WEEK-6 19131A05P1

Aim: Write a program to implement factorial using stack applications.

Description

Stack is an ordered data structure to store data in LIFO order. Last in first out.

Insertion operation is called push while deletion operation is called pop.

Applications of stack.

* stacks can be used
  i. for evaluation of infix, postfix expressions.
  ii. To check parantheses matching in an expression.
  iii. for conversion from 1 form of expression to another.
  iv. For memory management.
  v. In backtracking problems.

Algorithm

Start

step 1: Initialize top to -1

step 2: Read the value of n from user.

step 3: set value of t to n.

step 4: Run a while loop until the condition n >0 becomes false

  4.1: Increment top value i.e top++.

4.2: Set sta[top] = n;

4.3: Decrement n value i.e n--

step 5: Set fact = 1

step 6: Run a while loop untill top<0.

6.1: Perform fact* = sta[top]

6.2: Decrement top value i.e top--.

step 7: Print, 'fact' is the factorial of 'n'.

Program

```
#include <iostream>
using namespace std;
int main()
{
    int n, top=-1;
    cin>>n;
    int sta[n], t=n;
    while(n>0)
    {
        top++;
        sta[top]=n;
        n--;
    }
    int fact=1;
    while(top>=0)
    {
        fact*=sta[top];
        top--;
    }
```

cout << fact << " is the factorial of " << t;

}

Sample Input :

5

Sample Output :

120 is the factorial of 5.

Aim : Write a program to implement the evaluation of postfix expression in stack applications.

Postfix expression is an expression in which the operator appears in the expression after the operand. Every character of the postfix expression is scanned from left to right.

If character is an operator then the 2 values (from top) are popped and operation is performed, result pushed to stack.

If character encountered is operand, it is pushed into the stack.

Algorithm

start

step 1: Initialize and declare the required variables of respective data types and formats.

step 2 : Read the postfix expression.

step 3 : Check each character of postfix string entered

step 4: If ch == operand, then goto 6

step 5: If ch == operator, then goto 7

step 6: Push the operand into the stack

step 7: Pop last 2 operands from stack and evaluate them and push back result into the stack.

stop.

Program

```
#include <iostream>
#include <stack>
#include <string.h>
using namespace std;
int main()
{
    stack <int> stack1;
    char s[100];
    cout << "Enter the expression\n";
    cin >> s;
    int i, n1, n2, a;
    for(i=0;s[i] ;i++)
    {
        a = s[i];
        if (isdigit (a))
        {
            a = s[i] - 48;
            stack1. push(a);
        }
```

```
else
{
    n1 = stack1.top();
    stack1.pop();
    n2 = stack1.top();
    stack1.pop();
    switch(a)
    {
        case '+':
            a = n1 + n2;
            stack1.push(a);
            break;
        case '-':
            a = n2 - n1;
            stack1.push(a);
            break;
        case '*':
            a = n1 * n2;
            stack1.push(a);
            break;
        case '/':
            a = n2/n1;
            stack1.push(a);
            break;
        default:
            cout << "Something went wrong!";
    }
}
```

```
}
cout << stack1.top() << " is the evaluated output";
}
```

Sample Input1 :

2 31 * + 9 -

Sample Output1

-4 is the evaluated output .

Aim :-

Write a program to implement number conversion using stack.

Theory

* We can convert a number from 1 form to other like, binary ← decimal , decimal → octal etc using stacks.

Algorithm

Start

step1: Declare required variables , stack variable.

step2: set top = -1.

step3: Read users choice, ch and Read decimal no, n

(a) Decimal to binary

(b) Decimal to ternary

(c) Decimal to Octal

step4: (i) if ch == 1, set b = 2

(ii) if ch == 2 set b = 3

(iii) if ch == 3 set b = 8

step5: set n1 = n.

step 6: Repeat while loop untill n<=0

6.1 : set top += 1

6.2 : d = n%b;

6.3 : arr.push(d);

6.4 : n = n/b

step 7: Repeat the following steps untill while loop condition top!=0 becomes false

7.1 : print arr.top()

7.2 : arr.pop()

