

CPTC 2018 (ODD)**GROUP – “A”**

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

GROUP – “B”

Q.2. What is a library function? Mention any two library functions in C. (4)

Ans. : Library functions in C language are inbuilt functions which are grouped together and placed in a common place called library. Each library function in C performs specific operation. Programmers can make use of these library functions to get the pre-defined output instead of writing their own code to get those outputs. These library functions are created by the persons who designed and created C compilers. All C standard library functions are declared in many header files which are saved as file_name.h. Actually, function declaration, definition for macros are given in all header files. A programmer does include these header files in their C program using “#include<file_name.h>” command to make use of the functions those are declared in the header files. Then, this C program is compiled by compiler and executed.

Here is two predefined library functions are : `sqrt()` and `pow()`. Both are mathematical library function to perform specific predefined mathematical operations. Before using these functions, “math.h” named header files must be specified in the beginning of program using “#include<math.h>” directive.

`sqrt()` function : This function accepts a double constant/variable as parameter, and returns Square Root of that value.

Declaration : `double sqrt(double x);`

Usage : `double y = sqrt(23.45);` return : 4.842520

`double y = sqrt(x);`

`pow()` function : This function accepts two double constant/variable as parameter first for coefficient (x) and another for power (y), and returns value of x^y .

Declaration : `double pow(double x, double y);`

Usage : `double val = pow(12,3);` return : 1728

```
double val = pow(a,b);
```

Q.2. Write a C program to print “hello” message using while loop.

(4)

Ans. :

//Program to print “hello” message on screen.

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

```
int main()
```

```
{
```

```
    printf(“\n\nhello\n”);
```

```
    return 0;
```

```
}
```

Q.3. Explain the different data types in C.

(4)

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

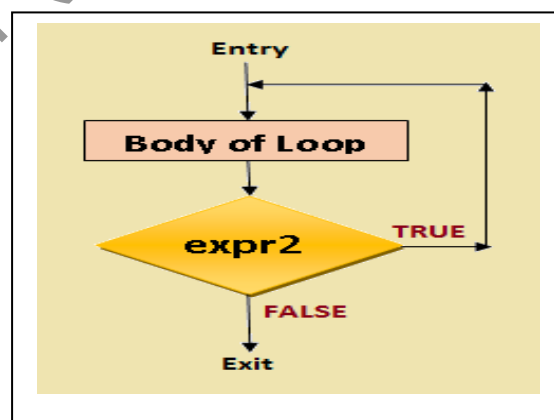
Q.3. What is the general form of “do ... while” statement? Explain with example. (4)

Ans.: **do...while()** is an **Exit-Controlled** Looping construct. In **do ... while ()** statement, it is not known in advance that how many times a statement will be executed. It is known that the statement will be executed at least once. In that case, there is no need to test the condition at the beginning. The condition is tested at the end followed by **semicolon (;)** to determine that should the body of the loop be executed again or not. On getting **TRUE** result of condition, control goes back to execute next pass to the body of loop. On getting **FALSE**, the loop terminates. The loop control variable must be initialized before initiating loop. The update expression to modify the value of loop control variable is placed inside the body of loop.

Syntax :

```

expr1; //Initialize Loop variable
do
{
    Statement-Block;
    expr3; // Looping variable update expression
} while ( expr2 ); // testing the condition
  
```



where **expr2** is a constant, a variable or an expression. The statement-block is executed repeatedly till the expression evaluates to a non-zero (True) value.

Example. :

```

int n = 1;

do
{
    printf("%5d", n);
  
```

```

++n;

} while (n<=100);

```

Q.4. Write a C program to find the sum and average of three numbers from the user. (4)

Ans. :

//Program to print sum and average of three numbers taken from users

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

```
int main()
```

```
{
```

```
    int  a, b, c, sum;
```

```
    float  avg;
```

```
    printf("\n\n Enter any three integers : ");
```

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

```
    sum = a + b + c;
```

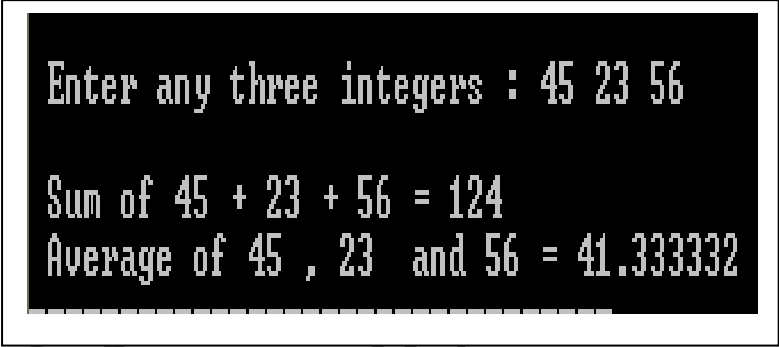
```
    avg = (float) sum / 3;
```

```
    printf ("\n Sum of %d + %d + %d = %d",a, b, c, sum);
```

```
    printf ("\n Average of %d , %d and %d = %f",a, b, c, avg);
```

```
    return 0;
```

```
}
```



```

Enter any three integers : 45 23 56

Sum of 45 + 23 + 56 = 124
Average of 45 , 23 and 56 = 41.333332

