UNIT - I

INTRODUCTION TO C

'C' is one of the most popular programming language, it was developed by Dennis Ritchie at AT & T's Bell laboratories at USA in 1972.

FEATURES AND APPLICATIONS OF 'C' LANGUAGE

i) 'C' is a general purpose, structured programming language.

ii) 'C' is powerful, efficient, compact and flexible.

iii) 'C' is highly portable (i.e., It can be run in different operating systems environmental).

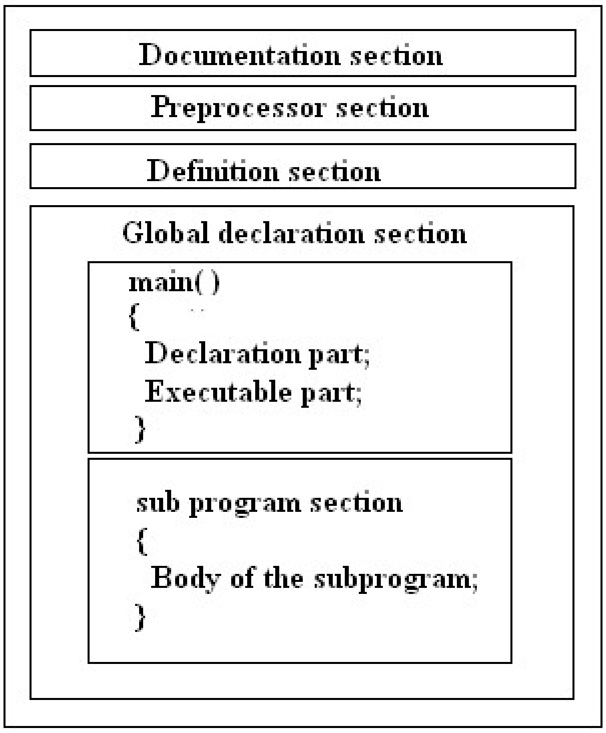
iv) 'C' is well suited for writing system software as well as application software.

v) 'C' is a middle level language, i.e. it supports both the low level language and high level language features.

vi) 'C' programs are fast and efficient.

STRUCTURE OF A 'C' PROGRAM

A 'C' program may contain one or more sections given below.



i) **Documentation section:** It consists a set of comment lines used to specify the name of program, the author and other details etc.,

ii) **Comments:** Comments are very helpful in identifying the program features and under

laying logic of the program. The lines begins with '/*' and ending with '*/' are known as comment lines. These are not executable, the compiler is ignored any thing in between /* and */. iii) **Preprocessor section:** It is used to link system library files, for defining the macros and for defining the conditional inclusion.

Eg: #include<stdio.h>, #define A 10, #if def, # endif… etc.

iv) **Global Declaration Section:** The variables that are used in more than one function.

throughout the program are called global variables and declared outside of all the function i.e., before main( ).

Every 'C' program must have one main( ) function, which specify the starting of 'C' program. It contains the following two parts.

v) **Declaration part**: This part is used to declare all the variables that are used in the executable part of the program and these are called local variables.

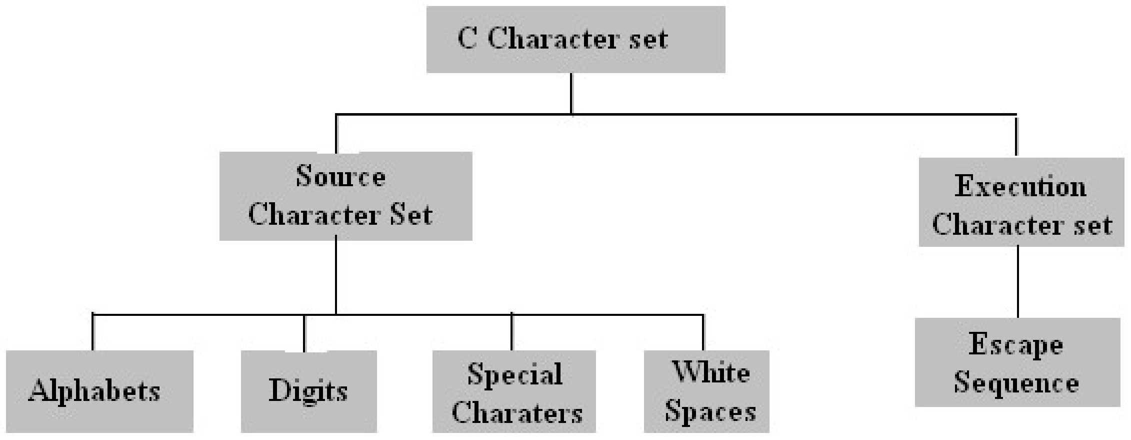
vi) **Executable part** : It contains atleast one valid 'C' statement. The execution of a program begins with opening brace '{' and ends with closing brace '}'. The closing brave of the main function is the logical end of the program.

'C'S CHARACTER SET

The character set is the fundamental raw material of any language and they are used to represent information.

'C' programme are basically of two types, namely

1. **Source character set**
2. **Execution character set**



Source character Set

These are used to construct the statements in the source program These are of four types

Source character set	Notation
Alphabets	A to Z and a to z
Decimal Digits	0 to 9
White Spaces	Blank space, Horizontal tabs, Vertical tab, New line, Form feed.
Special Characters	+ Plus * asterisk , comma : semicolon (left paranthesis { left brace } right brace # number sign _ underscore = equal to

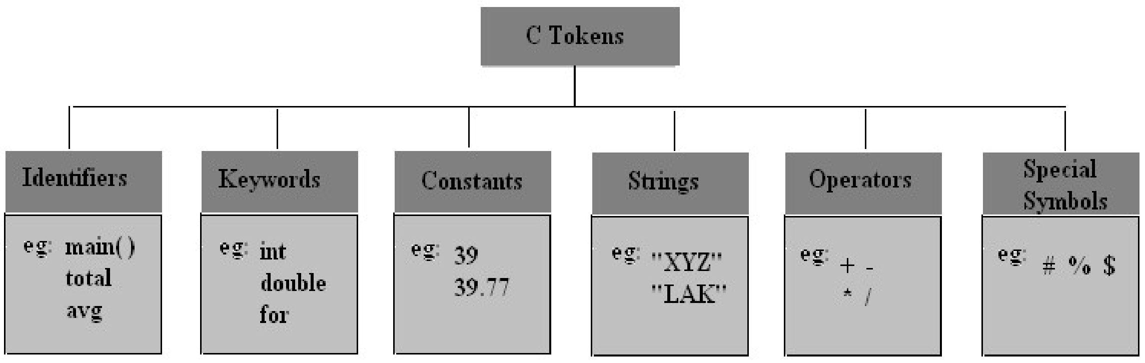
Execution Character Set:

These are employed at the time of execution. These are also called as 'escape sequences'.

Character	Escape Sequence	ASC-II Value	Result
Bell(Alert)	\a	007	Beep sound
Backspace	\b	008	Moves previous position
Horizontal tab	\t	009	Moves next horizontal tab
Vertical tab	\v	011	Moves next vertical tab
New line (line feed)	\n	010	Moves next line
Form feed	\f	012	Moves initial position of next page
Carriage return	\r	013	Moves beginning of the line
Quotation mark	\”	034	Present Double quotes
Apostrophe	\’	039	Present Apostrophe
Question Mark	\?	063	Present question mark
Back slash	\\	092	Present back slash
Null	\0	000	Null (‘\0’ is used to indicate the end of the string)

C TOKENS

The tokens are usually referred as individual text and punctuation in a passage of text.



IDENTIFIERS AND KEYWORDS

Identifiers: Identifiers are names given to various program elements, such as variables, functions and arrays etc.

Rules for naming an identifier:

- Identifiers consist of letters and digits.
- The first character must be a letter/character or may begin with underscore (_).

- Both upper / lower cases are permitted although uppercase character is not equivalent to corresponding lowercase character.
- The underscore '_' can also be used and is considered as a letter.
- An identifier can be of any length while most of the 'C' compiler recognizes only the first 8 characters.
- No space and special symbols are allowed between the identifier.
- The identifier cannot be a keyword.

Some valid Identifiers are

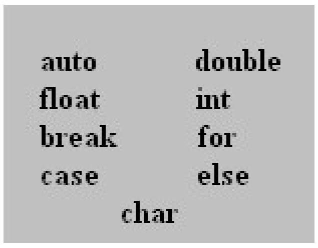
STDNAME, SUB, TOT_MARKS, _TEMP, Y2K.

Some invalid identifiers are

Return, $stay, 1RECORD, STD NAME.

Keyword: There are certain reserved words called keywords, that have standard and predefined meaning in 'C' language. Which cannot be changed.

Example:



auto	double
float	int
break	for
case	else
char	

All keywords must be written in lower case.

DATA TYPES:

Data type is the type of the data that a variable is going to hold.

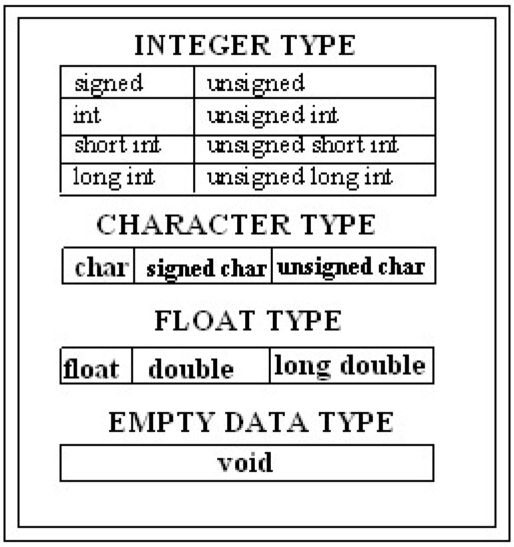
'C' supports the following 4 classes of data types

'C' Data Types			
Primary	User defined	Derived	Empty
char int float double	Typedef	Arrays Pointer Structures Union	void

The bytes occupied by each of the primary data types are

Data type	Description	Memory bytes	Range	Control string	Example
int	Integer quantity	2 bytes	-32,768 to +32,767	%d or %i	int a=39;
char	Single character	1 bytes	-128 to +128	%C	char s='n';
float	Floating pointing no's (i.e., a number containing a decimal point or an	4 bytes	3.4E-38 to 3.4E+38	%f Or %g	float f=29.77;
	exponent)				
double	Double precision Floating pointing no's (i.e., more significant Figures, and an exponent which may be larger than magnitude.	8 bytes	1.7E-308 To 1.7E+308	%1f	Double d=39977154896

The primary data types are divided into the following.


