Page No.:
Date:
YOUVA

:- Introduction to C :-

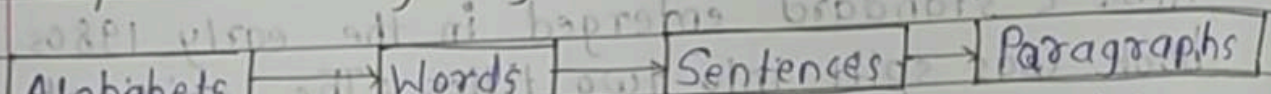
- C is a programming language developed at AT & T's Bell laboratories of USA in 1932. It was designed and written by a man named Dennis Ritchie
- In the late seventies C began to replace the more familiar languages of that time like PL/I, ALGOL etc.
- ANSI C standard emerged in the early 1980s, this book was split into two tilles : The original was still called Programming in C, and the title that covered ANSI C was called Programming in ANSI C.
- It was initially designed for programming UNIX operating system.
- Major parts of popular operating systems like windows, UNIX, Linux is still written in C.
- C seems so popular is because it is reliable, simple and easy to use.
- Often heard tody is - "C has the been already superceded languages like C++, C# and Java.
-

Program

- There is a close analogy between learning English C language. The classical method of learning English is to first learn the alphabets used in the language, then learn to combine these alphabets to form words, which in turn are combined to from sentences and sentences are combined to form paragraphs.
- We must first know that alphabets, numbers, and special symbols are used in C, then how using them constants, variables & keywords

are constructed; and finally how are these combined to form an instruction.
- A group of instructions would be combined later on to form a program.

Steps in learning English language :



Steps in learning C

Alphabets Digits Special symbols	Constants Variables Keywords	Instructions	Program

- a computer program is just a collection of the instructions necessary to solve a specific problem.
- The basic operations of a computer system form what is known as a computer's instruction set.
- And the approach or method that is used to solve the problem is known as an algorithm.

So for us programming languages concern these are of two types.

1) Low level language
2) High level language.

1) Low level language :-

- Low level languages are machine level and assembly level language. In machine level language computer only understand digital numbers i.e in the form of 0 and 1.
- The assembly language is on other hand modified version of machine level language.
- Where instructions are given in English like words as ADD, SUM, MOV etc.
- So the translator used there is assembler to translate into machine level.
- In the assembly level language the data are stored in the computer register, which varies for different computer. Hence it is not portable.

2) High level language :-

- These languages are machine independent, means it is portable. The languages in this category is Pascal, Cobol, Fortran etc.
- High level languages are understood by the machine.
- So it need to translate by the translator into machine level.
- A translator is software which is used to translate high level language as well as low level language in to machine level language.

Three types of translator are follows:
Compiler
Interpreter
Assembler.

Compiler and interpreter are used to convert the high level languages into machine level languages. The program written in high level languages is known as source program and the corresponding machine level language program is called as object program. Both compiler and interpreter perform the same task but there working is different.

Integrated Development Environments (IDE):

- The process of editing, compiling, running and debugging programs is often managed by a single integrated application known as an Integrated Development Environment, or IDE for short.

- On Mac Os X, CodeWarrior and Xcode are two IDEs that are used by many programmers. Under Windows, Microsoft Visual Studio is a good example of a popular IDE.

- Kylix is a popular IDE for developing applications under Linux.

- Structure of C language program :-

1) Comment line ;
2) Preprocessor directive
3) Global variable declaration
4) Main function ()
{
Local variables ;
Statements ;
}
User defined function
}

- Comment Line
It indicates the purpose of the program. It represented as,
/* ..... */

- Comment line is used for increasing the readability of the program.

- Preprocessor Directive :-

- #include <stdio.h> tells the compiler to include information about that standard input/output library.
- It is also used in symbolic constant such as #define PI 3.14 (value).

- The stdio.h (standard input output header file) contains defination & declaration of system defined function such as printf(), scanf(), pow() etc.

- Global Declaration :- This is the section where variable are declared globally so that it can be access by all the functions used in the program. And it is generally declared outside the function.

- Main :-

- It is the user defined function and every function has one main() function from where actually program is started and it is encloses within the pair of curly braces.
- The main() function can be anywhere in the program but in general practice it is placed in the first position.

Syntax :

```
main()
{
    ____
    ____
    ____
}
```

The main() functn return value when it declared by data type as.

```
int main ()
{
 return 0;
}
```

The main function does not return any value when void (means null/empty) as
void main (void) or void main()

```
{
 printf("Hello");
}
```

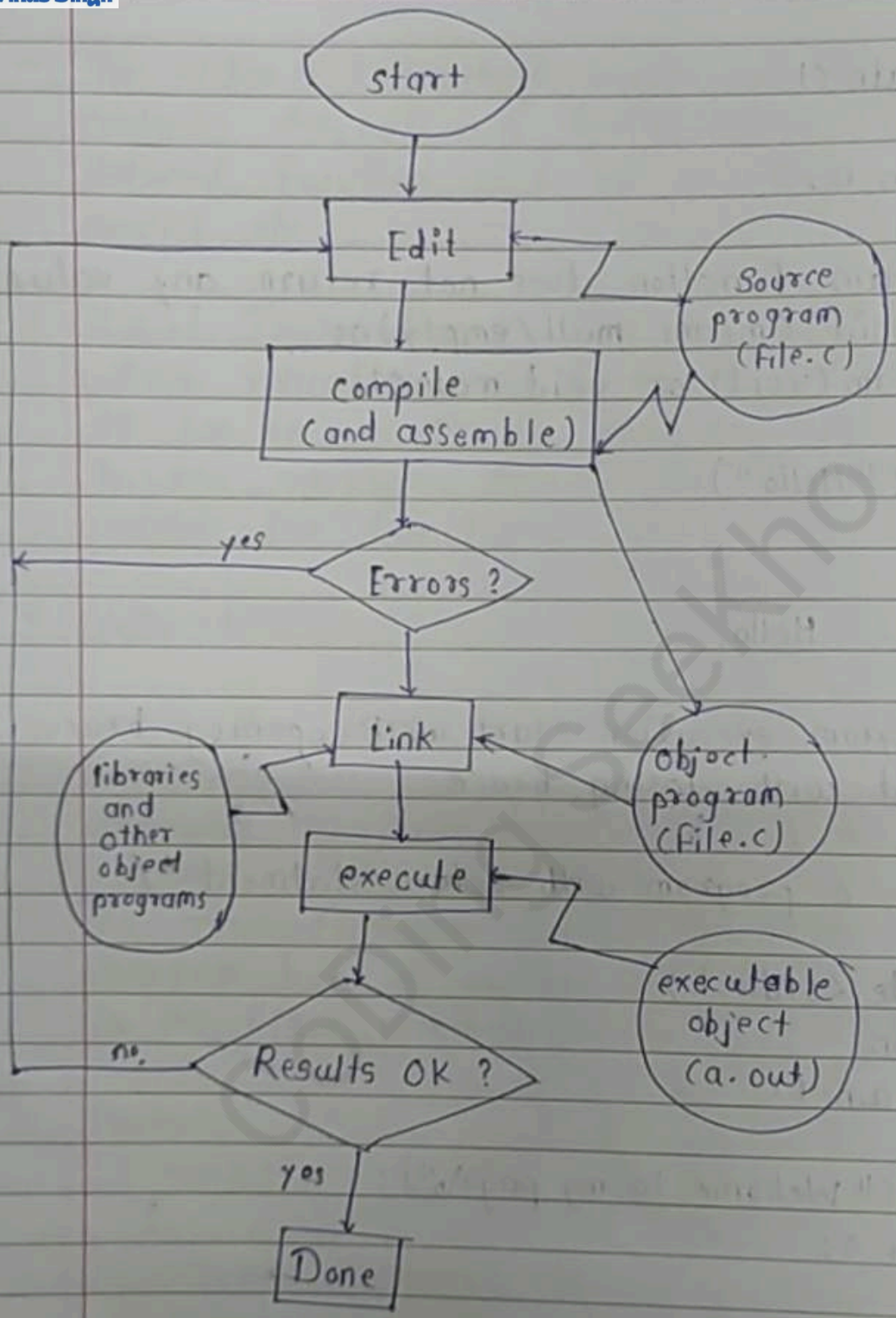
Output :- Hello.

