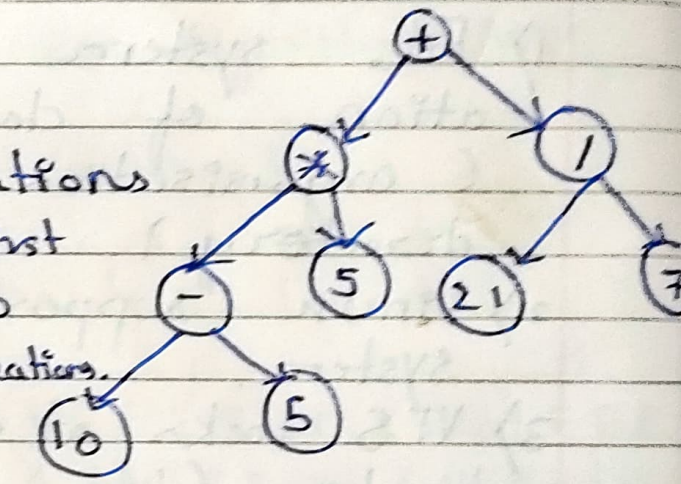
May 14

ADS Unit 6.

Q.1 Evaluation of expression tree.

logic :

do the operations in bracket, first preference to leaf nodes operations.



→ ((10 - 5) * 5) + (21 / 7)

→ 25 + 3

→ 28

Let's get into the processing part

```
public static double PROCESS
  ( string op , double x , double y)
 {
   if ( op. equals ( " + " ) )
        {
          return  x + y ;
        }
```

basic struct is

```
class Node{
    String val
    Node Left, Right
    Node (String val){ this.val = val;
                       this.Left = this.Right = null; }
}
else if (op.equals("-")) {
        return x - y;
    }
else if (op.equals("*")) {
        return x * y;
    }
else if (op.equal("/")) {
        return x / y;
    }

    return 0;
}

public static double evu (Node root){
{
  if (root == null) {
      return 0;
  }

  if (isLeaf(root)) {
      return Double.valueOf(root.val);
  }

  // recurssively checking
  double x = evu(root.left);
  double y = evu(root.Rght);

  return PROCESS(root.val, x, y);
}
```

Page No.:
Date:
YOUVA

Hard coded method.

```
public int evu(Node node) {
    if (node.value.equals("/")) {
        return evu(node.Left) / evu(node.
                                    Right);
    }
    else if (node.value.equals("*")) {
        return evu(node.Left) * evu(node
                                    .Right);
    }
    else if (node.value.equals("-") {
        return evu(node.Left) - evu(node.
                                    Right);
    }
    else if (node.value.equals("+") {
        return evu(node.Left) + evu(node
                                    .Right);
    }
    else {
        return Integer.ValueOf(node.val
    }
}
```

Trick : make a complete binary tree (heap) and traverse in Breadth first order traversal to get Max / Min heap.

q2 Heap and priority queue.

a] Heap :-

→ A heap is a nearly complete binary tree with the following two properties :

1) All levels are full except the last one.

2) For Max heap : parent(x) ≥ X
   For Min heap : parent(x) ≤ X

→ In An array :

root : A[1]
left child : A[i] = A[2i]
Right child : A[i] = A[2i + 1]
Parent of A[i] : A[i] : A[i/2]

→ Three operations can be done on heaps

→ Max - heapify (Restore max heap property)
→ build - Max - heap (building Max heap)
→ Heapsort (sort an array)

Additional pt

4) Algo Max-heapify (A, i, n)

```
l ← left(i)
R ← Right(i)
if (l ≤ n and A[l] > A[i])
    then largest ← l
    else largest ← i.
if r ≤ n and A[r] > A[largest]
    then largest ← r
if largest ≠ r
    then exchange A[i] ↔ A[largest]
    Max-heapify (A, largest, n)
```

Algo Build Max-heap (A)

```
n = length[A]
for i = n/2 , i - -
    do Max-heapify (A, i, n)
```

Priority queue :-

→ It is an ADT for maintaing a set S of elements with each element have a priority with it.

→ An element with high priority is served before an element with low priority and vice-versa.

→ If elements have same priority, they are served as their order in the queue.