INTEGER TYPE

signed	unsigned
int	unsigned int
short int	unsigned short int
long int	unsigned long int

CHARACTER TYPE

char	signed char	unsigned char

FLOAT TYPE

float	double	long double

EMPTY DATA TYPE

void

a) **Integer type:** Integers are the numbers with the supported range. Usually the integers occupy 16 or 32 bits. The size of the integer depends upon the system.
E.g.: If we use 16 bit word length, the size can be limited up to -32768 to +32767.

b) **Character type:** Characters are generally stored in 8 bits and a single character can be defined as char data type.

c) **Float type:** The floating point numbers are generally stored in 32 bits with the 6 digits of precision. These numbers are defined by using the keywords float and double, whereas the double data type uses the 64 bits with the 14 digits of precision.

d) **Null data type**: The void is the null data type in 'C' language. This is generally specified with the function which has no arguments.

Type Qualifiers:

Sizes and Ranges of data types with type qualifiers

Type	Size(bytes)	Range	Control String
char or signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
int or signed int	2	-32,768 to 32,767	%d or %i
unsigned int	2	0 to 65535	%u
short int or signed short int	1	-128 to 127	%d or %i
unsigned short int	1	0 to 255	%d or %i
long int or signed long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
float	4	3.4E-38 to 3.4E+38	%f or %g
double	8	1.7E-308to 1.7E+308	%lf

long double	10	3.4E-4932 to 1.1E+4932	%lf

Variable:Variable is a data name used to store a data value. A variable is an identifier.

A variable may take different values at different times during the execution.

Rules for naming the variables:

a) A variable name can be any combination of 1 to 8 alphabets, digits or underscore.

b) The first character must be an alphabet or an underscore (_).

c) The length of the variable cannot exceed 31 characters long.

d) No commas or blank spaces are allowed within a variable name.

e) No special symbol, an underscore can be used in a variable name.

Variable Declaration:

The declaration tells the compiler what the variable name and type of the data that the variable will hold.

Syntax	Data_type v1,v2,v3,……,vn;
Description	Data type is the type of the data v1,v2,v3,……,vn; are the list of variables
Example	int code; char sex; float price; char name[10];

Here the variables, code is type of integer, sex is type of character, price is type of float and name[10] is defined an array of character.

Initializing variables

Initialization of variables can be done using the assignment operator (=). The variables can be initialized while declaration itself.

Syntax	Variable = constant; Or
	datatype variable =
	constant;

Description	i, f, c are the type of int, float, char data types.

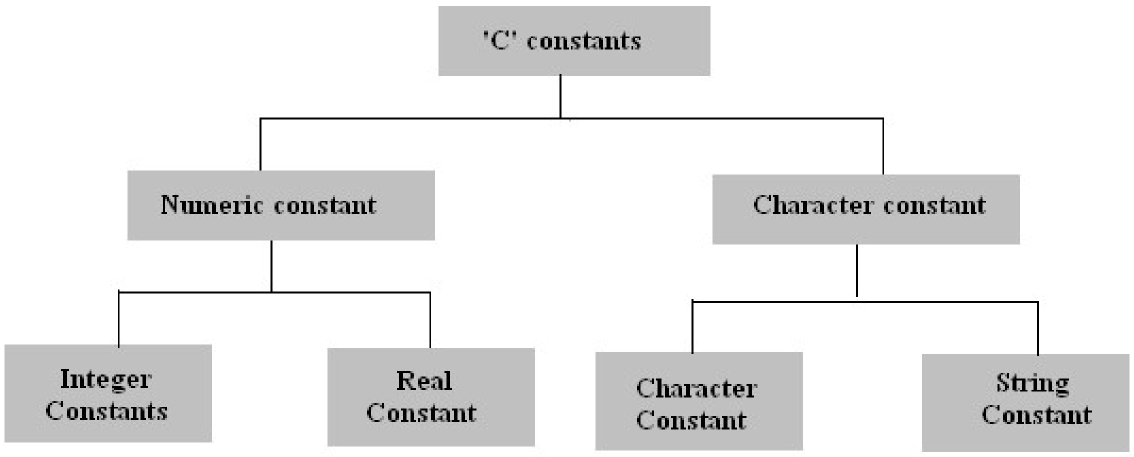
Example1	i=29;
	f=29.77; c='l';

Example 2	int i=29;
	float
	f=29.77
	char c='l'; int
	l=a=k=29;

The last example shows that the variables l,a,k are initialized with the contents of 29 itself for all variables.

CONSTANTS:

The item whose values cannot be changed during execution of program are called constants. 'C' constants can be classified as follows.



Numeric Constants:

a) **Integer constants**: A integer constant formed with the sequence of digits.

Examples:

Marks =90;

Per =75;

Discount=15;

Rules for defining an Integer constant:

a) It must have atleast one digit.
b) Decimal point is not allowed.
c) It can be either positive or negative.
d) If it is negative, the sign must be preceded, for positive the sign is not necessary.
e) No commas or blank spaces are allowed.
f) The allowable range for integer constants is -32,768 to +32,767.

b) Real constants: A real constant is made up of a sequence of numeric digits with presence of a decimal point.

Example:

Distance =126.0;

Height=5.6;

Speed=3e11;

Rules for defining real constant

a) A real constant must have one digit.
b) A real constant must have decimal digit.
c) It can be either positive or negative.
d) If it is negative, the sign is must or if it is positive, sign is not necessary.
e) No commas or blank spaces are allowed.

Character constants

a) Single Character Constants: The character constant contains a single character enclosed within a pair a single inverted commas both pointing to the left.

Example: ‘s’,’M’,’3’,’-‘

b) String constants: A string constant is a sequence of characters enclosed in double quotes, the characters may be letters, numbers, special characters and blank spaces etc. At the end of string ‘\0’ is automatically placed. Example :

"HI"

"Muni"

"39.77"

"50"

b) **Declaration of a variable as constant:** When the value of some of the variables may remain constantly during the execution of the program, in such a situation, this can be done by using the keyword const.

Syntax	const datatype variable =constant
Description	const is the keyword to declare constant variable is the name of the variable. datatype is the type of the data constant is the constant.
Example	const int dob=3977;

OPERATORS:

An operator is a symbol that specifies an operation to the performed on the operands.

Types of operators:

'C' provides a rich set of operators, depending upon their operation.

They are classified as

a) Arithmetic operators.
b) Relational operators.
c) Logical operators.
d) Assignment operators.
e) Short Hand operators.
f) Increment and decrement operators.
g) Conditional operators (Ternary operator).
h) Short Hand operators.
i) Bitwise operators.
j) Special operators.

a) Arithmetic operators

The following table shows the Arithmetic Operators and their meaning.

Operator	Meaning	Examples
+	Addition	2+9 = 11
-	Subtraction	9-2=7
*	Multiplication	2*9 =18
/	Division	9/3=3
%	Modulo division	9%2=1

All the above operators are called ‘Binary’ operators as they acts upon two operands at a time. Example:

```
main( )
{
        int a=10,b=3,c;
        c=a%b;
        printf(“%d”,c);
}
```

OUTPUT

```
   1
```

Here the remainder value only is assigned to variable ‘c’.

Program to find out all arithmetic operation on two given values

```
/* Program to find out all arithmetic operation on two given values*/
#include<stdio.h>
#include<conio.h>
void main( )
{
        /*Local
definitions*/           int
a,b,c,d;
int sum,sub,mul,rem;
float div;
/*Statements*/
clrscr();
        printf("Enter values of b,c,d:");
        scanf("%d%d%d",&b,&c,&d);su
        m=b+c; sub=b-c; mul=b*c;
        div=b/c;
        rem=b%d; a=b/c*d; printf("\n sum=%d, sub=%d, mul=%d,
        div=%f",sum,sub,mul,div); printf("\n Remainder of division of b&d
        is %d",rem); getch( );
}  /*main*/
```

OUTPUT:

```
         ENTER VALUES OF b,c,d : 2    6     45

       sum=8, sub=-4, mul=12, div=0.00000
        Remainder of division of b&d is 2
        a=0
```

b) Relational Operators

Relational operators are used to compare two or more operands. Operands may be variables, constants or expression.

The following table shows the Relational operators.

Operator	Meaning	Example	Return value
<	Is less than	2<9	1
<=	Is less than or equal to	2<=2	1
>	Is greater than	2>9	0
>=	Is greater than or equal to	3>=2	1

= =	Is equal to	2= = 3	0
! =	Is not equal to	2!=2	0

Syntax	AE1 relational operator AE2
Description	AE1 & AE2 are constants or an expression variables.
Example	9>2 yields True, which is equal to 1.

Suppose the ‘a’, ‘b’ and ‘c’ are integer variables, whose values are 1,2,3 respectively.

Expression	Interpretation	Value
a<b	True	1
(a+b)>=c	True	1
(b+c)>(a+5)	False	0
c!=3	False	0
b = = 2	True	1

c) Logical Operators:

Logical operators are used to combine the results of two or more conditions. ‘C’ has the following logical operators.

Operators	Meaning	Example	Return value
&&	Logical AND	(9>2) && (17>2)	1
| |	Logical OR	(9>2) | | (17 = = 7)	1
!	Logical NOT	29 ! =29	0

&& This operator is usually used in situation, where a set of statements are to be executed, if two or more expressions are true.

Example: (exp1) && (exp2)

| | This is used in situation, if either of them (one atleast) is true from two or more conditions, then the set statements are executed.

Example: (exp1) | | (exp2)

! This operator reverses the value of the expression it operators on i.e., It makes a true expression false ad false expression true.

Example : (!exp1)

Example : ‘i’ is an integer variable, whose value is 7.

‘f’ is a float variable, whose value is 5.5.

‘c’ is a character variable, that represents a character ‘w’.

d) Assignment operator

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

Syntax	Variable=expression (or) value;
Description	Variable is any valid ‘c’ variable, assigned value can be anything

Example	x=10;
	x=a+b; x=y;

e) Short Hand Operators

Some of the short hand operators and their meaning are given in the table.