The program execution start with opening braces and end with closing brace.

```
/* First c program with return statment */

# include <stdio.h>
~~void main~~
void main ()
{
  printf("Welcome to my page\n");
  return 0;
 }
```

Output :- Welcome to my page.



UNIX Command

Vi file.c

:

e.c File.c

a.out

```
/* Simple program to add two numbers.... */

# include <stdio.h>
void main ()
{
   int a, b, sum ;          // a,b,sum are varibles and int
   a= 150;                  is data type declared
   b = 60;
   sum = a + b;
   printf("The sum of %i and %i is = %i\n", a, b, sum);
   return 0;
}
```

Output - The sum of 150 & 60 = 210.

charachter set :- valid alphabets, number and symbols allowed in C are:

Alphabets	A, B, ..., Y, Z
	a, b, ..., y, z
Digits	0, 1, 2, ..., 9
Special symbols	~ ' ! @ # % ^ & * ( ) _
	- + = | \ { } [ ] : ;
	" ' < > , . ? /

- Identifiers :- Identifiers are users defined word used to name of entities like variables, arrays, functions, structures etc.

1) name should only consists of alphabets (both upper and lower case), digits and underscore (_) sign.
2) First characters should be alphabet or underscore
3) name should not be a keyword.
4) Since C is a case sensitive, the upper case and lower case considered differently, for example code, Code, CODE etc.
5) identifiers are generally given in some meaningful name such as value, net_salary, age, data etc.

- Keyword :- These are certain words reserved for doing specific task, these words are known as reserved word or keywords. These words are predefined and always written in lower case small letter. These keywords can't be used as a variable name as it assigned with fixed meaning.
  - Some examples are int, short, signed, unsigned, default, volatile, float, long, double, break, continue, typedef, static, do, for, union, return, while, do, extern, register, enum, case, goto, struct, char, auto, const etc.

* Data types :- Data types refers to an ~~extensive~~ extensive system used for declaring variables or functions of different types before its use.
- The type of a variables determines how much space it occupies in storage and how the bit pattern stored in interpreted.
- The value of a variable can be changed any time.
- C has the following 4 types of data types.

basic built-in data types :- int, float, double, char

Enumeration data type :- enum

Derived data type :- pointer, array, structure, union.

Void data type : void

A variable declared to be of type int can be used to contain integral values only - that is, values that do not contain decimal places.

o There are two types of type qualifier in C.

Size qualifier : short, long

Sign qualifier : signed, unsigned.

When the qualifier unsigned is used the number is always positive and when signed is used

may be positive or negative. If the sign qualifier is not mentioned, then by default sign qualifier is assumed. The range of values for signed data types is less than that of unsigned data type.

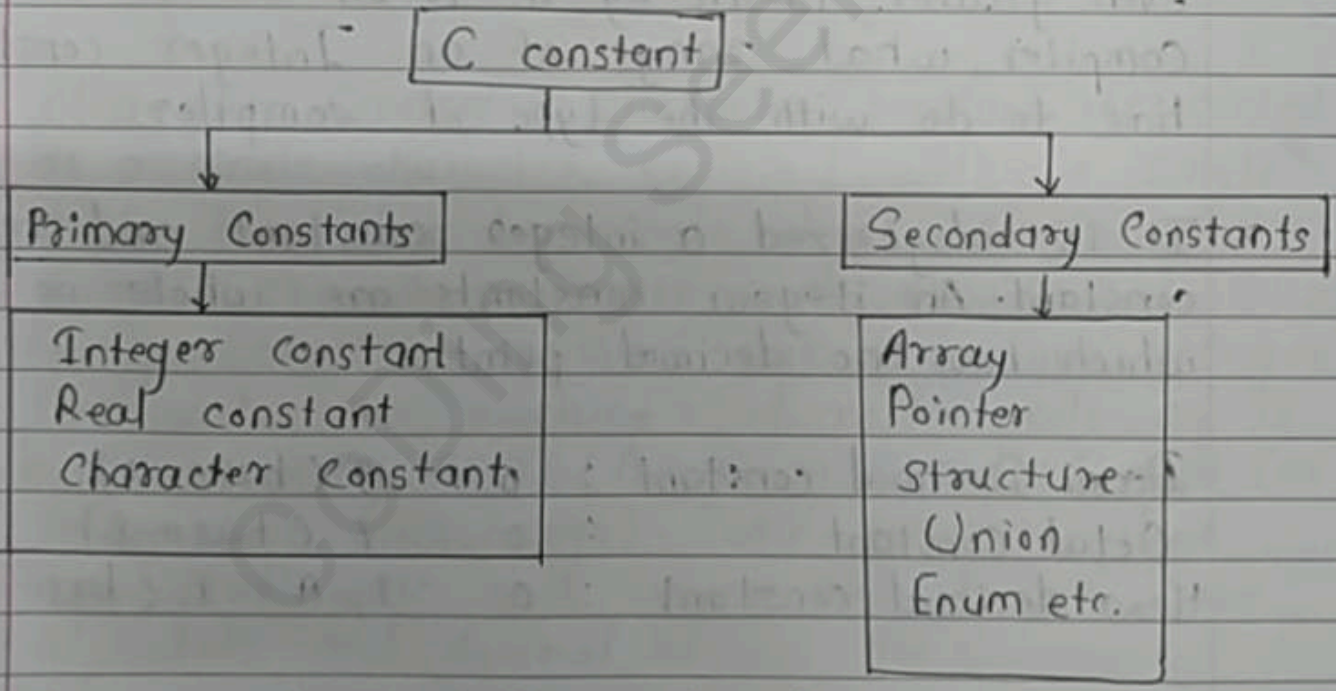
- The size and range of the different data types on a 16 bit machine is given below:

Basic data type	Data type with type qualifier	Size (byte)	Range
char	char or signed char	1	-128 to 127
	Unsigned char	1	0 to 255
int	int or signed int	2	-32768 to 32767
	unsigned int	2	0 to 65535
	short int or signed short int	1	-128 to 127
	usigned short int	1	0 to 255
	long int or signed long int	4	-2147483648 to 2147483647
	unsigned long int	4	0 to 4294967295
Float	Float	4	-3.4E-38 to 3.4E+38
double	double	8	-1.7E-38 to 1.7E+308
	Long double	10	3.4E-4932 to 1.1E+4932

- Constants - Constant is a any value that cannot be changed during program execution.
- In C, any number single character, or character string is known as a constant.
- A constant is an entity that doesn't change whereas a variable is an entity that may change.

Primary Constant
Secondary Constant

These constants are further categorized as.



Numeric constant
Character constant
String constant

- **Numerical Constant** :- Numerical constant consists of digits. It required minimum size of 2 bytes and max 4 bytes. It may be positive or negative but by default sign is always positive.
- No comma or space is allowed within the numeric constant and it must have at least 1 digit.
- For a 16-bit compiler like Turbo C or Turbo C++ the range is -32768 to 32767.
- For a 32-bit compiler the range would be even greater. Mean by a 16-bit or 32-bit compiler, what range of an Integer constant has to do with the type of compiler.

- It is categorized a integer constant and real constant. An itegeor constants are whole no. which have no decimal point.

Decimal constant : 0 ... 9 (base 10).
Octal Constant : 0 .... 7 (base 8)
Hexa decimal constant : 0 ..... 9, A ... F (base 16)

Real constant is also called floating point constant. To construct real constant we must follow the rule of,
- real constant must have at least one digit.
- It must have a decimal point.
- It could be either positive or negative.
- Defaultte sign is positive.

M T W T F S S
Page No.:
Date:
YOUVA

- No commas or blanks are allowed within a real constant

Ex- +325.34
426.60
-32.76

- To express small/large real constant exponent (scientific) form is used where number is written in mantissa and exponent from separated by eE.
- By default type of floating point constant is double, it can also be explicity it by suffix of F/F.

