National University of Computer and Emerging Sciences

Department of Computer Science **Chiniot-Faisalabad Campus**

0126

Question No. 1 Stack ADT and Applications [15 Marks]

Part A (10): Consider the usual algorithm to convert an infix expression to a postfix expression. Suppose that you have read 10 input characters during a conversion and stack now contains these symbols:

/
+
(
*

Now, consider that you read and process the 11th symbol of the input. Considering the above-mentioned stack state and draw the updated states for all the cases mentioned below separately where the 11th symbol is:

1. A number { 5 }:
2. A left parenthesis { (}:
3. A right parenthesis {) }:
4. A minus sign { - }:
5. A division sign { / }:

1. The number 5 will not go to the stack. I will go to the output ✓	2. (/ + (* • '(' will go the top of the stack. ✓
3. (/ + (* → / + (* • ')' wille parenthisis will pop all the operator until '(' comes. And send them to the output.	4. - + (* • as '-' has low precedence then '/' that's why pop '/' from stack And push - on stack.
5. / / + (* '/' could be pushed on stack because '/' are same operators. 04	

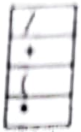
Part B (5): Consider the following infix expression and convert it to polish notation using stacks. Clearly mention the all the steps using stacks to claim ANY marks.

((A*B/C)^D)*(E+F)^2

1) Firstly reversed the expression becaue converting it into prefix. 2) In this expression I'm not checking / taking care of the precedence of operators. 3) Pushing the operands into output and operator into stacking.	4) Poping the operators after any closing bracket. 5) The final expression is given: 2FE+DCBA*/^*^.

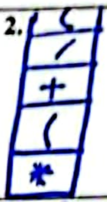
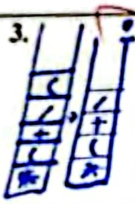
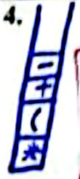
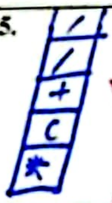
Question No. 1 Stack ADT and Applications [15 Marks]

Part A (10): Consider the usual algorithm to convert an infix expression to a postfix expression. Suppose that you have read 10 input characters during a conversion and stack now contains these symbols:



Now, consider that you read and process the 11th symbol of the input. Considering the above-mentioned stack state and draw the updated states for all the cases mentioned below separately where the 11th symbol is:

1. A number (5):
2. A left parenthesis (( ):
3. A right parenthesis ( )):
4. A minus sign (-):
5. A division sign (/):

1. The number 5 will not go to the stack. I will go to the output	2.  • '(' will go the top of the stack.
3.  ')' write parenthisis will pop all the operators until '(' comes. And send them to the output.	4.  • As '-' has low precedence then '/' that's why pop '/' from stack And push - on stack.
5.  '/' could be pushed on stack because '/' are same operators. 04	

Part B (5): Consider the following infix expression and convert it to polish notation using stacks. Clearly mention the all the steps using stacks to claim ANY marks.

((A*B/C)^D)*(E+F)^2

1) Firstly reversed the expression because converting it into prefix.

2) In this expression I'm not checking /taking care of the precedence of operators.

3) Pushing the operands into output and operators into stacking.

4) Poping the operators after any closing bracket.

5) The final expression is given:

2FE+DCBA*/^*^.

Question No. 2 Queue ADT [10 Marks]

Given a Queue data structure that supports standard operations like *enqueue(), dequeue(), isFull(),* and, *isEmpty()*. The Queue ADT is given as follows for your reference.

```
class QueueADT{
    private:
        int *queueArray; //Pointer to array implemented as Queue
        int queueSize;  //Total size of the Queue
        int front; int rear; int numItems;
    public:
        QueueADT(int);
        ~IntQueue();
        bool isEmpty();
        bool isFull();
        bool enqueue(int);
        int dequeue();
};
```