Program

```
#include<iostream>
#include<stack>  #include<stdlib.h>
using namespace std;
int main()
{
   int n, n1, d, top=-1, b, ch;
   cin>>n
   cout<< "Enter your choice \n 1. Decimal to binary
      \t 2. Decimal to ternary \t 3. Decimal to Octal \t
       0. exit \n";
   cin>> ch;
   switch (ch)
   {
     case 0: exit(0)
```

```
        case 1: b=2;
                break;
        case 2: b=3;
                break;
        case 3: b=8;
                break;
    }
    while (n>0)
    {
        top++;
        d= n%b;
        arr.push(d);
        n= n/b;
    }
    while (top!=0)
    {
        cout<< arr.top();
        arr.pop();
    }
}
```

Sample Input :

15

1

Expected Output

1111

Actual Output

1111

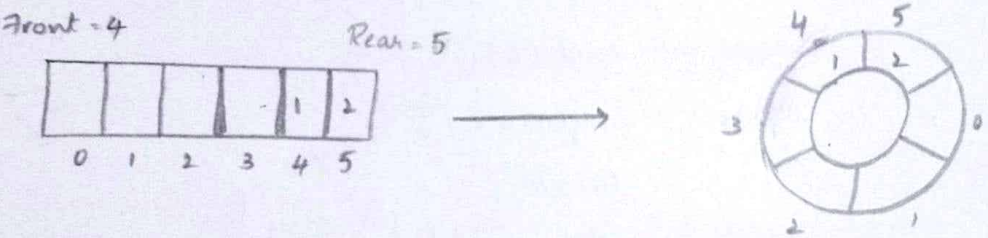
WEEK-7

19-5P1

Aim:- Implementation of circular queue

Theory:

One drawback of array implementation of Queue is, if the rear reaches the end position of the queue, then there might be possibility of some vacant places available at the beginning can't be used; so inorder to overcome this, circular queue is introduced.




A circular queue is a linear data structure, which follows FIFO (first in first out) order but instead of ending the queue at the last position it starts again

PROGRAM

```
#include <stdlib.h>
#include <iostream>
using namespace std;
void enqueue();
void dequeue();
void display();
```

```
int front = -1 , rear = -1;
int queue [5], n=5;
int main ()
 {
    int choice;
    cout << "Enter your choice \n 1. Enqueue \t 2. Dequeue
         \t 3. Display \n";
    cin >> choice;
    while (ch != 0)
     {
        switch (ch)
         {
            case 0 : exit (0);
            case 1 : enqueue ();
                     break;
            case 2 : dequeue ();
                     break;
            case 3: display()
                     break;
            default : cout << " Enter correct choice \n";
         }
        cout << " Enter your choice \n 0. Exit \t 1. enqueue
         \t 2. Dequeue \t 3. Display" << endl;
        cin >> ch;
     }
   return 0;
}
```

```
void enqueue ()
{
    int val;
    cout << " Enter the element ";
    cin >> val;
    if ( front == 0 && rear == n-1)
    {
        cout << " Overflow occured " << endl;
        return ;
    }
    else if ( front == -1 && rear == -1 )
    {
        front = 0;
        rear = 0;
    }
    else if ( rear == n-1 && front != 0 )
    {
        rear = 0;
    }
    else
        rear = rear + 1;
    cout << " Value entered " << endl;
    queue [rear] = val;
}
void dequeue ()
{
    if ( front == -1)
    {
```

```
        cout << " Underflow Occured " << endl;
        return ;
    }
    else if (front == rear)
    {
        front = -1 ;
        rear = -1 ;
    }
    else if (front == n-1)
        front = 0;
    else
        front = front + 1;
}
void display ()
{
    int i ;
    if (front == -1)
    cout << " circular queue is empty!!! " << endl;
    else
    {
        i = front;
        cout << " Circular queue elements are : " << endl;
        if ( front <= rear )
        {
            while (i <= rear)
                cout << queue [i++] << endl;
        }
```

```
        else
        {
            while (i<= n-1)
             cout << queue [i++] << endl;
             i = 0;
             while (i<= rear)
              cout << queue [i++] << endl;
        }
      }
   }
```

Sample Output

```
Enter your choice
1. Enqueue  2. Dequeue  3. Display
1
Enter the element 45
Value entered.
Enter your choice
0. exit  1. Enqueue  2. Dequeue  3. Display.
1
Enter the element 78
Value entered.
Enter your choice
0. Exit  1. Enqueue  2. Dequeue  3. Display.
1
Enter the element 95
Value entered.
```

Enter your choice
0. Exit 1. Enqueue 2. Dequeue 3. Display.
3
Circular queue elements are
45
78
95
Enter your choice
0. Exit 1. Enqueue 2. Dequeue 3. Display.
2
Enter your choice
0. exit 1. Enqueue 2. Dequeue 3. Display.
3
Circular queue elements are
78
95

Aim: Implementation of Priority Queue.