Character Constant :- Character constant represented as a single character enclosed within a single quote. These can be single digit, single special symbol or white spaces such as '9', 'c', '$' etc.
- Every character constant has a anique integer like value in machine's character code as if machine using ASCII (American Standard code for information interchange). Some numeric value associated with each upper and lower case alphabets and decimal integers are :

A ..... Z ASCII value (65-90)
a ..... b ASCII valbe (97-122)
0 ..... 9 ASCII value (48-59)
; ASCII value (59)

program :-

```
void main ()
{
    char a;
    printf("Enter a any key :");
    scanf("%c", &a);
    printf(" ASCII Code is : %d ", a);

    getch();
}
```

String Constant :-

- Set of characters are called string and when sequence of characters are enclosed within a double quote is a string constant.
- String constant has zero, one or more than one character and at the end of the string null character (\0) is automatically placed by compiler.

program :-

```
void main ()
{
    char a[10];
    int l, i;
    printf(" Enter a string :");
    scanf(" %s ", a);
    l = strlen (a);
```

```
for(i=0 ; i<l/2 ; i++)
{
  if( a[i] != a[l-1-i])
  {
    printf(" It is not palindrome :");
    break;
  }
}
if( i==l/2)
{
  printf(" It is palindrome. ");
}

getch();
}
```

~~Int~~ int, float, char program of size.

```
void main()
{
  int a=23;
  float b 23.29;
  char c = 'A';
  printf("%d\n%f\n%c\n",a,b,c);
  printf("%d\n", size of (int);
  printf("%d\n", size of (float);
  printf("%d\n", size of (char);

  getch();
}
```

Symbolic constant - symbolic constant is a name that substitute for a sequence of characters and, characters may be numeric, character or string constant. These constant are generally defined as the begining of program as.

define name value, here name generally written in upper case for example

```
# define MAX 10
# define CH 'b'
# define NAME "sony"
```

Maximum No. - program

```
void main()
{
    int x,y;
    printf("Enter 2 number :");
    scanf("%d%d", &x, &y);

    printf("Maximum is %d \n\n", max(x,y));

    getch();
}
```

Qwerty p

Vowels program :-

```
void main()
{
  char ch;
  printf("Enter a character:");
  scanf("%c",&ch);

  if(ch=='a' || ch=='e' || ch=='i' ||
    ch=='o' || ch=='u')

  {
    printf("%c is vowel:", ch);
  }
    else
    {
      printf("%c is not vowel.:", ch);
    }

    getch();
}
```



```
void main()
{
  char ch[100];
  printf("Enter a Name:");
  scanf("%s",ch);
  if(ch=='A
```

- Variables :- Variable is a data name which is used store some data value or symbolic names for storing program. computations and results.
- The value of the variable can be change during the execution. The rule for naming the variables is same as the naming identifier.
- Before used in the program it must be declared.
- Declaration of variables specify is name, data types and range of the value that variables can store depends upon its data types.

- Syntax :-
```
int a ;
float f;
char c;
```

- Variable Initialization - When we assign any initial value to variable during the declaration, is called initialization of variables.
- When variables is declared but contain undefined value it is called garbage value

e.g -
```
int a=20;
```
Or
```
int a;
a=20;
```

```
void main()
{
    int a=20;
    printf("%d", a);
    getch();
}
```

Expressions :- An expression is a combination of variables, constants, operators and function call. It can be arithmetic, logical & relational.

e.g- int x+y=z its wrong

int z=x+y; // arithmatic expression.

a>b // relational

a==b // logical

func(a,b) // Function call

- Expressions consisting entirely of constant values are called expressions.
- So, the expressions 121+17-110
- is a constant expression because each of the terms of the is expression is a constant value.
- But if i were declared to be an integer variable the expression 180+2-i
- would not represent a constant expression.

Operator - This a symbol use to perform some operation on variables, operands or with the constant.

- Some operator required 2 operand to perform operation or some required single operation.
- Several operators are there those are, arithematic operator, assignment, increment, decrement, logical, conditional, comma, size of, bitwise and others.

1. Arithematic Operator

- These operator used for numeric calculation. These are of either Unary arithematic operator, Binary arithematic operator. Where Unary arithematic operator required only one operand such as +, -, ++, -- & tiled. And these operators are addition, substractn, multiplicatn and division.
- Modulus cannot applied with floating point operand as well as there are no exponent operator in C.
- Unary (+) and Unary (-) is different ftom addition and substraction.
- When both the operand are integer are then it is called integer arithmetic & the result is always integer.
- When both the operand are floating point then it is called floating arithmetic and
- when operand is of integer and floating point when it is called floating arithmetic & then when operand is of integer & floating point then it is called mix type or mixed mode arithmetic.
- Add

Multiplication -

```
void main()
{
    int x,y,z;
    printf("Enter 2 numbers:");
    scanf("%d%d",&
```

```
scanf("%d %d", &x, &y);
z = x*y;
printf("Multiplication is : %d", z);

getch();
}
```

2. Assignment Operator - A value can be stored in a variable the use of assignment operator. The assignment operator (=) is used in assignment statement expression.

- Operand on the left hand side should be variable and the operand on the right hand side should be variable or constant or any expression.
- When variable on the left hand side is occur on the right hand side then we can avoid by writting the compound statement.

e.g. int x = y;
int Sum = x+y+z.

3. Increment and Decrement -

- The Unary operator ++, --, is used as increment and decrement which act as upon signal operand.
- Increment operator increases the value of variable by one Similary decrement operator decreases the value of the variable by one.
- And these operator can only used with the variable, but can't use with expression and constant as ++6 or ++ (x+y+z).

- It again categories into prefix post fix. In the prefix the value of the variable is incremented 1st, then the new value is used, where as in postfix the operator is written after the new value is used, where as in postfix the operator is written after the operand (such as m++, m--)

e.g. ① let y = 12;
z = ++y;
y = y+1.

②
```
x = 5;
y = x++;
y = x;
x = x+1;
```

```
void main ()
{
    int x=5;
    int y++;
```

```
void main ()
{
    int i,j;
    i=34;
    j= i++;
    printf("%d", j);

    getch();
}
```

4. Relational Operator - It is use to compared value of two expressions depending on their relation.
- Expression that contain relational operator is called relational operator. / expression.
- Here the value is assign according to true or false value.
  a. (a>=b) || (b>20)
  b. (b>a) && (e>b)
  c. o(b%==7)

5. Conditional Operator - It sometimes called as ternary operator. Since it required three expression as operand and it is represented as.

Syntax - exp? exp2 : exp3

e.g-
```
void main()
{
    int a=10; b=2
    int s=(a>b)?a:b;
    printf("value is:%d",s);

    getch();
}
```

Output :- value is:10.

6. Comma Operator - Comma operator is use to permit different expression to be appear in a situation where only one expression would be used.

- All the expression are separated by comma & are evaluated from left to right.

e.g-
```
void main()
{
  int a;
  for(a=0; a<500; a++)
  {
    printf("Hello");
  }
  getch();
}
```

= int , i, j, k, l ;

- ~~int~~ for(i=1 j=2 ; i<=5 ; j<=10; i++; j++)

7. Size of Operator - Size of operator is a Unary operator, which gives size of operand in terms of byte that occupied in the memory.
- An operand may: be variable, constant or data type qualifer.

e.g-
```
main()
{
  int sum;
  float f;
  printf("%d %d", sizeof(f), sizeof(sum));
  printf("%d %d", sizeof(235L), sizeof(A));

  getch();
}
```

8. Bitwise Operator - Bitwise operator permit programmer to access and manipulate of at data bit level. Various bitwise operator enlisted are-

one's complement	(~)
bitwise AND	(&)
bitwise OR	(|)
bitwise XOR	(^)
left shift	(<<)
right shift	(>>)

- These operator can seperate on integer and character value but not on float and double. In bitwise operator the function showbits() function is used to display the binary represen-tation of any integer or character value.
- In one's complement all 0 changes to 1 and all 1 changes to 0. In the bitwise OR its value would obtaining by 0 to 2 bits.
- As the bitwise OR operator is used to set on par-ticular bit in a number. Bitwise AND the logical AND.
- It operate 2 operands and operands are compared on bit by bit basic. And hence both the operand-s are of same type.

Logical or Boolean Operator - Operator used with one or more operand and return either value-zero or one.
(False) (True).
- The operand and true may be constant, variable or expressions.

