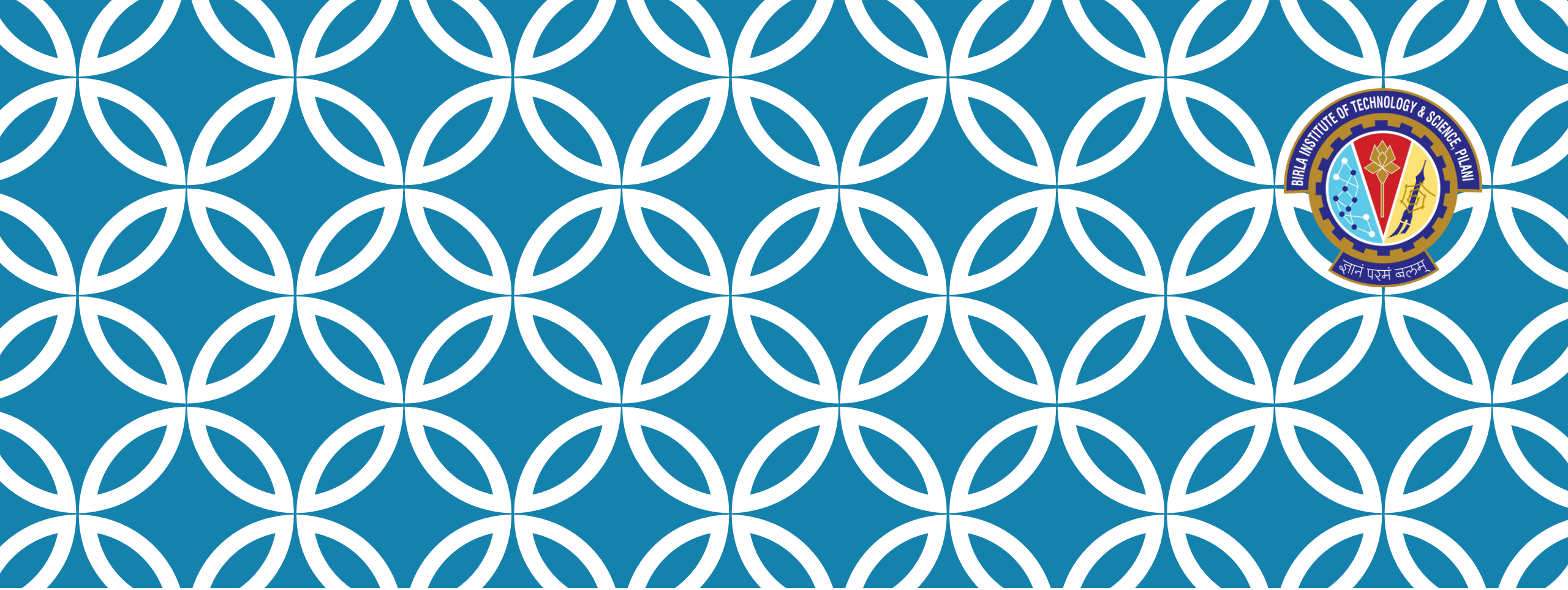
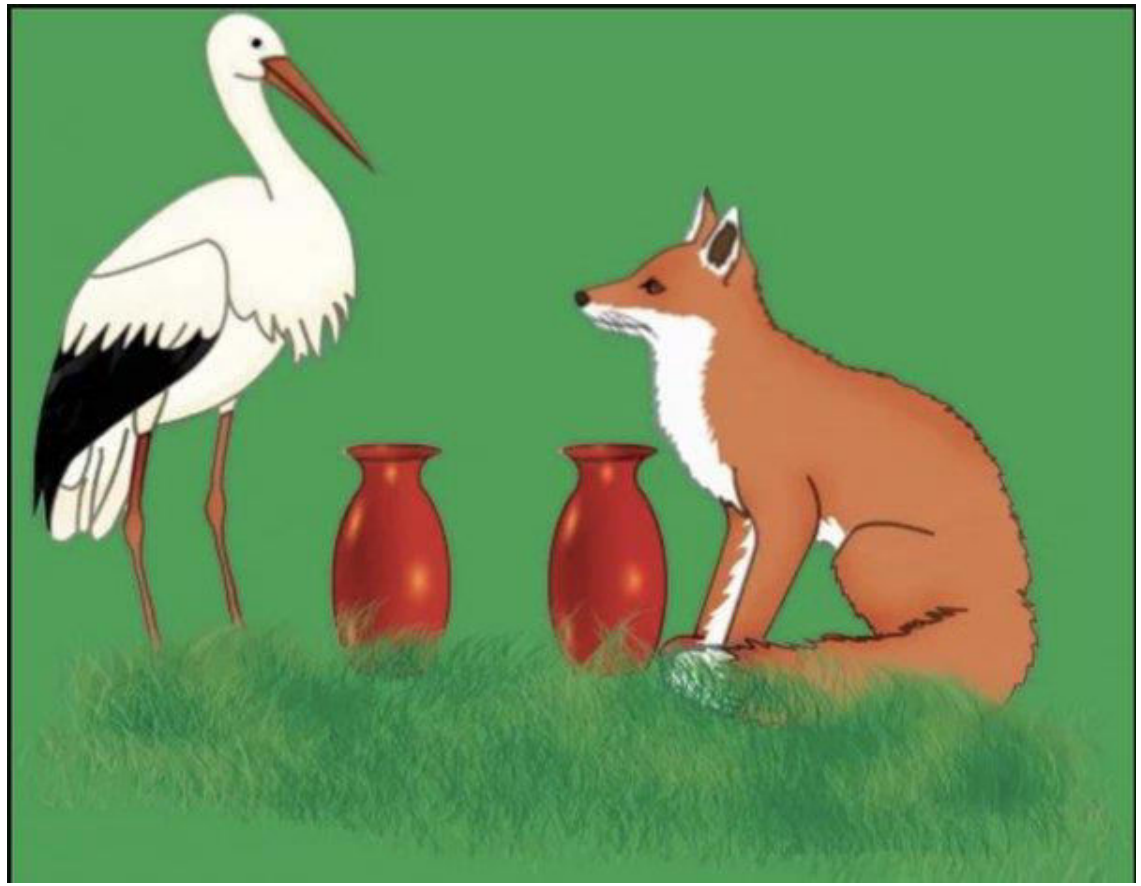


BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) INTRODUCTION

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

WHY SHOULD YOU STUDY THE COURSE?



You will be able to analyze a program to find out where to improve.

Searching your name in the MCN list.

Driving through Hyd using maps.

Image source: https://www.quora.com/

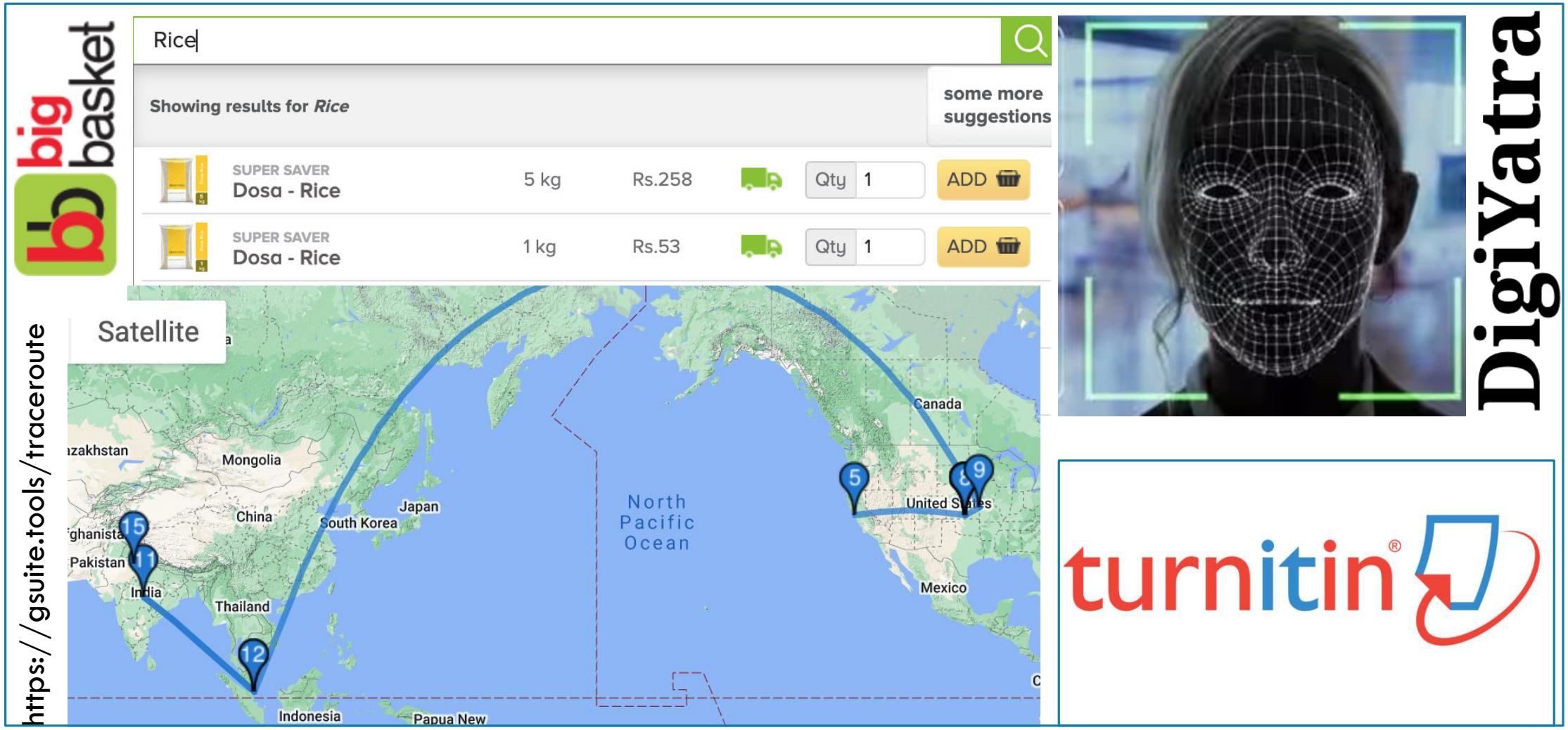
FIBONACCI: RECURSIVE OR ITERATIVE?

Algorithm 2: $F(n)$

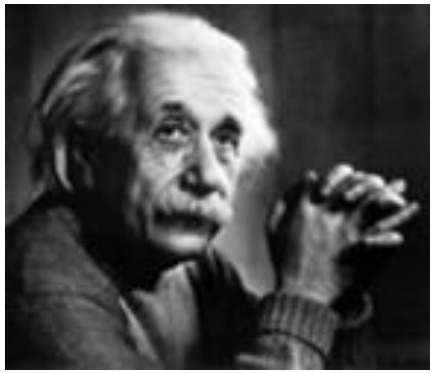
Input: Some non-negative integer n
Output: The nth number in the Fibonacci Sequence

$A[0] \leftarrow 0;$
$A[1] \leftarrow 1;$
for $i \leftarrow 2$ ***to*** $n-1$ **do**
  $A[i] \leftarrow A[i-1] + A[i-2];$
return $A[n-1]$

WHAT KIND OF PROBLEMS CAN YOU SOLVE?

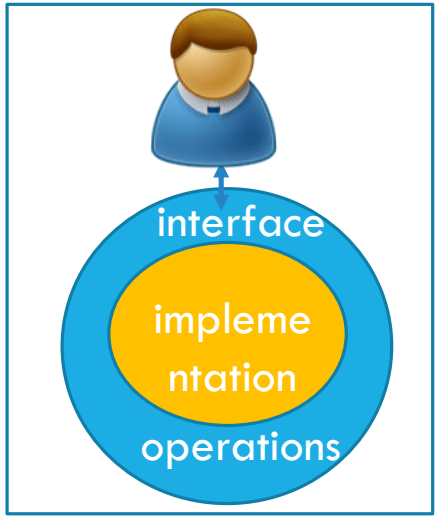


THE PROBLEM SOLVING PROCESS

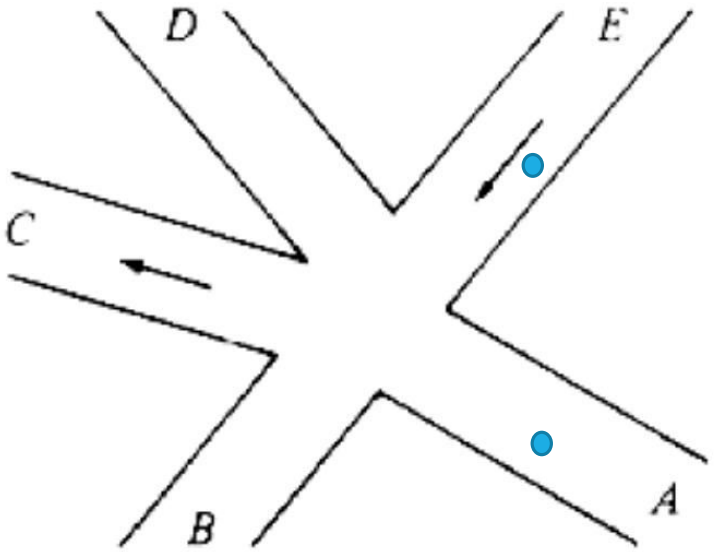


If I had 1 hour to save the world, …

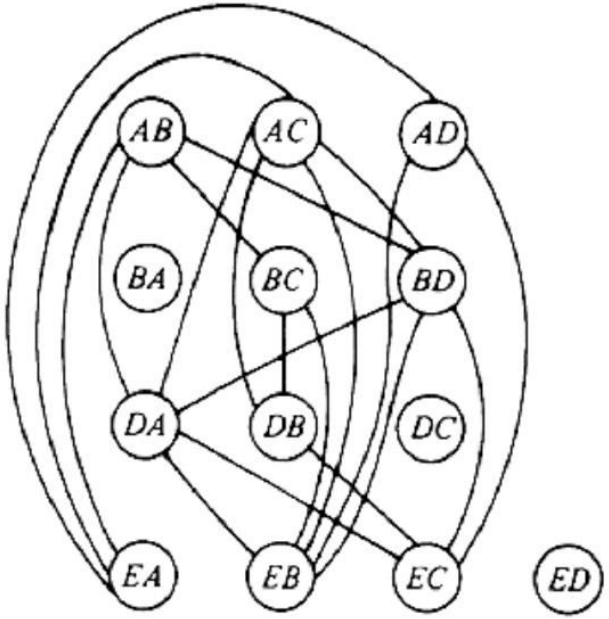
- finding currents in electrical circuits, predicting spread of covid-19
- Language translator
- An abstract data type (ADT), is a logical description of how we view the data and the operations that are allowed without knowing how they will be implemented.
- Data structure (physical description) is the implementation of ADTs.



AN EXAMPLE: A TRAFFIC LIGHTING SYSTEM



(Problem of road intersection)



(Graph with incompatible turns)

1. Select an uncolored vertex and color it with a new color.
2. Scan the list of uncolored vertices. For each uncolored vertex, determine whether it has an edge to any vertex already colored with the new color. If there is no such edge, color the present vertex with the new color.

(Greedy coloring algorithm)

- The approach is called "greedy" because it colors a vertex whenever it can, without considering the potential drawbacks inherent in making such a move. AB,AC,AD,BA,DC,ED;

CONTINUED...

```
SET greedy_graph_coloring (Input:G:GRAPH,
Output: Newclr: SET) {
    Newclr ← Ø;
    for (each uncolored vertex 'v' ∈ G)
    {
        if 'v' is not adjacent to any vertex in Newclr
        {
            v ← colored;
            Newclr ← Newclr ∪ 'v'
        }
    }
}
```

(Pseudo Code)

https://www.techiedelight.com/greedy-coloring-graph/

GRAPH ADT: G

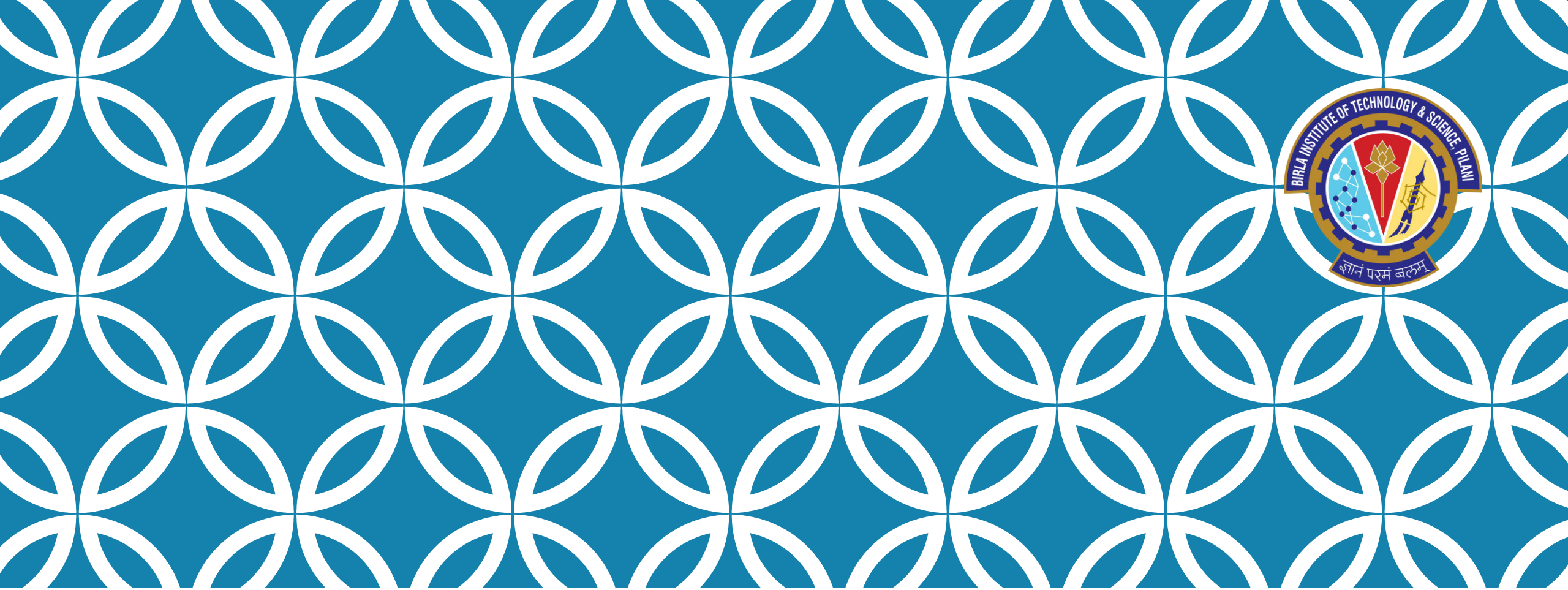
1. graphNew(): creating a graph
2. addVertex (v)
3. addEdge (v1, v2)
4. getVertex (unclored)
5. markVertex (colored)
6. …

	AB	AC	AD	BA	BC	BD	DA	DB	DC	EA	EB	EC	ED
AB					1	1	1			1			
AC						1	1	1		1	1		
AD										1	1	1	
BA													
BC	1							1			1		
BD	1	1					1				1	1	
DA	1	1				1					1	1	
DB		1			1							1	
DC													
EA	1	1	1										
EB		1	1		1	1	1						
EC			1			1	1	1					
ED													

(Adjacency matrix to implement the graph)

THANK YOU!

Next Class: Introduction to C++



BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) INTRODUCTION TO C++

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

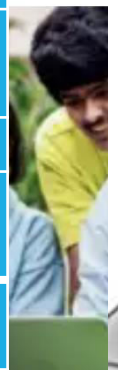
COURSE CONTENT AND ADMINISTRATION

Evaluations:

Component	Duration	Weightage	Date & Time	Nature
Mid sem Test	90 mins.	20%	13/10/2023 (4:00 pm)	Closed Book
Class Interaction	In the class	15%	In the class (best 25/30)	Quiz (Open)
Lab Interaction	In the lab	15%	In the lab (best 10/ 13)	Quiz (Open)
Lab Test (One)	60 mins.	10%	To be announced	Open Book
Programming Assignments (1)	-	10%	To be announced	Take Home
Comprehensive examination	180 mins.	30%	19/12/2023, AN	Part Open

ates

s



Sword of Damocles

Course Content:

Intro to C++, Elementary data structures and algo. analysis techniques, More common data structures, Advanced data structures, Understanding algorithmic techniques.

Course notices and material: google class page

Chamber consultation hour: Every Monday (5 to 6 pm)

WHY C++ FOR BITS F232?

- ✓Developed (in 1979) by Bjarne Stroustrup: Why is it called C++?
- ✓Mid-level: Used for both application level and system level programming tasks.
- ✓Has Object-oriented features improving the quality and reusability of the program.
- ✓Rich library (iostream, iomanip, cmath, cstdlib, iterator, algorithm etc.), Efficiency and speed (competitive coding) …

- ✓Adobe (Photoshop, Illustrator etc are developed using C++, Microsoft used C++ for all of its versions of OS starting from Windows 95, Microsoft Office too is developed using C++, Apple uses C++ to code its OS, MySQL also is written using C++, Mozilla uses a subset of C++, Amazon AWS SDK for C++. Meta, Capgemini, IBM, …

C++ EXAMPLES

```
#include <iostream>

int main( )
{
  int x , y;
  std::cout << "Please enter two numbers: ";
  std::cin >> x >> y;
  int sum = x + y;
  std::cout << "Sum = " << sum << std::endl;
  return EXIT_SUCCESS;
}
```

```
Please enter two numbers: 45 7
Sum = 52


...Program finished with exit code 0
Press ENTER to exit console.
```

```
1  #include <iostream>
2  using namespace std;
3  bool  testSum (int  a[ ], int  n) {
4     int  sum = 0;
5     for (int  i = 0; i < n; i++)
6        sum +=  a[ i ];
7     return (sum % 2 ) == 0;
8  }
9  int  main( )
10 {
11    int a [ 6 ] = {4, 4, 7, 6, 5, 2};
12    bool  result = testSum ( a, 6);
13    if (result)
14       cout << "Sum of all the nos. is even\n";
15    else
16       cout << " Sum of all the nos. is odd\n";
17 return  EXIT_SUCCESS;
18 }
```

```
Sum of all the nos. is even
```

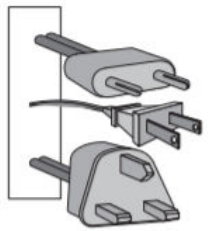
OBJECT-ORIENTED DESIGN: GOALS AND PRINCIPLES

What is Object-Oriented Design?

- Style of writing computer programs using objects, and their interactions. (Minor degree admissions at BITS, Hyderabad: How many objects and what are their interactions)

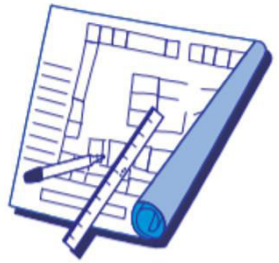
What are the Design Goals?

- Robustness
- Adaptability
- Reusability

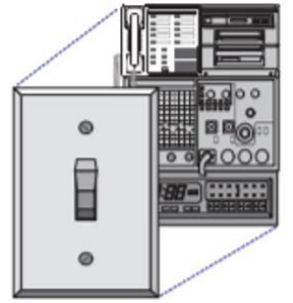


(Design Principles)

Abstraction (ADTs are realized by classes in C++)



Encapsulation (Access to data is provided through member functions)



Modularity (different components): supported through hierarchy.

CLASSES IN C++

- Class: A user-defined type or data structure that has data and functions as its members whose access is governed by the access specifiers.
- Object: A variable declared to be of some class, hence includes both data and functions for that object.
- Usage: A variable is an instance of a type. Similarly, an object is an instance of a class.
- Ex:

```
class  Passerger {
    private:
        string      name;
        MealType    mealPref;
        bool        isFreqflyer;
        string      freqFlyerNo;
    public:
        Passenger( );
        bool  isFrequentFlyer( ) const { return  isFreqFlyer; }
        void makeFrequentFlyer(const  string&  newFreqFlyerNo) {
              isFreqFlyer = true;
              freqFlyerNo = newFreqFlyerNo;
              }
} ;
```

Member variables

Member functions

```
Passenger pass;
if (!pass.isFrequentFlyer()) { pass.makeFrequentFlyer ("12345"); }
```

ILLEGAL: pass.name = "Amit";

ACCESS MODIFIERS: PUBLIC, PRIVATE AND PROTECTED

main.cpp

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

class Circle
{
    public:
        double radius;

        double  compute_area()
        {
            return 3.14*radius*radius;
        }

};

int main()
{
    Circle obj;

    obj.radius = 7.2;

    cout << "Radius is: " << obj.radius << "\n";
    cout << "Area is: " << obj.compute_area();
    return 0;
}
```

```
Radius is: 7.2
Area is: 162.778

...Program finished with exit code 0
Press ENTER to exit console.
```

main.cpp

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

class Circle
{
    private:
        double radius;

    public:
        double  compute_area()
        {
            return 3.14*radius*radius;
        }

};

int main()
{
    Circle obj;

    obj.radius = 7.2;

    cout << "Area is:" << obj.compute_area();
    return 0;
}
```

input

Compilation failed due to following error(s).

```
main.cpp:21:9: error: 'double Circle::radius' is private within this context
     obj.radius = 7.2;
         ^~~~~~
main.cpp:7:16: note: declared private here
         double radius;
                ^~~~~~
```

main.cpp

```cpp
using namespace std;

class Circle
{
    private:
        double radius;
    public:
        void compute_area(double r)
        {
            radius = r;

            double area = 3.14*radius*radius;

            cout << "Radius is: " << radius << endl;
            cout << "Area is: " << area;
        }

};

int main()
{
    Circle obj;

    obj.compute_area(7.2);


    return 0;
}
```

```
Radius is: 7.2
Area is: 162.778

...Program finished with exit code 0
Press ENTER to exit console.
```

CONTINUED...

main.cpp

```cpp
#include<iostream>
using namespace std;
// base class
class Parent
{
    // protected data members
    protected:
        int id_protected;
};

// sub class or derived class from public base class
class Child : public Parent
{
    public:
    void setId(int id)
    {
        // Child class is able to access the inherited
        // protected data members of base class

        id_protected = id;

    }

    void displayId()
    {
        cout << "id_protected is: " << id_protected << endl;
    }
};
```

```cpp

// main function
int main() {

    Child obj1;

    // member function of the derived class can
    // access the protected data members of the base class

    obj1.setId(12345);
    obj1.displayId();
    return 0;
}
```

```
id_protected is: 12345


...Program finished with exit code 0
Press ENTER to exit console.
```

CONSTRUCTORS IN C++

```
1  #include <iostream>
2  using namespace std;
3
4  class Rectangle {
5  public:
6      void set_values(int x,int y) {width=x; height=y;}
7      int area() {return width*height;}
8  private:
9      int width, height;
10 };
11
12 int main () {
13   Rectangle rect;
14   rect.set_values (7,9);
15   cout << "area of rectangle: " << rect.area();
16   return 0;
17 }
```

g++ 17 GCC 9.1.0

CommandLine Arguments

Result

CPU Time: 0.00 sec(s), Memory: 3272 kilobyte(s)

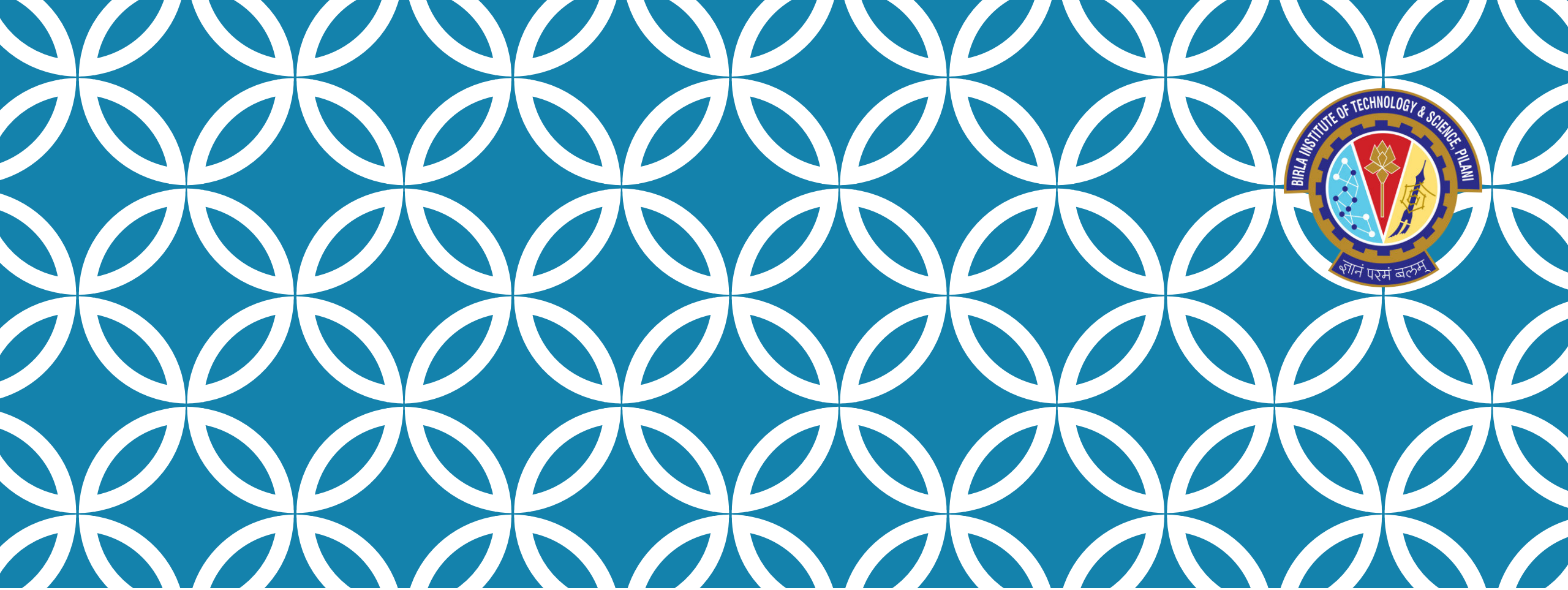
```
area of rectangle: 63
```

```
14    cout << "area of rectangle: " << rect.area();
15    rect.set_values (7,9);
```

```
area of rectangle: -897836400
```

THANK YOU!

Next Class: C++ to be continued…

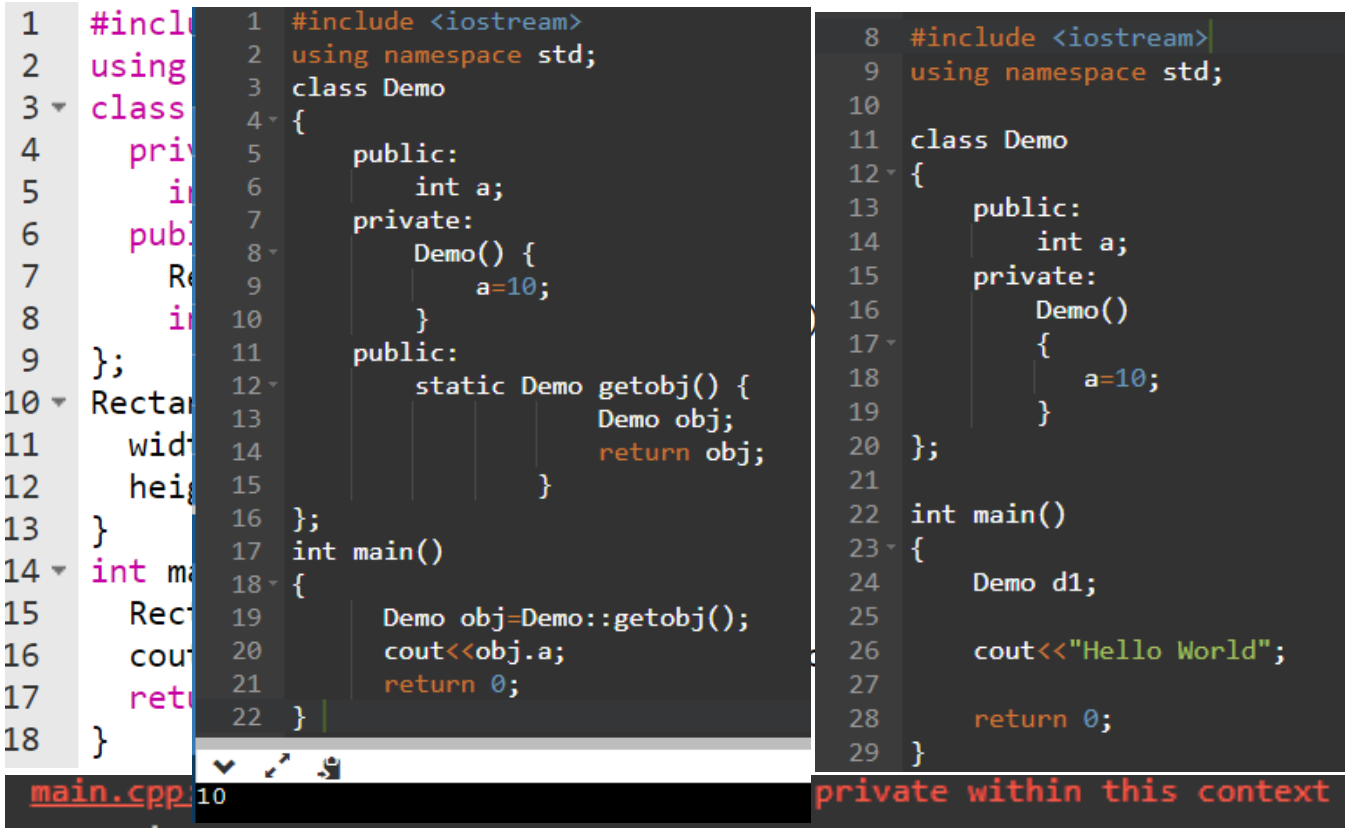


BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) C++ CONTINUED…

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

CONSTRUCTORS IN C++

- A constructor in C++ is a member function of a class which initializes objects of a class and allocates storage. Automatically called (with the same class name) when the object is created.



```
#include <iostream>
using namespace std;
class Demo
{
    public:
        int a;
    private:
        Demo() {
            a=10;
        }
    public:
        static Demo getobj() {
                    Demo obj;
                    return obj;
                }
};
int main()
{
    Demo obj=Demo::getobj();
    cout<<obj.a;
    return 0;
}
```

```
10
```

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

class Demo
{
    public:
        int a;
    private:
        Demo()
        {
            a=10;
        }
};

int main()
{
    Demo d1;

    cout<<"Hello World";

    return 0;
}
```

```
private within this context
```

Result

CPU Time: 0.00 sec(s), Memory: 3508 kilobyte(s)

```
area of rectangle: 45
```

- Default constructor (No args.)
- Parameterized (Takes args.)
- Copy constructor (initializes one object using another object)

Are there any return types for constructors?

EXAMPLES OF DEFAULT AND COPY CONSTRUCTORS

Constructor Overloading

```cpp
#include <iostream>
using namespace std;
class Area {
    public:
        int area;
        Area() {
            area = 0;
        }
        Area(int side) {
            area = side * side;
        }
        Area(int length, int width) {
            area = length * width;
        }
        int disp() {
            return area;
        }
};
int main() {
    Area obj1;
    Area obj2(3);
    Area obj3(7, 5);
    cout << "Area of obj1: " << obj1.disp() << endl;
    cout << "Area of obj2: " << obj2.disp() << endl;
    cout << "Area of obj3: " << obj3.disp() << endl;
    return 0;
}
```

```
Area of obj1: 0
Area of obj2: 9
Area of obj3: 35
```

Result

CPU Time: 0.00 sec(s), Memory: 3508 kilobyte(s)

```
a= 55
b= 78
```

A copy constructor is a member function that initializes an object using another object of the same class.

Result

CPU Time: 0.00 sec(s), Memory: 3276 kilobyte(s)

```
230
```

```cpp
#include <iostream>
using namespace std;
class A  {
public:
    int x;
    A(int a)
    {
      x=a;
    }
    A(A &i)
    {
        x = i.x;
    }
};
int main() {
  A a1(230);
  A a2(a1);
  cout<<a2.x;
  return 0;
}
```

DESTRUCTORS IN C++

- A destructor is a member function that is automatically called when a class object ceases to exist (delete operator is called, execution ends, scope of local variable ends)
- It takes no arguments and has no return type.

```
BITS Pilani
BITS Goa
BITS Hyderabad
```

Is destructor overloading allowed in C++?

```
1  #include <iostream>
2  using namespace std;
3
4  class Example1{
5    public:
6        Example1(){
7            cout << "BITS Pilani" << endl;
8        }
9        ~Example1(){
10           cout << "BITS Hyderabad" << endl;
11       }
12
13       void display()
14       {
15           cout << "BITS Goa" << endl;
16       }
17 };
18
19 int main() {
20     Example1 ex;
21     ex.display();
22 }
```

CLASS INHERITANCE IN C++

Why is inheritance used in C++?

```
class Person {
   private:
      string name;
      int Aadhaar;
   public:
      void print();
      string getName();
};
```

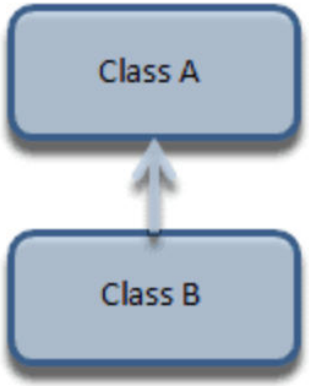
(Base/parent/superclass)

```
class Student : public Person {
   private:
      string branch;
      int gradYear;
      double cgpa;
      string idNo;
   public:
      void print();
};
```

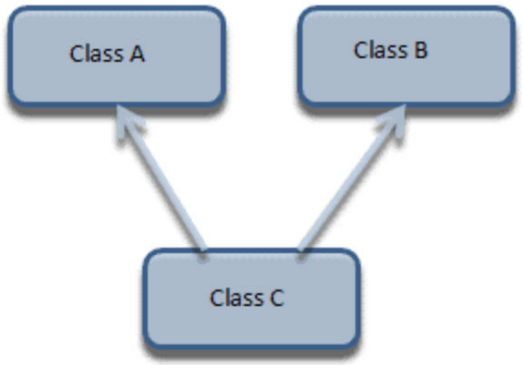
(Derived/child/subclass)

How will you draw the class inheritance diagram?

EXAMPLES



(Single Inheritance)



(Multiple Inheritance)

```cpp
#include <iostream>
#include <string>
using namespace std;
class Animal
{
   string name=" ";
   public:
   int tail = 1;

};
class Dog : public Animal
{
   public:
   void voiceAction()
   {
      cout<<"Barks!";
   }
};
int main()
{
   Dog d;
   cout<<"Dog has "<<d.tail<<" tail"<<endl;
   cout<<"Dog ";
   d.voiceAction();
}
```

```
Dog has 1 tail
Dog Barks!
```

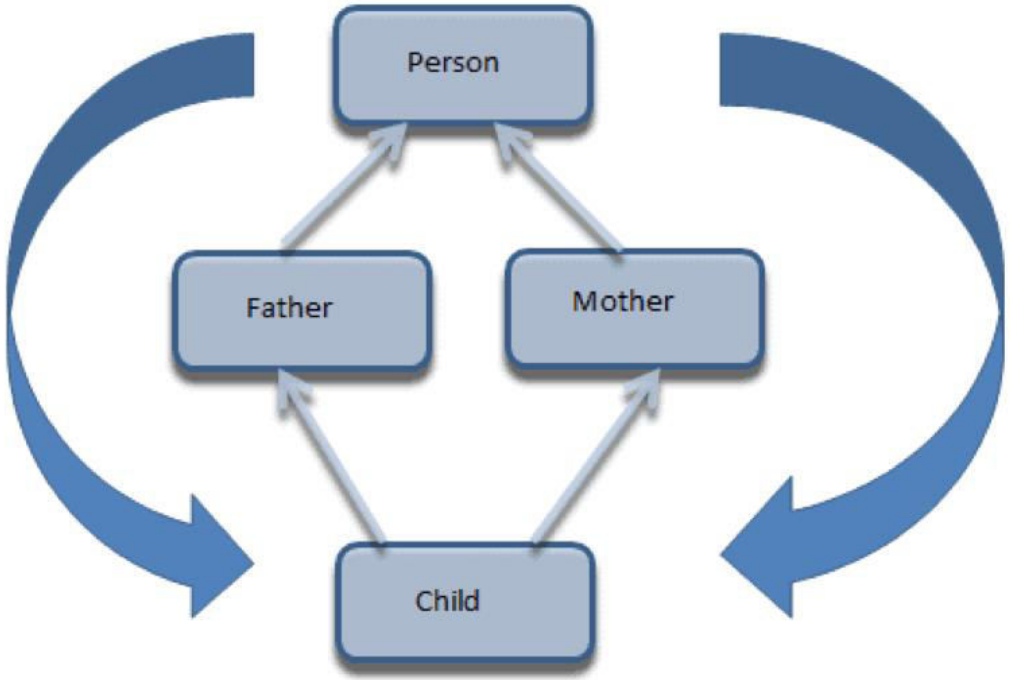
```cpp
#include <iostream>
using namespace std;
class student_marks {
protected:
  int rollNo, marks1, marks2;
public:
  void get() {
  cout << "Enter the ID No.: "; cin >> rollNo;
  cout << "Enter the Midsem and Compre marks: "; cin >> marks1 >> marks2;
  }
};
class lab_marks {
protected:
 int lmarks;
public:
 void getlm() {
 cout << "Enter the mark for lab exam: "; cin >> lmarks;
 }
};

class Result : public student_marks, public lab_marks {
   int total_marks;
   public:
   void display()
   {
      total_marks = (marks1 + marks2 + lmarks);
      cout << "\nID No: " << rollNo << "\nTotal marks: " << total_marks;
   }
};
int main()
{
   Result res;
   res.get();
   res.getlm();
   res.display();
}
```

input

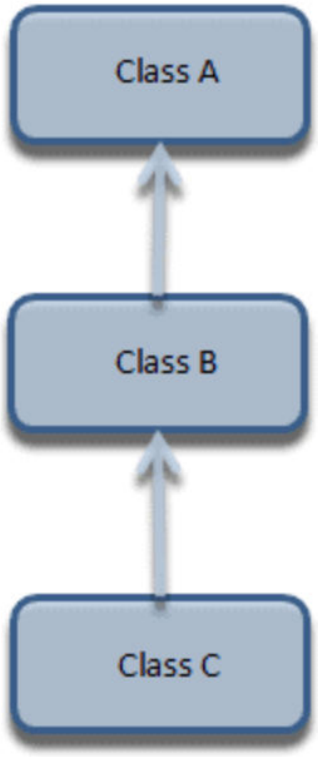
```
Enter the Midsem and Compre marks: 50 80
Enter the mark for lab exam: 30