```

Q.4. Convert the following mathematical expression into C expression. (4)

$$(i) \sqrt{1+x} + \frac{\log \cos(2x)}{1+|y|} \quad (ii) \frac{a b}{c^2-d}$$

Ans. :

$$(i) \text{ Sqrt } (1 + x) + (\log (\cos (2 * x)) / (1 + \text{abs } (y)))$$

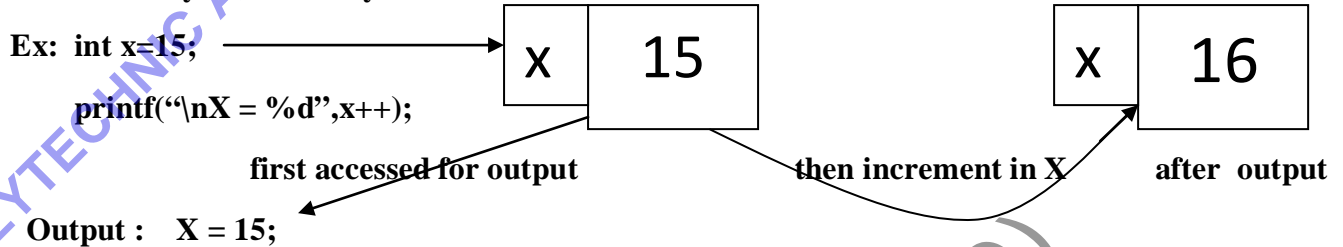
$$(ii) (a * b) / (c * c - d)$$

Q.5. How does X++ differ from ++X? Explain with suitable examples. (4)

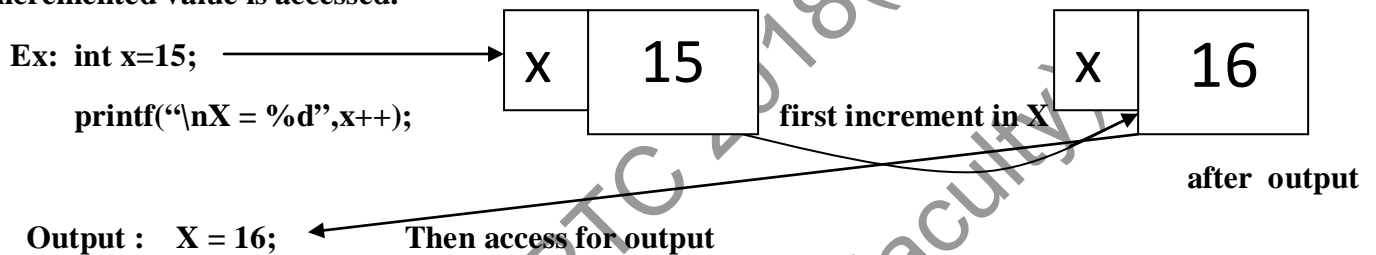
Ans. : In both the expressions the operator ‘++’ used is referred to as increment operator and is of unary nature. The operator ‘++’ increments the value of its operand by 1. Its function and behaviour changes with its position with operand. When the operator ‘++’ is associated with in prefix position with operand means the operator is placed before the operand it becomes pre-increment operator, where as when it is associated with in postfix position with operand means the operator is placed just after the operand it becomes post-increment operator.

In the expressions :

X++ : In this expression, the increment operator ++ is used as post-increment operator. In this case, the expression is accessed from left to right, first value of X is accessed and then the value of X is incremented by 1 in memory location.



++X : In this expression, the increment operator ++ is used as pre-increment operator. In this case, the expression, the value of X is incremented by 1 initially inside the memory location and then the incremented value is accessed.



Q.5. Explain the syntax of `scanf()` and `printf()` functions with suitable examples.

(4)

Ans. :

scanf() Function

scanf() is a formatted input library function which allows to take one or more inputs through standard input device Keyboard. It converts the input data into the specified format and stores into the calculated addresses of the specified memory variable locations. Address of variable is calculated using dereferencing (address) operator '&' which precedes the specified variables. The string of specifiers must be enclosed within pair of double quotes.

List of variables are separated by comma. Only string variables having %s or %[...] specifiers do not need dereferencing. Because string variables are itself de-referenced variables.

Syntax : `scanf("format specifier string", list of address of variables);`

In `scanf()` function, more than one format specifiers can be given separated by comma, space, or tab space. List of address of variables must be written after closing double quotes separated by comma. Each variable must be preceded with address '&' operator, except string variables.