Operator	Example	Meaning
+ =	x + = y	x =x +y
- =	x - = y	x =x -y
* =	x * = y	x =x *y
/ =	x / = y	x =x /y
% =	x % = y	x =x %y

f) Increment and Decrement operators(Unary operators)

‘C’ has two way useful operators not generally found in other languages, these are the increment (++) and decrement (--) operators. The ‘++’ adds one to the variables and ‘- - ‘subtracts one from the variable. These operators are called unary operators. Because they acts upon only one variable.

Operator	Meaning
++x	Pre increment
--x	Pre decrement
x++	Post increment
x--	Post decrement

f) Conditional operator (or) Ternary operator

Conditional operator itself checks the condition and executes the statement depending on the condition.

Syntax	Condition?exp1:exp2;
Description	The '?:' operator acts as a ternary operator, it first evaluate the condition, if it is true (non zero) then the 'exp1' is evaluated, if the condition is false (zero) then the 'exp2' is evaluated.

Example	main( ) { int
	a=5,b=3,big;
	big=a>b?a:b;
	    printf("Big is … %d",big);
	}
	
	Output
	  Big is ….5

In this case, it checks the condition 'a>b' if it is true, then the value of 'a' is stored in 'big' otherwise the value of 'b' is stored in 'big'.

g)Bitwise operators

Bitwise operators are used to manipulate the data at bit level. It operates on integers only. It may not be applied to float or real. The operators and its meaning are given below.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Shift left
>>	Shift right
~	One's complement

Bitwise AND (&) : This operator is represented as '&' and operates on two operands of integer type.

While operating upon these two operands they are compared on a bit by bit basis. (Both the operands must be of same type i.e. int).

Truth table for '&' is shown below

&	0	1
0	0	0
1	0	1

Eg : x = 7 = 0000
0111 y = 8 = 0000
1000 x & y = 0000 0000

Bitwise OR(|) : Similar to AND operator in all aspects, the truth table for Bitwise OR is shown below. But this operator gives if either of the operand bit is '1' then result is

'1' then result is '1' or both operands are 1's then also gives '1'.

|	0	1
0	0	1
1	1	1

Eg : x = 7 = 0000
0111 y = 8 =
0000 1000 x | y =
0000 1111

Bitwise Exclusive OR (^): Similar to AND operator in all aspects but integer gives if either of the operand bit is high(1) then it gives high(1) result. If both operand bits are same then it gives low(0) result.

^	0	1
0	0	1
1	1	1

Eg : x = 13 = 0000
1101 y = 8 =
0000 1000 x ^ y =
0000 0101

For all the above operators, in all possible combinations of bits as shown below.

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

h) **The Special operator**

Along with these operators, the C language supports some of the special operators given below.

Operators	Meaning

,	Comma operators
Sizeof	Size of operators
& *	Pointer operators
. →	Member selection operators

The pointer and member selection operators are discussed in the pointer concept.

(a) **comma operator(,)** : Usually the comma operator is used to separate the statement elements such as variables, constants or expression etc, and this operator is used to link the related expressions together, such expressions can be evaluated from left to right and the value of right most expression can be evaluated from left to right and the value of right most expressions together, such expressions can be evaluated from left to right and

the value of right most expressions is the value of combined expression.

Eg: val = (a=3,b=9,c=77, a+c);

Where, First assigns the value 3 to a

Then assigns the value 9 to b

Then assigns the value 77 to c

Finally assigns 80 to the val.

(b) **The sizeof( ) operator**: The sizeof() is a unary operator, that returns the length in bytes of the specified variable, and it is very useful to find the bytes occupied by the specified variable in the memery.

Syntax	sizeof(var);
Description	var is the variable for which to find the size.

Example	main( )
	{
	int a;
	printf(" Size of variable of a is …..%d", sizeof(a)); }
	OUTPUT
	Size of variable a is ……2

The sizeof( ) operator is a compile time operator.

(c) **Pointer operators:**

& : This symbol specifies the address of the variable.

* : This symbol specifies the value of the variable.

(d) **member selection operators:**

.→ : These symbols used to access the elements from a structure.

EXPRESSIONS

An expressions represents data item such as variables, constants and are interconnected with operators as per the syntax of the language. An expression is evaluated using assignment operator.

Syntax	Variable= expression;
Description	Any 'c' valid variable and expression
Example	x=a*b-c;

In the above statement the expression evaluated first from left to right. After the evaluation of the expression the final value is assigned to the variable from right to left.

Example:

y=(a/b)+c;   z=(a*b)-d;

User defined variables

a) **Type declaration:** 'C' language provides a features to declare a variable of the type of user defined type declaration. Which allows users to define an identifier that would represents an existing data type and this can later be used to declare variables.

Syntax	typedef data_type identifier;
Description	typedef is the user defined type declaration. data_type is the existing data type. Identifier is the identifier refers to the new name given to the data type.

Example	typedef int marks;
	marks m1,m2,m3;

Here marks is the type of int and this can be later used to declare variables.

b) **Enumerated data type:** The C language provides another user defined data type called enumerated data type.

Syntax	enum identifier { value1, value 2, ……., value n};
Description	identifier is the value defined enumerated data type. value1, value2,….., value n are the enumeration constants.
Example	enum day { mon, tue, wed, …., sun}; enum day w_st, w_end; w_st =mon; w_end=sun;

The identifier follows with the keyword enum is used to declare the variable, that can have only one value from the enumeration constants.

UNIT II

DATA INPUT AND OUTPUT FUNCTIONS

INPUT OUTPUT STATEMENTS

We know that the input, process, output are the three essential features of computer program. The program takes some input data, processes it and gives the output.

We have two methods for providing data to the program.

- Assigning the data to the variables in a program.
- By using the input/output statements.

In 'c' language, two types of input/output statements are available, and all inputs and output operations are carried out thought function calls. Several function are available for input/output operations in 'c'. These functions are collectively known as the standard I/O library.

i) Unformatted input/output statements
ii) Formatted input/output statements

Input and output functions

Unformatted Input/output Functions	
Input	output
getc( );	putc( );
getchar( );	putchar( );
gets( );	puts( );

Formatted input/output statements	
Input	output
scanf( );	printf( );
fscanf(	);
fprintf( );	

i) Single character Input-getchar() function

A single character can be given to the computer using 'c' input library function getchar ()

Syntax	Char variable=getchar();

Description	Char: data type; Variable : Any valid 'c' variable

Example	Char x;
	x=getchar ();

The getchar () function is written in standard i/o library. It read a single character from a standard input device. This function do not require any arguments, thought a pair of empty parentheses, must follow the statement getchar().

The first statement declared x as a character type variable. The second statement causes a single character to be entered from the standard input device and then assigned to variable x.

II) Single character output-put char() function

The put char () function is used to display one character at a time on the standard output device.

Syntax	putchar(character variable)
Description	Character variable is the valid 'c' variable of the type of char data type

Example:	Char x;
	putchar (x);

III) The gets and puts functions

The gets() function is used to read the string (string is a group of characters from the standard input device (keyboard).

Syntax	Gets(char type of array variable);
Description	Valid 'c' variable declared as one diamension char type
Example	gets(s);

The puts() function is used to display/write the string to the standard output device (Monitor).

Syntax	puts(char type of array variable);
Description	Valid 'c' variable declared as one diamension char type
Example	puts(s);

Example: **Program using gets() and puts() function**

```
/* Program using gets() and puts() function */
#include <stdio.h>
main()
{
char scientist [40]; /* Statements */
puts(“enter Name:”);
gets(scientist”); puts (scientist);
}
```

OUTPUTS:

```
Enter name : Abdul Kalam
Print the name : Abdul Kalam
```

IV)Formatted Inputs/Output statements

The following are the formatted Input/output statements.

Input	Output
scanf()	printf()
fscanf()	fprintf()

i) The scanf() function

Input data can be entered into the computer using standard input
‘c’ library function called scanf(). This function is used to enter any combination of input.

Then scanf() function is used to read information from the standard input device (keyboard), scanf function starts with a string argument and may contain additional arguments. Any additional arguments must be pointers.

Syntax	scanf(“control string”, & var2….&var n);
Description	‘the control string consists of character groups. Each character group must begin with a percentage sign ‘%’. The Character group consists percentage sign, followed by conversation character as specificied in table below, var1,var2,varn- are the arguments or variables in which the data is going to be accepted.

Example	int n;
	scanf(“%d”,&n);

CONTROL STRING:

Format code	Meaning
% c	Single character
% d or %i	integer
% s	Strings
% f	Float
% ld	Double

Each Variable name (argument) must be proceeded by an ampersand (&) The (&) symbol gives the meaning “address of “the variable.

Example:

```
/* Demonstrate of scanf() function */
#include<stdio.h>
void main()
{
int i,j;
clrscr();
printf(“enter the two number:”);
scanf(“%d%d”,&i,&j);
printf(“Two number is %d  \t%d\n”,i,j);
}
```

OUTPUT:

```
Enter the two number:50 78
Two number is 50 78
```

ii) printf () function

Output data or result of an operation can be displayed from the computer to a statement standard output device using then library function printf().

syntax	printf(“control string”,var1,var2..varn);

Description	Control string is any of the following. a) format code character b) execution character set of Escape sequences. c) Characters/string that will be displayed. Var1,var2,varn are the arguments or variables from which the data is going to output. Here the variables need not to be preceded with '&' sign.

Example	printf("Result is …%d",n);
	printf("%f".f); printf("%s",s);

Field width plays an important role in printf statements while producing report in a formatted manner. Field width indicated the least number of columns that will be located to the output.

Example: I=29, its field width is %4d.
So, the four columns are allocated for I and 2 blanks are added on leftside of value of i.
The output look like

/	/	2	9

b) **Writing:**
The integer data values can be displayed using the field width specification in printf() statement.

syntax	printf("%wd",var(s));
description	W is used to specify the minimum field width for the output. D is the control string specification.

The writing or displaying the real numbers may be displayed in decimal point notation using the field width specification.

Syntax	printf(“%w.p.f”, var(s));
Description	W is the number of position that are to be used for displaying the output. P specified the number of digits to be displaying after the decimal point.