ID No: 1
Total marks: 160
```

EXAMPLES CONTINUED…

(Sol: Make base class virtual)



(The Diamond Problem in C++)



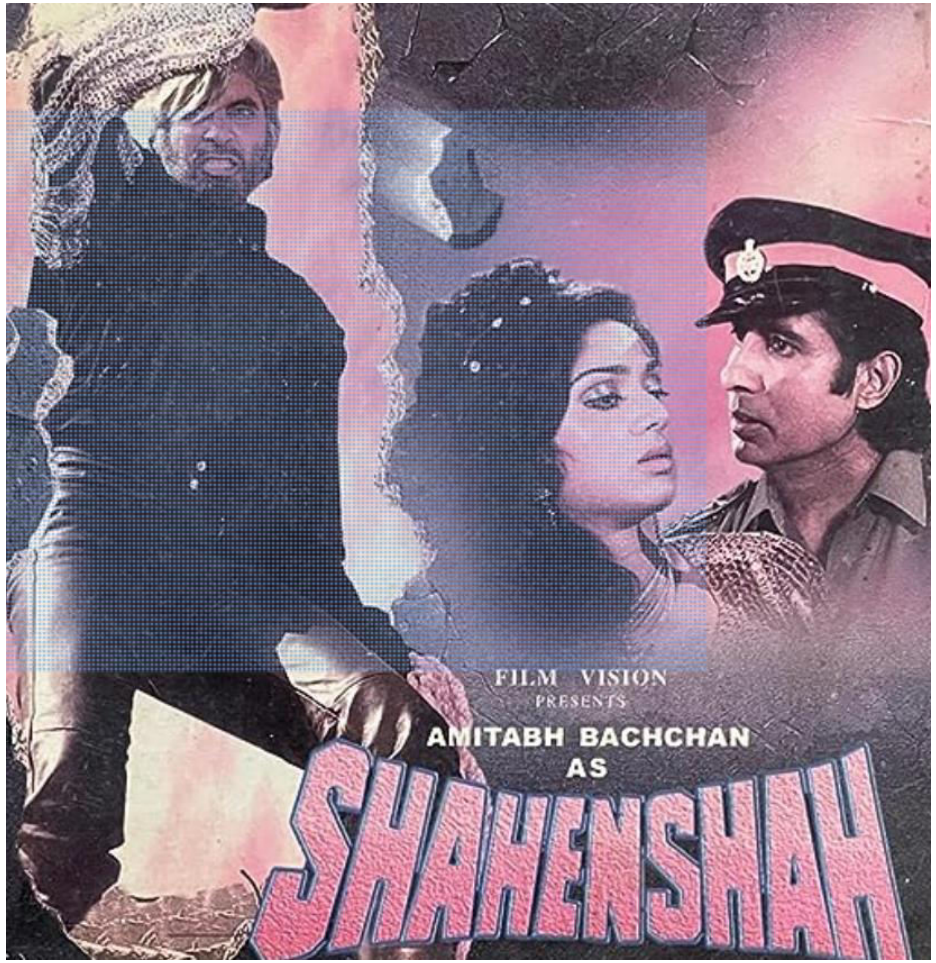
(Multi-level Inheritance)

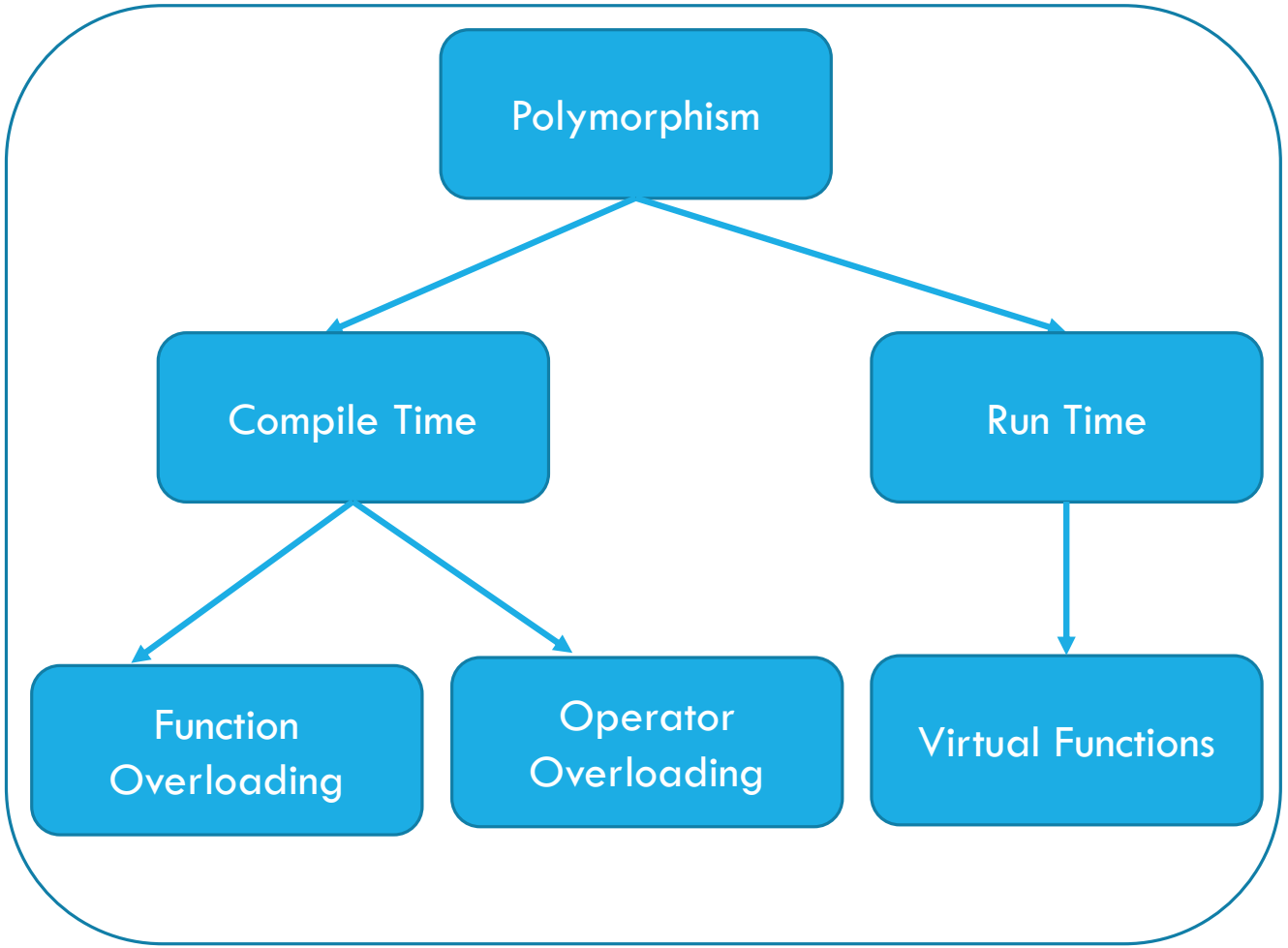
(Img. Source: www.softwaretestinghelp.com)

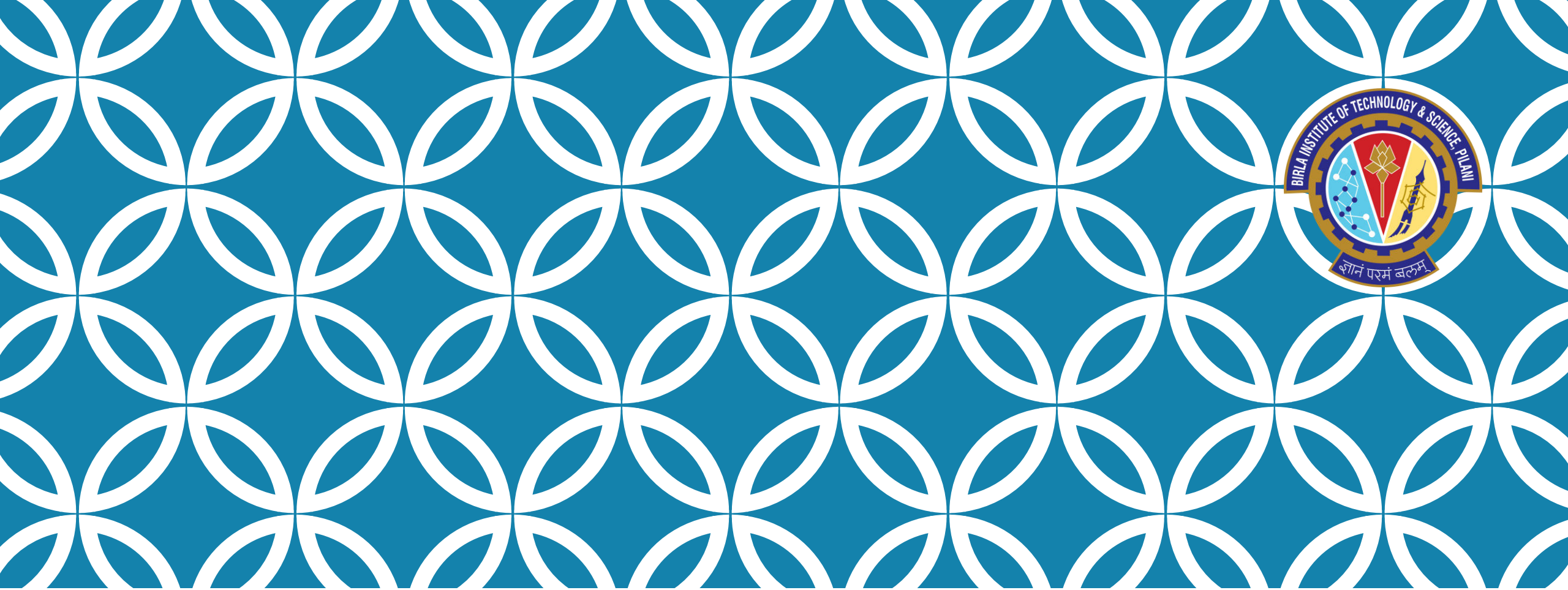
```
#include <iostream>
#include <string>
using namespace std;
class Animal
{
   string name=" ";
   public:
   int tail = 1;
};
class Dog : public Animal
{
   public:
   void voiceAction()
   {
      cout<<"Barks!";
   }
};
class Puppy : public Dog{
   public:
   void weeping()
   {
      cout<<"Sheds tears!";
   }
};
int main()
{
   Puppy p;
   cout<<"Puppy has "<<p.tail<<" tail"<<endl;
   cout<<"Puppy ";
   p.voiceAction();
   cout<<" Puppy ";
   p.weeping();
}
```

```
Puppy has 1 tail
Puppy Barks! Puppy Sheds tears!
```

POLYMORPHISM IN C++







BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) C++ CONTINUED…

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

POLYMORPHISM: FUNCTION OVERLOADING

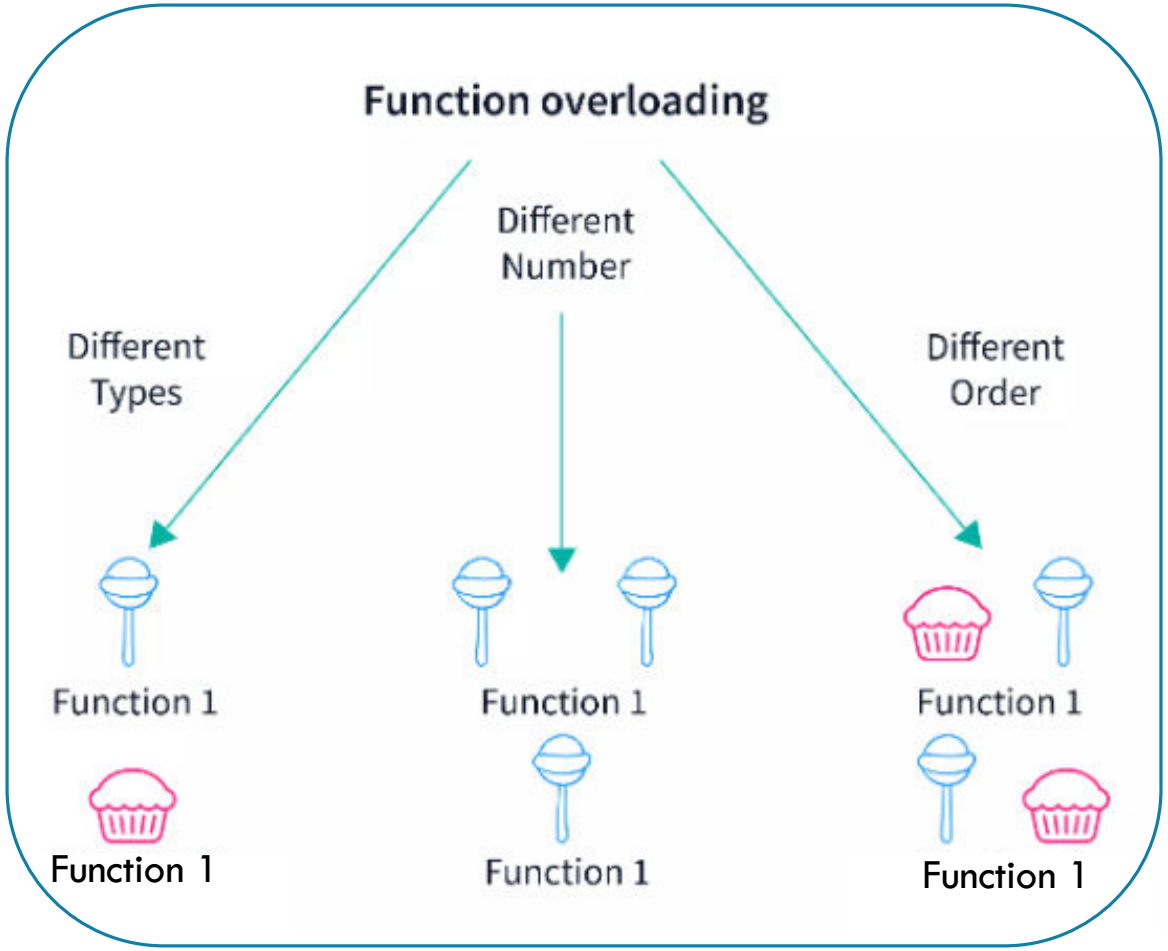
```cpp
#include <iostream>
using namespace std;

int square ( int x ) {
    return x * x;
}
double square( double y ) {
    return y * y;
}

int main()
{
    int i = square (6);
    cout << i << endl;

    double f = square (5.5);
    cout << f << endl;
    return 0;
}
```

main.cpp

```cpp
#include <iostream>
using namespace std;
class Add
 {
    public:
    int sum(int a,int b)
    {
        return (a+b);
    }
    int sum(int a,int b, int c)
    {
       return (a+b+c);
    }
 };
int main()
{
    Add obj;
    cout<<obj.sum(35, 10)<<endl;
    cout<<obj.sum(100, 50, 50);
    return 0;
}
```

```
45
200

...Program finished with exit code 0
Press ENTER to exit console.
```

CONTINUED...




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

int area(int length) {
    return length * length;
}

int area(int length, int breadth = 1) {

    return length * breadth;
}

int main() {

    int area_ = area(20);
    cout<<"Area square(length = 10) = "<<area_<<endl;

    return 0;
}
```

input

Compilation failed due to following error(s).

```
main.cpp: In function 'int main()':
main.cpp:15:21: error: call of overloaded 'area(int)' is ambiguous
   15 |     int area_ = area(20);
      |                 ~~~~^~~~
main.cpp:4:5: note: candidate: 'int area(int)'
    4 | int area(int length) {
      |     ^~~~
main.cpp:8:5: note: candidate: 'int area(int, int)'
    8 | int area(int length, int breadth = 1) {
      |     ^~~~
```

OPERATOR OVERLOADING

```
#include <iostream>
using namespace std;
int main()  {
    int a = 45;
    int b;
    b = a;
    cout << “value of b: “ <<b;
    return 0;
}
```

```
value of b: 45
```

```
#include <iostream>
using namespace std;
class employee {
public:
    int empno;
    float salary;
};
int main() {
    employee e1= {123, 60000.50}, e2;
    e2 = e1;
    cout << e1.empno << ‘\t’ <<  e2.salary;
    return 0;
}
```

```
123 60000.5
```

CONTINUED ...

```
#include <iostream>
using namespace std;
int main()  {
    int a = 45;
    int b;
    b = a;
    int c = a + b;
    cout << "value of c: " << c;
    return 0;
}
```

```
value of c: 90
```

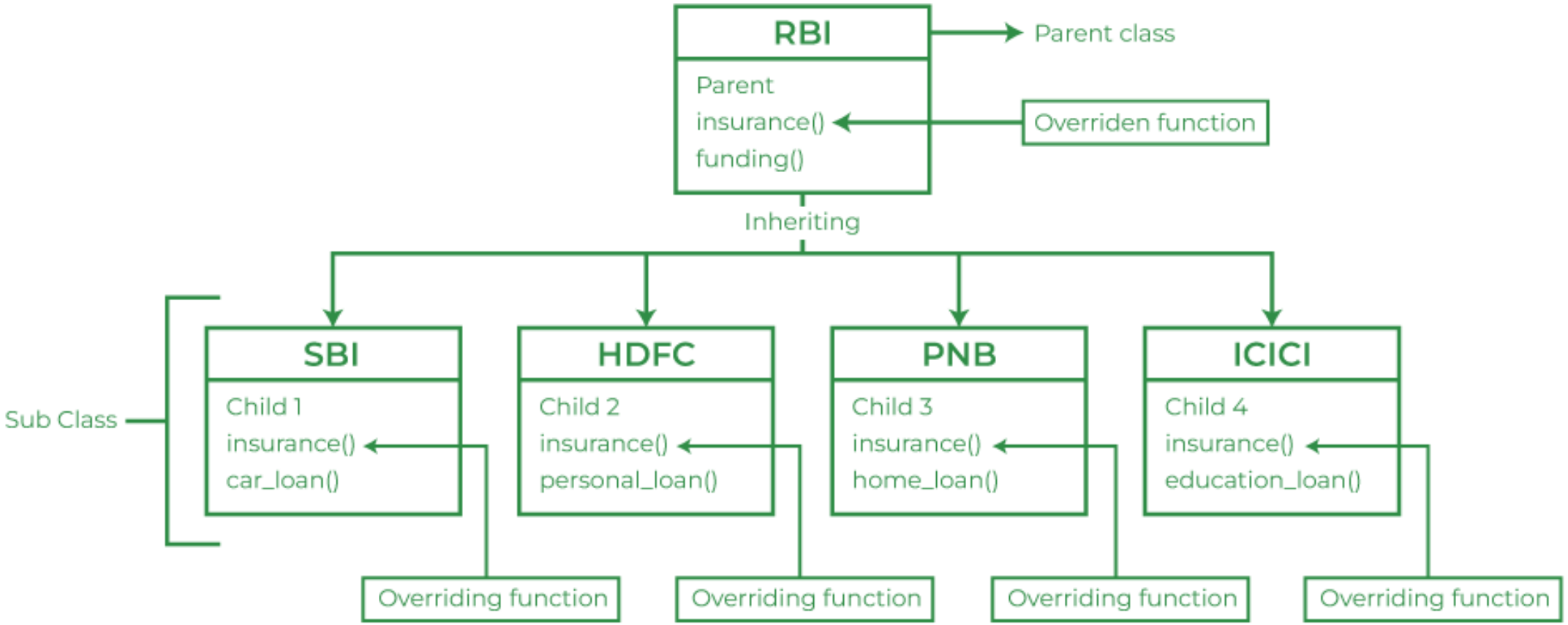
```
1  #include <iostream>
2  #include <string.h>
3  using namespace std;
4  class AddString {
5  public:
6      char str[50];
7      AddString() {}
8      AddString(char str[]) { strcpy(this->str, str); }
9      AddString operator+ (AddString& S2) {
10         AddString S3;
11         strcat(this->str, S2.str);
12         strcpy(S3.str, this->str);
13         return S3;
14     }
15 };
16 int main() {
17     char str1[] = "BITS,Pilani\t";
18     char str2[] = "Hyderabad Campus";
19     AddString a(str1);
20     AddString b(str2);
21     AddString c;
22     c = a + b;
23     cout << c.str;
24     return 0;
25 }
```

Operator Overloading

```
BITS,Pilani Hyderabad Campus
```

Runtime Polymorphism

FUNCTION OVERRIDING IN C++



Img. Source: www.geeksforgeeks.org

main.cpp

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

class Base {
   public:
     void print() {
         cout << "In Base Function" << e
     }
};

class Derived : public Base {
   public:
     void print() {
         cout << " In Derived Function"
     }
};

int main() {
     Derived d;
     d.print();
     return 0;
}
```

```
In Derived Function


...Program finished with exit code 0
Press ENTER to exit console.
```

```cpp
#include <bits/stdc++.h>
using namespace std;
class base
{
public:
    virtual void print ()
    { cout<< "Inside base class's print function" <<endl; }

    void show ()
    { cout<< "Inside base class" <<endl; }
};
class child:public base
{
public:
    void print ()
    { cout<< "Inside child class's print function" <<endl; }

    void show ()
    { cout<< "Inside derived class" <<endl; }
};
int main() {
    base *b;
    child c;
    b = &c;
    //virtual function, bound at runtime (Runtime polymorphism)
    b->print();
    // Non-virtual function, bound at compile time
    b->show();
    return 0;
}
```

```
Inside child class's print function
Inside base class
```

FRIEND CLASS IN C++

- ✓ A friend class is a class whose members have access to the private members of another class.
- ✓ Friendship is NOT transitive.
- ✓ Can a friend not access protected members?

Result

CPU Time: 0.00 sec(s), Memory: 3424 kilobyte(s)

```
49
```

```
1  #include <iostream>
2  using namespace std;
3  class Square;
4  class Rectangle {
5      int width, height;
6    public:
7      int area (){return (width * height);}
8      void convert (Square a);
9  };
10 class Square {
11   friend class Rectangle;
12   private:
13     int side;
14   public:
15     Square (int a):side(a) {}
16 };
17
18 void Rectangle::convert (Square a) {
19   width = a.side;
20   height = a.side;
21 }
22 int main () {
23   Rectangle rect;
24   Square sqr (7);
25   rect.convert(sqr);
26   cout << rect.area();
27   return 0;
28 }
```

FRIEND FUNCTION IN C++

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

```
void min (A a, B b)
{
   if (a.x < b.y)
         cout<< a.x << std::endl;
   else
         cout<< b.y << std::endl;
}
  int main ()
{
   A a;
   B b;
    a. setdata (100);
    b. setdata (250);
    cout << "Min:";
    min (a, b);
    return 0;
}
```

```
Min:100
```

PROTECTED ACCESS SPECIFIER

```cpp
#include <iostream>
using namespace std;
class BitsPilani {
    private:string Museum;
    public: int YearEst;
    BitsPilani (){
        Museum = "BirlaMuseum";
        YearEst = 0;
        FootballGround = "Nil";
    }
    protected: string FootballGround;
};
class BitsHyd : public BitsPilani {
    public: void DisplayGround(){
        FootballGround = "Grass";
        cout <<"Football Ground is made up of:"<<FootballGround<<endl;
    }
    void DisplayEst () {
        cout << "BITS Pilani was established in:" << YearEst << endl;
    }
};
int main () {
    BitsHyd obj;
    obj.YearEst = 1964;
    obj.DisplayGround();
    obj.DisplayEst();
    return 0;
}
```

Access Specifier	Same Class	Outside Class	Derived Class
Public Modifier	YES	YES	YES
Private Modifier	YES	NO	NO
Protected Modifier	YES	NO	YES

CPU Time: 0.00 sec(s), Memory: 3436 kilobyte(s)

```
Football Ground is made up of:Grass
BITS Pilani was established in:1964
```

Can you access Museum within BitsHyd?

Car and Will of Parents: who can access in what mode?

ABSTRACT CLASSES IN C++

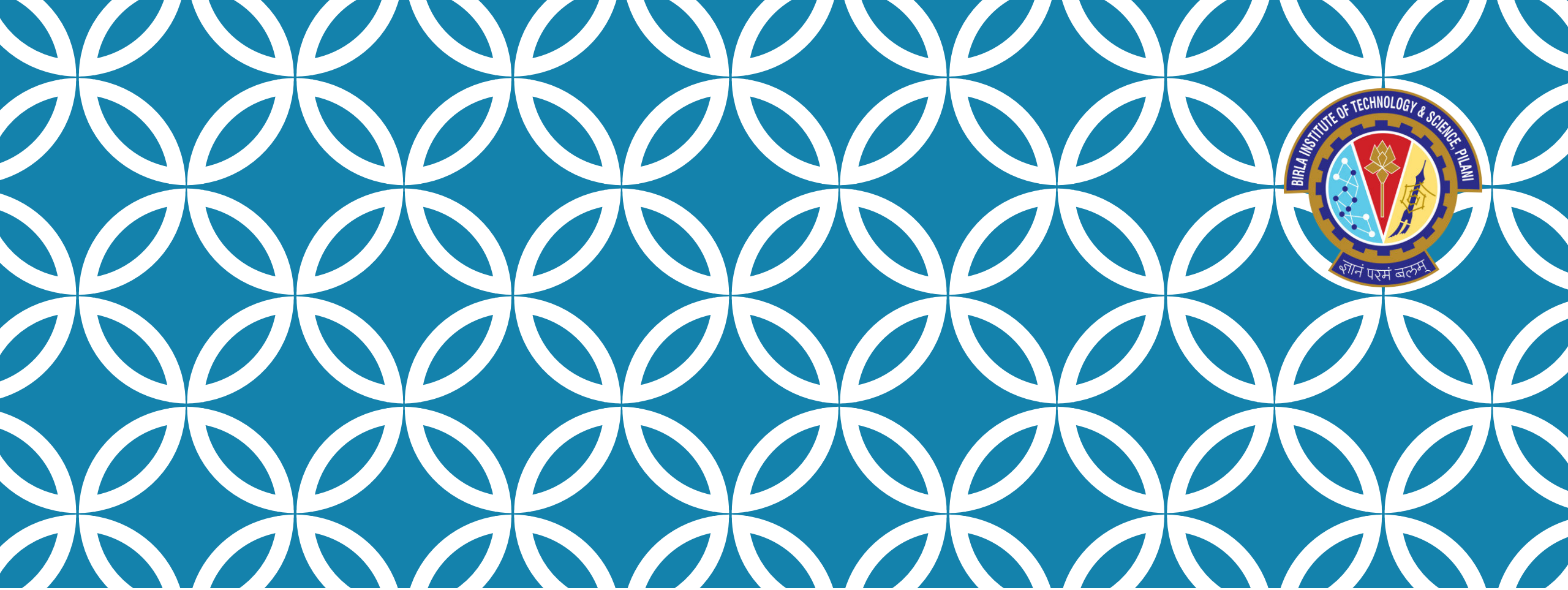
```cpp
#include <iostream>
using namespace std;

class Parent  //Base class
{
    public:
    virtual void show() = 0;    // Pure Virtual Function
};

class Child:public Parent //Derived class
{
    public:
    void show()
    {
        cout << "Implementation of Virtual Function in Child cla
    }
};

int main()
{
    Parent *b;
    Child c;
    b = &c;
    b->show();
}
```

```
Implementation of Virtual Function in Child class
```

main.cpp

```cpp
#include <iostream>
using namespace std;
class Shape {
   public:
      virtual int Area() = 0;
      void setWidth(int w) {
         width = w;
      }
      void setHeight(int h) {
         height = h;
      }
   protected:
      int width;
      int height;
};
class Rectangle: public Shape {
   public:
      int Area() {
         return (width * height);
      }
};
class Triangle: public Shape {
   public:
      int Area() {
         return (width * height)/2;
      }
};
int main() {
  Rectangle R;
  Triangle T;

  R.setWidth(3);
  R.setHeight(10);

  T.setWidth(10);
  T.setHeight(4);

  cout << "The area of the rectangle is: " << R.Area() << endl;
  cout << "The area of the triangle is: " << T.Area() << endl;
}
```

```
The area of the rectangle is: 30
The area of the triangle is: 20
```



BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) C++ CONTINUED…

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

DESIGN PATTERNS: TEMPLATES IN C++

```cpp
#include <iostream>
#include <string>
using namespace std;
template <typename T>
inline T const& Maximum (T const& a, T const& b) {
   return a < b ? b:a;
}

int main () {
   int i = 46;
   int j = 357;
   cout << "Maximum integer:" << Maximum(i, j) << '\t';

   float p = 345.8;
   float q = 577.5;
   cout << "Maximum float:" << Maximum(p, q) << endl;

   string r = "BITS";
   string s = "Pilani";
   cout << "Larger string:" << Maximum(r, s) << endl;

   return 0;
}
```

```
Maximum integer:357 Maximum float:577.5
Larger string:Pilani
```

```cpp
#include <iostream>
using namespace std;
template <typename T>
class Array {
   private: T *ptr; int size;
   public:
      Array(T arr[], int s);
      void print();
};
template <typename T>
  Array <T>::Array (T arr[], int s) {
    ptr = new T[s];
    size = s;
    for(int i = 0; i < size; i++) ptr[i] = arr[i];
}
template <typename T>
  void Array<T>::print() {
    for (int i = 0; i < size; i++) cout<<" "<<*(ptr + i);
    cout<<endl;
}
int main() {
  int arr1 [5] = {10, 20, 30, 40, 50};
  double arr2 [5] = {3.5, 6.5, 7.2, 7.3, 7.9};
  Array <int> a (arr1, 5);
  Array <double> b (arr2, 5);
  a.print();
  b.print();
  return 0;
}
```

```
10 20 30 40 50
3.5 6.5 7.2 7.3 7.9
```

WHAT TYPE?

```cpp
#include <iostream>
#include <vector>
#include <cstdlib>
#include <string>
#include <stdexcept>
using namespace std;

template <class T>
class Stack {
   private:
      vector<T> elems;
   public:
      void push(T const&);
      void pop();
      T top() const;

      bool empty() const {
         return elems.empty();
      }
};

template <class T>
void Stack<T>::push (T const& elem) {
   elems.push_back(elem);
}

template <class T>
void Stack<T>::pop () {
   if (elems.empty()) {
      throw out_of_range("Stack<>::pop(
   }
   elems.pop_back();
}
```

```cpp
template <class T>
T Stack<T>::top () const {
   if (elems.empty()) {
      throw out_of_range("Stack<>::top(): empty stack");
   }
   return elems.back();
}

int main() {
   try {
      Stack<int>         intStack;
      Stack<string> stringStack;
      intStack.push(345);
      cout << intStack.top() <<endl;

      stringStack.push("BITS F232");
      cout << stringStack.top() << std::endl;
      stringStack.pop();
      stringStack.pop();
   } catch (exception const& ex) {
      cerr << "Exception case: " << ex.what() <<endl;
      return -1;
   }
}
```

```
345
BITS F232
Exception case: Stack<>::pop(): empty stack
```

STANDARD TEMPLATE LIBRARY (STL) IN C++

- A library of container classes, algorithms, and iterators.

Can you name some?

```
Size of the vector: 1
Expanded size: 4
Vaue of vector0:56.5
Vaue of vector1:57.5
Vaue of vector2:58.5
Vaue of vector3:59.5
Value through iterator= 56.5
Value through iterator= 57.5
Value through iterator= 58.5
Value through iterator= 59.5
```

$$\text{InitialValue} + \sum_{i=0}^{n-1} a[i]$$

How to do this using STLs?

Can you name some of the STL functions in this code?

STL on strings:
insert, append, swap, size, resize, reverse etc.

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4  int main() {
5     vector <double> v;
6     int i;
7     v.push_back (56.5);
8     cout << "Size of the vector: " << v.size() << endl;
9     for(i = 1; i < 4; i++) {
10       v.push_back(v[0] + i);
11    }
12    cout << "Expanded size: " << v.size() << endl;
13
14    for(i = 0; i < 4; i++) {
15       cout <<"Vaue of vector"<<i<< ":"<< v[i] << endl;
16    }
17
18    vector<double>::iterator t = v.begin();
19    while( t != v.end()) {
20       cout << "Value through iterator= " << *t<< endl;
21       t++;
22    }
23    return 0;
24 }
```

ELEMENTARY DATA STRUCTURES: ARRAYS AND LINKED LISTS

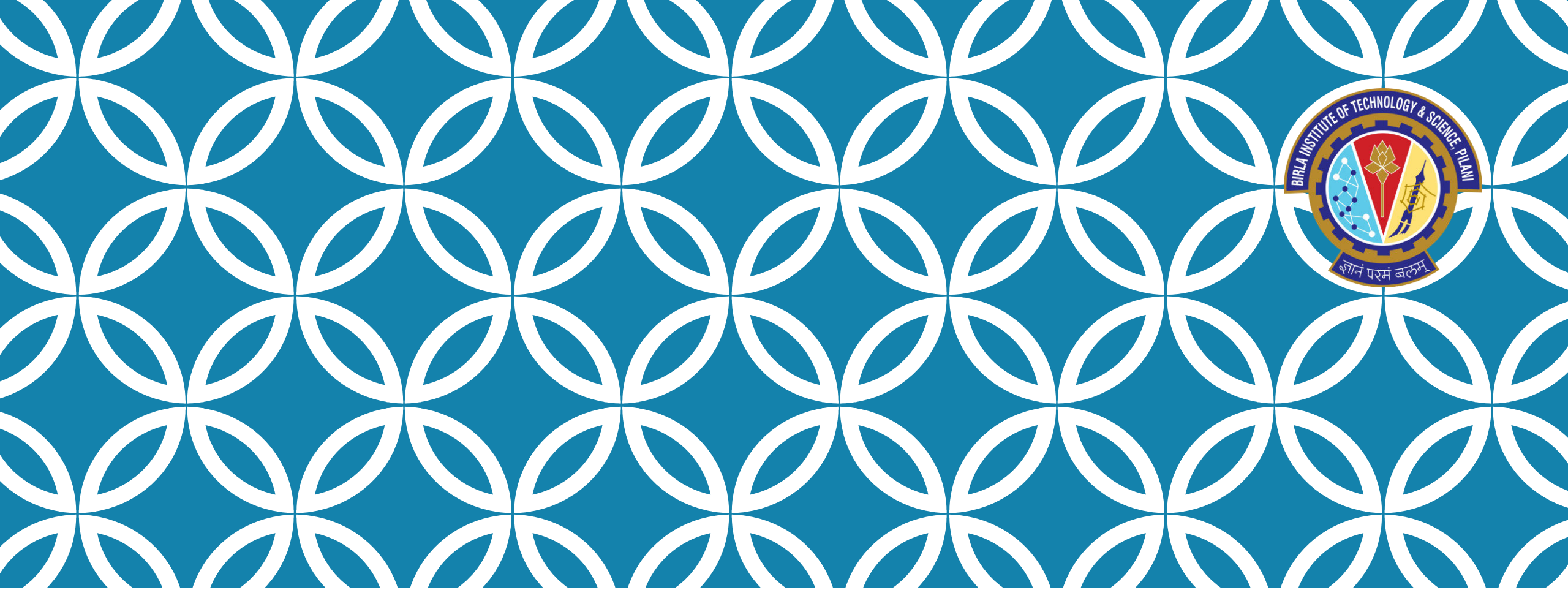
ARRAYS: LINEAR DATA STRUCTURES

- What are Arrays?
- Can you give some examples?
- Why are they called linear data structures?

Applications of arrays: Maths (vectors, matrices, polynomials,…), databases, compilers (control flow), dynamic memory allocations etc.

THANK YOU!

Next Class: Dynamic Arrays, and Linked List…



BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) ARRAYS AND LINKED LISTS

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

DYNAMIC ARRAYS EXAMPLE: LAB3

- Let us understand the operations needed to implement a dynamic array: insert, remove etc.

```
244     arr.insertItem(5);
245     arr.insertItem(3);
246     arr.insertItem(11);
247
248     arr.display();
```

```
249
250     arr.insertItemAtIndex(1, 7);
251
252     arr.display();
253
254     arr.sort();
255
256     arr.display();
257
```

```
258     arr.insertItem(15);
259     arr.insertItem(16);
260
261     arr.display();
262
263     cout << arr.getSize() << endl;
264
```

```
265     arr.deleteItem(11);
266
267     arr.display();
268
269     arr.deleteItem(16);
270
271     arr.display();
272
```

(DynamicArray.cpp given in the next week's lab sheet)

```
void Dynamic1DArray :: shrink() {

    capacity >>= 1;

    int *newArr = new int[capacity];

    for (int i = 0; i < size; i++)
        newArr[i] = arr[i];

    // update the global array pointer
    arr = newArr;
}
```

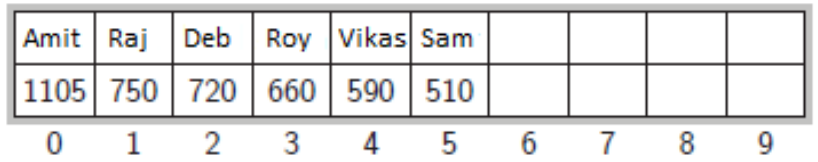
(shrink)

```
273     arr.deleteItemFromIndex(0);
274
275     arr.display();
276
277     arr.deleteItemFromIndex(1);
278
279     arr.display();
280
281     cout << arr.getSize() << endl;
```

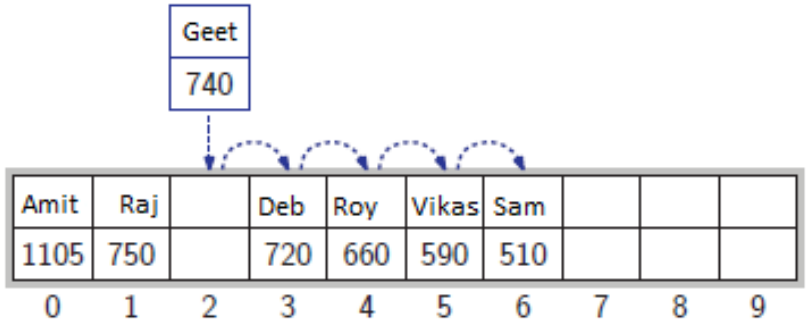
```
5 3 11
5 7 3 11
3 5 7 11
3 5 7 11 15 16
6
3 5 7 15 16
3 5 7 15
5 7 15
5 15
2
```

(Output)

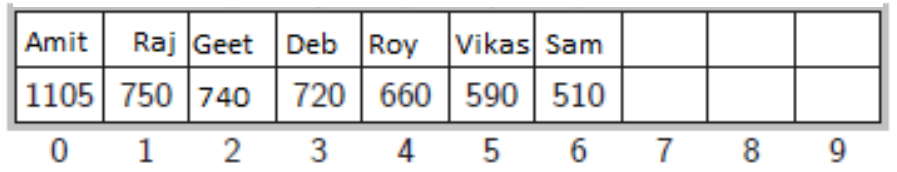
USING ARRAYS: AN EXAMPLE



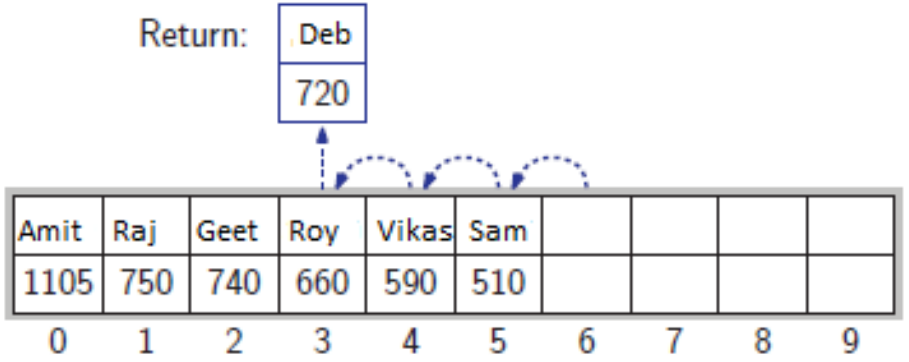
{An entries array of length 10 with 6 GameEntry objects (maxExntries: 10, numEntries: 6)}



{Preparing to add a new GameEntry object by shifting all the erntries with smaller scores to the right by one position}



{Copying the new entry into the position. Scenario after addition}



{Removing an element at index i requires moving all the entries at indices higher than i one position to the left}

IMPLEMENTATION: STORING GAME ENTRIES

```
class GameEntry {
public:
        GameEntry ( const string &n = "", int s = 0);
        string getName() const;
        int getScore() const;
private:
        string name;
        int score;
};
```

(A Class representing a Game entry)

```
GameEntry::GameEntry(const string &n, int s) : name(n),
score(s) { }
string GameEntry::getName() const { return name; }
int GameEntry::getScore() const { return score; }
```

(Constructor and member functions)

```
class Scores {
public:
        Scores(int maxEnt = 10);
        ~Scores();
        void add(const GameEntry &e);
        GameEntry remove(int i) ;
        void printAllScores();
private:
        int maxEntries; //maximum number of entries
        int numEntries;  //actual number of entries
        GameEntry *entries;
};
```

(A Class for storing Game scores)

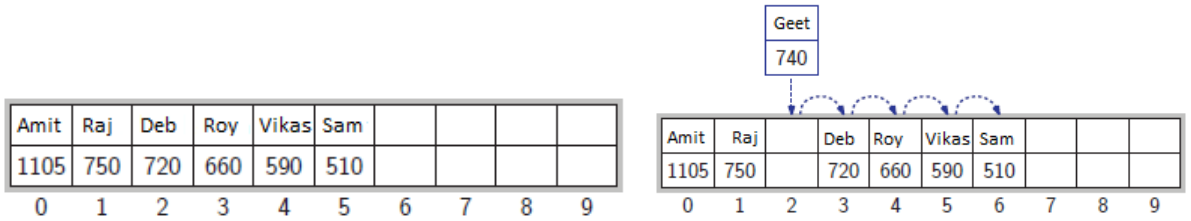
```
Scores::Scores(int maxEnt) {
    maxEntries = maxEnt; // save the max size
    entries = new GameEntry[maxEntries];
    numEntries = 0;
}
```

```
Scores::~Scores() {  delete[ ] entries; }
```

INSERTING INTO AND DELETING FROM ARRAY

```
void Scores::add(const GameEntry &e) {
    int newScore = e.getScore(); // score to add
    if (numEntries == maxEntries) { // the array is full
       if (newScore <= entries[maxEntries - 1].getScore())
            return; // not high enough - ignore
    }
    else  numEntries++; // if not full, one more entry

    int i = numEntries - 2; // start with the next to last
    while (i >= 0 && newScore > entries[i].getScore() ) {
         entries[i + 1] = entries[i]; // shift right if smaller
         i--;
     }
    entries[i + 1] = e; // put e in the empty spot
}
```

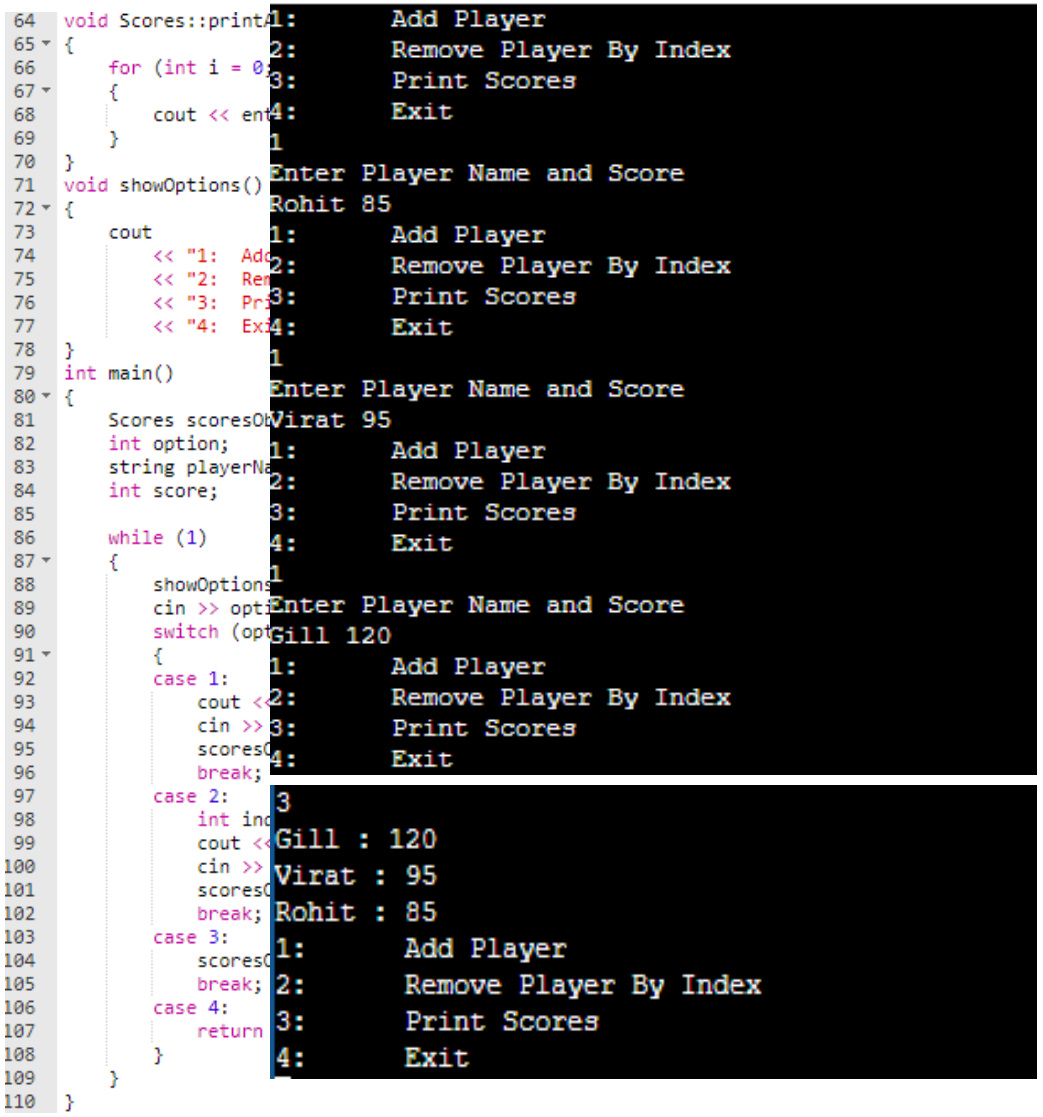
(Inserting a Game entry object)




```
GameEntry Scores::remove(int i)
{
      if ((i < 0) || (i >= numEntries)) // invalid index
            throw("IndexOutOfBounds - Invalid index");
      GameEntry e = entries[i]; // save the removed object
      for (int j = i + 1; j < numEntries; j++)
            entries[j - 1] = entries[j]; // shift entries left
      numEntries--;    // one fewer entry
      return e;        // return the removed object
}
```

(Removing a Game entry object)

DRIVER AND OTHER CLASSES FOR GAME ENTRY EX.



```
64  void Scores::print
65  {
66      for (int i = 0
67      {
68          cout << en
69      }
70  }
71  void showOptions()
72  {
73      cout
74          << "1:  Ad
75          << "2:  Re
76          << "3:  Pr
77          << "4:  Ex
78  }
79  int main()
80  {
81      Scores scoresO
82      int option;
83      string playerNa
84      int score;
85
86      while (1)
87      {
88          showOptions
89          cin >> opt
90          switch (opt
91          {
92          case 1:
93              cout <<
94              cin >>
95              scores
96              break;
97          case 2:
98              int ind
99              cout <<
100             cin >>
101             scores
102             break;
103         case 3:
104             scores
105             break;
106         case 4:
107             return
108         }
109     }
110 }
```

```
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Rohit 85
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Virat 95
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Gill 120
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
```

```
3
Gill : 120
Virat : 95
Rohit : 85
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
```

```
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Gill 200
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
3
Gill : 200
Gill : 120
Virat : 95
Rohit : 85
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
```

(Lab3: GameEntry.cpp)

LAB3 TASKS: GAME ENTRY

```
4
Gill : 2
Virat : 1
Rohit : 1
1:  Add Player
2:  Remove Player By Index
3:  Print Scores
4:  Print Players Count
5:  Exit
```

(How many number of entries are there for each player? Option 3)

```
Enter max value and min value of the score range
400 300
Gill : 320
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Print Players Count
5:      Print Unique Scores
6:      Print Players in Score Range
7:      Print Master Player
8:      Exit
```

(Display players in a score range: Option 6)

```
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Rohit 85
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Virat 95
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Gill 120
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Gill 200
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
3
Gill : 200
Virat : 95
Rohit : 85
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
```

(display unique entries for each player?)

