Data Structures

Iterative –

Factorial of a number. 8! – 8\*7\*6\*5\*4\*3\*2\*1

public long fact(int num)

{

long fact =1;

for(int n=num; n >0;n--)

fact = fact \* n; // 1 \* 8 8 \* 7 56 \* 6…

return fact;

}

//recursive

public long fact(int num) // 5

{

// base condition

If(num==1) return 1;



// call fact(..)



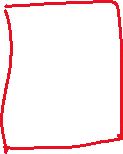
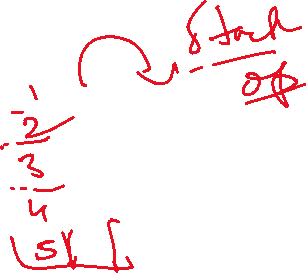
return num \* fact(num-1);

}

5 \* fact(4)



4 \* fact(3)



3 \* fact(2)



2 \* fact(1)



Recursive – too many stack operations are involved.

Towers of Hanoi –

Case study

1. Print the fibonocci number.
2. 1 1 2 3 5 8 13 …..

fib(int n) // fib(6) 🡪 5

{

}

Memory Management

C – malloc and free

Eligible for gc

C++ -- new and delete

Java – new and automatic gc

12

User u1=new User(12)

u1

User u3 = u1;

u3

u1 = new User(24)

24

u3 = new User(25)

25

mark and sweep – JVM for deallocation

u3=null;

new User(10).setEmail(“abc”).getEmail()

// problem

Write a java program to swap 2 User Objects.

User has name and id;

**private** **static** **void** swap(User user1, User user2) {

User temp=user1;

user1=user2;

user2=user1;

}

User1 (main)

12 sam

User1

User2

24 peter

User2

String s1=”hello”;

String s2=”hello”;

S3

String s3=new String(“hello”)

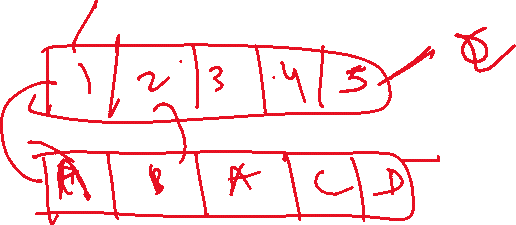
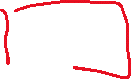
hello

S1

Constant pool

“hello”

S2



Printf(“%d %d”,12,34);

(3\*4)+16/(2\*6)

Push – (

3 \* 4

) - pop()

Push + 16 (

1. \* 6 )
2. Peek() topmost element

Public add(int x,int y)

{

Int z=x+y;

Create a class MyStack of Integers with array

push(int element)

int pop() -- topmost element

peek() – top element

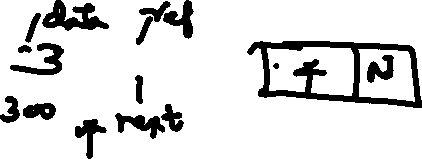
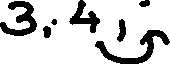
display all the elements()

Stack<Integer> stack1 = new Stack<Integer>()

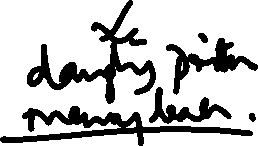
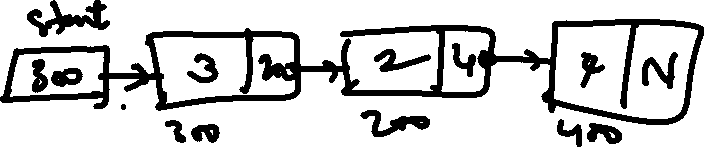
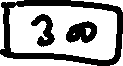
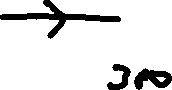
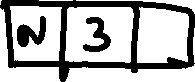
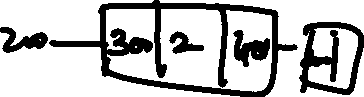
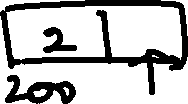
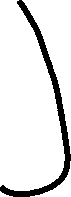
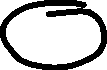
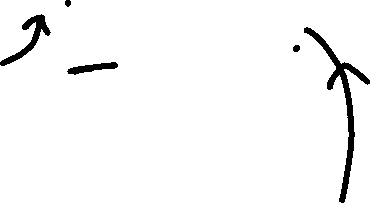
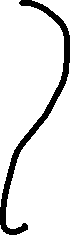
Stack1.push(10);



LinkedList



.



Create a Singly Linked List of Integers to perform the following

A Stack Operations

Class LinkedList

{

Class Node

{

}

start

10 Null

200

100

20 100

200

**public** **void** push(**int** element)

{

Node new1 = **new** Node(); // create a new node



new1.data=element;

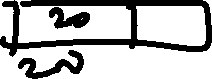
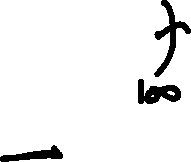


new1.next=start;



start = new1;

}



front rear when queue is empty

null

null

20 null

200

200

200

When queue has 1 element

40 null

400

400

20 400

200

200

Prefix –

+3\*45

Steps

1. Reverse the expression – reverse()
2. 54\*3+
3. Push(5) push(4)
4. Read \*
5. Pop() pop()
6. Cal result
7. Push the result
8. 9
9. Push 3
10. Read +
11. Pop() pop()
12. Cal result

Infix -- 3+4\*5

postfix -- 345\*+