Theory:

A priority queue is an extension of the queue, with some additional property that the elements are arranged on the basis of priority order.

We can perform insertion, deletion operations on priority queues.

This is used to find maximum and minimum value of the given list.

Program

```
#include <bits/stdc++.h>
using namespace std;
#define Max 5
void insert(int);
void delete(int);
void create()
void check(int);
void display();
int pri_queue[Max], front, rear;
int main()
{
    int n, ch;
    cout << "Enter your choice \n 1. Insert \t 2. Delete \t
        3. Display \t 0. exit" << endl;
    create();
    while(1)
    {
        cout << "\n Enter your choice : " << endl;
        cin >> ch;
        switch(ch)
        {
            case 0: exit(0);
            case 1: cout << "\nEnter the value to be
                    inserted : ";
```

```
            cin >> n;
            insert (n);
            break;
        case 2 : cout << "\nEnter value to delete\n";
            cin >> n;
            delete (n);
            break;
        case 3 : display();
            break;
        default : cout << "\nEnter correct choice";
    }
  }
  return 0;
}
void create ()
 {
    front = rear = -1;
 }
void insert (int data)
 {
    if (rear >= Max - 1)
      {
        cout << "\n Queue Overflow occurred\n";
        return;
      }
    if ( front == -1 && rear == -1)
      {
        front++;
        rear++;
```

```
            pri_queue [rear] = idata;
            return;
        }
        else
        {
            check (idata);
        }
        rear ++;
    }
void check(int idata)
{
    int i, j;
    for(i=0; i <= rear; i++)
    {
        if ( idata >= pri_queue[i])
        {
            for(j= rear+1; j>i ; j--)
            {
                pri_que [i] = pri_que [j-1];
            }
            priqueue[i] = idata;
            return ;
        }
    }
    priqueue [i]= idata;
}
void idelete (int idata)
{
    int i ;
```

```
if ( front == -1 && rear == -1)
{
    cout << "\n Queue Underflow occured ";
    return ;
}
for (i=0; i<=rear ;i++)
 {
     if (data == pri_queue [i])
       {
          for (; i<rear; i++)
           {
              priqueue[i] = pri_que [i+1];
           }
         prique [i] = -99;
         rear--;
         if (rear == -1)
           front = -1;
           return ;
       }
 }
    cout << "\n Element not found to delete ";
}
void display ()
{
    if ( front == -1 && rear == -1)
      {
         cout << "\n Queue is empty ";
         return ;
      }
```

```
        for ( ; front <= rear ; front++)
        {
            cout << pri_que[front] << endl;
        }
        front = 0;
    }
```

Sample Output

```
Enter your choice
0. exit   1. Insert   2. Delete   3. Display
1
Enter the value to be inserted : 45
Enter your choice : 2
Enter the value to delete : 45
Enter your choice 2
Queue underflow occurred.
Enter your choice : 3
Queue is empty
Enter your choice : 1
Enter value to be inserted : 67
Enter your choice : 3
67
```

WEEK-8 19-5P1

Aim: Implementation of singly linked lists.

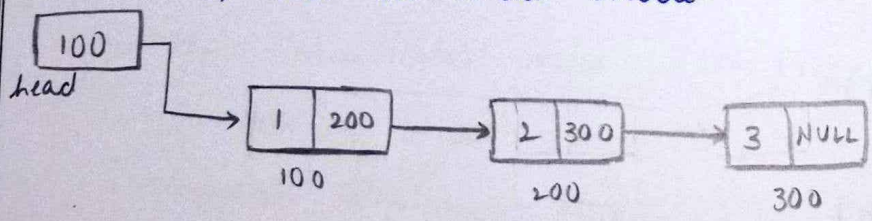
Description: Linked list is a dynamic data structure, and a linear collection of data elements, also called as nodes.

Each node contains 2 fields / 2 & more fields.

Types of linked lists :- ① Singly linked list
② Doubly linked list
③ Circular linked list.

Single linked list :-

Each node in a singly linked list contains 2 parts, a) data
b) link to next node.




* Memory is allocated dynamically. Sizing will not be a problem as we have in arrays. All the operations performed in arrays, insertion, deletion, displaying, searching, sorting can also be performed using a linked list.

