Array and function!

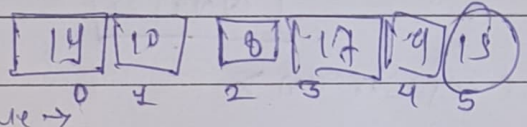
Array

* An array is collection of element with same data type that is refrenced by common name.

int x[5];

float grade[100];

Array members → x[0] x[1] x[2] x[3] x[4] x[5]



Index →

int e[2][3] = {{1,3,0}, {-1,5,4}};

	column 1	column 2	column 3	column 4
Row 1	x[0][0]	x[0][2]	x[0][3]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[2][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Function

Function is a collection of statement together as a unit, which used to perform arithmetic and logical operation and return value.

Syntax:-

return type functionName (parameter 1, parameter 2, ...)

```
void displaynum(int n1, float n2){
    cout<<"int number <<n1;
    cout<< float number<<n2;
}
int main(){ int num1=5; double num2=5.5;
    display num (num1, num2);
    return 0;
```

Page No.:
Date:

call by value :-

```
#include<iostream.h>
using namespace std;
void change(int);
int main()
{
  int data = 3;
  change(data);
  cout<<"value of data is "<<data
```

call by value | call by refrence

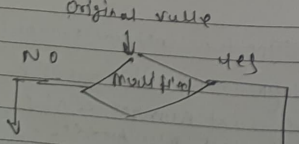
call by value	call by refrence
(1) A copy of value is passed to function	(1) An address of value is passed to function
(2) change made inside function not reflected other func	(2) change may apply in the function.
(3) Actual and formal arguments will be created in different memory	(3) — " — same memory

Pointer :-

The pointer in c++ is a variable it's also known as locator or indicator that point to an address of a value

address → value

```
#include<iostream>
using namespace std;
int main()
{
  int num=30;
  int *p;
  p=&number;
  cout<<"address of number variable"<<
  cout<<"address of p variable is<<;
  cout<<"value at address in p variable is"<<
  return 0;
```



Unit - 3

Date :

Object and classes :-

(I) Object oriented programming Paradigm :- In c++ rather than creating separate variable and function, we can also wrap these related data and function in single place. This programming paradigm is known as object oriented programing.

(II) c++ classes :- A class is a blueprint for the object (If class is house) it contains all the details about the floor, doors, window etc class is a blueprint.

Create a class :- A class is defined in c++ the body of the class is defined inside the curly brackets and terminated by semicolon at end.

```
class className { // some data
                  // some function };
```

Ex :-

```
class Room {
public:
    double length;
    double breadth;
    double height;

    double calculateArea() {
        return length * breadth; }
    double calculateVolume() {
        return length * breadth * height; }
```

Page No. :

Date :

(II) c++ objects :- when a class defined, only the specification for the object is defined, no memory or storage allocated.

Syntax :-

```
className objectVariableName;
```

Example :-

```
// sample function
void sampleFunction() {
    // create objects
    Room room1, room2; }
Int main() { Room room3, room4; }
```

room1.length = 5.5 - In this case it initializes length variable of room1 to 5.5,

Example - Object and classes,

```
// Program to illustrate the working of object and class in C++
#include <iostream>
using namespace std;
// create a class
class Room {
public:
    double breadth;
    double length;
    double height;
    double calculateArea() { return length * breadth; }
    double volume calculate() { return length * breadth * hor; }
};
int main() { // create object of room class
    Room room1;
// assign values to data members
    room1.length = 42.5;
    room1.breadth = 30.8;
    room1.height = 19.2;
```

// calculate and display the area and volume of room
cout << "Area of Room =" << room1.calculateArea() <<
cout << "Volume of Room =" << room1.calculateVolume() <<
return 0;
}

```
#include <iostream>
using namespace std;
class Room {
  private:
    double length;
    double breadth;
    double height;
  public:
    void intData(double len, double brth, double hgt){
      length = len; , breadth = brth; , height = hgt; }
    double calculateArea(){ return length * breadth; }
    // volume(){ return len * breath * height; }
};
int main(){ //create object of room class
    Room room1;
// pass the value of private variable as arguments
  room1.intData(42.5, 30.8, 14.2);
  cout << "Area of Room = " << room1.calculateArea() << ;
  cout << "Volume of Room = " << room1.calculateVolume() << ;
  return 0; }
```

Page No. :
Date :

* Destructors (C++)

* A destructor works opposite to constructor; it destructs the object of class, it can be defined only once in a class. like constructors.
* A destructor is defined like constructor. it must have same as a class. But it is prefixed with a tilde sign(~).
* A destructor can't have parameter.

