BROUGHT TO YOU BY

https://littlehelp3.wixsite.com/home

YAHA POORA STUDY MATERIAL MILEGA

COMPUTER PROGRAMMING WITH C++

PAGE NO.:
DATE:

languages

- High level language
  - C/C++, JAVA, VB.NET, COBOL, PHP, Pascal
- low level language
  - Machine lang.
  - Assembly lang.

Compiler

Computer = Electronic Machine

Software = cannot be touched but can be felt
ex: chrome, VLC player, Power point, etc.

Programs - Planned series of events

Hardware → That we can feel and touch
processor, floppy disk, RAM

- Reasons of learning programming
  - → coding develops structured & creative ~~thing~~ thinking
  - → Programming makes things easier for you.
  - → learning to program teaches you persistence

String → Collection of Character console : Output window

Rajshree
PAGE NO.:
DATE:

* Write a program to print your name.

```
# include <iostream.h>
# include <conio.h>
void main ()
{
    clrscr();
    cout << " Name ";
    getch();
}
```

Return type → void

→ function

compile Alt + F9
Run Ctrl + F9

clrscr() → clear screen

getch() → to hold on output window

* ~~12 it~~ WAP to add two integer no.

values are fixed.

```
# include <iostream.h>
# include <conio.h>
void main ()
{
    clrscr();
    int x, y, z;
    x = 20;
    y = 30;
    z = x+y;
    cout << z;
    getch();
}
```

← variable declaration.

Rajshree
PAGE NO.:
DATE:

For your choice

```
# include <iostream.h>
# include <conio.h>
void main()
{
 clrscr();
 int x, y, z;
 cout << " Enter 2 no";
 cin>> x>>y;
 z = x+y;
 cout << z;
 getch();
}
```

* Write a program to divide two integer no.

```
# include <iostream.h>
# include <conio.h>
void main ()
{ clrscr();
  int x, y; float z;
  cin >> x >> y;
  z = x/y
  cout << z;
  getch();
}
```

wrong

```
# include <iostream.h>
# include <conio.h>
void main ()
{ clrscr();
  int x, y; float z;
  cout << " Enter 2 no";

  z = (float) x/y;
  cout << z;
  getch();
}
```

← type casting.

* WAP to calculate area of triangle

```
# include <iostream.h>
# include <conio.h>
int main()
{ clrscr();
  int b,h;
  float Ar;
  cout <<" Enter base and height of triangle";
  cin >> b>>h;
  Ar = 1/2.0 * b * h;  or  1.0/2 * b * h;  or  b * h * 1/2;
  cout<<" Area of triangle is" << Ar;
  getch();
  return 0;
}
```

* WAP to calculate Area and Circumference and diameter of a circle:

```
# include <iostream.h>
# include <conio.h>
void main()
{ clrscr();
float area, circ, radius, dia;
cout <<" Enter radius of circle";
cin >> radius;
circ = 3.14 * 2 * radius;
cout <<" Circumference is" << circ;
area = 3.14 * radius * radius;
```

Rajshree
PAGE NO.:
DATE:

```
cout << " Area of circle is" << area;
dia = radius * 2;
cout << " Diameter of circle is" << dia;
getch();
}
```

* WAP to tech value in gallen by user & convert it into cubic foot.

7.41 gallen = 1 $(foot)^3$

g = gallen, x = cubic foot

```
# include <iostream.h>
# include <conio.h>
void main()
{   clrscr();
    float g, x;
    cout << " Enter value in gallen";
    cin >> g;

    x = g/7.418;

    cout << " value you enter, in cubic foot it is" << x;
    get ch();
}
```

Rajshree
PAGE NO.:
DATE:

* WAP to swap two integers; (using third variable)

```
#include <iostream.h>
#include <conio.h>
void main()
{   clrscr();
    int x, y, z;
    x = 20;
    y = 30;
    z = x;
    x = y;
    y = z;
    cout<<" now the value of x and y is "<<x<<", "<<y;
    getch();
}
```

WAP to ~~st~~ swap two integers no. without using third variable (using + operator).

```
#include <iostream.h>
#include <conio.h>
void main()
{
    clrscr();
    int x, y;
    cout<<" Enter any 2 values";
    cin>>x>>y;
    y = x + y;
    y = x - y;
```

Rajshree
PAGE NO.:
DATE:

```
x = x-y;
cout << " values after swapping" << x << ", " << y;
getch(); }
```

* WAP to find questient & remainder of two integers

```
# include <iostream.h>
# include <conio.h>
void main()
{ clrscr();
  int a, b, r, q;
  cout << " Enter dividend = " << endl;
  cin >> b;
  cout << " Enter divisor=" << endl;
  cin>> a;
  r = b % a;
  q = b/a;
  cout << " Quotient is" << q << endl;
  cout << " Remainder is" << r << endl;
  getch();
}
```

* WAP to print days corresponding to no.

```
# include <iostream.h>
# include <conio.h>
void main()
{ clrscr();
  int n;
  cout << " Enter any no";
  cin >> n;
  if (n==0)
    cout << "sunday";
  else if (n==1)
    cout << " Monday";
  else if (n==2)
    cout << " Tuesday";
  else if (n==3)
    cout << " Wednesday";
  else if (n==4)
    cout << " Thursday";
  else if (n==5)
    cout << " friday";
  else if (n==6)
    cout << " Saturday";
  else
    cout << " Invalid entry";
  getch();
}
```

Rajshree
PAGE NO.:
DATE:

- Coding for other than windows operating system (like Ubuntu)

```
# include <iostream>
using namespace std;
int main ()
{
    :
    :
    :
```

Q.) What is datatypes, operators & rules for variable declaration.

> Data types: A data type in programming, is a classification that specifies which type of value a variable has and what type of mathematical, relational or logical operations can be applied to it without causing an error.

for ex: String is a data type that is used to classify text and an integer is a datatype used to classify whole no.

> Operators: An operator is a symbol that tells compiler to perform specific mathematical or logical manipulations, c++ is rich in built-in operators and provide the following type of operators:

> Arithmetic operators: +, -, %, *, ++, --, /

> Relational Operators: ==, !=, >, <, >=, <=

> Logical operators: && , ||, !

> Assignment operators: =, +=, -=, *=, /=, %=, <<=, >>=, &=, ^=, |=

> Bitwise operators: &, |, ^, ~, <<, >>

Rules for variable declaration:

- Every variable name start with alphabet or underscore.
- No spaces are allowed in variable declaration.
- Except underscore no special symbol are allowed in the middle of variable name.
- Maximum length of variable is 8 character, depends on compiler and operating system.
- Every variable name always should exist in the left hand side of assignment operator
- No keyword should access variable name
- Always define the type of data at its declaration time.

* 1 byte = 8 bits

- $\boxed{\text{Range} = 2^n - 1}$ where n = no. of bits.

