

CPTC 2019 (ODD)**GROUP – “A”**

1	d	6	c	11	a	16	c
2	a	7	a	12	c	17	d
3	c	8	b	13	d	18	c
4	d	9	a	14	b	19	b
5	c	10	c	15	a	20	c

GROUP – “B”

Q.2. Define an algorithm. Write an algorithm to convert the temperature Fahrenheit to Celsius. (4)

Ans. : Algorithms are one of the most basic tools that are used to develop the problem solving logic.

An algorithm can be defined as a finite sequence of explicit and unambiguous instructions in stepwise logical manner up to a finite number of times until the solution of the problem is achieved. However, algorithms can have steps that repeat (iterate) or require decisions (logic and comparison) until the task is completed.

Step – 01 REM Algorithm to convert the temperature from Fahrenheit to Celsius

Step – 02 Declare Fah, Cel as Real

Step – 03 Read Fah

Step – 04 Calculate $Cel = (5.0 / 9.0) * (Fah - 32)$

Step – 05 Print “Equivalent Temperature of “,Fah,” °F = “, Cel, “°C”

Step – 06 Stop

Q.2. Explain different types of programming languages. (4)

Ans. : A programming language is a formal language comprising a set of instructions that produce various kinds of output. Programming languages are used in computer programming to implement algorithms. Most programming languages consist of instructions for computers. There are programmable machines that use a set of specific instructions, rather than general programming languages.

There are basically three types of programming languages:

Machine Language : A machine language consists of the numeric codes for the operations that a particular computer can execute directly. The codes are strings of 0s and 1s, or binary digits ("bits"), which are frequently converted both from and to hexadecimal (base 16) for human viewing and modification. Machine language instructions typically use some bits to represent operations, such as addition, and some to represent operands, or perhaps the location of the next instruction. Machine language is difficult to read and write, since it does not resemble conventional mathematical notation or human language, and its codes vary from computer to computer.

Assembly Language : Assembly language is one level above machine language. It uses short mnemonic codes for instructions and allows the programmer to introduce names for blocks of memory that hold data. One might thus write "add pay, total" instead of "0110101100101000" for an instruction that adds two numbers. Assembly language is designed to be easily translated into machine language. Although blocks of data may be referred to by name instead of by their machine addresses, assembly language does not provide more sophisticated means of organizing complex information. Like machine language, assembly language requires detailed knowledge of internal computer architecture. It is useful when such details are important, as in programming a computer to interact with peripheral devices (printers, scanners, storage devices, and so forth).

High Level Language : A high-level language is a programming language designed to simplify computer programming. It is "high-level" since it is several steps removed from the actual code run on a computer's processor. High-level source code contains easy-to-read syntax that is later converted into a low-level language, which can be recognized and run by a specific CPU.

Most common programming languages are considered high-level languages. Examples include:

C++	Objective C
C#	Pascal
Cobol	Perl
Fortran	PHP
Java	Python
JavaScript	Swift

Q.3. Write a 'C' program to find the bigger of given two numbers using conditional operator. (4)

Ans.: // program to find bigger of two given numbers using conditional operator

```
#include<stdio.h>

main()
{
    int a,b,c;

    printf("\nEnter any two integers : ");

    scanf("%d %d",&a,&b);

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

```
printf("\nBigger of two numbers is : %d",c);
return 0;
}
```

Run:

Enter any two integers : 34 28

Bigger of two numbers is : 34

Run:

Enter any two integers : 14 82

Bigger of two numbers is : 82

Q.3. What is an operator? Describe different types of operators used in C programming. (4)

Ans.: Operators are the verbs of any programming language. Operators do the action to perform the user computed values in the program. In an expression the operands are subjects and objects of those verbs which are in the form of operators upon which action is to be performed during processing.

Operators are the symbols that are used in program to manipulate operands in the form of data and variables embedded in the arithmetic and/or logical expressions.

Classification of 'C' Operators

All 'C' compilers support a numbers of different operators which has been classified into two broad categories.

1. Binary Operators (Require 2 operands on both sides of operator)
2. Unary Operator (Require 1 operand on left or right side of operator)

These operators have been further divided into a number of subcategories :

1. Binary Operators are further subdivided into : Arithmetic Operators (% * / + -), Relational Operators (< <= > >= != ==), Logical Operators (&& || !), Assignment Operators (= %= *= /= += -= &= |= ^= <<= >>=), Bitwise Operators (& ^ |), Conditional Operator (?:)
2. Unary Operators are also further subdivided into : Arithmetic Operators(- +), Increment Operator (++), Decrement Operator (--), Bitwise Operator (~), Logical Operator (!), Special Operators([] () * & . -> (type) sizeof())

Q.4. What is structure in C? How structure is declared? (4)

Ans. : Structure is a collection of heterogeneous (dissimilar) data items representing through a user defined To store related information in the form of record, 'C' provides a special data type : struct. More than one different data items can be stored and accessed through structure and is accessed and stored by a single variable name.

The structure should be first declared before use. The syntax for declaring a structure is :

Syntax :

```

struct structure_name
{
    data_type    variable 1;
    data_type    variable 2;
    data_type    variable 3;
    :
    :
    :
    data_type    variable n;
};

```

```

struct structure_name
{
    data_type    variable 1;
    data_type    variable 2;
    data_type    variable 3;
    :
    :
    :
    data_type    variable n;
} variable ;

```

Ex.:-

```

struct student_record
{
    char    roll[10];
    char    name[30];
    float   course_fee;
};

```

