DATE 16/8/2019

Pseudo code for the grade of the student compute the average & find the %

Step1 START
Step2 Read marks m_1, m_2, m_3, m_4
Step3 percentage = $\frac{m_1+m_2+m_3+m_4}{4}$
Step4 if ($P \geq 90$), print "Grade O"
else if ($P \geq 80$)
then print "Grade E"
else if ($P \geq 70$)
then print "Grade A"
else if ($P \geq 60$)
then print "Grade B"
else ~~if~~
print "Fail"

Pseudo code for leap year

Step1 START
Step2 Enter a year, y
Step3 ~~if (y % 4 == 0 &~~
if (y is divisible by 4 and not divisible by 100)
print y is leap year

else if (y is divisible by 400)
print y is leap year

else
print y is not leap year

Pseudo code for GCD of 2 nos

Step-1 START
Step2 input x, y
Step3 if (y=0) ——— ①
print GCD is x ~~Goto~~ ⊛
else if (x>y)
x = x-y and Goto ①

else
y = y-x and Goto ①

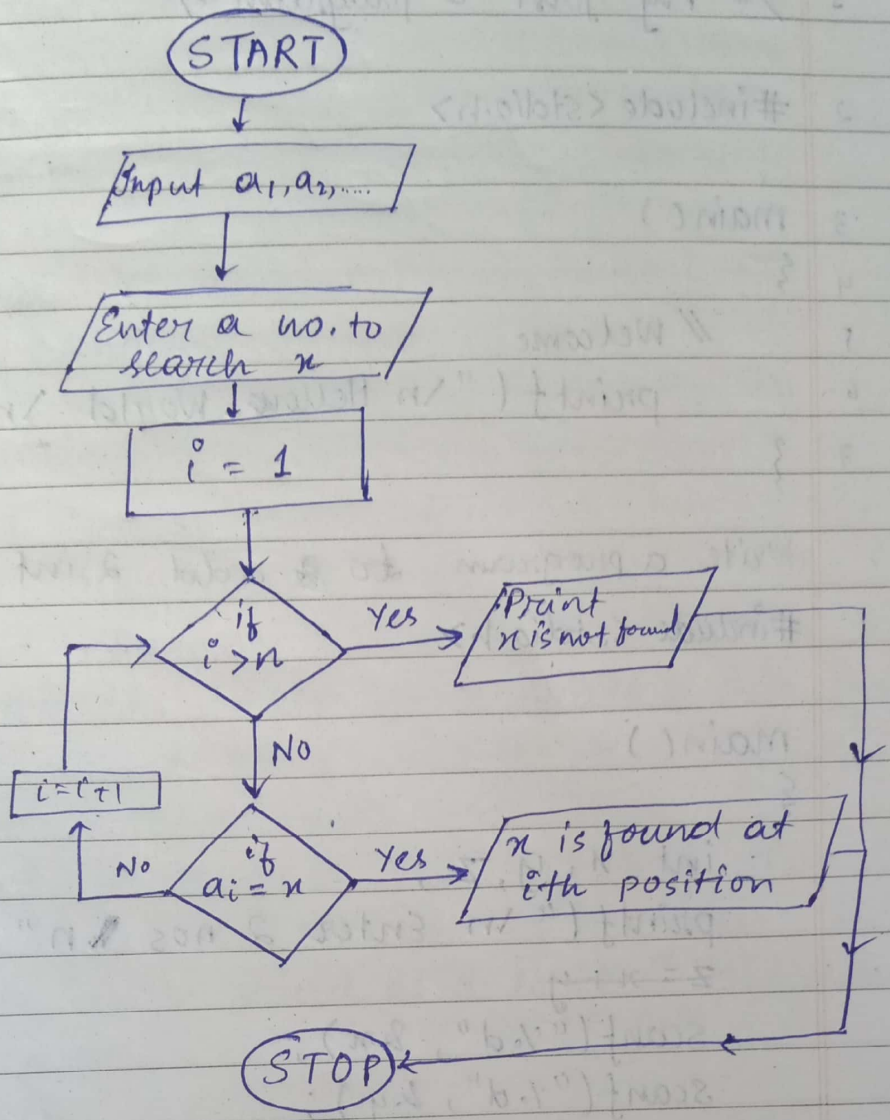
⊛STOP

Pseudo code for factorial of a number

Step-1 Start
Step-2 input a number, n
Step-3 i = 1, F = 1
Step-4 if (i > N)
print factorial = F & Goto ⊛
~~else if~~ (
F = F*i and i = i+1 & Goto ④

⊛ STOP

Draw the flow chart for linear search :



Write pseudo code for linear search

Step1 START
Step2 Input $a_1, a_2, \ldots, a_n$
Step3 Enter a no. to search, x
Step4 initialise i = 1
Step5 if (i > n)
Print x is not found

else if ($a_i = x$)
print 'x' is found

else
i = i+1 and goto 5

Teacher's Signature

PAGE NO. }
DATE 20/8/2019

```
1  /* My first C program */
2  #include <stdio.h>
3  main()
4  {
5      // Welcome
6      printf("\n Hello World \n");
7  }
```

Write a program to add 2 int

```
#include <stdio.h>

main()
{
    int x, y, z;
    printf("\n Enter 2 nos \n");
    z = x + y
    scanf("%d", &x);
    scanf("%d", &y);
    z = x + y;
    printf("Total = %d", z);
}
```

PAGE NO.
DATE 21 / 8 / 2019

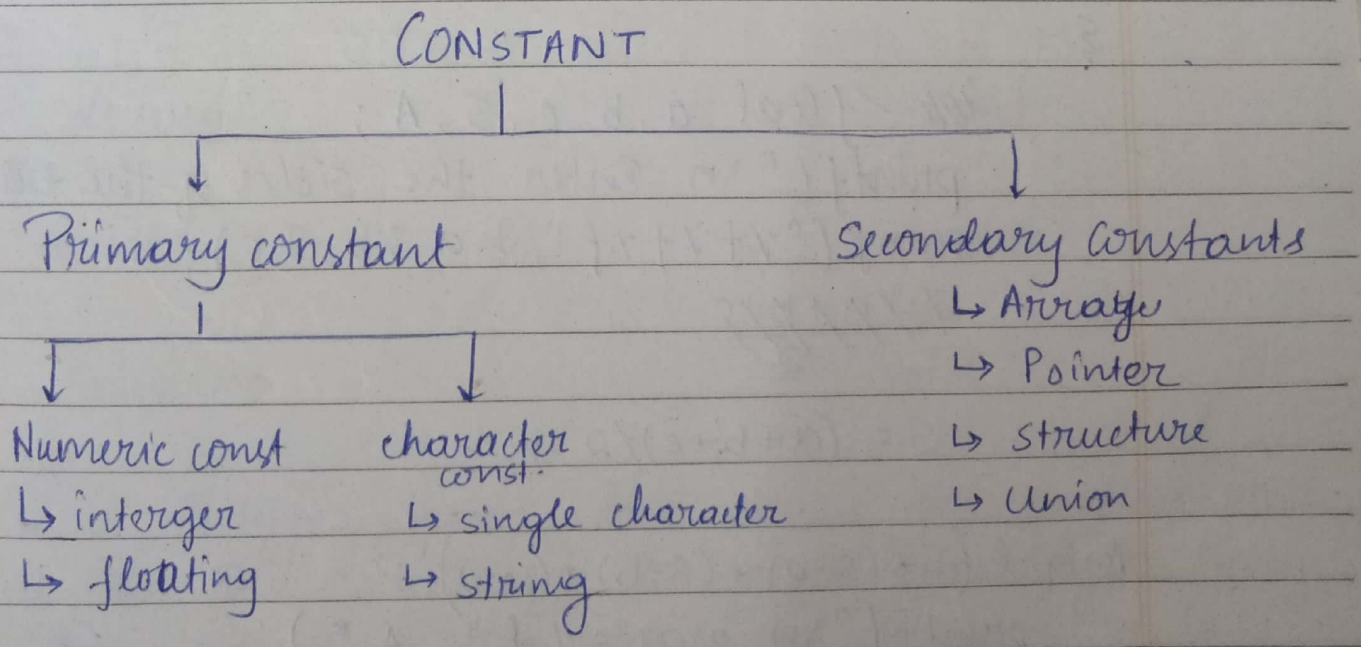
The basic building blocks of C programming language are called 'tokens'.

C - Tokens

1. Constant
2. Keyword
3. Name / identifier
4. Operators
5. strings
6. Special Symbols

Keyword - There are 32 Keyword in ANSI C & C89 These are the words whose meaning has already been explained to C compiler. e.g. do, while, switch, if, for, return, void, auto

Constant - A constant is a quantity that doesnt change during the execution of program. This quantity can be storeid at a locat" in the memory of computer.



The range of integer in a 16-bit comp is -2^{15} to $2^{15}-1$

~~count~~ cout cin

PAGE NO.
DATE : / / 20

Identifier - variables or names or identifier is a name that may be used to store a data value, function name etc.
A variable is a combinat" of alphabets, digits or underscore.

The first character of the identifier must be a name or underscore.
No comma or blanks within a variable name
Variable or identifier name shouldn't be a keyword

Compute the area of Δ using Heron's Rule

$$S = \frac{a+b+c}{2}$$

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

```
#include <stdio.h>
#include <math.h>
void main()
{
    float a,b,c,S,A;
    printf("\n Enter the sides of the triangle\n");
    scanf("%f %f %f",&a,&b,&c);
    S=(a+b+c)/2;

    S = (a+b+c)/2 ;

A=sqrt(S*(S-a)*(S-b)*(S-c));
  printf("\n area = %f", A*);
}
```

$(a+b+c).\frac{1}{2}$

~~Shashank I am a goo~~

Basic Structure of C Program

Documentation Section
Link section
Definition section
Global Declaration

main( )
{
 	
}

Sub Program Section

Note → Before using any variable we must declare them

DATA TYPES -

ANSI C and C89 defined 5 fundamental data types. Those are :-

~~char~~

	Type	declared as
1.	character	char
2.	integer	int
3.	decimal	float
4.	double floating point	double
5	Valueless	void

These 5 types form the basis for several other types. The size and range of these types vary from processor to processor & compiler to compiler

PAGE NO.
DATE: / / 20

12h
or
256

MODIFIERS -

Except the type void the basic datatype may have various modifiers preceeding them A type modifier alters the meaning of the base type to more precisely fit to a specific value or need.

The list of modifiers are :-

(I) signed
(II) unsigned
(III) long
(IV) short

The type void is either used as a returntype of a function which returns nothing or to create a generic pointer.

for 16-bit i.e., n = 16

Format	Data type	Range (-2^{n-1} to $2^{n-1}-1$)	Byte
%c	char	-128 to 127	1
%c	signed char	-128 to 127	1
%c	unsigned char	0 to 255	1
%d	int	-32768 to 32767	2
%d	signed int	-32768 to 32767	2
%u	unsigned int	0 to 65535	2
%u	longunsigned int	0 to 2147483648	4
%d	short signed int		

Teacher's Signature

for unsigned range : 0 to $2^n - 1$

%u	short unsigned int	
%ld	long signed int	4
%f	float	-3.4e38 to 3.4e38 4
%lf	double	8
%Lf	long double	10

Dt - 27.8.19

Conversion from Decimal to Binary

{0, 1}

$(815)_{10} = (1100101111)_2$

2	815	
2	407	1
2	203	1
2	101	1
2	50	1
2	25	0
2	12	1
2	6	0
2	3	0
2	1	1
	0	1

1 1 0 0 1 0 1 1 1 1

2^9 2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

$1\times2^0 + 1\times2^1 + 1\times2^2 + 1\times2^3 + 0\times2^4 + 1\times2^5 + 0\times2^6 \times 0\times2^7 \times 1\times2^8 + 1\times2^9 = (815)_{10}$

Conversion from Binary to Decimal

Octal-integer or Octal integer constant

{0, 1, 2, 3, 4, 5, 6, 7}

An octal integer const. is any combination of digits from 0 to 7 with a leading 0
e.g., 037, 0156, 0677, 0012, 0
[no 8, 9]

Conversion from Octal to Decimal

$(037)_8 \approx 8^0 \times 7 + 8^1 \times 3 = 7 + 24 = (31)_{10}$

$(0156)_8 \approx 8^0 \times 6 + 8^1 \times 5 + 8^2 \times 1 = 6 + 40 + 64$
$= (110)_{10}$

Hexa-decimal integer constant

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
it always written as 0x or 0X
e.g, 0X2 ; 0x9F , 0x910AC

$(0x9F)_{16} \approx 16^0 \times 15 + 16^1 \times 9 = 15 + 144 = (159)_{10}$

Swap 2 nos without using 3rd variable.

```
x = x+y;
y = x-y;
x = x-y;
```

/* ~~Computi~~ swapping two nos */

```
#include<stdio.h>
void main()
{
    int x,y;
    printf("\n Enter 2 nos \n");
    scanf("%d%d", &x, &y);
    x = x+y;
    y = x-y;
    x = x-y;
    printf("\n the values of x= %d", x);
    printf("\n the value of y = %d", y);
}
```

Assignment-1

Write a short note on escape sequences in C
e.g., (\n, \t, \a, ...)