> Integer
- short int → 1 byte = 8 bits
- int → 2 byte = 16 bits
- long int → 4 byte = 32 bits

> float
- float → 4 byte = 32 bits
- double → 8 byte = 64 bits
- long (double) → 10 byte = 80 bits

> Character → char → 1 byte = 8 bits

DATE:

Q). find the range of int

int = 2 bytes = 16 bits

$n = 16$

Range $= 2^n - 1 = 2^{16} - 1 = 65536 - 1 = 65535$

Range $= \frac{65535}{2} = 32767.5$

Range $= -32767.5 \text{ ------ } +32767.5$

* ASCII

A/65 = 10000001 a/97

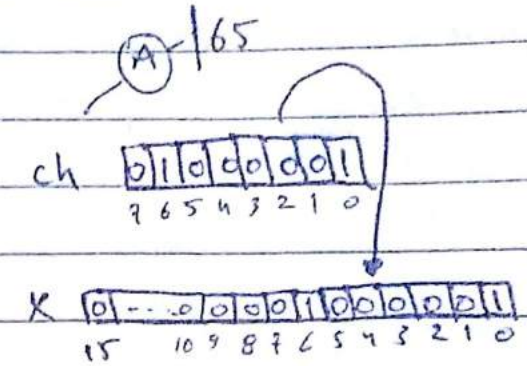
A → 65 + 32 = a → 97

A/65	a / 97	0 / 48
B/66	b / 98	1 / 49
⋮	⋮	⋮
Z/90	z / 122	

@ / 64 , <Enter> / 13 or 8 , <space> 32

1. • WAP to find out ASCII value of any character.

I/P	O/P
A	65
@	64
⋮	



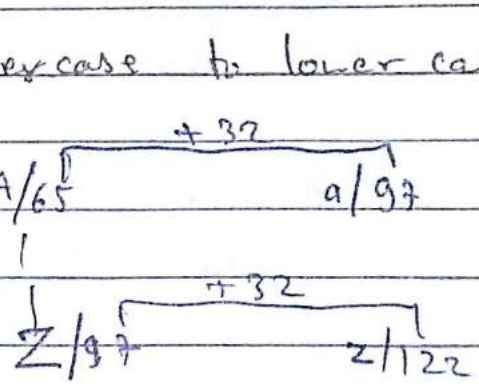

```
char ch;        int x;

cout << " enter any character";
cin >> ch;
   x = ch;
cout << x;
```

• WAP to find out the character corresponding ~~to~~ ASCII no.
→ next page.

• WAP to convert a character from uppercase to lower case.

I/P	O/P
A	a
Z	z
@	Invalid I/P




```
char ch;
cout << "enter any charachter";
cin >> ch;
if ( ch >=65 && ch <=90)
{ ch = ch+32;
  cout << ch; }
else
  cout << "Invalid I/P";
```

Rajshree
PAGE NO.:
DATE:

* WAP to find out the character corresponding ASCII no.

```
# include <iostream>
using namespace std;
int main ()
{
  char ch;
  int x;
  cout << " Enter any no to find its corresponding character";
  cin>>x;
  ch = x;
  cout << ch;
}
```

* WAP to print your name ten times;

```
# include <iostream>
using namespace std;
~~void~~ int main()
{ cout << "Ayur" << endl;
  cout << "Ayur" << endl;
  cout << "Ayur << endl;
  cout << "Ayur" << endl;
  cout << "Ayur" << endl;
  cout << "Ayur" << endl;
  cout << "Ayur" << endl;
  cout << "Ayur" << endl;
  cout << "Ayur" << endl;
  cout << "Ayur" << endl;
  cout << "Ayur" << endl;
}
```

LOOP

while loop

initialization;

while (condition)

{

increment/decrement

}

with help of loop

* WAP to print your name 10000 times;

LOOP

while loop

```
# include <iostream>
using name space std;
int main()
{
    int i=1;
    while (i<=10)
    { cout << "Ayur";
      i++;
    }
```

output.

AyurAyurAyurAyur- - - - - Ayur

(10 times)

* WAP to print a series 1, 2, 3, ... 10

```
#include <iostream>
using namespace <iostream>
int main()
{   int i = 1;
    while ( i <= 10)
      cout << i;
      i++;
}
```

Output

~~1, 2, 3, 4~~

1 2 3 4 5 6 7 8 9 10

* WAP to print following series

(i) 2, 4, 6, 8, ... 128

(ii) 1, 3, 5, 7, 9 ..... n^{th}

(iii) 0, 1, 1, 2, 3, 5, 8, ... n^{th}

(iv) 2, 4, 3, 6, 5 ..... n^{th}

- while loop → checks condition first

- do-while loop → checks condition at last & also in this there is ; after condition

```
initialization;
do
{
  ____
  ____
  ____
  increment/decrement;
}
while (cond);
```

→ basically do-while loop is use with switch

4 characters WAP to ~~Rout~~ Enter a character and print next four characters

DATE:

```
#include <iostream>
using namespace std;
int main()
{
 clrscr();
 char ch;
 int X;
 cout <<"enter any alphabet"<<endl;
 cin>> ch;
 if(ch > 64 && ch <91 || ch> 96 && ch<123)
{  X = ch;
   ch = X+1;
   cout << "next four ~~characters~~ alphabets are" <<endl;
   cout << ch;
   ch = X+2;
   cout <<endl<<ch;
   ch =X+3;
   cout <<endl <<ch;
   ch = X+4;
   cout <<endl <<ch; }
   else { cout <<"invalid input"; }
   getch(); }
```

a — z

ch = = 90

a j

ch = = 'w' && c

= = 'k'

ch = = 'A'

ch = = 'a'

* ARRAY

Array is a collection of similar data types

→ Primitive

Primitive		Non primitive datatype	
int x size 2 type - integer	fixed by system	int x [10] size type	by user

int x [10]

2 byte x 10 = 20 byte

~~int [10]~~

Rajshree
PAGE NO:
DATE:

* WAP to find max value from any 3.

```
#include <iostream>
using namespace std;
int main()
{
int x, y, z, max;
cout << " Enter 3 no. ";
cin >> x >> y >> z;
max = x;
if (y > max)
    max = y;
if (z > max)
    max = z;
cout << "\n" << " the largest of " << x << " , " << y << " and " << z << " is " << max;
return 0;
}
```

output:

```
Enter three numbers : 4
5
7

the largest of 4, 5 and 7 is 7
```

Note if there is one single statement inside if then if we don't apply { } then also it works

✓ if ( )
{ cout << .....
}

✓ if ( )
cout << .....

✓ if ( )
{ cout << ___
cout << ___
}

error.
✗ if ( )
cout << ___
cout << ___

* **L-Value** ← a type of error

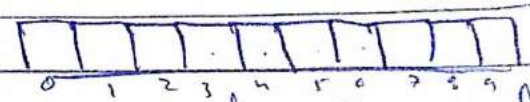
$\underline{\underline{6}} = 6 + 1$

↓

const. never present here, always a variable present.