```

struct student_rec    studrec;

```

```

struct student_rec    studrec = { "1010023457", "AJAY KUMAR", 4500.75 };

```

1 Record

1010023457

AJAY KUMAR

4500.75

Where each declaration of variable inside the structure template is called Member of Structure.

Accessing Structure Elements :→

The elements of a structure variable are accessed using dot operator “.”. It refers an individual element of a structure.

Syntax: structure_variable_name . structure_member_name ;

To use for Input :-

Ex. :- studrec.roll; studrec.name; &studrec.course_fee; etc.

To use for Access :-

Ex. :- studrec.roll; studrec.name; studrec.course_fee; etc.

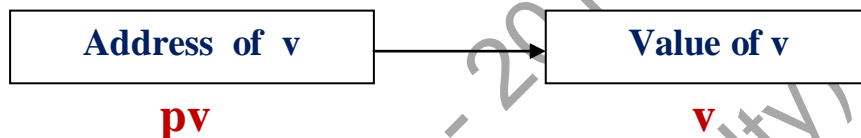
Q.4. What is pointer? How pointer variable is declared and initialized? (4)

Ans. : A Pointer is a variable that represents the location (rather than the value) of a data item within the memory, such as a variable or an array element. Pointers can be used to pass information back and forth between a function and its reference point. Pointers provide a way to return multiple data items from a function via function arguments. Pointers also permit references to other functions to be specified as arguments to a given function. This has the effect of passing functions as arguments to the given function.

Every data items stored within memory. It occupies one or more contiguous memory cells (i.e., adjacent words or bytes). The number of memory cells required for storing data item depends on the type of data items. The data item can be accessed by its address within memory. This address of location can be determined by a variable preceded by '&' called address operator, that evaluates the address of its operand.

Let us assign the address of a variable say v to another variable pv.

pv = &v will store the address of variable v into pv.



Where & is a **Unary Operator** always printed by '%u' format string.

Pointer variables must be declared before use. When a pointer variable is declared, the variable name must be preceded by an **Asterisk (*)**. This identifies the fact that the variable is a pointer. The data type that appears in the declaration refers to the Object of the pointer, i.e., the item that is stored in the address represented by the pointer, rather than the pointer itself.

Syntax :

data-type *pointer-variable-name ;

Ex.:--

int *ptr;

int x, *px;

px = &x; // Storing address of x into pointer variable px

Q.5. Define the function with suitable example. (4)

Ans. : A Function is a self-contained program segment that carries out some specific, well-defined task. Every 'C' program consists of one or more functions. One of these functions must be called main.

If a program contains multiple functions, their definitions may appear in any order, though they must be independent of one another. That is, one function definition cannot be embedded within another.

A function will carry out its intended action whenever it is accessed (i.e., Called) from some other portion of the program. Once the function has been carried out its intended action, control will be returned to the point from the function was accessed.

Generally, a function will process information passed to it from the calling portion of the program, and return a single value. Information will be passed to the function through special identifier called Arguments or Parameters and returned through return statement.

In particular, the proposed ANSI standard permits each of the argument data types within function declaration to be followed by an argument name, that is ,

Syntax :

data_type function_name (type1 arg1, type2 arg2, , type-n arg-n) ;

Where arg1, arg2, arg3, , arg-n refer to the first argument, the second argument and so on.

Function declaration written in this form is called Function Prototype. Their use is not mandatory in 'C'.

Ex. :-- int add (int a , int b) ;

//Program to demonstrate function to Add two nos.

#include<stdio.h>

main()

{

// Declaring Prototype of User Defined Functions

int add(int, int);

// Declaring local variables

int n,m,s;

printf("\nEnter Any Two Intgers : ");

scanf("%d %d",&n,&m);

s = add(n,m); // Calling function add()

printf("\nSum of %d and %d = %d\n",n,m,s);

return 0;

}

//Defining above declared functions

int add(int j, int k)

```

{
    int h; //declare local variable to function add()
    h=j+k;
    return h; //returning the value of h to calling function
}

```

Q.5. What is call by value and call by reference? Explain with suitable example. (4)

Ans. : The mechanism used to pass data to a function is via argument list, where individual arguments are called Actual Arguments. These arguments are enclosed in parentheses after the function name. The actual arguments must correspond in number, type and order with formal arguments specified in the function definition. The actual argument can be constants, variables, array names or expressions.

There are two approaches to passing arguments to a function. These are :--

- Call by Value.
- Call by Reference.

☞ **Call by Value** In this approach, the names of actual arguments are used in the function call. In this way the values of the actual arguments are passed to the function. When control is transferred to the called function, the values of the actual arguments are substituted to the corresponding formal arguments and the body of the function is executed. If the called function is supposed to return a value, it is returned via return statement.

For Example : int sum(int a, int b)

```

{
    int c;
    c = a + b;
    return c;
}

```

Function Call : int s = sum(x, y); // Function Call by Value

☞ **Call by Reference** In this approach, the names of actual arguments are used in the function call. In this way, the addresses of the actual arguments are passed to the function. When control is transferred to the called function, the addresses of the actual arguments are substituted to the corresponding formal arguments and the body of the function is executed. The formal arguments are declared as pointers to types that match the data types of the actual arguments.

This approach is of practical importance while passing arrays and structures among function, and also for passing back more than one value to the calling function.

Example : void swap(int *a, int *b)

```

{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
    return;
}

```

Function Call : `int x=56, y=34;`
`swap(&x, &y);` // Function Call by Reference

Q.6. Write a C program to find biggest number in a given array.

(4)

Ans. :

// Program to find Biggest Numbers from a list of 20 data in 1-D Array

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