However, in this task you need to implement *Stack ADT* using only instances of *Queue ADT* operations allowed on the instances. To keep the task simple following *Stack ADT* functions are required to implement:

bool push(int data) // push an element in the queue maintaining the stack order
bool pop(int& data) //pop an item from queue and return it

<u>Note: You are not allowed to change the functionality of Queue ADT. However, you can use multiple instances of queue object as per your logic.</u>

bool push(int data)

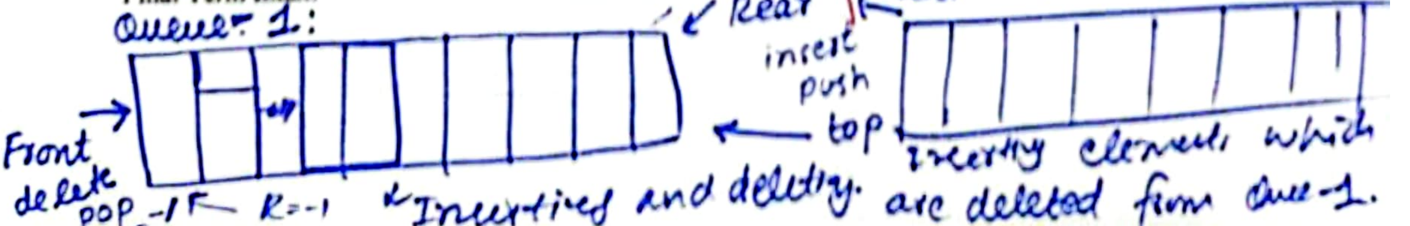
```
{
if(isFull()){
 cout<<"Stack is Full"<<endl;
}
 rear ++;
queueArray[rear]=data;
 numofItems++;
 return true;

}
```

// This task could also be done by using "2 Queue" Adts. insertion at "1-Queue". And deleting elements from "Queue-1" and storing into "Queue-2". will give us "Stack ADT."

bool pop(int&data)

```
if(isEmpty(){
coutcc"cann't pop element"<<endl;
}
int temp;
~~queueArray~~queArray[rear]=obj.data;
temp=queArray[rear];
front++;
rear--;         // As, deletion and
numofItems--;    insertion will take
return temp;     at rear.
retur true;
}
}
```

work like Stack ADT.



0126

National University of Computer and Emerging Sciences

Department of Computer Science **Chiniot-Faisalabad Campus**

Question No. 3 Linked List [10 Marks]

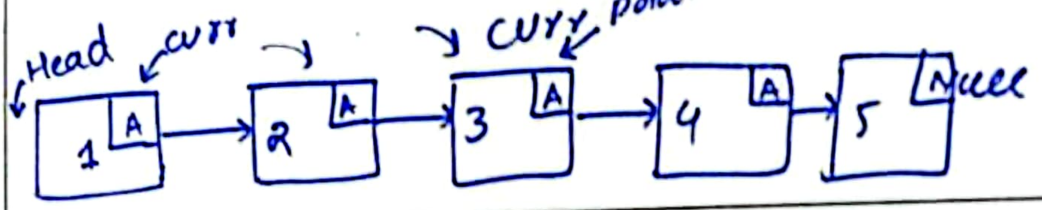
Given a singly linked list, rotate it to the right by *k* nodes, where *k* denotes the number of right shifts you need to perform.

```
Sample Input: 1->2->3->4->5 for k = 3
Sample Output:3->4->5->2->1

node * rotateSinglyList(node* head, int k)
{
    Node* current = head;
       currIndex = 1;
while (current && (currIndex ≤ 3)
   {  current = current->next;
       currentIndex++;
   }
while (current->next != NULL) {
       current->next = head;
          head = current;
    }
 return current;

}
 // Code will perform 3 shifts and
    3->4->5->1->2.
```

// Now the node with data(3) will become the first node. And head points to this node;

