End - Sem - Examination

Name :- Tanishq Chauhan

Roll No. :- 21C3184

Subject :- Data Structure (CER3C3)

Semester :- 3rd Semester

Branch :- Computer Science & Engineering (CS-B)

Chauhan

Tanishq Chauhan
21C3184
CS-B

Chauhan

Date
Page 01

Data Structure (CER3C3)

* Answer of Q. No. 4.

(a). Insertion of a Node

1. Create a new BST node and assign value to it

2. insert (node, key)

(i) If root = NULL,
return the new node to the calling function.

(ii) If root => data < key
call the insert function with root => right
and assign the return value in root => right.

root => right = insert (root => right, key)

(iii) If root => data > key
call the insert function with root -> left
and assign the return value in root => left.

root => left = insert (root => left, key)

3. Finally, return the original root pointer to the calling function.

Tanishq Chauhan
21C3184
CS-B

(b) No. of leaf nodes

getLeafCount(node)

1. If node is NULL then return 0.

2. Else If left and right child nodes are NULL return 1.

3. Else recursively calculate leaf count of the tree using below formula.

Leaf count of a tree = Leaf count of left subtree
+
Leaf count of right subtree

getLeafCount(node -> left) +
getLeafCount(node -> right)

(c) Number of nodes

CountNodes(node)

1. If node is NULL then return 0

If x = NULL
return 0
[End If]

Tanishq Chauhan
21C3184
CS-B

2. ~~If~~ If (x -> left != NULL)

n = n+1;

CountNode (x->left)

[End If]

If [x -> right != NULL)

n = n+1

Count Node (x->right

[End If]

3. Retur n;

Or Simply, recursively we can calculate Number of Nodes of a tree using below formula.

Total No. of Nodes = Total No. of nodes in leftsub tree
+
Total No. of nodes in right sub tree
+
1

```
int *res = 0

res += (CountNode (root->left) + CountNode(root->right));
return res;
```

Tanishq Chauhan
21C3184
CS-B

(e) Find the address of parent node to specific node

We can find the parent of a specific node using the following code below:

```
void findParent(struct Node* node, int val, int parent)
{
    if(node == NULL)
        return;

    // If current node is the required node
    if(node->data == val)
    {
        cout<<parent;
    }

    else
    {
        // Recursive calls for the children
        // of the current node
        // Current node is now the new parent

        findParent(node->left, val, node->data);
        findParent(node->left, val, node->data);
    }
}
```

Tanishq Chauhan
21C3184
CS-B

Chauhan

Date
Page 05

Answer of Q. No. 1. (b)

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

  int M = 3;    // Declared Globally
  int N;

  void transpose (int array[N][M], int rows, int
                                          columns);
  {
     int transp[N][M]; // transpose sparse matrix

     if (N>0)

     {
        // non zero matrix
        int current = 0;

     // current row of transp[][]

     for (int c=0; c<columns ; c++)

     {
        // for each column of matrix

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

        {
           // iterate each non-zero transp
           // if it has column index = c;
```

Tanishq Chauhan
21C3184
CS-B


```
            if (array[i][1] == c)
            {
                transp[current][0] = c;
                transp[current][1] = array[i][0];
                transp[current++][2] = array[i][2];
            }
        }
    }
} // end if (N>0)

cout << "Transpose Matrix:" << endl;

for (int i=0; i<N; i++)
{
    for (int j=0; j<M; i++)
    {
        cout << array[i][j] <<
        cout << endl;
    }
}
}
```

Tanishq Chauhan
21C3184
CS-B

Chauhan

Date
Page 07

Answer of Q. No-1 (c)

```
#include <iostream>
#include < string.h>
using namespace std;

void substring (string s, int n)
{
    String r = s . substr (0, n-1);
    cout << " Substring is " << r << endl;
}

string substring (string s, int n)
{
    return s.substr(0,n);
}

int main()
{
   // Take any string

   string s1 = "Hello India";
   string result = substring (s1, 5);
```

Tanishq Chauhan
21C3184
CS-B

Chauhan

Date
Page 08

Answer of Q. No 1(c)

```
#include <iostream>
#include <string.h>
using namespace std;

string substring (string s , int n)
{
  return s.substr (0, n);
}

int main ()
{
  // Take any string

  string s1 = "Hello India";
  string result = substring (s1, 5);

  // print the result

  cout << "Substring is : " << result;

  return 0;
}
```

Tanishq Chauhan
21C3184
CS-B

Chauhan

Date
Page 09

Answer of Q. No. 1 (b)

```
#include <bits/stdc++.h>
using name space std;

void transpose (int matrix[][3], int n)
{
    int m1[size][3], m2[size][3];

    for (int j=0; j<3; j++
    {
      for (int i=0; i<n; i++
      {
        if (j==0)
        {
          m1[i][i] = matrix[i][j]
        }

        if (j==1)
        {
          m1[i][0] = matrix[i][j];
        }
        if (j==2)
        {
          m1[i][j] = matrix[i][j];
        }
```

Tanishq Chauhan
21C3184
CS-B


```
    }
  }
for (int i=0; i<n; i++)
{
  for (int j=i+1; j<n; j++)
  {
    if (m1[i][0] > m1[j][0] || (m1[i][0] == m1[j][0] && m1[i][1] > m1[j][1]))
    {
      for (int k=0; k<3; k++)
      {
        int c = m1[j][k];
        m1[j][k] = m1[i][k];
        m1[i][k] = c;
      }
    }
  }
}
```

Tanishq Chauhan
2103184
CS-B

Chauhan

Date
Page 11