If const present, then it is type of error, called L-value

Binary Search → search from centre



no. arranged in asc. or desc. order

$$M = \frac{LB + UB}{2}$$

```
while (LB <= UB)

if (search == X[MB]
    cout<<" found";

    else if (search> X [MB]
            LB = MB+1;
    else
        search < X[MB];
            UB = MB-1;
```

search	LB	UB	MB
20	0	9	4
8	0	3	1
88	5	9	7

PAGE NO.:
DATE:

* WAP to take 12 integers no. by user, arrange all the no. in a 3 x 4 matrix & print the matrix to user.

```
# inclue <iostream>
using namespace std;
int main();
{
  int X [3][4], i, j ;

for (i=0 ; i<3 ; i++)
{
  for (j=0; j<4 ; j++)
  {
    cout << " Enter any no";
    cin >> X [i][j];
  }
}
  for ( i=0 ; i<3 ; i++)
  {
    for ( j=0 ; j<4 ; j++)
    {
      cout << X[i][j];
    }
    cout <<"\n";
  }
return 0;
}
```

* WAP to do ~~inverse~~ transpose a matrix (Here always $i = j$)

$$\begin{bmatrix} 00 & 01 & 02 \\ 10 & 11 & 12 \\ 20 & 21 & 22 \end{bmatrix} \longleftrightarrow \begin{bmatrix} 00 & 10 & 20 \\ 01 & 11 & 21 \\ 02 & 12 & 22 \end{bmatrix}$$

```
# include <iostream>
using namespace std;
int main()
{
  int x [3][3] ; i , j ,

  cout << x [j] [i];
```

DATE:

* WAP to calculate sum of Diagonal element 3 x 3 matrix.

$$\begin{bmatrix} 00 & 01 & 02 \\ 10 & 11 & 12 \\ 20 & 21 & 22 \end{bmatrix}$$

i j
0 0
1 1
2 2

$\boxed{i == j}$

```
#include<iostream>
using namespace std;
int x[3][3], i, j, sum = 0;
for (i = 0; i < 3; i++)
{
  for (j = 0; j < 3; j++)
  {
  cout << " Enter 9 elements ";
  cin >> x[i][j];

if (i == j)
  {
    sum = sum + x[i][j];
  }
  }
}
```

Rajshree
PAGE NO.:
DATE:

- WAP to calc. sum of UTM (upper Δ matrix) & LTM (lower Δ matrix) of 3×3 matrix

$$\begin{bmatrix} 00 & 01 & 02 \\ 10 & 11 & 12 \\ 20 & 21 & 22 \end{bmatrix}$$

UTM i j	LTM
01 ~~02~~ ~~12~~	10
02	20
12	21
→ $\boxed{i<j}$	→ $\boxed{i>j}$

Functions

- Function is a collection of statement that perform a ~~cho~~ coherent task of some kind.

funct^n are divided in 2 categories:

(a) User defined funct^n
ex:- ayur();

(b) System defined | Pre-defined | Built-in | Library funct^n
ex: clrscr(); getch()

Note: main() → declare by system & defined by user.

```
# include<iostream.h>
# include <conio.h>
void show();          // funct^n declarat^n
void main()
{
cout << "Hello";
show();               // funct^n calling
getch();
}
void show()           // funct^n definat^n
{
cout << "In show";
}
```

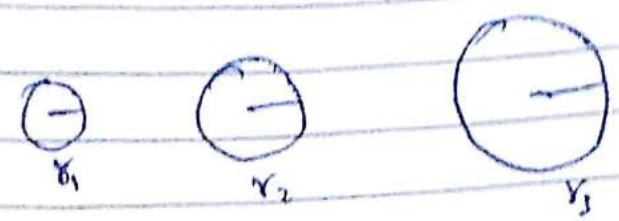
declarat^n & defining can be together

calling is only within main

Hello In show

* OBJECT ORIENTED PROGRAM

Any object is real world entity and it have two things
(a) Attributes
(b) Behaviour : depends upon attributes.



r → attribute
behaviour → diameter, area, circumferance } depends up on radius ie attribute.

Parameter | variable | Data Member | Attributes.
Function | operation | Method | Behaviour | member funct^n.

Class is user defined data type that encapsulate the data member and member function in a single unit.

ex :-
```
class Circle
{ private:
    float r;
  public:
    get-area();
    get_dia();
    get - circum();
};
```

(data member) → float r;
+
(member funct^n) → get-area(); get_dia(); get - circum();

Rajshree
PAGE NO.:
DATE:

②. Concept of OOP's :

(a) Encapsulation
(b) ~~Ih~~ Inheritance
(c) Polymorphism
(d) Abstraction

- **Encapsulation** :

  Wrapping up of data ie wrapping up a data member and a method into a single unit

- **Aheritance** :

  When a class includes property of another class in it.

- **Polymorphism** : Many forms of single object

- **Abstraction** :

  To represent the essential feature without representing background details.

  i.e.

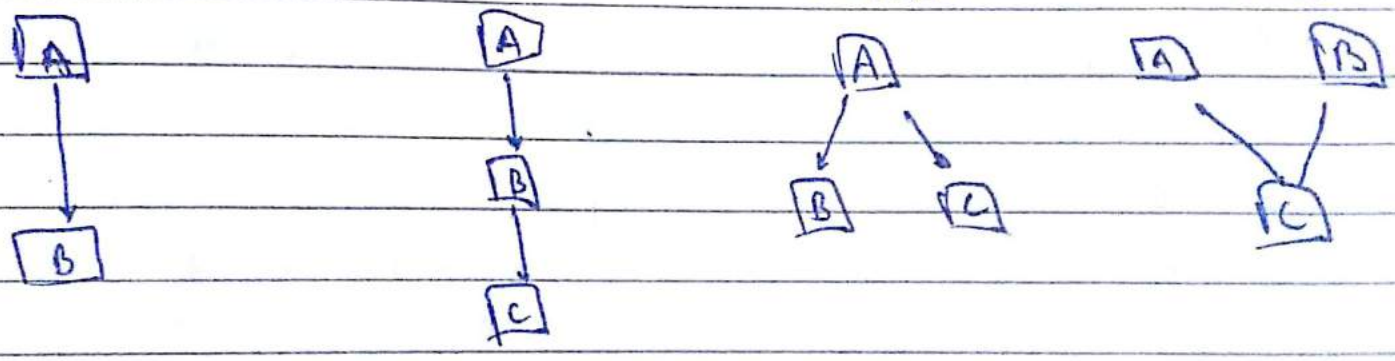
  Hiding the non essential ~~or &~~ data or hiding the internal process of ~~p~~ in running the program.

```
void main()
{
    Rectangle R1, R2 (1.2, 2.1);
    cout << R1.get_area();
    cout << R1.get_perimeter();
    cout << R2.get_area();
    cout << R2.get_perimeter();
    return 0;
}
```

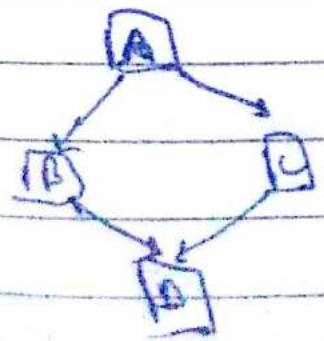
INHERITENCE

Deriving a new class of old one or existing one, that concept is known as inheritance

- Single Inheritance
- Multi level inheritence.
- Hirarchy Inheritence
- Multiple Inheritence



- Hybrid inheritence



Object always creates of derived Class

Rajshree
PAGE NO.:
DATE:

* Design a Class for multi level inheritance.

* what is the ~~ada~~ advantage of inheritance.

Advantages:

⇒ Reusability : (oops concept) facility to use public methods of base class without rewriting the same.

Extensibility : Extending the base class w/o rewriting the same

Data hiding : Base class can decide to keep some data private so that it cannot be altered.

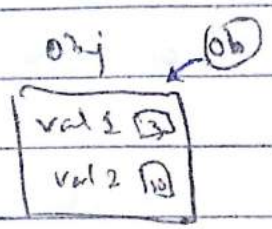
Overriding : With inheritance, we will be able to override the methods of base class so that meaningful implementation of base class method can be designed in derived class.