DATE 28/8/2019

Expression

Operators, constants, functions, variables are the constituents of C expression.
An expression in C is any valid combinat" of these element.

Operators

These are the symbols which specify a particular operation on data item.
The data item on which operation is applied are called operands.
Operators can be classified as:-
According to the type of operation they perform they are classified as:-

1. Arithmatic Operators (+, -, *, /, %.)
2. Relational Operators (<, >, ==, >=, <=)
3. Logical " (&&, ||, !)
4. Assignment " (=)
5. Conditional " (? :)
6. Bitwise " (<<, >>, &, ~, |, ^)
7. increament & decreament operator (++, --)
8. Special operators (, , [ ], ( ), ...)

ARITHMATIC OPERATORS

+	addition, Unary plus
-	subtraction, Unary minus
*	multiplication
/	division (quotient)
%	remainder

Note → integer division truncates the fractional part
In " " if both the operands are of same sign then the result truncated towards zero.

if one of them is -ve, the direct" of truncation is implementation dependent.

6/7 = 0

but -6/7 = 0 or -1 → depends on compiler

Note→ modulo division (%) always provides the remainder. It can only be applied on integers i.e., floating point operands cannot be used.

The sign of the result is always the sign of 1st operand. e.g., -25%6 = -1

25%-6 = 1

WAP to check the sign of modulo division

WAP to compute the sum of the digits of a 3-digit number.

```
#include <stdio.h>
void main()
{
    int x, O, T, H, sum;
    printf("\n Enter a 3-digit number \n");
    scanf("%d", &x);
    O = x%10;
    x = x/10;
    T = x%10;
    x = x/10;
    H = x;
    sum = H+O+T;
    printf("\n sum = %d", sum);
}
```

WAP to print the output in ~~year~~, months, days if the input is in days

e.g., 366 days = 12 months 1 day

= → assignment
== → relational operator

PAGE NO:
DATE: / /20

```
#include<stdio.h>
void main()
{
   int x, M, D;
   printf("Enter no. of days \n");
   scanf("%d", &x);

   M = x/30;
   D = x%30;
   printf("month = Days = %d %d", M, D);
}
```

409/30 13

Dt - 29.8.19

Relational Operators

The comparisions bet" 2 data items can be done in C using relational operator which are
<, >, <=, >=, ==, !=
The value of relational operations are evaluated to either true or false

```
int main()
{
    int x;
    x = (4<=25);
    printf("%d", x);
}
```

→ if true x=1
if false x=0

Logical Operators

Logical operator combine two or more relational expression to form compound relational expression
&&, ||, !

Assignment operator (=)

Assignment operator is used to assign the result of an expression to a variable. It is computed from right to left.

C has a set of short hand assignment operator.

var op = exp;

or

var = var op exp;

1. It is easier to write or read.
2. It is more concise.
3. It is more efficient than the general statement bcoz the variable is calculate only ones.

Dt - 30.8.19

Increament and Decreament Operation

A special type of operator which can be used to increase or decrease the value of the operand by one. Increament & decreament are unary operator. Both the increament & decreament operator may either preceed the operand or follow the operand.

The difference betn Prefix & Postfix

* The prefix increament returns the value of variable after it has been increamented	* The postfix increament returns the value of variable before it has been ~~imp~~ increamented.

Ex.

```
int main()
{
    int x = 6, y = 8, z = 7;
    z = (x++) + (y + 10);
    printf("%d", z);
    printf("%d", x)
```

$z = 6 + 18$
$= 24$

$z = 6 + 9 - 10$
$z = 5$
$x = 7$

Output
24 7

Conditional Operator –
It is a terminary operator.

(expression1) ? (expression2) : (expression3);

e.g., `x = (y<10) ? 10 : 20;`

if true → x = 10
if false → x = 20

e.g,

```
int main()
{
    int x, y, z;
    scanf("%d%d", &x, &y);
    z = (x>y) ? x : y;
    printf("the bigger number is = %d", z);
    return 0;
}
```

WAP to compute greatest among 3 nos using conditional operator

```
int main()
{
    int x, y, z, a;
    scanf("%d%d%d", &x, &y, &z);
```


Teacher's Signature

PAGE NO.
DATE 3/9/2019

Bitwise Operators

Bitwise operators operates on variable or constants bitwise i.e., the value of operand is first converted to binary form, then the operation is carried out.

These operators may not be applied to float or double.

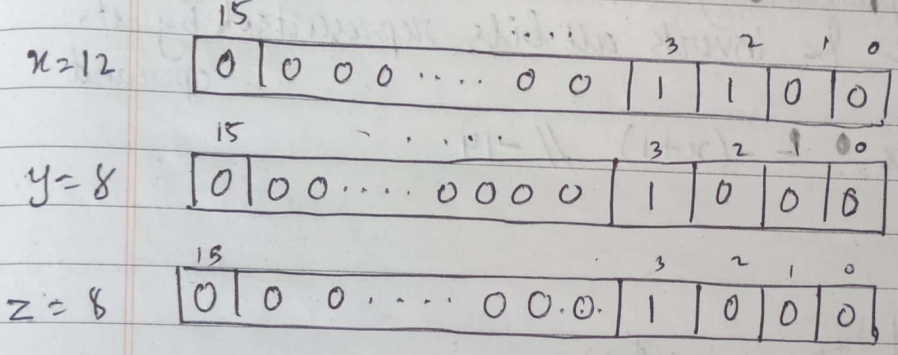
Bitwise (AND) &

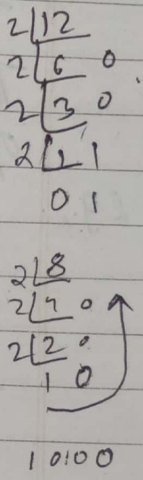
The result of bitwise AND is one if the both the corresponding bits are one otherwise zero.

e.g.,

$x = 12$, $y = 8$

$z = x \& y;$





Bitwise (OR) |

The resultant bit will be 1 if atleast one corresponding bit is 1

```
z = x | y
printf("%d", z); // 12
```

Bitwise Exclusive OR (XOR) ^

The result of XOR operation is 1 if only one bit is one.

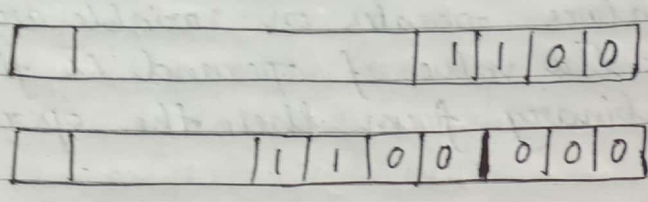
$$1\ 0\ 0 \quad (2^2\ 2^1\ 2^0)$$

$$2^0 \times 0 + 2^1 \times 0 + 2^2 \times 1 = 4$$

$2^5 + 2^6$

32
+64
96

Left Shift (<<)



$z = x << 3$

In general

$x << n \equiv x \times 2^n$

$12 << 3 \equiv 12 \times 2^3 = 96$

Right shift (>>)

$x >> n \equiv x / 2^n$

2|12
2|6 0
2|3 0
1 1

Bitwise NOT (~)

The complementary operator is a unary operator & inverts all bits represented by its operand

e.g., $x = 13$

$y = \sim x; \quad = -(x+1)$ // -14

sizeof

```
float y;
printf("%u", sizeof(y));
```

sizeof is a compile time operator and when used with an operand it returns the number of bytes the operand occupies in the memory. The operand may be a variable, const or a datatype

a = (x>y && x>z)

a = (x>y) ? x : y;

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

Dt - 4.9.19

Comma Operator

The comma operator can be used to link the related expression together.
A comma linked list of operators are evaluated from left to right.
The value of right most operator is the value at the combined operator.

```
int main()
{
   int x = (2, 3, 4);
    printf("%d", x); // 4
}
```

Type cast Operator

```
int main()
{
    int x=5;
    float y = (float) x;
}
```

Type conversion

During the evaluation of expression involved with different type of data at assignment operation

Teacher's Signature

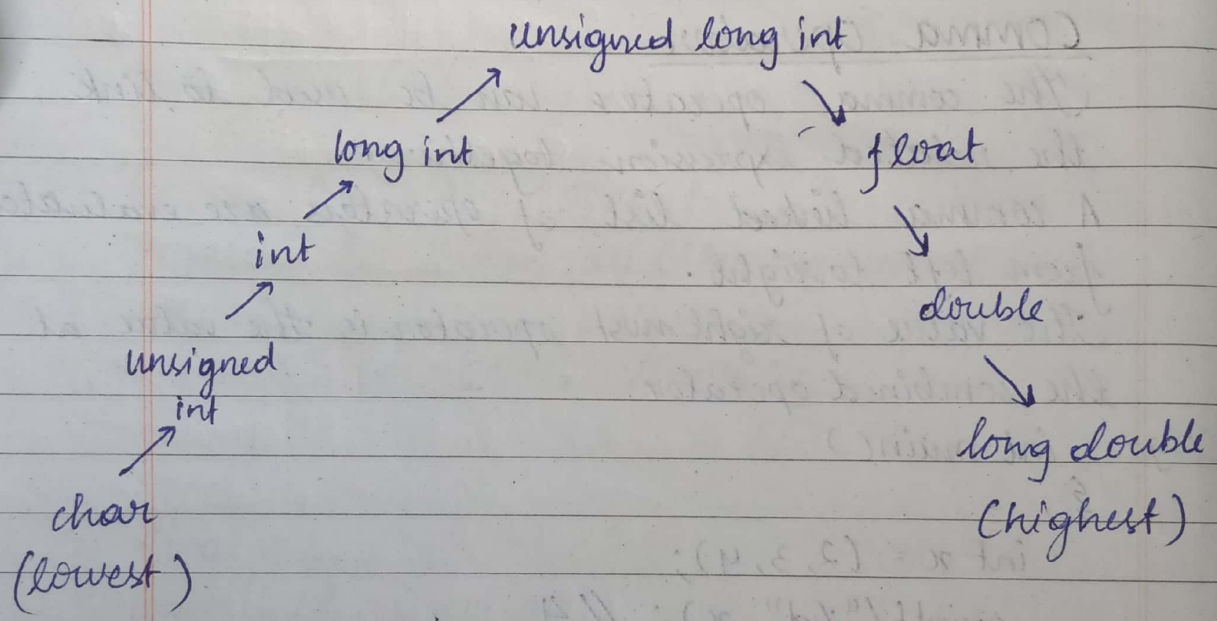
C - allows us to mix constants & variables of different types in an expression. C automatically converts any intermediate values to proper type so that it can be evaluated without losing any significance. This automatic conversion is known as Implicit type conversion

variable of one data type is converted to other. This process is called Type conversion or type casting.