Syntax :- ~ constructor-name();

* Properties of destructor :- (1) Destructor function is automatically invoked when the object are destroyed.
(2) It can't be declared static or const.
(3) The destructor does not have arguments.
(4) It has no return type not even void.
(5) An object of classs with

* When destructor called :- A destructor called when when object goes out of scope : (1) the function ends
(2) The programend (3) a block containg local variable ends.
(4) A delete operator is called.

* Destructor have same name as a class preceded by tilde(~)
Destructor don't take any argument and don't return any

```
#include <iostream>
using namespace std;
class Employee
{ public: Employee()
  { cout << "constructor invoked" << endl; }
~Employee() { cout << "Destructor invoked" << endl; }
};
int main(void)
{ Employee e1;
  Employee e2;
  return 0; }
```

output:- constructor invoked
constructor invoked
Destructor invoked
Destructor invoked

* Scope Resolution operator :- In c++, Scope resolution operator is(::) two colons. It used to following purpose:

① To access a global variable where there is a local variable with same name.

```
// c++ program to show global variable and using of scope resolution operator :: when there is local
#include <iostream>
using namespace std;
int x;
int main()
{ int x=10;
  cout << "value of global x is " << ::x;
  cout << "\n value of local x is " << x;
  return 0; }
```

② To define a function outside of class.

```
#include <iostream>
using namespace std;
class Simplefun
{
  public:
    void fun(); };
void Simplefun::fun()
{ cout << "fun() called"; }
int main() {
  Simplefun sf;
  sf.fun(); }
```

③ To access a class's static variables.

```
#include <iostream>
using namespace std;
class Test
{ static int x;
public:
  static int y;
void func(int x)
{
  cout << "value of static x is" << Test::x;
  cout << "\n value of local x is " << x;
}
};
int Test::x = 1;
int Test::y = 2;
int main()
{ Test obj;
  int x=3;
  obj.func(x);
  cout << "\n Test::y=" << Test::y;
  return 0;
}
```

4) In case of multiple Inheritance: If same variable name exist in two ancestor classes, we can use scope resolution operator to distinguish.

// use of scope resolution operator in multiple // inheritance.

```
#include <iostream>
using namespace std;
class A
{ protected:
   int x;
  public:
   A() { x=10; } };
class B
{ protected:
   int x;
public: class
   B() { x=20; }
   };
class C: public A, public B
{ public:
  void fun()
  {
    cout << "A's is x is" << A::x;
    cout << " B's is x is" << B::x; }
  };
  int main()
  {
    C c;
    c.fun();
    return 0;
  }
```

Output:- A's x is 10
B's x is 20

5) For namespace:-

// use of scope resolution operator for // namespace.

```
#include <iostream>
int main(){
    std::cout << "Hello" << std::endl;
  }
```

6) Refer to a class inside another class:-

// Use of scope resolution operator to class inside another.

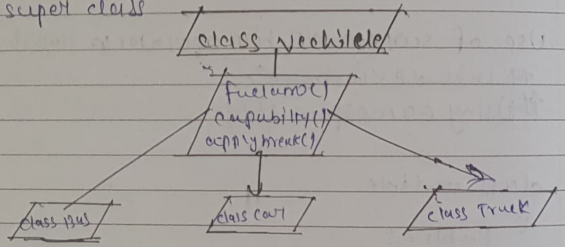
```
#include <iostream>
using namespace std;

class outside
{ public:
    int x;
    class inside
  { public:
    int x;
    static int y;
    int foo(); };
};
int outside::inside::y=5;
int main(){ outside A;
            outside::inside B;
```

* C++ Inheritance:- The capability of a class to derive properties and characteristics from another class is called inheritance. Inheritance is one of the most important feature of object oriented progra.

* Sub class:- The class that inherits properties from another class is called sub class.

* Super class:- The class whose properties are inherited by sub class is called Base class or super class



* Syntax for inheritance and access mode:-

(1) Public:- members are accessible from outside the class

(2) Private:- members can't accessible from outside the class.