PAGE NO.:
DATE:

Friend function

Example.

```
class Base
{
  int val1, val2;
  public :
     void get()
     {
       cout << "Enter 2 no:" ;
       cin >> val1 >> val2;
     }
  friend float mean (Base ob);
};
  float mean (Base ob)
  {
     return( ob. val1 + ob.val2)/2;
  }
void main()
{  Base obj;
   obj . get();
   cout << " mean is" << mean (obj);
}
```



```
void D:: mul()
{
    c = b * get_a();
}
void D:: display()
{
    cout << "a = " << get_a() << "\n";
    cout << "b = " << b << "\n";
    cout << "c = " << c << "\n";
}

int main()
{
D d;
d.get_ab();
d.mul();
d.show_a();
d.display();
d.b = 20;
d.mul();
d.display();
return 0;
```

a = 5
a = 5
b = 10
c = 50

a = 5
b = 20
c = 100

we can pass the object as a parameter in the method

```
class height
{
int feet, inches;
public:
void getht (int f, int i)
{
feet = f,
inches = i;
}
void putheight()
{
cout << "\n Height is :" << feet << "feet \t" << inches << "inches"
<<endl;
}
void sum (height a, height b)
{
height n;
n.feet = a.feet + b.feet
n.inches = a.inches + b.inches;
if (n.inches >= 12)
{
n.feet ++;
n.inches = n.inches - 12;
}
cout << endl << "Height is" << n.feet << " feet and " << n.inches << endl;
}
};
void main()
```

* default constructor
* parameterized constructor
* copy constructor

```
{
height h,d,a;
clrscr();
h.getht(6,5);
a.getht(2,7);
h.putheight();
a.putheight();
d.sum(h,a);
getch();
```

/*** Output ***/

Height is : 6 feet
5 inches

Height is : 2 feet
7 inches

Height is 9 feet and 0 inches

* Static Data Member

a single memory created

> if we declare a member static, then all objects shares that one memory

> If we declare member defaul then for every object there is diff memory of that member

> Only one copy of that member is created for the entire object and shared by all the objects of that class no matter how many objects are created

PAGE NO.:
DATE:

* Static member function

- A static funct^n can have access to only other static members (funct^n or variable) declares in the same class
- A static member funct^n can be called using the class name (instead of its object)
- Classname :: function_name;

```
class Something
{
  private:
      static int s_n value;
  public:
      static int GetValue()
      {
        return s_n value;
      }
};
int Something :: s_n value = 1; // initializer
void main()
{
  cout << Something :: GetValue();
}
```

As array same as str

Rajshree
PAGE NO.:
DATE:

* Selection sort

5 ← let min

4 min > a[j] loop = i = 0

3 5 > 4

2 true

min = 4 loop = i

1 4 > 3

true

min = 3

3 > 2

true

min = 2

2 > 1

true

min = 1

1. Explain what is the oops features?

Answer: The object oriented programming has been developed with a view to overcome the drawbacks of conventional programming approaches.

The General features of OOP are given below:

→ Data Abstraction

Abstraction is the concept of simplifying a real world concept into its essential elements. or we can say abstraction refers to the act of representing essential features without including the background details or explanations

To understand abstraction, let us take an example. One driving a car. He only know the essential features to drive a car e.g., gear handling, steering handling, use of clutch, accelerator, brakes etc. But while driving he didn't get into internal details of car like wiring, motor working etc. He just change gears or apply brakes etc. What is happening inside is hidden from him. This is abstraction where we get essential features without including background details or explanations.

Scanned by CamScanner

→ Data Encapsulation

Encapsulation is the most fundamental concept of OOP. It is the way of combining both data and the functions that operate on that data under a single unit. Or we can say the wrapping up of data and operations/functions into a single unit (called class) is known as encapsulation.

The only way to access the data is provided by the functions. These functions are called member functions in C++. The data cannot be accessed directly. If we want to read a data item in an object, we call a member function in the object. It will read the item and return the value to you. We can't access the data directly. The data is hidden, so it is safe from accidental alteration. Data and its functions are said to be encapsulated into a single entity.

Let us take an example, in a big company there are so many departments like sales, Accounts, Purchase, Production etc. Each department has its own personnel that maintains its data. If an employee in production dept. wants to know how

Scanned by CamScanner

much raw material purchased for the next month, ~~the~~ he will not be allowed to himself go through purchase dept. data files. Rather he'll have to issue a memo to the 'purchase' requesting for the required information. Then some employee of 'purchase' dept. will go through purchase data files and send reply with asked information. This practice ensures that data is accessed accurately and that it is not corrupted by inept outsiders. Therefore, we can say here 'Department data and department employees are encapsulated into a single entity, the department'.

→ Modularity

Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules. Or the act of partitioning a program into individuals components is called modularity.

The justification for partitioning a program:

(i) It reduces its complexity to some degree

(ii) It creates a no. of well-defined, documented boundaries within the program

Scanned by CamScanner

A module is a separate unit in itself. It can be compiled separately though it has connections with other modules. Modules work hand in hand in order to achieve the programs goal.

For example we must have seen a complete music system, let us assume that our program represents the music system. The music system comprises of speakers, cassette-player, record-player, cd-player etc. Now, these parts are complete units in themselves, yet they are a subpart of music system. This is modularity.

In object oriented languages, classes and objects form the logical structure of a system. We place them in modules to produce system's physical architecture

- Class : A class is a group of objects that share common properties and relationships. or we can say a class is a way to bind the data describing an entity and its associated functions together. Class represents a group of similar objects.

Scanned by CamScanner

• objects: An object is an identifiable entity with some characteristics and behaviorsr.

For instance, we can say 'orange' is an object. Its characteristics are: it is spherical shaped, its colour is orange etc. Its behaviour is: it is juicy and it tastes sweet-sour. While programming using OOP approach, the characteristics of an object are represented by its data and its behaviour is represented by its functions associated. Therefore, in OOP programming object represents an entity that can store data and has its interface through functions.

→ Inheritance

Inheritance is the capability of one class of things to derive capabilities or properties from another class.

Understanding Inheritance is critical to understanding the whole point behind object oriented programming.

For instance, we are humans. We inherit from the class 'Human' certain properties, such as ability to speak, breathe, eat, drink etc. etc. But these properties are not unique to humans.

Scanned by CamScanner

The class 'Human' inherits these properties from the class 'Mammal' which again inherits some of its properties from another class 'Animal'.

→ Polymorphism

Polymorphism is the ability for a message or data to be processed in more than one form. Polymorphism is a property by which the same message can be sent to objects of several different classes, and each object can respond in a different way depending on its class. Polymorphism is key to the power of object oriented programming. It is so important that languages that don't support polymorphism cannot advertise themselves as OO language.

For instance, 'Human' is a subclass of 'Mammal'. Similarly 'dog', 'cat', are also subclasses of 'mammal'. Mammals can see through day-light. So if a message 'see through day light' is passed to all mammals, they all will behave alike. Now if a message 'see through darkness' is passed to all mammals then humans and dogs will not be able

Scanned by CamScanner

to view at night whereas cats will be able to view during night also. Here cats (mammals) can behave differently than other mammals in response to a message or action. This is polymorphism.

2. What is copy constructor? Illustrate using a suitable code.

Answer. A copy constructor is a constructor of the form classname (classname &). The compiler will use the copy constructor whenever you initialize an instance using values of another instance of same type.

For example,

```
Sample S1;          //default constructor
Sample S2 = S1;     // copy constructor
```

In the above code, for the second statement, the compiler will copy the instance S1 to S2 member by member. If ~~you~~ we have not defined a copy constructor, the compiler automatically creates it and it is public.