It is of 2 types :-

① implicit type conversion
② explicit " "

This conversion takes place automatically in the following order :-




```
int K
float a;
K = 2/9 ; //0
K = 2.0/9; //0
a = 2/9;  //0.000...
a = 2.0/9; //0.222...
```

The type cast operator is used for explicit type conversion. To overcome automatic implicit conversion.

n/

Ex-1 Given days, write the output in Year, Month, days:-

```
int main()
{
    int days, Y, M, D, Days;
    scanf("%d", &days);
    Days = days;
    Y = days / 365;
    days = days %365;
    M = days /30;
    D = days %30;
    print("%days = %d years  %d months  %days", Days
          Y, M, D);
    return 0;
}
```

Ex- rectangle cartesian system to polar cartesian :-

```
float x, y;
r = sqrt(x*x + y*y);
θ = tan⁻¹(y/x)
```

⟶

```
float r, θ, x, y;
r = sqrt(x*x + y*y);
θ = atan(y/x);
```

polar cartesian system to rectangle cartesian sys

```
float r, θ;
x = r cosθ
y = r sinθ

rad = (3.142/180)*degree;
```

an

```
printf(x=100<50); // 0
```

97 → a z → 122
65 → A

```
char c='5';
printf("%d",'a'); //97
printf("%d",'0'); //48
printf("%c",'a'-'0'); //
printf("%d",c); //53
printf("%c",c); //5
printf("%d",c-'0');
```

as c='5'

48 → zero

Decision making / Control Statement

1. **if Statement**

```
if (condition)
{

}
```

WAP to check whether a no. is more than 100 or not

```
int main()
{
    int n;
    scanf("%d",&n);
    if (n>100)
    {
      printf("n is greater than 100);
    }

}
```

WAP to check the entered no. is even or odd

```
int main()
{
    int n;
    printf("Enter a number");
    scanf("%d", &n);
    if(n%2==0)
    {
        printf("n is even");
    }
    else
    {
        printf("n is odd");
    }
    return 0;
}
```

WAP if else to check the given year is leapyear

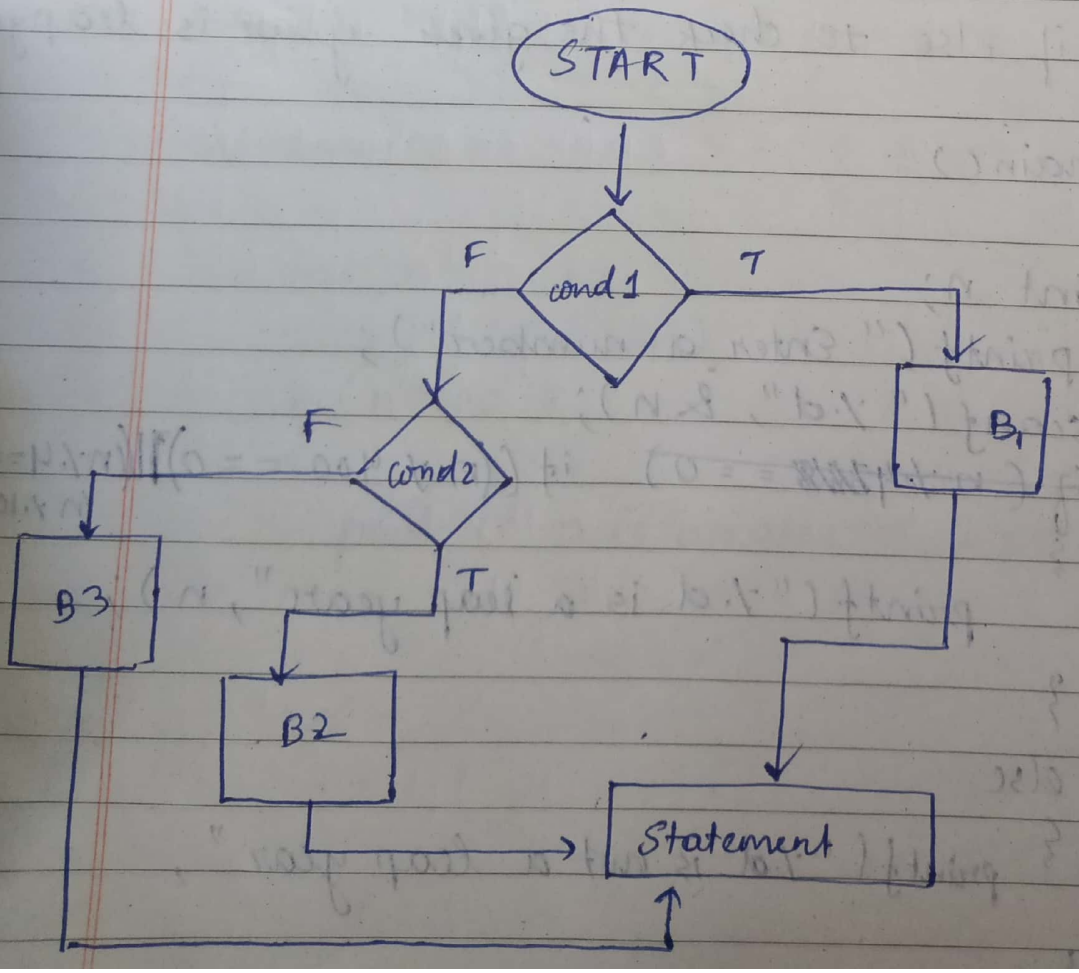
```
int main()
{
    int n;
    printf("Enter a number");
    scanf("%d", &n);
    if((n%400==0)||((n%4==0)&&(n%100!=0)))
    {
        printf("%d is a leap year", n);
    }
    else
    {
        printf("%d is not a leap year",
    }
    return 0;
}
```

Teacher's Signature

PAGE NO.
DATE : 11/9/2019

if..... else ladder

```
if (condition1)
{

}
else if ( condition2)
{

}
else if ( condition3)
{

}
else
{

}
```



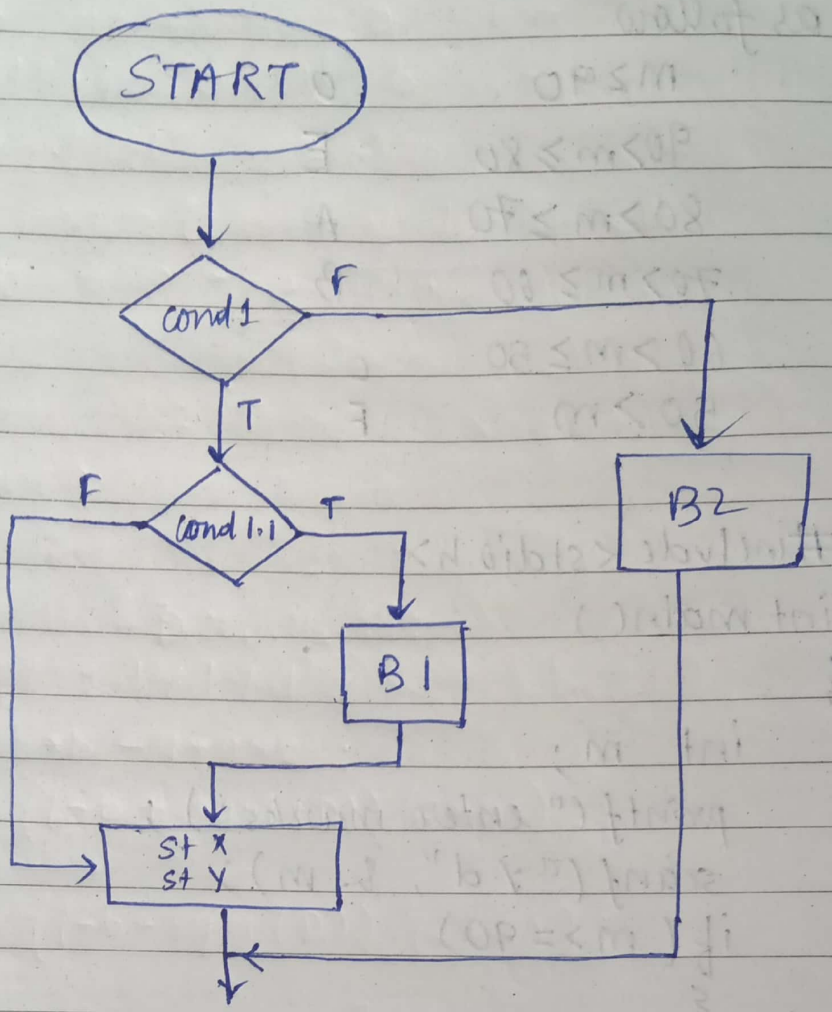
WAP to accept the marks and print the grade as follow

$m \geq 90$	O
$90 > m \geq 80$	E
$80 > m \geq 70$	A
$70 > m \geq 60$	B
$60 > m \geq 50$	C
$50 > m$	F

```
#include <stdio.h>
int main()
{
    int m;
    printf("enter marks");
    scanf("%d", &m);
    if(m>=90)
    {
        scanf" printf("Grade = O", m);
    }
   else if(m>=80 && m<90)
        printf("Grade = E");

   else if(m>=70 && m<80)
        printf("Grade = A");

   else if(m>=60 && m<70)
        printf("Grade = B");

   else if(m>=50 && m<60)
        printf("Grade = C");

   else
      printf("Grade = F");
  return 0;
}
```

Teacher's Signature

Nested if




```
if ( cond 1)
{
    if (cond 1.1)
    {
        B1
    }
    st . x
    st . y
}
else
{
    B2
}
```

WAP to find the greatest among 3 numbers using nested if ... else state

```
if (a>b)
{
    if (a>c)
        printf ("%d is greatest", a);
    else
        printf ("%d is greatest", c);
}
else if (b>c)
{
    if (b>a)
        printf ("%d is greatest", b);
    else
        printf ("%d is greatest", a);
}
else
    printf ("%d is greatest", c);
```

WAP to compute whether a given year is leap year or not.

PAGE NO.
DATE 12/9/2019

Q. WAP to compute the real roots of a quadratic equation $ax^2+bx+c=0$ when

(i) No solutn if a and b is zero
(ii) Only one solutn, if $a=0$
(iii) There is no real root if $b^2-4ac<0$
(iv) otherwise $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$

```
int main()
{
    int a, b, c, D, r1, r2, r;
    scanf("%d %d %d", &a, &b, &c);
    D = b*b - 4*a*c;
    r1 = ((-b + sqrt(D))/(2*a));
    r2 = ((-b - sqrt(D))/(2*a));
    if (a==0 && b==0)
        printf("no solution");
    else if (a==0)
    {
        printf("Only one solution");
        r = -c/b;
        printf("%d", r);
    }
    else if (D<0)
        printf("There is no real root");
    else
    {
        printf("the solutions are %d %d", r1, r2);
    }
    return 0;
}
```

Teacher's Signature

Q WAP to check whether a 3-digit number is palindrome or not

```
int main()
{
    int a, r, s
```

Switch .... case

```
switch (integer expression)
{
  case const1:
        Statement
        Statement
        break;

  case const2: ___
               ___
        break;

  :
  :

  default: ___
           ___
           ___

}
```

Check a number even or odd using switch case :-

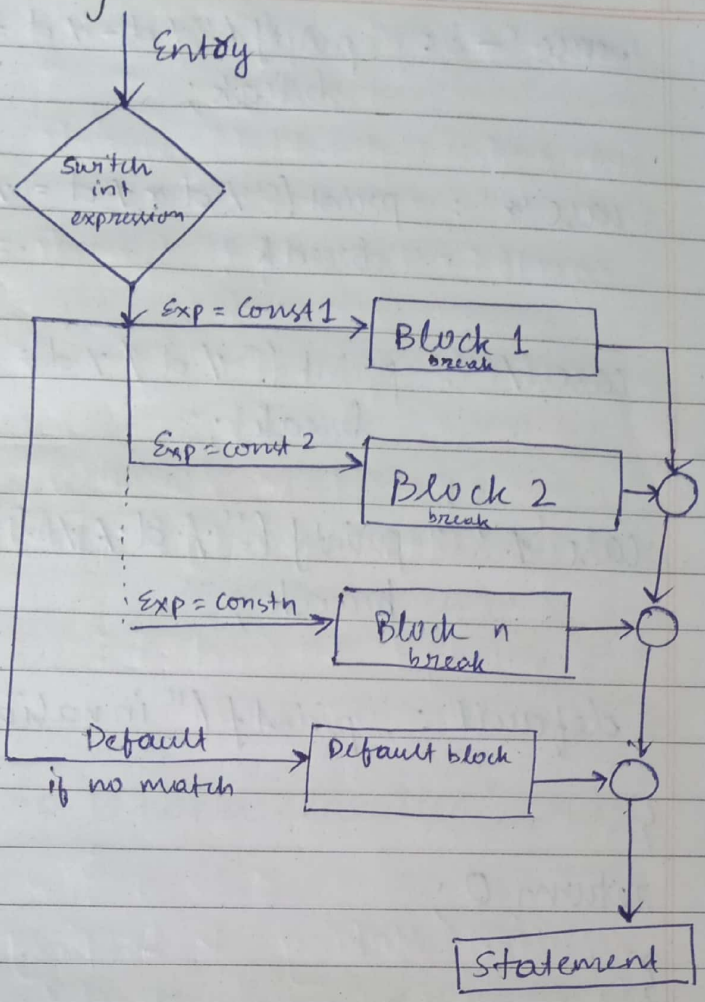
```
int main()
{
    int n, r ;
    scanf ("%d", &n) ;
    r = n %. 2 ;
    switch (r)
    {
      case 0 :  printf ("\n Even no.") ;
                break ;

      case 1 :  printf ("\n Odd no.") ;
                break ;

    }
    return 0 ;
}
```

Rules :-

1. The switch expression must be integer or character expression.
2. Constant used at the case level must be either an integral constant or constant expression
3. Each constant used in the case should be unique.
4. The number of statement within a case may be zero or more.
5. The default in switch case is optional & atmost one.
6. Cases can be arranged in any order.