Program

```
#include <bits/stdc++.h>
using namespace std;
struct node {
      int data;
       node * next;
 } * head = NULL, * temp;
void insert_end (int n);
void display ();
void delete_node (int n);
int main ()
{
   cout << " Enter your choice \n 0. Exit \t 1. Insert at
the end of list \t 2. Delete a node \t 3. Display \n";
   int ch, n, x;
   cin >> ch;
   while (ch != 0)
   {
      switch (ch)
      {
         case 0: exit (0);
         case 1: cout << " Enter element to be
                  inserted \n";
                 cin >> n;
                 insert_end (n);
                 break;
         case 2: cout << " Enter node to be deleted
                      \n";
```

```
            cin >> n;
            delete_node (n);
            break;
        case 3: cout << "Elements are : ";
                display();
                break;
        default: cout << " Enter correct choice \n";
    }
    cout << " Enter your choice \n 0. Exit \t 1. Insert \t 2. Delete
            \t 3. Display \n";
    cin >> ch;
  }
  return 0;
}
void insert_end (int n)
{
    node * newnode = new node();
    newnode → data = n;
    newnode → next = NULL;
    temp = head;
    if (head == NULL)
        temp = head = newnode;
    else {
        while (temp → next != NULL)
        {
            temp = temp → next;
        }
```

```
        temp -> next = newnode;
        cout << " Inserted \n";
    }
}
void display()
{
   temp = head;
    if (head == 0)
        cout << " Empty linked list " << endl;
    else {
        while (temp != NULL)
        {
            cout << temp -> data << endl;
            temp = temp -> next;
        }
    }
}
void delete_node (int n) {
  node * p = head;
  temp = head;
  int flag = 0;
   if (head == NULL)
          cout << " Cant Delete! " << endl;
    else {
        while (temp -> data != n && temp != NULL)
        {
            p = temp;
            temp = temp -> next;
        }
```

```
        if (temp == head)
          head = NULL;
        else
          p→next = temp→next;
          free(temp);
      }
  }
```

Sample Output

```
Enter your choice
0. Exit  1. Insert at the end of list  2. Delete a node
3. Display
1
Enter the element to be inserted.
45
Inserted.
Enter your choice
0. Exit  1. Insert  2. Delete  3. Display.
1
Enter the element to be inserted.
12
Inserted.
Enter your choice
0. Exit  1. Insert  2. Delete  3. Display.
3
Elements are : 45
12
Enter your choice
0. Exit  1. Insert  2. Delete  3. Display.
```

```
2
Enter node to be deleted
45
Enter your choice
0. Exit    1. Insert    2. Delete    3. Display
3
Elements are : 12
Enter your choice
0. Exit    1. Insert    2. Delete    3. Display.
0
```

Aim: Implementation of a doubly linked list.

Description

A doubly linked list contains an additional pointer to point to previous node (in singly linked list).

We can traverse backward easily in doubly linked list using previous node pointer.

Deletion operation will be most efficient. But major disadvantage is that, all operations require an additional pointer to be maintained, and it requires additional space.

As in singly linked lists, all operations like, insertion, deletion, display, sorting, searching, traversing can be performed using doubly linked lists.

Program

```
#include <cstdlib.h>
#include <iostream>
using namespace std;
struct node {
        int data;
        node * pre, * next;
    } * head = NULL, * temp;
void insert_end (int);
void display ( );
void delete_node(int);
int main ()
{
        int ch, n, x;
        cout << Enter your choice \n 0. Exit \t 1. Insert node
        at end \t 2. Delete specified node \t 3. Display" << endl;
        cin >> ch;
        while (ch! = 0)
        {
            switch (ch)
            {
              case 0 : exit (0);
              case 1 : cout << "Enter the element to be
                       inserted \n";
                       cin >> n;
                       insert_end (n);
```

```
            break;
        case 2 : cout << "Enter the node to be deleted \n";
                 cin >> x;
                 delete_node(x);
                 break;
        case 3 : display();
                 break;
        default : cout << "Enter correct choice ! " << endl;
    }
    cout << "Enter your choice \n";
    cin >> ch;
  }
  return 0;
}
void insert_end(int n) {
    temp = head;
    node * newnode = new node();
    newnode -> data = n;
    newnode -> pre = NULL;
    newnode -> next = NULL;
    if (temp == NULL)
        temp = newnode;
    else {
        while (temp -> next != NULL)
        {
            temp = temp -> next;
        }
```

```
        temp -> next = newnode;
        newnode -> pre = temp;
    }
}
void delete_node (int x)
{
    temp = head;
    node *p, *q;
    if (head == NULL)
        cout << " Can't Delete \n";
    else
    {
        if (head -> next == NULL)
        {
            head = NULL;
        }
        else {
            while (temp -> data != x)
            {
                temp = temp -> next;
                if (temp -> next == NULL)
                {
                    cout << "Can't delete \n";
                    goto label;
                }
            }
            if (temp -> next == NULL)
            {
                p = temp -> pre;
            }   p -> next = NULL;
```

```
            else
            {
               p = temp -> pre;
               q = temp -> next;
               p -> next = temp -> next;
               q -> pre = temp -> pre;
            }
         }
         label: free (temp);
      }
}
void display ()
{
   temp = head;
   if (head == NULL)
   {
      cout << " Empty Doubly linked list " << endl;
   }
   else
   {
      cout << " Elements are : " << endl;
      while (temp != NULL)
      {
         cout << temp -> data << "\t";
         temp = temp -> next;
      }
      cout << endl;
   }
}
```