We can define our own copy constructor also. A copy constructor takes a reference to an object of the same class an argument.

Scanned by CamScanner

Consider the following class definition

```
class Sample{ int i, j;
        public:
            Sample(int a, int b)      //constructor
            {i=a; j=b;}
            Sample(Sample & s)
            {j=s.j; i=s.j;}           // copy
                                        constructor
            cout<<"In copy constructor
                  working \n";
            }
            void print(void)
            {cout<<i<<j<<"\n";}
            :
            :
        };
```

The above definition defines two constructors: one parameterized constructor for when initial values are provided and the other a copy constructor to copy data values of a Sample object onto another.

These constructors may be used as follows:

```
Sample S1(4,6);      // S1 initialized first constructor
Sample S2(S1);       // S1 copied to S2. Copy constructor called
Sample S3 = S2;      // S2 copied to S3. Copy
                        constructor called again
```

But if we give an assignment statement a
s4 = s1;
it will not invoke the copy constructor.
However, if s4 and s1 are objects of same type, this statement is legal and simply assigns values of s1 to s4, member-by-member.

The process of initializing through a copy constructor is known as copy initialization

3. How many types of constructors are exists in a C++?

ver. A constructor is a member function with the same name as its class is called constructor.

There are mainly three types of constructors:
- Default Constructor
- Parameterized Constructor
- Copy Constructor

→ Default Constructor

A constructor that accepts no parameters is called the default constructor.

```
class X { int i;
        public:
            int j,k;
            X();
        };
```

Scanned by CamScanner

here X() is default constructor, since no parameters are here. With a default constructor, objects are created just the same way as variables of other data types are created.

for instance,

X Ob1;

will create the object Ob1 of type X by invoking the default constructor.

When a user-defined class does not contain an explicit constructor, the compiler automatically supplies a default constructor, having no arguments. This implicity-declared default constructor is an inline public members of its class

The default constructor provided by the compiler does not do anything specific. It simply allocates memory to data members of objects.

→ Parameterized constructors

A constructor that accepts parameters for its invocation is known as parameterized constructor. Like other functions we can also write constructors that can accept parameters. Such constructors

Scanned by CamScanner

are called parameterized constructors (or regular constructors).

Once we declare a constructor with arguments, the default constructor becomes hidden. After this ~~you~~ we cannot invoke the default constructor.

This means that we must always specify the arguments whenever we declare an instance (object) of the class. Consider the following code snippet.

```
class Test{ int ant;
            public:
                Test (int i)      //constructor with
                { ant = i;          arguments
                }
                :                 //other members
            };
int main()
{ Test object1(45);
  Test object2;
  :
}
```

There can be default constructor as well as another constructor with arguments for a class but that's a case of multiple constructors which is known as constructor overloading.

Scanned by CamScanner

Just like any other function, a parameterized constructor can also have default arguments.

Consider the following example

```
class A {int i;
        float j;
        public :
            A (int a=0, float b = 1000.0);
                // prototype declaration with
                default arguments other
                members
        };

A :: A (int a, float b)
{   i = a;
    j = b;
}

int main ()
{ A o1 (27, 2715.50);   // ok, Argument values
                           passed for o1
  A o2;
  :                     // ok, takes default argum-
                           ents values. Equivalent to
  :                        O2 (0, 1000.0)
}
```

A constructor with default argument values allows to create objects without any values provided just the same way as default constructor allows us to do.

Scanned by CamScanner

→ Copy Constructor

A copy constructor is a constructor of the form classname (classname &). The compiler will use the copy constructor wherever we initialize an instance using values of another instance of same type.

For example,

```
Sample S1 ;          // default constructor used
Sample S2 = S1;      // copy constructors used
```

In the above code, for the second statement, the compiler will copy the instance S1 to S2 member by member. If you have not defined a copy constructor, the compiler automatically creates it and it is public.

We can define our own copy constructor also. A copy constructor takes a reference to an object of the same class as argument. Consider the following class definition:

```
class Sample{ int i, j;
        public:
            Sample (int a, int b)      // constructor
            {i=a; j=b}
            Sample(sample &S)          // copy constructor
            {j = S.j; i=S.j;
            cout << "\n copy constructor working";
            }
```

Scanned by CamScanner

```
void print (void)
{ cout << i << j << "\n"; }
    :
    :
};
```

Here copy constructor defines to copy data values of a sample object onto another. All the constructors defined above may be used as follows:

```
Sample S1 (4,a);        // S1 initialized first constructor used

Sample S2 (S1);         // S1 copied to S2. copy constructor called

Sample S3 = S1;         // S1 copied to S3. copy constructor called again.
```

But if we give assignment statement as
S4 = S1;
it will not invoke the copy constructor. However, if S4 and S1 are objects of same type, this statement is legal and simply assigns value of S1 to S4, member by member.

Scanned by CamScanner

4. What is recursion?

When a function called by itself, that concept is known as recursion and that function is known as recursive function

```
void recurse ()
{
  ........
  recurse ();
  ........
}
```

recurse() called by itself, hence here recurse() is a recursive function.

Through recursion to make a program is not easy but no. of lines are reduced.

The figure below shows how recursion works by calling itself over and over again.

```
void recurse ()
{
  ........
  recurse ();
  ........ }
int main ()
{ ........
  recurse();
}
```

recursive call

Scanned by CamScanner

The recursion continues until some condition is met.

To prevent infinite recursion, if...else statement (or similar approach) can be used where one branch makes the recursive call and other doesn't.

Ex: Factorial of Number using recursion

```
#include <iostream>
using namespace std;
int factorial (int);
int main ()
{
  int n;
  cout<<" Enter a no to find factorial :";
  cin>>n;
  cout<<" Factorial of "<<n<<" = "<<factorial (n);
  return 0;
}
int factorial (int n)
{
    if (n>1)
    { return n*factorial (n-1);
    }
    else
    { return 1;
    }
}
```

Scanned by CamScanner

Output

Enter a no to find factorial : 4

factorial of 4 = 24

Explanation : How this example works!

```
int main()
{.........
}
int factorial (int num)
  { if(num > 1)
    return num* factorial (num-1);
    else
    return 1;
  }
int factorial(int num)
  { if(num>1)
    return num*factorial (num-1);
    else
      return 1; }
int factorial(int num)
  { if (num>1)
    return num* factorial(num-1);
    else return 1; }
int factorial(int num)
  { if (num>1)
    return num*factorial(num-1);
    else return 1; }
```

4*6 = 24 is returned to main and displ

3*2 = 6 is returned

2*1 = 2 is returned

1 is returned

Scanned by CamScanner

DATE
PAGE

5. What is function overloading? Explain the advantages of function overloading.

Answer: When several function declarations are specified for a single function name in the same scope, the (function) name is said to be overloaded.

Or we can say a function name having several definitions that are differentiable by the number or types of their arguments is known as function overloading.

For example, following two functions are different for C++.

```
float divide(int a, int b);
float divide (float x, float y)
```

That is, divide () taking two int arguments is different from divide () taking two float arguments.

This is known as function overloading.

• Advantages of function overloading:

> Function overloading not only implements polymorphism but also reduces number of comparisons in a program and thereby makes the program run faster.

Scanned by CamScanner

DATE
PAGE

> The use of function overloading is to save the memory space, consistency and readability.

> We can develop more than one function with the same name.