(GameEntry_Unique.cpp)

SORTING & SEARCHING IN AN ARRAY

```
void Dynamic1DArray ::sort()
{
    for (int j = 1; j < size; j++)
    {
        int key = arr[j];
        int i = j - 1;
        while (i > -1 && arr[i]>key)
        {
            arr[i + 1] = arr[i];
            i = i - 1;
        }
        arr[i + 1] = key;
    }
}
```

(Insertion Sort)

```
int Dynamic1DArray
::binarySearch(const int item)
{
    int low = 0, high = size - 1;
    while (low <= high){
       int mid = low + ((high -
                        low) >> 1);
        if (item == arr[mid])
            return mid;
        if (item < arr[mid])
            high = mid - 1;
        else
            low = mid + 1;
    }
    return -1; }
```

(Binary Search)

More sorting & searching algos later…

MULTI-DIMENSIONAL ARRAYS

Month

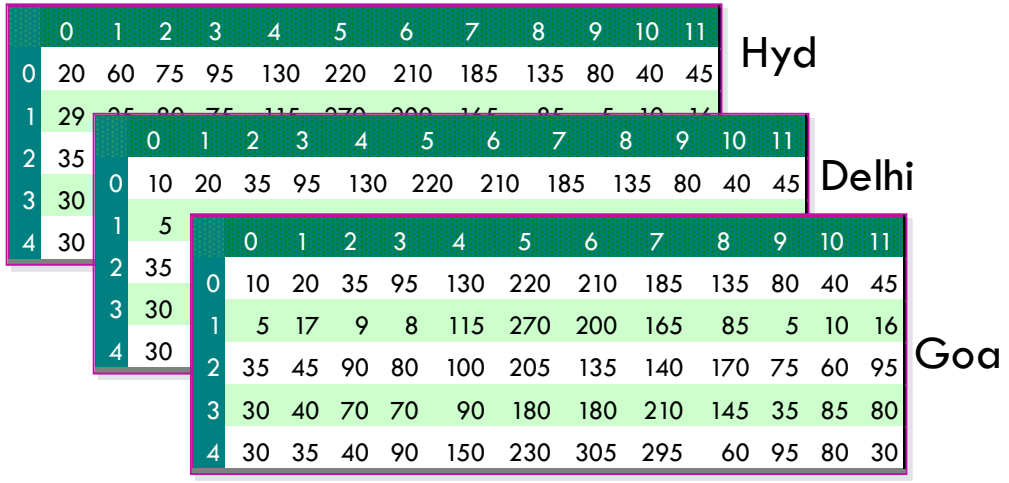
Year

	0	1	2	3	4	5	6	7	8	9	10	11
0	30	40	75	95	130	220	210	185	135	80	40	45
1	25	25	80	75	115	270	200	165	85	5	10	16
2	35	45	90	80	100	205	135	140	170	75	60	95
3	30	40	70	70	90	180	180	210	145	35	85	80
4	30	35	40	90	150	230	305	295	60	95	80	30

Average Yearly Rainfall (in mm of Hyd)

Arrays in C++ are one-dimensional. However, we can define a 2D array as "an array of arrays".

3-dimensional



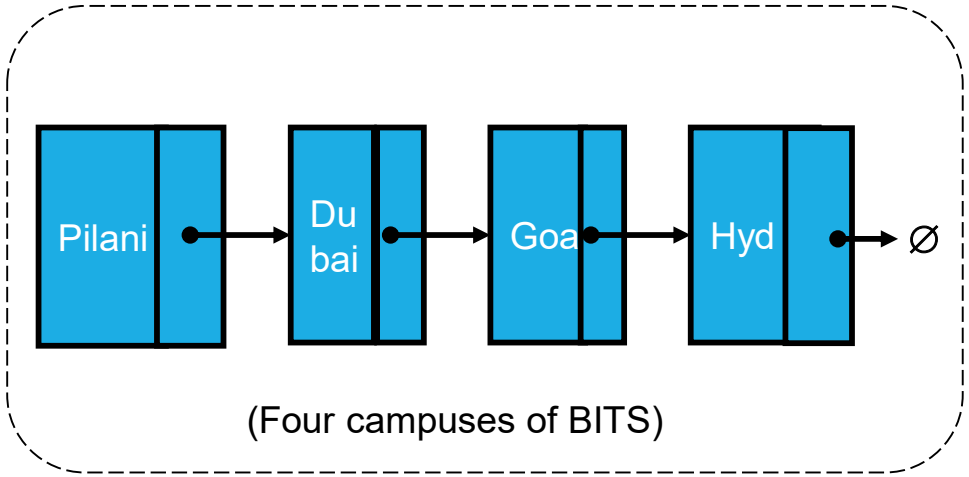

```
#include<iostream>
using namespace std;
int main()
{
    int x[5][12]={{30,40,75,95,130,220,210,185,135,80,40,45},
        {25,25,80,75,115,270,200,165, 85, 5,10, 16},
        {35,45,90,80,100,205,135,140,170,75,60,95},
        {30,40,70,70, 90,180,180,210,145,35,85,80},
        {30,35,40,90,150,230,305,295, 60,95,80,30}
};
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 12; j++)
        {
            cout << "Element at x[" << i
                 << "][" << j << "]: ";
            cout << x[i][j]<<endl;
        }
    }

    return 0;
}
```

```
Element at x[0][0]: 30
Element at x[0][1]: 40
Element at x[0][2]: 75
Element at x[0][3]: 95
Element at x[0][4]: 130
Element at x[0][5]: 220
Element at x[0][6]: 210
Element at x[0][7]: 185
Element at x[0][8]: 135
Element at x[0][9]: 80
Element at x[0][10]: 40
Element at x[0][11]: 45
Element at x[1][0]: 25
Element at x[1][1]: 25
Element at x[1][2]: 80
Element at x[1][3]: 75
Element at x[1][4]: 115
Element at x[1][5]: 270
Element at x[1][6]: 200
Element at x[1][7]: 165
Element at x[1][8]: 85
Element at x[1][9]: 5
Element at x[1][10]: 10
Element at x[1][11]: 16
Element at x[2][0]: 35
Element at x[2][1]: 45
Element at x[2][2]: 90
Element at x[2][3]: 80
Element at x[2][4]: 100
Element at x[2][5]: 205
Element at x[2][6]: 135
Element at x[2][7]: 140
Element at x[2][8]: 170
Element at x[2][9]: 75
Element at x[2][10]: 60
Element at x[2][11]: 95
Element at x[3][0]: 30
Element at x[3][1]: 40
Element at x[3][2]: 70
Element at x[3][3]: 70
Element at x[3][4]: 90
Element at x[3][5]: 180
Element at x[3][6]: 180
Element at x[3][7]: 210
Element at x[3][8]: 145
Element at x[3][9]: 35
Element at x[3][10]: 85
Element at x[3][11]: 80
Element at x[4][0]: 30
Element at x[4][1]: 35
Element at x[4][2]: 40
Element at x[4][3]: 90
Element at x[4][4]: 150
Element at x[4][5]: 230
Element at x[4][6]: 305
Element at x[4][7]: 295
Element at x[4][8]: 60
Element at x[4][9]: 95
Element at x[4][10]: 80
Element at x[4][11]: 30
```

SINGLY LINKED LISTS

- Linked list: A linear data structure?
- A singly linked list is a concrete data structure consisting of a sequence of nodes, where each node has?

Arrays	Vs.	Linked lists
1. Arrays are stored in contiguous location.		1. Linked lists are not stored in contiguous location.
2. Fixed in size.		2. Dynamic in size.
3. Memory is allocated at compile time.		3. Memory is allocated at run time.
4. Uses less memory than linked lists.		4. Uses more memory because it stores both data and the address of next node.
5. Elements can be accessed easily.		5. Element accessing requires the traversal of whole linked list.
6. Insertion and deletion operation takes time.		6. Insertion and deletion operation is faster.



How will you store mid-sem scores of say, 4 students in a linked list?

Applications: Implementation of Stacks, queues, Graphs (Adj list), Keep track of OS process states, Free memory blocks

IMPLEMENTING A SINGLY LINKED LIST

Step 1: Define a class for the Node

```
class StringNode {
   private:  string elem;
             StringNode* next;
   friend class StringLinkedList;
};
```

Step 2: Define a class for the Linked list

```
class StringLinkedList {
   public: StringLinkedList();
           ~StringLinkedList();
   bool empty() const;
   const string& front() const;
   void addFront(const string& e);
   void removeFront();
   private: StringNode* head;
};
```

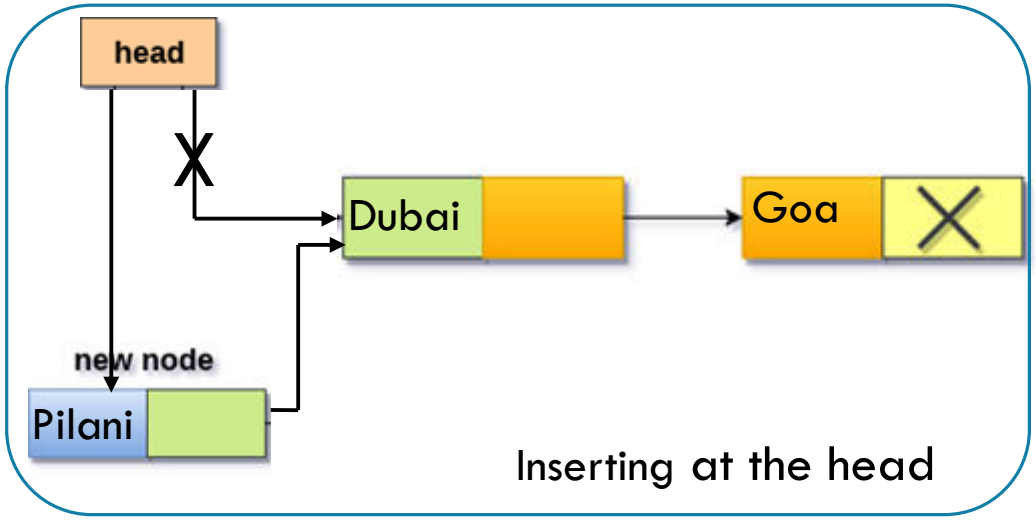
Step 3: Define a set of member functions for the Linked list class defined in Step 2

```
StringLinkedList::StringLinkedList()  :  head(???){  }
StringLinkedList::~StringLinkedList() {
   while(!empty())
        ???;
}

bool StringLinkedList::empty() const {  //Is list empty?
    return head == NULL;
}

const string& StringLinkedList::front() const {
return ???;
}
```

INSERTING & REMOVING AT THE HEAD OF LINKED LIST

1. Create a new node
2. Store data into this node
3. Have new node point to old head
4. Update head to point to new node

```
void StringLinkedList::addFront(const string& e)
{
   StringNode* v = new StringNode;
   v->elem = e;
   v->next = head;
   head = v;
}
```

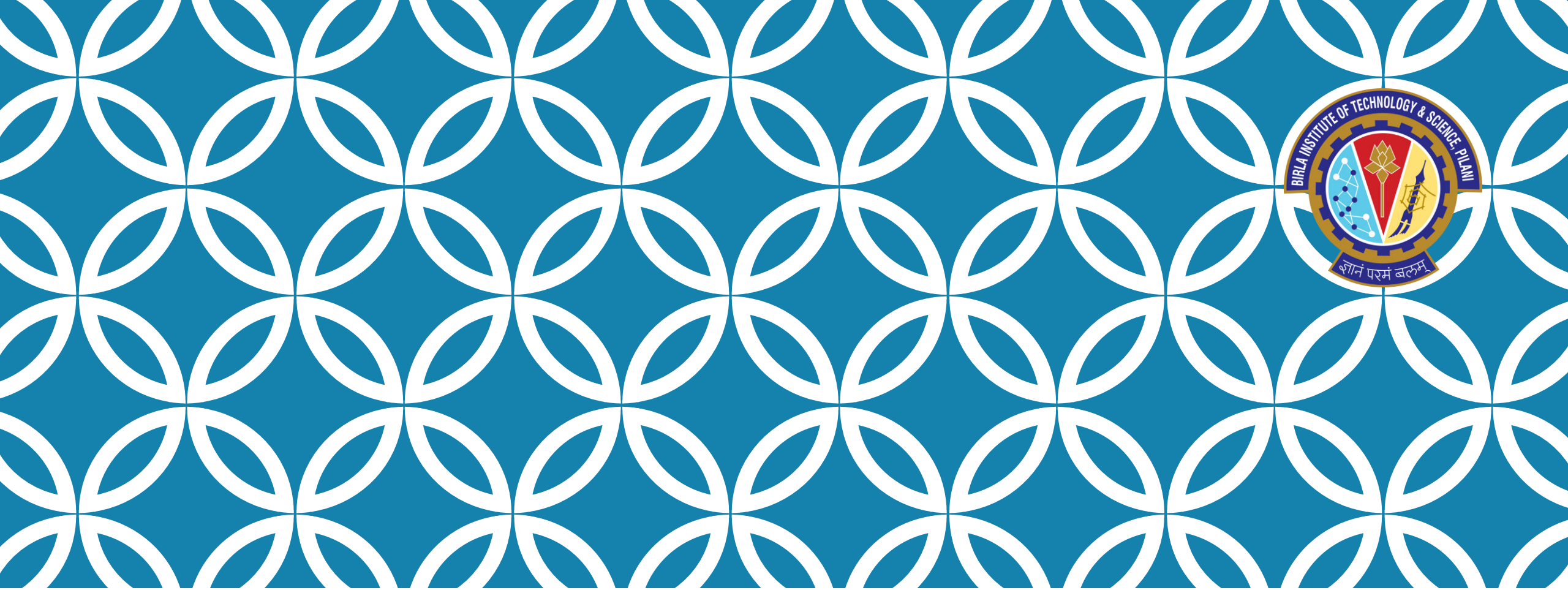


Inserting at the head

```
void StringLinkedList::removeFront()
{
  StringNode* old = head;
  head = old->next;
  delete old;
}
```

Deleting at the head

1. Save old head
2. Advance head to the next node
3. Delete the old head node

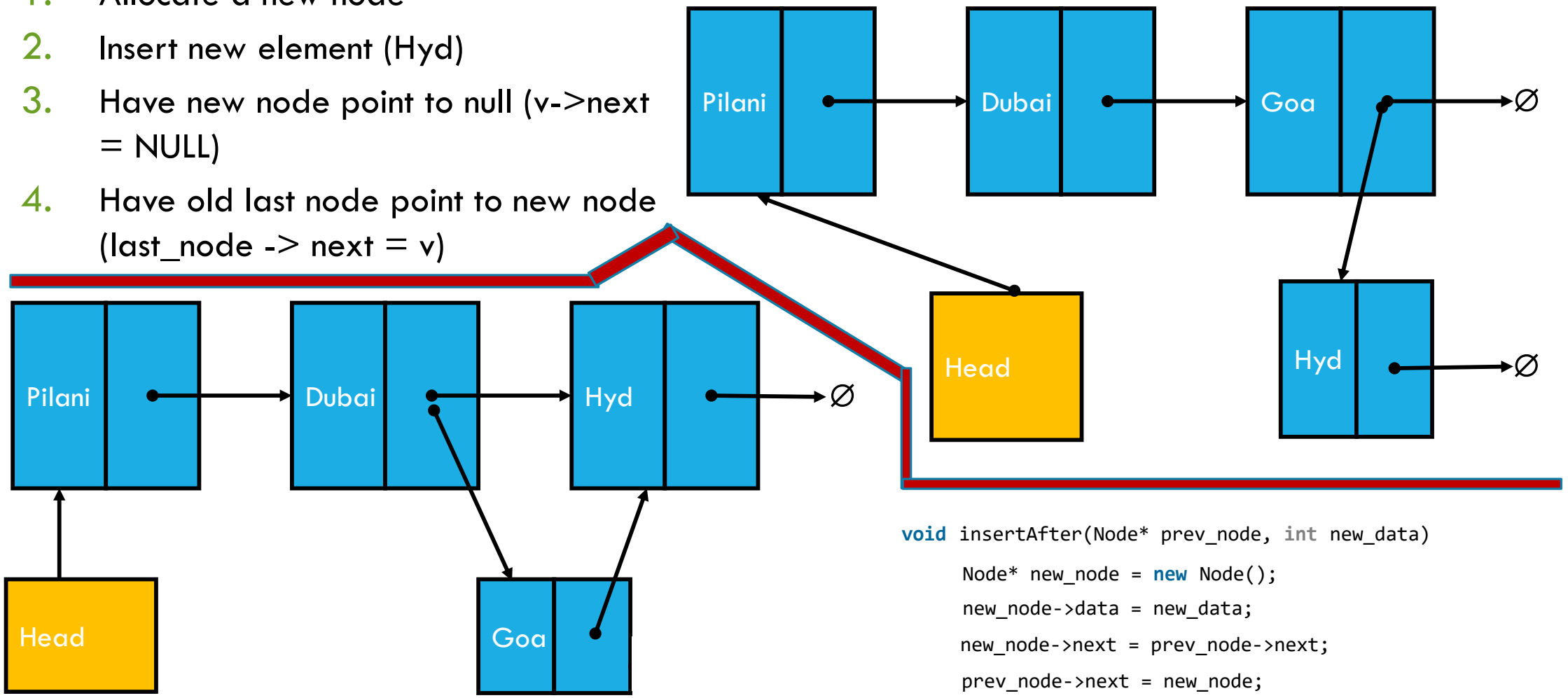


BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) LINKED LISTS CONTINUED…

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

INSERTING AT THE TAIL & INSIDE A LINKED LIST

1. Allocate a new node
2. Insert new element (Hyd)
3. Have new node point to null (v->next = NULL)
4. Have old last node point to new node (last_node -> next = v)

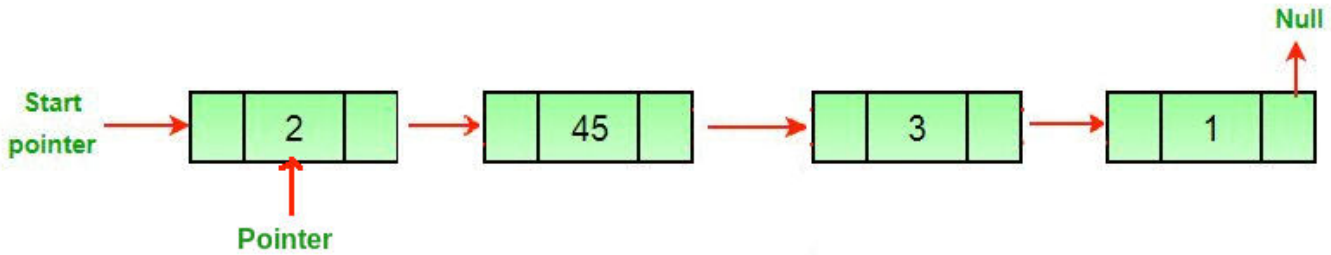



```
void insertAfter(Node* prev_node, int new_data)
    Node* new_node = new Node();
    new_node->data = new_data;
    new_node->next = prev_node->next;
    prev_node->next = new_node;
```

DELETING THE LAST NODE

Algorithm:

1. If (headNode == null) //if the first node is null
   then return null
2. If (headNode.next == null) //if there is only one node
   then free head and return null
3. while secondLast.next.next != null //traverse till secondLast
   secondLast = secondLast.nextNode
4. Delete last node and set the pointer of secondLast to null.

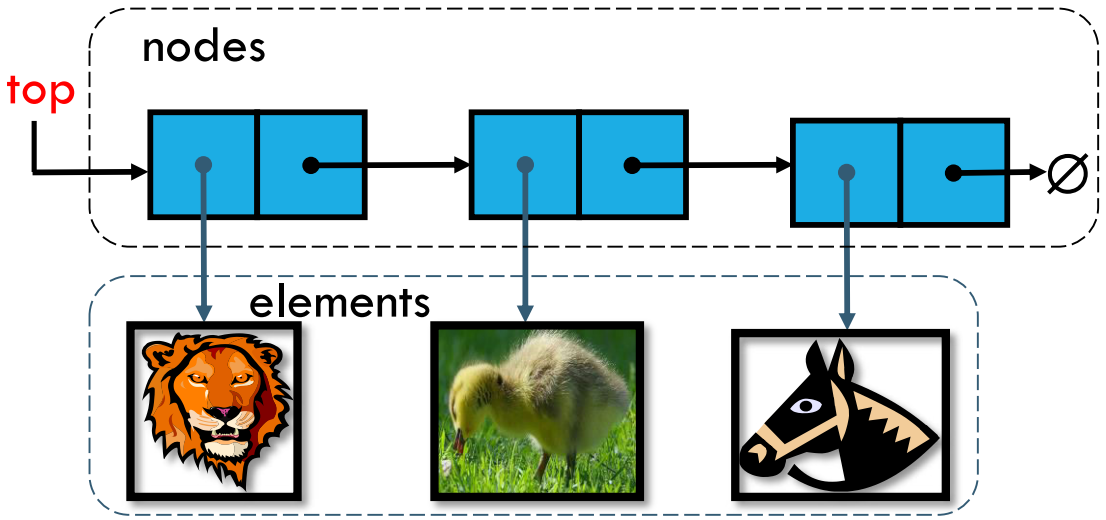


Img. Source: https://www.geeksforgeeks.org/

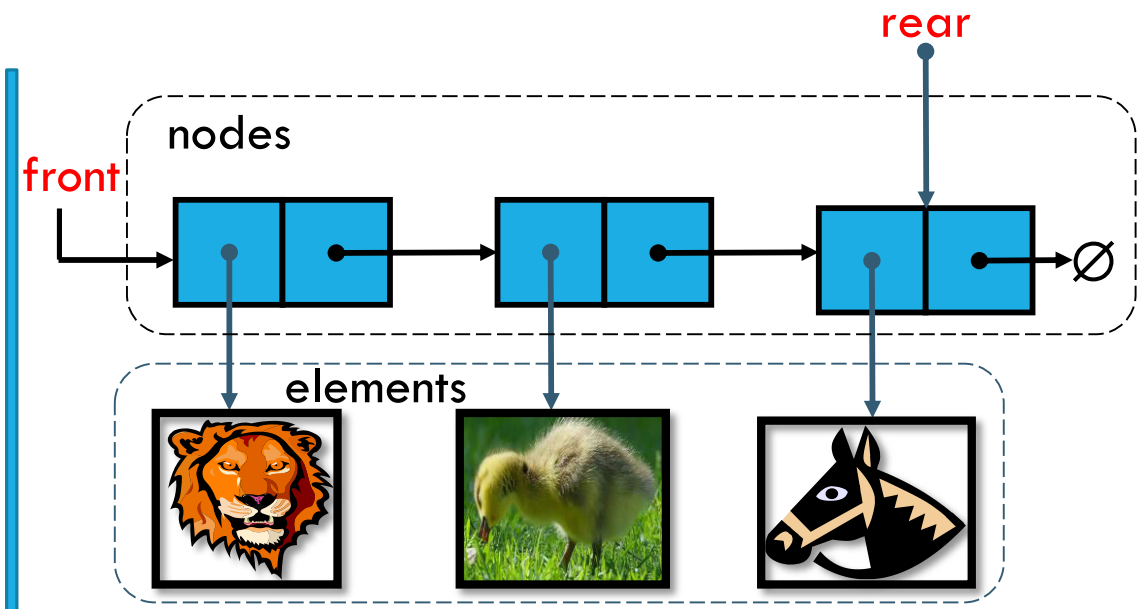
```
#include <iostream>
using namespace std;
struct Node {
    string data;
    struct Node* next;
};
Node* removeLastNode(struct Node* head) {
    if (head == NULL)
        return NULL;
    if (head->next == NULL) {
        delete head;
        return NULL;
    }
    Node* second_last = head;
    while (second_last->next->next != NULL)
        second_last = second_last->next;
    delete (second_last->next);
    second_last->next = NULL;
    return head;
}
void insertNode (struct Node** head_ref, string new_data) {
    struct Node* new_node = new Node;
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}
int main() {
    Node* head = NULL;
    insertNode(&head, "Hyd");
    insertNode(&head, "Goa");
    insertNode(&head, "Dubai");
    insertNode(&head, "Pilani");
    head = removeLastNode(head);
    cout << "After deleting the last node:"<<endl;
    for (Node* temp = head; temp != NULL; temp = temp->next)
        cout << temp->data << " ";
    return 0;
}
```

```
After deleting the last node:
Pilani Dubai Goa
```

STACK & QUEUE AS SINGLY LINKED LISTS



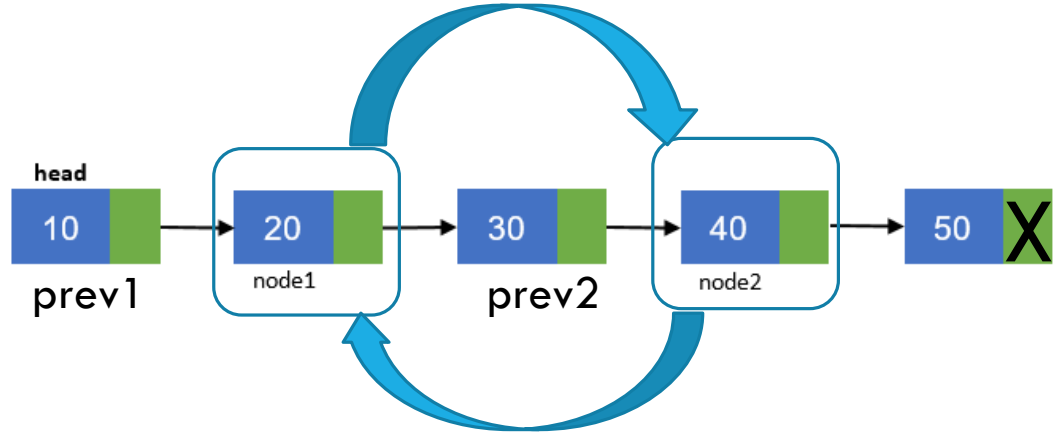
Stack: We can implement stack as linked list. Top element is stored as first element of the linked list.

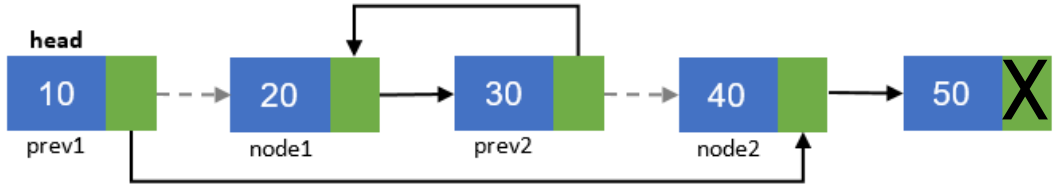


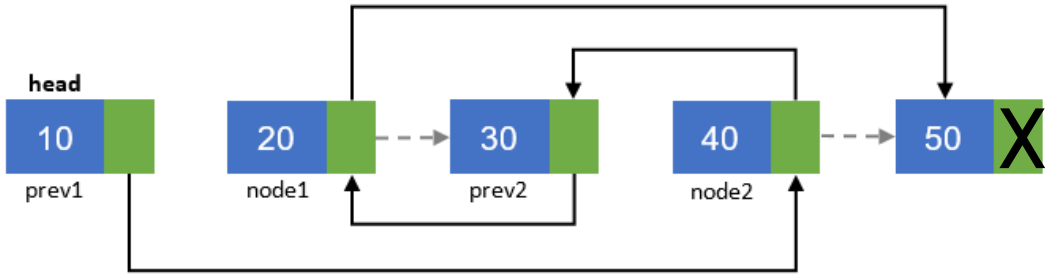
Queue: We can implement a queue as a linked list. Front element is stored as first element of the linked list, and rear element is stored as the last element.

Implementation in later chapters…

SWAPPING TWO NODES IN A LINKED LIST







Lab 4 next week

GENERIC SINGLY LINKED LISTS: USING TEMPLATES

```
#include <iostream>

using namespace std;

template<typename E>
class SLinkedList;                      //forward declare the class

template <typename E>
class SNode {                           // singly linked list node
private:
    E elem;                             // linked list element value
    SNode<E>* next;                     // next item in the list
    friend class SLinkedList<E>;        // provide SLinkedList access
};

template <typename E>
class SLinkedList {                     // a singly linked list
public:
    SLinkedList();                      // empty list constructor
    ~SLinkedList();                     // destructor
    bool empty() const;                 // is list empty?
    const E& front() const;             // return front element
    void addFront(const E& e);          // add to front of list
    void removeFront();                 // remove front item list
    void traverse();                    // traverse the list
private:
    SNode<E>* head;                     // head of the list
};

template <typename E>
SLinkedList<E>::SLinkedList()           // constructor
    : head(NULL) { }
```

```

template <typename E>
bool SLinkedList<E>::empty() const      // is list empty?
{ return head == NULL; }

template <typename E>
const E& SLinkedList<E>::front() const  // return front element
{ return head->elem; }

template <typename E>
SLinkedList<E>::~SLinkedList()          // destructor
{ while (!empty()) removeFront(); }

template <typename E>
void SLinkedList<E>::addFront(const E& e) {     // add to front of list
    SNode<E>* v = new SNode<E>;                 // create new node
    v->elem = e;                                // store data
    v->next = head;                             // head now follows v
    head = v;                                   // v is now the head
}

template <typename E>
void SLinkedList<E>::removeFront() {            // remove front item
    SNode<E>* old = head;                       // save current head
    head = old->next;                           // skip over old head
    delete old;                                 // delete the old head
}

template <typename E>
void SLinkedList<E>::traverse(){
    SNode<E> *temp = head;
    while(temp != NULL){
        cout<<temp->elem<<" ";
        temp = temp->next;
    }
    cout<<endl;
}
```

```
SNode<E>
*SLinkedList<E>::search
(const E &e){
//complete code here


}
```

(Lab 4)

LAB 4

```
+----------------------------------------+
Please enter one of the following choices:
1 : Add at the front
2 : Get frontmost element
3 : Remove front element
4 : Check if list is empty
5 : Traverse the list
6 : Search for an element
7 : Swap two nodes
8 : Exit
1
Enter the element: Rohit
+----------------------------------------+
Please enter one of the following choices:
1 : Add at the front
2 : Get frontmost element
3 : Remove front element
4 : Check if list is empty
5 : Traverse the list
6 : Search for an element
7 : Swap two nodes
8 : Exit
1
Enter the element: Virat
+----------------------------------------+
Please enter one of the following choices:
1 : Add at the front
2 : Get frontmost element
3 : Remove front element
4 : Check if list is empty
5 : Traverse the list
6 : Search for an element
7 : Swap two nodes
8 : Exit
2
Frontmost element is : Virat
+----------------------------------------+
Please enter one of the following choices:
1 : Add at the front
2 : Get frontmost element
3 : Remove front element
4 : Check if list is empty
5 : Traverse the list
6 : Search for an element
7 : Swap two nodes
8 : Exit
```

```
5
Traversing the list : Virat Rohit
+-------------------------------------+
Please enter one of the following choices:
1 : Add at the front
2 : Get frontmost element
3 : Remove front element
4 : Check if list is empty
5 : Traverse the list
6 : Search for an element
7 : Swap two nodes
8 : Exit
6
Enter the element to search: Rohit
Rohit is present in the list.
+-------------------------------------+
Please enter one of the following choices:
1 : Add at the front
2 : Get frontmost element
3 : Remove front element
4 : Check if list is empty
5 : Traverse the list
6 : Search for an element
7 : Swap two nodes
8 : Exit
6
Enter the element to search: Sachin
Sachin is NOT present in the list.
+-------------------------------------+
Please enter one of the following choices:
1 : Add at the front
2 : Get frontmost element
3 : Remove front element
4 : Check if list is empty
5 : Traverse the list
6 : Search for an element
7 : Swap two nodes
8 : Exit
7
Enter the first element: Rohit
Enter the second element: Virat
+-------------------------------------+
Please enter one of the following choices:
1 : Add at the front
2 : Get frontmost element
3 : Remove front element
4 : Check if list is empty
5 : Traverse the list
6 : Search for an element
7 : Swap two nodes
```

```
5
Traversing the list : Rohit Virat
+-------------------------------------+
Please enter one of the following choices:
1 : Add at the front
2 : Get frontmost element
3 : Remove front element
4 : Check if list is empty
5 : Traverse the list
6 : Search for an element
7 : Swap two nodes
8 : Exit
2
Frontmost element is : Rohit
+-------------------------------------+
Please enter one of the following choices:
1 : Add at the front
2 : Get frontmost element
3 : Remove front element
4 : Check if list is empty
5 : Traverse the list
6 : Search for an element
7 : Swap two nodes
8 : Exit
4
List is not empty
+-------------------------------------+
Please enter one of the following choices:
1 : Add at the front
2 : Get frontmost element
3 : Remove front element
4 : Check if list is empty
5 : Traverse the list
6 : Search for an element
7 : Swap two nodes
8 : Exit
8
Exiting

...Program finished with exit code 1
Press ENTER to exit console.
```

DOUBLY LINKED LIST

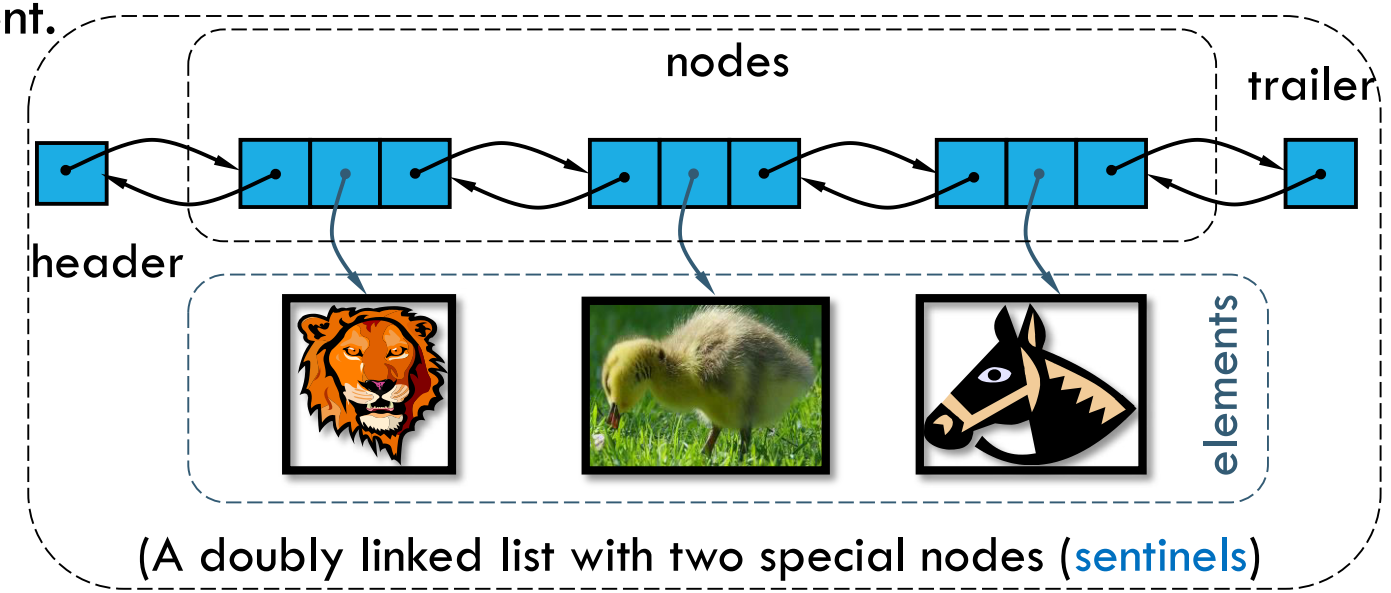
- Deleting the last node in a singly linked list is not efficient. Why? (rather any node other than first one or two)
- What is a doubly linked list?
- Insertions and deletions are more efficient.

```
typedef string Elem;
class DNode {
  private: Elem elem;
           DNode* prev;
           DNode* next;
           friend class DLinkedList;
};
```

(Implementation of DLL Node)

Applications:

- Used by browsers for what functionality?
- Used to implement MRU, and LRU caches?
- Undo/ Redo functionality in Word.
- Used to implement hash tables, stacks, binary tree etc.

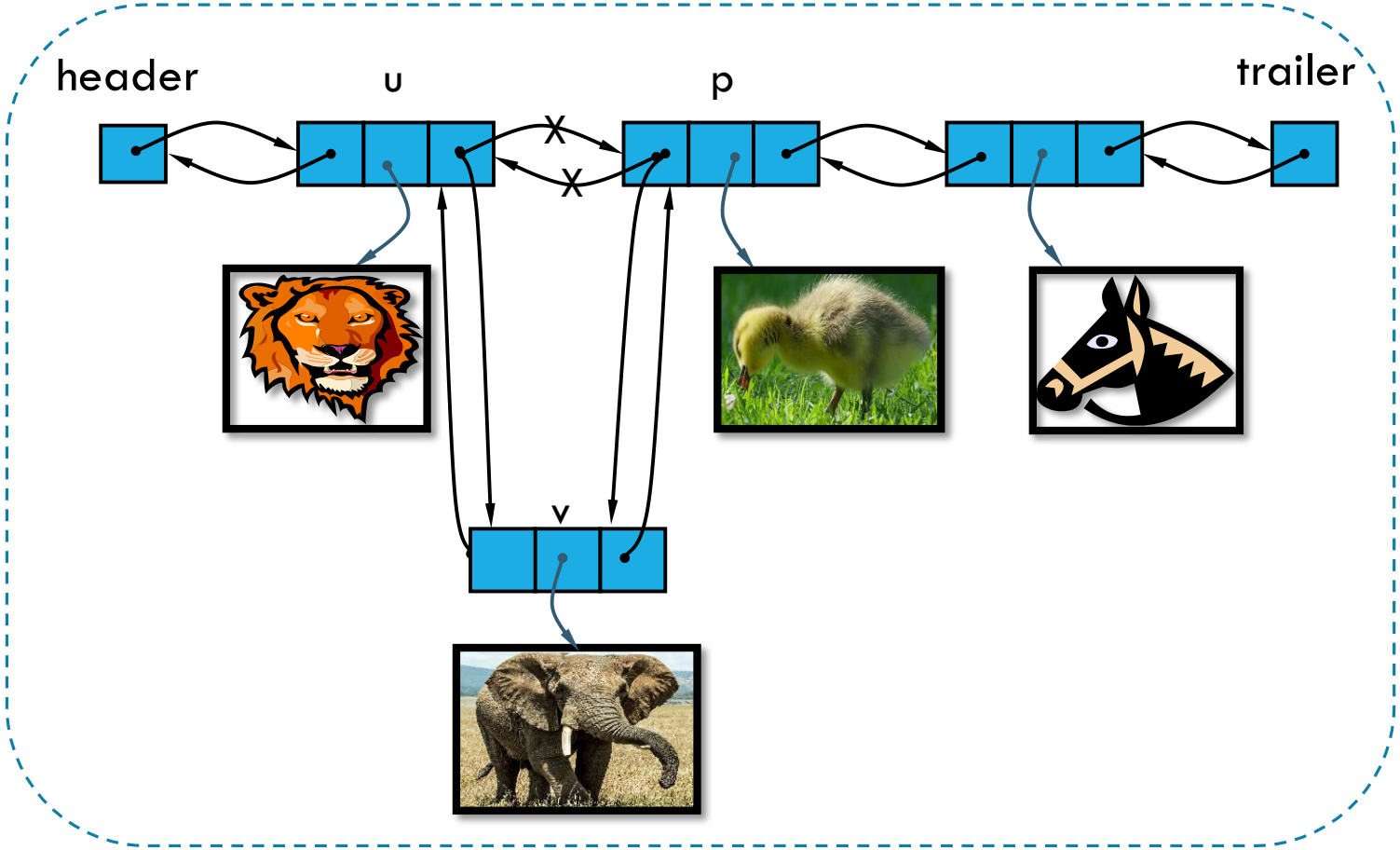


(A doubly linked list with two special nodes (sentinels)

INSERTING INTO DOUBLY-LINKED LIST

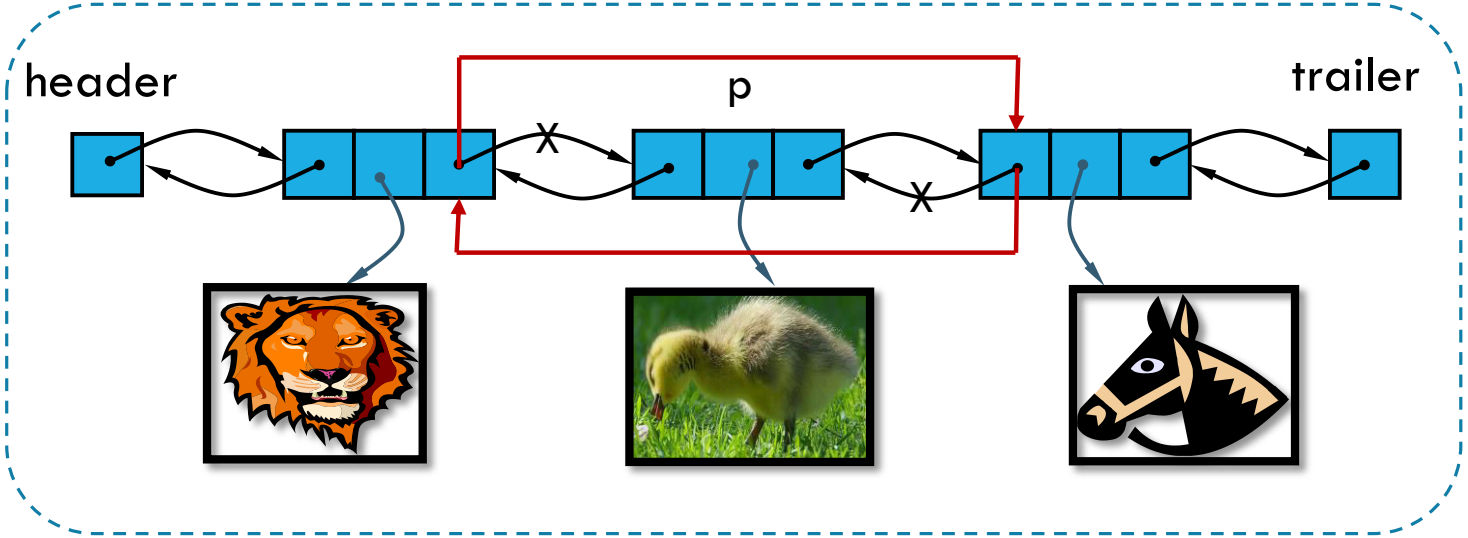
Algorithm insert(p, e): //insert e before p

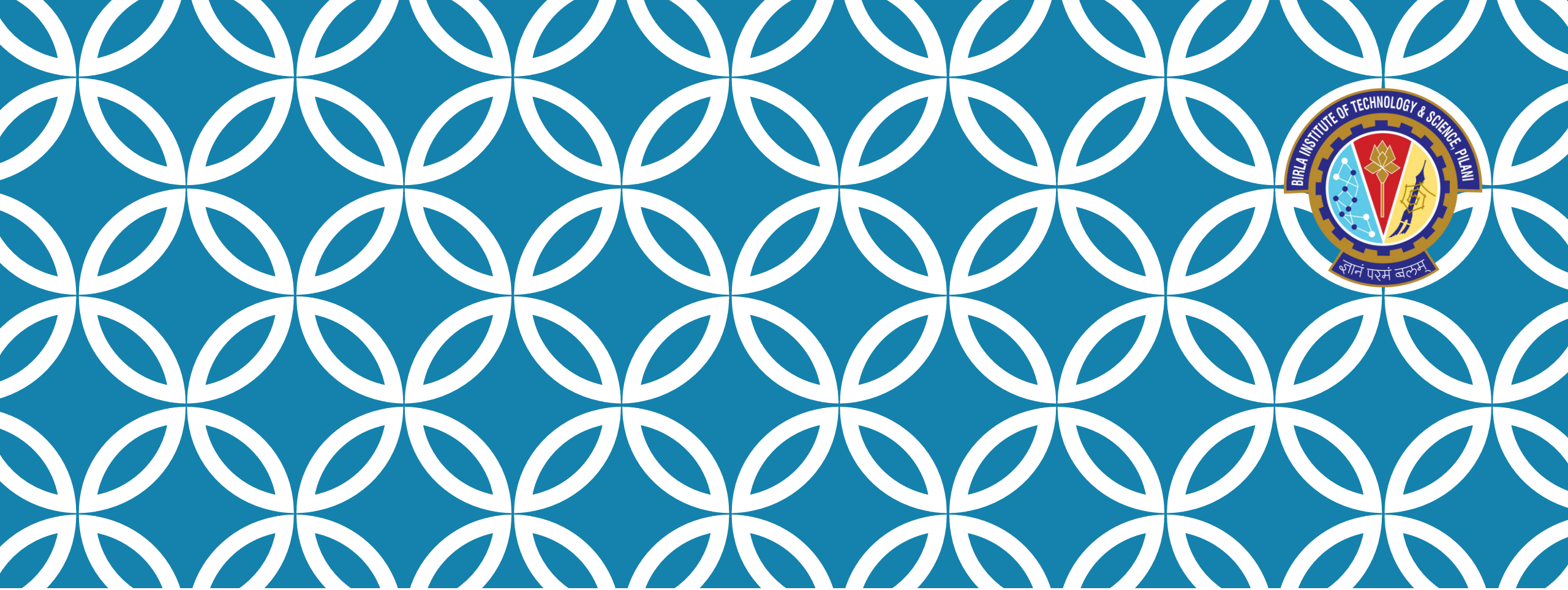
Let us write the pseudo code in parallel…



REMOVING A NODE IN DOUBLY-LINKED LIST

```
Algorithm remove (p: position ) {
    if (p->previous != nil) // not first
        p->previous->next = ???;
    if (p->next != nil) //not the last
        p->next->previous  = ???;
}
```





BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) DOUBLY LINKED LISTS CONTINUED…

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

REVERSING A DOUBLY-LINKED LIST

```
void listReverse(DLinkedList& L) {

    DLinkedList T; // temporary list
    while (!L.empty()) { // reverse L into T
            string s = L.front(); L.removeFront();
            T.addFront(s);
     }
    while (!T.empty()) { // copy T back to L
            string s = T.front();
            T.removeFront();
            L.addBack(s);
     }
}
```

```
struct node* reverse(struct node* head)
{
    struct node* ptr1 = head;
    struct node* ptr2 = ptr1->next;

    ptr1->next = NULL;
    ptr1->prev = ptr2;

    while(ptr2 != NULL)
    {
        ptr2->prev = ptr2->next;
        ptr2->next = ptr1;
        ptr1 = ptr2;
        ptr2 = ptr2->prev;
    }
    head = ptr1;
    return head;
}
```

Let us see on the board its' working!