Sample Output

```
Enter your choice
0. Exit  1. Insert node at end  2. Delete specified node
3. Display
1
Enter the element to be inserted.
45
Enter your choice
1
Enter the element to be inserted
14
Enter your choice
3
Elements are:
45  14
Enter your choice
2
Enter the element to be deleted
45
Enter your choice
2
Enter the element to be deleted
14
Enter your choice
2
Enter the element to be deleted
7
Cant delete
Enter your choice
3
Empty doubly linked list.
```

WEEK-9

19-5P1

Aim: Write a program to implement binary trees using arrays and to perform the following operations traversals i, Inorder ii, Pre Order iii, Post Order.

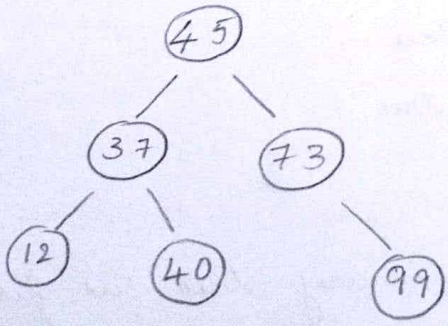
Description:

Binary tree is a non linear data structures and hierarchical representation of nodes where each node contains minimum of 0 children and maximum of 2 children.

The node at the top of the hierarchy of tree is called root node.

Every node in the tree has left and right subtrees.

A parent node contains 2 child node, left, right child nodes, but each child node contains only 1 parent node.




Here 37 is parent node of 12, 40 & 12, 40 are child nodes of node 37.

→ Leaf nodes are external nodes which do not contain any child nodes.

→ In above example 12,40,99 are leaf nodes.

→ Internal nodes are inner nodes which contain atleast 1 node as child node.

→ In above example, 37,73 are internal nodes.

→ Depth of a tree is the number of edges from tree's node to the root.

→ Height of tree is the number of edges from the node to the deepest leaf.

→ Each node in binary tree has a max of 2 children.

→ Types of Binary trees

1. Full Binary Tree
2. Complete Binary tree
3. Perfect Binary tree
4. Balanced Binary tree.
5. Degenerate Binary tree.

* Benefits of Binary trees

* Search Operation in binary tree is faster when compared to other trees.
* Graph traversal uses binary trees.
* It is easy to pick maximum and minimum elements.