Input Variable Declaration:

```
int          a, b;;

unsigned int  num, hexnum, octnum;
```

```
long      dist;
float     x, y;
char      ch, name[30], msg[100];
```

Examples :

```
scanf("%c,%d,%f,%o",&ch,&a,&x,&octnum);
```

```
input: s,45,7.98,063 ↵
```

```
scanf("%c %d %f",&ch,&a,&x);
```

```
input: s 45 7.98 ↵
```

```
scanf("%x,%u %ld",&hexnum,&num,&dist);
```

```
input: 0x2b3,34545 789432 ↵
```

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

```
Input: Ajay Kumar Singh ↵
```

```
Scanf("%[...]",msg);
```

```
Input: Everybody has to attained the class. ↵
```

printf() Function

printf() function is a formatted library output function which sends data from specified memory variable onto the standard output device, Monitor screen. It reads data from the variable location and convert it according to the specified format specifier. It displays the output data at the current cursor position. This function also permits to print string constants. Specifiers must be enclosed within pair of double quotes.

The list of output variables must be separated by comma in the list. This function also allows to use escape sequence characters along with specifier string within the double quotes.

It displays the output as the format string is customized. It replaces the format specifiers with the value of variables.

Syntax :

```
printf("string constant");
```

```
printf("format specifier string", list of variables);
```

In printf() function, is frequently used to interact with user of program through messages displayed on screen helping the user for required input operations. The output of the program can be displayed in customized format by placing messages along with format specifiers and the list of variables whose

output data are to be displayed on screen in the expected format. The format specifier string can have message as string constant including escape sequence characters, comma, space, or tab space. List of address of variables must be written after closing double quotes separated by comma.

Format Specifiers forms:

%[±] w.p type-specifier

String Specifier : %±w.ps

W : field width of output data in no. of column occupied on output screen, must be integer constant.

P : first column position of the text started to be displayed

+ : Right aligned text

- : Left aligned text

Ex: printf("\nEnter any two integers : "); // Output : Enter any two integers :

printf("\nSum of %d and 5d = %d",a,b,sum); // Output : Sum of 25 and 35 = 60

printf("\nArea of triangle = %.2f",area); // Output : Area of triangle = 327.45

Q.6. Write the algorithm and draw the flowchart to find the sum and product of given two numbers. (4)

Ans. : Algorithm :

Step – 01 REM To find Sum and Product of two numbers

Step – 02 Declare A, B, S, P as Integer

Step – 03 READ A, B

Step – 04 Calculate $S = A + B$

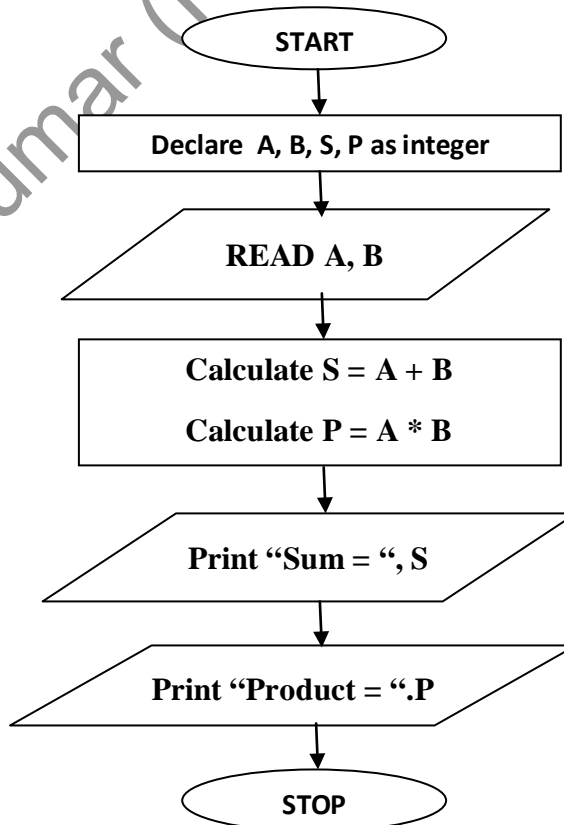
Step – 05 Calculate $P = A * B$

Step – 06 Print "Sum = ", S

Step – 07 Print "Product = ".P

Step – 08 STOP

Flowchart :



Q.6. What is pointer give examples.

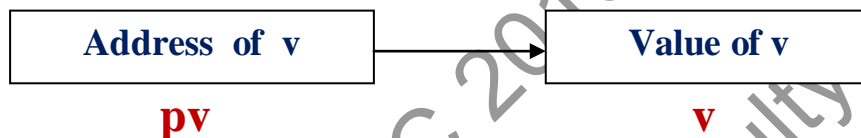
(4)

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

GROUP - "C"

Q.7. Write a C program to find biggest of given two numbers.

(6)

Ans.

// Program to find Biggest of Two Numbers

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

```
int main()
```

```
{
```

```
    int a, b, big;
```

```
    printf("\n\n Enter any two Integers : ");
```

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

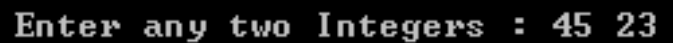
```
    printf("\n\n The Biggest Number is : ");
```

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

```
    printf("%d",big);
```


```
    return 0;
```

```
}
```



```
Enter any two Integers : 45 23
```

```
The Biggest Number is : 45
```



```
Enter any two Integers : 29 34
```

```
The Biggest Number is : 34
```

Q.7. Explain various if structures in C language.