Operator	Meaning
&&	AND
||	OR
!	NOT

- Where logical NOT is unary operator & other two are binary operator. Logical AND gives result true if both the conditions are true, otherwise result is false. AND logical OR gives result false if both the condition false, otherwise result is true.

Precedence and associativity of operators

Operators	Description	Precedence level	Associativity
()	function all	1	left to right
[]	array subscript		
→	arrow operator		
.	dot operator		
+	unary plus	2	right to left
-	unary minus		
++	increment		
--	decrement		
!	logical not		
~	1's complement		
*	indirection		
&	address		

(data type) type cast

size of size in byte

*	multiplication	3	left to right
/	division		
%	modulus		
+	addition	4	left to right
-	substraction		
<<	left shift	5	left to right
>>	right shift		
<=	less than equal to	6	left to right
>=	greater than equal to		
<	less than		
>	greater than		
==	equal to	7	left to right
!=	not equal to		
&	bitwise AND	8	left to right
^	bitwise XOR	9	left to right
|	bitwise OR	10	left to right
&&	logical AND	11	
||	logical OR	12	
?:	conditional operator	13	

=, *=, /=, %= &=, ^=, <<= >>= assignment operator 14 right to left

, comma operator 15

Control Statement - Generally C program statement is executed in order in which they appear in the program, But sometimes we use decision making condition for execution only a part of program, that is called control statement.

- Control statement defined how the control is transferred from one part to the other part of the program. There are several control statement like if...else, switch, while, do....while, for loop, break, continue, goto etc.

Loops in C -

- Loops :- It is a block of statement that performs set of instructions.
- In loops Repeating particular portion of the program either a specified number of time or until a particular no of condition is being satisfied.

1. While loop
2. do while loop
3. For loop

While loop -

Syntax -

```
while (condition)
{
   Statement 1;
   Statement 2;
}
OR  while (test condition)
      Statement;
```

e.g -

```
void main ()
{
   int i=0;
   while (i<5)
   {
      printf("A");
      i++;
   }
   getch();
}
```

- The test condition may be any expression: when we want to do something a fixed no of times but not known about the number of iteratn. in a program then while loop is used.
- Here first condition is checked if, it is true body of the loop is executed else, If condition is false control will be come out out of loop.

e.g - /* "wap to print 5 times welcome to C" */

```
# include <stdio.h>
void main()
{
    int p=1;
    While(p<=5)
    {
      printf("Welcome of C \n");
       p=p+1;
    }
     getch();
}
```

Output :- Welcome to C
Welcome to C
Welcome to C
Welcome to C
Welcome to C

- so as long as condition remains true statements within the body of while loop will get executed repeatedly.

- do while loop - This statement is also used for looping. The body of this loop may contain single statement or block of statement. The Syntax for writting this statement is -

- Syntax :-

```
Do
{
  statement;
```

```
}
while(condition);
```

e.g -
```
# include < stdio.h>
 void main()
{
   int X=4;
   do
   {
     printf("%d", X);
     X = X+1;
   }
   while(X<=10);
   print(" ");
}
```

Output - 4 5 6 7 8 9 10

- Here firstly statement inside body is executed then condn. is checked.

- For loop - In a program, for loop is generally used when number of iteration are known in advance.
- The body of the loop can be single statement or multiple statements.

Syntax -
```
for(initialized counter; test counter; update counter)
{
   Statement;
}
```

- Here condition expression generally users relational and logical operators.

e.g -
```
void main ()
{
  int i;
  for(i=1;i<10;i++)
  {
    printf("%d", i);
  }
  getch();
}
```

Output - 1 2 3 4 5 6 7 8 9

Nesting of loop - When a loop written inside the body of another loop then, it is known as nesting of loop. Any type of loop can be nested in any type such as while, do while

eg-
```
void main()
{
  int i,j;
  for(i=0;i<2;i++)
  for(j=0;j<5;j++)
  printf("%d %d ",i,j);
}
```

Output -
```
i=0
j= 0 1 2 3 4
i=1
j= 0 1 2 3 4
```

o Break statement (break) - Sometimes it becomes necessary to come out of the loop even before loop condition becomes false then break statement is used. Break statement is used inside loop & switch statements. It is written with the keyword as break.

- When break is encounted inside any loop, control automatically passes to the first statement after the loop.

e.g - ∀ Prime Number -

```
void main()
{
   int x,i;
   printf("Enter a no.");
   scanf("%d", &x);
   for(i=2; i<x; i++)
   {
      if(x%i==0)
      {
         printf("It is not prime number.");
         break;
      }
   }
   if(i==x)
   {
      printf("It is prime number.");
   }
   getch();
}
```

e.g-
```
void main()
{
  int j=0;
  for(j=0;j<6;j++)
  if(j==4);
  break;
  :
  getch();
}
```
Output - 0 1 2 3

- Continue Statement (key word continue)
Continuing next iteration of loop after skipping some statement of loop. When it encountered control automatically passes through the beginning of the loop.

e.g-
```
void main()
{
  int n;
  for(n=2; n<=9; n++)
  {
    if(n==4)
    continue;
    printf("%d",n);
  }
}
  printf("Out of loop");
  getch();
}
```
Output - 2 3 5 6 7 8 9 out of loop

- if statement - statement execute set of command like when condition is true.

```
If (condition)
statement
```

- The statement executed by only when condition is true.

e.g.

```
void main()
{
   int n;
   printf("Enter a no:");
   scanf("%d", &n);
   if (n>10)
    printf("number is greater");
}
```

Output - Enter a no : 12
Number is greater

- if... else statement - it is bidirectional conditional control statement that contains one condition & two possible action.
- Condition may be true or false, where non-zero value regarded as true & zero value regarded as false.

e.g.

```
void main()
{
   int x;
   printf("Enter a no:");
   scanf("%d", &x);
   if (x%2==0)
   {
     printf("%d is even", x);
   }
```

```
    else
    {
      printf("%d is odd", x);
    }
  getch();
}
```

- Nesting of if....else
When there are another if else statement in if-block or else-block:, then is called nesting of if-else.
Syntax -

```
if (condition)
{
   Statement 1;
  else
  {
     statement 2;
  }
     Statement 3;
```

- IF:-else. LADDER- In this type of nesting there is an if else statement in every else part except the last part.
- If condition is false control pass to block where condition is again checked.
- Syntax =

```
void main()
{
   int x=0;
   printf("Enter a number :");
   scanf ("%d", &x);
   if (x>0)
   {
     printf("%d is positive :", x);
   }
    else if (x<0)
   {
     printf ("%d is negative :", x);
   }
   else if (x==0)
   {
    printf("%d is neither negative or positive :", x);
   }
   getch();
}
```

Array

- Array is the collection of similar data types or collection of similar entity stored in contiguous memory location.
- ADVANTAGES - array variables can store more than one value at a time where other variable can store one value at a time.

e.g - int mark[100];

DECLARATION OF AN ARRAY-

Data type array name[size];

```
int arr[100];
int mark[100];
int a[5] = { 10,20,30,400,5 }
```

- The declaration of an array tells the compiler that, the data type, name of the array, size of the array and for each element it occupies memory space.

```
int ar[5];
ar[0], ar[1], ar[2]; ar[3], ar[4];
```

INITIALIZATION OF ARRAY:- After declaration element of local array has garboge value. If it is global or static array then it will be automatically initialize with zero.

Data type array name[size] = { value 1, value 2, value 3.... }

e.g- int ar[5] = { 20, 60, 90, 100, 120 }

- Array is always start from zero which is known as lower bound and upper value is known as upper bound and last subscript value. is one less than the size of array.
- So if i & j are not variable then the valid subscript are

```
ar[i*7], ar[i*1], ar[i+1], ar[7];
```

Total size in byte for 1D Array is -
Total bytes = size of (data type) * size of array empty.

e.g-

```
int [20];
```

Total byte = 2 * 20 = 40 byte.