```
cout << " The transpose of the entered matrix
        is " <<endl;

for
```

Tanishq Chauhan
2103184
CS-B

Answer of Q. No 2 (c)

Algorithm for Infix to postfix (using stack)

Step 1 : Scan the infix expression from left to right.

Step 2 : If the scanned character is an operand output it.

Step 3 : Else.

(i) If the precedence of the scanned operator is greater the precedence of the operator in the stack (or the stack is empty or the stack contains '(', ')' push it.

(ii) Else, pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that push the scanned operator to the stack. (If you encounter parenthesis) while popping then stop there and pushed the scanned operator in stack);

Step 4 : If the scanned character is an '(' push it to the stack.

Tanishq Chauhan
21C3184
CS-B

Step 5: If the scanned character is an ')' pop the stack & output it until a '(' is encountered & discard both parathensis.

Step 6: Repeat steps 2-6 until fix expression is scanned.

Step 7: Print the output.

Step 8: Pop and output from the stack until it is not empty.

Procedure:

```
: x is an operand : print(x)
: x = ')' while stack (top) ≠ '(' do
    print (stack(top)) ; top ← top - 1
    end
  top ← top - 1
: else while ISP ( stack(top) = ICP(x) do
    print (stack (top))

  top ← top - 1
  end

  call ADD (x, stack, n, top)
  end
  forever
  end postfix
```

Tanishq Chauhan
2103184
CS-B

Answer of Q. No 2 (b)

(i) To insert an element (enqueue) in a circular queue :-

Step 1 : [check for the ~~oper~~ overflow]

If Front = 1 and rear = n
Output "Overflow" and return

Step 2 : [Insert element in the queue]

Else if front = 0
Front = 1
rear = 1
Q [rear] = value

Step 3 : [check if the rear at the end of the queue]

Else if rear = n
rear = 1
q [rear] = value

Step 4 : [Insert an element]
else
rear = rear + 1
Q [rear] = value

Step 5 : Return

Tanishq Chauhan
21C3184
CS-B

(ii) To delete an element or dequeue in a circular queue

Step 1 : [Check for underflow]
If front = 0
Output "Underflow" and return

Step 2 : [Remove the element]
value = Q[front]

Step 3 : [Check whether the queue is empty or not]

If front = rear
front = 0
rear = 0

Step 4 : [Check the front pointer position]

else if front = n
front = 1
else
front = front + 1

Step 5: [Return the removed element]
Return [value]

Tanishq Chauhan
21C3184
CS-B

Answer of Q. No 3 (a)

Reversing the singly linked list

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

struct node

{
    int data;
    struct node * next;
};

// function for reversal

void reverse (node * & head)

{
    node *p = head;   //initializing node pointer)
    node *q = NULL;
    node * r = NULL;

    while (p)

    {
        r = q;
        q = p;
        p = p->next;
        q -> next = r;
    }
```

Tanishq Chauhan
21C3184
CS-B

Chauhan

Date
Page 17

```
    head = q;
}
```

Tanishq Chauhan
2103184
CS-B

Answer of Q. No. 3 (c)

Insert and Delete Element

```
# // for insertion          // Insert
void insertmiddle (node * head , int val)
{
   int ctr = 0;  // for counting no. of nodes)
   node * ptr = head;
   while (ptr -> next ! = NULL)
   {
      ctr ++;
      ptr = ptr -> next;
   }

   node* p = new (node);
   node * q = new (node);

    p = head;

    int i = 0;

while (i != (ctr/2) -1)  // condition for reaching
                            the middle.
```

```
    {
        p = p -> next;
        p++;
    }

    q -> data = val;
    q -> next = p -> next;
    p -> next = q;
    q -> prev = p -> next -> prev;
    p -> next -> prev = q;
}
```

// for deleting middle element // Delete

```
void deletemid (node * head)
{
    int ctr = 0;
    node * ptr = head;

    while (ptr -> next != NULL)
    {
        ctr ++;          // counting
        ptr = ptr -> next;
    }

    node *p = head;      // initializing nodes
    node *q;
    node * r = p -> next;
    int i=0;
```

Tanishq Chauhan
21C3184
CS-B


```
while (i != (ctr/2)-1)      // reaching middle node
{
    r = r -> next;
    p = p -> next;
    i++;
}

p -> next = r -> next;      // Re-linking the elements

r -> next -> prev = p;

free(r);                    // deleting mid
}
```

Tanishq Chauhan
21C3184
CS-B

Answer of Q. No 5 (C)

Breadth First Search (BFS)	Depth First Search (DFS)
1. BFS uses Queue data structure for finding the shortest path	1. DFS uses stack data structure.
2. BFS operates using FIFO list	2. DFS operates using LIFO list.
3. BFS can be used to find single source shortest path in an unweighted graph because in BFS, we reach a vertex with minimum number of edges from a source vertex.	3. In DFS, we might traverse through more edges to reach destination vertex from a source.
4. BFS is more suitable for searching vertices which are closer to the given source	4. DFS is more suitable when there are solutions away from source.
5. BFS considers all neighbours first and therefore not suitable for decision making tree used in games or puzzles.	5. In DFS we make a decision, then explore all paths through this decision. And if this decision leads to win situation, we stop.

Tanishq Chauhan
21C3184
CS-B

6 In BFS, siblings are visited befor the children.	6. In DFS, children are visited before the siblings.

* Depth - First Search Without Recursion

```
dfs (r)
visit (r, noparent)
 v = r
 next
    edge

for all vertices w adjacent to v

    if evaluate (v, w) is unvisited
      depth = depth + 1
       visit (w, v);
         v = w;

    goto next edge

    if v ≠ r
    v = evaluate (p, v)
     depth = depth - 1

    goto next edge.
```