PAGE NO.
DATE 13 / 9 / 2019

Flow diagram for switch case



WAP to take two numbers and an operator & execute the operation according to the inputted operator.

```
int main()
{
    int x, y;
    char op;
    scanf("%d %d", &x, &y);
    printf("Enter operator");
    scanf("%c", &op);
    switch(op)
    {
        case '+': printf("\n %d + %d = %d", x, y, x+y)
                  break;
```

(use space here " %c")

A = 65
Z = 97

PAGE NO.
DATE : / /20

```
case '-' : printf("%d - %d = %d", x, y, x-y);
           break;

case '*' : print("%d * %d = %d", x, y, x*y);
           break;

case '/' : print("%d / %d = %d", x, y, x/y);
           break;

case '%' : printf("%d %% %d = %d", x, y, x%y)
           break;

default : printf("invalid operator");
}
return 0;
}
```

WAP to check whether inputted character is consonant or vowels

ch = toupper(ch)

```
int main()
{
   char ch;
   scanf("%c", &ch);
   if((ch>='A' && ch<='Z') || (ch>='a' && ch<='z'))
   {
    switch(ch)
    {
       case 'a' : printf("it is a vowel");
                  break;
       case 'e' : printf("it is a vowel");
                  break;
```

Teacher's Signature

```
case 'i':
case 'o':
case 'u':
case 'A':
case 'E':
case 'I':
case 'O':
case 'U': printf("%c is a vowel", ch);
          break;
default: printf("%c is not a vowel, it is a
                 consonant", ch);

}
}
else
   printf("%c is not a character", ch);

return 0;
}
```

WAP to print the grade of a student if the mark is entered using switch case.

```
int main()
{
    int n, index;
    printf("Enter the marks of the student");
    scanf("%d", &n);
    index = n/10;

    switch(index)
    {
      case 10:
      case 9: printf("Grade = O");
              break;
```

```
case 8 : printf(" Grade = E ");
         break;
case 7 : printf (" Grade = A ");
         break;
case 6 : printf (" Grade = B ");
         break;
case 5 : printf (" Grade = C ");
         break;

default : printf (" FAIL!! ");
}
}
```

Dt - 17.9.19

Decision Making & Looping

In looping a sequence of statements are executed until some condition for termination are satisfied

Loop has 2 parts :-

1. Body of loop
2. the control statement.

A looping process, in general, includes the following 4 steps :-

① setting & initialisation of the condition variable / counter

② test for a specified value of the condition variable for execution of a loop.

③ execution of body of the loop.

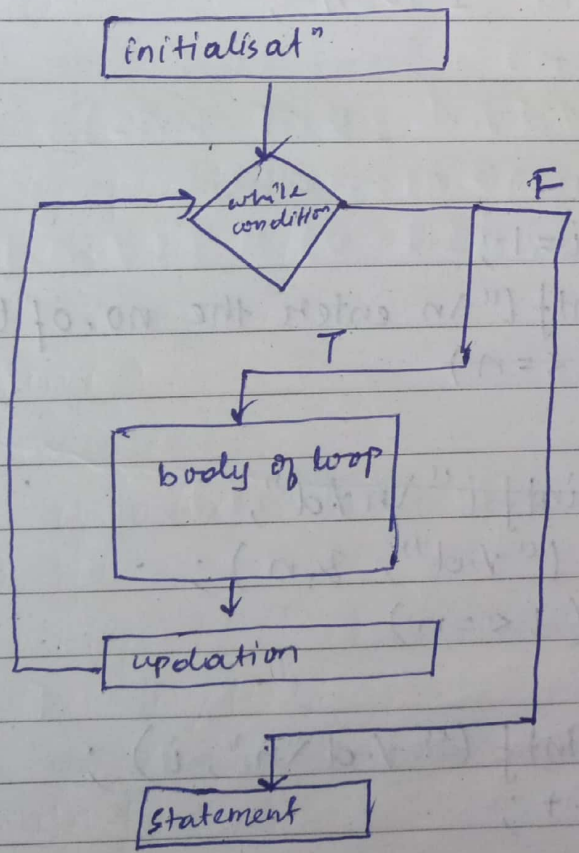
④ increamenting or updating the condition variable

C language provides 3 different types of construct to implement loop

1. while loop
2. do while loop / do statement
3. for loop

While loop SYNTAX

```
while ( condition )
{

   update the counter
}
```



Print 'silicon' 10 times using while loop

```
int main()
{
    int i=0;

    while(i<10)
    {
        printf("\n silicon");
        i++;
    }
    printf("\n End");
    return 0;
}
```

WAP to print 1 to n;

```
int main()
{
    int n, i=1;
    printf("\n enter the no. of terms");
    while(i<=n)
    {
        printf("\n%d",
    scanf("%d", &n);
    while(i<=n)
    {
        printf("%d\n", i);
        i++;
    }
    return 0;
}
```

Based on the position of control statement of the loop, loops can be categorised as :-

1. entry control loop
2. exit control loop → do..while,

The while & for loops

1. WAP to test the entered no. is prime or not

```
int main()
{
    int n, i=2, ~~count~~ ctr;
    scanf("%d", &n);
    printf(" enter a number");
    ctr = 0;
    while (i<=n/2)
    {
        if (n%i==0)
        {
            ctr++;
        }
        i++;
    }
    if (ctr>=1)
        printf(" it is compute ");
    else
        printf(" it is PRIME ");
```

2. WAP a program to compute GCD of 2 given positive integer

```
int main()
{
    int n1, n2;
    scanf("%d%d", &n1, &n2);
```

```
while (n1 != n2)
{
    ~~if (n1 = n1 - n2)~~
        if (n1 > n2)
            n1 -= n2;
        else
            n2 -= n1;
}
printf ("GCD = %d", n1);
}
```

12 4
8 4
4 4

2 | 18, 2
9, 1

WAP to print LCM of 2 given +ve integers

```
int main()
{
    int n1, n2, i;
    scanf ("%d %d", &n1, &n2);
    i = (n1 > n2) ? n1 : n2;
    while ((i % n1 != 0) || (i % n2 != 0))
    {
        i++;
    }
    printf ("LCM = %d", i);
}
```

WAP to print m terms of the Fibonacci series
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

```
int main()
{
    int m, x=0, y=1, i=3, fibo;
    scanf ("%d", &m);   // no. of terms
```

Teacher's Signature

r = n%.10
s = s + r*10;

```
printf("%d \t %d \t ", x, y);
while (i<=n)
{
    fibo = x+y;
    printf("%d \t", fibo);
     x = y;
     y = fibo;
      i++;
}
return 0;
}
```

WAP to compute the sum of the digits of a number

```
int main()
{
    int n, r, s=0;
    scanf("%d", &n);
    while (n!=0)
    {
        r = n%.10;
        s = s+r;        s = s*10;
        n = n/10;
    }
    printf("\n SUM of its digits are : %d", s);
    return 0;
}
```

12
r = 2
s = 30

WAP to reverse a number

```
int main()
{
    int n, r, s = 0, n1;
    scanf("%d", &n);
    n1 = n;
```

Teacher's Signature

```
while (n!=0)
{
    r = n%10;
    s = s*10 + r;
    n = n/10;
}
printf (" reversed no = %d", s);
}
if (s == n1)
    printf (" it is palindrome");
}
```

n = 53 → 0
r = 53 → 3
s = 5×10 + 3

s = 0
n = 53 → 0
r = 53 → 3
s = 5×10 + 3 = 53

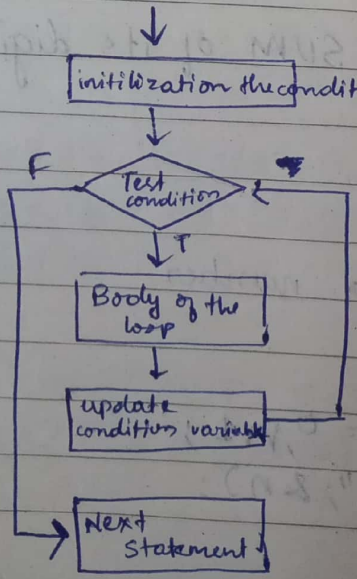
Dt - 20.9.19

Palin

for loop

The general syntax

```
for (initialisation ; test condition ; updation)
{
    ____
    ____
    ____
    ____
}
____
____
____
```



WAP to add all even nos between 1 to 100

```
int main()
{
    int n, sum = 0;
    scanf("%d",

    for( n = 1; n <= 100; n++)
    {
        if (n % 2 == 0)
            sum = sum + n;

    }
    printf(" SUM = %d", sum);

}
```

WAP to compute factorial of a number

```
int main()
{
    int n, f=1, N;
    scanf("%d", &n);

    for(N=n ; N != 0 ; N--)
    {
        f = f * N;

    }
    printf(" Factorial = %d", f);
    return 0;
}
```

2i+1

WAP to find

Sum = $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$

```
int main()
{
    float sum = 0.0;
    int n, N;
    scanf("%d", &N);

    for(n=1; n<=N; n++)
    {
        sum = sum + 1/(float)n;
    }

}
```

WAP to find

sum = $1 + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}$

```
int main()
{
    float sum = 0.0;
    int i, n, j, fact = 1;
    scanf("%d", &n);

    for(i=1; i<=n; i++)
    { fact = 1;
      for(j=1; j<=i; j++)
      {
          fact = fact * j;
      }
          sum = sum + 1/(float)fact;
    }
    printf("\n sum = %f", &sum);
```

Teacher's Signature

Assignment-2

WA Write all programs written on 19th .09.19 using for loop.

Dt - 24.9.19

Reversing a number using 'for loop'

```
rev=0
for( ; n>0 ; )
{
    rev = rev*10 + n%10;
    n = n/10;
}
```

break :- the keyword break allow us to jump out of the loop instantly without checking the conditional statement.

When the keyword break is encountered in a loop the flow control automatically passes to the first statement after the loop.
break or continue is applicable to the body of the loop it is defined

```
int main()
{
    int i;
    for(i=0 ; i<5 ; i++)
    {
        if(i<4)
        {
            printf("Hello");
            break;
        }
    }
}
```

i = 0123

Hello
Hello
Hello
Hello

Br. break is only for loop. So hello will be printed once

Teacher's Signature

```
int main()
{
    int i, j;
    for(i=0; i<5; i++)
    {
        for(j=0; j<4; j++)
        {
            if(i>1)
                continue;
            printf("Hello");
        }
    }
}
```

Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello

i = 0 1
j = 0 1 2 3 4
j = 0

n = 7
i = 2 3 i<=3

Continue -

The use of continue statement in a loop takes the program control directly to the test condition and then continues with the iteration process.