MIDDLE NODE AND LOOP IN A LINKED LIST

```
DoublyLinkedNode<DT> *DoublyLinkedList<DT>::getMiddleNode()
{
    // Take two pointers
    DoublyLinkedNode<DT> *slowPtr, *fastPtr;

    // initially both pointers point to the head node
    slowPtr = fastPtr = head;

    while (fastPtr != NULL && fastPtr->next != NULL)
    {
        fastPtr = fastPtr->next->next; // jump twice
        slowPtr = slowPtr->next;       // jump once
    }

    // slow pointer points to middle node
    return slowPtr;
}
```

```
153 bool DLinkedList::isPalindrome()
154 {
155   DNode *begin = header;
156   DNode *end = trailer->prev;
157
158   while (begin != end)
159   {
160     if (begin->elem.compare(end->elem) != 0)
161       break;
162
163     begin = begin->next;
164     end = end->prev;
165   }
166   return 1;
167 }
```

Lab 4

CIRCULAR LINKED LISTS

- A circular linked list is a singly-linked list except element of the list pointing to the first. Without s we can go back to the first.
- What is the need of cursor node?

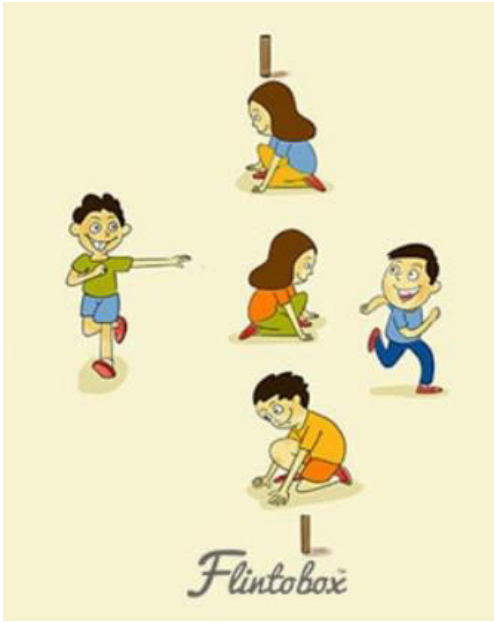
```
class CircleList;
typedef string Elem;
class CNode {
private: Elem elem;
         CNode* next;
         friend class CircleList;
};
class CircleList {
public:   CircleList();
          ~CircleList();
           bool empty() const;
           const Elem& front() const;
           const Elem& back() const;
           void advance();
           void add(const Elem& e);
           void remove();
private:   CNode* cursor;
};
```

```
CircleList::CircleList() : cursor(NULL)
CircleList::~CircleList() { while (!emp
bool CircleList::empty() const {retur
const Elem& CircleList::back() const
                                    ret
const Elem& CircleList::front() const
                                return cu
void CircleList::advance() { cursor =
void CircleList::add(const Elem& e)
   CNode* v = new CNode;
   v->elem = e;
   if (cursor == NULL) {
      v->next = v;  cursor = v; }
   else {
      v->next = cursor->next; curs
   }

}
```

Result

compiled and executed in 120.499 sec(s)

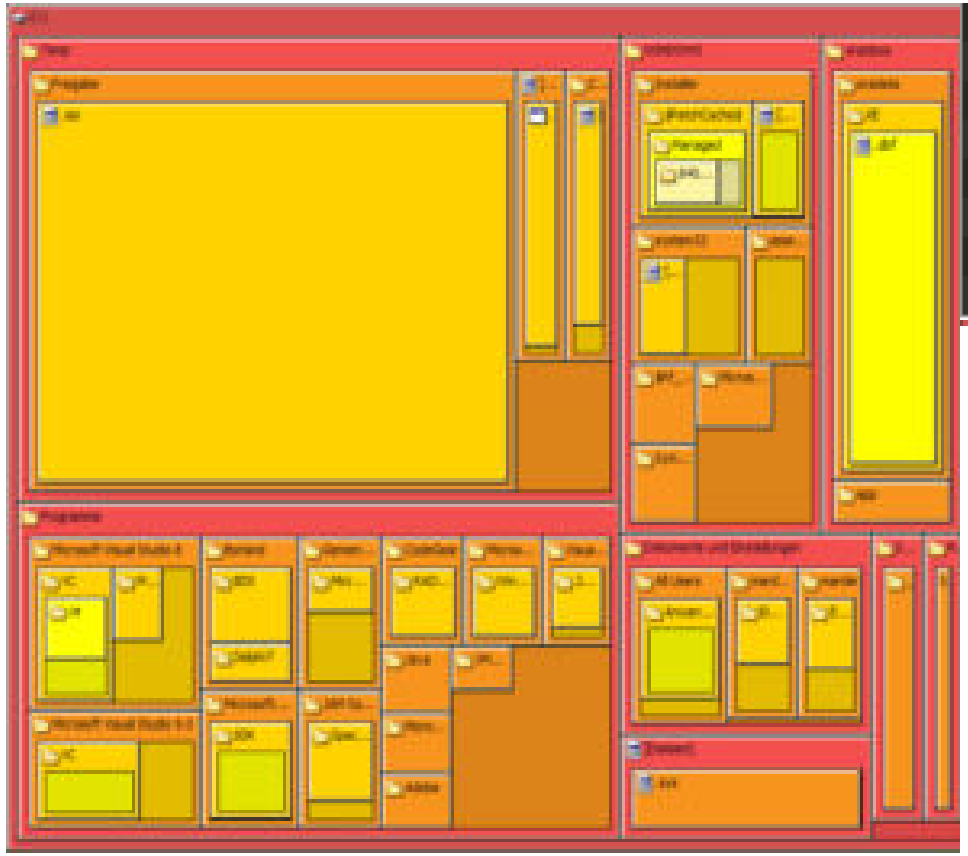
```
Please enter one of the following choices:
1 : Add
2 : Get front element
3 : Get back element
4 : Advance cursor
5 : Remove element pointed by cursor
6 : Check if list is empty
7 : Exit
1
s1
Adding the following element : s1
1
s2
Adding the following element : s2
1
s3
Adding the following element : s3
3
Back element is : s1
4
Advancing the cursor
2
Front element is : s2
5
Removing element pointed by the cursor
6
List is not empty
```



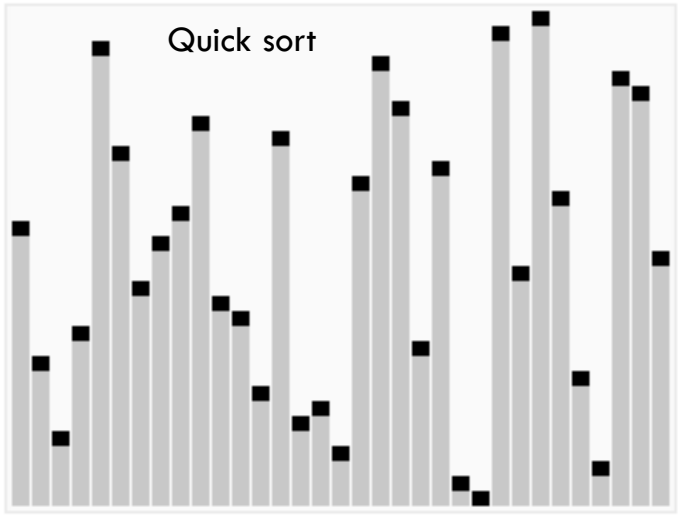

```
// remove the node after the cursor
void CircleList::remove() {
  CNode* old = cursor->next;
  if (old == cursor)
     cursor = NULL;
  else
     cursor->next = old->next;
     delete old;
}
```

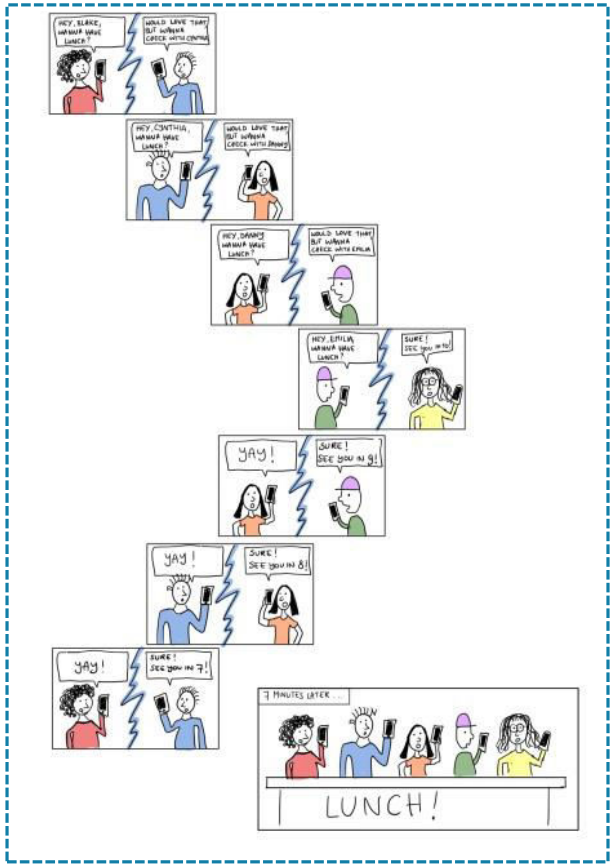
RECURSION: ELEGANT WAY FOR REPETITIVE TASKS

Recursion: When a function or a method calls itself. A set of problems can be solved easily using recursion (a powerful programming tool).



```
func search(currentDir):
    if targetFile in currentDir:
        return currentDir
    for childDir in currentDir:
        result = search(childDir)
        if result != null:
            return result
    return null
```





https://www.techiedelight.com/recursion-practice-problems-with-solutions/ (for practice)

https://abetterscientist.wordpress.com/

RECURSION: ELEGANT WAY FOR REPETITIVE TASKS

Recursion: When a function or a method calls itself. A set of problems can be solved easily using recursion (a powerful programming tool).

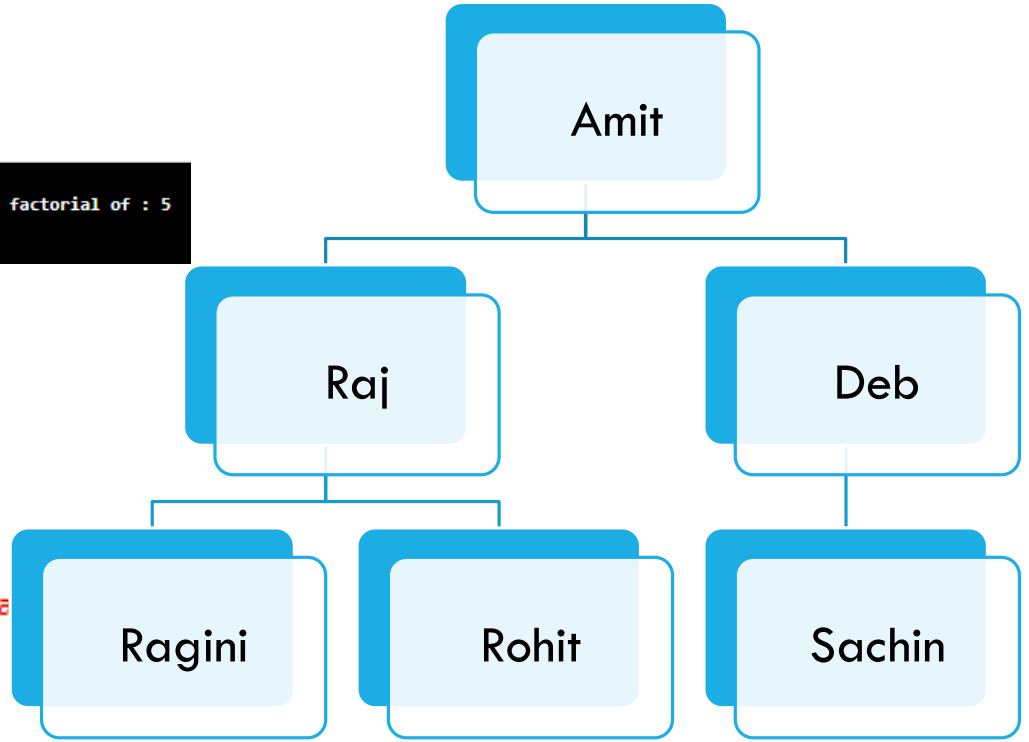
Recursive definition:

```cpp
#include <iostream>

using namespace std;

long long int factorial(int num){
    if(num == 0){
        return 1;
    }
    return num * factorial(num-1);
}
int main() {
    int num;
    cout<<"Please enter the number you want the factoria
    cin>>num;
    cout<<endl;
    cout<<num<<"! = "<<factorial(num)<<endl;
    return 0;
}
```

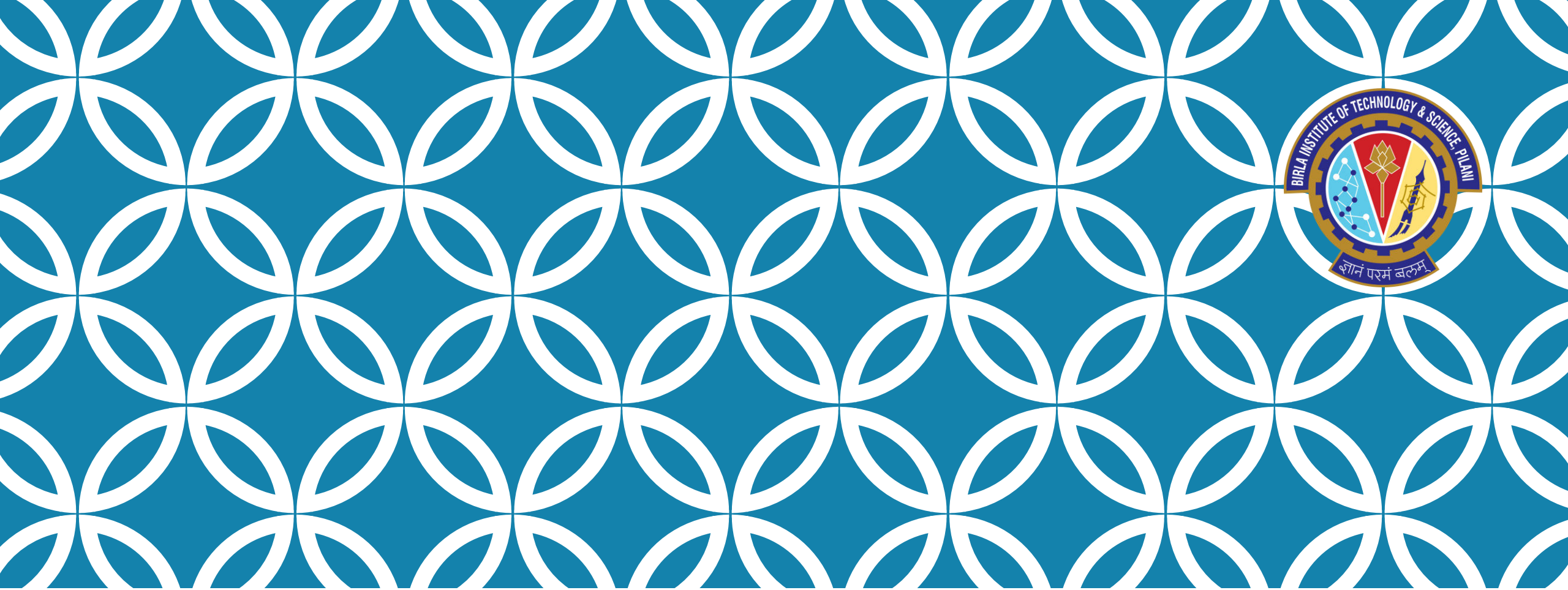
Result

compiled and executed in 6.915 sec(s)

```
Please enter the number you want the factorial of : 5

5! = 120
```



(Business Organization Chart)



BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) RECURSION CONTINUED…

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

LINEAR RECURSION

- A linear recursive function is a function that makes at most one recursive call each time it is invoked (as opposed to one that would call itself multiple times during its execution).

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

int linearSum(int A[],int n){
    if(n == 1) return A[0];
    return linearSum(A,n-1) + A[n-1];
}

int main(){
    int len;
    cout<<"Enter length of input array : ";
    cin>>len;
    cout<<endl;
    int A[len];
    int aux;
    cout<<"Enter the array : ";
    for(int i=0;i<len;i++){
        cin>>aux;
        A[i] = aux;
    }
    cout<<"\nSum of the array is : "<<linearSum(A,len)<<endl;
    return 0;
}
```

Result

compiled and executed in 13.95 sec(s)

```
Enter length of input array : 5

Enter the array : 4 3 6 2 5

Sum of the array is : 20
```

TAIL RECURSION: REVERSING AN ARRAY

```cpp
#include <iostream>
using namespace std;
void swap(int *a,int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}
void reverseArray(int A[],int start,int end){
    if(start < end){
        swap(&A[start],&A[end]);
        reverseArray(A,start+1,end-1);
    }
}
int main(){
    int len;
    cout<<"Enter length of the input array : ";
    cin>>len;
    cout<<endl;
    int A[len];
    cout<<"Enter the array : ";
    int aux;
    for(int i=0;i<len;i++){
        cin>>aux;
        A[i] = aux;
    }
    reverseArray(A,0,len-1);
    cout<<"\nReversed array : ";
    for(int i=0;i<len;i++) cout<<A[i]<<" ";
    cout<<endl;
    return 0;
}
```

- Tail recursion occurs when a linearly recursive method makes its recursive call as its last step.
- Such methods can be easily converted to non-recursive methods (which saves on some resources).
- Is Linear sum (previous program, tail recursive?)

Algorithm IterativeReverseArray(A, i, j):
Input: An array A and nonnegative integer indices i and j
Output: The reversal of the elements in A starting at index i and ending at j

while $i < j$ **do**
  Swap $A[i]$ and $A[j]$
  $i \leftarrow i + 1$
  $j \leftarrow j - 1$
return

Result
compiled and executed in 12.089 sec(s)

```
Enter length of the input array : 4

Enter the array : 4 5 7 8

Reversed array : 8 7 5 4
```

BINARY RECURSION

- What is binary recursion?

Problem: add all the numbers in an integer array A:

Algorithm BinarySum(*A*, *i*, *n*):
  Input: An array *A* and integers *i* and *n*
  Output: The sum of the *n* integers in *A* starting at index *i*
  if $n = 1$ **then**
    return *A*[*i*];
  return BinarySum(*A*, *i*, $n/2$) + BinarySum(*A*, $i + n/2$, $n/2$)

Let us see the recursion trace…

Used heavily in merging and tree traversals…

COMPUTING FIBONACCI NUMBERS

```cpp
#include<bits/stdc++.h>
using namespace std;

int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}

int main ()
{
    int n = 9;
    cout << fib(n);
    getchar();
    return 0;
}
```

Is binary recursion better here?

n_k denote the number of calls performed in the execution of fib(k)

$$
\begin{aligned}
n_0 &= 1 \\
n_1 &= 1 \\
n_2 &= n_1 + n_0 + 1 = 1 + 1 + 1 = 3 \\
n_3 &= n_2 + n_1 + 1 = 3 + 1 + 1 = 5 \\
n_4 &= n_3 + n_2 + 1 = 5 + 3 + 1 = 9 \\
n_5 &= n_4 + n_3 + 1 = 9 + 5 + 1 = 15 \\
n_6 &= n_5 + n_4 + 1 = 15 + 9 + 1 = 25 \\
n_7 &= n_6 + n_5 + 1 = 25 + 15 + 1 = 41 \\
n_8 &= n_7 + n_6 + 1 = 41 + 25 + 1 = 67
\end{aligned}
$$

Let us draw the tree too…

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

// A tail recursive function to
// calculate n th fibnacci number
int fib(int n, int a = 0, int b = 1)
{
    if (n == 0)
        return a;
    if (n == 1)
        return b;
    return fib(n - 1, b, a + b);
}

// Driver Code
int main()
{
    int n = 9;
    cout << "fib(" << n << ") = "
         << fib(n) << endl;
    return 0;
}
```

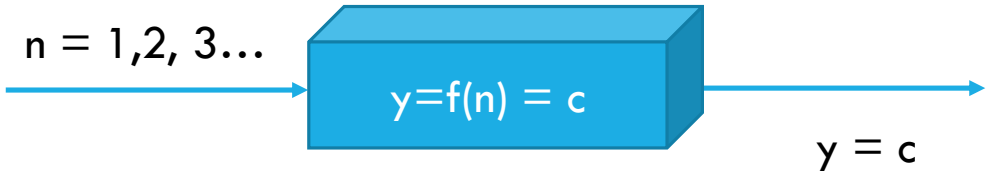
What is the type of this rec.?

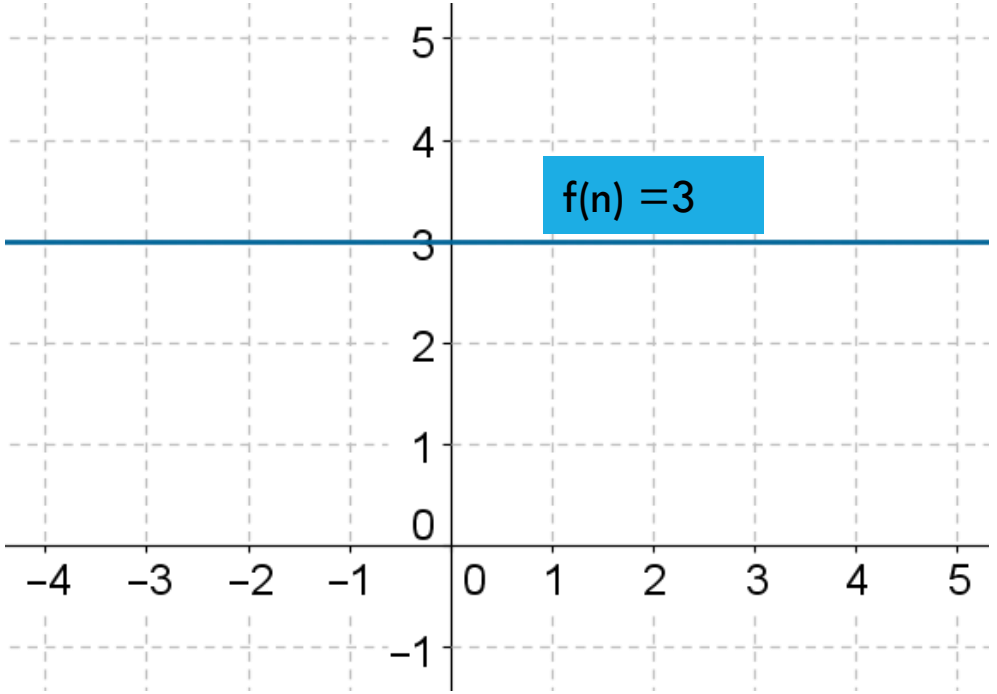
WHAT IS COMPLEXITY & HOW IMPORTANT IS IT?

You want to look for a word in a dictionary that has every word sorted alphabetically. How many algorithms are there and which one will you prefer?

FUNCTIONS FOR ALGORITHM ANALYSIS

- The Constant function: Output is same i.e. independent of input. It characterizes the number of steps needed to do a basic operation on a computer like, what types of operations? Constant algorithm does not depend on the input size.

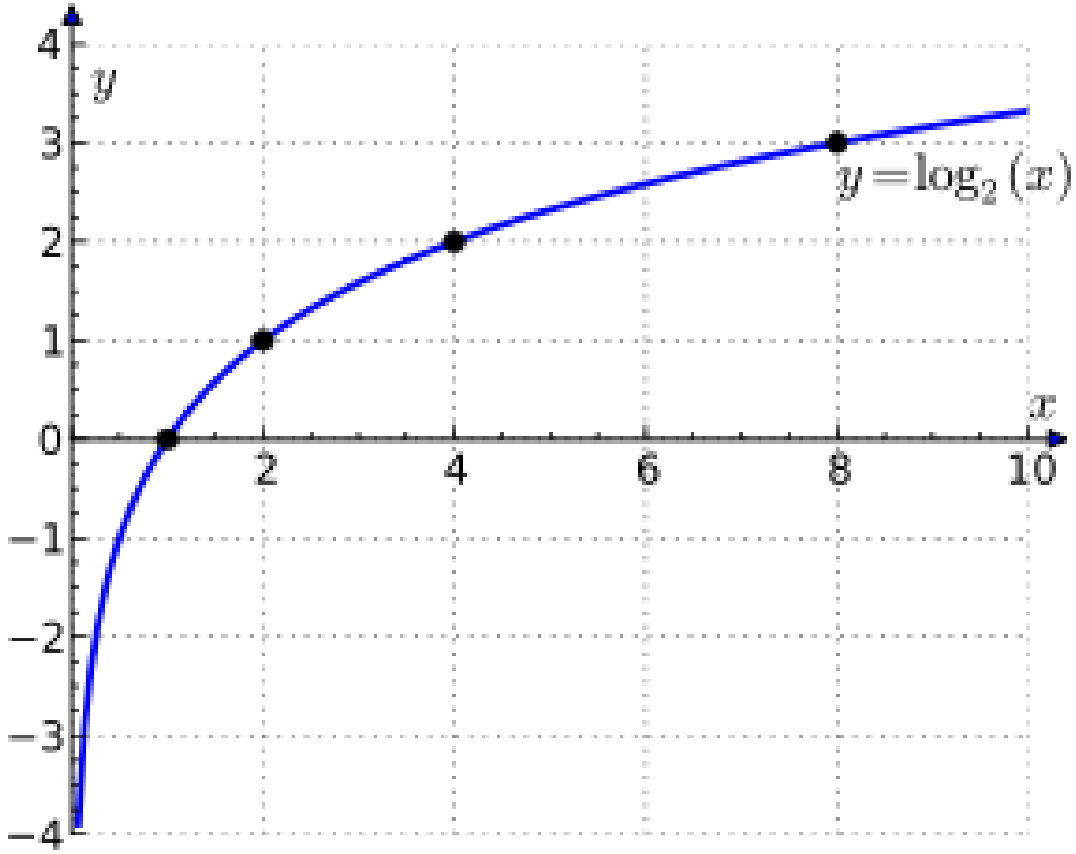




What about $f(w) = w^2$?

LOGARITHMIC FUNCTION

- Heavily used in Analysis of algorithms.
- Why is it used with base 2 most?



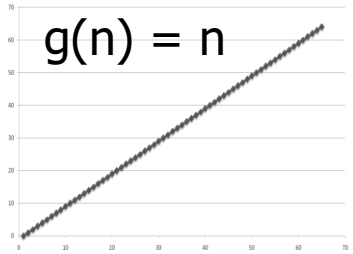
(Rules)

1. log(nm) = log n + log m
2. log (n/m) = log n – log m
3. Log (n^r) = r log n
4. Log_a n = $\log_b$ n/$\log_b$ a

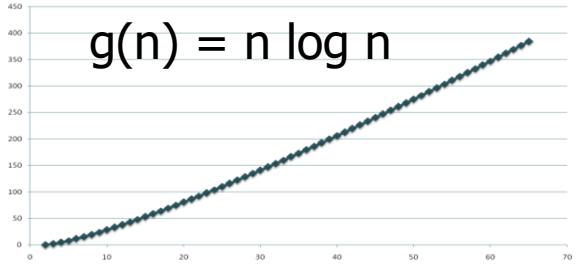
(Img. source: wiki)

LINEAR AND N-LOG-N FUNCTIONS

Linear Function: Given an input value ‘n’, the linear function g (n) assigns the value ‘n’ itself.

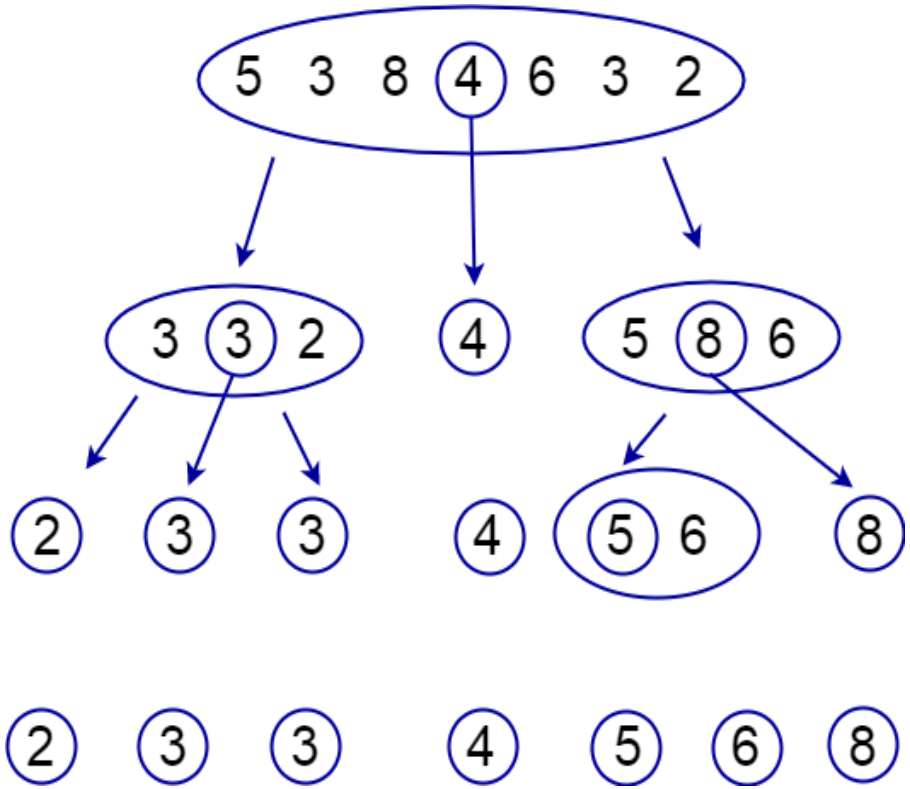


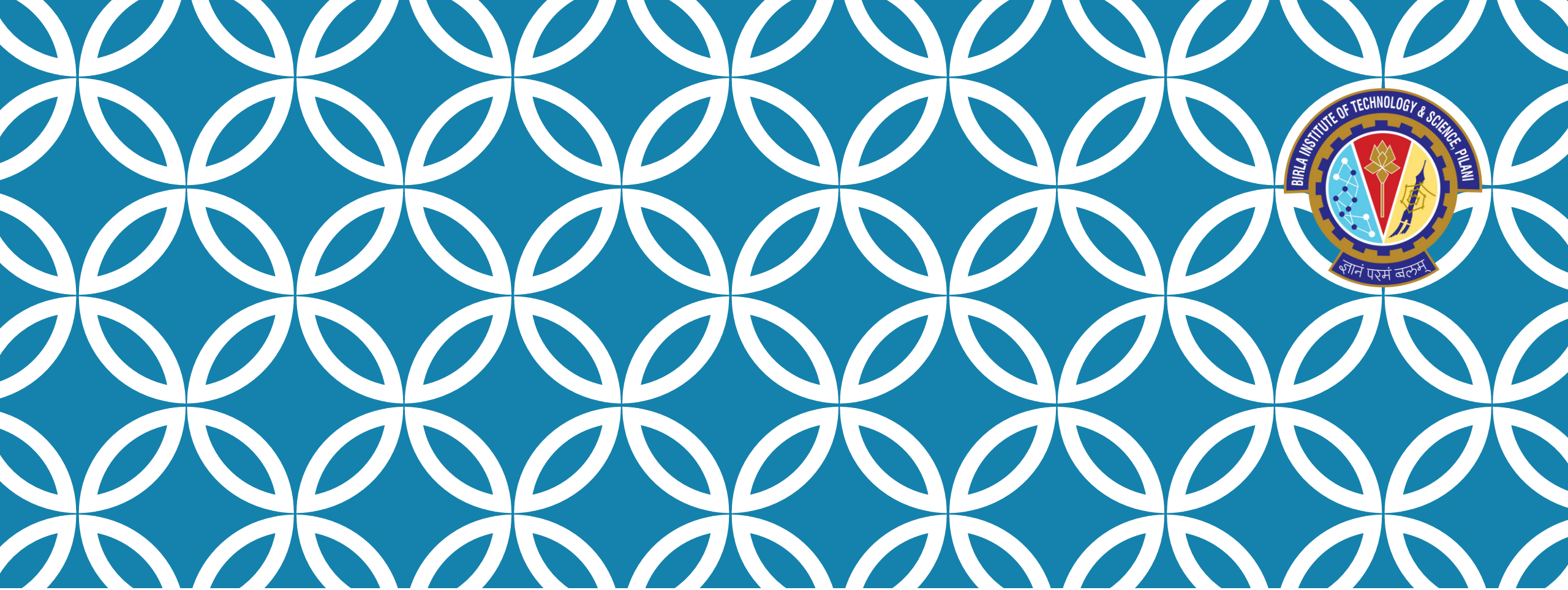
nlogn function: A function that assigns to an input ‘n’ the value of ‘n’ times logarithm base 2 of ‘n’.




```
void quicksort (list[ ], left, right) {
   int pivot = partition (list, left, right);
   quicksort (list, left, pivot-1);
   quicksort (list, pivot+1, right);
}
```

complexity?





BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) ALGORITHM COMPLEXITY

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

RECAP: CONSTANT, LINEAR, LOGN, NLOGN

```
function isEvenOrOdd(n) {
    if (n%2 == 0)
        return even;
    else
        return odd;
}
```

```
list<int> numbers {1, 2, 3, 4};
for(int number : numbers)
{
    cout << number <<", ";
}
```

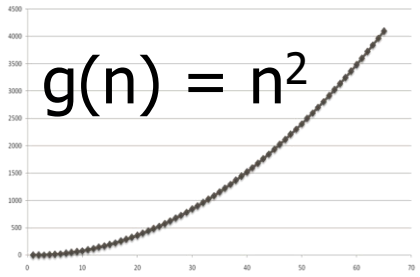
(printing out all the elements)

```
int partition(int arr[], int low, int high) { int pivot=arr[high];
   int i = (low - 1);
   for (int j = low; j <= high - 1; j++) {
     if (arr[j] < pivot) { i++; swap(&arr[i], &arr[j]); } }
   swap(&arr[i + 1], &arr[high]); return (i + 1);
}
```

```
int binarySearch(int array[], int x, int low,
int high)
{
  while (low <= high)
  {
      int mid = low + (high - low) / 2;
      if (array[mid] == x) return mid;
      if (array[mid] < x) low = mid +1;
      else
         high = mid - 1;
   }
   return -1;
}
```

QUADRATIC FUNCTIONS

Quadratic: Given an input value 'n', the function 'g' assigns the product of 'n' with itself. Also, called 'n squared'.

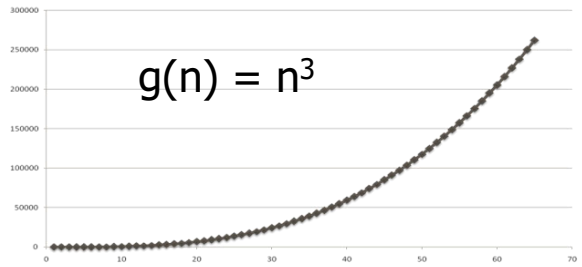


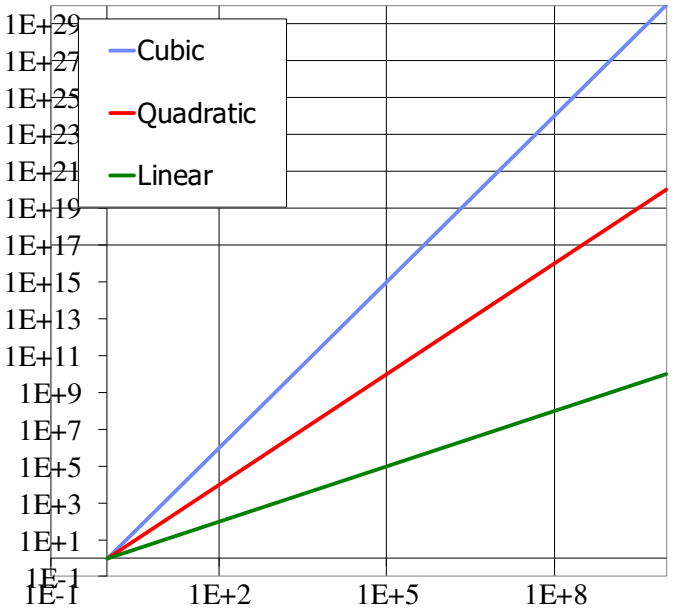
Used in analysing algorithms where nested loops are used.

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        // Print if their modulo equals to k
        if (i != j && arr[i] % arr[j] == k) {
            cout << "(" << arr[i] << ", "
                 << arr[j] << ")"
                 << " ";
            isPairFound = true;
        }
    }
}
```

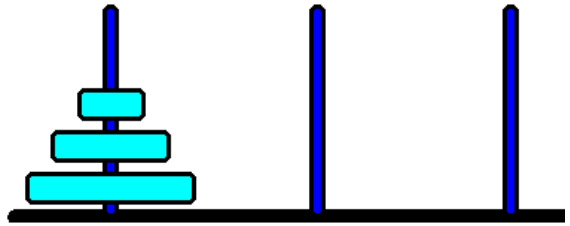
CUBIC, AND EXPONENTIAL FUNCTIONS

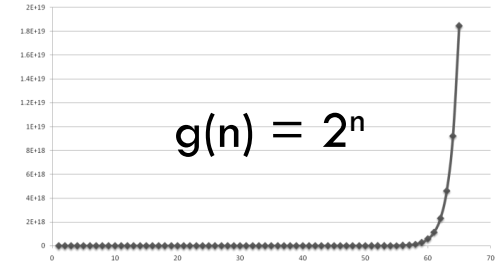
```
1: procedure NAIVE-MATRIX-MULTIPLY(A, B)
2:     n = A.rows
3:     let C be a new n × n matrix
4:     for i = 1 to n do
5:         for j = 1 to n do
6:             c_ij = 0
7:             for k = 1 to n do
8:                 c_ij = c_ij + a_ik · b_kj
9:             end for
10:        end for
11:    end for
12:    return C
13: end procedure
```





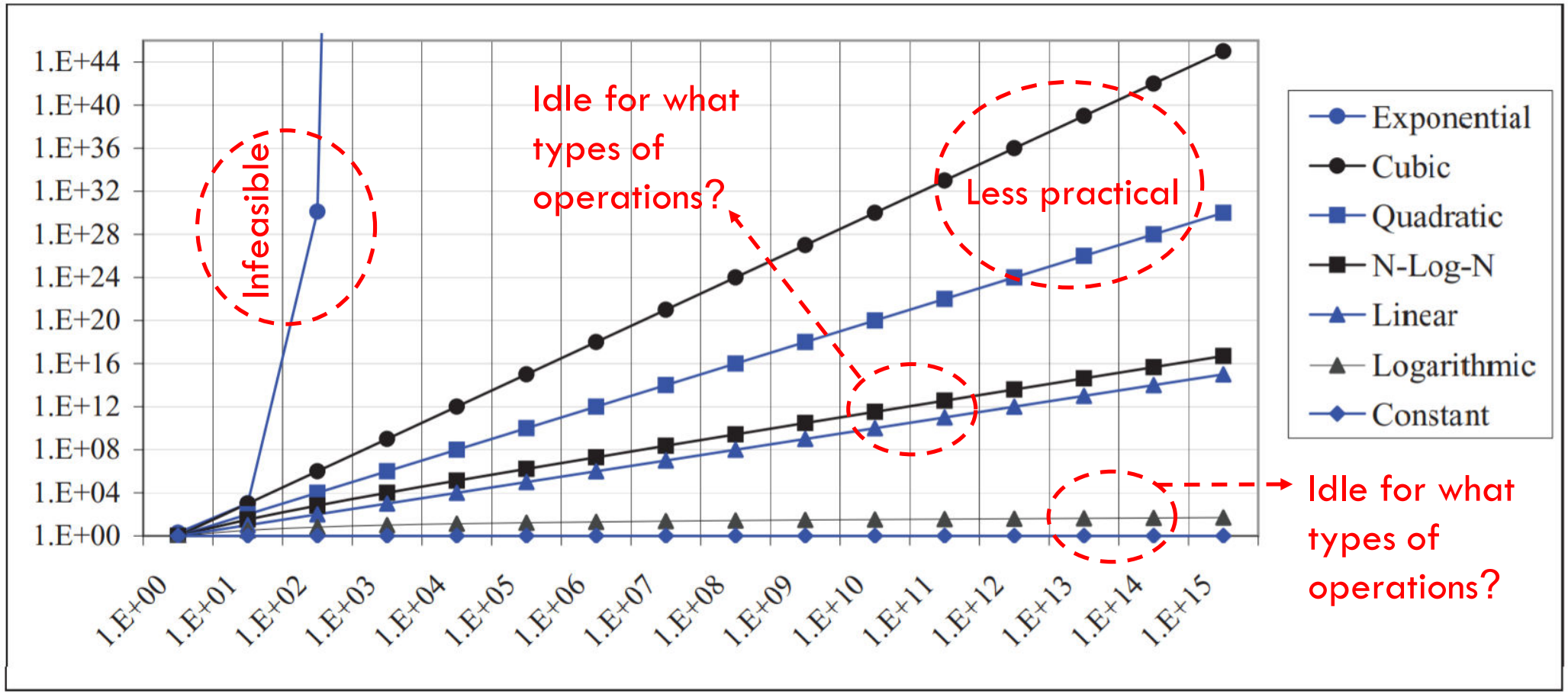
(log-log graph)






```
int Fibonacci (int number) {
   if (number <= 1)
         return number;
   return Fibonacci(number - 2) +
         Fibonacci(number - 1);
}
```

GROWTH RATES OF SEVEN FUNCTIONS



[log-log plot with growth rates (running times)as slopes]

ANALYSIS OF ALGORITHMS: EXPERIMENTAL STUDIES

```
// example1.cpp

#include <cstdlib>
#include <stdio.h>
using namespace std;
//declaration of functions
int func1();
int func2();

int func1(void) {
    int i=0,g=0;
    while(i++<100000) {
        g+=i;
    }
    return g;
}

int func2(void) {
    int i=0,g=0;
    while(i++<400000) {
        g+=i;
    }
    return g;
}

int main(int argc, char** argv) {
    int iterations = 10000;
    printf(`Number of iterations = %d\n`, iterations);
    while(iterations--) {
        func1();
        func2();
    }
}
```

gprof

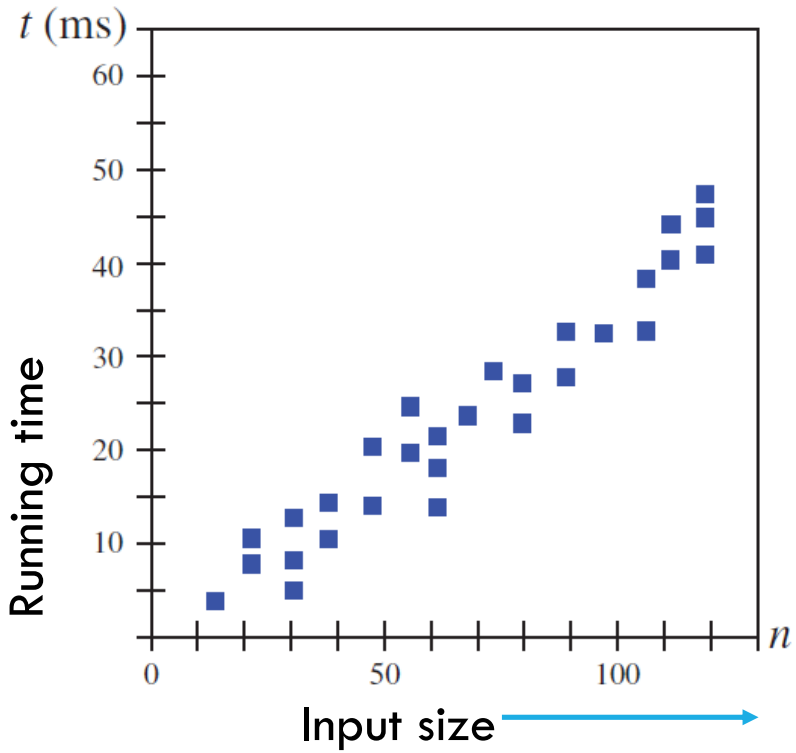
NEXT LAB

```
Flat profile:
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 80.80      9.59     9.59    10000   959.15   959.15  func2()
 20.33     12.00     2.41    10000   241.31   241.31  func1()
```

and the call graph:

```
Call graph (explanation follows)
granularity: each sample hit covers 2 byte(s) for 0.08% of 12.00 seconds
index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00   12.00                 main [1]
                9.59    0.00   10000/10000       func2() [2]
                2.41    0.00   10000/10000       func1() [3]
-----------------------------------------------
                9.59    0.00   10000/10000       main [1]
[2]     79.9    9.59    0.00   10000         func2() [2]
-----------------------------------------------
                2.41    0.00   10000/10000       main [1]
[3]     20.1    2.41    0.00   10000         func1() [3]
-----------------------------------------------
```

http://web.cecs.pdx.edu/~karavan/perf/book_gprof.html



Limitations:

- It is necessary to implement the complete algorithm, which may be difficult.
- Results may not be indicative of the running time on other inputs not included in the experiment.
- In order to compare two algorithms, the same hardware and software environments must be used

EXAMPLE RUN TIME...