02

0



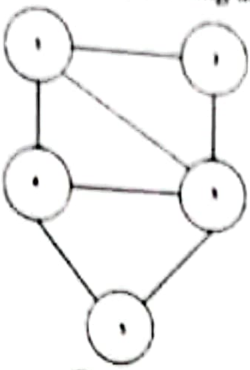
National University of Computer and Emerging Sciences

Department of Computer Science Chiniot-Faisalabad Campus

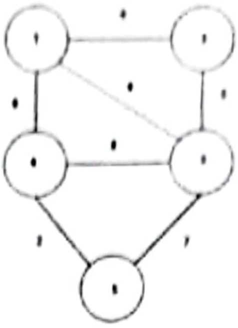
0126

Question No. 5 Graphs ADT [12 Marks]

A graph is a collection of edges and vertices, i.e., G = (V, E), where V is the set of vertices and E is the set of edges. We can represent a graph using an adjacency matrix, adjacency list, and compact list. Consider the following undirected graphs.



Graph A



Graph B

Part A (04): Represent graph A and B using the compact list approach

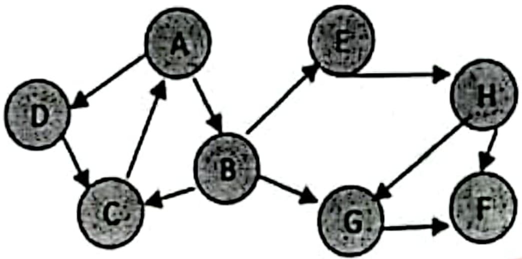
compact list = {(1,2),(1,3)(1,4), (2,1),(2,3),(3,2),(3,1), (3,4),(3,5),(4,1),(4,3), (4,5),(5,4),(5,3)}	compat list = {(1,2),(1,3),(1,4),(2,1), (2,3),(3,2),(3,1),(3,4), (3,5),(4,1),(4,3),(4,5), (5,4),(5,3)}.

Part B (08): Consider the following given graph and assume that there is a decision between multiple neighbor nodes in the BFS or DFS algorithms; we will always choose the neighbor node closest to the visiting node; for example, if we must visit P and R from Q, we will visit P first followed by R. Keeping this in mind, answer the following given the starting node A,

i) In what order will the nodes be visited using a breadth first search

ii) In what order will the nodes be visited using a depth first search

Note: write the answers only. You can use the backside of this page for calculations.



Breadth First Search

A, B, D, C, E, G, H, F

3

Depth First Search

C F, G, H, E, B, D, A

0126

National University of Computer and Emerging Sciences

Department of Computer Science **Chiniot-Faisalabad Campus**

Question No. 4 Hash Maps [20 Marks]

Note: Put only the final values in the blank cells of the table. Use another side of the paper for rough work which carries no marks. More than 2 mistakes in filling the table will result in zero marks.

(a) **[Linear Probing] (04)** Suppose that the following keys are inserted in the order

A B C D E F G

into an initially empty linear-probing hash table of size 7, using the following hash function:

08

KEYS	A	B	C	D	E	F	G
HASH(KEY,7)	3	1	4	1	5	2	5

What is the result of the linear-probing array? Assume that the array size is fixed.

0	1	2	3	4	5	6
1	1	3	3	4	5	2

(b) **[Double Hashing] (10)** Load the keys **18, 26, 35, 9, 64, 47, 96, 36,** and **70** in this order, in an empty hash table of size **n = 13.** Use double hashing **h_i(key) = [h(key) + i*h_p(key)]% n** with the first hash function: **h(key) = key % n** and the second hash function: **h_p(key) = 1 + key % (n-1).** For your convenience, a few entries are already filled in the table.

06

0	1	2	3	4	5	6	7	8	9	10	11	12
26			70	9	18	96		47	35	36		64

(c) **[Quadratic Probing] (06)** Load the keys **23, 13, 21, 14, 7, 8,** and **15,** in this order, in a hash table of size 7 using quadratic probing with **c(i) = ±i²** and the hash function: **h(key) = key % 7.** For your convenience, a few entries are already filled in the table.