```
main()
```

```
{
```

```
    int num[20],i, big;
```

```
    printf("\nEnter the 20 numbers : \n");
```

```
    for(i=0;i<20;i++)
```

```
        scanf("%d",&num[i]);
```

```
    big = num[0];
```

```
/* Finding the Biggest No. */
```

```
    for(i=1;i<20;i++)
```

```
        if (num[i]>big)
```

```
            big = num[i];
```

```
    printf("\nThe Biggest No. Found is %d ",big);
```

```
    return 0;
```

```
}
```

Q.6. Write a C program to input 20 students name and print them.

(4)

Ans. :

//Program to take input of 20 students name and print them

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

```
main()
```

```
{
```

```
    char name[20][30];
```

```
    int i;
```

```
    printf("\nEnter the name of 20 students and press enter after each name : \n");
```

```
    for(i=0;i<20;i++)
```



```

{
    gets(name[i]);
    fflush(stdin); // to clean input buffer after each input
}

printf("\n\nThe name of each students are :\n");
for(i=0;i<20;i++)
    printf("\n(%2d)  %s",i+1,name[i]);
return 0;
}

```

GROUP - "C"

Q.7. Write a C program to find the value of Y using

(6)

$Y(x,n) = 1+x$, when $n = 1$

$= 1 + x/n$, when $n = 2$

$= 1 + x^n$, when $n = 3$

$= 1 + n^x$, when $n > 3$ or $n < 1$

(by the help of nested if)

Ans.

// program to find value of y for given conditions

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

```
#include<math.h>
```

```
main()
```

```
{
```

```
    int  x, n;
```

```
    float s;
```

```
    printf("\nEnter the values of x and n : ");
```

```
    scanf("%d %d",&x,&n);
```

```
    if(n==1)
```

```
        s = 1 + x;
```

```
    elseif(n==2)
```

```

    s = (float) 1 + (x / n);
elseif(n==3)
    s = 1 + pow(x,n);
elseif (n<1 or n>3)
    s = 1 + pow(n,x);

printf("\n\nThe value of Y = %f",s);
return 0;
}

```

Q.7. Write a C program to print the following output using for loop.

(6)

```

5 5 5 5 5
4 4 4 4
3 3 3
2 2
1

```

Ans.

//Program to print shape

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

```
main()
```

```

{
    int r,c;
    printf("\n\nThe asked shape is :\n\n");
    for(r=5;r>=1;r--)
    {
        for(c=r;c>=1;c--)
            printf("%5d",r);
        printf("\n");
    }
}

```

Q.8. Define the data types of C language. How many data types are available in C language? (6)

Ans. A data-type in C programming is a set of values and is determined to act on those values. C provides various types of data-types which allow the programmer to select the appropriate type for

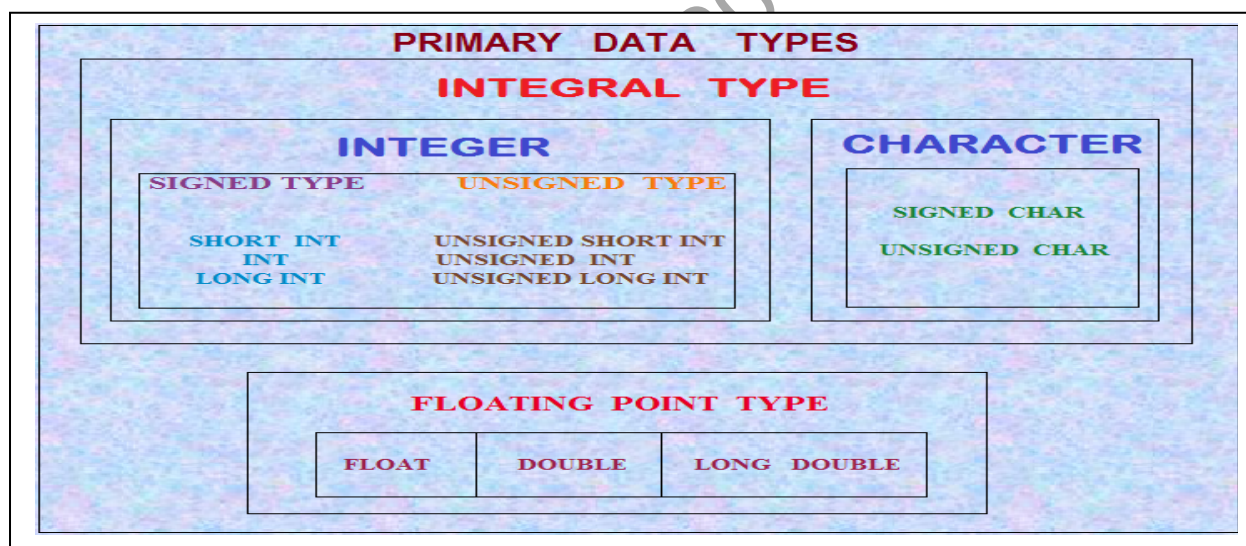
the variable to set its value. The data-type in a programming language is the collection of data with values having fixed meaning as well as characteristics. Some of them are an integer, floating point, character, etc. Usually, programming languages specify the range values for given data-type.

C Data Types are used to:

- Identify the type of a variable when it declared.
- Identify the type of the return value of a function.
- Identify the type of a parameter expected by a function.

ANSI C provides three types of data types:

1. Primary(Built-in) Data Types:
void, int, char, double and float.
2. Derived Data Types:
Array, References, and Pointers.
3. User Defined Data Types:
Structure, Union, and Enumeration.



SIZE and RANGE of DATA TYPES on a 16-Bit Machine

Type	Size (Bits)	Range
char or signed char	8	-128 to +127
short int or signed short int	8	-128 to +127
unsigned char	8	0 to 255
int or signed int	16	-32768 to +32767
unsigned int	16	0 to 65535

long int or signed long int	32	-2,147,483,648 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295
float (single precision)	32	-3.4E-38 to +3.4 E+38
double (double precision)	64	-1.7E-308 to +3.4E+308
long double (Large Double Precision)	80	-3.4E-4932 to +1.1E+4932

Primary Data Types

Every C compiler supports five primary data types:

void	As the name suggests, it holds no value and is generally used for specifying the type of function or what it returns. If the function has a void type, it means that the function will not return any value.
int	Used to denote an integer type.
char	Used to denote a character type.
float, double	Used to denote a floating point type.
int *, float *, char *	Used to denote a pointer type

Derived Data Types

C supports three derived data types:

Data Types	Description
Arrays	Arrays are sequences of data items having homogeneous values. They have adjacent memory locations to store values.
References	Function pointers allow referencing functions with a particular signature.
Pointers	These are powerful C features which are used to access the memory and deal with their addresses.

User Defines Data Types

C allows the feature called *type definition* which allows programmers to define their identifier that would represent an existing data type. There are three such types:

Data Types	Description
Structure	It is a package of variables of different types under a single name. This is done to handle data efficiently. "struct" keyword is used to define a structure.
Union	These allow storing various data types in the same memory location. Programmers can define a union with different members, but only a single member can contain a

	value at a given time. It is used for
Enum	Enumeration is a special data type that consists of integral constants, and each of them is assigned with a specific name. "enum" keyword is used to define the enumerated data type.

Q.8. If ages of Ram, Shyam and Ajay are input through the keyboard.

(6)

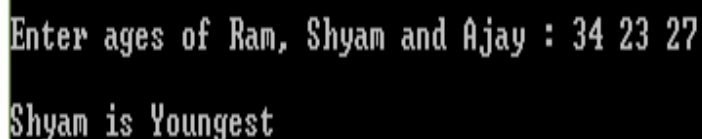
Write a program to determine youngest of three.

Ans. :

//Program to check youngs of three persons