→ It has operations

1) push (x) : inserting element x

2) top or peek() : returns the element of S with highest or lowest priority.

3) pop() : returns and pops the top element.

9.3 Maps :-

→ Algorithm used here is Dijkstra's

→ It is used in G Maps to find the shortest, easiest path to travel

→ Algo : we will be having vertexes and their weight G = (V, E) and we have to find shortest path to destination from the source.

1) All paths are assumed to be ∞ except the dist[s] which will be zero.
2) We will be finding new paths and updating d[v] for all v ∈ Adj[u]. This is called as relaxation.
3) Then we need to change the old path to new shorter path

if d[u] + w(u,v) < d[v] then

d[v] = d[u] + w[u,v].

4) This is how it works.

Unit 5.

Q.1 File class with example

1) Acts like a wrapper class for file name.
2) A file name like "num.txt" has only string properties.
3) File has some very useful methods, they are as follow

→ exists: test if a file exist.

→ canRead: tests if OS will let you read a file.

→ canWrite: tests if the OS will let you write a file.

→ delete: deletes the file.

→ length: returns the no of bytes in that file.

→ getName: returns the name of file

4) Eg

```
File numFile = new File("num.txt");
if (numFile.exist()) {
    System.out.println(numFile.length());
}
```

2 Serialization in file system.

→ To serialize an object means to convert its state to a byte stream so that the byte stream can be reverted back into a copy of

→ As o/p stream accepts only binary it doesn't understand text.

→ In order to tackle this problem whatever we want to accept and display we put it in a class that implement serializable.

→ Java assigns a serial number to each object written out.

→ The serializable class has class instance variable then they should also be serializable

Q.3 File parameter

→ We can give the path of the file as parameter

```
File f = new File("Path");
         ↓ name of file
         String
parse(f.getAbsolutePath());
```

↳ This can be passed to methods

→ Using printWriter we use write method we provid it string as parameter

```
FileWriter myWriter = new
                      FileWriter("Assi.txt");
myWriter.write("Write to a file");
myWriter.close();
```

→ During reading we just have to provide the path of file to file reader.

```
FileReader reader = new FileReader
                    ("My XD.txt");
                     File name →
```

```
while((character = reader.read())
    != -1) {
        System.out.print((char) character)
    ;
    }
    reader.close();
```

q.4 q.5 File IO stream and buffered reader

→ Java does provide strong, flexible support for I/O as it relates to files and network

→ Java's I/O system is cohensive and consistent.

→ Program Data is retrieved from input source.

→ Program Results are sent to an o/p destination.

→ Input stream : a stream that provides i/p to program
System.in is an input stream.
Output stream : accept o/p from program
System.out is an o/p stream.

→ Code :-

```
import java.util.scanner;
class steam {
    public static void main (string
            args[]) {
Scanner xd = new Scanner (System.in);
System.out.println (" Enter a letter
String s = sc.nextLine();
    String s = sc.nextLine();
}
```

input stream

o/p stream

b] Buffer Reader

→ To read using buffer means like reading in chunks.

→ There is a little delay but it's better than non buffered methods.

→ Disk operation per buffer of byte -- lower overhead.

→ Code :

```
import java.io.*;

class mHead {
  public static void main (String
    args [ ]) throws IOException {
      char c;
      BufferedReader br =
        BufferedReader ( new
      InputStreamReader (System.in);
      System.out.println (" Enter
      characters \n 'q' to quit ");
      do {
        c = br.read( );
        System.out.println (c);
      } while ( c != 'q' );
  }
}
```

Page No.:
Date:
YOUVA

Q.5 File reader and File writer class

a] File reader

1) FileReader is a subclass of reader.

2) We have to import java.io.Reader this is about how we create a File reader

```
Reader input = new FileReader();
```

3) It can take a txt file as parameter.

4) [read (char [] arr, int st, int len)] - reads the no of char equal to length and stores it into array.

b] ~~Print Writer~~ (File Writer)

1) FileWriter is a subclass of OutputStreamWriter.

2) Syntax :

```
FileWriter fw = new FileWriter
( );
```

3) In its constructor we can give it a path or an object of file class.

4) Main Method

write (String txt) → It is used to write the string into FileWriter.

flush () → flushes data of fileWriter.

close () → used to close FileWriter.