Example										
The value of a = 39.7736	3	9	.	7	7	3	6			
printf(“%7.$f”,a);			3	9	.	7	7			
printf(“%7.2f”,a);	3	9	.	7	7					
printf(“%10.2e”,a);			3	.	9	7	e	+	0	1

Like, we can read or write the character and string also.

CONTROL STATEMENTS:

‘C’ language provides three general categories of control statements.

I. **Sequential structures**, in which instructions are executed in sequence.

Example: i=i+1;
J=j+1;

The above statements are executed one by one.

II) **Selection structure**, here the sequence of the instruction is determined try using the result of the condition.

```
Example:    if(x>y)
                  I=i+1;
       Else
                  J=j+1;
```

If the conditional true, then the statement i=i+1; will be executed otherwise it executes j=j+1;

III) **Iteration structure**, in which statements are repeatedly executed. These forms program loops.

```
Examples:   for (i=1; i<=5; i++)
                  {
                        I=i+1;
                  }
```

Where the statements i=i+1 will executed 5 times and value of I will change from 1,2,3,4 and 5.

'C' languages provides the following conditional (decision making statements)

If statements

If…else statements

Nested if…else statements

1. The if statement

The if statement is a decision making statements. It is used to control the flow of execution of the statement and also used to test technology whether the condition n is true or false.

```
Syntax:
  if (condition is
         true)
{
  True
statements;
}
```

If the condition is true , then the True statements are executed. The 'True statements' may be a single statements are group of statements. If the condition is false then the true statements are not executed, instead the program skip pass it. The conditional is gives by the relational operator like ==,!=, <=,>=, etc.

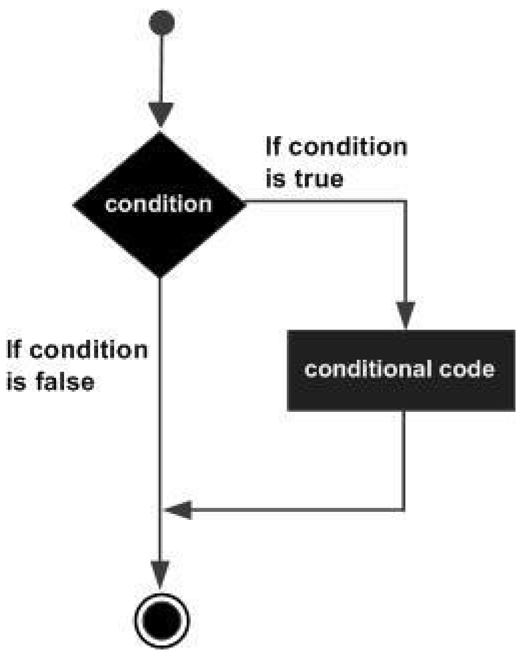
Example

```
#include <stdio.h>
 int main () {

  /* local variable definition */   int a = 10;

  /* check the boolean condition using if statement */

  if( a < 20 ) {
    /* if condition is true then print the following */     printf("a is less than 20\n" );
  }      printf("value of a is : %d\n", a);
    return 0;
}
```

Output:

```
a is less than 20; value of a is
 : 10
```

Flow Diagram



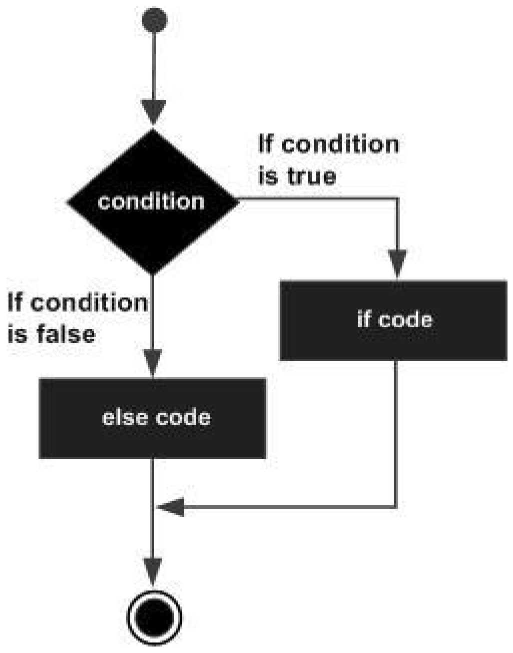
2. The if-else statement

It is basically two way decision making statement and always used in conjunction with condition.
It is used to execute some statements when the condition is true and execute some other statements, when the condition is false.

Syntax:

```
if (condition)
{
    True statements;
}
else
{
    False statements;
}
```

Flow Diagram



Example 1:

```
/* Program using if…else statement */
#include<stdio.h>
main()
printf("Enter the number..");
scanf("%d",&i);
if(i<10)
printf("\n The number is less than 10");
else
printf("\n The number is greater than 10");
)
```

Output:

```
Enter the number …11
The number is greater than 10
```

3. Nested if..else statement

When a series of if..else statements are occurred in a program, we can entire an entire if..else statement in another if..else statement called nesting, and the statement is called nested if.

```
Syntax:
if (condition 1)
{
if (condition 2)
{
    True Statement2;
}
Else{
    False Statement
2;
      }
}
Else
{
    False statement
1;
}
```

Example

```
#include <stdio.h>
 int main () {

  /* local variable definition */    int a =
100;    int b = 200;

  /* check the boolean condition */    if( a == 100 )
{

     /* if condition is true then check the following */       if( b == 200 ) {
       /* if condition is true then print the following */
```

```
       printf("Value of a is 100 and b is 200\n" );
     }
  }       printf("Exact value of a is : %d\n", a );
printf("Exact value of b is : %d\n", b );
    return 0;
}
```

```
OUTPUT:
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200
```

2. LOOPING AND BRANCHING

The loop is defined as the block of statements which are repeatedly executed for certain number of times.

Any looping statement, WOULD INCLUDE THE FOLLOWING STEPS:

I) initialization of variable
II) Test the control statement.
III) Executing the body of the loop depending on the condition.
IV) Updating the condition variable.

The following are the loop structure available in 'c'

While…
Do…while…
For…

THE WHILE LOOP

It is a repetitive control structure used to execute the statements within the body until the condition becomes false.

The while loop is an entry controlled loop statement, means the condition is evaluated first and if is true then the body of the loop is executed. After executing the body of the loop, the condition is once evaluated and it is True, the body is executed once again, the process of repeated execution of the body of the loop continues until the condition finally becomes false and the control is transferred out of the loop.

```
Syntax:
While(condition)
{ …..
   Body of the
       loop
….
}
```

The body of the loop may have one or more statements, the blocking with the braces are needed only of the body contain two or more statements.

Example 1:

```
/*program for addition of number using while loop*/
#include<stdio.h>
main()
{
int i=1,sum=0;
while (i<=10)
{
sum=sum+i;
i++;
}
printf("The sum of number up to 10 is %d",sum);
}
```

Output:

```
The sum of number up to 10 is 55
```

THE DO WHILE LOOP:

The while loop makes a test of condition before the loop is executed. Therefore, the body of the loop may not be executed at all, if the condition is not satisfied at first attempt. In some situations it may be necessary to exercise the body of the loop before the te4st condition is performed, such a situation the do.. while loop is useful.

It is also repetitive control structure and executes the body of the loop once respective of the condition, then it checks the condition and condition, then it checks the condition and conditional the execution until the condition becomes false.

```
Syntax:
do
{ …..
Body of the loop
….. }
while(condition);
```

Here , the statements with in the body of the loop is executed once, then it evaluated for the condition, if it is true, the it executed body until the condition becomes false.

Example:

```
/*program for addition of number using do..while loop */
#include<studio.h>
main()
{
int i=1, sum=0;
do
{
sum=sum+I;
i++;
}
while(i<=10);
printf("sum of the number up to 10 is ..%d", sum);
}
```

Output:

```
Sum of the numbers up to 10 is 55
```

Comparison between while and do while statements.

S.No.	While	Do. while
01	This is the top tested loop.	This is the bottom tested loop.
02	The conditional is first tested, If the condition is true then the Block is exercised until the Condition becomes false.	It executes the body once, after it checks the condition, if it is true the body is executed until the condition becomes false.

03	Loop will not be executed If the condition is false.	Loop is executed at least once even thought the condition is false.

THE FOR LOOP:

The for loop is another repetitive control structure, and is used to execute set of instructions repeatedly until the condition becomes false.

The assignment, incrementation or decementation and condition checking is done In for statement only, where as another control structures are not offered all these features In one statement.

Syntax:

```
for(initialize counter; test conditional ;increment/decrement
counter) {
….
Body of the loop;
….
}
```

For loop has three parts

i) initialize counter is used to ini9tialize counter variable.
ii) Test condition is used to test the condition.
iii) Increment/Decrements counter is used to increment or decrement counter variable.

Example:

```
/*program for addition of number using for loop*/
#include<stdio.h>
main()
{
int i,sum=0;
for(i=1;i<=10;i++)
{
sum=sum+I;
}
printf(“the addition of number up to 10 is %d”,sum);
}
```

Output:

```
The addition of the number 10 is 55.
```

NESTED OF FOR LOOP:

Example 1: Program to demonstrate nesting of for loop.

```
/*program using nesting of for loop */
#include<stdio.h>
main()
{
int i, j,; /*local definition */
for (i=1;i<=3;i++)
{
printf (“\n”);
for (j=1;j=3;j++)
printf(“%d \t “,j);
}
}
```

Output:

```
1       2       3
1       2       3
1       2       3
```

THE SWITCH STATEMENT:

It is a multi way decision statement, it tested the value of given variable or expression against a list of case values and when a match is found, a block of statements associated with the case is executed.

Syntax:

```
Switch(expression)
{
```

```
Case constant 1;
Block;
Break;
Case statement 2;
Block2;
Break;
.
..
..
Default:
Default block;
Break;
}
```

The expression following the keyword switch is any 'c' expression that must yield an integer value. It must be an integer be an constant like 1,2,3 or an expression that evaluate to an integer.

The keyword case is followed by an integer or a character constant, each constant in each case must be different from all the other.

First the integer expression the key word switch is evaluated. The value that gives is search against the constant values that the follow the case statements. When a match is found, the program executed the statements following the case. If no match found with any of the case statements, then the statements following the default are executed.