```cpp
#include <chrono>
class Timer
{
private:
    std::chrono::time_point<std::chrono::high_resolution_clock> startTimePoint;
    std::chrono::time_point<std::chrono::high_resolution_clock> endTimePoint;
    double getTimeDifference();

public:
    Timer();
    void start();
    void stop();
    double getDurationInSeconds();
    double getDurationInMilliSeconds();
    double getDurationInMicroSeconds();
};
Timer::Timer() {}
void Timer::start()
{
    startTimePoint = std::chrono::high_resolution_clock::now();
}
void Timer::stop()
{
    endTimePoint = std::chrono::high_resolution_clock::now();
}
double Timer::getTimeDifference()
{
    auto start = std::chrono::time_point_cast<std::chrono::microseconds>(
    auto end = std::chrono::time_point_cast<std::chrono::microseconds>(en
    return end - start;
}
double Timer::getDurationInSeconds()
{
    return getDurationInMilliSeconds() * 0.001; // in seconds
}
double Timer::getDurationInMilliSeconds()
{
    return getTimeDifference() * 0.001; // in milli seconds
}
double Timer::getDurationInMicroSeconds()
{
    return getTimeDifference(); // in micro-seconds
}
```

NEXT LAB

```
Linear Search took: 37.647 ms.
Binary Search took: 0.002 ms.
Enter the size of the array: 7
Enter a sorted list of 7 elements:
10 20 30 40 50 60 70
Enter the target item to search for: 40
40 FOUND at index 3
Binary Search took: 0 ms.
...Program finished with exit code 0
Press ENTER to exit console.
```

recursive

Inside main()

```cpp
    Timer timer; // initialize timer class object.

    timer.start(); // start timer.

    linearSearch(arr, n, n); // call to linear search

    timer.stop(); // stop timer.

    // function to get time in milli seconds
    double milliSecs = timer.getDurationInMilliSeconds();

    cout << "Linear Search took: " << milliSecs << " ms." << endl;

    timer.start(); // start timer.

    binarySearch(arr, n, n); // call to binary search

    timer.stop(); // stop timer.

    // function to get time in milli seconds
    milliSecs = timer.getDurationInMilliSeconds();

    cout << "Binary Search took: " << milliSecs << " ms." << endl;
```

CONTINUED...

```
221 // finds and returns the n'th node from the end of the list.
222 template <typename DT>
223 SinglyLinkedNode<DT> *SinglyLinkedList<DT>::nthNodeFromEnd(int n)
224 {
225     // code here
226     counter = 0;
227     tmp = NULL;
228     nthNodeFromEndRecursive(head, n);
229     return tmp; // return the n'th node from the end
230 }
```

```
232 // recursive solution to find out the n'th node from the end.
233 template <typename DT>
234 void SinglyLinkedList<DT>::nthNodeFromEndRecursive(SinglyLinkedNode<DT>*head,int n)
235 {
236     if (head == NULL)
237         return;
238
239     nthNodeFromEndRecursive(head->next, n);
240     counter++;
241     if (counter == n)
242     {
243         tmp = head;
244     }
245 }
```

```
309         case '6':
310             cout << "Enter N: ";
311             cin >> a;
312             timer.start();
313             node = list.nthNodeFromEnd(a);
314             timer.stop();
315             if (node == NULL)
316                 cout << "Such a node does not exist." << endl;
317             else
318                 cout << "N'th node from the end: " << node->dataItem << endl;
319             cout << "Time spent: " << timer.getDurationInMilliSeconds() << " ms." << endl;
320             break;
```

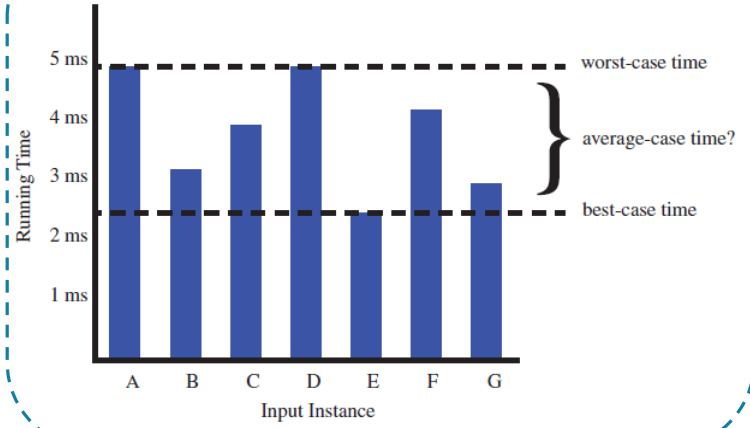
```
Please enter one of the following choices:
1 : Insert at end
2 : Delete from end
3 : Print Forward
4 : Print Backward
5 : Reverse List
6 : Get N'th node from the end
7 : Exit
3
10 20 30 40
Time spent: 0.019 ms.
+-------------------------------------+
Please enter one of the following choices:
1 : Insert at end
2 : Delete from end
3 : Print Forward
4 : Print Backward
5 : Reverse List
6 : Get N'th node from the end
7 : Exit
6
Enter N: 2
N'th node from the end: 30
Time spent: 0.001 ms.
+-------------------------------------+
Please enter one of the following choices:
1 : Insert at end
2 : Delete from end
3 : Print Forward
4 : Print Backward
5 : Reverse List
6 : Get N'th node from the end
7 : Exit
```

NEXT LAB

THEORETICAL ANALYSIS

- Uses a high-level description of the algorithm instead of an implementation.
- Characterizes running time as a function of the input size, *n*.
- Takes into account all possible inputs
- Allows us to evaluate the speed of an algorithm independent of the hardware/software environment

Natural measure of "goodness"



Pseudo

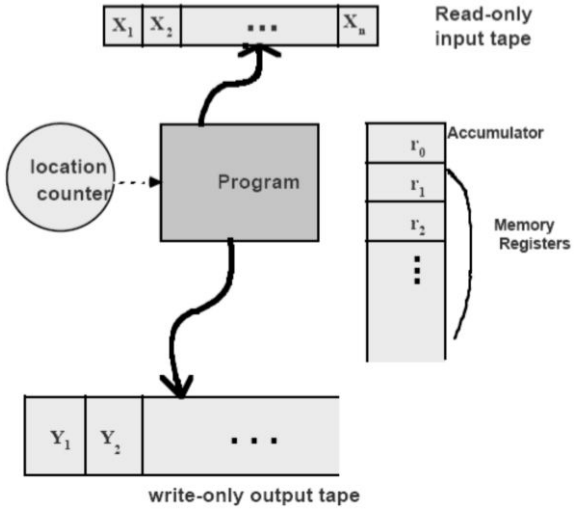
Algorithm *arrayMax*(*A*, *n*)

Input: array *A* of *n* integers

Output: maximum element of *A*

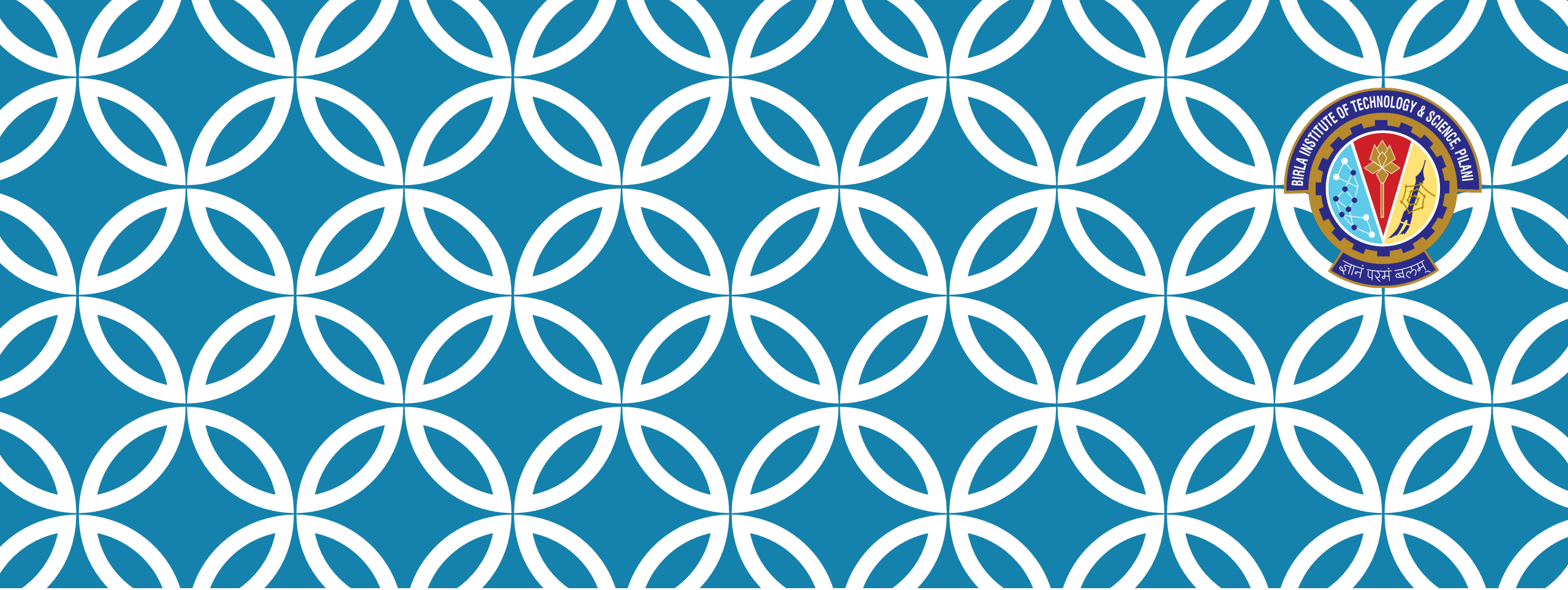
```
max ← A[0]
for i ← 1 to n − 1 do
        if A[i] > max then
                max ← A[i]
return max
```

(The RAM Model)



Algorithm *arrayMax*(*A*, *n*)	# operations
currentMax ← *A*[0]	2
for *i* ← 1 **to** *n* − 1 **do**	2*n*
if *A*[*i*] > *currentMax* **then**	2(*n* − 1)
currentMax ← *A*[*i*]	2(*n* − 1)
{ increment counter *i* }	2(*n* − 1)
return *currentMax*	1
	8*n* − 3

The algorithm arrayMax executes about 8*n*-3 primitive operations in the worst case.



BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) ALGORITHM COMPLEXITY CONTINUED…

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

ASYMPTOTIC NOTATION

- In maths, what is an **Asymptote**?

- In data structures and algorithms, what is Big O notation?

BIG-OH RULES

$13n^4 - 8n^2 + \log_2 n$?

```
function findBiggestNumber(array) {
    let biggest = array[0];

    for (let i = 0; i < array.length; i++) {
        if (array[i] > biggest) {
            biggest = array[i];
        }
    }

    return biggest;
}
```

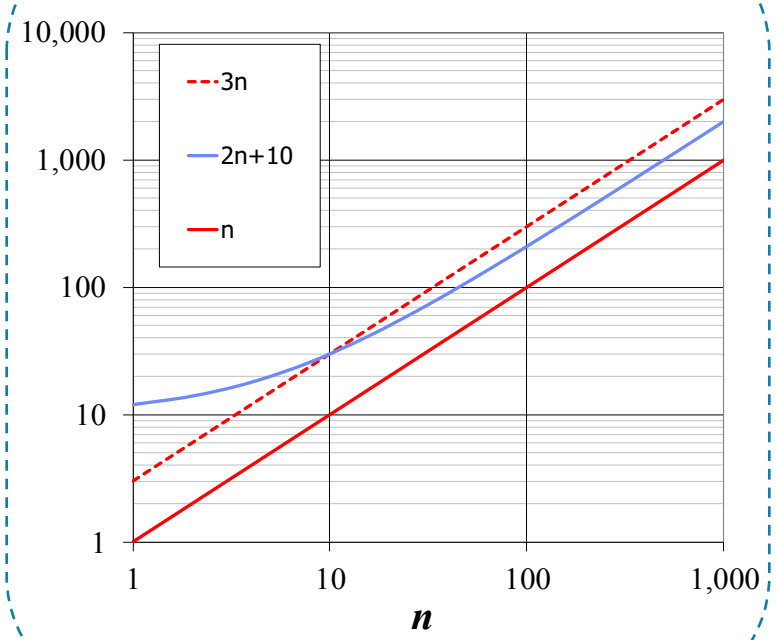
```
function findBiggestNumber(array) {
    let biggest = array[0];

    for (let i = 0; i < array.length; i++) {
        if (array[i] > biggest) {
            biggest = array[i];
        }
    }

    return biggest;
}
```

```
function containsDuplicates(array) {
    for (let i = 0; i < array.length; i++) {
        for (let r = 0; r < array.length; r++) {
            if (i === r) {
                continue;
            }
            if (array[i] === array[r]) {
                return true;
            }
        }
    }

    return false;
}
```

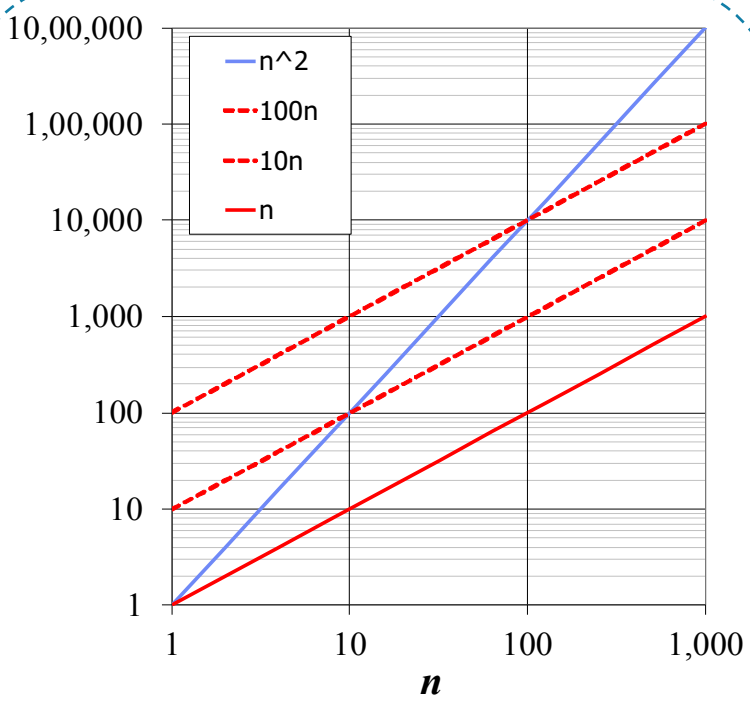
Source: https://medium.com/

BIG-OH EXAMPLES

Example: $2n + 10$ is $O(n)$



What about n^2 ?



MORE BIG-OH EXAMPLES

$f(n) = 7n-2$

$7n-2$ is $O(n)$

need $c > 0$ and $n_0 \geq 1$ such that $7n-2 \leq c.n$ for $n \geq n_0$

this is true for $c = 7$ and $n_0 = 1$

$f(n) = 3n^3 + 20n^2 + 5$

$3n^3 + 20n^2 + 5$ is $O(n^3)$

need $c > 0$ and $n_0 \geq 1$ such that $3n^3 + 20n^2 + 5 \leq c.n^3$ for $n \geq n_0$

this is true for $c = 4$ and $n_0 = 21$

$f(n) = 3 \log n + 5$

$3 \log n + 5$ is $O(\log n)$

need $c > 0$ and $n_0 \geq 1$ such that $3 \log n + 5 \leq c.\log n$ for $n \geq n_0$

this is true for $c = 8$ and $n_0 = 2$

$f(n) = 2^{n+2}$

$f(n) = 2n^3 + 10n$ ➡ $O(n^3)$

$(2n^3 + 10n) \leq c.n^3$ 

For $c = 12$, and $n_0 = 2$

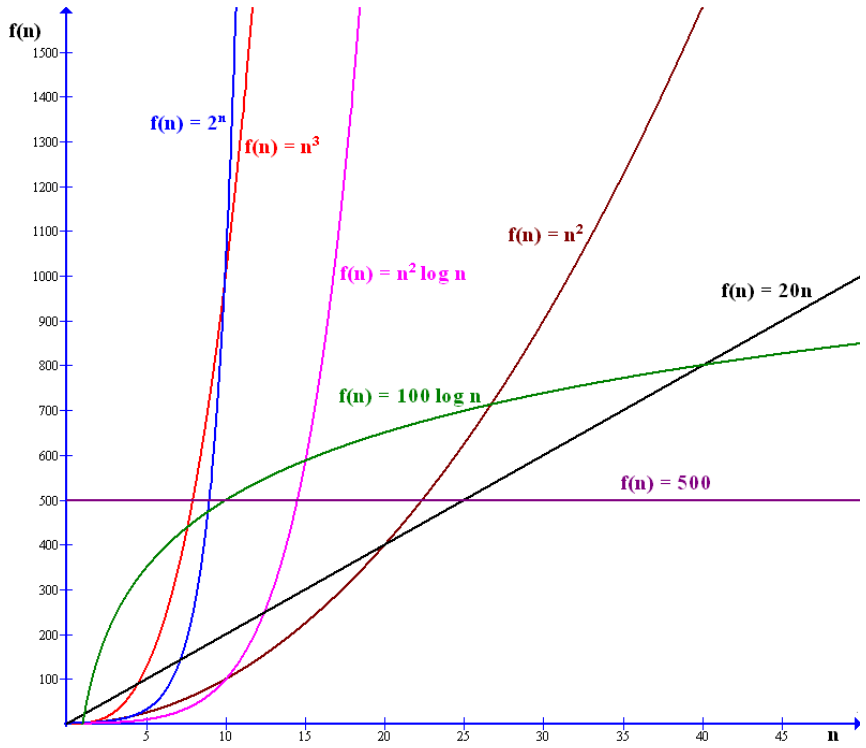
LET US TRY FINDING BIG O...

Assume that you lost your wedding ring on the beach, and have no memory of when it came off. Thus, you decide to do a brute force grid search with your metal detector, where you divide the beach into strips, and walk down every strip, scanning the whole beach, until you find it. For simplicity, assume the beach is a square of side length 'l' meters, each of your strips has a constant width of 1 meter, and it takes 10 seconds to walk 1 meter (it's hard to walk while searching). Find the big-oh performance of your ring finding algorithm.

Source: https://brilliant.org/

BIG-O AND GROWTH RATE

What is growth rate of an algorithm?



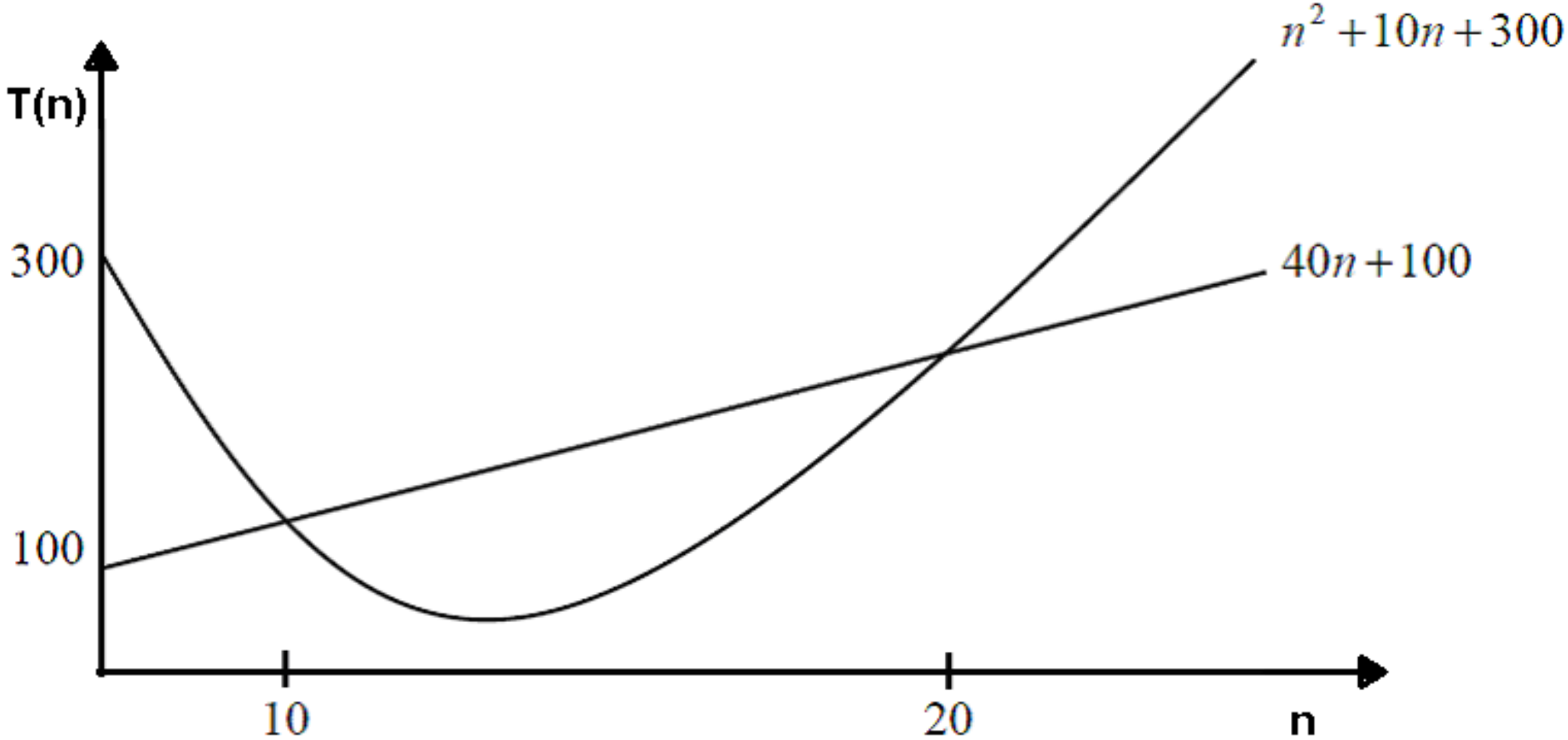
Increasing complexity

Increasing problem size

n	$\log_2 n$	$n^{0.5}$	$n \log_2 n$	n^2	n^3	2^n
2	1	1.41	2	4	8	4
4	2	2	8	16	64	16
8	3	2.83	24	64	512	256
16	4	4	64	256	4,096	65,536
32	5	5.66	160	1,024	32,768	4,294,967,296
64	6	8	384	4,094	262,144	$1.84 * 10^{19}$
128	7	11.31	896	16,384	2,097,152	$3.40 * 10^{38}$
256	8	16	2,048	65,536	16,777,216	$1.15 * 10^{77}$
512	9	22.63	4,608	262,144	134,217,728	$1.34 * 10^{154}$
1024	10	32	10,240	1,048,576	1,073,741,824	$1.79 * 10^{308}$
2048	11	45.25	22,528	4,194,304	8,589,934,592	$3.23*10^{616}$

Source: The Internet

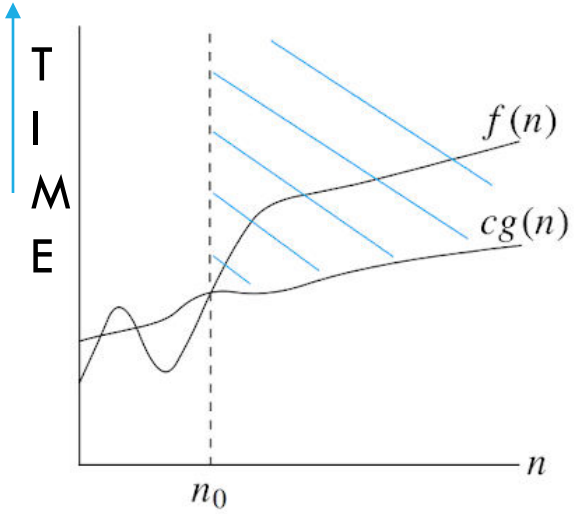
GROWTH RATE CONTINUED…



RELATIVES OF BIG-OH (Ω AND Θ)

Big-Omega Notation (Ω)

- Just like Bio-O provides asymptotic upper-bound, Big-Ω provides asymptotic lower-bound on the running time.
- f(n) is Ω(g(n)) if there exists a constant c > 0 and an integer constant $n_0 \geq 1$ such that $f(n) \geq c.g(n)$ for all $n \geq n_0$



f(n) = Ω(g(n))

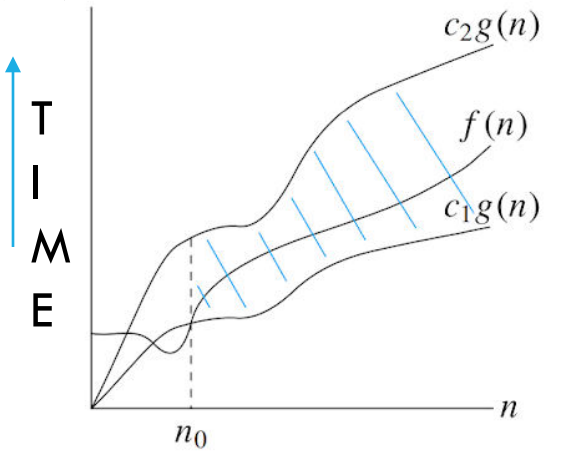
Let, f(n) = 3n.logn + 2n Justification: 3n.logn+2n ≥ 3n.logn, for n ≥ 2

Ω(nlog n)

Big-Theta Notation (Θ)

f(n) is Θ(g(n)) , if: *f*(*n*) is <u>both</u> O(*g*(*n*)) and Ω(*g*(*n*))

f(n) is Θ(g(n)) if there are constants $c_1 > 0$ and $c_2 > 0$ and an integer constant $n_0 \geq 1$ such that $c_1.g(n) \leq f(n) \leq c_2.g(n)$ for $n \geq n_0$



f(n) = Θ(g(n))

3nlogn+4n+5logn is Θ(nlog n) 3nlogn ≤ 3nlogn+4n+5logn ≤ (3+4+5) nlogn for n ≥ 2

EXAMPLES CONTINUED...

```
#include <stdio.h>
// Linearly search x in arr[].
// If x is present then return the index,
// otherwise return -1
int search(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++) {
        if (arr[i] == x)
            return i;
    }
    return -1;
}
```

```
/* Driver's code*/
int main()
{
    int arr[] = { 1, 10, 30, 15 };
    int x = 30;
    int n = sizeof(arr) / sizeof(arr[0]);

    // Function call
    printf("%d is present at index %d", x,
           search(arr, n, x));

    return 0;
}
```

What are the best case, worst case, and average case complexities of the above code?

ASYMPTOTIC ANALYSIS

What is the need of Asymptotic Analysis?

In Asymptotic Analysis, we evaluate the performance of an algorithm in terms of input size (we don't measure the actual running time). We calculate, how the time (or space) taken by an algorithm increases with the input size using Big-O notation(Worst-case no. of primitive operations).

Ex: Searching in a sorted array using Linear search (on fast computer A) and Binary search (on slow computer B).

n	Running time on A	Running time on B
10	2 sec	~ 1 hr
100	20 sec	~ 1.8 hrs
10^6	~ 55.5 hrs	~ 5.5 hrs
10^9	~ 6.3 yrs	~ 8.3 hrs

(with linear search running time on A=0.2*n)

(with binary search running time on machine B = 1000 * log(n))

Source: https://www.geeksforgeeks.org/

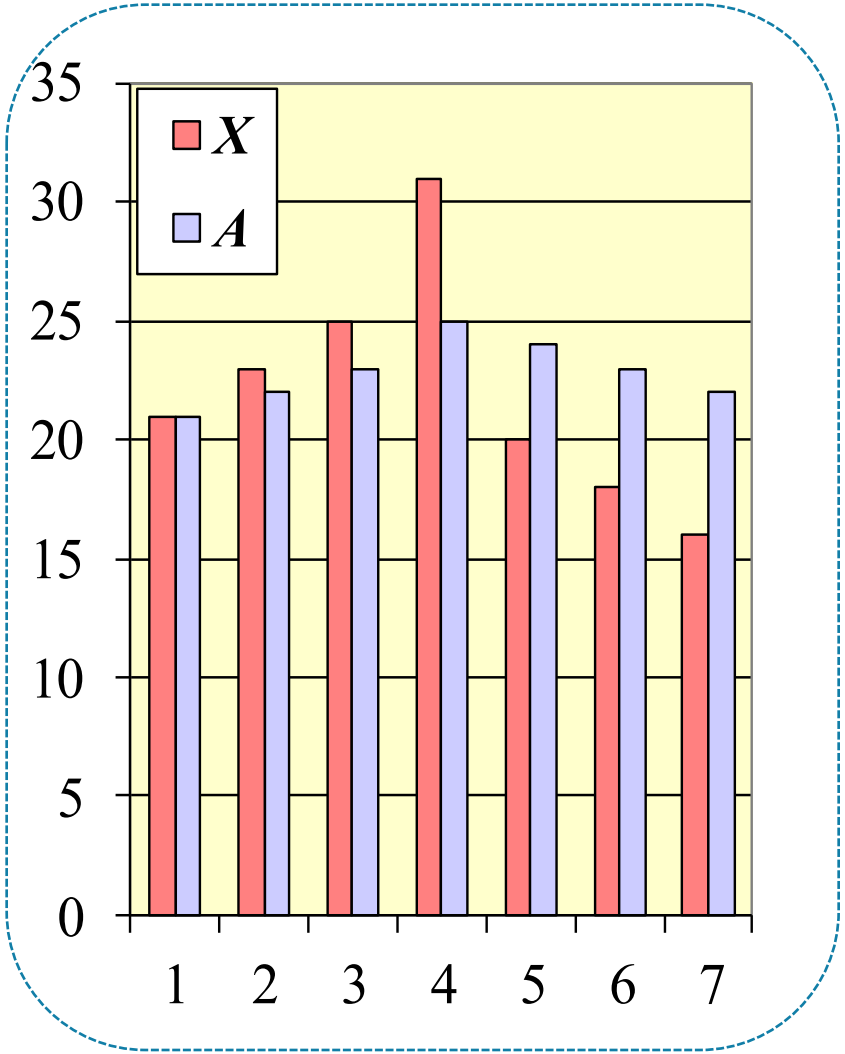
PREFIX AVERAGE EXAMPLE

- A[i] = (X[0] + X[1] + … + X[i])/(i+1)

Applications: Eco (Mutual fund averages)

```
Algorithm prefixAverages1(X, n)
Input array X of n integers
Output array A of prefix averages of X
A←new array of n integers                n
for i ← 0 to n − 1 do                    n
  s ← 0                                  n
  for j ← 0 to i do                      1 + 2 + …+ n
    s ← s + X[j]                         1 + 2 + …+ n
  A[i] ← s / (i + 1)                     n
return A                                 1
```

```
Algorithm prefixAverages2(X, n)
Input array X of n integers
Output array A of prefix averages of X
A ← new array of n integers              n
s ← 0                                    1
for i ← 0 to n − 1 do                    n
  s ← s + X[i]                           n
  A[i] ← s / (i + 1)                     n
return A                                 1
```



TRY YOURSELF…

```
int i = 1, j;
while(i <= n) {
   j = 1;
   while(j <= n)
   {
      statements of O(1)
      j = j*2;
   }
   i = i+1;
}
```

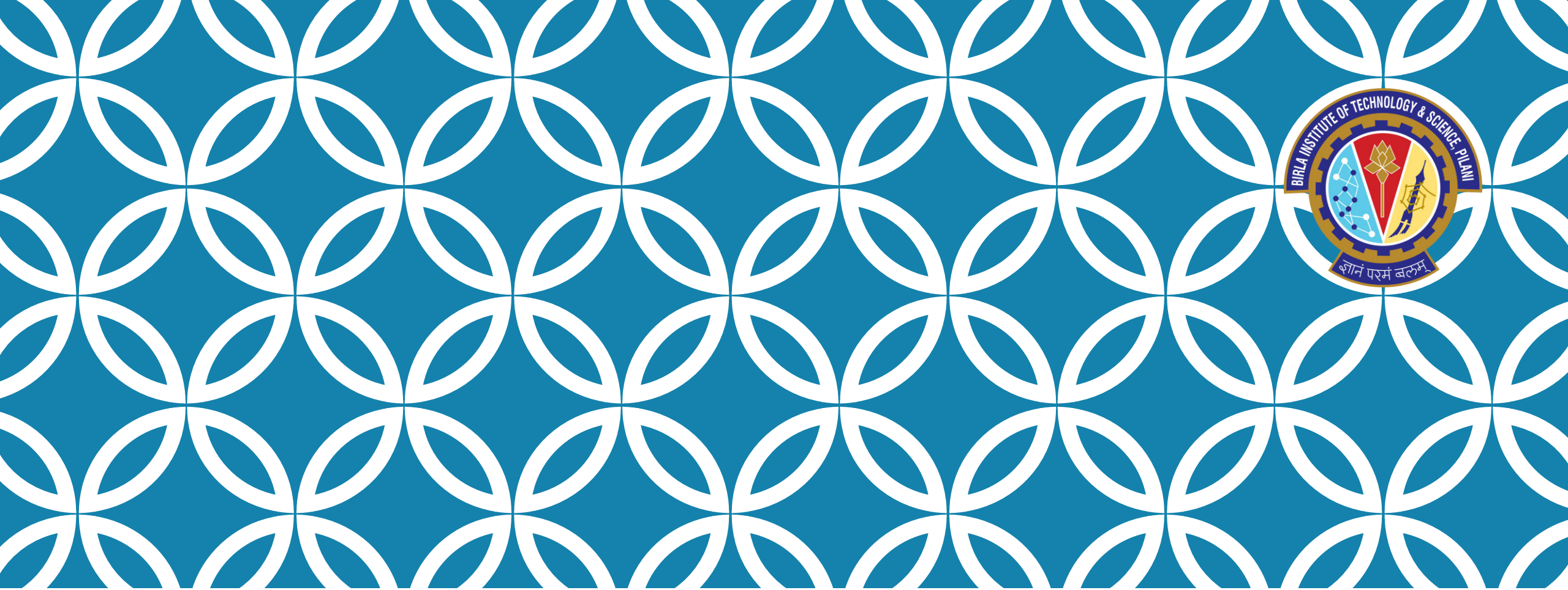
O(nlogn)

```
int sum = 0;

for(int i = 1; i <= n; i++)

   for(int j = i; j < 0; j++)

         sum += i * j ;
```

O(n)

```
for (int j = 0; j < n * n; j++)
   sum = sum + j;

for (int k = 0; k < n; k++)
   sum = sum - l;

print("sum is now " + sum);
```

$O(n^2)$



BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) STACK ADT

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

RECURSIVE FUNCTIONS: RECURRENCE RELATION (EX1)

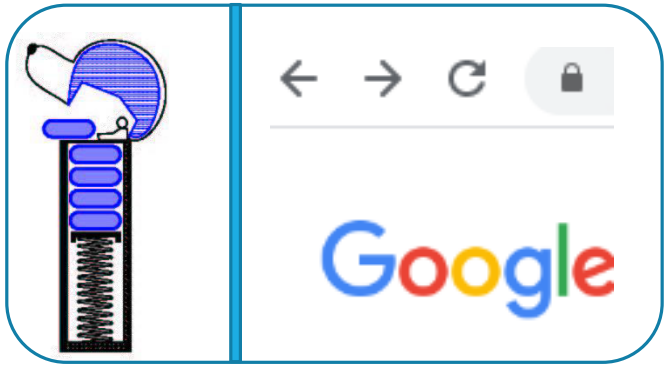
```
void fun(int n) {
   if (n > 0)
   {
     printf("%d", n);
     fun(n-1);
   }
}

Int main() {
  int x = 4;
  fun(x);
  return 0;
}
```

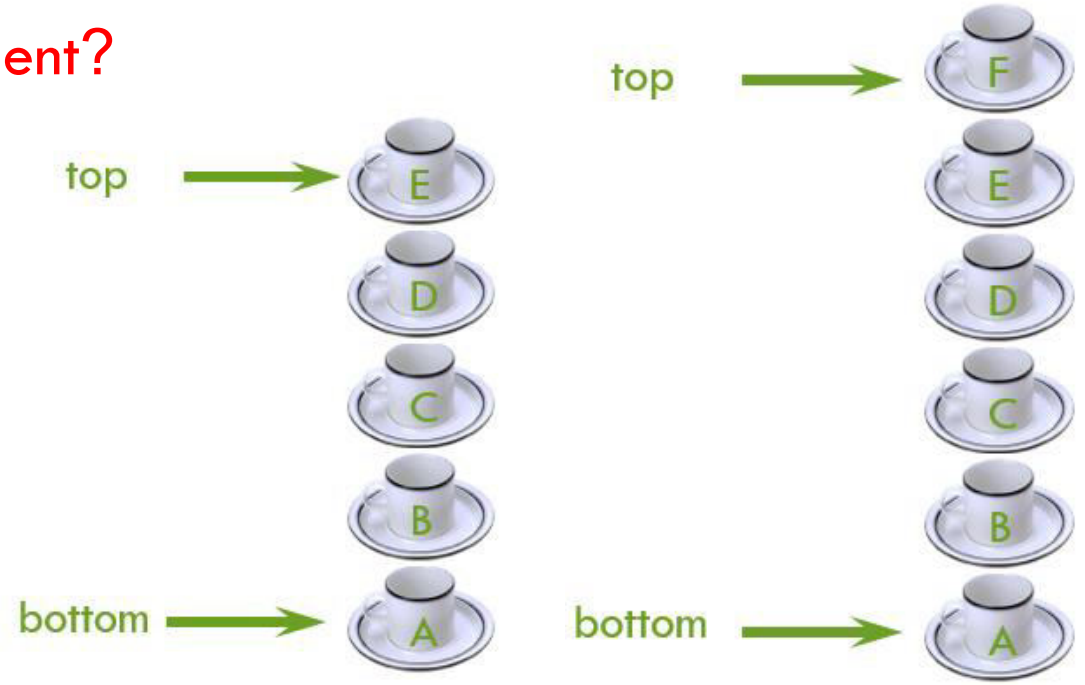
EXAMPLE 2

$T(n) = 2\,T(n/2) + n$ Where, $T(1) = 1$

STACK ADT

- A stack S is a linear sequence of elements to which elements x can only be inserted and deleted from the head of the list in the order they appear.
- What type of policy does a stack implement?

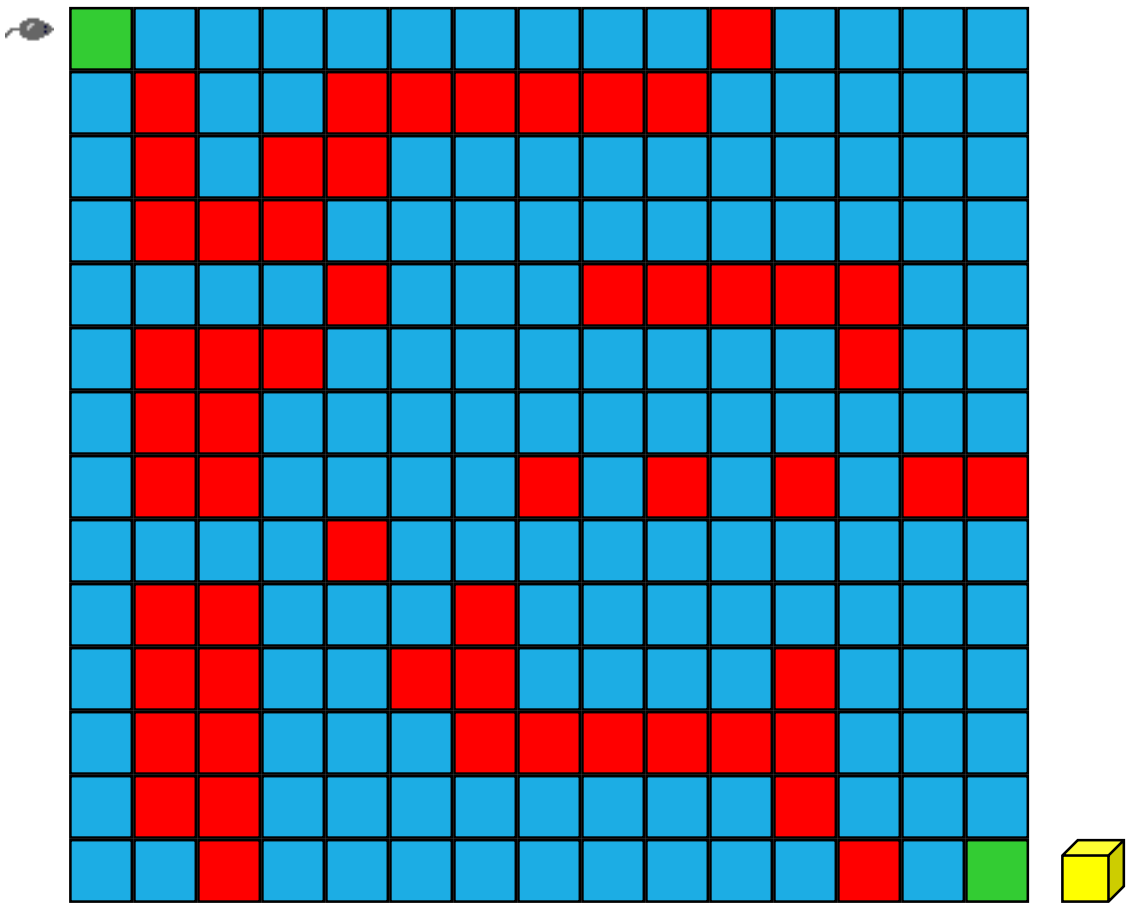


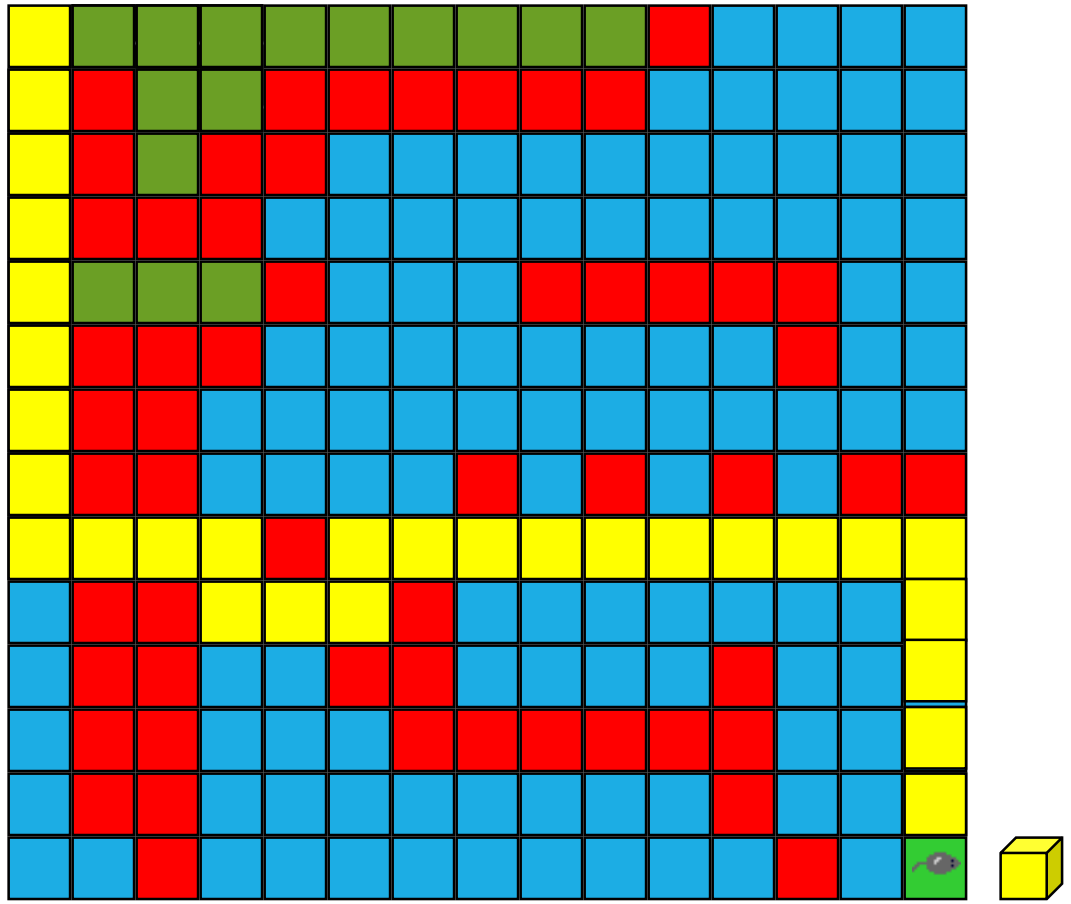


Example usage: Parenthesis matching

Depth first search, expression conversion/evaluation etc.

ANOTHER EXAMPLE USAGE: RAT IN A MAZE





STACK USAGES CONTINUED…