0	1	2	3	4	5	6
21	14	23		7	15	13

02

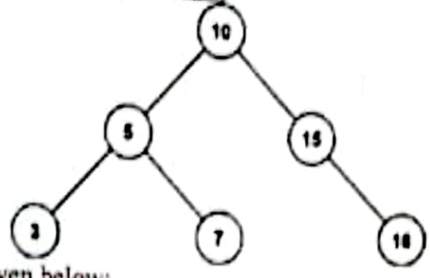
National University of Computer and Emerging Sciences

Department of Computer Science

0126

Question No. 7 Binary Search Tree [02+05+03 = 10 Marks] **Chiniot-Faisalabad Campus**

Assume that you are writing a destructor for the BST ADT which in turn calls a function named as *destorySubTree(TreeNode *node)* with tree root pointer as a value argument. <u>You are required to complete the following recursive function strictly keeping the context in view. Tree traversals may be a bit helpful here. Identify which traversal will be directly applied and provide a strong justification for it. After that provide the code implementation with recursive call stack as well, a reference tree is given to use it for the call stacks.</u>



3.5

<u>The function signature is given below:</u>

(02) Choose the appropriate Traversal and justify your rationale:

Inorder traveral will be used to destroy tree. Because, inorder will give the exact values' order in which we inverted. So, it will be appropriate for this code.

(05) Implementation

```
void IntBST:: destroySubTree(TreeNode *node){
// All the nodes of the tree must be deleted before returning to the
caller function (i.e., Destructor)
   Node* left=Nullptr;
   Node* right = Nullptr;
if(Node!=NULL){
  left = destroySubtree(node->left);
   cout << ~~d~~node->left;
  right = destroySubtree(node->right);
   deleted node*->left;
}
}
```

02.5

(03) Call Stack Explanation using the given tree:

Firstly, Function has root node which is 10. After that function will be called for node's left child, The whole subtree is visited by calling using recursive functions. After this node's right subtree will be called recursivily. And whole tree will be visited. And deletion will be done according to these stack recursive calls.

01

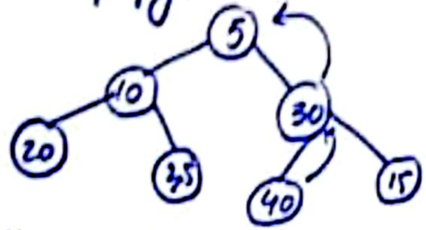
0126

Question No. 6 Binary Heaps [08 Marks]

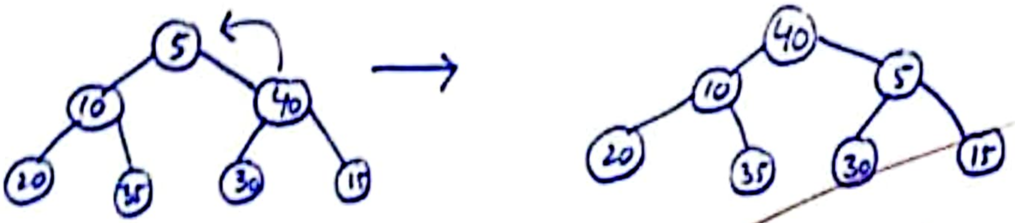
Heapify is the fast method to create a heap. Create a Max heap using heapify for the elements 5, 10, 30, 20, 35, 40, and 15. Clearly mark and write down all intermediatory steps. There will be no mark for direct answer.

Max Heap:

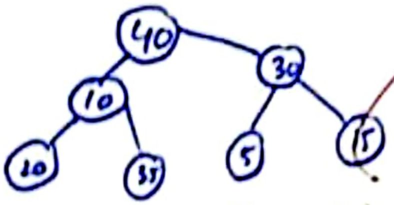
- First inserting all the element than performing heapify.