Example 1:

```
/* program using switch statement*/
#include<stdio.h>
#include<conio.h>
main()
{
int a=1;
switch(a)
{
case 1:
printf("I am in case 1 \n");
break;
case 2:
printf(" I am in case 2 \n");
break;
default:
printf(" I am in case default \n");
break;
}
}
```

Output:

```
I am in case 1
```

THE BREAK STATEMENT:

The break statement is used to terminate the loop. When the keyword break is used inside any ‘c’ loop, control automatically transferred to the first statement after the loop. A break is usually associated with an I f statement and switch.

When the break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop.

Syntax: break;

Example 1:

```
/* program using break statement */
#include<stdio.h>
#include<conio.h>
main()
{
int i;
for (i=1;i<=10;i++)
{
if (i==6)
break;
printf(“%d”,i);
}
}
```

Out put :

```
1 2 3 4 5
```

THE CONTINUE STATEMENT:

In some situation, we want to take control top the beginning of the loop, bypassing the statements inside the loop which have not yet been exercised for this [purpose the continue is used. When the statement continue is encountered inside any ‘c’ loop control automatically passes to the beginning of the loop.

Syntax: continue;

Like break statement the continue statements also associate with is statement.

Example: calculate the sum of the given positive number

```
/*program to calculate the sum of the given positive number */
#include<stdio.h>
main
{
int i,n,sum=0;
for (i=1;i<=5;i++)
{
printf("enter any number..\n");
scanf("%d",&n);
if(n<0)
continue;
else
sum=sum+n;
}
printf("sum is ..%d",sum);
}
```

Output:

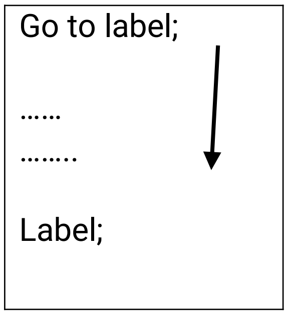
```
Enter the number…10 Enter
any number…5
Enter the number…1
```

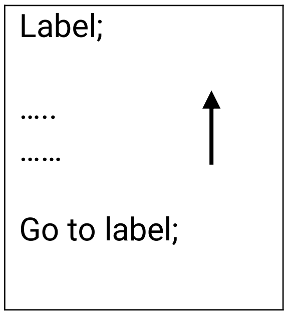
THE GOTO STATEMENT:

A goto statement can cause program control almost anywhere in the program unconditionally.

The go to statement requires a label to identify the place to move the execution. A label is a valid variable name and must be ended with colon(:)

Syntax:





Example:

```
/*program using go to statement */
#include<stdio.h>
#include<conio.h>
main()
{
int a,b;
printf("/n Enter the numbers");
Scanf("%d %d",&a,&b);
if (a==b)
goto equal;
else
{
printf("\n A and b are not equal");
exit(0);
}
equal:
printf("A and B are equal");
}  /*main*/
```

Output:

1. Enter the number 3,3

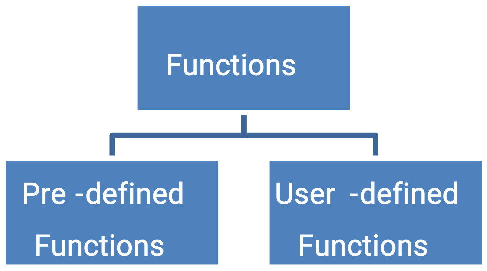
A and B are equal

Unit III

Functions:

A function is a set of instruction that are used to perform specified tasks which repeatedly occurs in the main program.

Functions are classified into two type as shown below.



Uses of C functions: C functions are used to avoid rewriting same logic/code again and again in a program.

- There is no limit in calling C functions to make use of same functionality wherever required.
- We can call functions any number of times in a program and from any place in a program.
- A large C program can easily be tracked when it is divided into functions.
- The core concept of C functions are, re-usability, dividing a big task into small pieces to achieve the functionality and to improve understandability of very large C programs.

Defining a Function:

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

```
return_type function_name( parameter list )
{   body of the
function
}
```

A function definition in C programming language consists of a *function header* and a*function body*. Here are all the parts of a function:

- **Return Type**: A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.

- **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument.
  The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body:** The function body contains a collection of statements that define what the function does.

FUNCTION DECLARATION, FUNCTION CALL AND FUNCTION DEFINITION:

There are 3 aspects in each C function. They are,

- **Function declaration or prototype** - This informs compiler about the function name, function parameters and return value's data type.
- **Function call** – This calls the actual function
- **Function definition** – This contains all the statements to be executed.

S.no	C function aspects	syntax
1	function definition	return_type function_name (arguments list) { Body of function; }
2	function call	function_name (arguments list);
3	function declaration	return_type function_name (argument list);

Example:

```
#include<stdio.h>
// function prototype, also called function
declaration float square ( float x );

// main function, program starts from
here int main( )
{
```

```
float m, n ;
printf ( "\nEnter some number for finding square \n");
scanf ( "%f", &m ) ;
// function call n
= square ( m ) ;
printf ( "\nSquare of the given number %f is %f",m,n );

}

float square ( float x )   // function definition
{ float p ; p
= x * x ;
return ( p ) ;
}
```

Output:

```
 Enter some number for finding square
   2
Square of the given number 2.000000 is 4.000000
```

USER DEFINED FUNCTION

The function defined by the user according to their requirements are called user defined functions. The users can modify the function according to their requirement. These are written by the programmer to perform a particular task, that is repeatedly used in main program. .

Advantages of user defined:

a) The length of the source program can be reduced by dividing it into the smaller functions.
b) By using functions it is very easy to locate and debug.
c) Functions avoid coding of repeated programming of the smaller instructions.

FUNCTION PROTOTYPES

The function are classified into the following types of depending o n a whether the arguments are present or not an s whether a value is returned or not. These are also called function prototypes.

The following are the prototypes:

i) Function with no arguments and no return
value. ii) Function with arguments and no return
values iii) Function with argument an with return

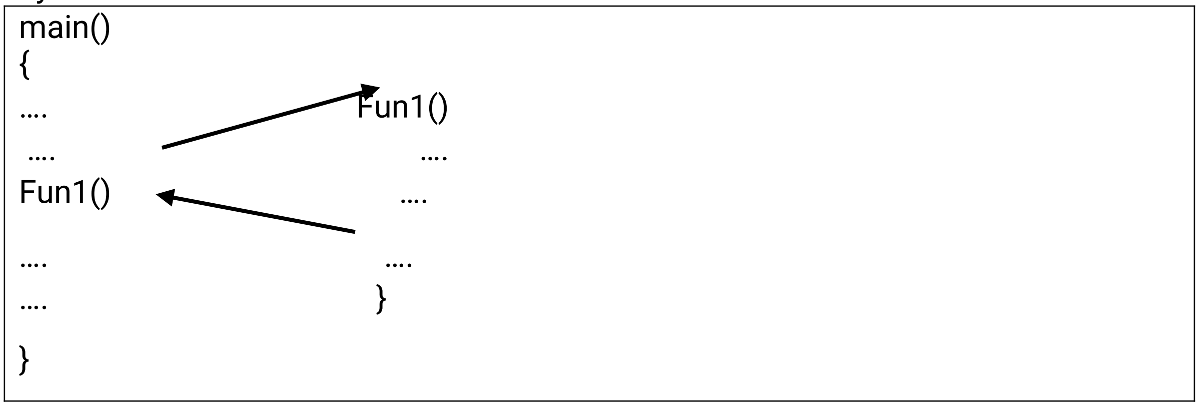
values.

iv) Function with no argument and with return value

Function with no argument and no return values:

In this argument, no data transfer takes place between the calling function and the called function, the called program does not receive any data from the calling program and does not send back any value to the calling program.

Syntax:



```
main()
{
....                    Fun1()
 ....                      ....
Fun1()                    ....

....                     ....
....                    }

}
```

The dotted lines indicates that, there is only transfer of control but no data transfer.

Example: Function call without parameters.

```
#include<stdio.
h> main()
{ message();
printf("main message");
}
message(
0
{ printf (printf("Function message
\n "); }
```

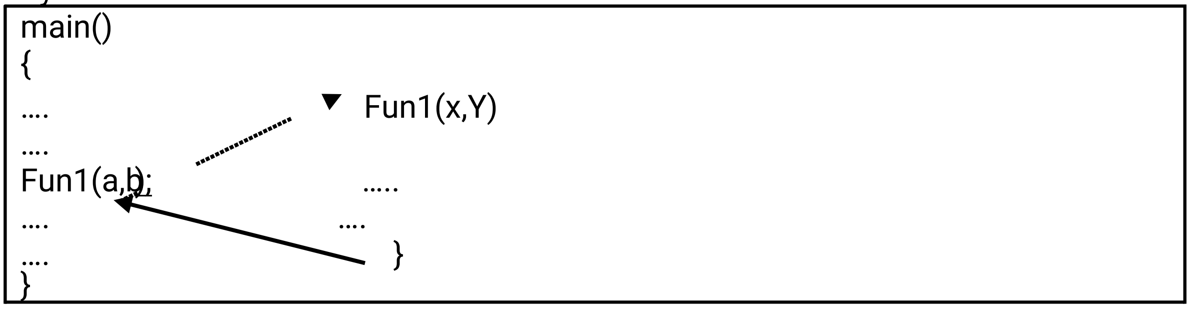
Output:

```
Function Message
 Main Message
```

Function with argument and no return values

In this prototype, data is transferred from calling function to called function I.e., The called program r3eceives some data from the calling program and does not send back any values to calling program.

Syntax:



```
main()
{                         Fun1(x,Y)
....
....
Fun1(a,b);                 .....
....                      ....
....                        }
}
```

The continuous line indicate data transfer and doted lines (2) indicates transfer of control **Example:**

```
#include<stdio.
h> main() {
void add(int,int); int a,b;
printf("Enter the
value");
scanf("%d",&a,&b);
add(a,b);
)
void add(int x, inty)
{
int z;
z=x+y; printf("Sum
is...%d",z);
}
```

Output:

```
Enter the value  10  20
 Sum is... 30
```

Function with arguments and with return values:

In this prototype, the data is transferred between the calling function an called function, i.e., The called program receives some data from the calling program and send back a value return to the calling program.