```
#include<stdio.h>

main()
{
    int ram_age, shyam_age, ajay_age;
    printf("\nEnter ages of Ram, Shyam and Ajay :");
    scanf("%d %d %d",&ram_age,&shyam_age,&ajay_age);
    if(ram_age<shyam_age && ram_age<ajay_age)
        printf("\nRam is Youngest");
    else if(shyam_age<ajay_age)
        printf("\nShyam is Youngest");
    else
        printf("\nAjay is Youngest");
    return 0;
}
```



```
Enter ages of Ram, Shyam and Ajay : 34 23 27
Shyam is Youngest
```

Q.9. If lengths of three sides of a triangle are input through the keyboard,

(6)

write a program to calculate area of triangle.

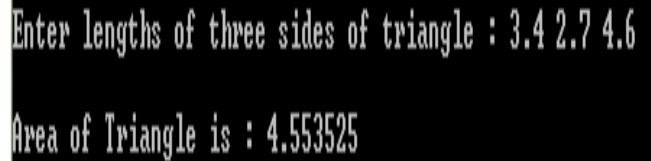
Ans. :

//Program to calculate area of a triangle whose length of sides are given

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

main()
{
    float a, b , c, s, area;
```

```
printf("\nEnter lengths of three sides of triangle : ");
scanf("%f %f %f",&a,&b,&c);
s=(a+b+c)/2;
area = sqrt(s*(s-a)*(s-b)*(s-c));
printf("\nArea of Triangle is : %f",area);
return 0;
}
```



```
Enter lengths of three sides of triangle : 3.4 2.7 4.6
Area of Triangle is : 4.553525
```

Q.9. What is an array? Write a program to search an element from an array.

(6)

Ans. : An **ARRAY** is a collection of homogeneous data elements (i.e., of same data type) described by a single variable name. Each individual element of an array is referred by a **Subscripted Variable** or **Integer Constant**. These subscripted variables or indexed variables are enclosed within square brackets and affixed with array variables.

If single subscript is required to refer to an element of an array, the array is known as One Dimensional or Linear Array. If two subscripts are required to refer to an element of an array, the array is known as Two Dimensional Array and so on.

Note: The no. of dimensions supported in an array depends upon RAM capacity and operating system used in the computer system.

Declaring an ARRAY :→

Syntax :

```
DataType ArrayName [ dim-1 ][ dim-2 ] ... [ dim-n ] ;
```

An array in a program, must contain following information :→

- ◆ The type and name of the array.
- ◆ The number of subscripts in the array.
- ◆ The total number of memory locations to be reserved, or more specifically, the maximum value of each subscript.

```
Ex. :--      # define   MAX      100

              int      mat[MAX];

              float    num[20];

              char     subject[12];  etc.
```

The first element position in an array is accessed as 0 (zero) and last as (MAX – 1).

```
Ex. :--      num[10];    //The first element is num[0] and last is num[9].
```

Initialising an ARRAYOne Dimensional Array

```
int num[5] = { 16, -5, -12, 8, 10 };
```

```
int n[ ] = { 2, 4, 7, 1, 6, 8, -9, 10, -45 };
```

In this case, the size of the array will be assumed as the number of elements supplied with it.

```
int a[10] = { 1, 2, 4, 7, 8, -3, 9 };
```

In this case, the Garbage values will be accepted for rest of the elements, because the sufficient data have not been supplied to the array according to its declared size.