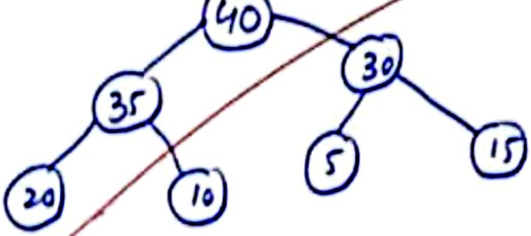
- As, 40 is the max element. So, making 40 the root node



- Now in Right subtree 30 is max element after 30 . So, swapping 30 with 5.



- In left-subtree 35 is max after 40 and is greater than 10 and 20. So swapping 35 with 10.



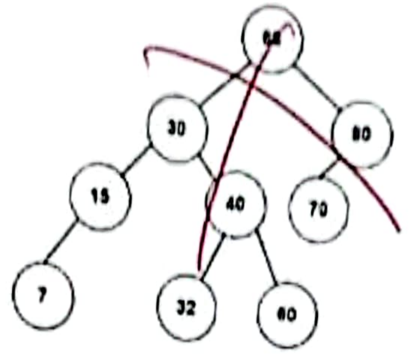
8

This is the final max heap.

0126

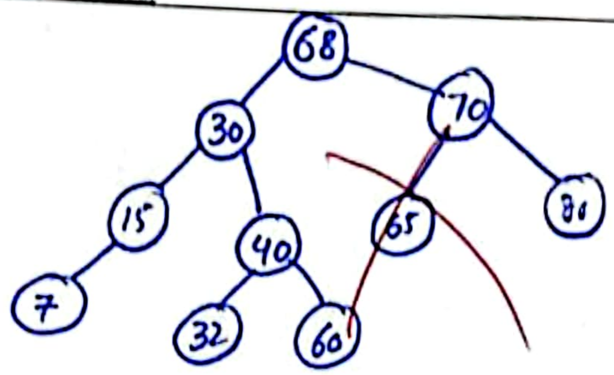
Question No. 8 AVL Tree [15 Marks]

Part A (03): Consider the AVL Tree and calculate the balance factor of each node.



Part B (06): Consider the given AVL Tree again in **Part A**, Insert a new element **65**. Recalculate the BF and identify the imbalanced path if any and, make it balanced by performing the rotations.

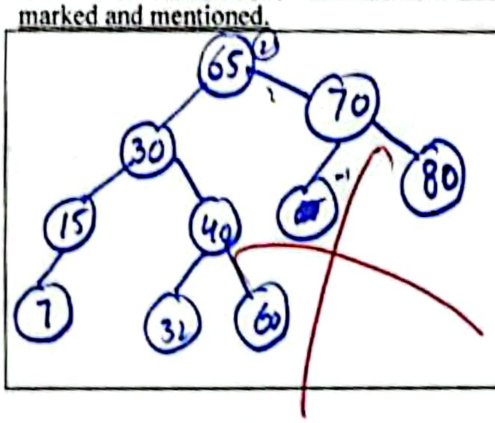
Note: For rotations, construct the final AVL tree in the given box as under to get full marks. You can perform all steps involve in the process of rotations as **rough** work (if it is necessary) on the backside side of the paper but it contains **NO** marks. Rest of the operations should by clearly marked and mentioned.



→ By RR rotation

Part C (06): Consider the given AVL Tree in **Part A** once again, delete node **68** with appropriate node from right subtree. Recalculate the BF and identify the imbalanced path if any and, make it balanced by performing the rotations.

Note: For rotations, construct the final AVL tree in the given box as under to get full marks. You can perform all steps involve in the process of rotations as **rough** work (if it is necessary) on the backside side of the paper but it contains **NO** marks. Rest of the operations should by clearly marked and mentioned.



→