Syntax:

```
main()
{
```

```
....                                   fun1(int x,int y)
....                                  {
fun1(a,b);                              .....
....                                  ....
return();
....                                        }
}
```

Addition of two numbers:

```
#include<stdio.h>
main()
{
int add (int,int); int
a,b,c;
printf("Enter the number");
Scanf("%d",&a,&b);
c=add(a,b); printf("Result
is..%d",c);
}
injt add(int x,inty)
{
int z;
z=x+y; return(z);
}
```

Output:

```
Enter the number  10 20
Sum is...30
```

Function with no argument and with return value:

In this prototype , one can way data communication takes place i.e., the calling program can not pass any argument to the called program but, the called program may send some return value to the calling program. Syntax:

```
main()
{
....                           data type fun1(x,Y)
 ....                          .....

Fun1(a,b);                      .....
....                         .... Return (z)
....                                 }

}
```

Addition of two number:

```
#include<stdio.h>
main()
{ int
add();
c=add();
printf("Result in ..%d",c);
} int
add()
{ int
a,b,c;
printf("Enter two number:"); Scanf("%d
 %d",&a,&b);
c=a+b; return(c)
 ;
}
```

Output:

```
Enter the two numbers:
Result is….30
```

PARAMETER PASSING METHODS

There are two ways the parameter can be passed to the function, they are

i) Call By Value ii) Call By Reference

CALL BY VALUE:

This methods copies the values of actual parameters into the formal parameters of the function. Here, the change of the formal parameters cannot affect the parameters, because formal arguments are photocopy of actual arguments. Change made in the formal arguments are local to the block of the called functions. Once control returns back to the calling function the changes made disappear.

EXAMPLE:

Find the cube of the given value?

```
/*Program to find the cube of the given value*/
#include<stdio.h>
int cube(int x)
main()
{
/*local definition*/
```

```
int n=5;
/*Statements*/
printf("Cube4 of %d is .. %d", m,cube(n));
}
int cube(int);
{
x=x*x*x;
{
return(x);
} /*cube*/
```

OUTPUT:

```
        Cube of 5 is.. 125
```

CALL BY REFERENCE:

Call by reference is Another way of passing parameters to the function. Here , The address of argument are copied into the parameters inside the function, the address is used to the access the actual arguments used in call. Hence, changes made in the arguments are permanent.

Here pointers are passed to function, just like any other arguments. Hence we need not declared the parameters as pointers type.

EXAMPLE: Interchange two values

```
/*program to interchange two values*/
#include<stdio.h>
void interchange (int *a, int*b);
void main()
{ int
i=5,j=10;
clrscr();
printf("I and j values before interchange : %d %d \n ",I,j);
interchange(&i,&j);
printf("i and j values after interchange : %d %d \n ",I,j); getch();
}
void interchange (int *a, int *b)
{ int t;
t=*a;
*a=*b;
*b=t;
}
```

OUTPUT:

```
        i and j values before interchange : 5
10      i and j values after interchange : 10 5
```

RECURSION

Recursion is the process of repeating items in a self-similar way. Same applies in programming languages as well where if a programming allows you to call a function inside the same function that is called recursive call of the function as follows.

```
void recursion()
{
  recursion(); /* function calls itself */
}

int main() {    recursion();
}
```

The C programming language supports recursion, i.e., a function to call itself. But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go in infinite loop.

Recursive function are very useful to solve many mathematical problems like to calculate factorial of a number, generating Fibonacci series, etc.

```
#include<stdio.h>
      int fact(int);
        int main()
        {   int
        num,f;
          printf("\nEnter a number: ");
        scanf("%d",&num);
        f=fact(num);
          printf("\nFactorial of %d is: %d",num,f);
        return 0;
        }

        int fact(int
        n){    if(n==1)
        return 1;    else
             return(n*fact(n-1));
         }
```

STORAGE CLASS

A storage class defines the scope (visibility) and life time of variables and/or functions within a C Program.

There are following storage classes which can be used in a C Program

- auto
- register static
- extern **auto - Storage Class auto** is the

  default storage class for all local

  variables.

```
    {
int Count;
auto int Month;
    }
```

The example above defines two variables with the same storage class. auto can only be used within functions, i.e. local variables.

register - Storage Class

register is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and cant have the unary '&' operator applied to it (as it does not have a memory location).

```
    {
register int  Miles;
    }
```

Register should only be used for variables that require quick access - such as counters. It should also be noted that defining 'register' goes not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register - depending on hardware and implimentation restrictions.

static - Storage Class

static is the default storage class for global variables. The two variables below (count and road) both have a static storage class.

```
      static int Count;
  int Road;
```

```
    {
       printf("%d\n", Road);
    }
```

static variables can be 'seen' within all functions in this source file. At link time, the static variables defined here will not be seen by the object modules that are brought in.

static can also be defined within a function. If this is done the variable is initalised at run time but is not reinitalized when the function is called. This inside a function static variable retains its value during vairous calls.

```
  void func(void);


  static count=10; /* Global variable - static is the default */


  main()
{
   while (count--)
   {
      func();
   }


  }
  void func( void )

  {

    static i = 5;
i++;

    printf("i is %d and count is %d\n", i, count);

  }
```

This will produce following result

```
  i is 6 and count is
9    i is 7 and count
is 8    i is 8 and
count is 7    i is 9
and count is 6    i is
10 and count is 5    i
is 11 and count is 4
i is 12 and count is
3    i is 13 and count
is 2    i is 14 and
count is 1    i is 15
and count is 0
```

extern - Storage Class

extern is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern' the variable cannot be initalized as all it does is point the variable name at a storage location that has been previously defined. When you have multiple files and you define a global variable or function which will be used in other files also, then *extern* will be used in another file to give reference of defined variable or function. Just for understanding *extern* is used to decalre a global variable or function in another files. File 1: main.c

```
int count=5;

main()   {     write_extern();   }
```

File 2: write.c

```
  void write_extern(void);

extern int count;

void write_extern(void)
{
  printf("count is %i\n", count);
}
```

VARIABLE:

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable can not be accessed. There are three places where variables can be declared in C programming language:

1. Inside a function or a block which is called **local** variables,
2. Outside of all functions which is called **global** variables.
3. In the definition of function parameters which is called **formal** parameters.

Let us explain what are **local** and **global** variables and **formal** parameters.

Local Variables

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. Following is the example using local variables. Here all the variables a, b and c are local to main() function.

```
#include <stdio.h>

int main ()
{
  /* local variable declaration
*/   int a, b;   int c;

  /* actual initialization
*/   a = 10;   b = 20;   c
= a + b;


  printf ("value of a = %d, b = %d and c = %d\n", a, b, c);



  return 0; }
```

Global Variables

Global variables are defined outside of a function, usually on top of the program. The global variables will hold their value throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. Following is the example using global and local variables:

.

```
#include <stdio.h>


/* global variable declaration */
int g;


int main ()
{
  /* local variable declaration */
int a, b;



  /* actual initialization
*/   a = 10;   b = 20;   g
= a + b;


  printf ("value of a = %d, b = %d and g = %d\n", a, b, g);


  return 0;
}
```

A program can have same name for local and global variables but value of local variable inside a function will take preference. Following is an example:

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

```
/* global variable declaration */
int g = 20;

int main ()
{
  /* local variable declaration */   int g = 10;

  printf ("value of g = %d\n",  g);

  return 0; }
```

When the above code is compiled and executed, it produces the following result:

```
value of g = 10
```

Formal Parameters

Function parameters, formal parameters, are treated as local variables with-in that function and they will take preference over the global variables. Following is an example:

```
#include <stdio.h>

/* global variable declaration */
int a = 20;

int main ()
{
  /* local variable declaration in main
function */   int a = 10;   int b = 20;   int c = 0;


  printf ("value of a in main() = %d\n",  a);
```

```
    c = sum( a, b);
  printf ("value of c in main() = %d\n",  c);

  return 0;
}


/* function to add two integers
*/ int sum(int a, int b)
{    printf ("value of a in sum() = %d\n",
a);     printf ("value of b in sum() =
 %d\n",  b);


    return a + b;
}
```

When the above code is compiled and executed, it produces the following result:

```
value of a in main() = 10
value of a in sum() = 10

value of b in sum() = 20
value of c in main() = 30
```

Initializing Local and Global Variables

When a local variable is defined, it is not initialized by the system, you must initialize it yourself. Global variables are initialized automatically by the system when you define them as follows:

Data Type	Initial Default Value
int	0
char	'\0'
float	0
double	0

pointer	NULL

It is a good programming practice to initialize variables properly otherwise, our program may produce unexpected results because uninitialized variables will take some garbage value already available at its memory location.

.

UNIT IV

ARRAYS

An array is a collection of similar data item that are stored under a common name. a value is an array is identified by index or subscript enclosed in square brackets with array name.

Arrays can be classified into

- One-dimensional arrays
- Two-dimensional arrays
- Multi-dimensional arrays

ARRAYS DECLARATION

Arrays are declared in the same manner as an ordinary variables except that each array name must have the size of the array i.e., number of elements accommodated in that array.

Like variables, the array must be declared before they are used.

Syntax	Data_type [size or subscript of the array]
description	Data_type specifies the type of the data that will be contained in the array. Array_variable specifies the name of the array. Size or subscript specifies the maximum number of elements that the array can hold.

EXAMPLE:

int a[5];

Where, ‘a’ is the name of the array with 5 subscription of integer data type and the computer reserves five storage locations as shown below.

a[0]
a[1]
a[2]
a[3]
a[4]

ARRAYS INITIALISATION

The values can be initialized to an array. When they are declared like ordinary variables, otherwise they hold q garbage value.

Syntax	Data_type array_name[size]=[list of value];

Description	The list of values must be separated by commas.
Example	int mark[3]={70,80,90};

EXAMPLE:

```
#include<stdio.h>]
#include<conio.h>
void main()
{ int a[5],I;
clrscr();
for(i=0;i<5;i++)
{
printf("Enter the %d number:",i+1);
scanf("%d",&a[i]);
}
for(i=0;i<5;i++)
{ if(a[i]%2==0) printf("%d Number
is even.\n ",a[i]); else printf("%d
Number is odd. \n ",a[i]);
} getch();
}
```

INPUT:

```
Enter the number : 17                17  number is odd
Enter the number:   38               38  number is even
```

TWO DIMENSIONAL ARRAYS:

Two dimensional array are used in situation where a table of values need to be stored in an array.