SYNTAX-

```
while (condⁿ)
{
    if (condⁿ)
        continue;
}
```

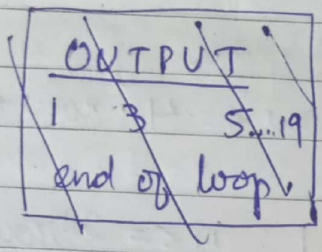
Example

```
int i=1;
while (i<=20)
{
    if (i%2==0)
```

i = 1 2 3 4 5 6 7

Teacher's Signature

```
    {  i++;
       continue;
    }
    printf ("%d \t", i);
     i++;
}
printf ("\n End of loop");
```



OUTPUT

1 3 5 7 9 11 13 15 17 19
End of loop.

```
int x;
for ( x = -1 ; x <= 10 ; x++)
{
        if ( x < 5 )
            continue;
        printf ("India");

}
```

```
int k;
for (k = 0 ; k <10 ; k+2)
        printf ("%d \t", k);
return 0;
```

PAGE NO.
DATE 15 / 10 / 2019

1. WAP to press the ~~posit~~ pattern

```
*
* * *
* * * * *
* * * * * * *
```

```
int main()
{
    int row, n;
    for (row = 1; row <= 4; row++)
    {
        for (n = 1; n <= 2*row - 1; n++)
        {
            printf (" * ");
        }
        printf ("*\n");
    }
    return 0;
}
```

2. WAP to print the following pattern

```
   *
  * * *
 * * * * *
* * * * * * *
```

```
int main()
{
    int row, n, s;
```

row = 2 3 2

```
for (row = 1 ; row <= 4 ; row++)
{
    for (s = 3 ; s >= 1 ; s--)

    for (s = 3 ; s >= row ; s--)
    {
        printf ("  ");   // use 3 spaces
    }
        for (n = 1 ; n <= 2*row - 1 ; n++)
        {
            printf ("* ");   // space
        }
        printf ("\n");
}
```

3. WAP to print * in triangle shape of any no. of rows entered by user.

```
int main ()
{
    int row, n, NR, space ;
    scanf ("%d", &NR) ;
    space = NR - 1 ;
    for (row = 1 ; row <= NR ; row++)
    {
        for (j = 1 ; j <= space ; j++)
        {
            printf (" ") ;
        }
        space -- ;
        for (n = 1 ; n <= 2*row - 1 ; n++)
        {
            printf ("*") ;
        }
        printf ("\n") ;
}
```

Teacher's Signature

```
ch = 'A'
ch++;
```

5. Print

```
        A
      B B B
    C C C C C
  D D D D D D D
```

```
int main()
{
    char ch = 'A';
    int row, n, space, j;
    space = 3;
    for (row = 1; row <= 4; row++)
    {
        for (j = 1; j <= space; j++)
        {
            printf(" ");
        }
        space--;
        for (n = 1; n <= 2*row - 1; n++)
        {
            printf("%c", ch);
        }
        ch++;
        printf("\n");
    }
    return 0;
}
```

Assignment 6.

Print

```
        A
      A B A
    A B C B A
  A B C D C B A
```

Teacher's Signature

ARRAY

PAGE NO.
DATE 16 / 10 / 2019

An array is a collection of variables of same data type that are referred through a common name. A specific element / variable of the array is accessed by it's index

An array is an fixed sized sequenced collection of same type of elements. An array is also known as subscripted variable.

However big an array, its elements are always stored in contiguous memory location.

DECLARATION

```
data type  Name of array[size];
```

```
int A[5];
float B[100000];
char C[3];
```

INITIALISATION

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

OR

```
int A[5];
for(i=0; i<5; i++)
{
    scanf("%d", &A[i]);
}

for(i=0; i<5; i++)
{
    printf("%d", A[i]);
}
```

Teacher's Signature

Before using an array its type and dimension must be declared.

```
int main()
{
    int A[100], i, size;
    print(" Enter the size of the Array \n");
    scanf("%d", &size);
    print(" \n enter the size of the
    printf("\n Enter the elements of the array: ");
    for(i=0; i<size; i++)
    {
        scanf("%d", &A[i]);
    }
```

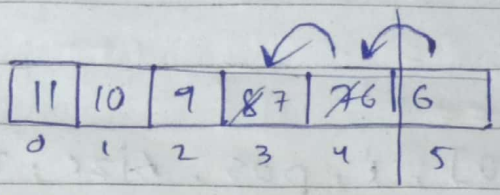
Operation on Array :-

SEARCHING

```
printf("\n enter the element you want to
        search \n");
scanf("%d", &x);

for(i=0; i<size; i++)
{
    if(A[i]==x)
    {
        printf("\n element is found at %d",
                i);
        break;
    }
}
  if(i==size)
      printf("\n element not found");
return 0;
}
```

Teacher's Signature _____

Deletion




```
int main()
{
    int A[100], i, pos, size;
    printf("\n Enter the size of the array\n");
    scanf("%d", &size);
    printf("\n Enter the elements of the array");
    for(i=0; i<size; i++)
    {
        scanf("%d", &A[i]);
    }
    printf("\n enter the position of the element to
            delete\n");
    scanf("%d", &pos);
    for(i=pos; i<size; i++)
    {
        A[i-1] = A[i];
    }
    size--;
    for(i=0; i<size; i++)
    {
        printf("%d", A[i]);
    }
    return 0;
}
```

WAP to insert an element in an array

```
int main()
{
    int A[100], i, pos, size, x;
    printf("\n Enter the size of array \n");
    scanf("%d", &size);
    printf("\n enter the element");
    for( i=0; i<size ; i++)
    {
        scanf("%d", &A[i]);
    }
    printf(" Enter the element you want to
            enter and position");
    scanf("%d", &x);
    scanf("%d", &pos);
    size++;
    for( i=size-1 ; i>=pos-1 ; i--)
    {
        A[i] = A[i-1];
    }
    A[i] = x ;

    for( i=0; i<size ; i++)
    {
        printf("%d", A[i]);
    }
    return 0;
}
```

WAP to find the smallest element present in in an array.

```
int main()
{
    int A[100], i, size, max;
    printf("\n enter the size of array");
    :
    :

    max = A[0];
    for ( i = 1 ; i < size ; i++ )
    {
        if ( max < A[i] )
            max = A[i];

    }
    printf("\n the minimum number in the
            array is %d", max );

    return 0;
}
```

WAP to print the second smallest

```
min = A[0];
for ( i = 1 ; i < size ; i++ )
{
    if ( min < A[i] )
    {
        min = A[i];
        break;
    }
}
i = j;
```

Teacher's Signature

```
for (i=0; i<size; i++)
{
    if (i==j)  if (
        conti.
    if (i!=j)
    {
        if(n
```

```
int main()
{
    int A[] = { 5, 1, 9, 7, 2, 1, 10};
    int i, s, ss, size = 7;
    if (A[0] < A[1])
    {
        s = A[0];
        ss = A[1];
    }
    else
    {
        s = A[1];
        ss = A[0];
    }
    for(i = 2; i < size; i++)
    {
        if (A[i] < s)
        {
            ss = s;
            s = A[i];
        }
        else if (A[i] < ss && A[i] != s)
        {
            ss = A[i];
        }
}
```

Teacher's Signature _____

9	5	6	2
0	1	2	3

size = 4

t = 2

2	9	5	6
0	1	2	3

PAGE NO.
DATE 18 10 20 19

```
    printf("the smallest is %d, second smallest is %d",
            s, ss);
}
```

WAP to reverse the elements of an array

```
int main()
{
    int A[100], t, size, i;
    :
    :
    t = A[size-1];
    for(i=0; i<size; i++)
    {
    }
    for(i=size-2; i>=0; i--)
    {
        A[i+1] = A[i];
    }
    A[0] = t;
}
```

size = 4

i = ~~1~~ 0

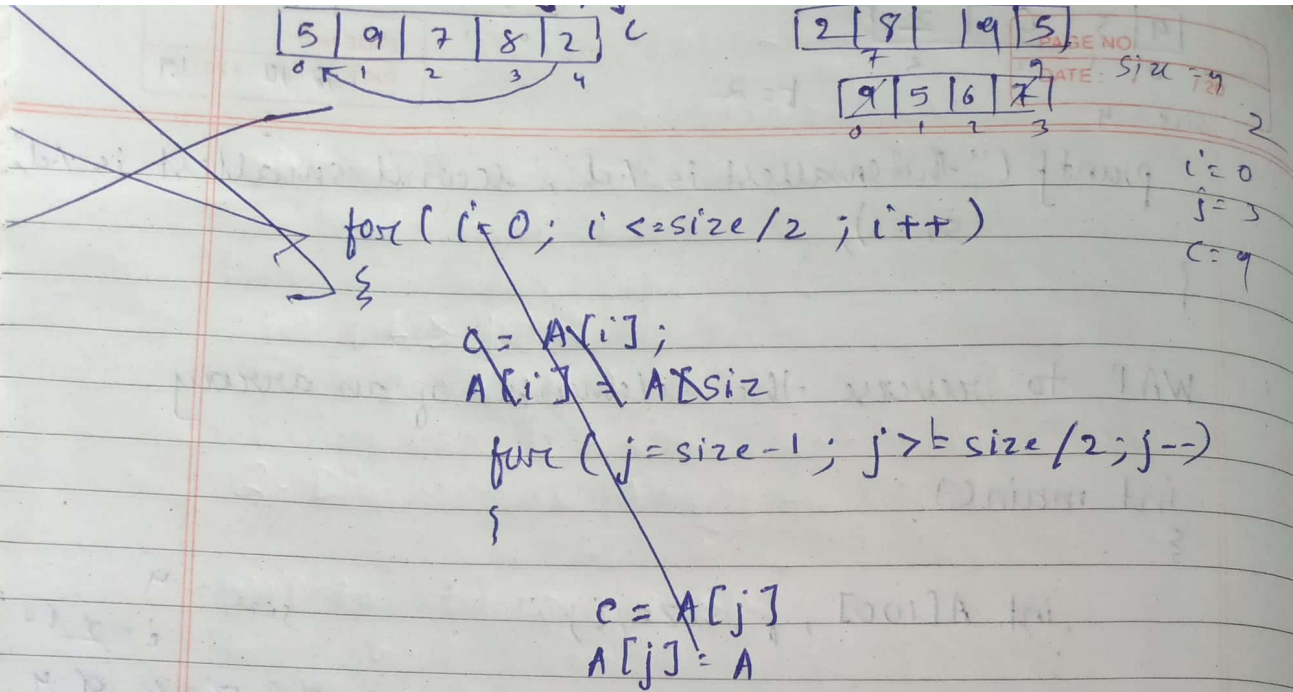
9 5 2 6 9

0 1 2 3 4 5 6

t = 9

i = 2

```
    for(i=0; i<size/2; i++)
    {
        temp = A[i];
        A[i] = A[size-1-i];
        A[size-1-i] = temp;
```



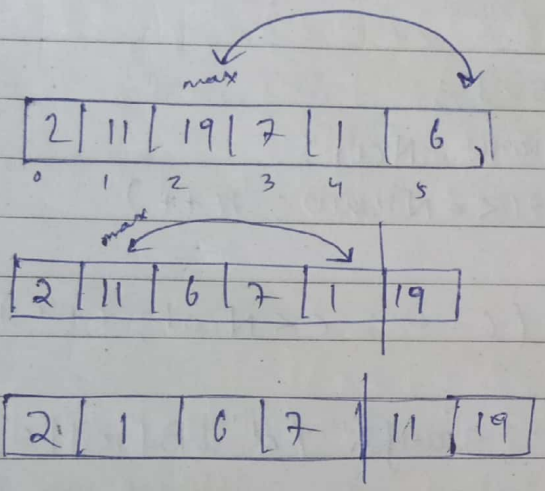

```
for ( i=0, j = size-1 ; i<= size/2, j>=size/2 ; i++, j--)
{
        c = A[i];                    i = 0 1 2
        A[i] = A[j];                 j = 4 3 2
        A[j] = A c;                  c = 5 9
}
```

Dt - 22.10.19

SORTING OF ARRAY

```
int main()
{
    int A[100],i, max , pass , size, temp;
    .
    .
    .
    // max is the variable, stores the index where the max element is pre
    for (pass = 0 ; pass < size-1 ; pass ++)
    {
        max = 0 ;
        for (i=1 ; i <= size - pass ; i++)
        {
            if ( A[i] > A[max])
                max = i;
        }
```

```
        temp = A[max];
        A[max] = A[max-pass-1];
        A[max-pass-1] = temp;
    }

    printf("\n the array after sorting \n");
    for (i=0; i<size; i++)
    {
        printf(" %d", A[i]);
    }
    return 0;
}
```



Binary search -

It can be only applied on sorted array.

```
while (l<=f)
{
    mid = (l+f)/2;
    if (A[mid] == x)
    {
        printf("\n Element is found at %d", mid);
        break;
    }
    else if (A[mid] < x)
        l = mid + 1;
```

present

Teacher's Signature

```
for(i=0; i<size; i++)
{
    if (A[i] < min)
        min = A[i];
    if (A[i] > max)
        max = A[i];
else
    {
        l = mid - 1;
}
  if (l > f)
      printf("\n element is not found");
```