(6)

Ans. : There are many situations arise when there will be possibility to change the order of execution of program. When a situation arises to choose one from two or more options, it needs to take a decision based on a conditional expression. 'C' provides such facility to perform this type of tasks using specific constructs, referred as Decision Making Statements.

- ❖ if ... Statement
- ❖ if ... else... Statement
- ❖ if ... else... [Nested]
- ❖ if ...else if... [Ladder]

if () Statements

The if () statement is used to take decision on the given condition. The condition may be either relational or logical expression. If the condition will be TRUE, it will return the execution of statement block; otherwise they are bypassed.

Syntax :

if (condition)

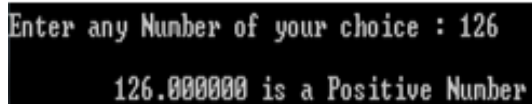
Statement-block;

If there will be more than one statements in the statement block, then they must be enclosed within the pair of “{ }”.

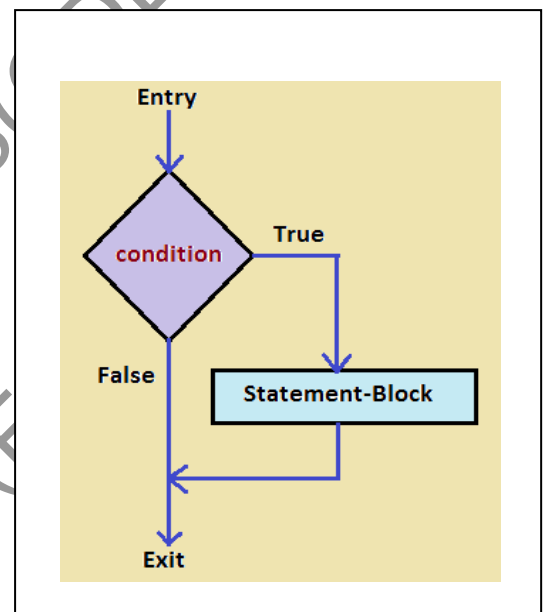
//Program Name : testIf.c

// To demonstrate the use of if()

```
main()
{
    float num;
    printf("Enter any Number : ");
    scanf("%f",&num);
    if (num > 0)
        printf("\nThe number %f is a Positive Number",num);
    return 0;
}
```



```
Enter any Number of your choice : 126
126.000000 is a Positive Number
```



if ()...else... Statement

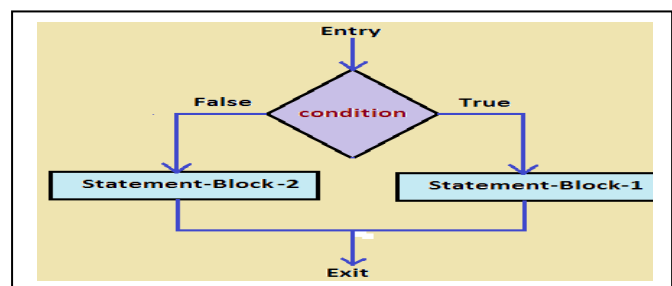
The if ()...else... statement is a bi-directional conditional control transfer construct, used to take decision on the given condition when one of the two possible options is to choose. The condition may be either relational or logical expression. If the condition will be TRUE, it will return the execution of statement-block-1; otherwise execute statement-block-2.

Syntax : **if (condition)**

Statement-block-1;

else

Statement-block-2;



//Program Name : testIfElse.c

//Program to demonstrate if()...else... construct

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

```
main()
```

```
{
```

```
    int num;
```

```
    printf("\nEnter any Number of your choice : ");
```

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

```
    if (num % 2 == 0)
```

```
        printf("\n\t%d is an Even Number",num);
```

```
    else
```

```
        printf("\n\t%d is an Odd Number",num);
```

```
    return 0;
```

```
}
```

```
Enter any Number of your choice : 37
```

```
37 is an Odd Number
```

```
-----  
Process exited after 6.111 seconds with return value 0  
Press any key to continue . . .
```