(3) Protected:- members can't accessed in inherited classes.

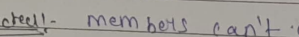
Syntax:- class subclass_name: access_mode base_class_name
{
// body of subclass
};

(1) code - (1) :-> Demonstration of inheritance:-

```
#include <iostream>
using namespace std;
class parent
{
public:
    int id_p;
};
// sub class inheriting from // Base class
class child : public parent
{ public: int id_c;
};
int main()
{ child obj1;  (// object of class child has all data member and member func)
obj1.id_c = 7;
obj1.id_p = 91;
cout << "child id is" << obj1.id_c << endl;
cout << " Parent id is" << obj1.id_p <<
return 0; }
```

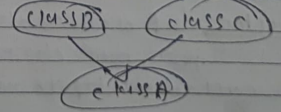
* Types of Inheritance:- (1) single inh (3) multi-level (5) virtual (2) multiple (4) Hierarchical (6) multi-path

(1) Single Inheritance:- In single inheritance, a class is allowed to inherit from only one class. one sub class is inherited by one base class only.



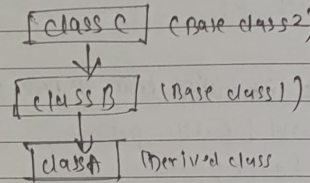
Syntax:- class subclass_name: access_mode base_class_name
{ // body of subclass };

(2) multiple inheritance:- In this a class can inherit from one more than one class. one sub class is inherit from more than one base class.

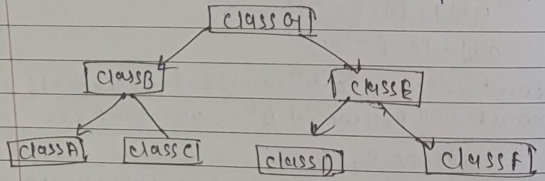


class subclass_name:
access mode base_class1,
- // - base - class 2

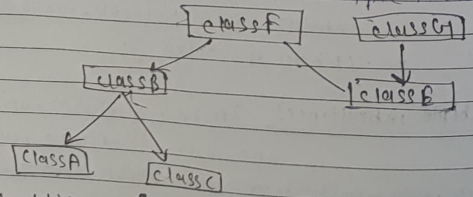
③ **Multi-level inheritance**:- In this type of inheritance a derived class is created from another derived class.



④ **Hierarchical inheritance**:- In this type of inheritance more than one sub class is inherited from a single base class, more than one derivered class is created from single base.



⑤ **Hybrid (virtual) inheritance**:- Hybrid inheritance is implemented by combining more than one type of inheritance. combining hierarchical inheritance and multiple inheritance.



⑥ **Multipath inheritance**:- A derived class with two base classes and these two base classes is called multipath inheritance.

Friend function & classes

Friend function in c++

- If a function is defined as a friend function in c++, then the protected and private data of a class can be accessed using the function.
- By using the keyword friend compiler knows the given function is friend function.
- For accessing the data, the declaration of a friend function should be done inside the body of class starting with the keyword friend.

Declaration of friend function in c++

```
class class_name
{ friend data_type function_name;
};
```

Characteristic of a friend function:-

- The function is not in the scope of class which it has been declared as a friend.
- It can't be called using the object as it is not in the scope of that class.
- It can be invoked like a normal function without using object.
- It can't be access the member names directly and need to use an object name and dot membership operator with the member name.
- It can be declared either in the public or private.

// Friend Function example.

```
#include<iostream>
using namespace std;
```

```
class Box
{ Private:
  int length;
public:
  Box(): length(0){}
  friend int printLength(Box); // Friend function
};
int printLength(Box b)
{ b.length += 10;
  return b.length;
}
int main()
{ Box b;
  cout<<"Length of box:" << printLength(b)<<endl;
  return 0;
}
```

Output : length of box : 10

// Friend of two classes.

```
#include<iostream>
using namespace std;
class B;
class A
{ int x;
  public:
    void setdata(int);
  { x = 5;
  }
    friend void min(A, B);
};
```

```
class B;
{ public:
    int y;
    void setdata(int);
  { y = 9
  }
  friend void main(A,B);
void min(Aa, Bb)
{ if(a.x <= b.y)
    std::cout << a.x << std::endl;
else
    std::cout << b.y << std::endl;
}
A a;
B b;
a.setdata(10);
b.setdata(20);
min(a,b);
return 0;
}
```