Program

```
#include <bits/stdc++.h>
using namespace std;
void root_node(char);
void left_subtree(char, int);
void right_subtree(char, int);
int length_arr();
int get_right(int n);
int get_left(int n);
void inorder(int ind);
int traversal();
void postorder(int ind);
void preorder(int ind);
char tree[100];
int main()
{
    root_node('A');
    int ch, n;
    char c;
    cout << " Enter your choice \n 0. Exit \t 1. Left subtree
    insertion \t 2. Right subtree insertion \t 3. Inorder
    \t 4. post Order \t 5. preorder \t 6. traversal of array
    \n";
    cin >> ch;
    tree[0] = '\0';
```

```
while (ch != 0)
{
    switch (ch)
    {
        case 0: exit (0);
        case 1: cout << " Enter element and parent
                         node index " << endl;
                cin >> c >> n;
                if (ch != '!')
                 left_ subtree(c,n);
                 break;
        case 2: cout << " Enter element and parent
                         node index " << endl;
                cin>>c >>n;
                if (ch != '!')
                right_ subtree (c,n);
                break;
        case 3: cout << " Enter the root node
                         position \n";
                cin >> n;
                cout << " Inorder traversal \n";
                inorder (n);
                break;
        case 4: cout << " Enter the root node position\n";
                cin >> n;
                cout << " Post Order traversal \n";
                postorder (n);
                break;
```

```
        case 5: cout << " Enter the root node position \n";
                cin >> n;
                cout << " Pre Order traversal \n";
                preOrder(n);
                break;
        case 6: cout << " Array traversal \n";
                traversal();
                break;
        default: cout << " Enter correct choice \n";
    }
    cout << "\nEnter your choice \n";
    cin >> ch;
  }
}
void root_node(char c)
{
    if (tree[1] != '\0')
        cout << " Root node is already present \n";
    else {
        tree[1] = c;
    }
}
void left_subtree(char c, int n)
{
    if (tree[n] == '\0')
        cout << " Parent node " << n << " is empty \n";
```

```
        else
            tree [n*2] = c;
}
void right_subtree (char c, int n)
{
    if (tree[n] == '\0')
        cout<<" Parent node is empty \n";
    else
        tree [(n*2)+1] = c;
}
void inorder(int ind)
{
    if (ind >0 && tree [ind] != '\0')
    {
        inorder (get_left(ind));
        cout << tree [ind] <<" \t";
        inorder (get_right(ind));
    }
}
void postorder(int ind){
    if (tree [ind] != '\0' && ind >0)
    {
        postorder (get_left (ind));
        postorder (get_right (ind));
        cout << tree [ind] <<" \t";
    }
}
```

```
void preorder(int ind){
    if(tree[ind]!='\0' && ind>0)
    {
        cout<<tree[ind]<<"\t";
        preorder(get_left(ind));
        preorder(get_right(ind));
    }
}
int get_left(int n)
{
    if(tree[n]!='\0' && n*2<10)
    {
        return n*2;
    }
    return -1;
}
int get_right(int n)
{
    if(tree[n]!='\0' && n*2+1<10)
    {
        return n*2+1;
    }
    return -1;
}
int traversal()
{
```

```
    int i;
    for(i=0; i<20; i++)
      {
        if(tree[i] != '\0')
          cout << tree[i];
        else
            cout << "-";
      }
  }
```

Sample Output

```
Enter your choice
0. Exit  1. Left subtree insertion  2. right subtree insertion
3. Inorder  4. Postorder  5. Pre Order.  6. traversal
1
Enter element and parent node index.
B 1
Enter your choice
2
enter element and parent node index.
C 2
Enter your choice
1
Enter element and parent node index.
D 2
Enter your choice
6
-AB-DC----------
Enter your choice
1
```

Enter element and parent node index

E 4

Enter your choice

6

-AB-DC--E----------

Enter your choice

3

Enter the root node position

1

Inorder traversal

E D B C A Enter your choice

4

Enter the root node position.

1

Post order traversal

E D C B A

Enter your choice

5

Enter the root node position.

1

pre order traversal

A B D E C

Enter your choice

0.

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING Page No.

WEEK-10 19-5P1

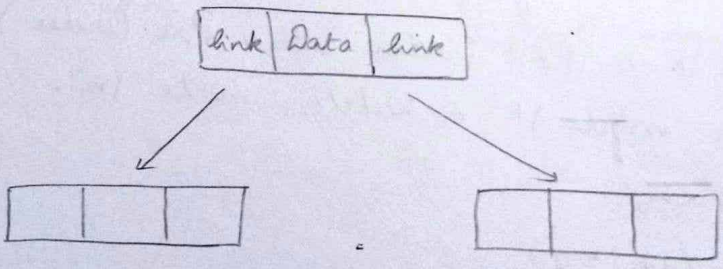
Binary search tree is an extension to binary tree which has following properties.

* Value of left subtree key is less than value of parent node key.
* Value of right subtree key is greater than parent node key.
* Elements of BST are to be in sorted order.

→ Searching, Insertion, Deletion operations are faster when compared to arrays.

Every node of linked list representation of BST contains 3 fields.

* Data field.
* link to left subtree
* link to right subtree.