ACCESSING OF ARRAY ELEMENT-

```
/* Write a program to input values into a array
and display them */
#include<stdio.h>
int main()
{
  int arr[5], i;
  for(i=0; i<5; i++)
  {
    printf("enter a value for arr[%d] \n",i);
    scanf("%d", &arr[i]);
  }
  printf("the array elements are:\n");
  for(i=0; i<5; i++)
  {
    printf("%d \n", arr[i]);
  }
  return 0;
}
```

Output :- Enter a value of arr[0] = 12
Enter a value of arr[1] = 45
Enter a value of arr[2] = 59
Enter a value of arr[3] = 64
Enter a value of arr[4] = 98
The array elements are 12 45 59 64 98

e.g:- Write a program to add 10 array elements.

```
#include<stdio.h>
  void main()
{
   int i;
   int arr[10];
   int sum=0;
   for(i=0;i<=9;i++)
   {
    printf("Enter the %d element\n",i+1);
    scanf("%d",&arr[i]);
   }
   for(i=0;i<=9;i++)
   {
     sum=sum+a[i];
   }
   printf("the sum of 10 array elements is %d",
          sum);
   getch();
}
```

Output -

```
Enter a value for arr[0]=5
Enter a value for arr[1]=10
Enter a value for arr[2]=15
Enter a value for arr[3]=20
Enter a value for arr[4]=25
Enter a value for arr[5]=30
Enter a value for arr[6]=35
Enter a value for arr[7]=40
Enter a value for arr[8]=45
Enter a value for arr[9]=50
```

Sum = 275

- while initializing a single dimensional array, it is optional to specify the size of array.

e.g - int marks[] = {99, 78, 50, 45, 67, 89}

If during the initilization of the number the intializers is less then size of array, then all the remaining elements of array are assigned value zero.

- Single dimensional arrays and functions.

```
/* program to pass array elements to a funetion */
#include<stdio.h>
void main()
{
  int arr[10]; i ;
  printf("enter the array elements \n");
  For(i=0; i<10; i++)
  {
   scanf("%d", &arr[i]);
   check (arr[i]);
  }
  getch ();
}

void check(int num)
{
  if (num %2 =0)
  {
   printf("%d is even \n", num);
  }
```

```
else
{
  printf("%d is odd \n", num);
}
  getch();
}
```

o. Two dimensional arrays - Two dimensional array is known as matrix. The array declaration in both the array i.e. single dimensional array single subscript is used in two dimensional array two subscript are is used.

Syntax-

Data type array name[row][column]

e.g- int a[2][3];

Total no. of elements = row * column is 2*3 = 6

e.g- 20 2 7
8 3 15

positions of 2-D array elements in an array are.

00 01 02
10 11 12

a[0][0]	a[0][0]	a[0][0]	a[0][0]	a[0][0]	a[0][0]
20	2	7	8	3	15
2000	2002	2004	2006		2008

Accessing 2-d array / processing 2-d array

For processing 2-d array, we use two nested for loops. The outer for loop corresponds to the row & the inner for loop corresponds to the column.

```
void main()
{
    int x[2][2], y[2][2], z[2][2], i, j;
    printf("Enter the first matrix :");
    for( i=0 ; i<2 ; i++)
    {
        for( j=0 ; j<2 ; j++)
        {
            scanf("%d", &x[i][j]);
        }
    }
    printf("Enter the second matrix:");
    for (i=0 ; i<2 ; i++)
    {
        for( j=0 ; j<2 ; j++)
        {
            scanf("%d", &y[i][j]);
        }
    }
    printf("Addition is :");
    for(i=0 ; i<2 ; i++)
    {
        for( j=0 ; j<2 ; j++)
        {
            printf("%d", z[i][j]);
        }
    }
    getch();
}
```

- Initialization of 2-d array - 2-D array can be initialized in a way similar to that 1-D array.

e.g- int mat[4][3] = {11,12,13,14,15,16,17,18,19,20,21,22};

These values are assigned to the elements row wise, so the values of elements after his this initialization are -

Mat[0][0] = 11, Mat[1][0] = 14, Mat[2][0] = 17, Mat[3][0] = 20
Mat[0][1] = 12, Mat[1][1] = 15, Mat[2][1] = 18, Mat[3][1] = 21
Mat[0][2] = 13, Mat[1][2] = 16, Mat[2][2] = 19, Mat[3][2] = 22

while initializing we can group the elements row wise using inner braces.

e.g- int mat[4][3] = {{11,12,13},{14,15,16},{17,18,19},{20,21,22}};

- And while initializing, it is neccessary to mention the 2nd dimension where 1st dimension is optional.

int mat[7][3];
int mat[2][3];

int mat[][]; } invalid
int mat[2][];

If we initialize an array as
int mat[4][3] = {{11},{12,13},{14,15,16},{17}};

We can also give the size of the 2-D array by using symbolic constant

```
#define ROW 2;
#define COLUMN 3;
int mat[ROW][COLUMN];
```

String :- Array of character is called a string. It is always terminated by the NULL character. String is a one dimensional array of character.

- char name[] = {'j', 'o', 'h', 'n', '\0'};

- ASCII value of '\0' is 0 and ASCII value 0 is 48.

- | J | o | h | N | '\0' |

- the terminating NULL important because, it is only the way that the function that work with string can know, where string end.
String can also be initialised as,
char name[] = "John";

String constant (string literal)
A string constant is a set of character, that enclosed within the double quotes and is also called a literal.

e.g- "m"
"Tajmahal"

The string constant itself becomes a pointer to the first character in array.

1000	1001	1002	1003	1004	1005	1006	1007	1008	1009
T	a	j		M	A	H	a	l	\0

It is called base address.

- String library function- there are several string library functions used to manipulate string and the prototypes for these functn are header file "string.h".

Strlen() - This function return the length of string i.e the no. of charachters in the string excluding the terminating NULL character
- It accepts a single argument which is pointer to the first character of the string.

e.g-

```
void main ()
{
  char a[10];
  int l, i;
  printf(" Enter a string : ");
  scanf(" %s ", a);
  l = strlen(a);
  for(i=0 ; i<l/2 ; i++)
  {
    if (a[i] != a[l-1-i])
    {
     printf(" It is not palindrome : ");
     break ;
    }
  }
  if (i == l/2)
  {
    printf ("It is palindrome : ");
  }

  getch();
}
```

e.g -
```
# include <stdio.h>
# include<string.h>
void main()
{
  char str[50];
  printf(" Enter a string :");
  gets(str);
  printf("Length of the string is %d \n", strlen(str));
}
```

Output - Enter a string : C in Depth
Length of the string is 8!

- Strcmp () - This Function is used compare two strings. If the two string match, strcmp() return a value 0 otherwise it return a non-zero value.

strcmp(S1, S2)
return a value :
- <0 when S1<S2
- =0 when S1=S2
- >0 when S1>S2

- The exact value returned in case of dissimilar strings is not defined, we only know that if S1<S2 then a negative value will be returned and if S1>S2 then a positive value will be returned.

e.g - /* string comparison ..... */

```
# include<stdio.h>
# include <string.h>
void main ()
{
    char str1[10], str2[20];
    printf(" Enter two strings :");
     gets(str 1);
     gets(str 2);
     if(strcmp(str1, str2) == 0)
    {
      printf(" string are same \n");
    }
    else
     {
       printf(" string are not same \n");
     }
   getch();
}
```

strcpy() - This function is used to copying one string to another string. The function strcpy(str1, str2) copies str2 to st1 including the NULL character. Here str2 is the source string & str1 is the defination string.

e.g -

```
void main ()
{
   char a[15];
   char b[15];
   strcpy (a, "Mansi");
   strcpy (a, "Aparna");
```

```
printf("\n The copy of string is : %s ", a,b);

  getch();
}
```

- strcat - This function is used to append a copy of a string at the end of the other string. If the first string is "Purva" and second string is "Belmot" then after using this function the string becomes "PurvaBelmont"

e.g -
```
void main()
{
  char a[15];
  char b[15];
  strcpy(a, "Mansi");
  strcpy(b, "Mayuri");
  strcat(a,b);
  print("\n the string is concat : %s ", a);

  getch();
}
```

* Function *

- A function is a self contained block of codes or sub program with a set of statements that perform some specific task or coherant task when it is called.

1) Library function
2) User defined function

- The user defined function defined by the user according to its requirements.

- Here in system defined function description
- Function defination - predefined, precompiled, stored in the library
- Function declaration - In header file with or Function prototype.
- Function call - By the programmer.