Output=10

☆ Friend classes:- A friend class can access both private and protected members of class in which it has been declared as friend.

```
class ClassB;
class ClassA { // classB is friend class of classA
    friend class ClassB
    ....
  }
class ClassB{ ..... }
```

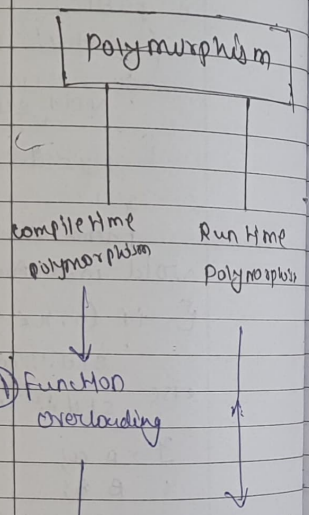
When a class is declared a friend class all the member function of the friend class become friend function

Date :

Example for friend class;

```
#include<iostream>
using namespace std;
class A{
private:
    int a;
public:
    A(){a=0;}
    friend class B;
};
class B{
private:
    int b;
public:
    void showA(A &x)
    {cout<<"A::a="<<x.a;}
};
int main()
{
    A a;
    B b;
    b.showA(a);
    return 0;
}
```

Polymorphism



C++ Polymorphism:-

The word polymorphism means many forms.

In C++ polymorphism is an important concept of object oriented programming. it means more than one form. that is same entity in different scenarios.

(1) Function overloading
(2) Operator Overloading
(3) Function overriding
(4) Virtual functions.

Page No. :
Date :

(1) C++ Function overloading:-

- In C++ we can use two functions having same name if they have different parameters, and depending upon the num. of argument, different function are called

Example for C++ function overloading:-

```
#include<iostream>
Using namespace std;
int sum(int num1, int num2){
    return num1+num2; }
double sum(double num1, double num2){
    return num1+num2; }
int sum(int num1, int num2, int num3){
    return num1+num2+num3; }
int main(){
    cout<<"sum1="<<sum(5,6)<<endl;
    cout<<"sum2="<<sum(5,5+6,6)<<endl;
    cout<<"sum3="<<sum(5,6,7)<<endl;
    return 0; }
```

(2) C++ operator overloading:-

In C++, we can change the way operators work for user defined type like object and structure. this is known operator overloading. if we suppose object c_1, c_2 and result from class complex then operator allow overload

- allow us to change how operator work we can redefine how the (+) work and we it to add complex

SYNTAX

```
result = c1 + c2;
result = c1.addNumber(c2);
class className{
    .....
public:
    returnType operator symbol(arguments){ ..... }
    ..... }
```

3. Function Overriding :- Inheritance is feature of OOP allow us to create derived classes from a base class. the derived class inherit features of base class. If function is defined in both the derived class and base class we use call this function using the object of derived class, function of derived execute

- The function in derived class overrides the function in base.

** C++ function Overriding :-

```
#include <iostream>
using namespace std;
class Base { public:
    void print() {
        cout << "Derived function" << endl; }
};
int main() {
    Derived derived1;
    derived1.print();
    return 0; }
```

```
class Base {
public:
    void print() { //code }
};
class Derived : public Base {
    void print() { //code }
};
int main() {
    Derived derived1;
    derived1.print();
    return 0; }
```

4) Virtual Function :- A virtual function is a member function which is declared within a base class and re-defined (overridden) by derived class when you refer to derived class object using a pointer or a reference to base class, you can call a virtual function for that object and execute the derived class's version of function

- Virtual function ensure that the correct function is called for an object, irrespective of type of reference used for function
- They are mainly used to achieve Runtime polymorphism
- The resolving of function call is done at runtime
- virtual function actually fall under function overriding.