Dt - 23.10.19

2- Dimensional Array -

WAP to declare a 2D array to read the value and print them.

```
int main()
{
    int B[50][50];
    int r, c, Nrow, Ncol;
    for(r=0; r<Nrow; r++)
    {
        for (c=0; c<Ncol; c++)
        {
            scanf("%d", &B[r][c]);
        }
    }
    printf("\n the array is \n");
    for (r=0; r<Nrow; r++)
    {
        for (c=0; c<Ncol; c++)
        {
            printf("%d", B[r][c]);
        }
    }
}
```

Column major order

```
for (j=0; j<Ncol; j++)
{
    for (i=0; i<Nrow; i++)
    {
        printf("%d", B[i][j]);
    }
}
```

WAP to add two given matrices.

```
for(i=0; i<Nrow; i++)
{
    for(j=0; j<Ncol; j++)
    {
        A[i][j] = A[i][j] + B[i][j];
    }
}
```

WAP to multiply 2 matrices $A_{m \times n} * B_{q \times r}$

```
int main()
{
    int A[20][20], B[20][20], C[20][20];
    int i, j, k, m, n, q, r;
    printf("\n enter the dimension of matrix A \n");
    scanf("%d %d", &m, &n);
    printf("\n enter the dimension of matrix B \n");
    scanf("%d %d", &q, &r);
    if(m==q)
    {
        for(i=0; i<m; i++)
        {
```

Column major order

```
for (j=0; j<Ncol; j++)
{
    for (i=0; i<Nrow; i++)
    {
        printf("%d", B[i][j]);
    }
}
```

WAP to add two given matrices.

```
for(i=0; i<Nrow; i++)
{
    for (j=0; j<Ncol; j++)
    {
        A[i][j] = A[i][j] + B[i][j];
    }
}
```

WAP to multiply 2 matrices $A_{m \times n} * B_{q \times r}$

```
int main()
{
  int A[20][20], B[20][20], C[20][20];
  int i, j, k, m, n, q, r;
  printf("\n enter the dimension of matrix A \n");
  scanf("%d %d", &m, &n);
  printf("\n enter the dimension of matrix B \n");
  scanf("%d %d", &q, &r);
  if(n==q)
  {
    for (i=0; i<m; i++)
    {
```

```
for (j=0; j<n; j++)
{
    C[i][j] = 0;
    for(k=0; k<n; k++)
    {
        C[i][j] = C[i][j] + A[i][k] *
                  B[k][j];
    }
}
}
```

Transpose of a Matrix

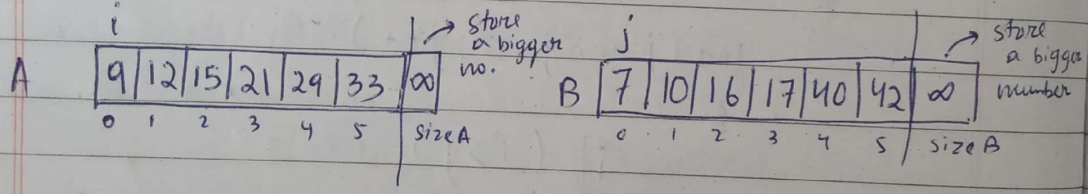
```
int main()
{
    .
    .
    .
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            if(i>j)
            {
                temp = A[i][j];
                A[i][j] = A[j][i];
                A[j][i] = temp;
            }
        }
    }
```

```
#include<stdio.h>
int main()
{
    int m,n,i,j,A[][];
    printf("\n enter the size of the matrix \n");
    scanf("%d%d",&m,&n);
    printf("\n enter values into the matrix \n");
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&A[i][j]);
        }
    }
    if(m>n)
    {
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            if(i>j)
            {
                temp=A[i][j];
                A[i][j]=A[j][i];
                A[j][i]=temp;
            }
        }
    }
}
else
{
        for(i=0;i<m;i++)
        {
            for(j=0;j<n;j++)
            {
                if(i<j)
                {
                temp=A[i][j];
                A[i][j]=A[j][i];
                A[j][i]=temp;
                }
            }
        }
}
```

10) vi sort.c

WAP to merge two sorted array to create another sorted array.

```
int sizeA, A[10], B[10], C[20], k=0, sizeB;
int i=0, j=0;
scanf("%d", &A[sizeA]); scanf("%d", &B[
                                  A[sizeA] = 9999;
for(k=0; k< sizeA+sizeB ; k++)    B[sizeB] = 9999;
{
    if (A[i] < B[j])
    {
        C[k] = A[i];
        i++;
    }
    else if (B[j] < A[i])
    {
        C[k] = B[j];
        j++;
    }
}
```



13. vi shuffle.c

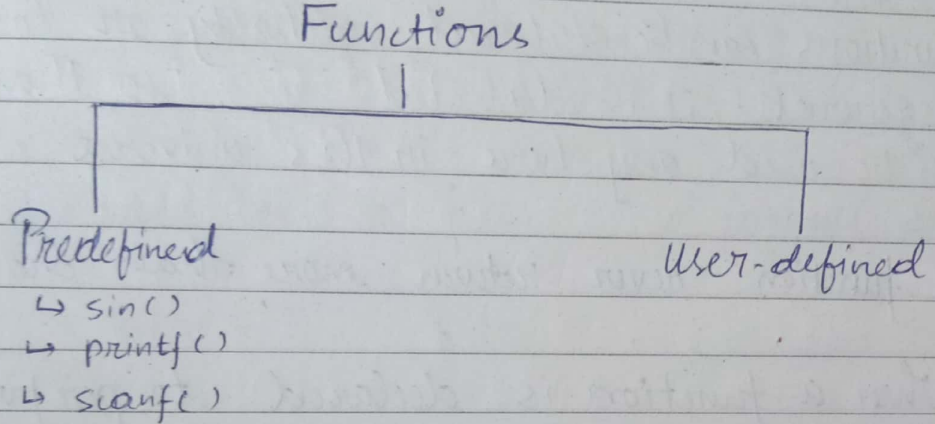
```
if (size%2 == 0)
{
    for (i=0; i<size; i=i+2)
        temp = A[i];
        A[i] = A[i+1];
        A[i+1] = temp;
}
else
{
    for (i=0; i<size-1; i=i+2)
        same
```

14. vi selection.c

FUNTIONS

PAGE NO.
DATE 25 10 19

A function is a self content block of statement that perform a coherent task of some kind and may return a single value.
C language provides an approach in which we can declare and define a group of statements once in the form of a function and it can be called and used whenever required.



Library functions are those which are already defined in C library.
User defined funct" are those which are declared by user during writing of program.

Advantages-

1. It provides modularity (divide the prog. into small module) to a funct".
2. It makes the code reusable
3. In cases of large program, debbugging and editing will be easier.
4. It makes the program readable & more easy to understand

C functions are written in Program in 3 phases

1. Function declarat"
2. Function defination

DATE 29/10/2019

3. Function call

Funct" are also called derived data type in C

Function Declaration -

return_datatype Namefunct"(typeId1, ... typeIdn);

e.g; int add(int a, int b); — arguement list

Functions can be declared globally or locally In general, it is declared " so that it can be used anywhere in this universe

A function never return more than one value

When a function is declared to perform some calc or operat", it is expected to provide some result at the end. In such cases, a return statement is added in the end of the funct" definat". In case, a funct" does not return any value, the return type will be void.

The parameter list declare the type and the no. of arguement that the function expect when it is called. They are called formal parameter

Function Defination - in difinat" both datatype and name of arguement is compulsory

return_type Name(data type a1, ..., data type an)
{

return();
}

Teacher's Signature

```
#include<stdio.h>
int add(int , int );

int main()
{
        int x, y, sum;
        scanf("%d %d", &x, &y);
        sum = add(x, y);                    // actual argument
        printf("~~%d~~ total = %d", sum);
        return 0;
}
int add(int a, int b)                       // formal argument
{
    int s;
    s = a+b;
    return s;
}
```

Dt-30.10.19

1. Any function can be called from any other function. Even main() can be called from other "
2. The function which calls other funct" is known as calling function.
3. The function which is called by another funct" is called called funct"
4. The arguements used in the funct" call are known as actual arguement.
5. The arguements used in the funct" definat" are known as formal arguements.
6. A function can be called any number of times
7. The order in which the funct"s are defined in a program and the order in which they get called may not be necessarily be ~~that~~ same.

Teacher's Signature

8. A function cannot be defined in another function
9. The return statement can be written as return x; return (x);
10. The return type of a funct^n is optional
11. When the funct^n doesnot have any return type it returns an integer by default.

Function without return type & arguement

```
#include<stdio.h>
void add( ); // funct" declaration

int main()
{
    int res;
    res = add();
    printf("%d", res);
    return 0;
}
void add()
{
    int x, y, sum;
    scanf("%d %d", &x, &y);
    sum = x+y;
    printf("\n sum = %d", sum);
}
```

Function with arguement but no return type

```
#include<stdio.h>
void add(int, int);

int main()
{
    add( int x, y;
         scanf("%d %d", &x, &y);
```

Teacher's Signature

```
    add(x, y);
    return 0;
}
void add (int a, int b)
{
        int sum;
        sum = a+b;
        printf("\n sum = %d", sum);
}
```

Function with no arguement but with return type

```
#include <stdio.h>
int add();

int main()
{
    int s;
s = ~~void~~ add(); printf("\n sum = %d", s);
    return 0;
}
int add()
{
    int x, y, sum;
    printf("\n enter 2 values");
    scanf("%d %d", &x, &y);
    sum = x+y;
    return sum;
}
```

```
for (i=n; i>=1; i--)
    fact = fact*i;
```

WAP to compute factorial of a number using function.

```
#include<stdio.h>
int factorial(int);

int main()
{
    int n,f;
    scanf("%d", &n);
    f = factorial(n);
    printf("\n factorial = %d", f);
    return 0;
}
int factorial(int a)
{
    int fact=1,i;
    for(i=a; i>=1; i--)
    {
        fact = fact*i;
    }
    return fact;
}
```

WAP to compute sum $= \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}$ using functions

```
#include <stdio.h>

float add(int);
int main( int fact(int);
int main()
{
    int n;
```

```
    scanf("%d", &n);
    float s;
    s = add(n);
    printf("the sum = %f", s);
    return 0;
}
int fact(int m)
{
    int i, f = 1;
    for(i=1; i<=m; i++)
        f = f*i;
    return(f);
}

float add(int k)
{
    float sum = 0.0, term;
    int i;
    for(i=1; i<=k; i++)
    {
        term = 1/fact(i);
        sum += term;
    }
    return(sum);
}
```

WAP to compute x^y for any 2 int. x & y using function

```
#include<stdio.h>
float pow_z(int, int);

int main()
{
```

```
int x, y;
float p;
printf("\n enter values of x & y \n");
scanf("%d %d", &x, &y);
p = pow_z(x, y);
printf("\n answer = %f", p);
return 0;

}

float pow_z(int a, int b)
{
    int i; float m; float p = 1.0;
    for(i=1; i<=b; i++)
    {
        if(b>=0)
        {
            m = a*b;
        }
        else
            m = 1/(float) a*b;
    }
    return m;
}
    if(b>=0)
    {
      for(i=1; i<=b; i++)
          p = p*a;
    }
    else
    {
        for(i=1; i<=(-b); i++)
```

```
        p = p*( 1/(float) a);
    }
    return (p);
}
```

WAP to compute LCM of 2 nos using function

```
#include <stdio.h>
int LCM(int, int);

int main()
{
    int a, b; int L;
    scanf("%d %d", &a, &b);
    L = LCM(a,b);
    printf("\n LCM = %d", L);
    return 0;
}
int LCM(int x, int y)
{
    int i, lcm;

    for(i=1; i<=x*y; i++)
    {
        if(i%x==0 && i%y==0)
        {
            lcm = i;
            break;
        }
    }
    return(lcm);
}
```

Teacher's Signature

PAGE NO.
DATE : 1 / 11 / 2019

1. WAP to swap the value of two variables using function.

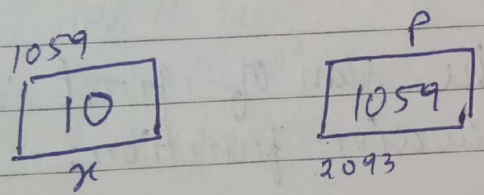
```
#include <stdio.h>
void swap(int, int);

int main()
{
    int x, y;
    printf("%d%d" scanf("%d%d", &x, &y);
    printf("\n values before swapping \n x=%d, y=%d",
            x, y);
    swap(x,y);
    printf("\n values after swapping \n x=%d, y=%d");
    return 0;
}
void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

Funct^n can be called by 2 methods:-

(i) call by value

(ii) call by reference or address

```
void swap(int *, int *);
in main()
{
    int x=10, y=22;
    swap(&x, &y);
    printf("\n x=%d, y=%d");
    return 0;
}
void swap(int *P1, int *P2)
{
    int temp;
    temp = *P1;
    *P1 = *P2;
    *P2 = temp;
}
```




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

printf
- p // ~~100~~ 1059
- x // 10
- &x // 1059
- *p // 10
- &p // 2093

PAGE NO.
DATE 5/11/2019

Recursion -

Recursion is a process where the function calls itself. The funct" with recursive call are called recursive funct".

Every recursive function definat" has 2 sections

1. Base condition testing
2. Recursive call.