Program

```
#include <bits/stdc++.h>
using namespace std;
struct bst {
    int data;
    bst *left, *right;
};
bst* insert(bst*, int);
void inorder(bst*);
void postorder(bst*);
void preorder(bst*);
void search(bst*, int);
bst* delete_node(bst*, int);
bst* minimum(bst*)
int main()
{
    int ch, n;
    bst * root = NULL;
    cout<<" Enter your choice \n 0. Exit \n 1. Insertion \t 2.
    Inorder \t 3. Post Order \t 4. Pre Order \t 5.
    Search node \t 6. Delete node \n";
    cin >> ch;
    while (ch!=0)
    {
        switch (ch)
        {
            case 0: exit(1);
            case 1: cout<<"Enter the element to be
                          inserted \n";
```

```
        cin >> ch;
        root = insert(root, ch);
        break;
    case 2: cout << "Inorder traversal \n";
        inorder(root);
        break;
    case 3: cout << "Post order traversal \n";
        postorder(root);
        break;
    case 4: cout << "Pre order traversal \n";
        preorder(root);
        break;
    case 5: cout << "Enter the element to be
        searched \n";
        cin >> ch;
        search(root, ch);
        break;
    case 6: cout << "Enter the element to be
        deleted \n";
        cin >> ch;
        root = delete_node(root, ch);
    }
    cout << "\nEnter your choice \n";
    cin >> ch;
  }
}
```

```
bst * insert ( bst * node, int n)
{
    if (node == NULL)
    {
        bst * newnode = new bst;
        newnode -> data = n;
        newnode -> left = newnode -> right = NULL;
        return newnode;
    }
    if( node -> data > n)
    {
        node -> left = insert( node -> left, n);
    }
    else if ( node -> data < n)
    {
        node -> right = insert ( node -> right, n);
    }
    else
    return node;
}

void inorder( bst * node)
{
    if (node == NULL)
        return ;
    inorder (node -> left);
    cout << node -> data << " ";
    inorder (node -> right);
}
```

```
void postorder(bst* node)
{
    if(node == NULL)
        return;
    postorder(node->left);
    postorder(node->right);
    cout<<node->data<<" ";
}
void preorder(bst* node)
{
    if(node == NULL)
        return;
    cout<<node->data<<" ";
    preorder(node->left);
    preorder(node->right);
}
void search(bst* node, int n)
{
    if(node == NULL)
    {
        cout<<"Element "<<n<<" not found"<<endl;
        return;
    }
    else if(n == node->data)
    {
        cout<<"Element "<<n<<" found"<<endl;
    }
```

```
    else if (n > node -> data)
    {
        search(node -> right, n);
    }
    else if (n < node -> data)
    {
        search(node -> left, n);
    }
}
bst *   minimum (bst * node1)
{
    if (node1 == NULL)
        return node1;
    if (node1 -> left != NULL)
        return minimum (node1 -> left)
    else
        return node1;
}
bst * delete_node( bst * node, int n)
{
    if (node == NULL)
    {
        cout << " Can't Delete \n";
        return node;
    }
    else if (n < node -> data)
        node -> left = delete_node(node -> left, n);
```

```
else if (n> node→data)
{
    node→right = delete_node(node→right,n);
}
else if (node→data==n)
{
    bst* temp;
    if (node→left==NULL && node→right==NULL)
    {
        delete(node)
        node=NULL;
        return node;
    }
else if (node→left==NULL)
{
    temp=node;
    node= node→right;
    delete(temp);
}
else if (node→right==NULL)
{
    temp=node;
    node=node→left;
    delete(temp);
}
```

```
        else {
                temp = minimum (node→right);
node→data = temp→data;
                node→right = delete_node (node→right,
                                          temp→data);
        }
    }
    return node;
}
```

Sample Output

```
Enter your choice
1. Insertion  2. Inorder  3. Postorder  4. PreOrder  5. Search
node  6. Delete node.
1
Enter the element to be inserted.
5
Enter your choice
1
Enter the element to be inserted
4
Enter your choice
1
Enter the element to be inserted
3
Enter your choice
1
Enter the element to be inserted.
8
```

Enter your choice
1
Enter the element to be inserted
9
Enter your choice
2
In Order traversal
3 4 5 8 9
Enter your choice
3
Post order traversal
3 4 9 8 5
Enter your choice
4
Pre Order traversal
5 4 3 8 9
Enter your choice
6
Enter the element to be deleted.
8
Enter your choice
2
Inorder traversal
3 4 5 9
Enter your choice
5
Enter the element to be searched
4
Element 4 found.

## WEEK-11

19131A05P1

Aim: Write a program to perform the following operations using linked lists.

