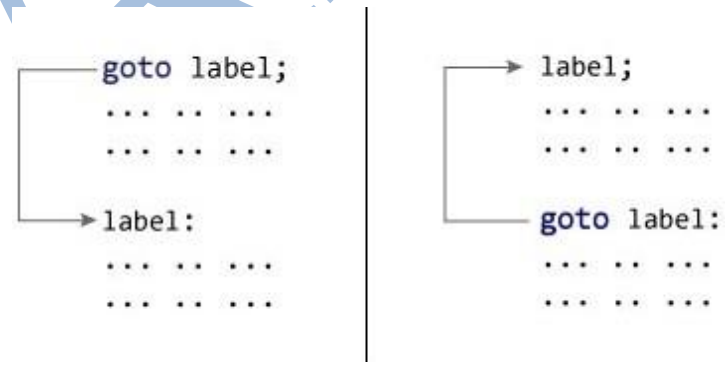
Output

```
hello hello hello hello hello hello hello hello hello hello
You came out of loop
```

The goto statement

The 'goto' statement transfer the control of the program to another part of the program where a particular label is defined.

Syntax:



APEC

```
/* Simple C Program to explain continue statement */

#include<stdio.h>
void main()
{
int i;
for(i=1; i<=10; i++)
{
    printf("hello\t");
    if(i==5)
        goto L1;
    }
  printf("\nYou came out of loop");
L1:
  printf("\nThis is the Label statement");
}
```

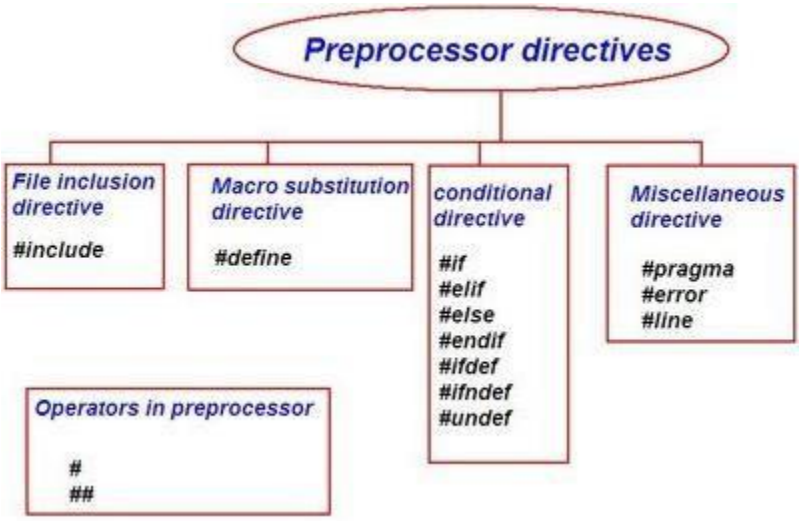
Execution:

Output

```
hello hello hello hello hello
This is the Label statement
```

APEC

PRE-PROCESSOR DIRECTIVES



Before a C program is compiled in a compiler, source code is processed by a program called pre-processor. This process is called pre-processing. Commands used in pre-processor are called pre-processor directives and they begin with ―#‖ symbol.

Macro

- Syntax: $\#define$
- Macro is a segment of code which is replaced by the value of macro. This macro defines constant value and can be any of the basic data types.

Header file inclusion

- Syntax: $\#include < file_name >$
- The source code of the file ―file_name‖ is included in the main program at the specified place.

Conditional compilation

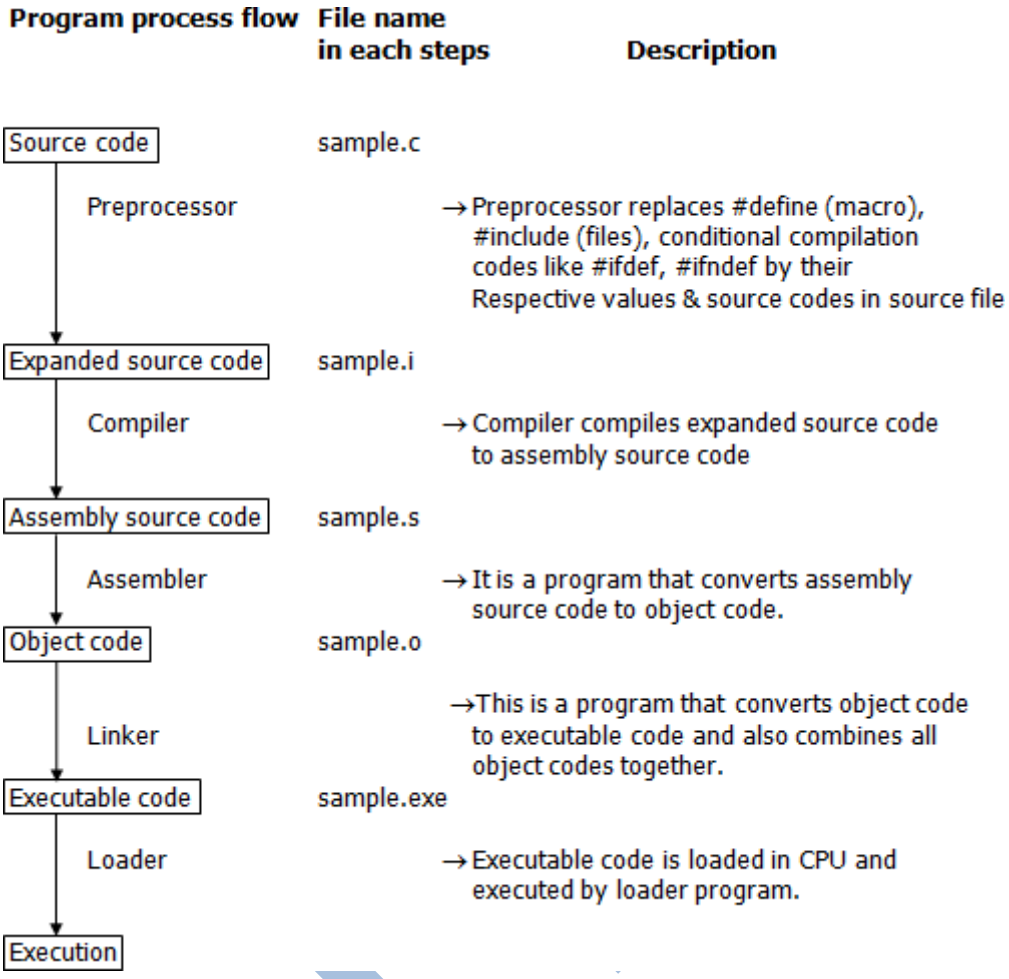
- Syntax: $\#ifdef, \#endif, \#if, \#else, \#ifndef$
- Set of commands are included or excluded in source program before compilation with respect to the condition.

Other directives

- Syntax: $\#undef, \#pragma, \#error$
- #undef is used to undefine a defined macro variable. #Pragma is used to call a function before and after main function in a C program.

APEC

The program in C language involves different processes and are :



/* Simple C Program to explain define and undefine statements */

```
#include <stdio.h>
#define PI 3.1415
void main()
{
   printf("%f",PI);
}
```

Execution:

Output

```
3.141500
```

```
#include <stdio.h>
#define PI 3.1415
#undef PI
void main()
{
   printf("%f",PI);
}
```

Execution:

Output

```
error: 'PI' undeclared
```

APEC