Unit 4.

Q.1 Hashing

→ Key, value pairs are stored in fixed size table called a hash table.

→ hash table is partitioned into many buckets.

→ Each bucket has many slots. Each slot holds record.

→ A hash f^n (x) transforms the identifier (key) into an address in the hash table.

Algo :-

a] Inserting :-

→ For inserting a new record, key must somehow be converted to array index.

→ Suppose n = 10 (size of table)

we have to insert 5

5 mod 10 = 5

→ 5 will be inserted at 5th position

b] Searching

→ We calculate the hash value first

→ Later we check that location for the key, if the value is not there we continue to check array till end.

c] deleting :-

→ Records may also be deleted from hash table

→ The only one thing is that, the records should not be left empty

d] Collision resolution methods :

→ The spot where we try to insert a element on to an occupied spot

→ It can be resolved by collision resolution technique

1) Quadratic Probing :-

→ Quadratic probing.

→ $f(i) = i^2$

→ In Quadratic probing, if i-th position is occupied we check the i+1st next we check the i+4th next i+9th etc.

2) Quad linear probing :-

→ $f(i) = i^2$

→ When there is a collision we just probe the next slot in the table.

3) Double hashing :-

→ Firstly we need to find the first key

$k1 = H(Key \% \text{ size of table}(M))$

The second hash function we need to find value in range from 1 to capacity - 1

$H(K2)$ hash2(obj) $= 1 + (k1 \% M)$

size of table.

4) Open hashing (chaining)

→ The key concept here is all elements that hash to a particular bucket are placed on that bucket's linked list.

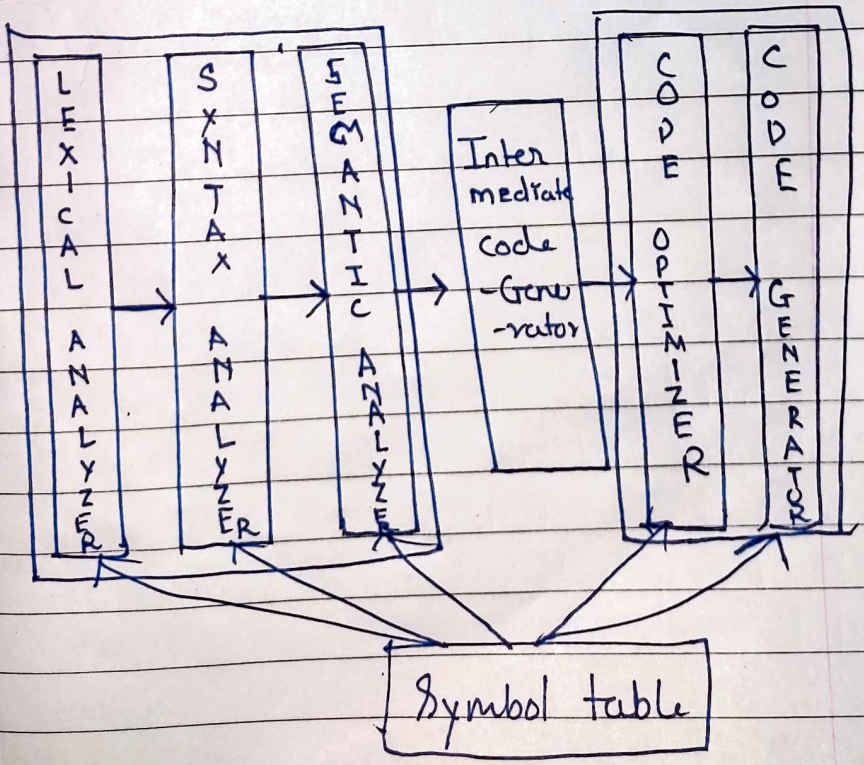
2 Symbol table

→ Symbol table is a data structure used by a language translator such as a compiler or interpreter.

→ It stores information about various source language.

→ The information is collected by analysis phase of the compiler and used by the synthesis phases to generate the output which is in machine language.

→



→ Symbol table has three operations

1) Insert().
2) Scope Management.
3) lookup().

→ Each table is list names and there associated attribute and the table are organized into a stack.