> Function overloading is essential to allow the function name (for example the constructor) to be used with different arguments types.

> In order to stimulate real-world objects in programming, it is necessary to have function overloading.

6. What is importance of friend function in C++? Why we use it? Which problem resolved by friend function.

Answer. The friend function is a 'non-member function' of a class. It can access non public members of the class. A friend function is external to the class definition.

The compiler knows a given function is a friend function by the use of the keyword friend.

Scanned by CamScanner

DATE
PAGE

```
class class-name
{
    .........
    friend return-type function-name(argument
    /s);
    .........
}
```

For accessing the data, the declaration of a friend function should be made inside the body of the class (can be anywhere inside class either in private or public section) starting with keyword friend.

One of the important concepts of OOP is data hiding, i.e. a nonmember function cannot access an object's private or protected data.

But, sometimes this restriction may force programmer to write long and complex codes. So, there is mechanism built in C++ programming to access private or protected data from non-member functions.

This is done using a friend function or and a friend class (not in syllabus)

Scanned by CamScanner

A friend function has following advantages.

1. Provides additional functionality which is kept outside the class.
2. Provides functions that need data which is not normally used by class.
3. Allows sharing private class information by a non member function

Now, we can define the friend function as a normal function to access the data of the class. No friend keyword is used in the definition.

```
class className
{
    ..........
    friend return-type functionName(argument/s)
    ..........
}
return-type functionName(argument/s)
{
    ..........
    // Private and protected data of className
    // can be accessed from this function
    // because it is a friend function of
    // className
    ..... }
```

Scanned by CamScanner

Example:

```
#include <iostream>
using namespace std;
class Distance
{
    private:
        int meter;
    public:
        Distance(): meter(0) { }
                                    // friend function
        friend int addfive (Distance);
};

                    // friend function definition

int addfive( Distance d)
{
    //accessing private data from non-member
    //function
    d.meter += 5;
    return d.meter;
}
int main()
{   Distance D;
    cout << "Distance :" << addfive (D)
    return 0;
}
```

Scanned by CamScanner

Output:
Distance : 5

Here, friend function addfive() is declared inside Distance class. So, the private data meter can be accessed from this function.

7. What is virtual function? Explain by suitable code.

Ans: A virtual function is a member function that we expect to be redefined in derived classes. When we refer to a derived class object using a pointer or a reference to the base class, we can call a virtual function for that object and execute the derived class's version of the function.

Virtual functions are used to support Run time Polymorphism. When the virtual function is called by using a Base class Pointer, the compiler decides at Runtime which version of the function i.e. Base Class version or the overridden Derived class version is to be called.

declaration: virtual return-type functionName( )

Example

```
#include <iostream>
using namespace std;
class Account
{ public:
    Account(double d)
    { balance = d; }
    virtual double GetBalance()
    { return balance; }
    virtual void PrintBalance()
    { cout << " Error. Balance not available
      for base type." << endl;
  private:
    double balance;
  };
class CheckingAccount : public Account
{ public:
    CheckingAccount (double d) : Account(d) { }
    void PrintBalance()
    { cout << " Checking account balance :" <<
    GetBalance()
  };
class SavingsAccount : public Account
{ public:
    SavingsAccount (double d) : Account (d){}
    void PrintBalance ()
    { cout << " Savings account balance :";
};
int main()
```

Scanned by CamScanner

```
{   //create objects of type CheckingAccount
    //and SavingsAccount.
CheckingAccount *pChecking = new CheckingAccount
(100.00);
SavingsAccount *pSavings = new SavingsAccount
(1000.00);

    //call PrintBalance using a pointer to Account
Account *pAccount = pChecking;
pAccount -> PrintBalance();

    //call PrintBalance using a pointer to Account
pAccount = pSavings;
pAccount -> PrintBalance();
}
```

In preceding code the calls to PrintBalance are identical except for the object pAccount points to. Because PrintBalance is virtual, the version of the function defined for each object is called. The PrintBalance function in the derived classes CheckingAccount and SavingsAccount.

If a class is declared that does not provide an overriding implementation of the PrintBalance function, the default implementation from the base class Account is used.

Scanned by CamScanner

DATE
PAGE

8. What is rules for operator overloading in c++? Explain via overload any binary operator

Answer There are certain restrictions and limitations in overloading the operator. Some of them are listed below

(i) Only existing operators can be overloaded New operators cannot be overloaded

(ii) The overloaded operator must have atleast one operand that is of user defined type.

(iii) We cannot change the basic meaning of an operator. That is to say, we cannot redefine the plus (+) operator to subtract one value from the other

(iv) Overloaded operators follow the syntax rules of the original operators. They cannot be overridden.

(v) There are some operators that cannot be overloaded like size of operator (sizeof), membership operator (.), pointer to member operator (.*), scope resolution operator (::), conditional operator (?:) etc

Scanned by CamScanner

(vi) We cannot use "friend" functions to overload certain operators. However, member function can be used to overload them. Friend functions can not be used with assignment operator (=), function call operator (()), subscripting operator ([]), class member access operator (->) etc.

(vii) Unary operators, overloaded by means of a member function, take no explicit arguments and return no explicit values, but, those overloaded by means of a friend function, take one reference argument (the object of the relevant class).

(viii) Binary operators overloaded through a member function take one explicit argument and those which are overloaded through a friend function take two explicit arguments.

(ix) When using binary operators overloaded through a member function, the left hand operand must be an object of the relevant class.

Scanned by CamScanner

(x) Binary arithmetic operators such as +, -, * and / must explicitly return a value. They must not attempt to change their own arguments.

operator overloading syntax:

keyword operator to be overloaded

Returntype Classname :: Operator OperatorSymbol(argument list)

{

// function body

}

Example: Overloading '+' operator to add two time object.

```
#include <iostream.h>
# include <conio.h>
class time
{
int h,m,s;
public:
time ()
{
h=0; m=0; s=0;
}
void getTime();
```

Scanned by CamScanner

```
void show()
{
  cout<<h<<":"<<m<<":"<<s;
}
time operator +(time);      //overloading '+' operator
};
time time::operator + (time t1)   //operator function
{
  time t;
  int a,b;
  a = s+t1.s;
  t.s = a%60;
  ~~t.h~~
  b = (a/60) + m + t1.m;
  t.m = b%60;
  t.h = (b/60) + h + t1.h;
  t.h = t.h % 12;
  return t;
}
void time::gettime()
{
  cout <<"\n Enter the hour (0-11)";
  cin >>h;
  cout <<"\n Enter the minutes (0-59)";
  cin >> m;
  cout <<"\n Enter the seconds (0-59)";
```

Scanned by CamScanner

```
cin>>s;
}
void main()
{
clrscr();
time t1, t2, t3;
cout<<"\n Enter the first time";
t1.gettime();
cout<<"\n Enter the second time";
t2.gettime();
t3=t1+t2;        //adding of two time
                   object using '+' operator
cout<<"\n First time";
t1.show();
cout<<"\n Second time";
t2.show();
cout<<"\n sum of times";
t3.show();
getch();
}
```

9. **Sorting:**

Sorting of an array means arranging the array elements in a specified order i.e either ascending or descending order.

Scanned by CamScanner

DATE
PAGE

- Selection Sort

One family of internal sorting algorithms is the group of selection sorts. The basic idea of a selection sort is to repeatedly select the smallest key in the remaining unsorted array.