```
main () {
        int i = 5;
        foo(i);
}

foo (int j)

{
         int k;
         k = j+1;
         bar(k);

}

bar (int m) {

        int h = 7;

        int l = m + h;

}
```

(Runtime Stack)

(Stock span: last 5 days)

ABSTRACT DATA TYPES (ADT) & STACK INTERFACE

- An ADT is an abstraction of a data structure. Specifies data stored, operations on data and error conditions associated with operations on these data, if any.
- Recollect the GRAPH ADT we used for traffic light system design (in the introductory lectures).
- Another example: Stock Trading(Data: Orders; Operations: Buy/Sell/Cancel; Errors: Buying non-existent stock etc).
- STACK ADT: (Data: Arbitrary objects; Operations: Push, Pop; Auxiliary operations: top (returns the last inserted element without removing it, size (returns the no. of elements stored, empty (if no elements are there in the stack).

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

template <typename E>
class ArrayStack {
    enum { DEF_CAPACITY = 100 };
    public:
        ArrayStack(int cap = DEF_CAPACITY);
        int size() const;
        bool empty() const;
        const E& top();
        void push(const E& e);
        void pop();
    private:
        E* S;     // array of stack elements
        int capacity;
        int t;
};
```

ARRAY-BASED STACK IMPLEMENTATION

```
56  template <typename E>
57  class ArrayStack
58  {
59      enum
60      {
61          DEF_CAPACITY = 100
62      }; // default stack capacity
63  public:
64      ArrayStack(int cap = DEF_CAPACITY);
65      int size() const;
66      bool empty() const;
67      const E &top();
68      void push(const E &e);
69      void pop();
70
71  private:
72      E *S;
73      int capacity;
74      int t;
75  };
76
77  template <typename E> // push element
78  void ArrayStack<E>::push(const E &e)
79  {
80      if (size() == capacity)
81          cout << "Push to full stack\n";
82      S[++t] = e;
83  }
```

COMPLEXITY?

```
85  template <typename E> // pop the stack
86  void ArrayStack<E>::pop()
87  {
88      if (empty())
89          cout << "Pop from empty stack\n";
90      --t;
91  }
```

```
97  template <typename E>
98  int ArrayStack<E>::size() const
99  {
100     return (t + 1);
101 } // number of items in the stack
102
103 template <typename E>
104 bool ArrayStack<E>::empty() const
105 {
106     return (t < 0);
107 } // is the stack empty?
108
109 template <typename E> // return top of stack
110 const E &ArrayStack<E>::top()
111 {
112     if (empty())
113         cout << "Top of empty stack\n";
114     return S[t];
115 }
```

BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24)

STACK ADT CONTINUED…

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

STACK USAGE EXAMPLES

Let S be an empty stack
for i=0 to n-1 **do**
 if $X[i]$ is an opening grouping symbol **then**
  S.push($X[i]$)
 else
  if $X[i]$ is a closing grouping symbol **then**
   if S.empty() **then**
    return false {nothing to match with}
   if S.pop() does not match the type of $X[i]$ **then**
    return false {wrong type}
if S.empty() **then**
  return true {every symbol matched}
else
  return false {some symbols were never matched}

```
((a + b) * c + d - e) / (f + g) - (h + j) * k - l / (m - n):  BALANCED! (0.005 ms.)
(({(){}}):  BALANCED! (0.001 ms.)
({}){(){}}:  BALANCED! (0.001 ms.)
(){}(){}:  NOT Balanced (0 ms.)
```

(Output)

```
138         if (token == '(' || token == '{') // this is an opening bracket
139         {
140             stack.push(token);
141         }
142         else if (token == ')') // found closing parentheses
143         {
144             if (stack.empty() || stack.top() != '(')
145             {
146                 return false; // match not found
147             }
148             stack.pop(); // match found
149         }
```

Lab-6 (Next week's lab)

STL

```
#include <stack>
using std::stack;                    // make stack accessible
stack<int> myStack;                  // a stack of integers
```

STACK USAGE: REVERSING A VECTOR EXAMPLE

```
template <typename E>
void reverse(vector<E>& V) {
    ArrayStack<E> S(V.size());
    for (int i = 0; i < V.size(); i++)
        S.push(V[i]);
    for (int i = 0; i < V.size(); i++) {
        V[i] = S.top();  S.pop();
    }
}
```

Non-recursive algo…

```
Enter size of input vector : 4

Enter input vector : 2 5 7 9

Input vector : 2 5 7 9

Reversed vector : 9 7 5 2
```

STACK ADT: LINKED LIST IMPLEMENTATION

IMPLEMENTING A STACK WITH A GENERIC LINKED LIST

```
88  int LinkedStack::size() const
89  { return n; }
90
91  bool LinkedStack::empty() const
92  { return n == 0; }
93
94
95  const Elem& LinkedStack::top() {
96      if (empty()) cout<<"Top of empty stack\n";
97      return S.front();
98  }
99
100 void LinkedStack::push(const Elem& e) {
101     ++n;
102     S.addFront(e);
103 }
104
105 void LinkedStack::pop() {          // pop the stack
106     if (empty()) cout<<"Pop from empty stack\n";
107     --n;
108     S.removeFront();
109 }
```

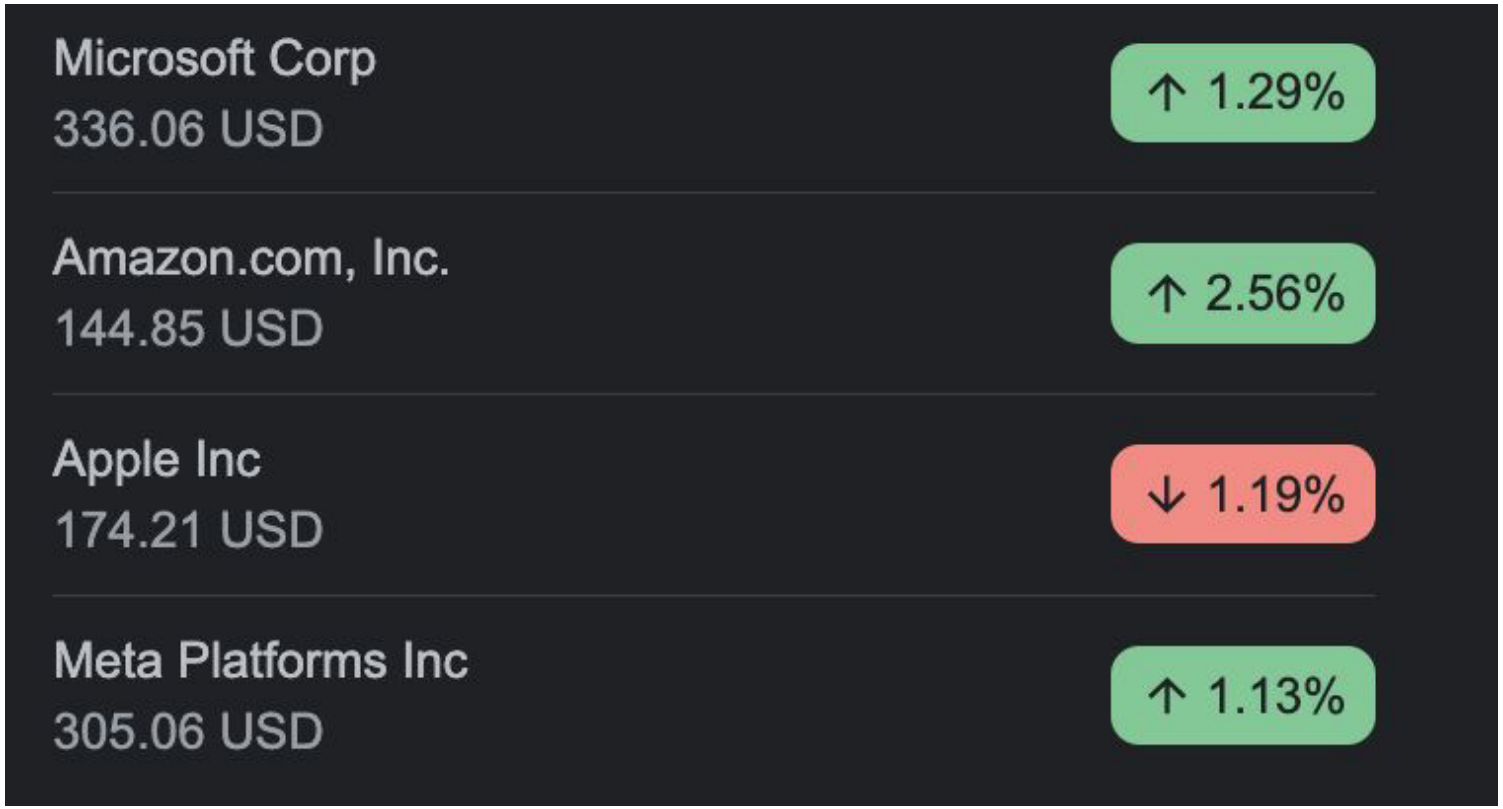
```
template <typename E>
void SLinkedList<E>::removeFront() {
    SNode<E>* old = head;
    head = old->next;
    delete old;
}

template <typename E>
void SLinkedList<E>::traverse(){
    SNode<E> *temp = head;
    while(temp != NULL){
        cout<<temp->elem<<" ";
        temp = temp->next;
    }
    cout<<endl;
}

typedef string Elem;
class LinkedStack {
    public:
        LinkedStack();
        int size() const;
        bool empty() const;
        const Elem& top();
        void push(const Elem& e);
        void pop();
    private:
        SLinkedList<Elem> S;
        int n;
};
```

```
Enter input 1
10
Pushing : 10
Enter input 1
20
Pushing : 20
Enter input 1
30
Pushing : 30
Enter input 3
Getting top
30
Enter input 4
Getting size
3
Enter input 5
Stack is not empty
Enter input 2
Attempting pop
Enter input 3
Getting top
20
Enter input
```

STACK USAGE: MATCHING TAGS IN AN HTML DOC

```
bool isHtmlMatched(const vector<string>& tags) {
    LinkedStack S;
    typedef vector<string>::const_iterator Iter;

    for (Iter p = tags.begin(); p != tags.end(); ++p) {
        if (p->at(1) != '/')
            S.push(*p);
        else {
            if (S.empty()) return false;
            string open = S.top().substr(1);
            string close = p->substr(2);
            if (open.compare(close) != 0) return false;
            else S.pop();
        }
    }
    if (S.empty()) return true;
    else return false;
}
```

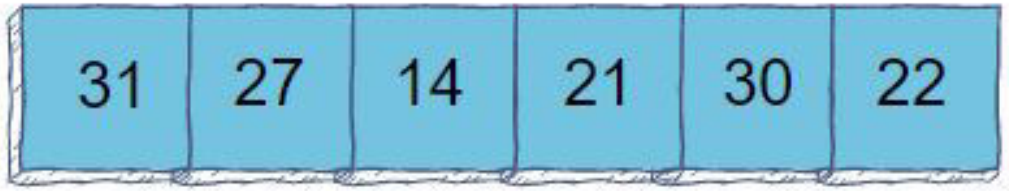
COMPUTING STOCK SPAN: STACK USAGE

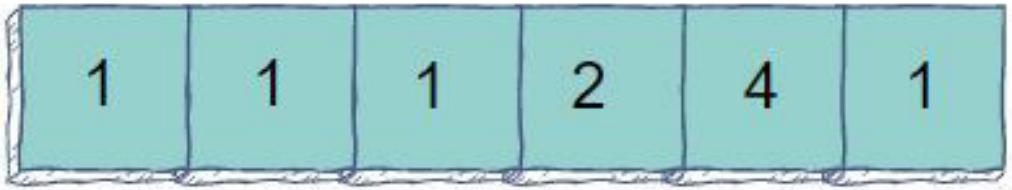


Source: Internet, 14th Sept 2023

COMPUTING STOCK SPAN CONTINUED…

X



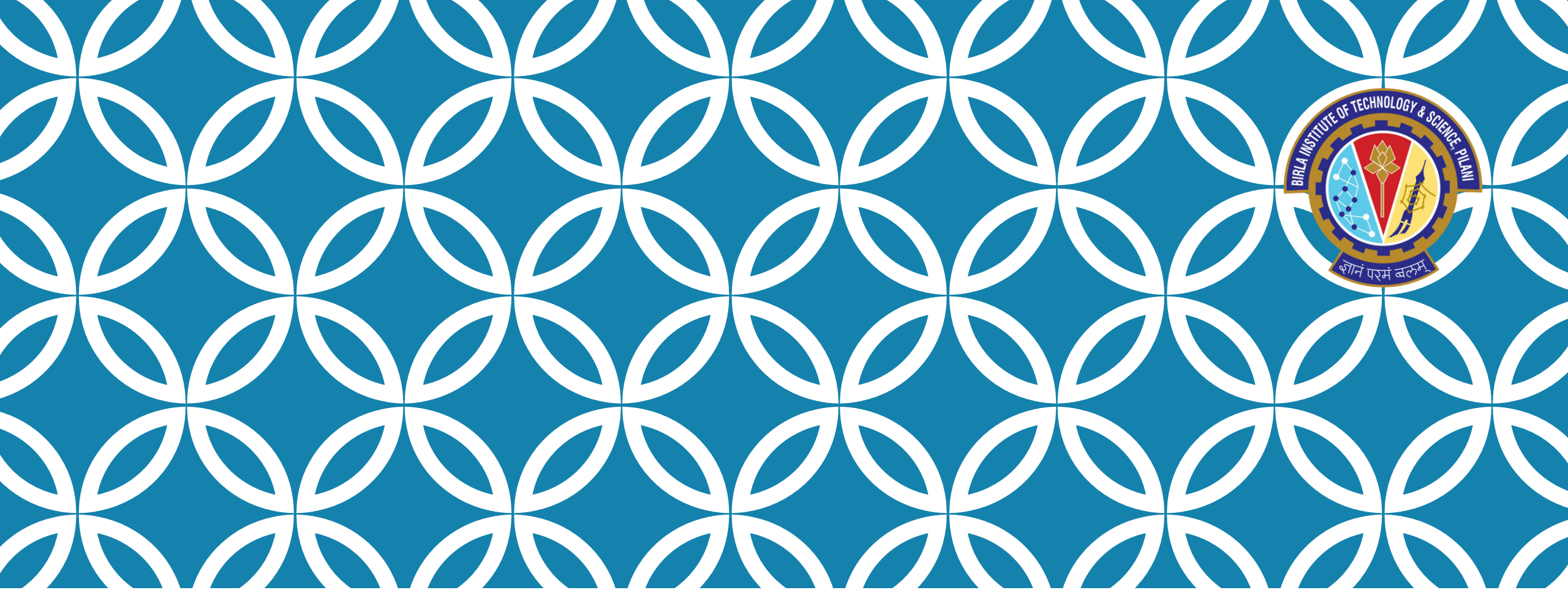


S

```
Input Stocks Data: 6 3 4 5 2
Output Spans: 1 1 2 3 1 (0.004 ms.)
Input Stocks Data: 2 4 5 6 7 8 9
Output Spans: 1 2 3 4 5 6 7 (0 ms.)
Input Stocks Data: 100 80 60 70 60 75 85
Output Spans: 1 1 1 2 1 4 6 (0.001 ms.)
```

Algorithm *spans2*(*X*, *n*) Complexity: O(n)

A ← new array of *n* integers	*n*
S ← new empty stack	1
for *i* ← 0 **to** *n* − 1 **do**	*n*
while (¬*S.empty*() ∧ *X*[*S.top*()] ≤)	*n*
S.pop()	*n*
if *S.empty*() **then**	*n*
A[*i*] ← *i* + 1	*n*
Lab 6: Next week's Lab	
	n



BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) QUEUE ADT

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

SOLVING A MAZE USING A STACK

```
1 1 1 1 1
1 1 0 0 1
1 0 0 0 1
m 0 1 e 0
```

```
1 1 1 1 1
1 1 0 0 1
1 0 1 0 1
m 0 1 e 0
```

```
Enter a rectangular maze using the following characters:
m - entry
e - exit
1 - wall
0 - passage
Enter one line at at time; end with Ctrl-z:
1111111
1111111
1110011
1100011
1m01e01
1111111

1111111
1111111
1110011
1100011
1m01e01
1111111

1111111
1111111
1110011
1100011
1m.1e01
1111111
```

Lab 6: Next week's Lab

```
1111111
1111111
1110011
11.0011
1m.1e01
1111111

1111111
1111111
1110011
11..011
1m.1e01
1111111

1111111
1111111
1110011
11...11
1m.1e01
1111111

Success
```

```
1111111
1111111
1110011
1101011
1m.1e01
1111111

1111111
1111111
1110011
11.1011
1m.1e01
1111111

Failure
```

QUEUES

Img. Source: https://www.javascripttutorial.net/

Some Applications of Queues:

1. Network routers
2. Scheduling a single shared resource (CPU)
3. Concurrent web servers

Is it NOT a linear data structure?

What operations would you like to see in a queue ADT?

EXAMPLE SERIES OF OPERATIONS ON A QUEUE

ARRAY-BASED IMPLEMENTATION OF A QUEUE

Approach 1: Similar to stack based implementation where Q[0] be the front of the queue and have the queue grow from there.

How good is this approach?

Approach 2: Using an Array with three variables to avoid moving objects once they are placed in the queue.

Use three variables: f, r, and n. Let us see the dequeue and enqueue operations… (What is the complexity?)

If we repeatedly enqueue and dequeue a single element, what problem it might cause?

CONTINUED...

Approach 3: Use a **circular array** with 'f' and 'r' indices wrapping around the end of the queue.

Which operator in C++ can do this?

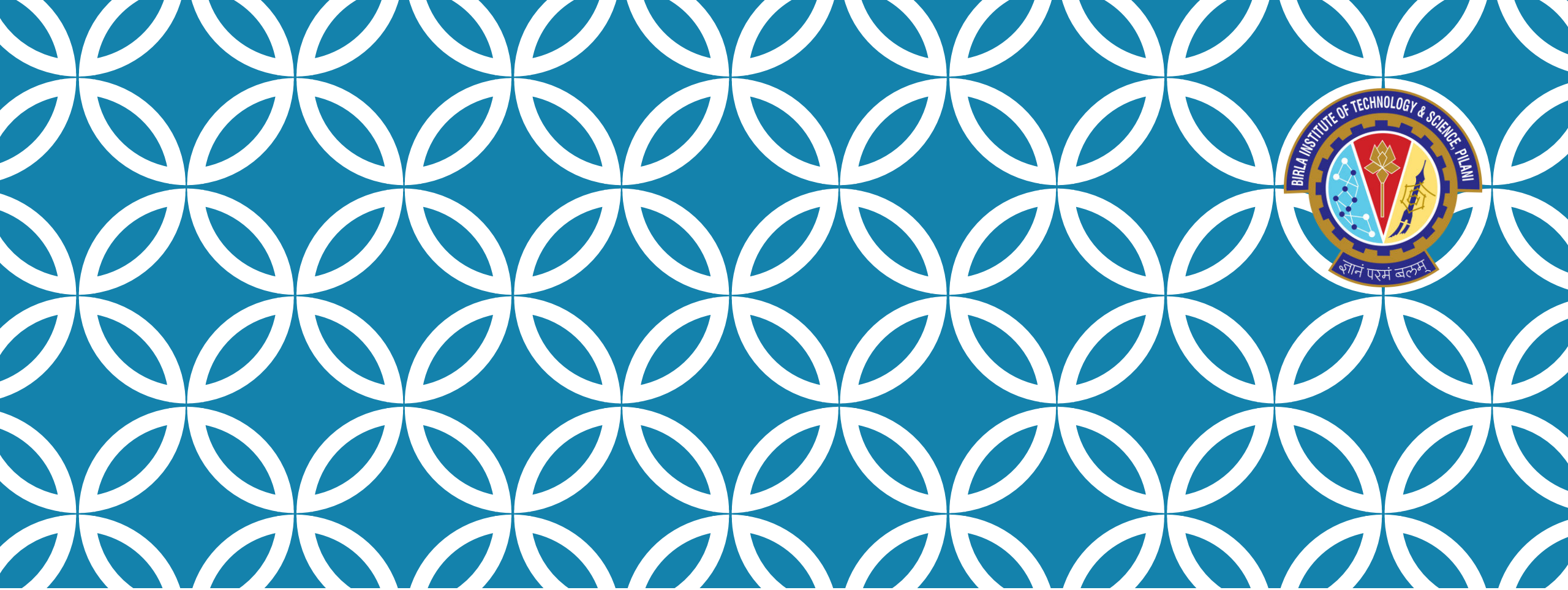
OPERATIONS USING CIRCULAR ARRAY & INTERFACE

```
Algorithm size()
    return n

Algorithm empty()
    return (n == 0)
```

```
Algorithm dequeue ()
    if empty() then
        throw QueueEmpty
    else
        ???
        ???
```

```
Algorithm enqueue (P) {

    if size() == N – 1 then
        throw QueueFull

    else {

        ???
        ???
        ???
    }
}
```

```
template <typename E>

class Queue {
public:
    int size() const;
    bool empty() const;
    const E& front() const
        throw(QueueEmpty);
    void enqueue (const E& e);
    void dequeue()
        throw(QueueEmpty);
};
```



BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) QUEUE ADT CONTINUED…

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

RECAP

QUEUE ADT USING CIRCULAR LINKED-LIST

```
typedef string Elem;
class LinkedQueue {
public:
  LinkedQueue();
  int size() const;
  bool empty() const;
  const Elem& front() const throw(QueueEmpty);
  void enqueue(const Elem& e);
  void dequeue() throw(QueueEmpty);
private:
  CircleList C;
  int n;
};
```

(Class structure for Linked queue)

QUEUE IMPLEMENTATION USING CIRCULAR LINKED-LIST IN C++

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

class CircleList;

typedef string Elem;
class CNode {
private:
    Elem elem;
    CNode* next;
    friend class CircleList;
};

class CircleList {
public:
    CircleList();
    ~CircleList();
    bool empty() const;
    const Elem& front() const;
    const Elem& back() const;
    void advance();
    void add(const Elem& e);
    void remove();
    void traverse();
private:
    CNode* cursor;
};
```

```cpp
CircleList::CircleList()
    : cursor(NULL) { }
CircleList::~CircleList()
    { while (!empty()) remove(); }

bool CircleList::empty() const
    { return cursor == NULL; }
const Elem& CircleList::back() const
    { return cursor->elem; }
const Elem& CircleList::front() const
    { return cursor->next->elem; }
void CircleList::advance()
    { cursor = cursor->next; }

void CircleList::add(const Elem& e) {
    CNode* v = new CNode;
    v->elem = e;
    if (cursor == NULL) {
        v->next = v;
        cursor = v;
    }
    else {
        v->next = cursor->next;
        cursor->next = v;
    }
}
```

```
1
23
Enqueing 23
1
56
Enqueing 56
1
78
Enqueing 78
3
Getting front element
23
4
Getting size
3
5
Queue is not empty
2
Dequeing
3
Getting front element
56
```

```cpp
LinkedQueue::LinkedQueue()
: C(), n(0) { }

int LinkedQueue::size() const
{ return n; }

bool LinkedQueue::empty() const
{ return n == 0; }

const Elem& LinkedQueue::front() {
    if (empty())
        cout<<"front of empty queue\n";
    return C.front();
}

void LinkedQueue::enqueue(const Elem& e) {
    C.add(e);
    C.advance();
    n++;
}

void LinkedQueue::dequeue() {
    if (empty())
        cout<<"dequeue of empty queue\n";
    C.remove();
    n--;
}
```

Lab 7: next week's lab

STACK USING TWO QUEUES: MAKING PUSH COSTLY

Algorithm 1: Push

```
Data: two queues: q1, q2
      element to push to stack: E
Result: element E is now at the head of queue q1
if q1.isEmpty() then
    q1.enqueue(E);
else
    q1Size:= q1.size();
    for i=0...q1Size do
        q2.enqueue(q1.dequeue());
    end
    q1.enqueue(E);
    for j=0...q1Size do
        q1.enqueue(q2.dequeue());
    end
end
```

Algorithm 2: Pop

```
Data: two queues: q1, q2
Result: the head element of the queue q1 is removed
element:= q1.dequeue();
return element;
```

What is the time complexity?

The STL Queue:

```
#include <queue>
using std::queue;
queue<float> myQueue;
```

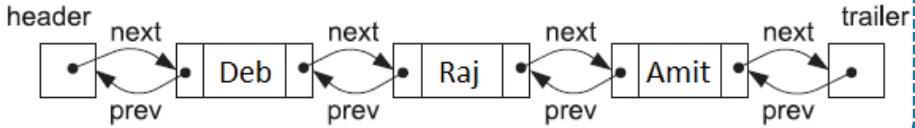
size(), empty(), push(e), pop(), front(), back()

Lab 7: next week's lab

Alternate way: making pop() costly…

Is it possible to do using only one queue?

DEQUE IMPLEMENTATION



(A Doubly linked-list with Sentinels)

```
typedef string Elem;
class LinkedDeque {
public:
  LinkedDeque();
  int size() const;
  bool empty() const;
  const Elem& front() const throw(DequeEmpty);
  const Elem& back() const throw(DequeEmpty);
  void insertFront(const Elem& e);
  void insertBack(const Elem& e);
  void removeFront() throw(DequeEmpty);
  void removeBack() throw(DequeEmpty);
private:
  DLinkedList D;
  int n;
};
```

```
void LinkedDeque::insertFront(const Elem& e) {
  D.addFront(e);
  n++;
}

void LinkedDeque::insertBack(const Elem& e) {
  D.addBack(e);
  n++;
}

void LinkedDeque::removeFront() throw(DequeEmpty) {
  if (empty())
    throw DequeEmpty("removeFront of empty deque");
  D.removeFront();
  n--;
}

void LinkedDeque::removeBack() throw(DequeEmpty) {
  if (empty())
    throw DequeEmpty("removeBack of empty deque");
  D.removeBack();
  n--;
}
```

```
1
10
Inserting 10 to the front
1
30
Inserting 30 to the front
1
60
Inserting 60 to the front
2
20
Inserting 20 to the end
2
40
Inserting 40 to the end
5
Front element
60
3
Removing from front
5
Front element
30
7
Queue is not empty
8
Size of queue : 4
```

Complexity of Operations?

A QUEUE USING TWO STACKS

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Queue {
    stack<int> s1, s2;
    void enQueue(int x)
    {
        while (!s1.empty()) {
            s2.push(s1.top());
            s1.pop();
        }
        s1.push(x);
        while (!s2.empty()) {
            s1.push(s2.top());
            s2.pop();
        }
    }
    int deQueue() {
        if (s1.empty()) {
            cout << "Q is Empty";
            exit(0);
        }
        int x = s1.top();
        s1.pop();
        return x;
    }
};
```

```cpp
int main() {
    Queue q;
    q.enQueue(10);
    q.enQueue(20);
    cout << q.deQueue() << '\n';
    q.enQueue(30);
    cout << q.deQueue() << '\n';
    cout << q.deQueue() << '\n';
    q.enQueue(40);
    cout << q.deQueue() << '\n';
    return 0;
}
```

```
10
20
30
40


...Program finished with exit code 0
Press ENTER to exit console.
```

DOUBLE-ENDED QUEUE ADT: DEQUE

- A queue-like data structure that supports insertion and deletion at both the front and the rear of the queue.
- Applications: Work-Stealing algorithm in Intel's parallel programming, etc.
- insertFront(), front(), eraseFront(), insertBack(), back(), eraseBack()

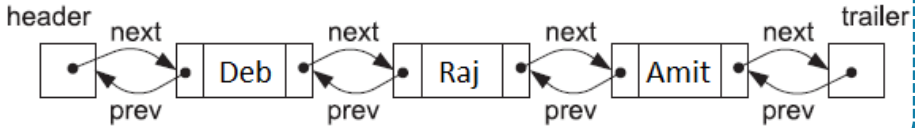
```
#include <deque>
using std::deque;
deque<string> myDeque;
```

(The STL deque)

size(), empty(),
push_front(e),
push_back(e), pop_front(),
pop_back(), front(), back()

What are some of the scenarios where Deque operations might be applicable?

DEQUE IMPLEMENTATION



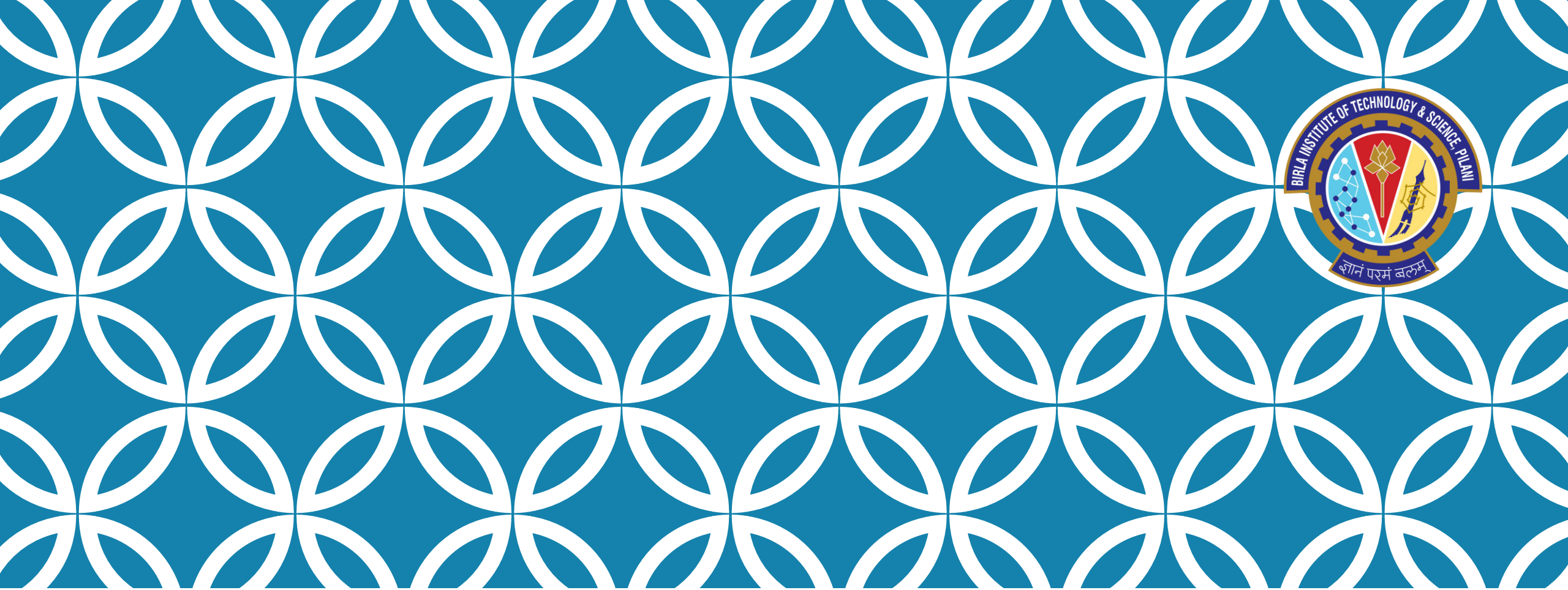
(A Doubly linked-list with Sentinels)

```
typedef string Elem;
class LinkedDeque {
public:
  LinkedDeque();
  int size() const;
  bool empty() const;
  const Elem& front() const throw(DequeEmpty);
  const Elem& back() const throw(DequeEmpty);
  void insertFront(const Elem& e);
  void insertBack(const Elem& e);
  void removeFront() throw(DequeEmpty);
  void removeBack() throw(DequeEmpty);
private:
  DLinkedList D;
  int n;
};
```

```
void LinkedDeque::insertFront(const Elem& e) {
  D.addFront(e);
  n++;
}

void LinkedDeque::insertBack(const Elem& e) {
  D.addBack(e);
  n++;
}

void LinkedDeque::removeFront() throw(DequeEmpty) {
  if (empty())
    throw DequeEmpty("removeFront of empty deque");
  D.removeFront();
  n--;
}

void LinkedDeque::removeBack() throw(DequeEmpty) {
  if (empty())
    throw DequeEmpty("removeBack of empty deque");
  D.removeBack();
  n--;
}
```

```
1
10
Inserting 10 to the front
1
30
Inserting 30 to the front
1
60
Inserting 60 to the front
2
20
Inserting 20 to the end
2
40
Inserting 40 to the end
5
Front element
60
3
Removing from front
5
Front element
30
7
Queue is not empty
8
Size of queue : 4
```

Complexity of Operations?



BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) ADAPTER DESIGN PATTERN

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

RECAP

Queue can also be implemented using one User stack and one Function Call Stack.

```
int deQueue() {
  if (s.empty()) {
     return -1;
  }
  int x = s.top();
  s.pop();
  if (s.empty())
    return x;
  int item = deQueue();
  s.push(x);
  return item;
}
```

```
void enQueue(int x) {
   s.push(x);
}
```

ADAPTERS DESIGN PATTERN

- What is an adapter/ a wrapper?

Deque
insertFront()
insertBack()
removeFront()
removeBack()
Size()
Empty()

```
typedef string Elem;
DequeStack {  // stack as a deque
   public:
        DequeStack();
        int size() const;
        bool empty() const;
        const Elem& top();
        void push(const Elem& e);
        void pop();
   private:
        LinkedDeque D; };
```

```
Queue is Empty!
Tim enqueued into the Queue.
Alex enqueued into the Queue.
Bob enqueued into the Queue.
FRONT -> Tim Alex Bob
Size of the Queue = 3.
Elem at the FRONT: Tim
Dequeuing...
Elem at the FRONT: Alex
FRONT -> Alex Bob
Smith enqueued into the Queue.
FRONT -> Alex Bob Smith
```

```
template <typename E>
void Queue<E>::enqueue(E elem)
{
    dq. insertBack(elem);
}

template <typename E>
void Queue<E>::dequeue()
{
    if (dq.empty())
        throw("Queue Underflows!");
    dq.removeFront();
}
```

Lab 7:next week's lab: queue using deque.

```
203 void DequeStack::push(const Elem& e)
204 {
205     D.insertFront(e);
206 }
```

```
206 void DequeStack::pop(){
207     if (empty())
208         cout<<"pop of empty stack\n";
209     D.removeFront();
210 }
```

VECTOR OR ARRAY LIST ADT

- Vector is a sequential container to store elements.

- Vector is dynamic in nature.

- Which one takes more time to access elements? Arrays or Vectors.

Main methods:

at(i), set(i, o), insert(i, o), erase(i), size(), empty()

Assuming that the vector is initially empty, find out the contents of V for the following operations?

insert (0, 60)

at (1)

insert (1, 70)

insert (0, 40)

erase (1)

set (1, 50)

SIMPLE ARRAY-BASED IMPLEMENTATION

Use an array ***A*** of size ***N***

A variable ***n*** keeps track of the size of the array list (number of elements stored)

How will you implement?

at (i)

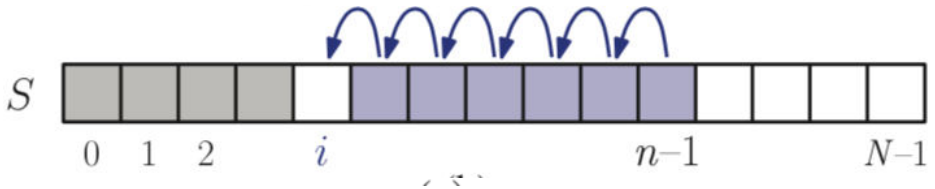
set (i, o)

insert (i, o)

erase (i)

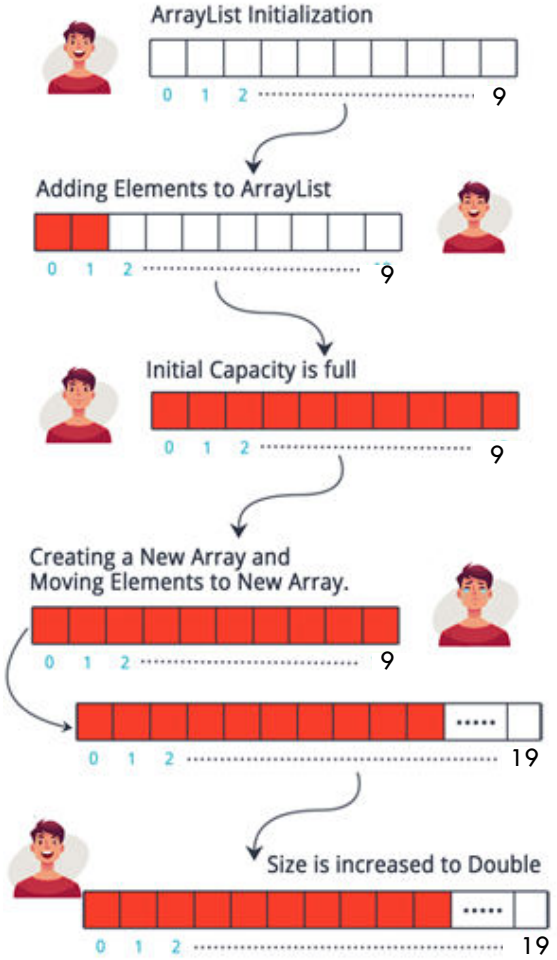
Algorithm insert(*i*,o):

Algorithm erase(*i*):



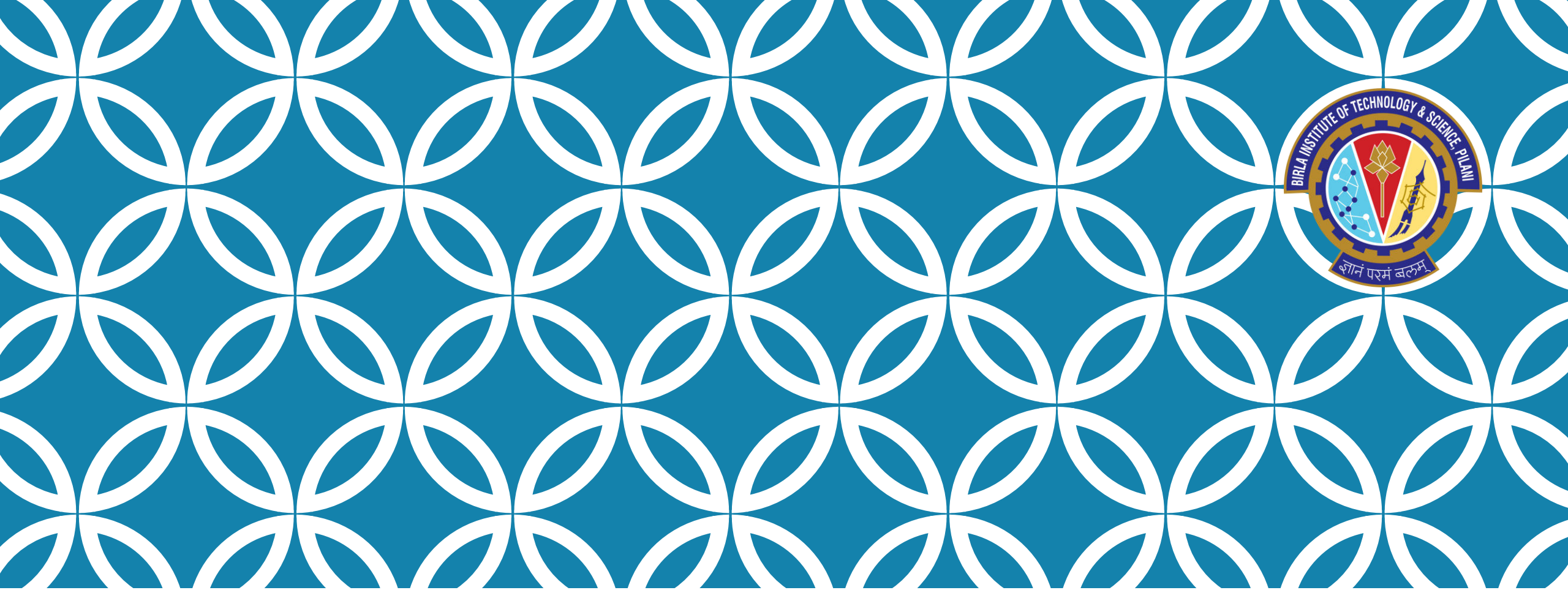
What would be the performance of a vector realized by an array?

```
1 : Insert
2 : Erase
3 : Get element at index i
4 : Get size
5 : Check if vector is empty
6 : Exit
1
Enter index and element :
0 10
3
Enter index :
0
10
1
Enter index and element :
1 20
4
Getting size
2
5
Vector is not empty
2
Enter index :
1
4
Getting size
1
```



AMORTIZATION (A DESIGN PATTERN)

- We compare the incremental strategy and the doubling strategy by analyzing the total time $T(n)$ needed to perform a series of n insert(o) operations.
- We assume that we start with an empty vector represented by an array of size 1
- We call **amortized time** of an insert operation as the average time taken by an insert over the series of operations, i.e., $T(n)/n$



BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) STL VECTORS, LIST ADT

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

RECAP

STL VECTORS WITH ALGORITHMS

```
#include <vector>
using std::vector;

vector<int> myVector(100);
```

sort(p,q):

random_shuffle(p,q):

reverse(p,q):

find(p,q,e):

min_element(p,q):

max_element(p,q):

for_each(p,q,f):

vector(n):

size():

empty():

resize(n):

reserve(n):

operator$[i]$:

at(i):

front():

back():

push_back(e):

pop_back():

```
#include <cstdlib>                                  // provides EXIT_SUCCESS
#include <iostream>                                 // I/O definitions
#include <vector>                                   // provides vector
#include <algorithm>                                // for sort, random_shuffle

using namespace std;                                // make std:: accessible

int main () {
  int a[] = {17, 12, 33, 15, 62, 45};
  vector<int> v(a, a + 6);                          // v: 17 12 33 15 62 45
  cout << v.size() << endl;                         // outputs: 6
  v.pop_back();                                     // v: 17 12 33 15 62
  cout << v.size() << endl;                         // outputs: 5
  v.push_back(19);                                  // v: 17 12 33 15 62 19
  cout << v.front() << " " << v.back() << endl;     // outputs: 17 19
  sort(v.begin(), v.begin() + 4);                   // v: (12 15 17 33) 62 19
  v.erase(v.end() - 4, v.end() - 2);                // v: 12 15 62 19
  cout << v.size() << endl;                         // outputs: 4

  char b[] = {'b', 'r', 'a', 'v', 'o'};
  vector<char> w(b, b + 5);                         // w: b r a v o
  random_shuffle(w.begin(), w.end());               // w: o v r a b
  w.insert(w.begin(), 's');                         // w: s o v r a b

  for (vector<char>::iterator p = w.begin(); p != w.end(); ++p)
    cout << *p << " ";                              // outputs: s o v r a b
  cout << endl;
  return EXIT_SUCCESS;
}
```

POSITION ADT & ITERATORS: LIST ADT

- What is a Position ADT?
- It gives a unified view of diverse ways of storing data, such as:
  - a cell of an array
  - a node of a linked list
- Just one method:
  - object p.element(): returns the element at position
  - In C++ it is convenient to implement this as what?