** Example of virtual function :-

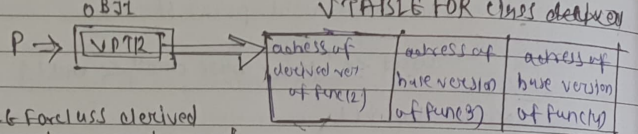
```
#include <iostream>
using namespace std;
class base { public:
    virtual void print()
    { cout << " print base class" << endl; }
    void show()
    { cout << "show base class" << endl; }
};
class derived : public base { public:
    void print()
    { cout << " print derived class" << endl; }
    void show()
    { cout << "show derived class" << endl; }
};
int main() { base *bptr; derived d; bptr = &d;
// virtual function, binded at runtime
    bptr->print();
// Non-virtual function, // binded at compile time
    bptr->show(); }
```

** working of virtual functions (concept of VTABLE & VPTR)

⇒ If class contain virtual function then compiler itself does two things.

1. Object of that class is created then a virtual pointer (VPTR) is inserted as a data member of class. for each new object created, a new virtual pointer is inserted as a data member of that class.

2) Irrespective of object is creating or not, class contains as a member a static array of function pointer called VTABLE.



VTABLE For class derived

address of base version func 2()	address of base version of func()	address of base version of func()

Working of Virtual Functions :-

```
#include <iostream>
using namespace std;
class base{ public:
    void fun-1(){ cout<<"base-1\n";}
    virtual void fun-2(){ cout<<" base-2\n";}
    virtual void fun-3(){ cout<<" base-3\n";}
    virtual void fun-4(){ cout<<" base-4\n";}
    };
class derived: public base{
  public:
      void fun-1(){ cout<<" derived-1\n";}
      -11 --21(){ cout<<" derived-2\n";}
      -11 --41(int x){ cout<<" derived-4\n";}
int main(){ base *p;
          derived obj1;
    p=&obj1;
p->fun-1();
p->fun-2();
p->fun-3();
p->fun-4();
```

Structures & Enumeration :-

Structure :- Structure is a collection of variables of different data types under a single name. It is similar to a class in that, both hold a collection of data of different data types. If u want to store some information about person, you can easily create different variable name, city, salary to store these information separately. So a better approach will be have a collection of all related information under a single name person and use it for every person."

"this collection of all related information under a single name person is a structure". The struct keyword defines a structure type followed by identifier then inside braces we can declare more members of that structure. Here structure person defined

```
struct Person    three members : name, age, salary
{
    char name[50];
    int age;
    float salary;
};
```

- when structure is created, no memory is allocated.
- structure is only the blueprint for creating of variables. The int specifies that variable bill can integer. / and the declaration with semicolon.
- Person bill; (Structure variable bill is defined type structure person
- when structure variable is defined only required memory allocated by compiler.
- The members of structure variable is accessed using (.) operator.
- if we want access age of structure variable bill & assign it to 50.

```
bill.age = 50;
```

Date :

* C++ structure :- #include <iostream>
using namespace std;
Struct person
{ char name[50];
int age;
float salary;
};
int main() { person p1;
cout << "Enter full name : ";
cin >> get(p1.name, 50);
cout << "Enter age";
cin >> p1.age;
cout << "Enter salary"; cin >> p1.salary;
cout << "\n Displaying information" << ;
cout << "Name" << p1.name << endl;
cout << "Age" << p1.age << endl;
cout << "Salary" << p1.salary;
return 0; }

Output
Enter full name : ABC
Enter age : 24
Enter salary : 1024.4

* For structre and function :- -// — };
void display Data(person);
-// — same -// — // —);

* C++ Enumeration :- An enumeration is a user defined data type that consists of integral constant. To define an enumeration, keyword enum is used.
enum season { spring, summer, autumn, winter };
enum season { spring = 0, summer = 4, autumn = 8, winter = 12 };

Date :

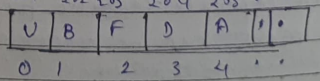
#include <iostream>
Using namespace std;
enum week { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };
int main() { week today;
today = wednesday;
cout << "Day" << today + 1;
return 0; }