- User defined function

Syntax -

Return type	name of Functn	(type 1, arg 1, type 2, arg 2, type 3, arg 3)
Return type	Functn name	argument list of the above isyntax.

- Function declaration
- Function defination
- Function call

These three things are represented like

```
int function (int, int, int); /* Functn declaratn */
main ()          /* calling function */
{
 function (arg 1, arg 2, arg 3)
}
 int function (type 1 arg 1, type 2 arg 2, type 3 arg 3)
          /* Function defination */
{
   Local variable declaration;
   Statement;
```

return value;
}

- Function declaration - Function declaration is known as function prototype. It inform the compiler about three thing, those are name of the function, number and type of argument received by the function & the type of value returned by the function.

- Function defination - Function defination consists of the whole description and code of the function.

Syntax -

```
return type Function(type 1 arg 1, type 2 arg 2, type
                    arg 3) /* function header */
{
    Local variable declaration;
    Statement 1;
    Statement 2;
    Return value
}
```

The return type denotes the type of the value that functn will return & it is optional and if it is omitted, it is assumed to be int by default.

- The arguments of the function defination are known as formal arguments.
- It can't be used anywhere in the program & its existence is only within this functn.

- Function call - When the functn get called by the calling function then that is called, functn call.

e.g - function (arg1, arg, arg3)

The argument that are used inside the functn call are called actual argument.

e.g - int S= sum (a, b) // actual arguments.

- Actual argument - The arguments which are mentioned or used inside the function call is knows as actual argument & these are original value & copy of these are actually sent to the called function.

```
Function (x);
Function (20, 30);
Function (a*b, c*d);
Function (2, 3, sum(a,b));
```

- Formal Arguments - The arguments which are mentioned in function definition are called formal arguments or dummy arguments.
- These arguments are used to just hold the copied of the values that are sent by the calling function through the function call.

1) The formal argument are declared inside the parenthesis where as the local variables declared at the beginning of the function block.

2) The formal argument are automatically initialized when the copy of actual arguments are passed while other local variable are assigned values through the statements.

- Return type -
It is used to return value to the calling function.
It can be used in two way as return.
OR return(expression);

e.g- return (a);
return (a * b);
return (~~a * bb+c~~); (a * b + c);

Here the 1st return statement used to terminate the function without returning any value.

e.g-
```
/* summation of two values */
int sum( int a1, int a2)
main()
{
   int a,b;
   printf(" enter two no");
   scanf(" %d %d ", &a, &b);
   int S =sum (a,b);
   printf(" summation is = %d", s);
}
  int sum (int x1, int y1)
{
   int z= x1+y1;
   Return z;
}
```

Advantage of Function - By using Function large & difficult program can be divided in to sub programs and solved.

- Notes :-
- C program is a collection of one or more Functn.
- A Function is get called when function is followed by the semicolon.
- A Function defined when a function name followed by a pair of curly braces Any Function can be labelled called by another Function even main() can be called by other function.

Syntax :-
```
main ()
{
  function 1 ()
}
function 1()
{
statement ;
function 2;
}
function 2 ()
{
}
```

- A function can call itself again and again & this process is called recursion.
- A function can be called from other function but a function can't be defined in another function.

- Categogory of Function based on argument and return type.

i) Function with no argument & no return value.
Function that have no argument and to no return value is written as -

```
void function (void);
main ()
{
  void function ()
  {
   statement ;
   }
```

e.g-
```
void me ()
  main()
 {
  me();
  printf (" in main ");
 }
  void me ()
 {
  printf(" come on ");
 }
```

Output :- come on
in main

ii) Function with no argument but return value.
Syntax -

```
int fun (void);
 main()
{
  int r;
```

```
r = fun();
}
int fun()
{
 return(exp);
}
```

e.g-
```
int sum();
main()
{
 int b = sum();
 printf("entered %d\n, b");
}
int sum()
{
 int a, b, s;
 s = a+b;
 return s;
}
```

- Here called function is independent and are initialized.
- The values aren't passed by the calling function

iii) Function with argument but no return value-
Here the function have argument so the calling function send data to the called function but called funtion desen't return value.

Syntax-
```
void fun(int, int);
 main()
 {
  int (a,b);
 }
 void fun(int x, int y);
```

```
{
   statement;
}
```

Here the result obtained by the called function.

iv) Function with argument and return value

Here the calling function has the argument to pass to the called function and the called function returned value to the calling function.

e.g-

```
void main()
{
  int fun(int);
  int a, num;
  printf("enter value:\n");
  scanf("%d", &a);
  int num = fun(a);
  }
 int fun(int x)
{
   ++x;
     return x;
}
```

- Call by value & call by reference

There are two way through which we can pass the arguments to the function such as call by value and by reference.

- Call by value - In the call by value copy of actual argument is passed to the formal argument and the operation is done on formal argument.

When the function is called by "call by value" method, it dosen't effect content of the actual argument.

e.g.-
```
void main()
{
   int x, y;
   change(int,int);
   printf("enter two values :\n");
   scanf("%d %d", &x, &y);
   change(x,y);
   printf("value of x= %d and y= %d\n", x,y);
}
change(int a, in b);
{
   int k;
   k = a;
   a = b;
   b = k;
}
```

Output - enter two values : 12
23
Value of x=12 and y=23

2. Call by reference - Instead of passing the value of variable, address or reference is passed and the function operate on address of the variable rather than value.
- Here formal argument is alter to the actual argument, it means formal arguments calls the acutal arguments.

e.g -
```
void main()
{
   int a,b;
   change(int *, int *);
   printf("enter two values :\n");
   scanf("%d%d", &a, &b);
   change(&a, &b);
   printf("after charging two value of a = %d and
          b = %d \n:" a,b);
}
  change(int * a, int * b)
  {
     int k;
     k = *a;
     *a = *b;
      *b = k;
   printf("value in this function a = %d and
                          b = %d \n", *a, *b);
}
```

Output : enter two values :12
32
value in this function a = 32 and b = 12
After changing two value of a = 32 and b = 12

o Local, Global and static variable -

Local variable - variables that are defined with a body of function or block. The local variables can be used only in that function or block in which they are declared.

```
Function()
{
   int a,b;
   Function1();
}
Function1()
{
   int a=0;
     b=20;
}
```

- Global Variable - the variables that are defined outside of the function is global variable. All functions in the program can access and modify global variables.

e.g -
```
# include <stdio.h>
void function(void);
void function1(void);
void function2(void);
int a, b=20;
void main()
{
   printf("inside main a= %d, b= %d \n", a, b);
   Function();
   Function1();
   Function2();
}
   function()
   {
      Printf(" inside function a= %d, b= %d \n", a,b);
   }
```

```
function1 ()
{
   printf(" inside function a = %d , b = %d \n" , a,b);
}
  function 2()
{
    printf(" inside function a = %d, b = %d \n", a);
}
```

- Static variables :- static variables are declared by writing the key word static.

e.g-
```
void fun1(void);
void fun2(void);
void main()
{
   fun1();
   fun2();
}
void fun1 ()
{
   int a=10, static int b=2;
   printf(" a = %d, b = %d" , a, b);
   a++;
   b++;
}
```

Output :- a = 10 b = 2
a = 10 b = 3

- Recursion - When Function calls itself again and again then it is called as recursive function.
- In recursive calling Function and called Function are same.
- Here statement with in body of the function calls the same function and some times it is called as circular defination.

e.g- Calculate factorial of no. using recursion.

```
int fact(int);
void main()
{
   int num;
   printf("enter a number");
   scanf("%d", &num);
   f = fact(num);
   printf("factorial is= %d \n", f);
}
fact(int num)
{
   If(num==0||num==1)
return 1;
else
return(num*fact(num-1));
}
```

- Monolithic Programming - The program which contains a single function for the large program not divided the program, it is huge long pieces of code that jump back and forth doing all the tasks like single thread of execution, the program requires.

1. Difficult to check error on large programs size.
2. Difficult to maintain because of huge size.
3. Code can be specific to a particular problem i.e it cannot be reused.

- Modular Programming :- The process of subdividing a computer a computer program into separate sub-programs such as functions and subroutines is called Modular programing. Modular programming sometimes also called as structed programming.