Input in 1-D Array :

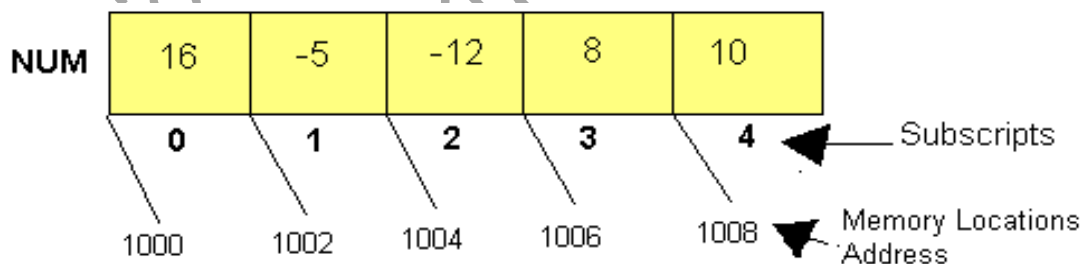
```
int a[50], k;
for( k=0; k<n; k++)
    scanf("%d", &a[k]);
```

Output of 1-D Array:

```
for ( k = 0; k<n; k++)
    printf("%7d", a[k]);
```

Let us understand the array within the memory through this figure :—

```
int num[5] = { 16, -5, -12, 8, 10 };
```



Representation of Array in Memory

Formula to Calculate the Address of an Element in 1-D Array

= Base Address + Column No. x Size of (Data Type)

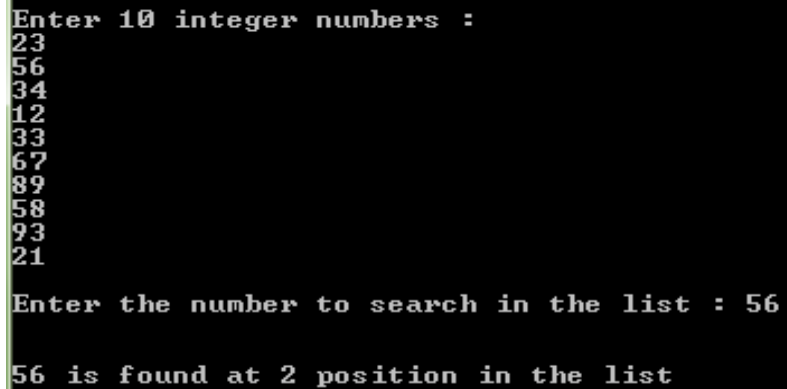
// Data Type Size is implicitly assumed in the formula

Ex. : $\&\text{num}[3] = 1000 + 3 \times 2 = 1006$

//Program to search an element from an array

```
#include<stdio.h>

main()
{
    int arr[10], j, val, flag=0;
    printf("\n\nEnter 10 integer numbers : \n");
    for(j=0;j<10;j++)
        scanf("%d",&arr[j]);
    printf("\nEnter the number to search in the list : ");
    scanf("%d",&val);
    for(j=0;j<10;j++)
        if(arr[j] == val)
        {
            flag=1;
            break;
        }
    if(flag == 1)
        printf("\n\n%d is found at %d position in the list",val,j+1);
    else
        printf("\n\n%d does not exist in the list");
    return 0;
}
```



```
Enter 10 integer numbers :
23
56
34
12
33
67
89
58
93
21
Enter the number to search in the list : 56
56 is found at 2 position in the list
```

Q.10. Define the switch statement with suitable example.

(6)

Ans. : The **switch** statement causes a particular group of statement to be chosen from several available groups expressed under **case** followed by a **value**. The selection is based upon the current value of an expression that is included within the switch statement and control is transferred in that section of the matching **case value**.

Syntax :

```
switch ( expression )
{
    case val-1 :
        statement block-1;
        break;
    case val-2 :
        statement block-2;
        break;
    :
    :
    case val-N :
        statement block-N;
        break;
    default :
        statement-block-default
}
```

Where expression takes any given value from val-1, val-2, , val-N, the control is transferred to that appropriate case.

In each case, the statements are executed and then the break statement transfers the control out of switch statement.

The default keyword is, usually mentioned at the end of switch statement, used if the value of the expression does not match any of the case values.

//Example Program Name : WeekDay.c

//Program to display day name of given weekday no.

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

```
int main()
```

```
{
```

```
    int dayNo;
```

```
    printf("\n\nEnter any day no. of a Week [0...6] : ");
```

```
    scanf("%d",&dayNo);
```

```
    switch(dayNo)
```

```
{
```

```

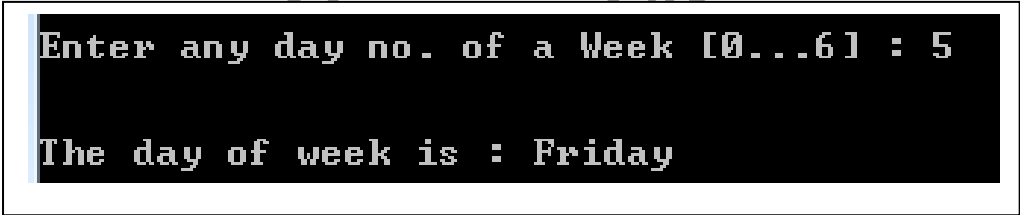
case 0: printf("\n\nThe day of week is : Sunday");
        break;
case 1: printf("\n\nThe day of week is : Monday");
        break;
case 2: printf("\n\nThe day of week is : Tuesday");
        break;
case 3: printf("\n\nThe day of week is : Wednesday");
        break;
case 4: printf("\n\nThe day of week is : Thursday");
        break;
case 5: printf("\n\nThe day of week is : Friday");
        break;
case 6: printf("\n\nThe day of week is : Saturday");
        break;
default : printf("\n\nSorry ! Wrong Input...");

```

```

}
}

```



```

Enter any day no. of a Week [0...6] : 5

The day of week is : Friday

```

Q.10. What is loop? Define for loop with syntax and example.