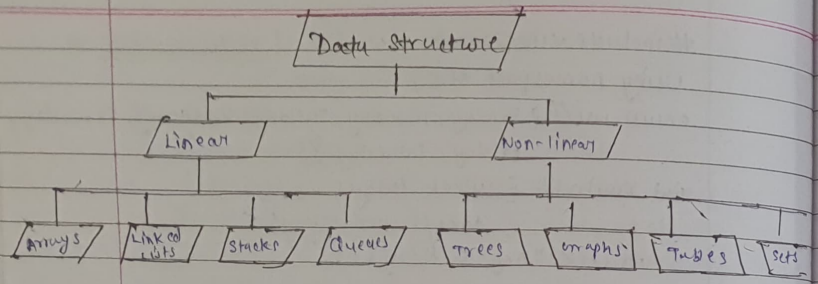
(2) -// — enum seasons { spring = 34, summer = 4, autumn = 9, winter = 2 }
int main() { season s;
s = summer;
cout << "summer = " << s << endl;

(3) -// — enum suit { club = 0, diamond = 10, heart = 20, spades = 3 } card; int main() { card = club;
cout << "size of enum variable" << sizeof(card) << "byte
return 0; }

* Introduction to Data structure :-

* Data structure :- A data structure is particular way of organizing data in a computer so that it can used effectively.
for example we can store a list of item having same data type using the array data structure.



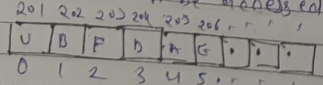


* Data Structure operation :- (1) Traversing (2) Searching (3) Inserting (4) Deleting (5) Sorting (6) merging (7) copying (8) concatenating

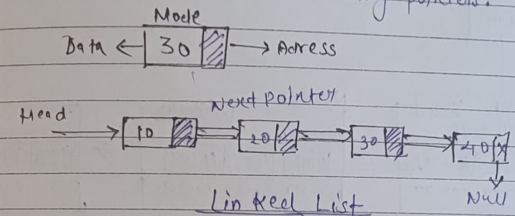
* Linear Data Structes :- In the data structure, data element are accessed in sequential fashion one by one. The idea is reduce the space and time complexities of different tasks. Below is an overview of some popular linear data structure (1) Array (3) STACK (2) Linked list (4) Queue

(1) Array :- The array is data structure used to store homogenous elements at contigous location. The size of an array must be provided before storing data.

for ex. let's we want to store marks all student in class, we can use an array to store them. this help in reducing the use of num of variable as we don't need to create separate variable for marks for every subject. All marks can be accessed by single,



(2) Linked list :- A linked list is linear data structure in which the elements are not stored at contigous memory location. The element in a linked list are linked using pointers.

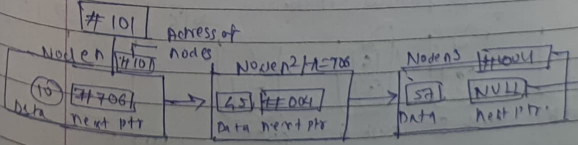


Type of Linked list :- (1) Single linked list :- In this type of linked list every node store address or refrences of next node in list and last node has the next address as NULL. for 1->2->3->4->Null

* Double linked list :- In this type there are two refrences associated with each node. one of refrences point to the next node and one to previous the advantage of structure is that we can traverse in both direction and for deciding.

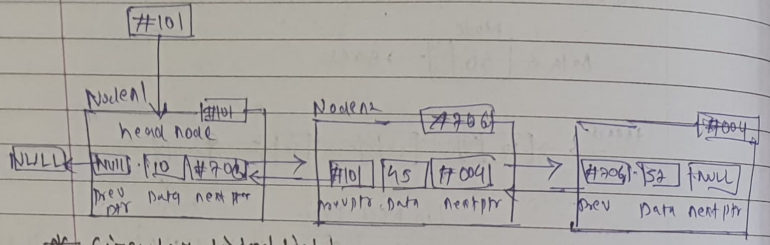
* Circular linked list :- Circular linked list where all nodes are connected to form a circle. there is no null at the end. It can singular or circular eg :- 1->2->3->1 [The next pointer of node is pointing to first]

* Simple Singly linked list :-

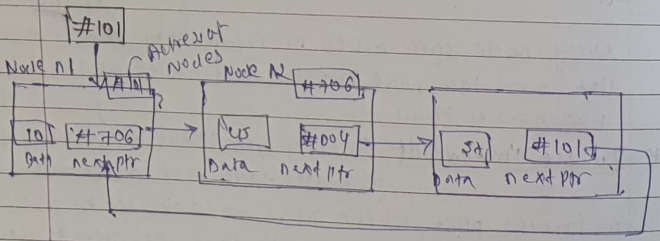


Date :