```
int main()
{
    int x=10;
    printf("%d", x);
    main();
    return 0;
}
```

1 WAP to compute the sum of first 'n' natural number using a recursive function.

```
#include<stdio.h>

int Sum(int);

int main()
{
    int n, s;
    scanf("%d", &n);
    s = Sum(n);

    printf("%d sum = %d", s);
    return 0;
}
```

```
int Sum(int a)
{
    if(a==0)
        return 0;
    else
        return(a+sum(a-1));
}
```

2. WAP to compute factorial using a recursive function

```
#include<stdio.h>
int factorial(int);
int main()
{
    int n, fact;
    scanf("%d", &n);
    fact = factorial(n);
}
int factorial(int f)
{
    if(f==0)
        return 1;
    else
        return(f*factorial(f-1));
}
```

```
if(k>0)
{
    f = f*fact(k);
    k--;
}
```

DATE 6/11/2019

WAP to compute GCD of 2 nos using recursive function

```
#include<stdio.h>

int GCD(int a, int b)
{
        if(a==b)
            return a;

        else if(a>b)
            return GCD(a-b,b);

        else
            return GCD(a,b-a);
}

int main()
{
    int m,n,gcd;
    scanf("%d %d",&m,&n);
    gcd = GCD(m,n);
    printf("\n GCD = %d",gcd);
    return 0;
}
```

WAP to print 'n' fibonacci numbers using recursive function.

```
#include<stdio.h
int fibonacci(int);

int main()
{
```

Teacher's Signature

```
int n, i;
scanf("%d", &n);
i=1;
while (i<=n)
{
        printf("%d", fibonacci(i));
        i++;
}
return 0;
}
int fibonacci(int k)
{
    if(k==0)
        return 0;
    else if (k==2)
        return 1;
    else
        return (fibonacci(k-1) + fibonacci(k-2));
}
```

Array and Function :-
The array can be passed by call by reference

```
#include<stdio.h>
void update(int A[ ], int );

int main()
{
    int Mark[100], i, size;
    scanf("%d", &size);
    for(i=0; i<size; i++)
        scanf("%d", &Mark[i]);
```

Teacher's Signature

```
    update(Mark, Size); for(i=0; i<size; i++)
                            printf("%d", Mark[i]);
}
void update(int B[], int s)
{
    int i;
    for(i=0; i<s; i++)
        B[i] += 5;
}
```

Dt - 7.11.19

WAP to search an element using function -

```
#include<stdio.h>
int search(int[], int, int);        int[],
int main()
{
    int A[50], size, i, x;
    .
    .
    .
    scanf("%d", &x);
    int loc;
    loc = search(A, x, size);
    if(loc==-1)
        printf("\n element not found");
    else
        printf("\n element found at index %d",
                loc);
    return 0;
}

int search(int A[50], int x, int s)
{
    int i, c;
```

```
    for(i=0; i<s; i++)
    {
        if(x==A[i])
            return(i);
    }
    if(i==s)
        return -1;
}
```

WAP to compute binary search using function and recursion

```
int B_search(int A[], int x, int l, int u);

int main()
{
    int A[50], x, i, size, loc;
    .
    .
    .
    loc = B_search(A, x, 0, size-1);
    if(loc==-1)
        printf("\n element not found \n");
    else
        printf("\n element is found at %d",
                loc);
    return 0;
}
int B_search(int A[50], int x, int l, int u)
{
    int mid;
    if(l>u)
        return -1;
```

```
else
{
        mid = (l+u)/2;
        if (A[mid] == x)
                return(mid);
        else if (A[mid] < x)
                return(A, x, mid+1, u);

        else
                B_search(A, x, l, mid-1);
    }
}
```

A = | 5 | 9 | 11 | 21 | 48 |
0 1 2 3 4

WAP using recursion to reverse an array :-

```
void(int reverse(int
```

Dt - 8.11.19

Passing 2-D array to a function -

```
void display (int A[][3], int, int);
int main()
{
        int B[50][50], m, n;
                :
                :
                :
        display(B, m, n);
        return 0;
}
void display(int B[50][50], int m, int n)
{
```

```
int i, j;
for(i=0; i<m; i++)
{
    for(j=0; j<n; j++)
    {
        printf("%d", B[i][j]);
    }
    printf("\n");
}
```

Strings :-

STRINGS

PAGE NO. ∞
DATE 8 / 11 / 19

The character array ended with null character '\0'

String is character array but not vice versa
A string const. is a 1-D dimen array of character terminated by '\0' or NULL.
e.g.,
char str[] = "KIIT"; |K|I|I|T|\0|

Or

char str[10] = {'K', 'I', 'I', 'T', '\0'};

Or

char str[50];
scanf("%s", str);

---

char str[50];
int size = 10, i;
for (i=0; i<10; i++)
    scanf("%c", &str[i]);

str[i] = '\0';

String Operation -
C library header file string.h define the following :-

① strlen(s)
to compute length
char s[50] = "silicon";
int l = strlen(s);

(2) strlwr(s)
To convert the string to lower case

(3) strupr(s)
To convert string to upper case

(4) strcat(s1, s2)
append the string s_2 at the end of the string s1.

```
char s1[50] = "ABC";
char s2[50] = "DEFG";
    strcat(s1, s2)      ⟶ it will be stored in s1
printf("%s", s1); //ABCDEFG
printf("%s", s2); //DEFG
```

(5) strncat(s1, s2, k);
Here only k character of s2 is appended on s1

(6) strcpy(s1, s2);
copies the string s2 to string s1

```
char s1[50] = "ABC";
char s2[50] = "DEFG";
strcpy(s1, s2);
printf("%s", s1);
printf("%s", s2);
```

(7) strncpy(s1, s2, n);

(8) strcmp(s1, s2)
Compare the string s1 & s2 and it returns an integer less than zero if s1 is found less than s2.
return 0 if s1 & s2 are equal
Greater than zero if s1 > s2

(9) strrev(s)
It reverses the string

Teacher's Signature

scanf("%[

PAGE NO.
DATE 13/11/2019

Computing length of the string :-

```
int Mystrlen(char [ ]);
int main()
{
        char str1[50];
        int len;
        scanf("%[^\n]s", str1);
        len = Mystrlen(str1);
        printf("\n length of the string = %d", len);
        return 0;
}
int Mystrlen(char S[50])
{
        int l=0, i;
        for (i=0; S[i] != '\0'; i++)
        {
                l++;
        }
        return l;
}
```

Computing ~~tolower~~ toupper for a string

```
int Mystrlen (char [ ]);
void Mystrupr (Char [ ]);
void Mystrlwr (char [ ]);
void Mystrcat (Char [ ], char [ ]);
void Mystrcpy (char [ ], char [ ]);
Void Mystrrev (char [ ]);
int Mystrcmp (char [ ], char [ ]);
```

```
int main()
{
    char str1[50], str2[50]; read strings
    int i, l, d;
    printf("\n enter 1 for strlen,
                enter 2 for upper,
                enter 3 for lower,
                    .
                    .
                    .
                enter 7 for strcmp\n");
    scanf("%d", &i);
    switch(i)
    {
        case 1:
            l = Mystrlen(str1);
            printf("%d", l);
            break;
        case 2:
            Mystrupr(str1); printf("%s",str1);
            break;
        case 3:
            Mystrlwr(str1);
            printf("%s", str1); break;
        case 4:
            Mystrcat(str1, str2);
            printf("%s", str1);
            break;
        case 5:
            Mystrcpy(str1, str2);
            printf("%s", str1);
                .
                .
                .
```

```
void Mystrupr(char S[50])
{
        int i;
        for(i=0; s[i]!='\0'; i++)
        {
                if(s[i]>='a' && s[i]<='z')
                        s[i] = s[i]-32;

        }
}

void Mystrlwr(char s[50])
{
        int i;
        for(i=0; S[i]!='\0'; i++)
        {
                if(S[i]>='A' && s[i]<='Z')
                        s[i] += 32;

        }
}

void Mystrcat(char s1[50], char s2[50])
{
        int l = Mystrlen(s1);
        int i;
        for(i=0; s2[i]!='\0'; i++)
        {
                s1[l] = s2[i];
                l++;
        }
        s1[l] = '\0';
}
```

```
void Mystrcpy (char S1[50], char S2[50])
{
    int i;
    for(i=0; S2[i]!='\0'; i++)
        S1[i] = S2[i];

    S1[i] = '\0';
}

void Mystrreev(char S[50])
{
    int l = Mystrlen(s);
    int i; char temp;
    for(i=0; i<l/2; i++)
    {
        temp = S[i];
        S[i] = S[l-1-i];
        S[l-1-i] = temp;
    }
}

int Mystrcmp(char S1[50], char S2[50])
{
    int l1 = Mystrlen(S1);
    int l2 = Mystrlen(S2);
    for(i=0, j=0; i<l1, j<l2; i++, j++)
    {
        if(S1[i] != S2[j])
            return(S1[i] - S2[j]);
    while(S1[i] != '\0' && S2[i] != '\0')
    {
        if(S1[i] != S2[i])
            return(S1[i] - S2[i]);
        else
            i++;
    }
```

Teacher's Signature

double → 1.000123910151 7

Char ch = 'A' print ch : A

int x = 'A'; //65

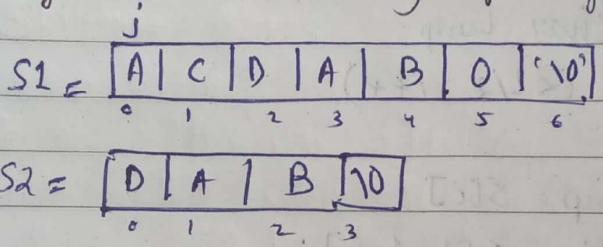
print x : 65

```
if (S1[i] == '\0' && S2[i] == '\0')
        return 0;

    else
        return (S1[i] - S2[i]);
}
```

WAP to search a pattern in a given string

Assignment

WAP to find all occurance of a pattern of length 'm' in a string of length 'n'.




```
if (S[j] == P[0] && S[j+1] == P[1] &&
    S[j+2] == P[2])
        return(j);
```

DL - 15.11.19

Pointer :-

- Pointer is used to store the address of another variable.
- it is a derived datatype.
- must be declared before using it.
- syntax

  data_type * variable_name ;
- data_type should be valid.
- pointer stores a long hexa decimal number that represents the address of variable.

The only diff bet^n the pointers of different datatype is the datatype of the variable that the pointer points.

9374999

Advantages of pointer

1. Pointer enables us to access a variable which is not defined within the scope of funct"
2. It increases the execution field.
3. Through pointer it can return multiple values
4. It reduces the length and complexity of program.
5. using pointer you can save memory by dynamic memory allocation.

WAP to compute the factorial of a number using pointers & functions :-

```
#include <stdio.h>
int fact(int *);

int main()
{
    int x=10, f;
    f = fact(&x);
    printf("factorial = %d", f);
    return 0;
}
int fact(int *f)
{
    int i, fa = 1;
    for(i=*f; i>=1; i++)
        fa = fa*i;
    return(fa);
}
```

Array And Pointers :-

```
int A[10] = {2,3,4,5,6};
```

```
int *ptr;
ptr = A; // here ptr = 0
printf("%d", A[1]);      // 3
printf("%d", *(ptr+1));  // 3
for(i=0; i<10; i++)
        printf("%d", *(ptr+i));
```

Example-2

```
char *p = {'a','b','c'};
```

Dt - 19.11.19

Storage class

Every variable we declare in C has a storage class
A variable's storage class tells us:-

- where the variable would be stored
- What will be the default initial value?
- What is the scope of the variable
- what is the life of the variable.

There are 4 storage classes:-

1. Automatic storage class
2. Register " "
3. static " "
4. External " "

Automatic Storage Class-

The features of a variable with automatic storage class (auto) are:-

- Storage: memory
- Default initial value: An unpredictable value, which is often called a garbage value
- Scope: local to block in which the variable is defined.

Teacher's Signature

- Life: Till the control remains within the block in which the variable is defined.

e.g.,
```
main()
{
    auto int x;
    printf("%d", x);    // 1233 (garbage value)
}
```

eg.
```
main()
{
    auto int x = 1;
    {
        {
            {
                printf("%d", x);
            }
            printf("%d", x);
        }
        printf("%d", x);
    }
}
```

Output
1 1 1

eg.
```
main()
{
    auto int x = 1;
    {
        auto int x = 2;
        {
            auto int x = 3;
            printf("%d", x);
        }
        printf("%d", x);
    }
    printf("%d", x);
}
```

Output
3 2 1

Teacher's Signature _____

PAGE NO.
DATE : / / 20

Register storage class

A value stored in a CPU register can always be accessed faster than one that is stored in memory.

So in a program if we use a variable in many places then we should declare its storage class as register

e.g. loop counter.

Features :-

- Storage : CPU registers
- Default initial value : Garbage value
- Scope : local to the block in which the variable is defined.
- Life : Till the control remains within the body in which the variable is defined.

e.g.,

```
main()
{
    register int i;
    for(i=1; i<=10; i++)
```

Even though we declare the storage class of a variable as register we can't say for sure that the variable is stored in CPU. Register because ~~the variable is stored~~ no. of CPU registers are limited, they may not be free. In that case the compiler will treat that variable.

We can't use register storage class for all type

Static Storage class

Pointers and Array

A[ ]
A

PAGE NO.
DATE 20/11/2019

WAP to compute the sum of the elements of an array using function and pointer.