(6)

Ans. : ‘C’ Language provides facility to handle such situations where a task or set of tasks is to be executed up to a finite number of times controlling through a specific relational or logical expression, say condition. These activities are referred as iterative operations. The whole form of activity done in ‘C’ like languages is known as Looping Operation, and the construct used for it is called Looping Construct or Statement.

‘C’ supports three types of constructs for Looping operations.

Looping Statements

- ◆ for () Statement
- ◆ while () Statement
- ◆ do ()...while Statement

for () Loop

The **for()** loop construct is an entry-controlled looping structure usually used in the situation where the loop execution time is known in advance. This statement includes an expression that specifies an initial value for an index, another expression that determines whether or not the loop is continued and a third expression that allows the index to be modified at the end of each pass.

Syntax :

```
for ( expr1 ; expr2; expr3)
    statement(s);
```

```
for ( n = 1 ; n<=100 ; n++)
    printf("%5d",n);
```

Where –

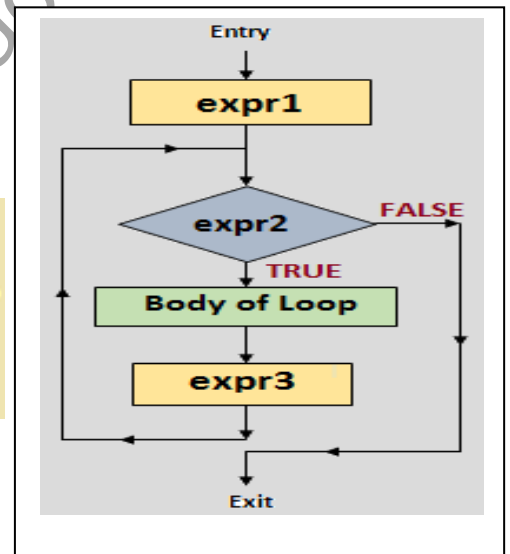
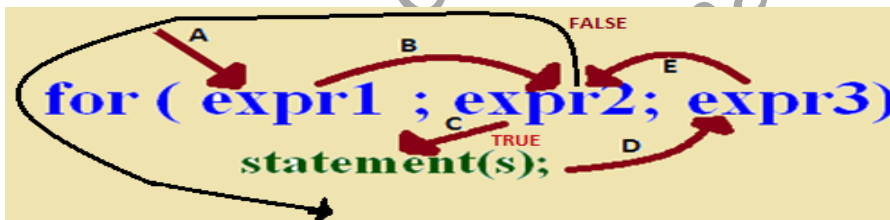
expr1 : Initialization of Looping Variable

expr2 : Conditional Expression (Output 0 / 1)

expr3 : Update Expression for Loop Variable

In the syntax of **for()** loop, **expr1** is used to initialise some parameters (called an index) that controls the looping action, **expr2** represents a condition that must be satisfied for the loop to continue execution, and **expr3** is used to alter the value of the parameter initially assigned by **expr1**.

Ex: for (n = 1 ; n<=100 ; n++)
 printf("%10d",n);



Output :

Nos from 1 to 100 are:							
1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88
89	90	91	92	93	94	95	96
97	98	99	100				

//Example Program Name : count-1to100-A.c

//Program to Print nos. from 1 to 100

```
#include<stdio.h>

main()
{
    int n;

    printf("\n\tNos from 1 to 100 are:\n\n");
    for(n=1;n<=100;n++)
        printf("%10d",n);

    return 0;
}
```

Some More Forms of for() Loop :

Syntax :

```
expr1;
for ( ; expr2;expr3 )
    statement(s);
```

Example Code :

```
n=1; // expr1
for ( ; n<=100 ;n++ )
    printf("%10d",n);
```

Syntax :

```
expr1;
for ( ; expr2; )
{
    statement(s);
    expr3;
}
```

Example Code :

```
n=1; // expr1
for ( ; n<=100 ; )
{
    printf("%10d",n);
    n++; // expr3
}
```

Syntax:

```
expr1;
for ( ; ; )
{
    expr2;
    statement(s);
    expr3;
}
```

Example Code :

```
n=1; // expr1
for ( ; ; )
{
    if ( n > 100) // expr2
        break; //Terminating Loop
    printf("%10d",n);
    n++; // expr3
}
```

Q.11. Write a C program to find roots of a quadratic equation.

(6)

Ans. :

//Program to calculate Roots of a Quadratic Eqn.

```
#include<stdio.h>
#include<math.h>
int main(void)
{
    float a, b, c, d, z, r1, r2;
    printf("\nEnter values of Coefficients a b c: ");
    scanf("%f %f %f",&a,&b,&c);
    d = b*b - 4*a*c;
    //sqrt() function returns Square Root value
    //abs() function returns +ve value
    z = sqrt(abs(d));
    r1 = (-b + z)/(2*a);
    r2 = (-b - z)/(2*a);
    if(d<0)
    {
        printf("\n\nBoth Roots are Imaginary and Unequal :\n");
        printf("\n\nRoot #1 = %fi",r1);
        printf("\n\nRoot #2 = %fi",r2);
    }
    else if(d>0)
    {
        printf("\n\nBoth Roots are Real and Unequal :\n");
        printf("\n\nRoot #1 = %f",r1);
        printf("\n\nRoot #2 = %f",r2);
    }
    else
    {

```

```

printf("\n\nBoth Roots are Real and Equal :\n");
printf("\n\nRoot #1 = Root #2 = %f",r1);
}
return 0;
}

```

```

Enter values of Coefficients a b c: 3 2 5

Both Roots are Imaginary and Unequal :

Root #1 = 0.913886i
Root #2 = -1.580552i