Doubly linked list :-

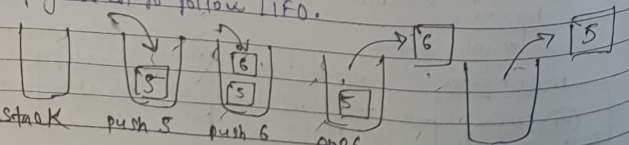


Circular linked list :-



3) **STACK** :- Stack is a linear data structures which follows a particular order in which the operat are performed. The order may be LIFO (Last in first out)

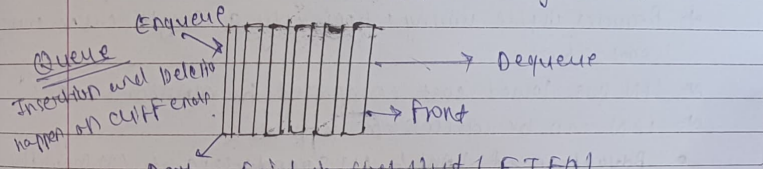
There are many real life ex of stack consider ex of plates stacked over another in canteen. the plate which is at the top is first one to be removed. the plate which has been placed at bottom most position remain in stack for longest period of time. so it can be simply seen to follow LIFO.



Last in first out (LILO)

Date :

4) **Queue** :- A Queue is a linear structure which follow a particular order in which the operations are performed. the order is first in first out (FIFO) diff b/w stack and queue is in removing. in stack we remove the item most recently added and in queue we remove the item least recently added.



Network

A computer or network consist of two or more computer that are linked in order to share resources exchange files or allow electronic communications.

The computer on network may be linked through cables, telephone lines, radio waves, satellites.

Basic types of network :-

1) Local Area Network (LAN)
2) Personal Area Network (PAN)
3) Metropolitan Area Network (MAN)
4) Wide Area Network (WAN)
5) Campus Area Network (CAN)
6) Storage Area Network

Local Area Network (LAN) :- A Lan is a network that is used for

communicating among computer devices, usually within an office building or home.
* LAN's enable the sharing of resources such as files or hardware devices that may be needed by multiple users. It is limited size.
* LAN is fast, with speed from 10 Mbps to 10 Gbps.
* Requires little wiring, typically a single cable connecting each other.
* LAN has lower cost compared to MAN or WAN's
* LAN can be wireless or wired.
- Advantage of LAN :- (1) Speed (2) Cost (3) Security (4) Resource sharing.

2) Personal Area Network (PAN) :- A pan is a network that used for communi
PAN network is same to LAN
* PAN points = or LAN points
* PAN can be wireless or wired.

3) Metropolitan Area Network (MAN) :- A metropolitan area network (MAN) is large computer network that usually spans a city or large campus
* A MAN is optimized for larger geographical area than a LAN ranging.
* MAN might be own or operated by single organization but it is used by individual and organization.
* MAN is often acts as high speed network to allow sharing of regional
* MAN typically covers area of network b/w 5 to 50 Km in diameter.
* MAN, Ex :- Telephone company.
- Advantage :- (1) Speed (2) cost (3) Security (4) Resource sharing

4) Wide Area Network (WAN) :- WAN covers a large geographic such as a country continent or even whole of the world.
- A WAN is two or more LANs connected together. LAN can be many miles apart.
- To cover great distance, WAN's may transmit data over leased high speed phone lines or wireless link such as satellites.
- Multiple LAN can be connected together using devices such as bridges routers.
- most popular WAN is INTERNET.

5) Campus Area Network (CAN) :- A campus area network of multiple interconnect local area network (LAN) in a limited geographical area. A CAN is smaller than a wide area network or metropolitan area network (MAN).
* CAN is also known as corporate area network (CAN)
* CAN is cost-effective, wireless versus cable, Multidepartmental network access.
- Advantage of CAN :- (1) speed (2) Reliability (3) campus interconnection (4) Better for every consumer.

6) Storage Area Network (SAN) :- A storage area network is specialized, high speed network that provides block level network access to storage. SAN are typically composed of hosts, switches storage elements and devices that interconnected using variety of technology.
- This enable each server to access shared storage as if it were a drive directly attached to server.
* Advantage :- (1) Less expense (2) fault tolerance (3) Real time update (4) Role monitoring