Let us, we sort a given array with selection sort

→ General concept

I	15	6	13	22	3	52	2	→ smallest
II	15	6	13	22	3	52	2	
III	2	6	13	22	3	52	15	→ smallest
IV	2	3	13	22	6	52	15	
V	2	3	6	22	13	52	15	
VI	2	3	6	13	22	52	15	
VII	2	3	6	13	15	52	22	
VIII	2	3	6	13	15	22	52	

sorted.

→ Exchange Selection Sort

Scanned by CamScanner

> Algorithm

```
1. small = AR[L]        /* Initialise small with
                           first array element */

2. for I = L TO U do    //I=L++, o To size-1
   {
3. small = AR[I], pos = I   /* Loop to find
                               smallest element and
                               its position */

4. for J = I to U do
   {
5.    If AR[J] < small then
6.    { small = AR[J]
7.        pos = J
      }
      J = J+1
   }                        /* end of inner loop */
   /* swap the smallest element with I^th element */
8. temp = AR[I]
9. AR[I] = small
10. AR[pos] = temp
   }                        /* end of outer loop */
11. END
```

> Program for Selection Sort in array

Scanned by CamScanner

```
# include <iostream.h>
void SelSort (int[ ], int);        //function for selection Sort
int main ()
{ int AR[50], ITEM, N, index;      //array can hold max 50 elements
cout<<"How many elements do U want to
        create array with ? (max.50)...";
cin>>N;
cout<<"\n Enter Array Elements...";
for(int i=0; i<N; i++)
  cin>> AR[i];
SelSort (AR,N);
cout<<"\n\n The Sorted array is as shown
        below...\n";
for(i=0; i<N; i++)
  cout << Ar[i] << "  ";
cout << endl;
return 0;
}
void SelSort(int AR[], int size)   //function to perform selection sort
{
 int small, pos, tmp;
for (int i=0; i<size-1; i++)
{ small= AR[i];
   pos =i;
for (int j=i+1; j<size; j++)
 { if (AR[j] < small)
     { small= AR[j]; pos = j; }
```

Scanned by CamScanner

```
}
tmp = AR[i];
AR[i] = AR[pos];
AR[pos] = tmp;
cout << "\n Array after pass-" << i+1 << "-is:";
for(j=0; j<size; j++)
  cout << AR[j] << " ";
}
}
```

OUTPUT

```
How many elements do U want to create array
with? (max. 50) ... 6

Enter Array elements ... 6 8 2 5 3 4
Array after pass-1-is: 2 8 5 6 3 4
Array after pass-2-is: 2 3 5 6 8 4
Array after pass-3-is: 2 3 4 6 8 5
Array after pass-4-is: 2 3 4 5 8 6
Array after pass-5-is: 2 3 4 5 6 8

The sorted array is as shown below...
```

- Bubble sort

The basic idea of bubble sort is to compare two adjoining values and exchange them if they are not in proper order. For instance, following unsorted array is to be sorted in ascending order using bubble sort.

2	no change	2		2	
5		5	swap	4	
4		4		5	no change
13		13		13	no change
17		17		17	no change
19		19		19	no change
24		24		24	

> Algorithm

1. For I = L To U
2. { For J = L To [(U-1) - I] //need not consider already settled heavy elements// //that is why (U-1)-I
3. { if AR[J] > AR[J+1] then /* swap the values */
4. temp = AR[J]
5. AR[J] = AR[J+1]
6. AR[J+1] = temp
   } /* end of if */
   } /* end of inner loop */
   } /* end of outer loop */
7. END

> Program for Bubble Sort in Array.

Scanned by CamScanner

```
#include <iostream.h>
void BubbleSort(int[], int);   //function for Bubble sort
int main()
{int AR[50], ITEM, N, index;   //array can hold max.50 elements
cout<<"How many elements do u want to create array with? (max.50)...";
cin>>N;
cout<<"\n Enter array elements...\n"
for (int i=0; i<N; i++)
  cin>>AR[i];
  BubbleSort (AR,N);
  cout<<"\n\n The Sorted array is as shown below.\n";
for (i=0; i<N; i++)
  cout<< AR[i]<<" ";
cout << endl;
  return 0;
}
void BubbleSort(int AR[], int size)   //function to perform bubble sort
{
int tmp, ctr=0;
for (int i=0; i<size; i++)
{ for (int j=0, j<((size-1)-i); j++)
  { if (AR[j]>AR[j+1])
    { tmp=AR[j];
      AR[j] = AR[j+1];
      AR[j+1] = tmp; }
  }
```

Scanned by CamScanner

```
cout << " Array after iteration - " << ++ctr << " -is :";
for (int k = 0; k < size; k++)
  cout << AR[k] << " ";
  cout << endl;
}
}
```

~~OUTPUT~~

How many elements do u want to create array with ? (max.50) ... 5

Enter Array elements

9 7 4 6 1

Array after iteration -1- is : 7 4 6 1 9

Array after iteration -2- is : 4 6 1 7 9

Array after iteration -3- is : 4 1 6 7 9

Array after iteration -4- is : 1 4 6 7 9

Array after iteration -5- is : 1 4 6 7 9

The sorted array is as shown below..

1 4 6 7 9

- Insertion Sort

Suppose an array A with n elements A[1], A[2], ..., A[N] is in memory. The insertion sort algorithm scans A from A[1] to A[N], inserting each element A[K] into its proper position in the previously sorted subarray A[1], A[2], ..., A[K-1]. That is:

DATE
PAGE

Pass 1. A[1] by itself is trivially sorted

Pass 2. A[2] is inserted either before or after A[1] so that: A[1], A[2] is sorted.

Pass 3. A[3] is inserted into its proper place in A[1], A[2], that is, before A[1], between A[1] and A[2], or after A[2], so that: A[1], A[2], A[3] is sorted.

Pass 4. A[4] is inserted into its proper place in A[1], A[2], A[3], so that: A[1], A[2], A[3], A[4] is sorted.

→ Suppose an Array A contains 8 elements as follows: 77, 33, 44, 11, 88, 22, 66, 55. Sort this array using insertion sort.

Pass	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K=1 :	$-\infty$	77	33	44	11	88	22	66	55
K=2 :	$-\infty$	77	33	44	11	88	22	66	55
K=3 :	$-\infty$	33	77	44	11	88	22	66	55
K=4 :	$-\infty$	33	44	77	11	88	22	66	55
K=5 :	$-\infty$	11	33	44	77	88	22	66	55
K=6 :	$-\infty$	11	33	44	77	88	22	66	55
K=7 :	$-\infty$	11	22	33	44	77	88	66	55
K=8 :	$-\infty$	11	22	33	44	66	77	88	55
Sorted :	$-\infty$	11	22	33	44	55	66	77	88

Insertion sort for n=8 times.