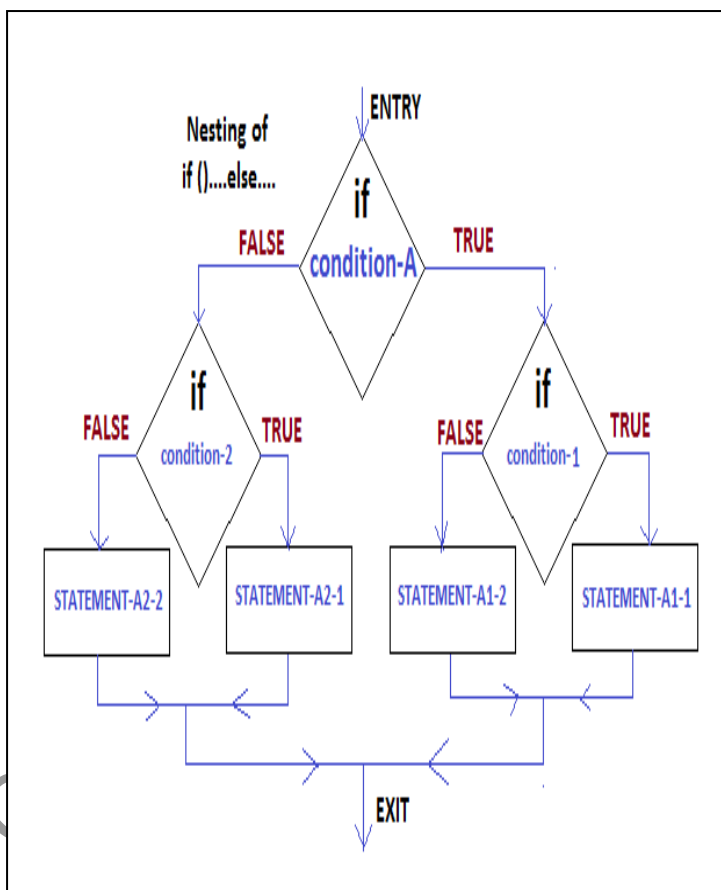
Nested if ()...else... Statement

Using the if()... else... statement in the if block or in the else block is referred as Nesting if... else... statements. nesting can be done in both if and else part. The outer if(condition) is tested first, on getting TRUE, the inner if(condition) is tested thereafter. On getting FALSE, the inner if(condition) in else part is executed. The construct looks like –

```

if (condition A)
{
    if (condition 1)
        statement-A1-1;
    else
        statement-A1-2;
}
else
{
    if(condition 2)
        statement-A2-1;
    else
        statement-A2-2;
}

```



// Program Name : nestedIf.c

// To demonstrate the use of Nested if()

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

```
int main()
```

```
{
```

```
    char grade;
```

```
    float prcnt;
```

```
    printf ("\n\nEnter Grade and Percentage : ");
```

```
    scanf ("%c %f",&grade,&prcnt);
```

```
    if (grade=='A' || grade == 'a')
```

```
        if(prcnt >= 95.0)
```

```
            printf ("\n\nExcellent\n\n");
```

```
        else
```

```
            printf ("\n\nWork Hard for Grade A\n\n");
```

```
    else
```

```

printf ("\n\nNot good ! Do Work more harder\n\n");
return 0;
}

```

Enter Grade and Percentage : A 93.57
Work Hard for Grade A

Enter Grade and Percentage : A 98.35
Excellent

Enter Grade and Percentage : B 85
Not good ! Do Work more harder

if ...elseif Ladder Statement

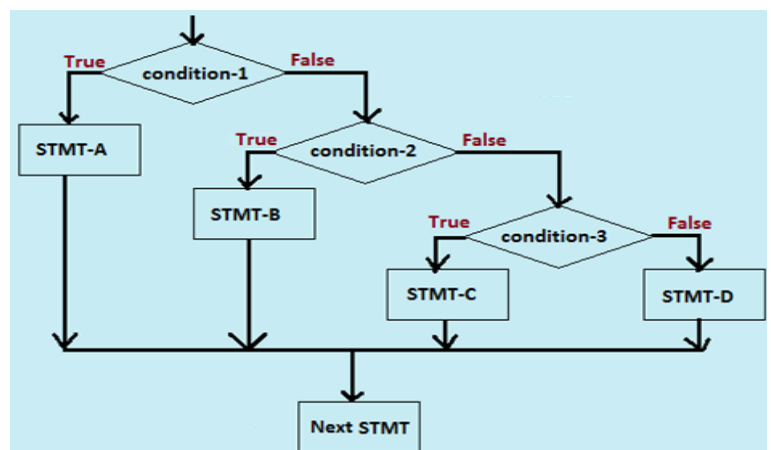
This is a type of nesting in which there is an if()... else... statement in every else part except the last else part. This type of nesting is frequently used in programs and is also known as if...elseif... ladder.

It is used when more than two conditions are to be tested to determine one of them to be TRUE. Only one condition will be TRUE and its related part will be executed, rest will be ignored, otherwise last else part will be executed.

```

if ( condition-1)
    statement-A;
else if (condition-2)
    statement-B;
else if ( condition-3)
    statement-C;
else
    statement-D;

```



//Program Name : ifLadder.c

/* Program to calculate percentage and Grade of Result

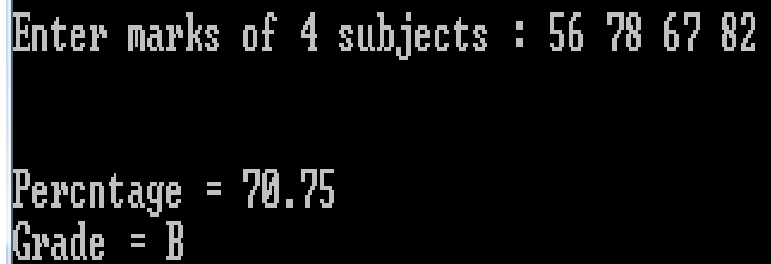
```

prcnt>=85 , Grade=A;
prcnt<85 and prcnt>=70 , Grade=B;
prcnt<70 and prcnt>=55 , Grade=C;
prcnt<55 and prcnt>=40 , Grade=D;
prcnt<40 , Grade=E; */