```

```

Enter values of Coefficients a b c: 3 8 4

Both Roots are Real and Unequal :

Root #1 = -0.666667
Root #2 = -2.000000

```

```

Enter values of Coefficients a b c: 2 4 2

Both Roots are Real and Equal :

Root #1 = Root #2 = -1.000000

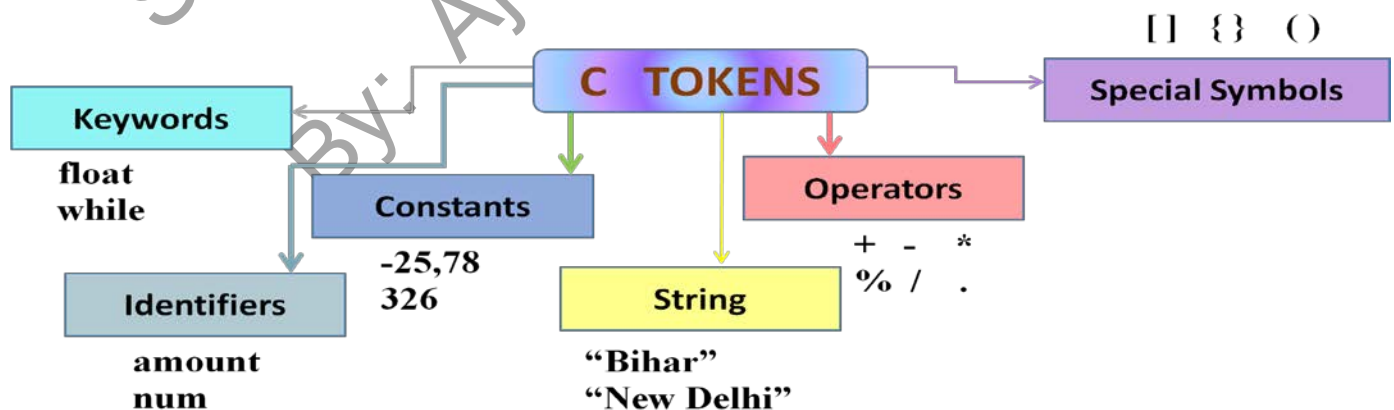
```

Q.11. What is token? How many tokens are used in C language? Explain in detail.

(6)

Ans.: C TOKENS

In a 'C' program, the smallest individual units are known as C Tokens. An individual word or punctuation marks is called Tokens. C programs are written using these tokens and syntax of the language. C language has 6 types of tokens, shown below:



KEYWORDS

Every C word is classified as either a keyword or an identifier. All keywords have fixed meanings which cannot be changed. Keywords serve as building blocks for program statements.

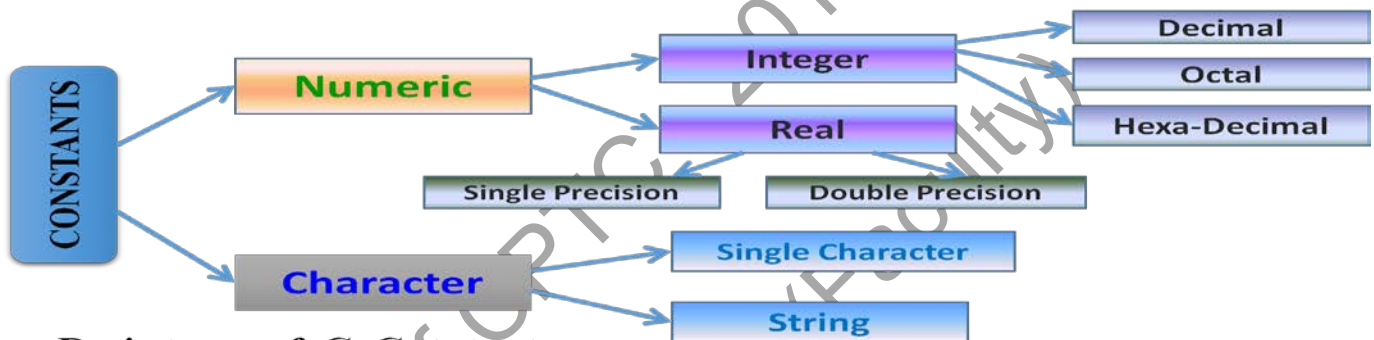
These keywords cannot be used as user-defined names such as variable names, function names, array names, or structure names by programmers.

There are 32 ANSI C Keywords

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

CONSTANT

In C program, Constants refer to fixed values that do not change its magnitude during execution of a program. There are several types of constants supported by C.



Basic types of C Constants

Integer Constants

An Integer constant contains digits (0...9), plus sign (+), or minus sign (-). There are three types of Integer constants namely, *Decimal*, *Octal*, and *Hexadecimal*.

Decimal Integers : Decimal Integers consist of a set of digits from 0...9, preceded optionally with either + or - sign. Embedded spaces, commas, period, and non-digit characters are not permitted.

Ex: 125, 21456, -987, 0, +65 etc. (allowed)

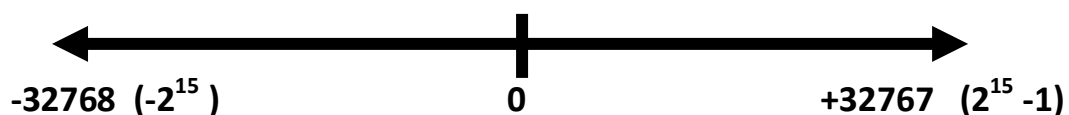
12 45 ; 3,456 ; \$100 ; 4.5 etc. (not allowed)

Signed Decimal Integers:

By default, an Integer constant is considered as Signed Integer, which may be either positive or even negative integer. It contains decimal digits from 0 9, a leading + or a - sign. It is represented in 16-bits.