Scanned by CamScanner

DATE
PAGE

> Algorithm

1. A(0) = minimum integer-value /* Now start sorting the array */

2. Repeat steps 3 through 8 for K = 1, 2, 3 .... N-1
{

3. temp = A[K]

4. ptr = K-1

5. Repeat steps 6 to 7 while temp < A[ptr]
{

6. A[ptr+1] = A[ptr] //Moves element forward

7. ptr = ptr-1 //End of inner repeat - step 5's loop
}

8. A[ptr+1] = temp //End of outer repeat - step 2's loop
}

9. END

> Program for insertion sort in array.

```
#include <iostream.h>
#include <limits.h>          // for INT_MIN
void InsSort(int [ ], int);  //function for insertion sort
int main()
{ int AR[50], ITEM, N, index;  //array can hold max. 50 elements

cout<<"How many elements do U want to create array with ? (max. 50) ....";
cin>> N;
cout<<"\n Enter Array elements...";
```

Scanned by CamScanner

```
for(int i=1; i<=N; i++)
  cin>> AR[i];
InsSort(AR,N);
cout<<"\n\n The sorted array is as shown below
      ...\n";
for (i=1; i<=N; i++)
  cout<< AR[i]<<" ";
  cout<<endl;
  return 0;
}
void InsSort(int AR[],int size)  //function to
                                   perform bubble sort
{
 int tmp,j;
 AR[0]=INT_MIN;
 for (int i=1; i<=size; i++)
 { tmp=AR[i];
   j=i-1;
 while (tmp<AR[j])
 {  AR[j+1]=AR[j];
    j--;
 }
 AR[j+1]=tmp;
 cout<<"Array after pass-"<<i<<"-is:";
 for(int k=1; k<=size; k++)
    cout<<AR[k]<<" ";
    cout<<endl;
  }
}
```

Scanned by CamScanner

DATE
PAGE

OUTPUT

How many elements do U want to create array with ? (max 50)... 7
Enter Array elements... 9 4 3 2 3 4 1 6
Array after pass-1-is: 9 4 3 2 34 1 6
Array after pass-2-is: 4 9 3 2 34 1 6
Array after pass-3-is: 3 4 9 2 34 1 6
Array after pass-4-is: 2 3 4 9 34 1 6
Array after pass-5-is: 2 3 4 9 34 1 6
Array after pass-6-is: 1 2 3 4 9 34 6
Array after pass-7-is: 1 2 3 4 6 9 34

The sorted array is as shown below...
1 2 3 4 6 9 34

10. Searching :

- linear Search

In linear search ~~eat~~ each element of the array is compared with the given item to be searched for, one by one. This method, which traverse the array sequently to locate the given item, is called linear search or sequential search

> Algorithm

/* Initialise counter by assigning lower bound value of the array */

1. Set ctr = L // In C++, L (lower bound) is 0 (zero)

2. /* now search for item */

Repeat steps 3 through 4 until ctr > U.
// In C++, U (upper bound) is size-1.

3. IF AR[ctr] == ITEM then
{ print "Search Successful"
print ctr," is the location of", ITEM
break /* go out of loop */
}

4. ctr = ctr + 1 /* End of Repeat */

5. IF ctr > U then
print " Search Unsuccessful!"

6. END

> Program for linear Search in Array

```
# include <iostream.h>
int Lsearch(int[], int, int);   // ie. Lsearch(the array, its size, search item)

int main ()
{int AR[50], ITEM, N, index;   //array can hold max. 50 elements

cout << " Enter desired array size (max. 50)...";
cin>>N;
cout << "\n Enter Array Elements \n";
for (int i=0; i<N ; i++)
{ cin >> AR[i]; }
```

Scanned by CamScanner

```
cout << "\n Enter Element to be searched for...";
cin>>ITEM;
index = lsearch(AR,N,ITEM);
if (index == -1)
  cout << "\n Sorry!! Given element could not
be found.\n";
else
cout << "\n Element found at index:" << index<<
    ", Position:" << index +1 << endl;
return 0;
}

int lsearch (int AR[], int size, int item)
                    //function to perform linear
                    search
{
 for(int i = 0; i<size ; i++)
  {  if (AR[i] == item)  return i;
            // return index of item in case of successful
                                        search
  }
 return -1;      //the control will reach here only
}                 when item is not found
```

Output

```
Enter desired array size(max.50)...7
Enter Array Elements
1 5 2 8 6 10 7
Enter element to be searched for...6
Element found at index:4, Position:5
```

Scanned by CamScanner

DATE
PAGE

- Binary Search

This popular search technique searches the given ITEM in minimum possible comparisons. The binary search requires the array, to be scanned, must be sorted in any order, (for instance say ascending order). In binary search, the ITEM is searched for in smaller segment (nearly half the previous segment) after every stage. For the first stage, the segment contains the entire array.

To search for ITEM in a sorted array (in ascending order), the ITEM is compared with middle element of segment (i.e. in the entire array for the first time). If the ITEM is more than middle element, latter part of the segment becomes new segment to be scanned; if the ITEM is less than the middle element, former part of the segment becomes new segment to be scanned. The same process is repeated for the new segment(s) until either the ITEM is found (search successful) or the segment is reduced to the single element and still the ITEM is not found (search unsuccessful)

Scanned by CamScanner

- Algorithm

Case I: Array AR[L:U] is stored in ascending order

/* Initialize segment variables */

1. Set beg = L, last = U // In C++ L is 0 and U is size-1.

2. REPEAT steps 3 through 6 UNTIL beg>last

// INT() is used to extract integer part

3. mid = INT((beg+last)/2)

4. if AR[mid] == ITEM then

{

print "Search Successful"

print ITEM," found at", mid

break /* go out of the loop

}

5. if AR[mid] < ITEM then

beg = mid +1

6. if AR[mid] > ITEM then

last = mid-1

7. if beg ≠ last

print "unsuccessful search"

8. END

Scanned by CamScanner

Case II : Array AR[L:U] is stored in descending order

```
/* initialise */
:
if AR[mid] == ITEM then
{ :
}
if AR[mid] < ITEM then
    last = mid-1
if AR[mid] > ITEM then
    beg = mid+1
:
```

// Rest is similar to algorith in CASE 1.

> Program for Binary Search

```
#include <iostream.h>
int Bsearch (int[], int, int);
int main()
{ int AR[50], ITEM, N, index;   //array can hold max. 50 elements
cout << "Enter desired array size (max. 50) ...";
cin >> N; cout << "\n Enter Array elements \n";
for(int i=0;i<N;i++)
cin>>AR[i];
cout << "\n Enter Element to be searched for ...";
cin >> ITEM;
index = Bsearch AR, N, ITEM);
if (index == -1)
```

```
cout << "\n Sorry!! Given element could not be
        found.\n";
else
  cout << "\n Element found at index:" << index <<
    "position:" << index + 1 << endl;
return 0;
}
int Bsearch (int AR[], int size, int item)
                        //function to perform binary
                                          search
{ int beg, last, mid;
  beg = 0;  last = size-1;
  while ( beg <= last )
  { mid = (beg + last)/2;
  if (item == AR[mid]) return mid;
  else if (item > AR[mid]) beg = mid + 1;
  else  last = mid - 1;
  }
  return -1;      // the control will reach here
                     only when item is not found
}
```

OUTPUT

```
Enter desired array size (max. 50)...7
Enter Array elements ...
2 5 7 8 9 10 15
Enter element to be searched for ...8
Element found at index : 3 position : 4
```

Scanned by CamScanner

DATE
PAGE

12 WAP to print first letters of any string in Upper case if all letters present are in lower case.

```
#include <iostream.h>
# include < string.h>
# include< stdio.h>
# include < conio.h>
void main()
{ clrscr(),
 char str[100];
 cout<<"\n enter the string \n";
 gets(str),
  str [0]-= 32;
  for (int i=0; i<=100; i++)
  { if (str[i] == ' ')
     str[i+1] - = 32;
  }
  getch ();
}
```

OUTPUT

enter the string

mohan das karamchand gandhi

Mohan Das Karamchand Gandhi

Scanned by CamScanner