```
int sum(int *, int);
int main()
{
    int A[50], size, s, i;
    scanf("%d", &size);
    for(i=0; i<size; i++)
        scanf("%d", &A[i]);

    s = sum(A, size);

}
int sum(int *ptr, int n)
{
    int i, total = 0;
    for(i=0; i<n; i++)
        total += *(ptr+i);
    return(total);
}
```

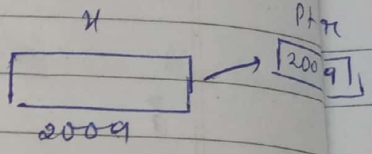
WAP to convert a string into lower case using function and pointer.

```
void tolwr(char *);

int main()
{
    char str[50];
    scanf("%[^\n]s", str);
    tolwr(str);
```

```
void tolwr (char *p)
{
    int i;
    for(i=0; *(p+i)!='\0'; i++)
    {
        if (*(p+i)>='A' && *(p+i)<='Z')
            *(p+i) += 32;
    }
}
```

Pointer Arithmatic

We can increament or decreament a pointer variable.



int size = 4

```
int *ptr, x;
ptr = &x;
ptr++;
printf("%d %lu", ptr); // 2013
```

When we increament a pointer it is incremented by size of. that pointer. The size of that variable is known as scale factor.

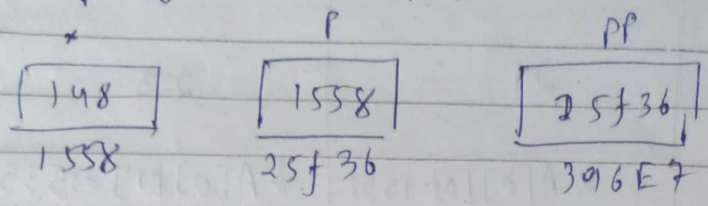
Dt - 21.11.19

Pointer to Pointer

```
int main()
{   int x=148, *p; **pp;
    p=&x;
    pp = &p;
    printf("%d", x);   // 148
    printf("%d", *p);  // 148
    printf("%d", **pp); // 148
    printf("%Lu", &x); // 1558
    printf("%lu", p);  // 1558
    printf("%lu", pp); // 2536
```

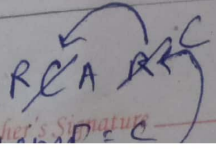
x=lu same

printf("%lu", &p); // 25f36
printf("%x", &pp);



WAP to reverse a string using function strrev where one arguement is pointer type

```
Void strrev(char *);
int main()
{
    char str[50];
    printf("\n enter a string \n");
    scanf("%[^\n]s", str);
    strrev(str);

}
void strrev(char *ptr)
{
    int l=0,i; char temp;
    for(i=0; *(ptr+i)!='\0'; i++)
            l++;
    for(i=0; i<l/2; i++)
    {
        temp = *(ptr+i);
        *(ptr+i) = *(ptr+l-i-1);
        *(ptr+l-i-1) = temp;
    }
}
```



Teacher's Signature

PAGE NO.
DATE : / / 20

Pointer to a 2-D Array

	0	1
0	21	23
	&A[0][0]=1531	&A[0][1]=1535
1	29	41
	&A[1][0]=1539	&A[1][1]=1543
2	46	50
	&A[20][0]=1547	&A[2][1]=1551

```
int main()
{
    int A[3][2];
    int *ptr, i, j;
    ptr = A;
    for(i=0; i<3; i++)
    {
        for(j=0; j<2; j++)
            printf("%d", *(p+i*2)+j);
        printf("\n");
    }
}
```

$A_{m \times n}$: A[i][j] = *((p+i*n)+j);

Pointer Arithmatic

1. Addition of an interger const to a pointer
2. Subtraction " " " "
3. Substraction of one pointer from another
4. comparision of 2 pointers

1.
```
int x;
int *p = &x;
p = p + 1;
```

3. one pointer variable can be subtracted from other provided both of them points to objects of the same data type. or elements of same array.

4.
```
Int A[5] = { 11, 10, 9, 21, 5 };
           1021 1025 1029 1033 1037
int *P1, *P2;
P1 = &A[3];
P2 = A + 3;
if (P1 == P2)
        printf ("Same location");
else
        printf ("Different location");
```

3.
```
P1 = &A[3];
P2 = A + 1;
printf ("%d \t %d", P1-P2, *P1-*P2); // 8  11
```

4. Pointer variables can be compared provided both variables point to objects of the same data type.

The comparision can be done for either equality or inequality.
A pointer variable can also be compared with 0 or NULL

The following operations on pointers are not allowed.

1. Addition of 2 pointers
2. Multiplication of a pointer with const
3. Division " " " " "

```
int main()
{
    int A[4] = {10, 11, 9, 7};
    int *ptr[4], i;
    for(i=0; i<4; i++)
        ptr[i] = &A[i];
}
```

Dynamic Memory Allocation (Management)

Dynamic memory management technique permits us to allocate memory space or to release unwanted space during execution or run time. It is the process of allocating or deallocating memory during run time.

Static Memory Allocation	Dynamic Memory Allocation
1. Memory is allocated at compilation time.	1. Memory allocated during execution time.
2. Insertion is not always possible.	2. Insertion is always possible if there is enough free space.
3. Deletion of element waste the memory.	

PAGE NO.
DATE 27 / 11 / 2019

e.g.; `int *ptr;`
`ptr = (int *) malloc (sizeof (int) *20);`

Malloc function reserves multiple blocks of memory of specific size and returns a pointer of type void. If it is sucessful it returns a pointer of the cast type to the allocated memory, else it returns NULL.
The allocated memory will be filled with garbage value.

```
#include <stdlib.h>
int main()
{
    int i, *ptr, size;
    ptr = (int *) malloc (sizeof(int) * 10); size);
    for (i=0; i<10 size; i++)
        scanf ("%d", ptr+i);
    for (i=0; i<size; i++)
        printf ("%d", *(ptr+i));
    return 0;
}
```

WAP to search an element in an array using pointer

```
#include <stdlib.h>
int main()
{
    int i, *ptr, size, n;
    scanf ("%d", &size);
    ptr = (int *) malloc (sizeof(int) * size);
    for (i=0; i<size; i++)
        scanf ("%d", ptr+i);
```

Teacher's Signature

```
scanf("%d", &x)
int ctr = 0;
for (i=0; i<size; i++)
{
        if (*(ptr+i) == x)
        {
                ctr++;
                break;
        }
}
if (ctr != 0)
        printf("\n element found \n");
else
        printf("\n element not found \n");
return 0;
}
```

Calloc()

```
double *ptr = (double *) calloc (sizeof(double)*size);
```

Free()

This function deallocates or releases the allocated memory by malloc or calloc

realloc()

```
ptr = realloc(ptr, newsize);
```

This function allocates a new memory space to the pointer variable & returns a pointer to the 1st byte of the new memory block. The new memory block may or maynot begin at same place as the old one.

```
#include <string.h>

int main()
{
    char *ptr[10],*temp;
    int i,j;
    for(i=0;i<10;i++)
    {
        ptr[i]=(char*)malloc(sizeof(char)*20);
        scanf("%s",ptr[i]);
    }
    for(i=0;i<10;i++)
    {
        for(j=0;j<i+1;j<10;jj++)
        {
            if(strcmp(ptr[i],ptr[j])>0)
            {
                temp=ptr[i];
                ptr[i]=ptr[j];
                ptr[j]=temp;
            }
        }
    }
```

PAGE NO.
DATE : / /20

Structure -

A no. of heterogeneous datatype grouped together to form a structure.

```
Struct  Name_of_structure
{
        data type1  member 1;
        data type2  member 2;
              :
              :
};
```

e.g.,

```
struct complex_NO
{
    int real;
    int img;
};  complex_NO;
```

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

The C programming language having keyword called 'typedef' which can be used to give a type a new name.

WAP to create a new datatype complex_No using structure & add 2 complex no.

```
#include<stdio.h>
typedef  struct
{
    int real;
    int img;
} Complex_NO;
```

```
int main()
{
    Complex_No C1, C2, C3;
    "Enter 2 complex nos"; scanf("%d%d", &C2.real, &C2.img);
    scanf("%d%d", &C1.real, &C2.img);
    C3.real = C1.real + C2.real;
    C3.img = C1.img + C2.img;
    printf("%d + %di", C3.real, C3.img);
}
```

Runtime or dynamic initialisation

A structure variable can be copied to another structure variable.

```
Complex_No C1 = {2,3};
Complex_No C2;
C2 = C1;
```

WAP to multiply 2 complex nos using structure and datatype Complex_No.

```
typedef struct
{
    int real;
    int img;
} Complex_No;

int main()
{
    Complex_No C1 = {2,3};
    Complex_No C2 = {5,4};
    Complex_No C3;
    C3.real = C1.real * C2.real - C1.img * C2.img;
    C3.img = C1.real * C2.img + C1.img * C2.real;
```

Teacher's Signature

WAP to ~~the~~ create a new datatype student having members roll no, name, address (city, Pincode) marks in phys, marks in maths, marks_C

```
#include<stdio.h>
typedef struct
{
    int empno;
    char Name[30];
    float sal;
    char cat;
}emp;

int main()
{
    int i;
    emp E[20];
    Enter 20 employee name no, name, salary, cat
    for(i=0; i<20; i++)
    {
        scanf("%d", &E[i].empno);
        scanf("%[^\n]s", E[i].Name");
        scanf("%f", &E[i].sal);
        scanf("%c", &E[i].cat);
    }
    int max, MAX;
    max = E[0].empno;
    for(i=01; i<20; i++)
    {
        if(E[i].empno
```

PAGE NO.
DATE : / / 20

```
// print the employee with max. salary
MAX = E[0].sal;
for (i=0; i<20; i++)
{
    if (E[i].sal > MAX)
    {
        max = i;
    }
}
printf (" employee number = %d ", E[max].empno);
printf ("employee name = %s ", E[max].Name);
printf (" employee salary = %f ", E[max].Sal);
printf (" employee cat = %c ", E[max].Cat);

// total amount paid to all employee
float sum = 0
for (i=0; i<20; i++)
    sum += E[i].Sal;
printf (" \n total amount = %f ", sum);
```

```
#include<stdio.h>
typedef struct
{
    int PIN;
    char CITY[50];
}ADD;

typedef struct
{
    int Roll;
    char Name[30];
    ADD Addr;
    float M_Phy;
    float M_Math;
    float M_Cprg;
}student;

int main()
{
    Student S1;
    scanf("%d %s", &S1.Roll, S1.Name);
    scanf("%d %s", &S1.Addr.PIN, S1.Addr.CITY);

    Student S2={ 1,"xyz", 751001, "BBSR", 80, 90, 99.
```