Its values range from -32768 to +32767.

Ex: 67, 0, -98, +567 etc.



Unsigned Decimal Integers :

An Integer followed by U or u is referred as Unsigned (only positive) integers. It contains decimal digits from 0 ... 9 without any sign. These are always a positive values. It is also represented in 16-bits. Its values range from 0 to 65535.

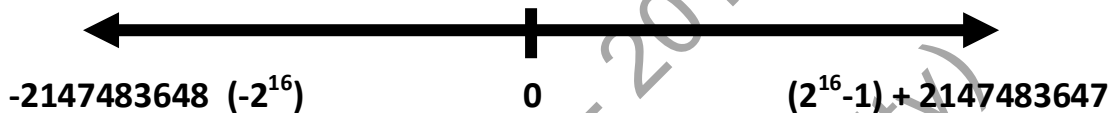
Ex: 654u, 0u, 589U, 56987u etc.



Signed / Unsigned Long Decimal Integers :

By default, An Integer followed by L or l is referred as Signed Long Integer. It contains decimal digits 0 ... 9, with leading + or _ sign, followed by suffix l or L. It is represented in 32-bits. Its values range from -2147483648 to + 2147483647.

Ex: 198765432 l, 7654382L, 0L, -48761200 l etc.



Unsigned Long Decimal Integers are represented followed by suffix ul or UL without any sign. They are always positive. It is also represented in 32 bits. Its values range from 0 to 4294967293.

Ex : 8745693269 ul, 0 ul, 345 UL etc.



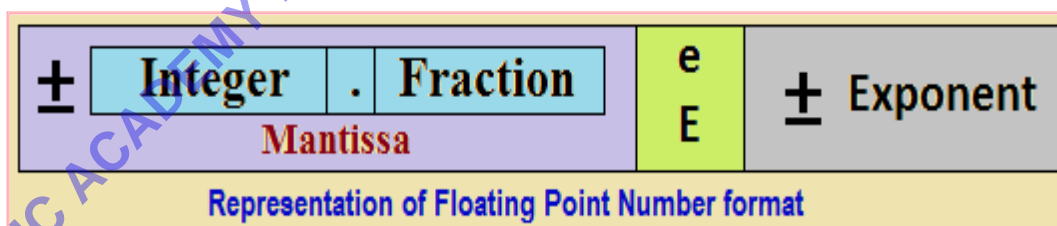
Real Constants

Real constants in C are termed as *Floating Point Constants*. These quantities are represented in decimal notation, having a whole number followed by a period sign (decimal point) and the fractional parts. These numbers are called floating point because its period sign may be shift left or right from its original position, and accordingly a product of 10 to the power its shifting position value with + or – sign is placed to maintain its magnitude. These values are always approximate values, not exact.

Ex: 0.2345, 12.67, -374.127654, +5.67, 87.0, 16. etc

Floating Point Constants are normally represented as Single/Double Precision quantities. The suffixes f or F is used to force Single Precision (32 bits) quantity, l or L to extend Double precision (64 bits) quantity, and LL or LL to extend Long Double precision (80 bits) quantity.

A Real or Floating Point number in C may also be expressed in *Exponential (or Scientific)* notation. A floating point number has two parts. First part namely Mantissa is either a real number expressed in decimal notation using period sign or an integer with optional leading + or - sign. The second part namely the Exponent is an integer with an optional plus (+) or a minus (-) sign. The letter e or E separates the mantissa and exponent parts.



The exponent represents the numerical value 10 to the power with + or – sign, depending upon the float position of period symbol.

Ex: 0.234e5f ($0.234 \times 10^{+5}$), -12.67e-7 lf (-12.67×10^{-7})
 0.0LF, -7.1E-7 (-7.1×10^{-7}), 15E-24LF (15×10^{-24}) etc.

Single Character Constants

A Single Character constant in C contains a single character enclosed within a pair of single quotes ('...'). It is represented in 8 bits in memory. Its covers all the characters whose ASCII code values range from -128 to +128 (for Signed Character) and range from 0 to 255 (for Unsigned Characters).

Digits within single quotes can not be processed arithmetically. They are also treated same as an alphabetic character. Ex: 'A' , 'a' , '5' , ';' , '=' , '≠' , '␣' (blank space) etc



String Constants

A String constant in C is a sequence of characters enclosed within pair of Double Quotes ("....."). The characters may be alphabets, numbers, and/or special characters, and even blank spaces. Actually, string is represented in memory in consecutive sequential spaces. A string is considered to be ended with a special character null('\0') whose ASCII code value is 0 (decimal) or 0x0 (hexadecimal). In memory it is considered as array of characters.

Ex: "Welcome to C Programming"

"This is year 2020"

"My age is : 22"

"All are \$\$\$\$ Well"

"1234567.789" (can not be numerically processed)

Escape Sequence Constants

Escape Sequence Character constants are represented using leading '\ (backslash) character and a specific predefined character together enclosed within pair of single quotes. These characters are not visible on input or output device, rather their effects can be seen only.

ESCAPE SEQUENCE CHARACTER CONSTANTS				
Constant	Meaning		Constant	Meaning
'\a'	Bell alert (Audible)		'\v'	Vertical Tab
'\b'	Back Space		'\"'	Single Quote
'\f'	Form feed		'\"'	Double Quote
'\n'	New Line		'\?'	Question Mark
'\r'	Carriage return (Enter Key)		'\\'	Back Slash
'\t'	Horizontal Tab		'\0'	null