- Storage Classes - Attributes of variables is known as storage class or in compiler point of view a variable identify some physical location within a computer where its string of bits value can be stored is known as storage class.
  - Storgeclass datatype variable name;

There are four type of storage classes and all are keywords -

1) Automatic (autto)
2) Register (register)
3) Static (static)
4) External (extern)

e.g - auto float x; or float x;
extern int x;
register char c;
static int y;

- Compiler assume different storage class based on.

1) Storage class - tells us about storage place.
2) Initial value - what would be initial value of the variable. If initial value not assigned, then what value taken by uninitialized variable.
3) Scope of the variable - what would be the value of the variable of the program.
4) Life time - It is the between the creation and distribution of a variable or how long would variable exists.

1. Automatic Storage class -
Storage - me
Default intial value - unpredictable value
Scope - local to the block or functn in which variable.
Life time - Till the control remain within function or block in which it is defined.

e.g.
```
main()
{
  auto int i;
  printf("i= ",i);
}
```

2. Register storage class - The keyword used to declare this storage class is register.

Features -
- Storage :- CPU register
Default - intial value - garbage value
Scope - local to the function or block in which it is defined.

e.g -
```
main()
{
 register int i;
 for(i=1; i<=12; i++)
  printf("%d", i);
}
```

3. Static storage class. - the keyword used to declare static storage class in static.
- Storage - memory location
- Default intial value - zero
- Scope - local to the block or funcn in which is defined.
- Life time - value of the variable persist or remain betn diffn funcn call.

e.g -
```
main()
{
 reduce();
 reduce();
 reduce();
}
reduce();
{
 static int x=10;
 printf("%d", x);
 x++;
}
```

Output - 10, 11, 12

- External storage classes - the keyword used for this class is extern.
- Storage -
- Default initial value - zero
- Scope - global
- life time - as long as program execution remain it retains.

e.g -

```
int i,j ;
void main ()
{
  printf(" i ; %d ", i);
  receive ();
  receive ();
  receive ();
  receive ();
  receive ();
}
  receive ()
{
  i= i+2;
  printf(" on increase i= %d ",i);
}
  reduce ()
{
  i= i-1;
  printf(" on reduce i= %d ", i);
}
```

Output - i = 0, 2, 4, 3, 2.

o Pointer - A pointer is a variable that store memory address or that contains address of another variable addresses are the locatn no. always contain whole no.

Data type *pointer name;

e.g-
```
void main ()
{
    int i = 105;
    int *p;
    p = &i;
{
    printf ("value of i= %d", *p);
    printf ("value of i= %d", *(&i));
    printf ("address of i = %d", &i);
    printf ("address of p= %d", p);
    printf ("address of p= %u", &p);
}
```

o Pointer Expression

- Pointer assignment

```
int i = 10;
int *p = &i;
```

- Here declaration tells the compiler that p will be used to store the address of integer value or in other word p is a pointer to an integer & *p reads the value at the address contain in p.

```
-p++;
printf("value of p= %d");
```

We can assign value of 1 pointer variable to other when their base type and data type is same or both the pointer points to the same variable as the array.

- To

```
int *p1, *p2;
p1 = &a[1];
p2 = &a[3];
```

- we can assign constant 0 to of any for that symbolic constant 'NULL' is used such as

```
*p = NULL;
```

- Pointer Arithmetic - Pointer arithmetic is different from ordinary arithemetic & it is perform relative to data type.

e.g -

```
void main()
{
  static int a[] = {20, 30, 105, 82, 97, 72, 66, 102};
  int *p, *p1;
  p = &a[1];
  p1 = &a[6];
  printf("%od", *p1 - *p);
  printf("%od", p1 - p);
}
```

/* Addition of a number through pointer */;

e.g-
```
int i = 100;
int *p;
p = &i;
p = p+2;
p = p+3;
p = p+9;
```

/* Substractn a number from a pointer. */

e.g-
```
int i = 22;
*p1 = &a;
p1 = p1-10;
p1 = p1-2;
```

o Procedence of dereference (*) Operator and increment operator and decrement operator.
This precedence level of difference operator increment or decrement operator is same & their associately from right to left.

e.g-
```
int x = 25;
int *p = &x;
```

i) `int y = *p++;` equivalent to `*(p++)`
`p = p++` or `p = p+1`

ii) `*++p;` → `*(++p)` → `p = p+1`
`y = *p`

iii) `int y = ++*p`
equivalent to `++(*p)`
`p = p+1` then `*p`

iv) y = (*p)++ → equivalent to *p++
y = *p then
p = p+1;

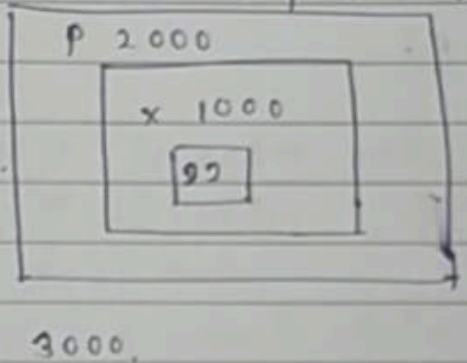
- Comparison Pointer - Pointer variable can be compared when both variable, object of same data type and it is useful when both pointers variables points to element of same array.
- ==, !=, <=, <, >, >= can be used with pointer.

e.g -
```
void main()
{
  static int arr[] = {20,25,15,27,105,96}
  int *x, *y;
  x = &a[5];
  y = &(a+5);
  if (x==y)
   printf("same");
   else
    printf("not");
}
```

- Pointer to pointer - pointer within another pointer is called pointer to pointer.

e.g -
```
Data type **p;
int x = 22;
int *p = &x;
int **p1 = &p;
printf("value of x = %d", x);
printf("value of x = %d", *p);
printf("value of x = %d", *&x);
printf("value of x = %d", **p1);
```

```
printf("value of p= %u", &p);
printf("address of p = %u", p1);
printf("address of x= %u", p);
printf("address of p1 = %u", &p1);
printf("value of p= %u)", p);
printf("value of p= %u", &x);
```



- Pointer VS array

e.g- void main()
{
static char arr[] = "Rama";
char *p = "Rama";
printf("%s %s", arr, p);

- printf("size of (p)", size of (ar));
size of (p) 2/4 bytes
size of (ar) 5 bytes

- Structure - it is the collection of dissimilar data types or heterogenous data types grouped together. It means the data types may or may not be of same type.

```
struct tagname
{
   Data type member 1;
   Data type member 2;
   Data type member 3;
    ....
   Data type member n;
};
```

OR

```
struct
{
   Data type member 1;
   Data type member 2;
   Data type member 3;
    ....
    ...
   Data type member n;
};
```

- Initialization of structure variable - structure templates can be defined locally or globally.

- Accessing structure elements -
  Dot operator is used to access to structure elements.
  - Its associativity is from left to right.

e.g-
```
# include <stdio.h>
# include <conio.h>
void main()
{
   int roll, age;
   char branch;
}
   s1, s2;
   printf ("\n enter roll, age, branch = ");
   scanf("%d %d %c ", &s1.roll, &s1.age, &s1.branch);
   s2.roll = s1.roll;
   printf(" students details = \n");
   printf("%d %d %c ", s1.roll, s1.age, s1.branch);
   printf("%d", s2.roll);
}
```

Unary, relational, arithmetic, bitwise operators are not allowed within structure variables.