These can be defined in the same fashion as in one dimensional array, except a separate pair of square brackets is required for each subscript.

Two pair of square brackets required for two dimensional array and three pairs required for three dimensional arrays and so on.

syntax	Data_type array_name[row size][column size];

description	Data_type specifies the type of the data that will be contained in the array. Array_name specifies the name of the array. [row size] specvifies the size of the row [column size] specifies the size of the column.
example	int a[3][3];

Two dimensional arrays are stored in a row-column matrix, where the left index indicated the row and the right indicates the column.

INITIALISING A TWO-DIMENSIONAL ARRAY

Like one dimensional array, the values can be initialized to the two dimensional arrays at the time of declaration.

syntax	Data_type array_name[row size][column size]={list of values};
description	{list of variables} specifies the list of elements.

example	int stud [4][2]=
	{
	{6680,80},
	{6681,95},
	{6682,82},
	{6683,85},
	};
	OR
	int stud [4] [2] ={6680.80.6682.82.6683.85};
	OR
	int stud[] [2]= {6680,80,6681,95,6682,82,6683,85};

Above all are perfectly acceptable Remember that while initializing an array it is necessary to remain the second dimension, where as the first dimension is optional. So, the following will never work.

```
int arr[2][]={1,2,3,4,5,6};
int arr[][]={1,2,3,4,5,6};
```

We must mention the column size then only the complier knows where the first row ends. The row size is optional is we initialized the array in the declaration part itself.

PASSING ARRAYS TO FUNCTIONS:

An entire array can be transferred to a function as a parameter. To transfer an array to a function, the array name is enough without subscripts as an actual parameters within the function call.

The corresponding formal parameters are written in same fashion , and must be declared as an array in formal parameters declarations.

Syntax:

```
main()                                        fun1(datatype x,data type ar[])
{                                              { int
a[10],n;                                     int x,ar[10];
.....                                          .....
fun1(n,a);                                       .....
 .....                                                 }
 }
```

EXAMPLE: Add and display the result of 5 number given from the input using arrays to function concept.

```
#include<stdio.h>
#include<conio.h>
void add (int,int[]);
main()
{
int a[5],i,n=5;
printf("Enter 5 values"); for
i=1;i<=5;i++)
scanf("%d",&a[i]);
add (n,a);
}
add(int x,int ar[])
{
for(i=1;i<=5;i++)
printf("%d",ar[i]);
}
```

OUTPUT

```
Enter 5 values
 10 20 30 40 50
 Sum is....150
```

ARRAYS OF CHARACTERS (STRINGS)

A string is a collection of characters. A string constant is a one dimensional array of characters terminated by a null('\0') character.

Example:

```
Char name[]={'L','A','K','\0'};
```

Example 2: program to print the character string from the array.

```
#include<stdio.h>
main()
{
char name []="LAK";     printf("%s",name);
}
```

OUTPUT:

```
LAK
```

STRINGS:

Example

```
Char name[ ]={'B','A','B','U',\0'};
```

Memory map of string.

String

B	A	B	U
5001	5002	5003	5004

INITIALIZATION OF STRING

```
Char name[ ]="BABU";
```

READING AND WRITING STRING

The "%s" control string can be used in scanf() statements to read a string from the terminal and the same may be used to write string to the terminal in printf() statements.

Example:

```
char name[10];
Scanf("%s",name);
```

```
Printf("%s",name);
```

There is no address (&) operator used in scanf() statement.

STRING STANDARD FUNCTION:

FUNCTION	PURPOSE
Strlen()	Used to find length of the string
Strcpy()	Used to copy one string to another
Strcat()	Used to combine two string
Strcmp()	Used to compare characters of two string (difference between small and capital letters)
Strrev()	Used to reverse a string

THE strlen() function.

The function is used to count and return the number of characters present in a string.

Syntax	Var=strlen(string);
Description	Var is the integer variable, which accepts the length of the string String is the string constant or constant variable, in which the length is going to be found . The counting ends with first null(\0) character.

EXAMPLE: PROGRAM USING STRLEN() FUNCTION.

```
#include<stdio.h>
#include<conio.h>
main()
{
char name[]="MUNI";
int len 1,len2; len1=strlen(name);
len2=strlen("LAK"); printf("\n string length of
 %s is %d", name, len1); printf("\n string length
of %s is %d", "LAK",len2); }
```

OUTPUT:

```
String length of MUNI  is 4
 String length of LAK is 3
```

THE strcpy() function

This function is used to copy the contents o one string to another and is almost works like string assignment operator.

syntax	Strcpy(string 1, string2);
Description	String 1 is the destination string String 2 is the source string

i.e., The contents of string 2 is assigned to the contents of string1. where string2 may be character array variable or string constant.

EXAMPLE 1

```
Char str1[]="MUNI";
Char str2[]="LAK";
Strcpy(str1,str2);
```

Where the content of str2 are copied into the str1 and the content pf str1 is replaced with new one.

The strcat() function

The strcat function is used to concatenate or combine; two strings together form a new concatenated string.

syntax	Strcat(str1,str2);
description	String1 and strring2 are character type arrays or string constants.

When the strcat() function is executed, string 2 is combined with struing 1 and it removes the null character (\0) of string 1 and places string 2 from there.

Example:

```
        Strcat("MUNI",LAK");
```

Yields MUNILAK.

The STRCMP() FUNCTION

This is the function which compared two string to find out whether they are some or different. The two strings are composed character by character until the end of one of the string is reached. If the two string are identical; strcmp() returns a value zero. If they are not equal. it returns the numeric difference between the first non-matching characters.

Syntax:	strcmp(str1,str2);
Description	string 1 and string 2 are character type arrays or string constants.

EXAMPLE:

```
#include<stdio.h
>
#include<conio.
h> main() {
char name[]="Kalai";
char name[]= "malai";
int I,j,k;
i=strcmp(name,"kalai");
j=strcmp(name1,name);
k=strcmp(name,"kalai mani");
}
```

Output:

```
0       1       6
```

THE STRREV( ) FUNCTION

The strrev function ius used to reverse a string. This function takes only one argument and return one argument. The general form a strrev( ) function function is

Syntax	strrev(string);
Description	string are characters type arrays or string constants.

EXAMPLE:

```
#include<stdio.h>
main( ); char y[30]; printf("enter the
string:"); gets(y); printf("The string
reversed is : %s",strrev(y)); }
```

Output:

```
Enter the string  :  book
The string reversed is : koob
```

STRUCTURES:

A structures is a collection of different data types under a common name.

Defining a structure.

A structure can be defined with the keyword struct following the name and opening brace with data elements of different type then closing brace with semicolon, as shown below.

Syntax

```
struct structure_name
 {
Structure member1;
Structure member2;
```

```
                                        ……
                                       Structure member n;
                                       };
```
E.g.

```
       Struct simple

{            int no
float m;
         };
```
Declaration of Structure:

```
               Struct structure name v1,v2….vn;
```
E.g.:

```
               struct simple s;
```

ACCESSING STRUCTURE ELEMENTS

After declaring the structure type, variables and members, the member of the structures can be accessed by using the structure variable along with the dot(.) operator.

For accessing the structure member from the above example

```
       s.no    s.m
```

Example for structure:

E.g.
```
       struct simple

{              int no;
float m;
         }s;
main()
         {
           scanf("%d %f",&s.no,&s.m);
            printf("%d%f",s.no,s.m);
         }
```

INITIALIZATION OF STRUCTURE:

Like normal variables, the structure variable can also be initialized, but this initialization can be made at the compile time.

Eg

```
        struct simple

{                 int no;
float m;
          }s={10,20.6};
main()
          {
                 printf("%d %f",s.no,s.m);
           }
```

Structure and functions (Passing structure to function):

The structure can be transferred from one function to another function by passing a copy of the entire structure to the called function. Since the function is working on a copy of the structure any changes to structure members within the function are not reflected in the calling function.

Syntax

Actual Parameters => function name(structure name);

Formal Paramters=> data type function name(struct structure name variable name1)

Eg:

```
   #include<stdio.h>
   struct simple
    {            int
no;
float m;
    }x1;
main()
   {
      scanf("%d%f",&x1.no,x1.m);
      fun(simple);
   }
   fun(struct simple x1)
   {
     printf("%d %f",x1.no,x1.m);
    }
```

UNION

Union is a derived data type and it is declared like structure. The difference between union and structure is in term of storage in structure each member has its own storage location whereas all the members of union use the same location, although a union may contain many members of different types. When we use union the compiler allocated a piece of storage that is large enough to hold. Like structure, union is also declared by using keyword **union.**

Syntax:

```
        union union name
         {
         union member1;
union member2;
         .....
         union member n;
         }union variable;
```

E.g.:

```
E.g.:       union
simple

{             int
no;
foat m;        }s;
main()
    {
         s.no=78;
printf("%d",s.no);
s.m=89.78;
         printf("%f",s.m);
    }
```

COMPARISION OF STRUCTURE AND UNION

S. NO	STRUCTURE	UNIONS
1	A structure is as collection of different data types under a common name.	A Union is as collection of different data types under a common name.

2	Syntax
	struct structure_name
	{
	
	Structure member1;
		Syntax:
		union union name
		{
		union member1;
		union member2;
		..... union

	Structure member2;
	……
	
	Structure member n;
	}v1,v2..vn;
		member n;
		}v1;

3.	Every member has its own memory space	All the members use the same memory space to store the value
4.	Keyword struct is used.	Keyword union is used.
5.	It is a user defined data type	It is a user defined data type
6.	It may initialized with all its members.	Only its first members may be initialized.
7.	More storage space is required.	Less storage space is required

Unit V

Pointer

C **Pointer** is a variable that stores/points the address of another variable. C Pointer is used to allocate memory dynamically i.e. at run time. The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc

Where, * is used to denote that “p” is pointer variable and not a normal variable.

Syntax

```
datatype *var-name;
```

- The *unary* or *monadic* operator & gives the ``**address of a variable**''.
- The *indirection* or **dereference** operator * gives the ``**contents of an object *pointed to* by a pointer**''.