- List ADT establishes a before/after relation between positions

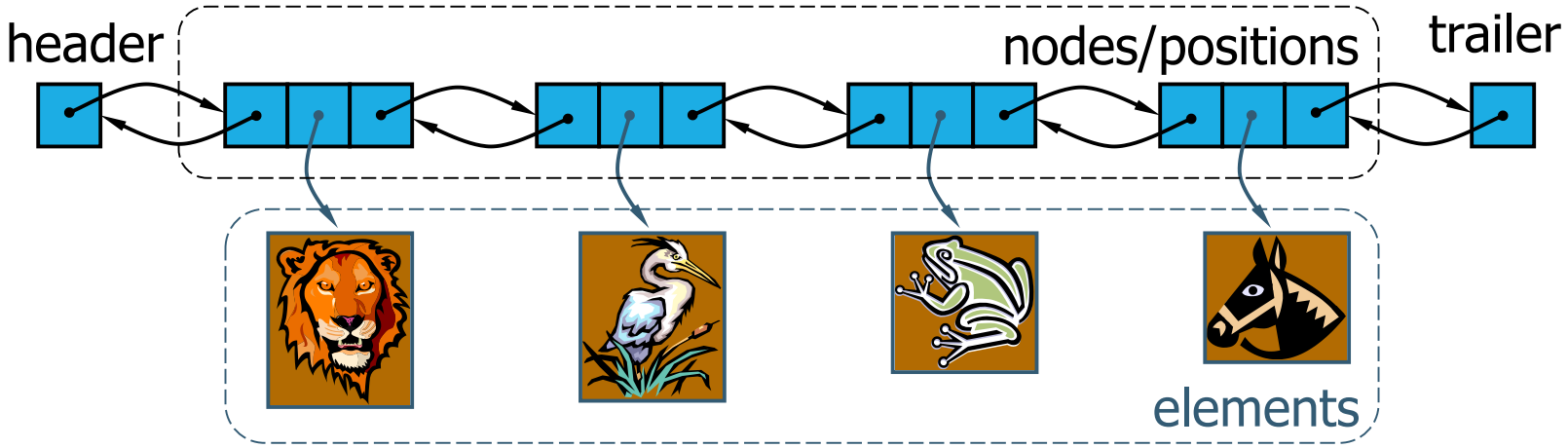
DOUBLY LINKED LIST

A doubly linked list provides a natural implementation of the List ADT.

Nodes implement Position and store:

- element
- link to the previous node
- link to the next node

Special trailer and header nodes are used as Sentinels.



Complexity?

Algorithm insert(p, e): {insert e before p}

1. Create a new node v
2. v→element = e
3. u = p→prev
4. v→next = p; p→prev = v {link in v before p}
5. v→prev = u; u→next = v {link in v after u}

Algorithm remove(p):

u = p→prev
w = p→next
u→next = w {linking out p}
w→prev = u

CONTAINERS AND ITERATORS

- What is a Container?
- Can you give some examples?
- Various notions of iterator:
  - (standard) iterator: allows read-write access to elements
  - const iterator: provides read-only access to elements
  - bidirectional iterator: supports both ++p and --p
  - random-access iterator: supports both p+i and p-i

Let C be a container and p be an iterator for C:

How will you iterate through the container?

Example: (with an STL vector)

```
typedef vector<int>::iterator Iterator;
int sum = 0;
for (Iterator p = V.begin(); p != V.end(); ++p)
  sum += *p;
return sum;
```

STL LISTS IN C++

```cpp
#include <algorithm>
#include <iostream>
#include <list>

int main()
{
    std::list<int> l = {17, 22, 10, 55, 86};

    l.push_front(30);

    l.push_back(40);

    auto it = std::find(l.begin(), l.end(), 55);
    if (it != l.end())
        l.insert(it,77);

    // Print out the list
    std::cout << "list = { ";
    for (int n : l)
        std::cout << n << " ";
    std::cout << "}\n";
}
```

```
list = { 30 17 22 10 77 55 86 40 }
```

```cpp
#include <iostream>
#include <list>
#include <iterator>
using namespace std;
//function for printing the elements in a list
void showlist(list <int> g)
{
    list <int> :: iterator it;
    for(it = g.begin(); it != g.end(); ++it)
        cout << '\t' << *it;
    cout << '\n';
}
int main() {
    list <int> gqlist1, gqlist2;
    for (int i = 0; i < 10; ++i)
    {
        gqlist1.push_back(i * 2);
        gqlist2.push_front(i * 3);
    }
    cout << "\nList 1 (gqlist1) is : ";
    showlist(gqlist1);
    cout << "\nList 2 (gqlist2) is : ";
    showlist(gqlist2);
    cout << "\ngqlist1.front() : " << gqlist1.front();
    cout << "\ngqlist1.back() : " << gqlist1.back();
    cout << "\ngqlist1.pop_front() : ";
    gqlist1.pop_front();
    showlist(gqlist1);
    cout << "\ngqlist2.pop_back() : ";
    gqlist2.pop_back();
    showlist(gqlist2);
    cout << "\ngqlist1.reverse() : ";
    gqlist1.reverse();
    showlist(gqlist1);
    cout << "\ngqlist2.sort(): ";
    gqlist2.sort();
    showlist(gqlist2);
    return 0;
}
```

https://en.cppreference.com/ https://www.geeksforgeeks.org/

INDEX VS POSITION: MORE EXAMPLES

Using Indexing Operator

```cpp
#include <iostream>
#include <vector>
using namespace std;
int vectorSum1(const vector<int>& V) {
    int sum = 0;
    for (int i = 0; i < V.size(); i++)
        sum += V[i];
    return sum;
}
int main(){
    vector<int> v;
    int size;
    cout<<"Enter size of input vector : ";
    cin>>size;
    int aux;
    for(int i=0;i<size;i++){
        cin>>aux;
        v.push_back(aux);
    }
    cout<<"\nSum : "<<vectorSum1(v)<<endl;
    return 0;
}
```

```
Enter size of input vector : 4
23 56 2 5

Sum : 86
```

Using Iterators

```cpp
#include <iostream>
#include <vector>
using namespace std;
int vectorSum2(vector<int> V) {
    typedef vector<int>::iterator Iterator;     // iterator type
    int sum = 0;
    for (Iterator p = V.begin(); p != V.end(); ++p)
        sum += *p;
    return sum;
}
int main(){
    vector<int> v;
    int size;
    cout<<"Enter size of input vector : ";
    cin>>size;
    int aux;
    for(int i=0;i<size;i++){
        cin>>aux;
        v.push_back(aux);
    }
    cout<<"\nSum : "<<vectorSum2(v)<<endl;
    return 0;
}
```

```
Enter size of input vector : 4
12 56 34 2

Sum : 104
```

SEQUENCE ADT

- The Sequence ADT generalizes the Vector and List ADTs
- Elements are accessed by:
  - Index, or
  - Position
- Methods and Usages?

```
class NodeSequence : public NodeList {
public:
  Iterator atIndex(int i) const;
  int indexOf(const Iterator& p) const;
};

                                          // get position from index
NodeSequence::Iterator NodeSequence::atIndex(int i) const {
  Iterator p = begin();
  for (int j = 0; j < i; j++) ++p;
  return p;
}

                                          // get index from position
int NodeSequence::indexOf(const Iterator& p) const {
  Iterator q = begin();
  int j = 0;
  while (q != p) {                        // until finding p
    ++q; ++j;                             // advance and count hops
  }
  return j;
}
```

(Doubly-linked list Implementation)

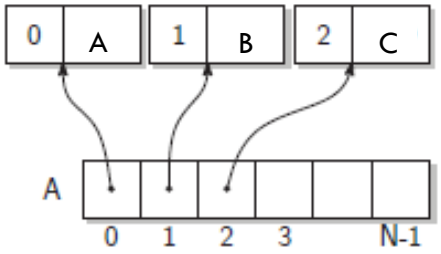
SEQUENCE ADT: ARRAY BASED

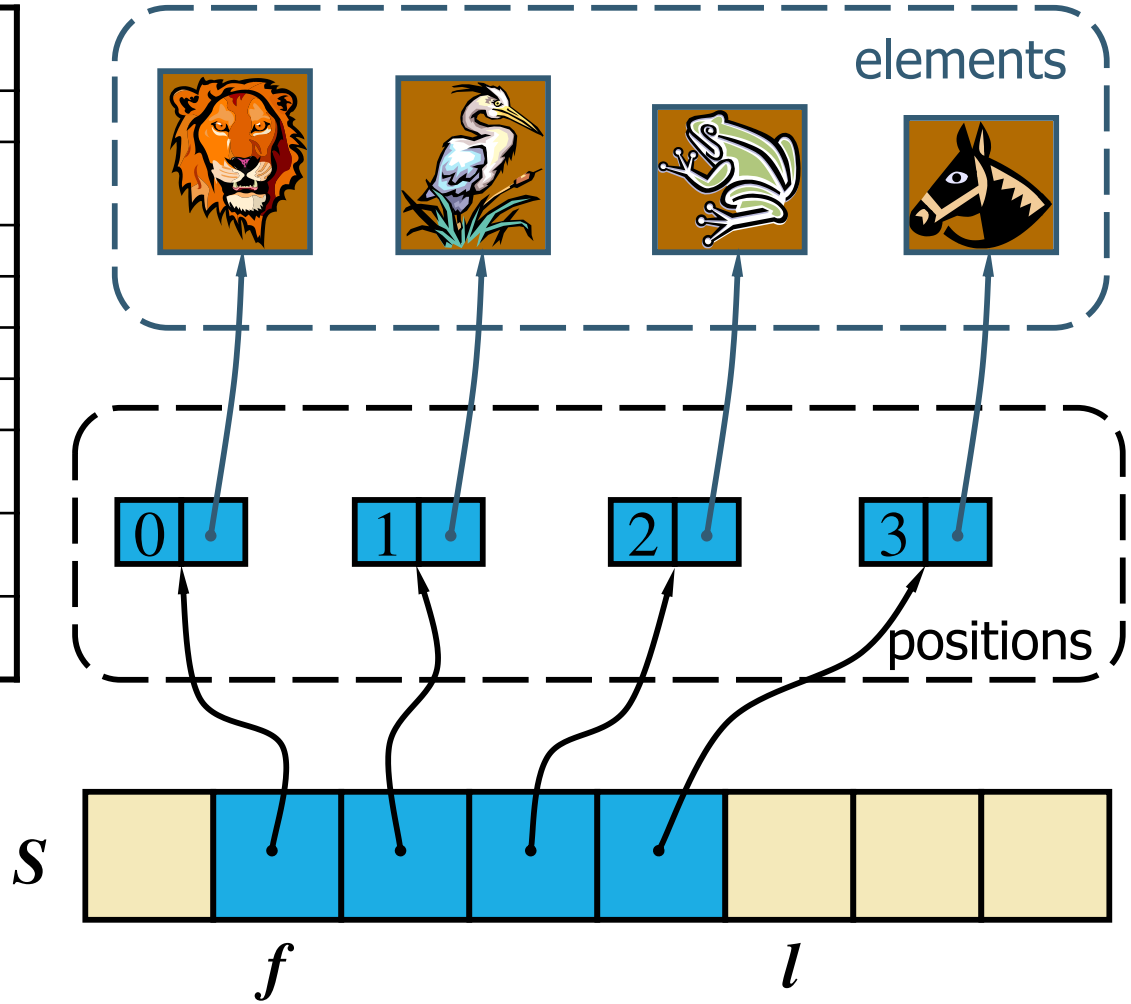
- We use a circular array storing positions.

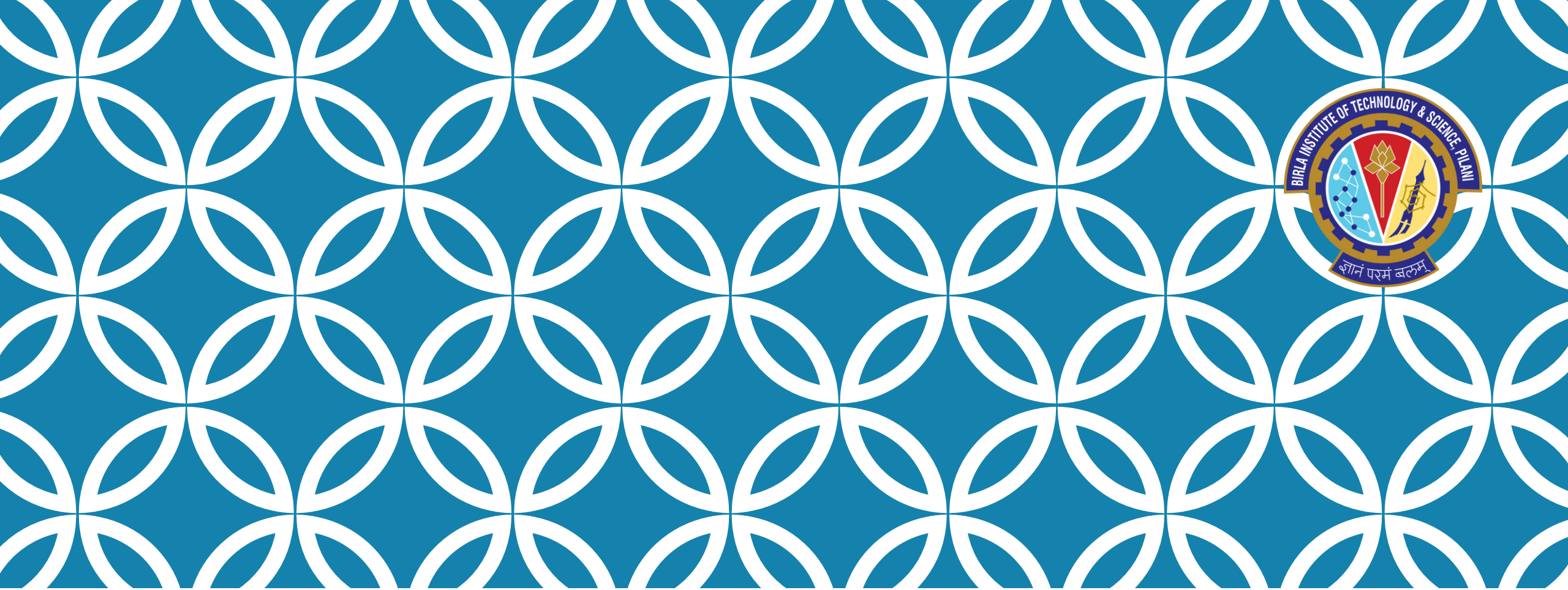
  A position object stores:
  - Element
  - Index

- Indices *f* and *l* keep track of first and last positions.

Operation	Array	List
size, empty	1	1
atIndex, indexOf, at	1	*n*
begin, end	1	1
set(p,e)	1	1
set(i,e)	1	*n*
insert(i,e), erase(i)	*n*	*n*
insertBack, eraseBack	1	1
insertFront, eraseFront	*n*	1
insert(p,e), erase(p)	*n*	1







BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) BUBBLE SORT, TREES

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

USAGE OF SEQUENCE ADT: BUBBLE SORT

- We will examine the usage of Sequence ADT and its implementation trade-offs using Bubble sort algorithm.

IMPLEMENTATION & ANALYSIS OF BUBBLE SORT

```
163 void bubbleSort1(NodeSequence& S) {
164     int n = S.size();
165     for (int i = 0; i < n; i++) {              // i-th pass
166         for (int j = 1; j < n-i; j++) {
167             NodeSequence::Iterator prec = S.atIndex(j-1);
168             NodeSequence::Iterator succ = S.atIndex(j);
169             if (*prec > *succ) {
170                 int tmp = *prec; *prec = *succ; *succ = tmp;
171             }
172         }
173     }
174 }
```

```
Enter size of input sequence : 6
5 2 6 7 3 9

Sorted sequence : 2 3 5 6 7 9
```

```
163 void bubbleSort2(NodeSequence& S) {     // bubble-sort by positions
164     int n = S.size();
165     for (int i = 0; i < n; i++) {           // i-th pass
166         NodeSequence::Iterator prec = S.begin(); // predecessor
167         for (int j = 1; j < n-i; j++) {
168             NodeSequence::Iterator succ = prec;
169             ++succ;          // successor
170             if (*prec > *succ) {        // swap if out of order
171                 int tmp = *prec; *prec = *succ; *succ = tmp;
172             }
173             ++prec;                     // advance predecessor
174         }
175     }
176 }
```

```
Enter size of input sequence : 6
5 2 6 7 3 9

Sorted sequence : 2 3 5 6 7 9
```

TREES: NON-LINEAR DATA STRUCTURES

- In computer science, what is a tree?

Formally, we define tree T to be a set of nodes storing elements in a parent-child relationship with the following properties:

- If T is nonempty, it has a special node, called the root of T, that has no parent.
- Each node v of T different from the root has a unique parent node w; every node with parent w is a child of w.

Applications?

TREE TERMINOLOGIES AND PROSPERITIES

Root: ???

Internal node: ???

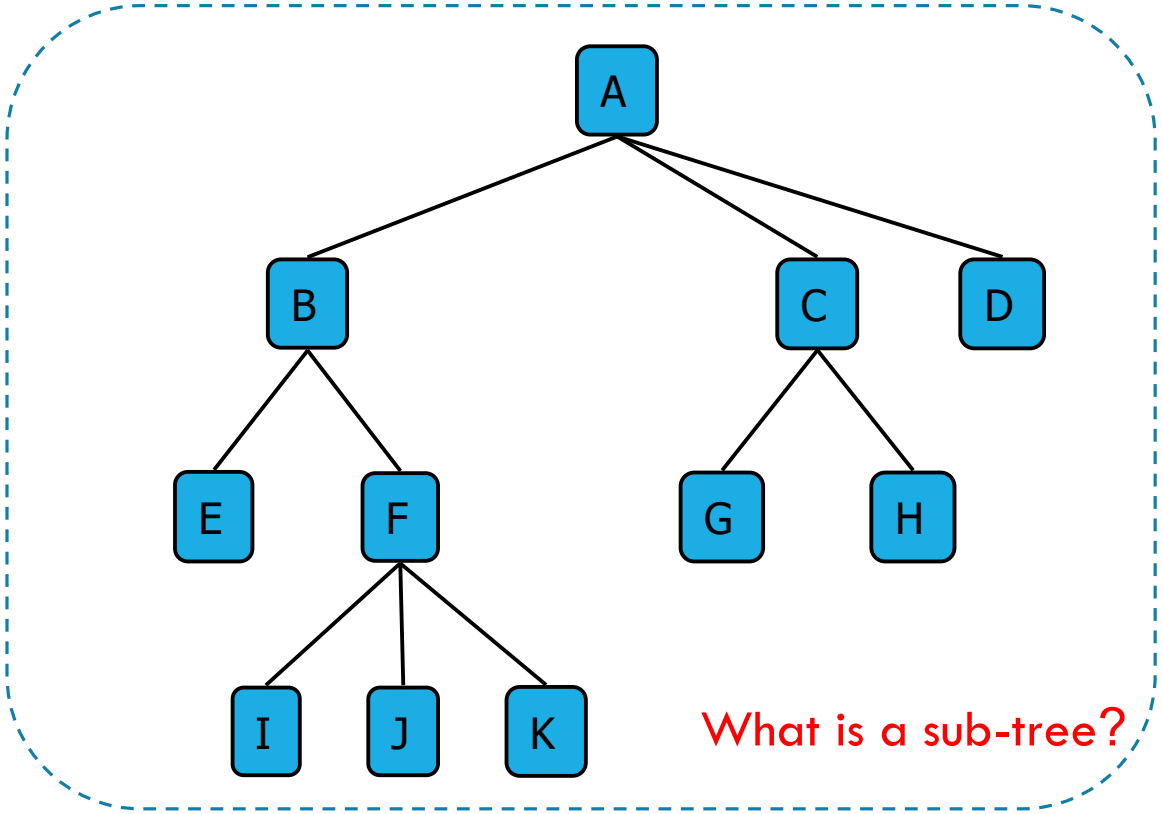
External node: ???

Ancestors of a node: parent, grandparent, grand-grandparent, etc.

Level of a node and Depth: ???

Height of a tree: maximum depth of any node (???)

Descendant of a node: child, grandchild, grand-grandchild, etc.



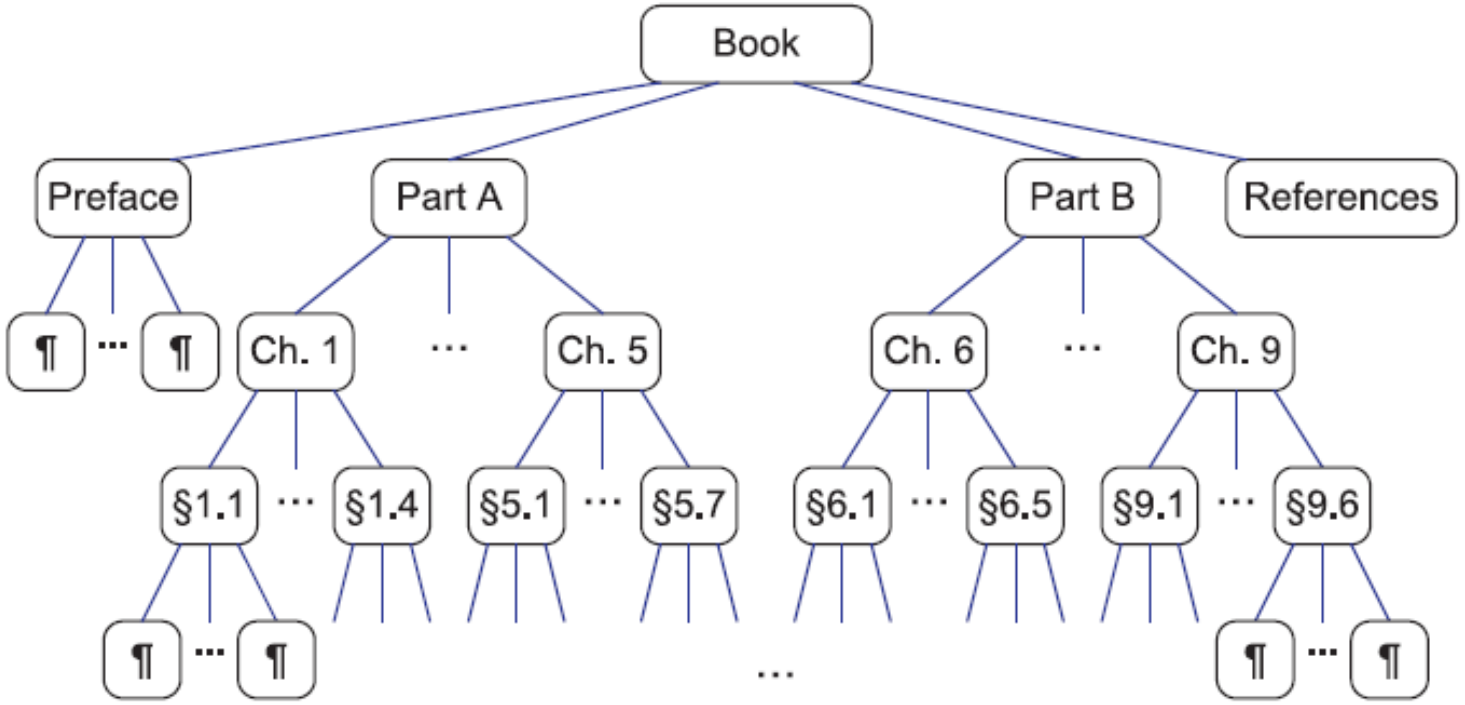
No of edges???

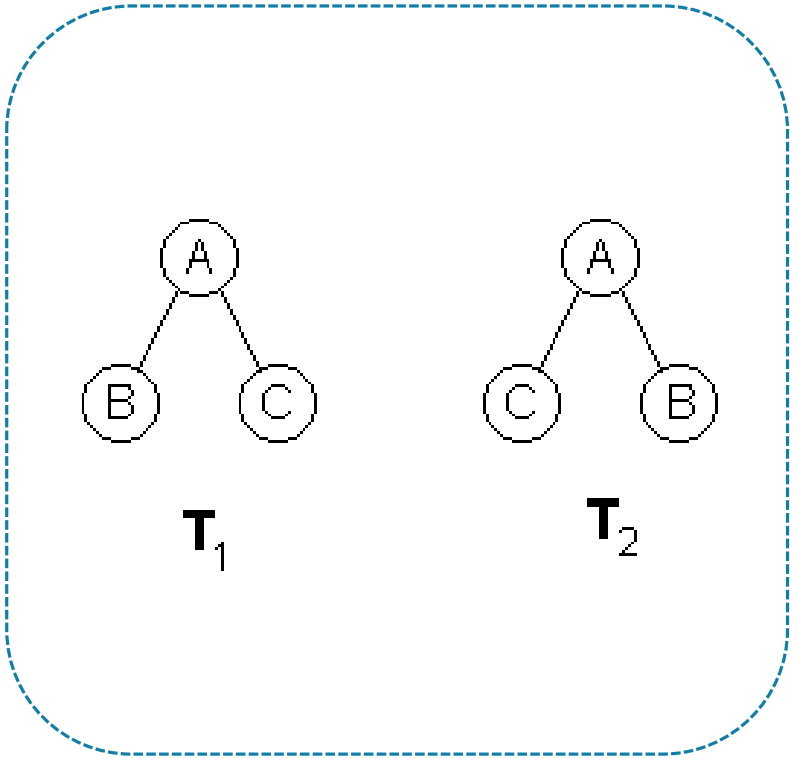
Degree of a node???

count of subtrees

ORDERED TREES

What are Ordered Trees?





TREE ADT

- Generic methods:
  - integer size()
  - boolean empty()
- Accessor methods:
  - position root()
  - list<position> positions()
- Position-based methods:
  - position p.parent()
  - list<position> p.children()
- Query methods:
  - boolean p.isRoot()
  - boolean p.isExternal()
- Additional update methods may be defined by data structures implementing the Tree ADT.

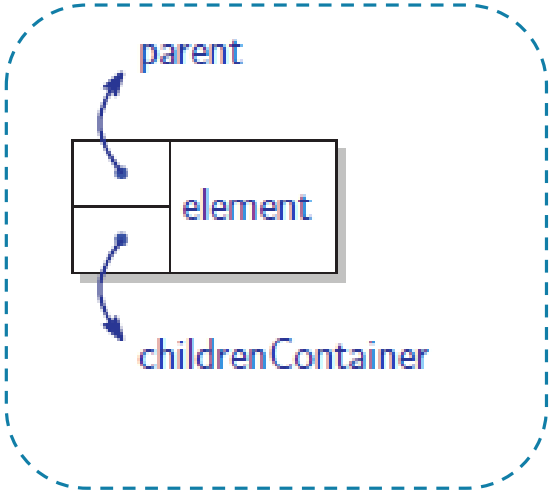
```
template <typename E>
class Position<E> {
public:
  E& operator*();
  Position parent() const;
  PositionList children() const;
  bool isRoot() const;
  bool isExternal() const;
};
```

(An informal interface for a position in a tree)

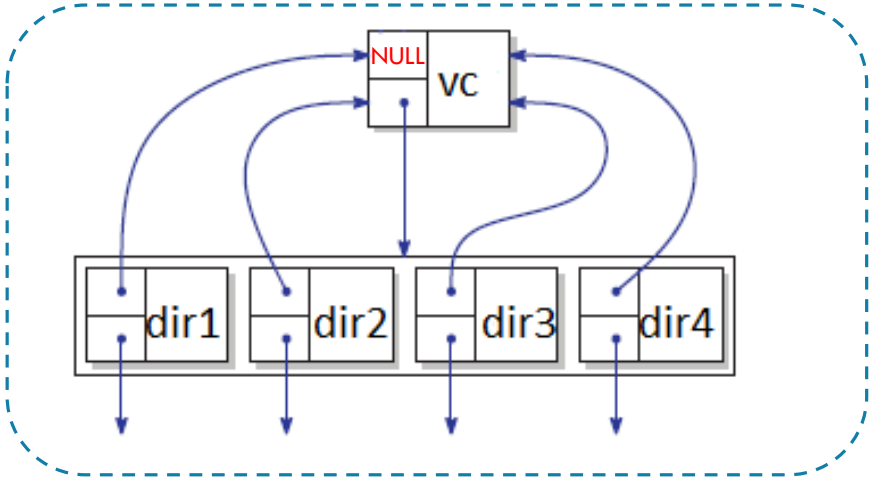
```
template <typename E>                  // base element type
class Tree<E> {
public:                                // public types
  class Position;                      // a node position
  class PositionList;                  // a list of positions
public:                                // public functions
  int size() const;                    // number of nodes
  bool empty() const;                  // is tree empty?
  Position root() const;               // get the root
  PositionList positions() const;      // get positions of all nodes
};
```

(An informal interface for the tree ADT)

A LINKED STRUCTURE FOR GENERAL TRESS



(The node structure)



(The portion of the data structure associated with root node and its children)

Operation	*Time*
isRoot, isExternal	$O(1)$
parent	$O(1)$
children(p)	$O(c_p)$
size, empty	$O(1)$
root	$O(1)$
positions	$O(n)$

(Running times of the functions of an *n*-node linked tree structure)

DEPTH AND HEIGHT OF A TREE

Algorithm depth(*T*, *p*):

Let us write it out using Recursion…

Complexity?

The height of a node p in a tree T is also defined recursively:

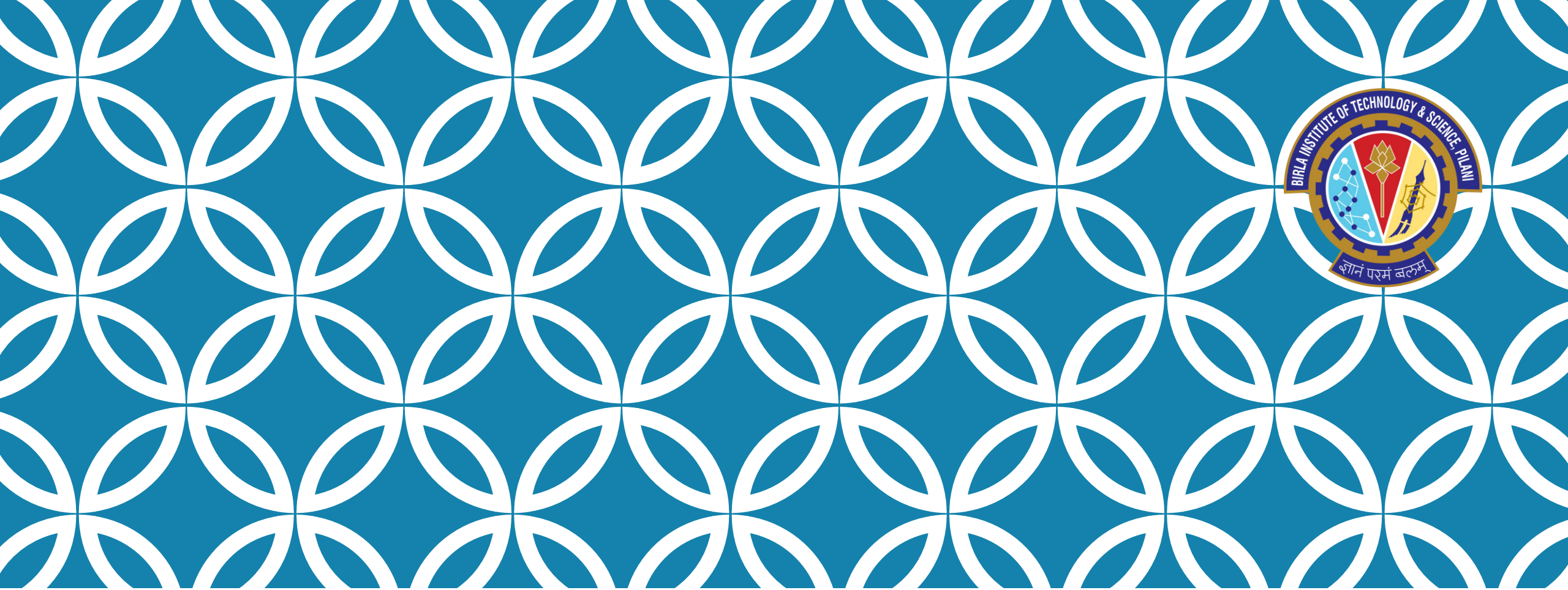
-If p is external, then the height of p is ???.

-Otherwise, the height of p is one plus the ??? height of a child of p

```
Algorithm height(T):
  h = 0
  for each p ∈ T.positions() do
    if p.isExternal() then
      h = max(h,depth(T, p))
  return h
```

(The height of a tree is equal to the maximum depth of its external nodes)

```
Algorithm height(T, p):
  if p.isExternal() then
    return 0
  else
    h = 0
    for each q ∈ p.children()
    do h=max(h,height(T,q))
    return 1+h
```



BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) TREE ADT CONTINUED…

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

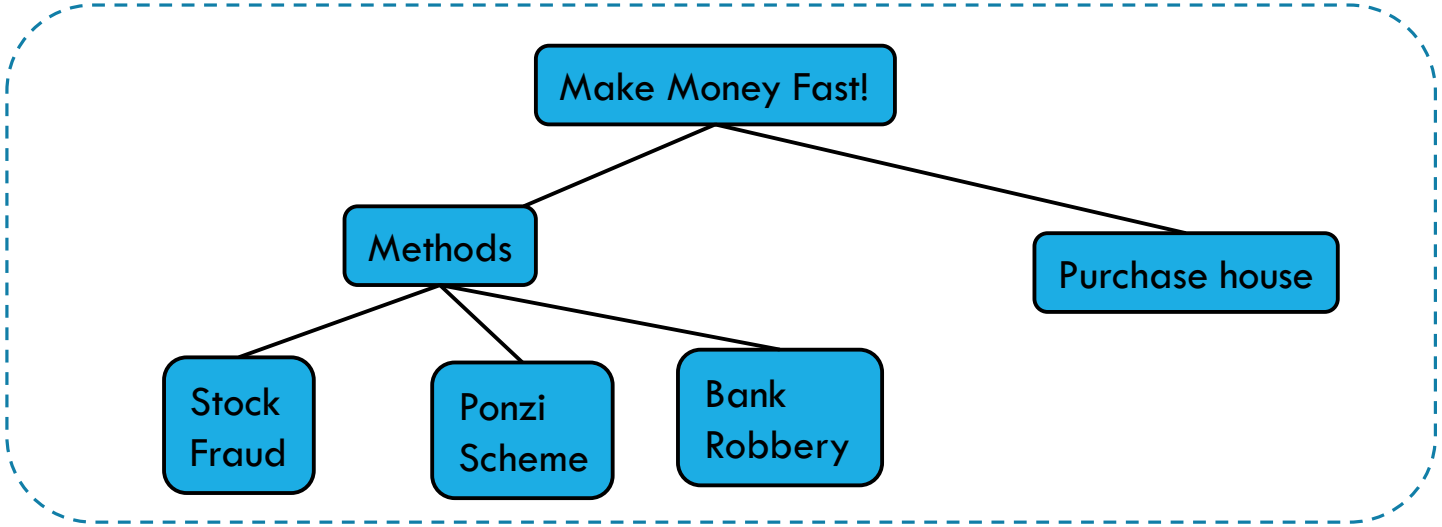
TREE TRAVERSAL ALGORITHMS

```
void iterativeLevelOrder(TreeNode* root)
{
    if (root == NULL)
        return;

    queue<TreeNode*> treeQueue;
    treeQueue.push(root);

    while (treeQueue.empty() == false)
    {
        TreeNode* currNode = treeQueue.front();
        treeQueue.pop();
        cout << currNode->data << " ";

        if (currNode->left != NULL)
            treeQueue.push(currNode->left);

        if (currNode->right != NULL)
            treeQueue.push(currNode->right);
    }
}
```

```
void printLevelOrder(node* root)
{
    int h = height(root);
    int i;
    for (i = 0; i < h; i++)
        print_current_level(root, i);
}
void print_current_level(Node* root, int level_no)  {
    if (!root) return;
    if (level_no == 0) {
       printf("%d -> ", root->value);
    }
    else {
        print_current_level(root->left, level_no - 1);
        print_current_level(root->right, level_no - 1);
    }
}
```

TREE TRAVERSAL: DEPTH-FIRST (PREORDER)

- In a preorder traversal, a node is visited before its descendants.
- Applications: print a structured document, get the prefix expression on an expression tree.




```
20  void printPreorder(struct Node* node)
21  {
22      if (node == NULL)
23          return;
24      cout << node->data << " ";
25
26      printPreorder(node->left);
27
28      printPreorder(node->right);
29  }
```

```
Preorder traversal of binary tree is
1 2 4 5 3
```

Algorithm *preOrder* (T, p) {

}

Lab 8 after mid sem

```
procedure iterativePreorder(node)
    if node = null
        return
    stack ← empty stack
    stack.push(node)
    while not stack.isEmpty()
        node ← stack.pop()
        visit(node)
        // right child is pushed first
        if node.right ≠ null
            stack.push(node.right)
        if node.left ≠ null
            stack.push(node.left)
```

iterative

POSTORDER TRAVERSAL

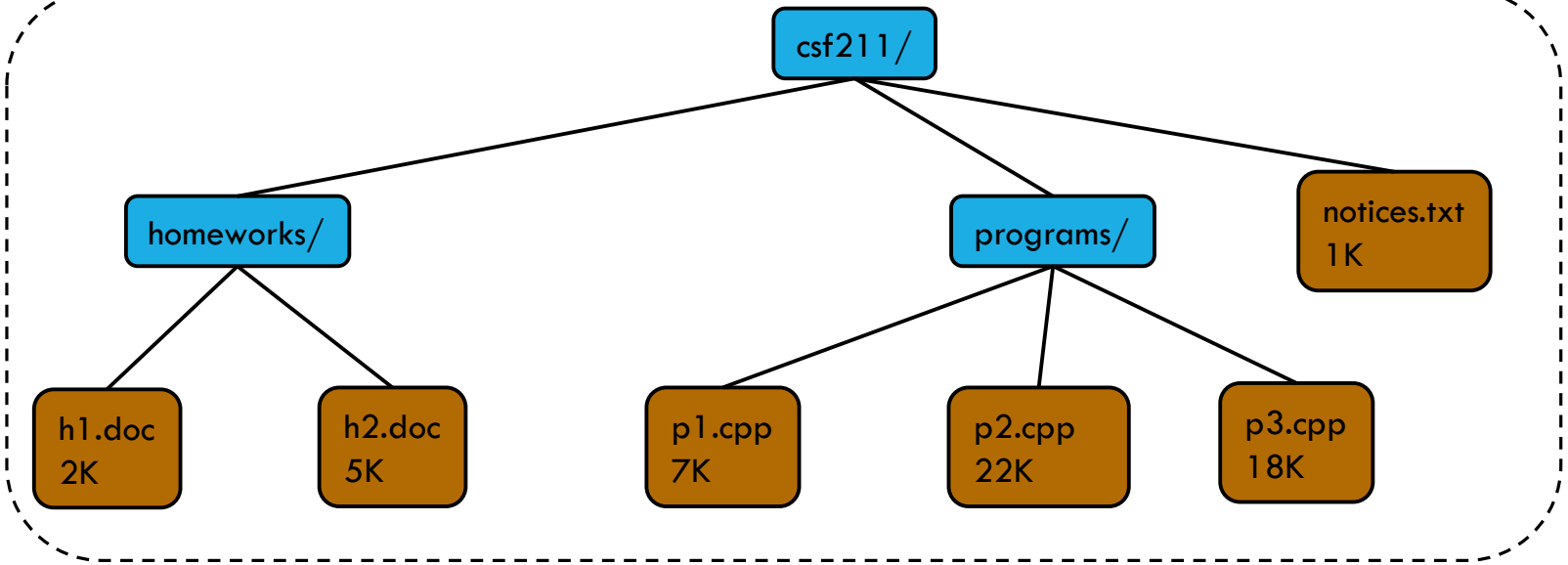
- In a postorder traversal, a node is visited after its descendants.
- Application: compute space used by files in a directory and its subdirectories, delete the tree, compute postfix expression…

```
23  void printPostorder(struct Node* node)
24  {
25      if (node == NULL)
26          return;
27
28      printPostorder(node->left);
29
30      printPostorder(node->right);
31
32      cout << node->data << " ";
33  }
```

Lab 8 after mid sem

```
Postorder traversal of binary tree is
4 5 2 3 1
```

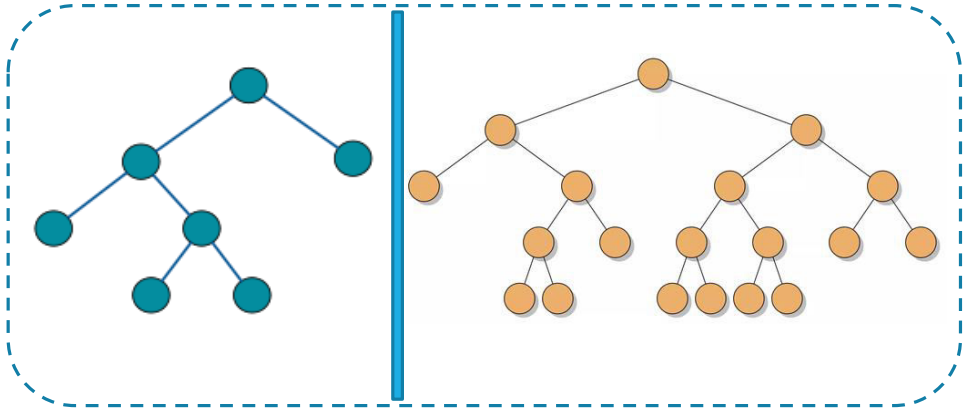
```
Algorithm postOrder(v){
        for each child w of v
                postOrder (w);
        visit(v);
}
```



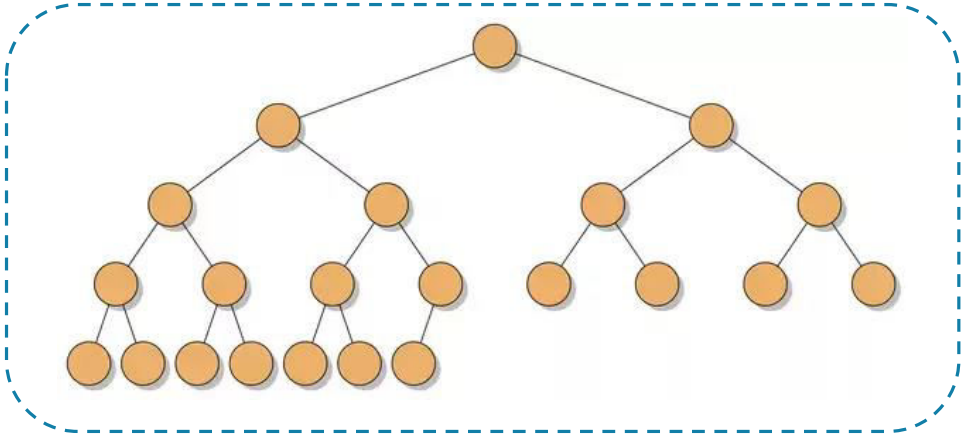
DEFINITIONS: BINARY TREE

- What is a binary tree?
- What is a full binary tree or proper binary tree?
  - All internal nodes have exactly ??? children
- What is a complete binary tree?
  - Every level except the last is filled. Last from left.
- Are the children of a binary tree ordered?