```

```
#include<stdio.h>

main()
{
    float m1, m2, m3, m4, total, prent;
    char grade;
    printf("\n\nEnter marks of 4 subjects : ");
    scanf("%f %f %f %f",&m1,&m2,&m3,&m4);
    total = m1+m2+m3+m4;
    prent = total/4;
    if (prent >= 85)
        grade = 'A';
    else if(prent >= 70)
        grade = 'B';
    else if (prent >= 55)
        grade = 'C';
    else if (prent >= 40)
        grade = 'D';
    else
        grade = 'E';
    printf ("\n\nPercentage = %.2f\nGrade = %c\n\n", prent, grade);
    return 0;
}
```



```
Enter marks of 4 subjects : 56 78 67 82

Percentage = 70.75
Grade = B
```

Q.8. Write down the general syntax of switch statement and explain with the help of 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)
```

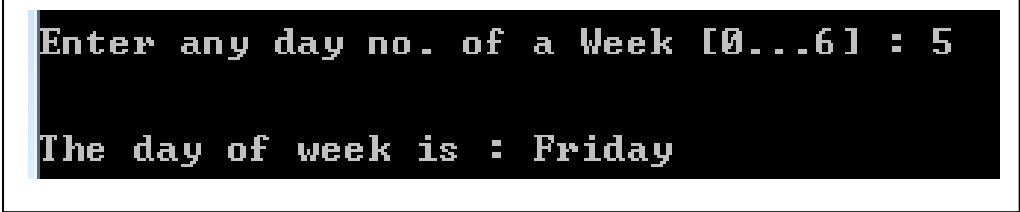
```
    {
```

```
        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.8. Write a C program to find square root of quadratic equation.

(6)

Ans. :

//Program to calculate Square 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)); // Calculating Discriminant

r1 = (-b + z)/(2*a); // Calculating First Root

r2 = (-b - z)/(2*a); // Calculating Second Root

if(d<0)

{

printf("\n\nBoth Roots are Imaginary and Unequal :\n");

printf("\n\nRoot #1 = %f",r1);

printf("\n\nRoot #2 = %f",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.9. Write a C program to print natural numbers from 1 to n.

(6)

Ans. :

// Program to print natural numbers from 1 to N

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

```
int main()
```

```
{
```

```
    int n, j;
```

```
    printf("\n\n Enter the last range of the natural no. to printf : ");
```

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

```
    printf("\n\n Natural Nos. from 1 to %d are :\n\n",n);
```

```
    for(j=1;j<=n;j++)
```

```
        printf("%5d",j);
```

```
    return 0;
```

```
}
```

Enter the last range of the natural no. to printf : 100

Natural 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												

Q.9. What do you mean by an algorithm? Write an algorithm of finding area of triangle.

(6)

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.

ALGORITHM Writing Terminology

To solve every problem, you can write the solution in stepwise logical order using some standard terminologies. These terminologies have specific meaning indicating action.

Terminals : Every Algorithm must have a START and a STOP step. START is always the first step whereas STOP is the last step.

Variable Declaration : Every Algorithmic solution needs to store and use data. Data are stored in variables. Variable needs to be introduced about its name and type of data to be stored in it. DECLARE word is used to do this task followed by Variable Name and Data Type.

Syntax : STEP n DECLARE var1, var2,... as DataType

Example : Step 3. Declare N, M As Integer

Step 4. Declare Z As Real

Step 5. Declare Name As String

Assignment : Every Algorithmic solution needs to consider some data to be stored to use to perform task. This action is called Assignment. Assignment is performed by putting a value in a named variable. LET or ASSIGN is used to perform this task.

Syntax : STEP n LET value to Var

STEP n ASSIGN value to Var

Example : STEP no. LET 25 to num

STEP no. ASSIGN 9.5 to wt

INPUT : Solution of your problem needs data to be used to process the task. Receiving data from external input source like Keyboard, Disk, or Mouse to specified memory variable is known as INPUT. Input is performed using READ statement.

Syntax : STEP n READ var1, var2, var3,

STEP n INPUT var1, var2, var3,

Example : STEP 4. READ Num

STEP 5. INPUT Length, Width

OUTPUT : After completing the task one or more results should be shown. Results are stored in Output Variables. Displaying the result on Output Device like Screen or Paper is known as OUTPUT. Output operation is shown by WRITE or PRINT statement.

Syntax : STEP n WRITE var1, var2, var3,

STEP n PRINT var1, var2, var3,

STEP n DISPLAY var1, var2, var3,

Example : STEP 4. PRINT "SUM = ", S

STEP 5. WRITE “Your age is “, Age

Decision Making Statements

In some cases it is needed to take decision among two or more options. At that situation Decision Making construct is used. An expression is checked logically to get one of the True or False output, depending upon which control transfer operation performed.

To perform Decision Making operation IF..., IF....ELSE.... or IF....ELSEIF.... construct is used.