KEY POINTS TO REMEMBER ABOUT POINTERS IN C:

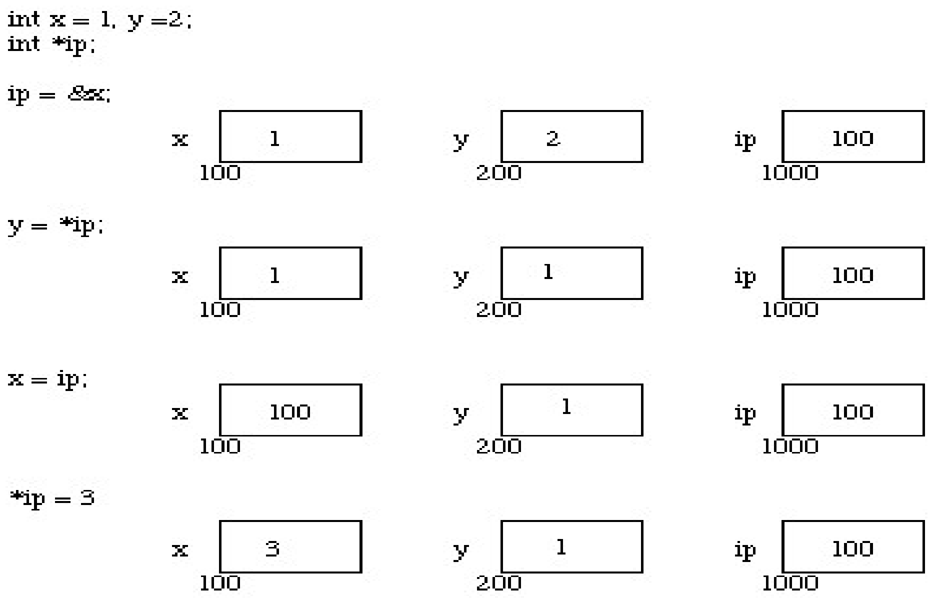
- Normal variable stores the value whereas pointer variable stores the address of the variable
- The content of the C pointer always be a whole number i.e. address   Always C pointer is initialized to null, i.e. int *p = null.
- The value of null pointer is 0
- & symbol is used to get the address of the variable
- symbol is used to get the value of the variable that the pointer is pointing to   If pointer is assigned to NULL, it means it is pointing to nothing.
- Two pointers can be subtracted to know how many elements are available between these two pointers
- But pointer addition, multiplication, division are not allowed
- The size of any pointer is 2 byte(for 16 bit compiler)

Declaration of Pointer

```
data_type* pointer_variable_name;


int* p;
```

In the above declaration :

1. data-type : It specifies the type of pointer. It can be int,char, float etc. This type specifies the type of variable whose address this pointer can store.
2. pointer-name : It can be any name specified by the user. Professionally, there are some coding styles which every code follows. The pointer names commonly start with ‘p’ or end with ‘ptr’



Example

```
#include <stdio.h>
 int main () {
   int  var = 20;   /* actual variable declaration */    int  *ip;        /*
pointer variable declaration */
   ip = &var;  /* store address of var in pointer variable*/
```

```
  printf("Address of var variable: %x\n", &var  );

  /* address stored in pointer variable */    printf("Address stored in
ip variable: %x\n", ip );
  /* access the value using the pointer */    printf("Value of *ip variable:
 %d\n", *ip );
   return 0; }
```

Output:

```
Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20
```

Passing pointers to function

When a pointer is passed as an argument to a function, address of the memory location is passed instead of the value.

This is because, pointer stores the location of the memory, and not the value.

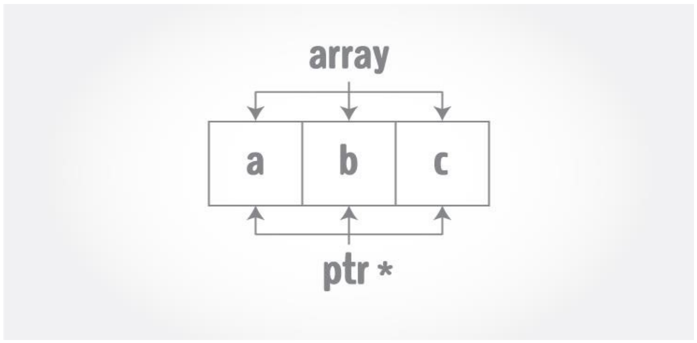
Example:

```
/* C Program to swap two numbers using pointers and function. */
#include <stdio.h> void swap(int
*n1, int *n2);
 int main()
{
    int num1 = 5, num2 = 10;

    // address of num1 and num2 is passed to the swap function     swap( &num1,
&num2);
    printf("Number1 = %d\n", num1);
    printf("Number2 = %d", num2);     return 0;
}
void swap(int * n1, int * n2)
{
    // pointer n1 and n2 points to the address of num1 and num2 respectively
int temp;     temp = *n1;     *n1 = *n2;
    *n2 = temp;
}
```

```
output

Number1 = 10

Number2 = 5
```

Pointer and Arrays

Arrays are closely related to pointers in C programming but the important difference between them is that, a pointer variable takes different addresses as value whereas, in case of array it is fixed



Example:

```
#include <stdio.h> int
main()
{
  char charArr[4];    int i;

  for(i = 0; i < 4; ++i)
  {
    printf("Address of charArr[%d] = %u\n", i, &charArr[i]);
  }
  return 0;
}
```

Output

```
Address of charArr[0] = 28ff44

Address of charArr[1] = 28ff45

Address of charArr[2] = 28ff46

Address of charArr[3] = 28ff47
```

Relation between Arrays and Pointers

Consider an array:

```
int arr[4];
```

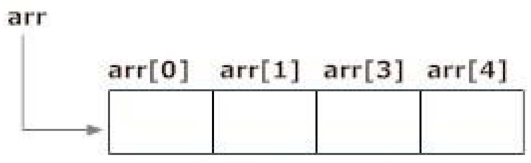


Figure: Array as Pointer

In C programming, name of the array always points to address of the first element of an array.

In the above example, **arr** and **&arr[0]** points to the address of the first element.

```
&arr[0] is equivalent to arr
```

Since, the addresses of both are the same, the values of arr and &arr[0] are also the same.

```
arr[0] is equivalent to *arr (value of an address of the pointer)
```

Example: Access Array Elements Using Pointers

```
#include <stdio.h>

int main()
{
  int data[5], i;
  printf("Enter elements: ");
   for(i = 0; i < 5; ++i)
scanf("%d", data + i);

  printf("You entered: \n");    for(i = 0; i < 5;
++i)
    printf("%d\n", *(data + i));

  return 0;
}
```

Output

```
Enter elements: 1
2
3
5
4
You entered:
1
2
3
5
4
```

File Handling in C Language

A **file** represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data. It is a ready made structure.

In C language, we use a structure **pointer of file type** to declare a file.

```
FILE  *fp ;
```

C provides a number of functions that helps to perform basic file operations. Following are the functions,

Function	**description**
fopen()	create a new file or open a existing file
fclose()	closes a file

getc()	reads a character from a file
putc()	writes a character to a file
fscanf()	reads a set of data from a file
fprintf()	writes a set of data to a file
getw()	reads a integer from a file
putw()	writes a integer to a file
fseek()	set the position to desire point
ftell()	gives current position in the file
rewind()	set the position to the begining point

Opening a File or Creating a File

The fopen() function is used to create a new file or to open an existing file.

General Syntax :

```
*fp = FILE    *fopen ( const char    *filename  , const char    *mode) ;
```

Here **filename** is the name of the file to be opened and **mode** specifies the purpose of opening the file. Mode can be of following types,

***fp** is the FILE pointer (FILE *fp), which will hold the reference to the opened(or created) file.

mode	description
r	opens a text file in reading mode
w	opens or create a text file in writing mode.
a	opens a text file in append mode
r+	opens a text file in both reading and writing mode
w+	opens a text file in both reading and writing mode

a+	opens a text file in both reading and writing mode
rb	opens a binary file in reading mode
wb	opens or create a binary file in writing mode
ab	opens a binary file in append mode
rb+	opens a binary file in both reading and writing mode
wb+	opens a binary file in both reading and writing mode
ab+	opens a binary file in both reading and writing mode

Closing a File

The fclose() function is used to close an already opened file.

General Syntax :

```
int fclose( FILE *fp );
```

Here fclose() function closes the file and returns **zero** on success, or **EOF** if there is an error in closing the file. This **EOF** is a constant defined in the header file **stdio.h**.

Input/Output operation on File

In the above table we have discussed about various file I/O functions to perform reading and writing on getc() and putc() are simplest functions used to read and write individual characters to a file.

file.

```
#include<stdio.h>

#include<conio.h> main()

{

 FILE *fp;  char ch;  fp = fopen("one.txt",

"w");  printf("Enter data");  while( (ch =

getchar()) != EOF) {     putc(ch,fp);

 }  fclose(fp);  fp = fopen("one.txt",

"r");

   while( (ch = getc(fp)! = EOF)

printf("%c",ch);

      fclose(fp);

}
```

Reading and Writing from File using fprintf() and fscanf()

```
#include<stdio.h>
#include<conio.h> struct emp
{   char name[10];
int age;
}; void main()
{   struct emp e;   FILE *p,*q;   p =
fopen("one.txt", "a");   q = fopen("one.txt", "r");
printf("Enter Name and Age");   scanf("%s %d",
e.name, &e.age);   fprintf(p,"%s %d", e.name,
e.age);   fclose(p);   do   {     fscanf(q,"%s %d",
e.name, e.age);     printf("%s %d", e.name,
e.age);
  }   while( !feof(q) );
getch();
}
```

1. Write a C program to read name and marks of n number of students from user and store them in a file.

```
#include <stdio.h> int
main()
{
  char name[50];   int
marks, i, num;
```

```
  printf("Enter number of students: ");    scanf("%d", &num);


  FILE *fptr;
  fptr = (fopen("C:\\student.txt", "w"));    if(fptr == NULL)
  {
     printf("Error!");       exit(1);
  }
  for(i = 0; i < num; ++i)
  {
    printf("For student%d\nEnter name: ", i+1);       scanf("%s", name);


    printf("Enter marks: ");       scanf("%d", &marks);


    fprintf(fptr,"\nName: %s \nMarks=%d \n", name, marks);
  }
  fclose(fptr);    return 0;
}
```