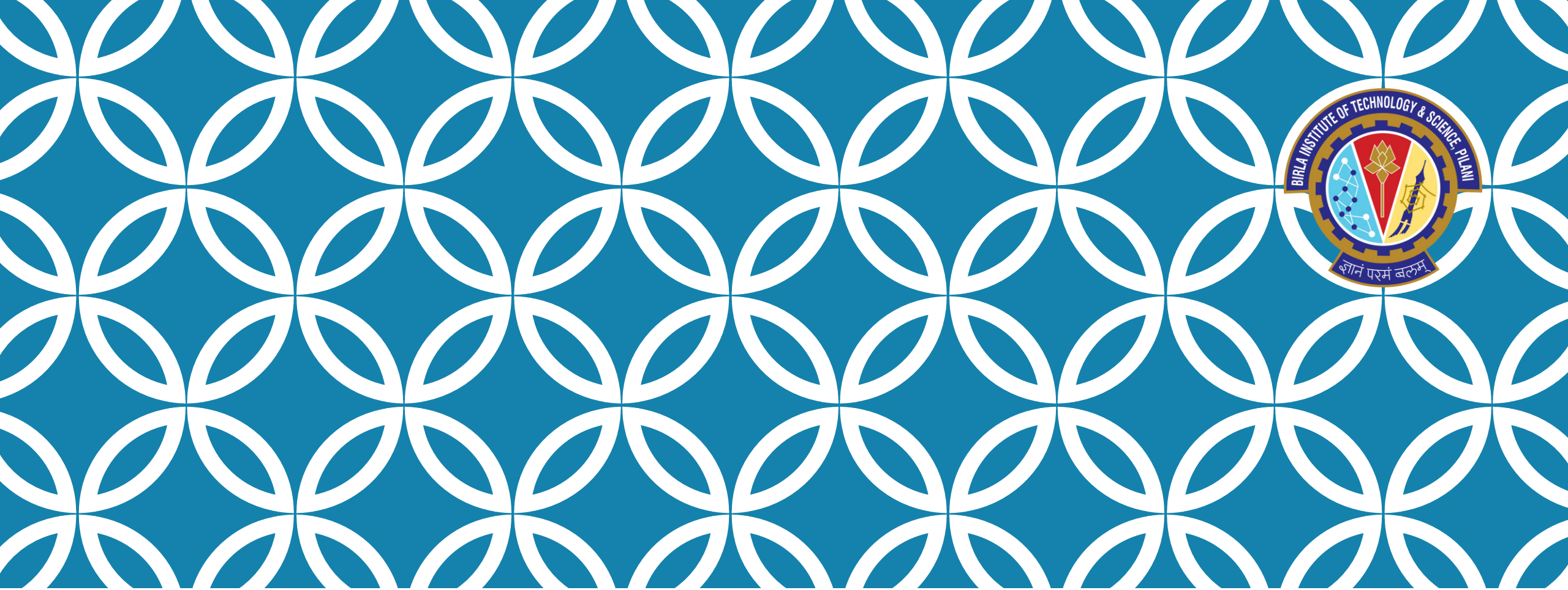
- Applications:
  - Problem representation (arithmetic expressions, decision trees)
  - Efficient algorithmic solutions (searching becomes faster, priority queues via heaps)
  - …



(what are these?)



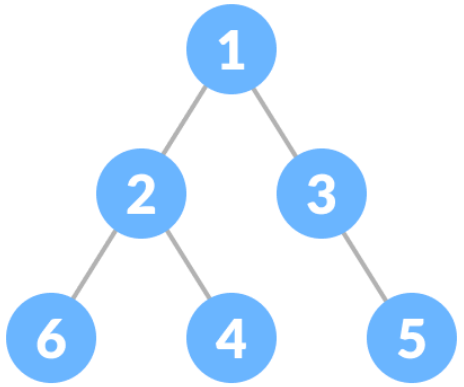
(what is this?)



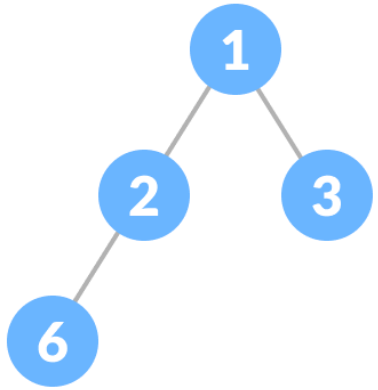
BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) TREE ADT CONTINUED…

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

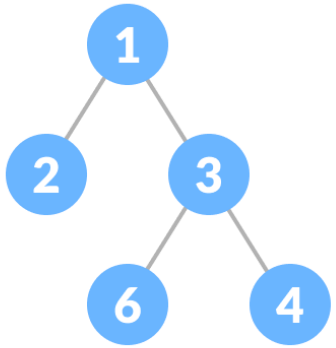
FULL VS COMPLETE: RECAP



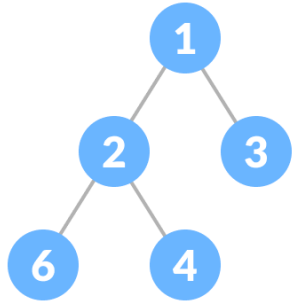
Full (X), or complete (X)



Full (X), or complete (Y)



Full (Y), or complete (N)



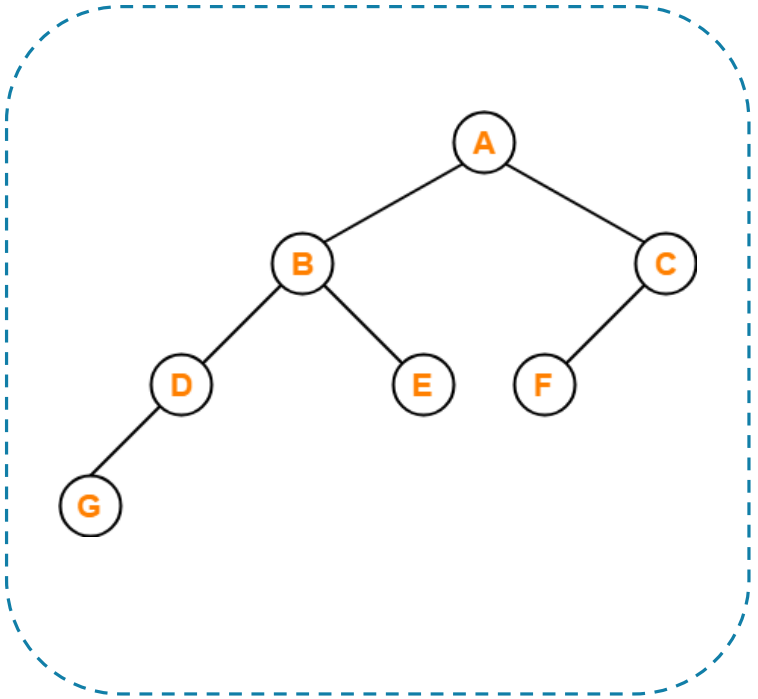
Full (Y), or complete (Y)

PROPERTIES OF A BINARY TREE

Notation:

n: number of nodes, ***e:*** number of external nodes, ***i:*** number of internal nodes, and ***h:*** height

- Minimum number of nodes in a binary tree with height h = h + 1
- Maximum number of nodes in a binary tree with height h = 2^{h+1} - 1
  - Let us draw the tree for h = 2.
- Total number of leaf nodes in a binary tree = total number of nodes with 2 children + 1
- Maximum number of nodes at any level 'l' in a binary tree = 2^l.
  - Let us draw it for level 2.

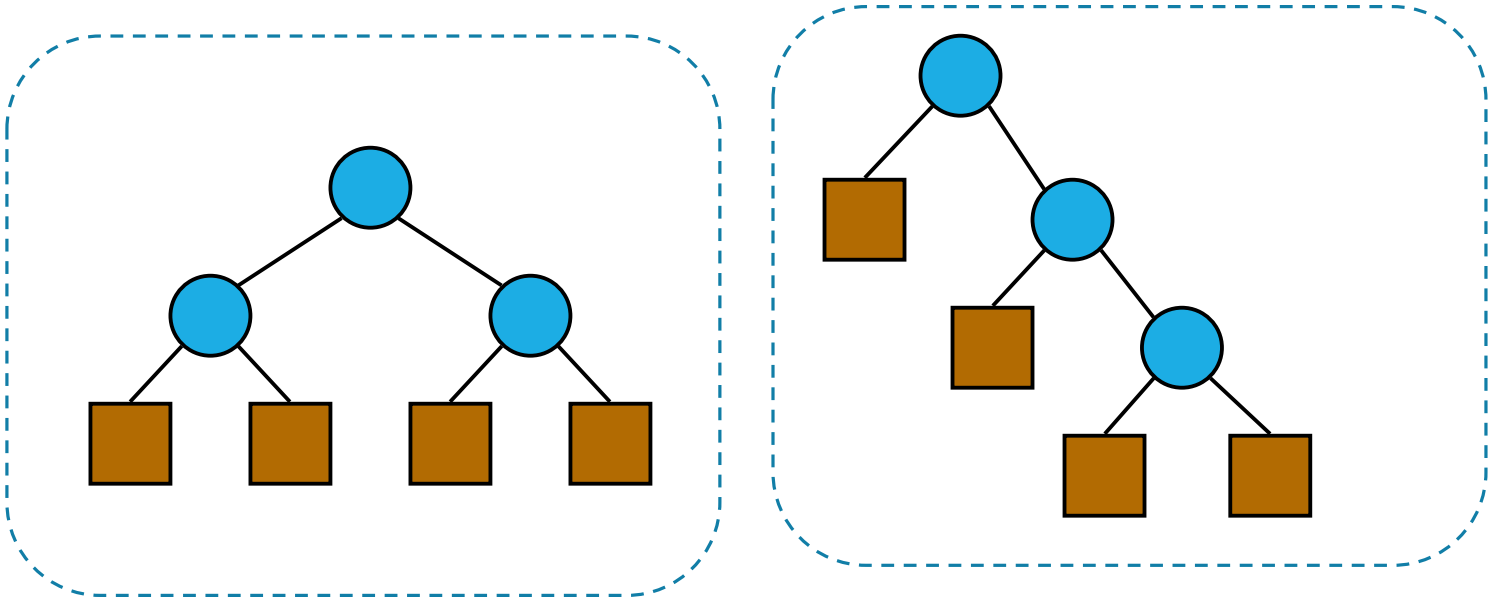


PROPERTIES OF A PROPER BINARY TREE

Notation:

n: number of nodes, ***e:*** number of external nodes, ***i:*** number of internal nodes, and ***h:*** height

- Properties:
  - $e = i + 1$
  - $n = 2e - 1$
  - $h \leq i$
  - $h \leq (n - 1)/2$
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1$

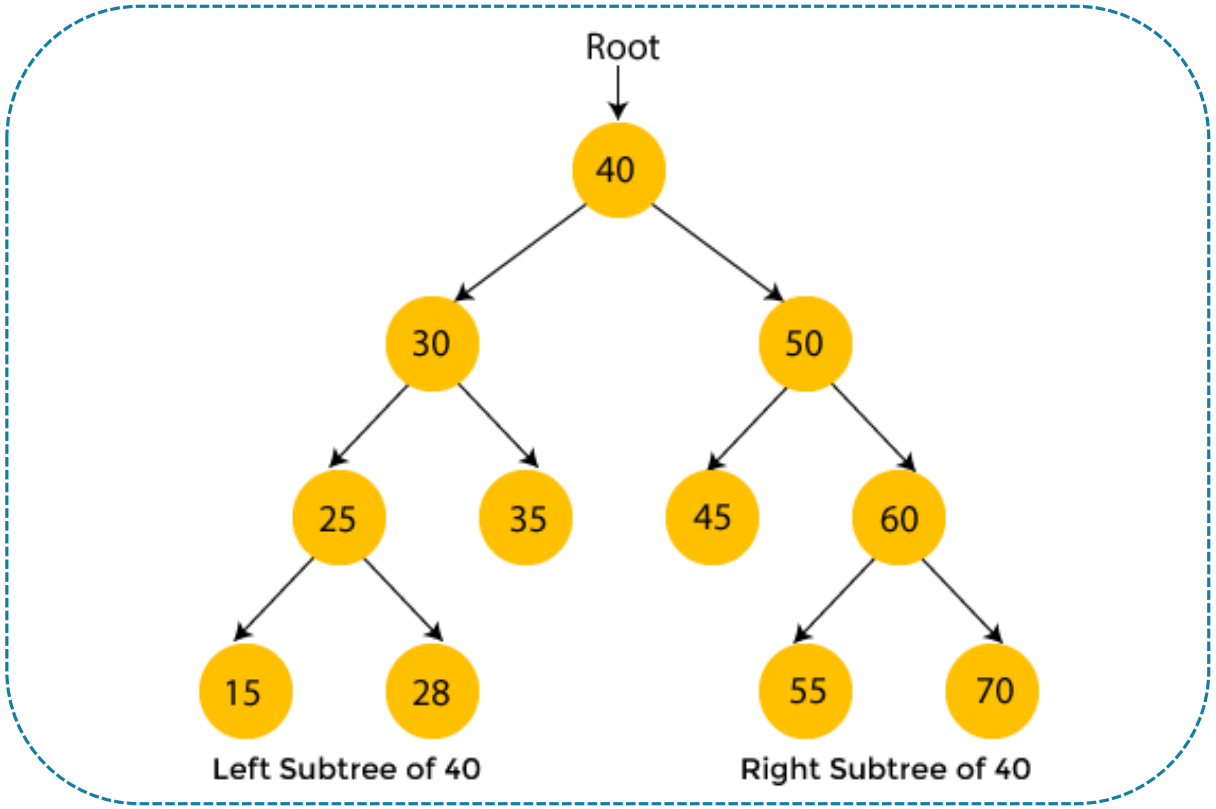


INORDER TRAVERSAL

- In an inorder traversal a node is visited after its left subtree and before its right subtree

Application: draw a binary tree

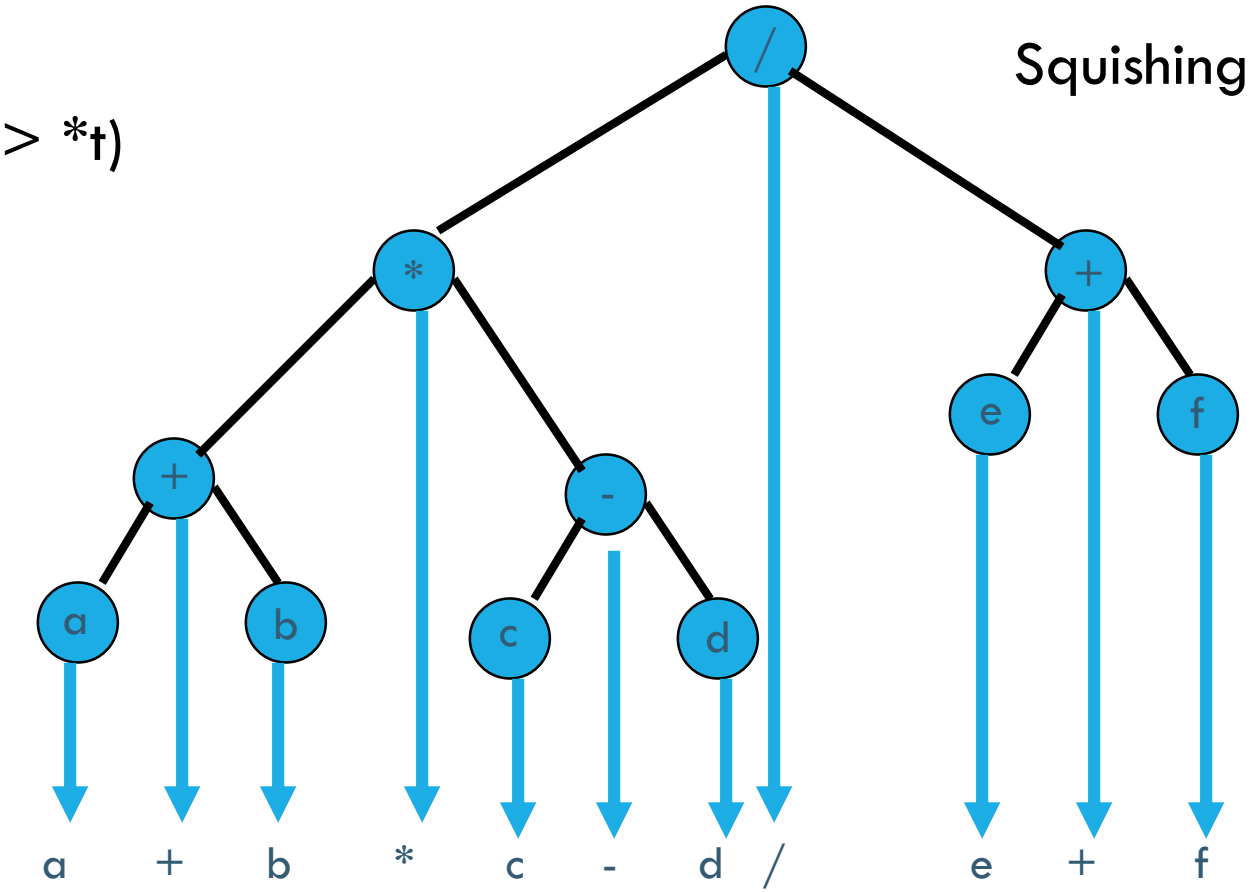
```
Algorithm inOrder(v)
    if ¬ v.isExternal()
        inOrder(v.left())
    visit(v)
    if ¬ v.isExternal()
        inOrder(v.right())
```



BINARY TREE TRAVERSAL: INORDER

```
template <class T>
void inOrder(binaryTreeNode<T> *t)
{
   if (t != NULL)
   {
      inOrder(t->leftChild);
      visit(t);
      inOrder(t->rightChild);
   }
}
```

Lab: after midterm (Lab no:8)



Squishing

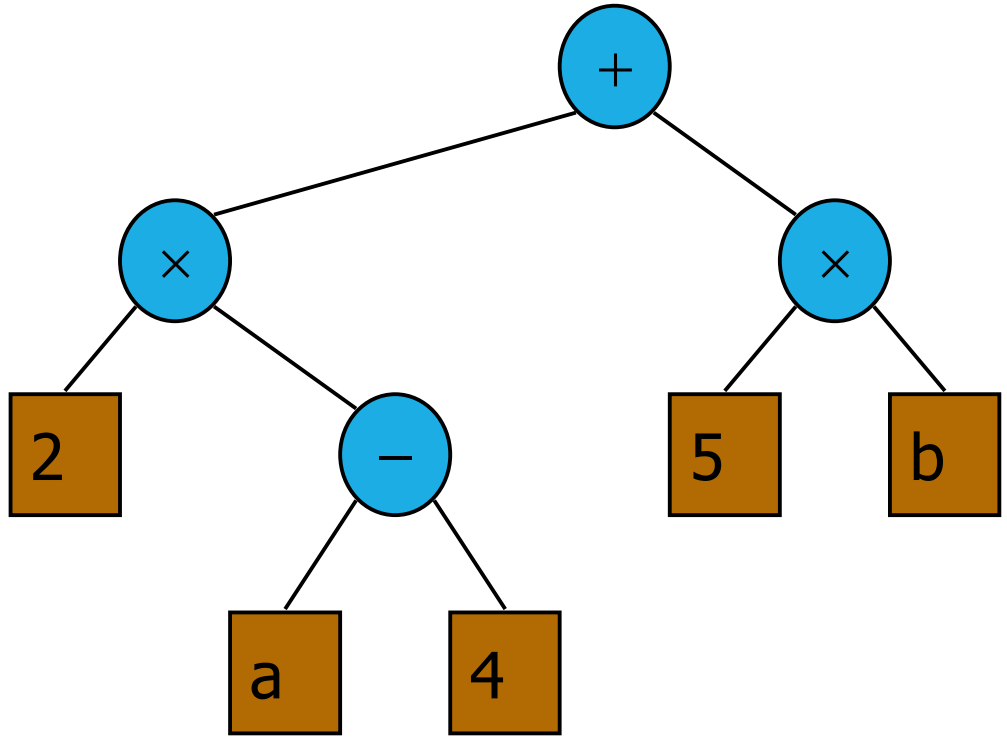
Gives infix form of expression!

PRINT ARITHMETIC EXPRESSION

Specialization of an inorder traversal

- print operand or operator when visiting node
- print "(" before traversing left subtree
- print ")" after traversing right subtree

```
Algorithm printExpression(v)
    if ¬v.isExternal()
        print("(")
        inOrder(v.left())
    print(v.element())
    if ¬v.isExternal()
        inOrder(v.right())
        print (")")
```

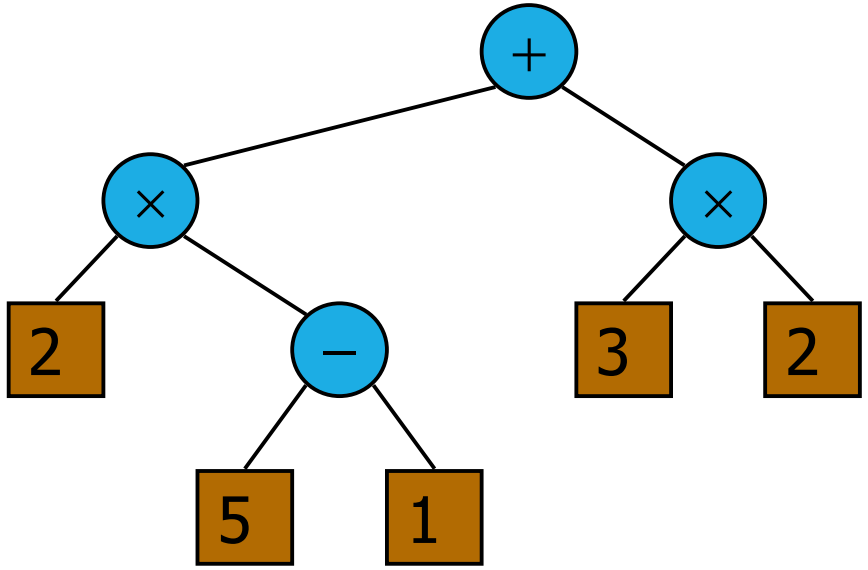


Let us write it out…

EVALUATE ARITHMETIC EXPRESSION

Specialization of a postorder traversal

- recursive method returning the value of a subtree
- when visiting an internal node, combine the values of the subtrees

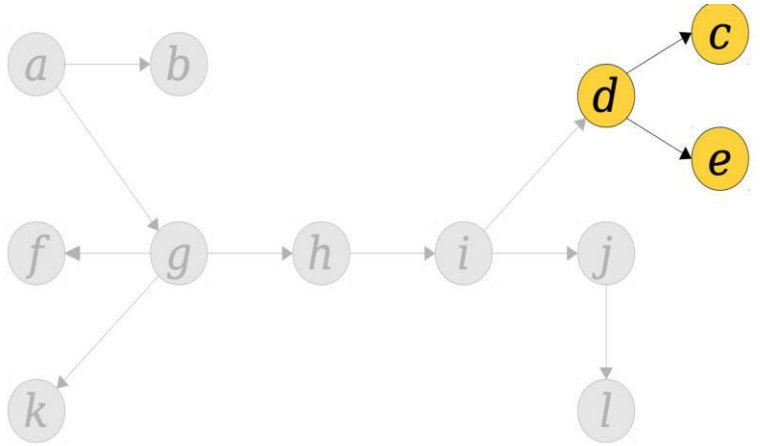



```
Algorithm evalExpr(v)
    if v.isExternal()
        return v.element()
    else
        x ← evalExpr(v.left())
        y ← evalExpr(v.right())
        ◊ ← operator stored at v
        return x ◊ y
```

EULER TOUR TRAVERSAL

- Generic traversal of a binary tree
- We can **unify** the tree-traversal algorithms (in-order, pre-order, and post-order) into a single framework by relaxing the requirement that each node be visited exactly once.
- Walk around the tree and visit each node three times:
  - on the left (preorder)
  - from below (inorder)
  - on the right (postorder)

+: It allows for more general kinds of algorithms to be expressed easily.



a b a g h i **d c d e d** i j l j i h g f g k g a

```
template <typename E, typename R>              // do the tour
int EulerTour<E, R>::eulerTour(const Position& p) const {
  Result r = initResult();
  if (p.isExternal()) {                        // external node
    visitExternal(p, r);
  }
  else {                                       // internal node
    visitLeft(p, r);
    r.leftResult = eulerTour(p.left());        // recurse on left
    visitBelow(p, r);
    r.rightResult = eulerTour(p.right());      // recurse on right
    visitRight(p, r);
  }
  return result(r);
}
```

Applications: finding no. of descendants, level of each node, lowest common ancestor, etc.

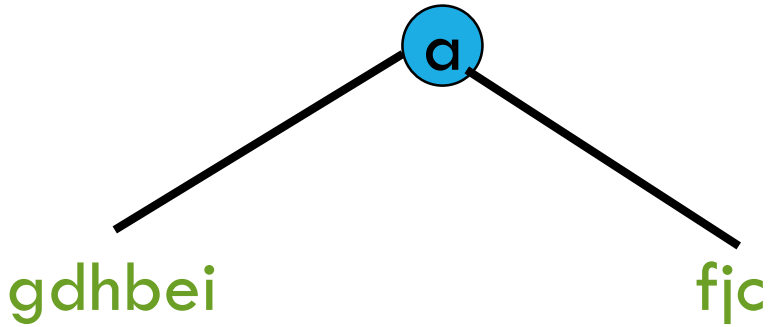
BINARY TREE CONSTRUCTION FROM TRAVERSAL ORDER

Can you construct the binary tree, given two traversal sequences?

preorder = ab

postorder = ba

Try:
Postorder: DEBFCA
Inorder: DBEAFC

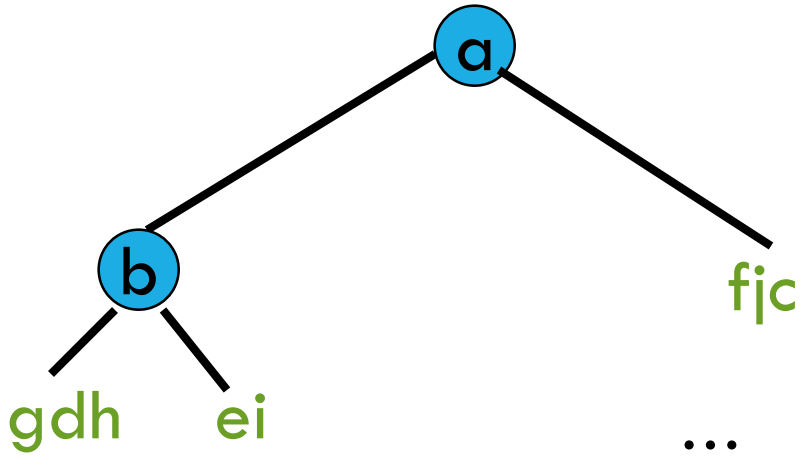


do they uniquely define a binary tree?

inorder = g d h b e i a f j c

preorder = a b d g h e i c f j

preorder = b d g h e i c f j



Similarly, Scan postorder from right to left using inorder.

BINARY TREE ADT

p.left(): Return the left child of p; an error condition occurs if p is an external node.

p.right(): Return the right child of p; an error condition occurs if p is an external node.

p.parent(): Return the parent of p; an error occurs if p is the root.

p.isRoot(): Return true if p is the root and false otherwise.

p.isExternal(): Return true if p is external and false otherwise.

size(): Return the number of nodes in the tree.

empty(): Return true if the tree is empty and false otherwise.

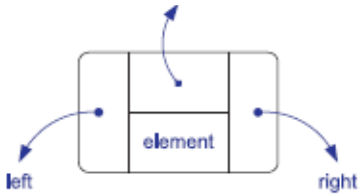
root(): Return a position for the tree's root; an error occurs if the tree is empty.

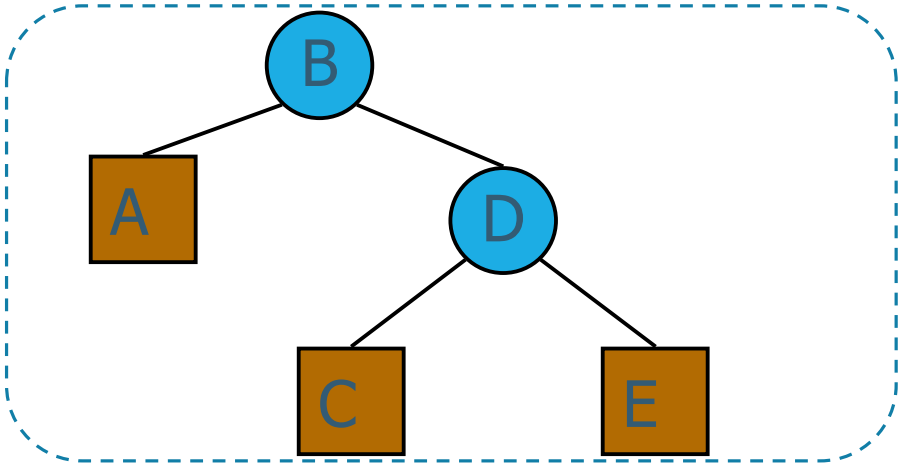
positions(): Return a position list of all the nodes of the tree.

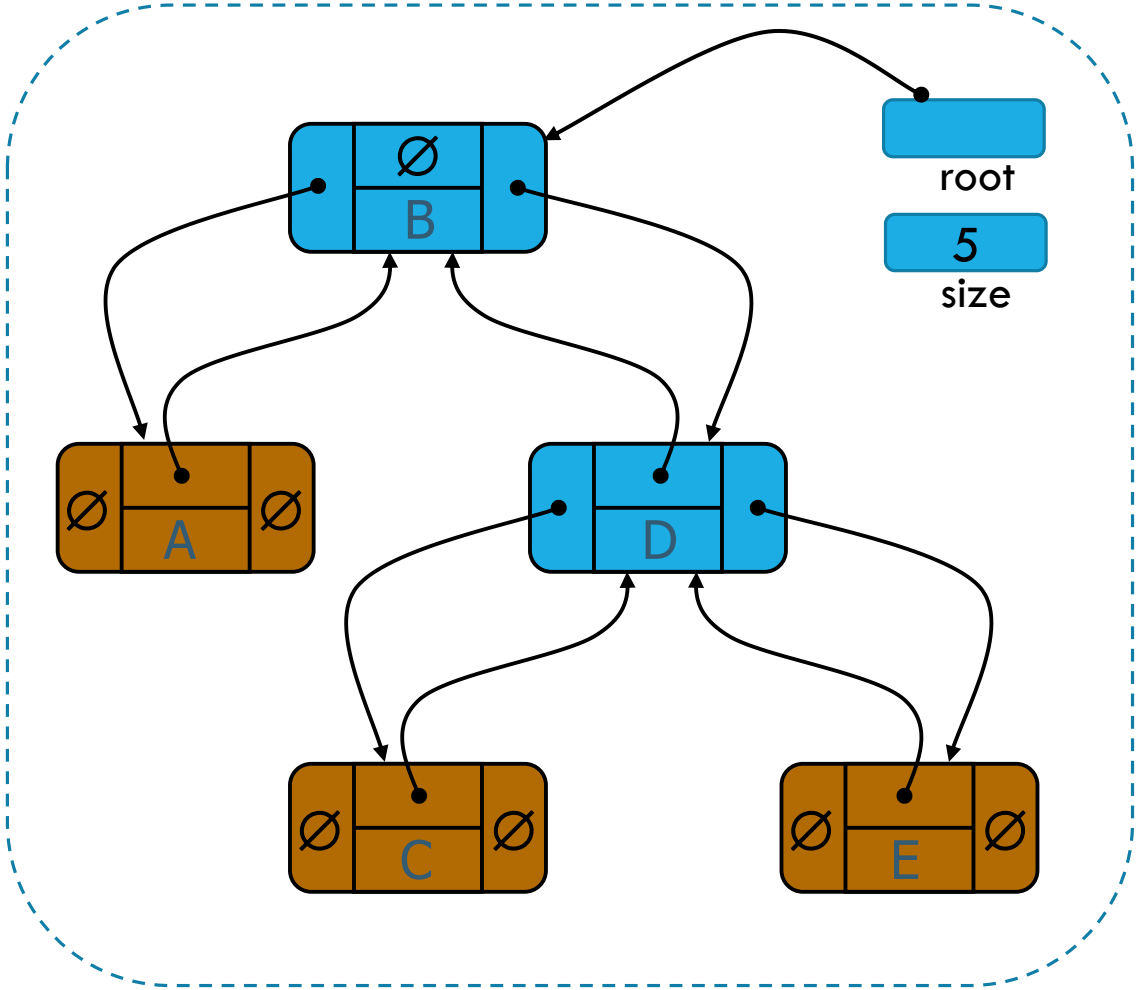
```
template <typename E>
class Position<E> {
public:
  E& operator*();
  Position left() const;
  Position right() const;
  Position parent() const;
  bool isRoot() const;
  bool isExternal() const;
};

template <typename E>
class BinaryTree<E> {
public:
  class Position;
  class PositionList;
public:
  int size() const;
  bool empty() const;
  Position root() const;
  PositionList positions() const;
};
```

LINKED STRUCTURE FOR BINARY TREES

```
struct Node{
    Node *left;
    Node *right;
    Node *par;
    int data;
    Node() : data(), par(NULL), left(NULL), right(NULL) { }
};
```

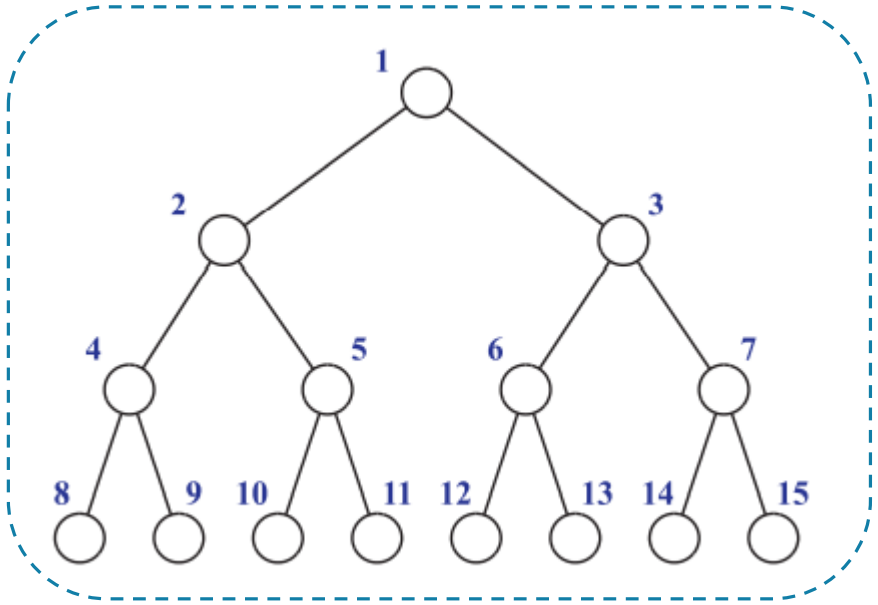




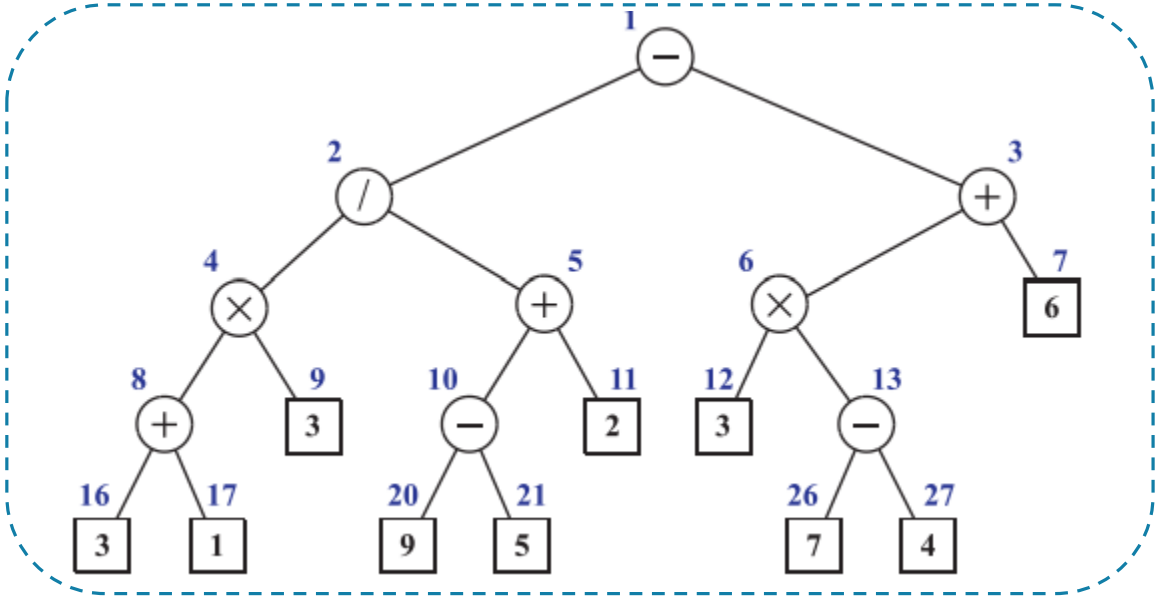


A VECTOR-BASED IMPLEMENTATION OF BINARY TREE

- A simple structure for representing a binary tree T is based on a way of numbering the nodes of T.
- If v is the root of T, then f(v) = 1; If v is the left child of node u, then f(v) = 2f(u); If v is the right child of node u, then f(v) = 2f(u) +1 ➔ function f is called level numbering function.

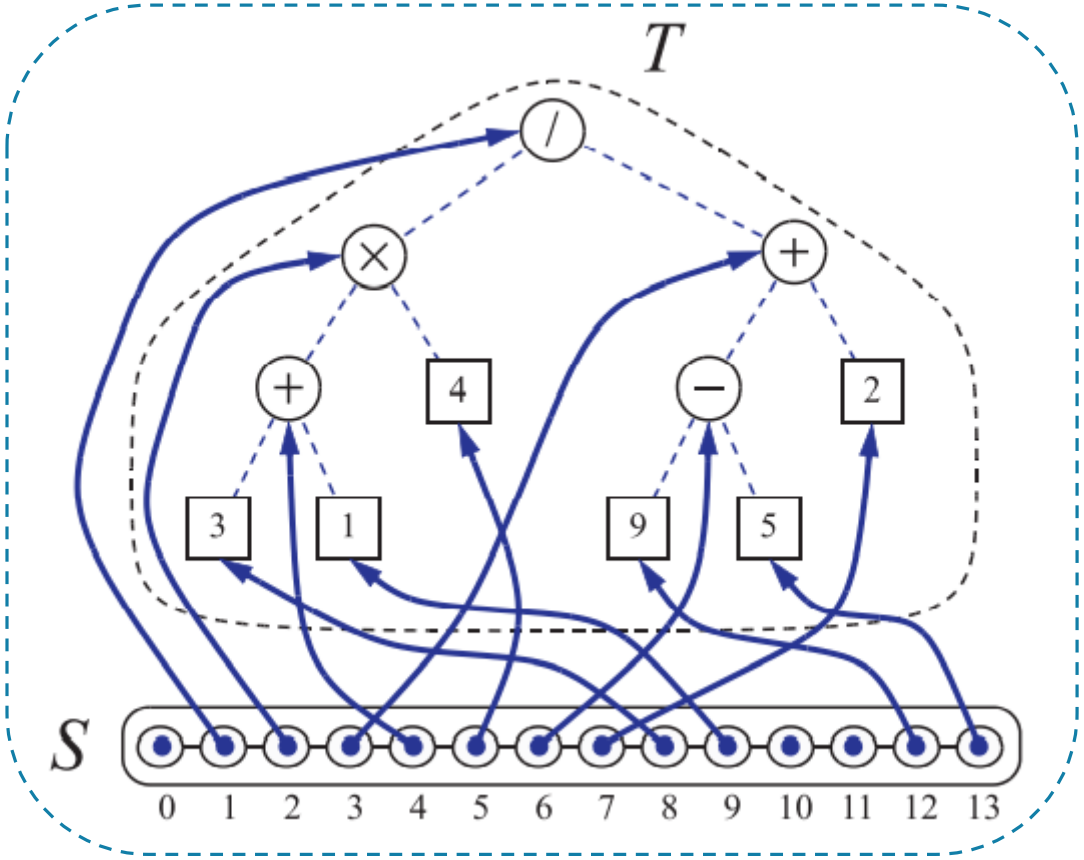


(General scheme)



(An Example Expression tree)

CONTINUED...



(Example binary tree using a vector)

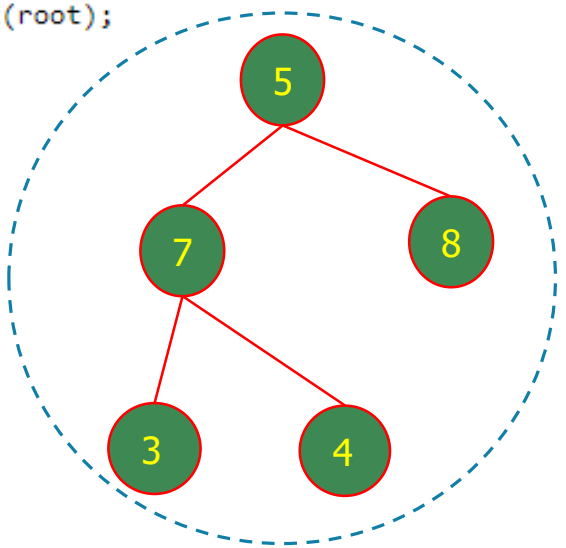
Operation	*Time*
left, right, parent, isExternal, isRoot	$O(1)$
size, empty	$O(1)$
root	$O(1)$
expandExternal, removeAboveExternal	$O(1)$
positions	$O(n)$

The vector implementation of a binary tree is a fast and easy way of realizing the binary-tree ADT, but it can be very space inefficient if the height of the tree is large. ➔ $O(2^n)$, where 'n' is no. of nodes in T.

BINARY TREE IMPLEMENTATION USING LINKED STRUCT

```
77  void BinaryTree::inorder(Node *ptr){
78      if(!ptr) return;
79      inorder(ptr->left);
80      cout<<" "<<ptr->data;
81      inorder(ptr->right);
82  }
83  void BinaryTree::postorder(Node *ptr){
84      if(!ptr) return;
85      postorder(ptr->left);
86      postorder(ptr->right);
87      cout<<" "<<ptr->data;
88  }
89  void BinaryTree::preorder(Node *ptr){
90      if(!ptr) return;
91      cout<<ptr->data<<" ";
92      preorder(ptr->left);
93      preorder(ptr->right);
94  }
```

```
98  Node* BinaryTree::createTree(vector<int> &v,Node *root,
99                               Node *parent,int i){
100     n = v.size();
101     if(i<v.size()){
102         Node *temp = new Node;
103         temp->data = v[i];
104         temp->par = parent;
105         root = temp;
106         root->left = createTree(v,root->left,root,2*i+1);
107         root->right = createTree(v,root->right,root,2*i+2);
108         all_nodes.insert(root);
109     }
110     main_root = root;
111     return root;
112 }
```




```
Enter size of input array : 6
Enter array : 5 7 8 3 4 9
1
Size : 6
2
Tree is not empty
3
Inorder traversal :  3 7 4 5 9 8
4
Preorder traversal : 5 7 3 4 8 9
5
Postorder traversal :  3 4 7 9 8 5
6
Height by height1 : 2
7
Height by height2 : 2
```

```
Enter size of input array : 5
Enter array : 5 7 8 3 4
1
Size : 5
2
Tree is not empty
3
Inorder traversal :  3 7 4 5 8
4
Preorder traversal : 5 7 3 4 8
5
Postorder traversal :  3 4 7 8 5
6
Height by height1 : 2
7
Height by height2 : 2
```