- Size of structure. - size of structure can be found out using sizeof() operator with str. variable name or tag name with the keyword.

```
sizeof(s1);
sizeof(s2);
```

- Array of structures - When database of any element is used in huge amount, we prefer array of structures.

e.g -

```
#include<stdio.h>
#include<string.h>
struct student
{
  char name[30];
  char branch[25];
  int roll;
};
void main()
{
   struct student s[200];
   int i;
   s[i].roll=i+1;
   printf("\n Enter informatn of students:");
   for(i=0;i<200;i++)
   {
   printf("\n Enter the roll no: %d\n", s[i].roll);
   printf("\n Enter the name:");
   scanf("%s", s[i].name);
   printf("\n Enter the branch:");
   scanf("%s", s[i].branch);
   printf("\n");
}
   printf("\n Displaying information of students
        :\n\n");
   for(i=0;i<200;i++)
   {
    printf("\n\n Information for roll no: %d %d:\n",
         i+1);
    printf("\n Name:");
    puts(s[i].name);
```

```
    printf("\nBranch:");
    puts(s[i], branch);
}
}
```

- Array within structures.

```
struct student
{
  char name[30];
  int roll, age, marks[5];
};
   struct student s[200];
```

- Nested structure - When a structure is within another structure, it is called Nested structure.

```
struct student
{
  element 1;
  element 2;
  ....
  ....
   struct student1
{
   member 1;
   member 2;
}
   variable 1;
    ...
    ...
   element n;
}
```

variable 2;

- Nesting of structure within itself is not valid. Nesting of structure can be extended to any level.

```
Struct time
{
  int hr,min;
};
  Struct day
{
   int date,month;
   struct time t1;
};
struct student
{
  char nm[20];
  struct day d;
}
  stud1, stud2, stud3;
```

- Passing structure elements to function - We can pass each element of the str. through funcn but passing individual element is difficult when no. of str. element increases.

```
#include <stdio.h>
#include <string.h>
void main()
{
  struct student
  {
   char name[30];
   char branch[25];
   int roll;
  }
  struct student s;
   printf("\n enter name = ");
   gets(s.name);
   printf("\n Enter roll: ");
   scanf("%d", &s.roll);
   printf("\n Enter branch :");
   gets(s.branch);
   display(name, roll, branch);
}
   display(char name, int roll, char branch)
  {
   printf("\n name = %s, \n roll = %d, \n branch = %s",
          s.name, s.roll, s.branch);
  }
```

- Union - Union is derived data type contains collection of digsimilar element.

- Syntax of Union-

```
union student
{
   datatype member 1;
   datatype member 2;
};
```

e.g-
```
struct student
{
   int i;
   char ch[10];
};
   struct student s;
```

- Nested Union - When one union is inside the another union it is called nested of union.

e.g-
```
union a
{
  int i;
  int age;
};
union b
{
  char name[10];
  union a aa;
};
  union b bb;
```

- Dynamic memory Allocation-
The process of allocating memory at the time of execution or at the runtime, is called dynamic memory location. These library fun are called as dynamic memory allocation functn.

- malloc() - this functn use to allocate memory during run time, its declaratn is void* malloc(size);

e.g-
```
void main()
{
  int n, avg, i, *p, sum=0;
  printf("enter the no. of marks");
  scanf("%d", &n);
  p=(int *) malloc (n* size(int));
  if(p== null)
  printf("not sufficient!");
  exit();
}
  for(i=0; i<n; i++)
  scanf("%d", (p+i));
  for(i=0; i<n; i++)
  printf("%d", *(p+i));
  sum = sum + *p;
  avg = sum/n;
  printf("avg=%d", avg);
```

- Calloc() - Similar to malloc only difference is that calloc funcn use to allocate multiple block of memory.

e.g-
```
int *p=(int *) calloc (5,2);
int *p=(int *) calloc (5, sizeof(int));
```

- realloc() - The function realloc use to ~~challenge~~ the change the size of the memory block & it alter the size of the memory block without lossing the old data, it is called reallocatn of memory.

e.g-

```
#include<stdio.h>
#include<alloc.h>
void main()
{
   int i,*p;
   p=(int*)malloc(5*sizeof(int));
   if(p==null):
   {
    printf("space not available");
    exit();
   }
   printf("enter 5 integer");
   for(i=0;i<5;i++)
   {
    scanf("%d",(p+i));
   int* ptr=(int*)realloc(9*sizeof(int));
   if(ptr==null)
   {
    printf("not availabk");
    exit();
   }
   printf("enter 4 more integer");
   for(i=5;i<9;i++)
   scanf("%d",(p+i));
   for(i=0;i<9;i++)
   printf("%d",*(p+i));
}
```

- Free () - Functn free() is used to relase space allocated dynamically, the memory released by free() is made available to heap again.

e.g void *(ptr)

- Free (p) - When program is terminated, memory relased automatically by the operating system.

- Dynamic array - Array is the example where memory is organized in continguous way, in the dynamic memory allocation functn used such as malloc(), calloc(), realloc().

- Subscript notation
- Pointer notation

e.g -

```
# include <stdio.h>
# include <alloc.h>
void main()
{
  printf("enter the no. of values");
  scanf("%d", &n);
  p=(int *) malloc(n* size of int);
  If (p==null);
  printf("not available memory");
  exit ();
}
  for(i=0; i<n; i++)
{
    printf("enter an integer");
    scanf("%od", &p[i]);
    for( i=0; i<n; i++)
    {
```

```
    printf("%d", p[i]);
  }
}
```

- File Handling :- File - the file is a permanent storage medium in which we can store the data permantly.

- Types of file can be handled. -
  1. sequential file
  2. random access file
  3. binary file.

- File Operatn - opening a file -
  Before performming any type of operatn, a file must be opened and for this fopen() fnctn is used.

eg. `FILE *fp = fopen("abc.c", "r");`

- File pointer - the file pointer is a pointer variable which can be store the address of a special file that means it is based upon the file pointer a file gets opened.

- Declaratn of a file pointer - `FILE * var;`

- Modes of open -
  Read mode 'r'/rt
  Write mode 'w'/wt
  Appened Mode 'a'/at

- Reading - a character from a file.
getc() is used to read a character into a file.

- Writing - character into a file.

putc() is used to write a charcter into a file:-
puts (character-var, file-ptr);

- CLOSING A FILE -
Fclose () func^n close a file
Fclose (file-ptr);
Fcloseall () is used to close all the opened file at a time.

- File Operation -
1) creat^n of a new file
2) writing a file
3) closing a file

- Before performing any type of allocat^n operat^n we must have to open the file c, lang..
communicate with file using A new type called file pointer.

- Operation with Fopen()

File pointer = Fopen(" FILE NAME ", "mode open");

- Reading & write a characters from / to a file
Fgetc() is used for reading a charcter from the file.

- Syntax - character variable = fgetc (file pointer)

- fputc() is used to writing a character to file.

e.g- Copy a file to another

```
#include <stdio.h>
void main()
{
   FILE *fs, *fd;
   char ch;
   If (fs= Fopen("scr.txt", "r")==0);
   {
   printf("sorry... The source file cannot be opened");
   return;
   }
   If(fd=Fopen("dest.txt", "w")!=0);
   {
   printf("Sorry.... The destinatn file cannot be
          opened");
   fclose(fs);
   return;
   }
   while (ch= fgets(fs)!= EOT)
   fputc(ch, fd);
   fcloseall();
}
```

- Reading & Writing a string from/to a file:
getw() is used for reading a string from the file.
- Syntax : get (file pointer);

putw () is used to writing a character to a file.

```
#include <stdio.h>
#include <stdlib.h>
void main ()
{
   FILE *fp;
   int word;

  /* place the word in a file */
  fp= fopen ("dgt.txt", "wb");
  If (fp== NULL)
  {
  printf("Error opening File");
   exit (1);
  }
    word = 94;
   putw(word, fp);
   If (ferror (fp))
  printf(" Error writing to file \n");
  else
  printf(" Successful write \n");
  Folose (fp);

  /* reopen the file */

fp= fopen (" dgt.txt", "rb");
```

```
IF (fp == NULL)
{
  printf(" Error opening file ");
  exit(1);
}

/* extract the word */
word = getw(fp);
IF (ferror(fp))
printf(" Error reading file \n");
 else
  printf(" Successful read: word = %d \n", word);

/* clean up */
fclose(fp);
}
```

- Reading & writing a string from/to a file

fgets() is used for reading a string from the file
- Syntax : fgets (string, length, file pointer);

fputs() is used to writing a character to a file
- Syntax:

```
fputs : (string, file_pointer);

# include <string.h>
# include <studio.h>
void main (void)
{
  FILE *stream;
  char string[] = "This is a test";
```

```
char msg[20];

/* open a file for update */
stream = fopen("Dumy.file", "w+");

/* write a string into the file */
fwrite (string, strlen(string), 1, stream);

/* seek to the start of the file */
fseek (stream, 0, SEEK_SET);

/* read a string from the file */
fgets (msg, strlen(string)+1, stream);

/* display the string */

printf ("%s", msg);
fclose (stream);
}
```