IF...THEN... : IF construct uses Logical or Relational expression to test the condition either True or False depending upon which, the control jumps either inside the body of construct or outside the construct. If condition becomes TRUE, the control enters into the body of the construct. If found FALSE, the control jumps at end of the IF construct, i.e., next statement to the body of the construct.

Syntax : IF <condition> THEN

Statements

ENDIF

Example : IF A is > than B THEN

Assign A to Big

ENDIF

IF...ELSE.... : IF...ELSE... construct uses Logical or Relational expression to test the condition either True or False depending upon which, the control jumps either inside the body of construct or outside the construct. If condition becomes TRUE, the control enters into the body of THEN part of the construct. If found FALSE, the control enters into the body of ELSE part of the construct.

Syntax :

IF <condition> THEN

statements

ELSE

statements

ENDIF

Example :

IF A is less than B THEN

Assign A to Big

ELSE

Assign B to Big

ENDIF

Looping Statements

FOR Loop : FOR Looping construct consists of FOR , TO and STEP key terms. In FOR Loop, Initialization, Condition and Change in Value of looping variable are specified on same line. Loop ends with END FOR key term.

Syntax(वाक्यविन्यास) :

```
FOR var = InitVal TO LastVal [ STEP CngVal ]
    _____
    _____
    _____
END FOR
```

Example :

```
FOR N = 1 TO 10 STEP 2
    WRITE N
END FOR
```

Output :

2 4 6 8

WHILE Loop : WHILE Looping construct consists of WHILE and DO key terms. In WHILE Loop, variable is Initialized before starting the loop. WHILE term is applied followed by a logical or relational expression to test the condition. If condition becomes True, control enters into body of loop. On getting False condition, the loop terminates and control jumps at END WHILE term.

Syntax(वाक्यविन्यास) :

```
Initialize var = InitVal
WHILE expression
    _____
    _____
    _____
END WHILE
```

} Body of Loop

Example :

```
LET N = 1
WHILE N ≤ 10
    WRITE N
    Increment N by 1
END WHILE
```

Output :

1
2

9
10

DO Loop : DO Looping construct consists of DO and END DO key term. In DO Loop, variable is Initialized before starting the loop. Condition is checked at the end of the loop using WHILE Term followed by a Logical or Relational expression. The control enters into the body of the loop initially without any hurdle. At end if condition becomes True, control returns back at DO loop for next pass. On getting False condition, the loop terminates and control jumps out of DO Looping construct.

Syntax(वाक्यविन्यास) :

```
Initialize var = InitVal
DO
    _____
    _____
    _____
WHILE expression
END DO
```

} Body of Loop

Example :

```
LET N = 1
DO
    WRITE N
    Increment N by 1
WHILE N ≤ 10
END DO
```

Output :

1
2

9
10

Algorithm

Step – 01 REM Algorithm to find Area of triangle

Step – 02 Declare A, B, C, S, AREA as Real

- Step – 03 Read A, B, C
- Step – 04 Calculate $S = (A + B + C)/2$
- Step – 05 Calculate $AREA = \text{SQRT}(S * (S-A) * (S-B) * (S-C))$
- Step – 06 Print “Area of Triangle = “, AREA
- Step – 07 Stop

Q.10. Explain different types of loop control structures used in C language.

(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

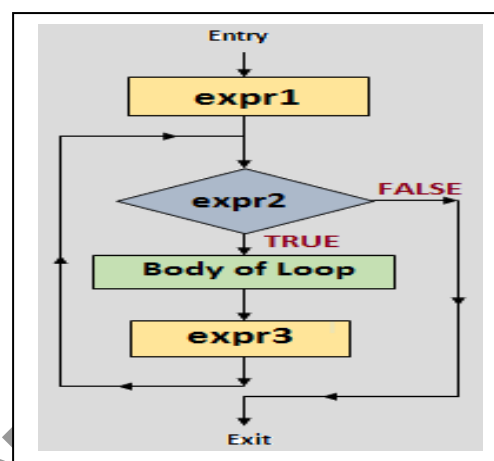
In the syntax of **for()** loop, **expr1** is used to initialise some parameters (called an index) that controls the looping action, **expr2** represents a condition that must be satisfied for the loop to continue execution, and **expr3** is used to alter the value of the parameter initially assigned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



Some More Forms of for () loop structure :

Syntax :

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

Example Code :

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

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


while() Statement

The **while()** construct is an **Entry-Controlled** loop statement. **While()** loop is used to carry out looping operations best suited for problems where it is not known in advance that how many times a statement or a statement-block will be executed. In the construct of **while()** statement, a loop-control variable is initialized, then the conditional expression is tested. On getting **TRUE**, the control enters into the body of loop to execute statement block. The update expression of loop-control variable is placed inside the body of the loop. This statement(s) are executed repeatedly the single or compound statement(s) until the condition is **False**. On getting **FALSE**, the control jumps on to the next statement outside the loop.

The general form of the **while()** statement is :--

Syntax :

```

expr1 // Var = initValue;
while ( expr2 ) // Condition Testing
{
    Statement-Block;
    expr3; // Update Expression
}

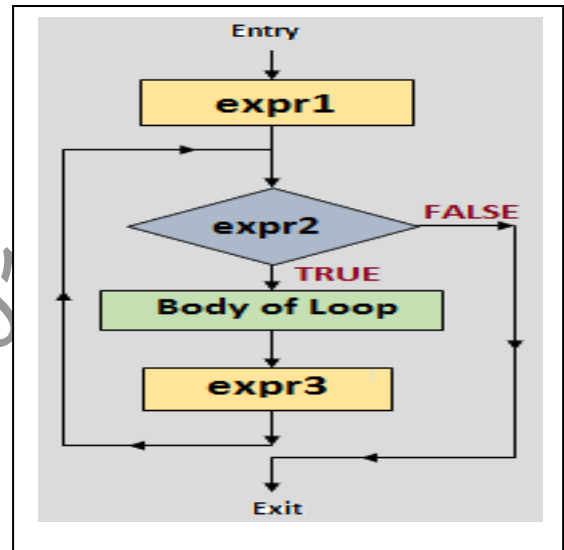
```

Example :

```

int    n=1;
while ( n<=100)
{
    printf(“%5d”, n);
    n++;
}

```

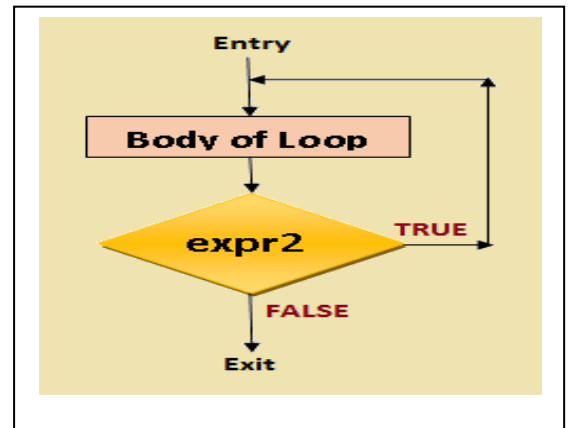
**do ... while() Statement**

do...while() is an **Exit-Controlled** Looping construct. In **do ... while ()** statement, it is not known in advance that how many times a statement will be executed. It is known that the statement will be executed at least once. In that case, there is no need to test the condition at the beginning. The condition is tested at the end followed by **semicolon (;)** to determine that should the body of the loop be executed again or not. On getting **TRUE** result of condition, control goes back to execute next pass to the body of loop. On getting **FALSE**, the loop terminates. The loop control variable must be initialized before initiating loop. The update expression to modify the value of loop control variable is placed inside the body of loop.

Syntax :

```

expr1; //Initialize Loop variable
do
{
    Statement-Block;
    expr3; // Looping variable update expression
} while ( expr2 ); // testing the condition
  
```



where **expr2** is a constant, a variable or an expression. The statement-block is executed repeatedly till the expression evaluates to a non-zero (True) value.

Example. :

```

int n = 1;
do
{
    printf("%5d", n);
    ++n;
} while (n<=100);
  
```

Q.10. Write a C program for addition of two matrices.

(6)

Ans. :

//Addition of 2 Arrays

#include<stdio.h>

#define MAX 100

int main()

{

int a[MAX][MAX], b[MAX][MAX];

int s[MAX][MAX], i,j,r,c;

printf("\nEnter Dimension of Matrix : ");

scanf("%d %d",&r,&c);

//Input in Matrix

```

printf("\nEnter %d numbers in 1st Matrix : ",r*c);
for(i=0;i<r;i++)
    for(j=0;j<c;j++)
        scanf("%d",&a[i][j]);
printf("\nEnter %d numbers in 2nd Matrix : ",r*c);
for(i=0;i<r;i++)
    for(j=0;j<c;j++)
        scanf("%d",&b[i][j]);
//Adding two matrices
for(i=0;i<r;i++)
    for(j=0;j<c;j++)
        s[i][j]=a[i][j] + b[i][j];
//Displaying Output
printf("\n\n1st Matrix A[%d][%d]:\n",r,c);
for(i=0;i<r;i++)
{
    for(j=0;j<c;j++)
        printf("%5d",a[i][j]);
    printf("\n");
}
printf("\n\n2nd Matrix B[%d][%d]:\n",r,c);
for(i=0;i<r;i++)
{
    for(j=0;j<c;j++)
        printf("%5d",b[i][j]);
    printf("\n");
}
printf("\n\nResultant Matrix S[%d][%d]:\n",r,c);
for(i=0;i<r;i++)
{

```

```

Enter Dimension of Matrix : 3 2
Enter 6 numbers in 1st Matrix : 10 20 30 40 50 60
Enter 6 numbers in 2nd Matrix : 11 22 33 44 55 66

1st Matrix A[3][2]:
 10  20
 30  40
 50  60

2nd Matrix B[3][2]:
 11  22
 33  44
 55  66

Resultant Matrix S[3][2]:
 21  42
 63  84
105 126

```

```
    for(j=0;j<c;j++)  
        printf("%5d",s[i][j]);  
    printf("\n");  
}  
return 0;  
}
```