(i) Insert an element into AVL tree

(ii) Delete an element from an AVL tree.

Description

AVL tree is a self balancing binary search tree (BST) where the difference between heights of left and right subtrees can't be more than one for all nodes.

Named after the inventor Adelson, Velski & Landis.

Program

```
#include <iostream>
#include <stdlib.h>
using namespace std;
struct avl {
    int data;
    avl* left, * right;
};
avl* insert_node (avl* node, int n);
avl* balance_node (avl* node1);
int max(int, int);
```

```
int height(avl*);
avl* left_rotation(avl*);
avl* right_rotation(avl*);
avl* left_right_rotation(avl*);
avl* right_left_rotation(avl*);
void inorder(avl*);
void preorder(avl*);
void postorder(avl*);
avl* delete_node(avl* node, int n);
avl* minimum(avl*);

int main()
{
    avl* root = NULL;
    int ch, n;
    cout<<"Enter your choice \n 0.Exit \t 1. Insert \t 2. Inorder
    \t 3. PreOrder \t 4. Postorder \t 5. Delete \n";
    cin>>ch;
    while(ch!=0)
    {
        switch(ch)
        {
            case 0: exit(0);
            case 1: cout<<"Enter the element to be
                    inserted \n";
                    cin>>n;
                    root = insert_node(root, n);
                    break;
```

```
        case 2 : cout << "Inorder traversal\n";
                 inorder ( root );
                 break;
        case 3 : cout << " Preorder traversal \n";
                 preorder (root);
                 break;
        case 4 : cout << " Postorder traversal \n";
                 postorder ( root );
                 break;
        case 5: cout << " Enter the element to be deleted\n";
                cin >> n;
                root = delete_node ( root , ch );
                break;
        default: cout << " Enter proper choice \n";
    }
    cout << "\nEnter your choice \n";
    cin >> ch;
  }
}
int max ( int a, int b)
{
    return a>b ? a:b;
}
int height ( avl* node)
 {
```

```
    int h=0;
    if (node == NULL)
       return 0;
    if (node != NULL)
      {
         int lh = height (node -> left);
         int rh = height (node -> right);
         h = max (lh, rh);
      }
      return h+1;
}
int balancing_factor (avl * node)
{
    return (height (node -> left) - height (node -> right));
}
avl * left_rotation (avl * node1)
{
    avl * temp;
    temp = node1 -> left;
    node1 -> left = temp -> right;
    temp -> right = node1;
    return temp;
}
avl * right_rotation (avl * node1)
{
    avl* temp;
    temp = node1 -> right;
```

```
        node1 -> right = temp -> left;
        temp -> left = node1;
        return temp;
    }
avl *   left_right_rotation (avl * node1)
{
    avl * temp;
    temp = node1 -> left;
    node1 -> left = right_rotation (temp);
    return left_rotation (temp);
}
avl *   right_left_rotation (avl * node1)
{
    avl * temp;
    temp = node1 -> right;
    node1 -> right = left_rotation (temp);
    return right_rotation (temp);
}
avl *   insert_node ( avl * node, int n)
{
    if (node == NULL)
    {
        node = new avl;
        node -> data = n;
        node -> left = node -> right = NULL;
```

```
    return node;
  }
  else if (n> node → data)
  {
     node → right = insert_node(node → right, n);
     node = balance_node(node);
  }
  else if (n < node → data)
  {
     node → left = insert_node(node → left, n);
     node = balance_node(node);
  }
  return node;
}

avl* balance_node(avl* node) {
  int bf = balancing_factor(node);
  avl* temp;
  if (bf < -1)
  {
     if (balancing_factor(node → right > 0)
        node = right_left_rotation(node);
     else
        node = right_rotation(node);
  }
  else if (bf > 1) {
     if (balancing_factor(node → left > 0)
        node = left_rotation(node);
     else
        node = left_right_rotation(node);
  }
```

```
    return node1;
}
void inorder(avl * node1) {
    if (node1 != NULL)
    {
        inorder(node1->left);
        cout << node1->data << " ";
        inorder(node1->right);
    }
}
void preorder(avl * node1) {
if (node1 != NULL)
    { cout << node1->data << " ";
      preorder(node1->left);
      preorder(node1->right);
    }
}
void postorder(avl * node1) {
    if (node1 != NULL)
     {
         postorder(node1->left);
         postorder(node1->right);
         cout << node1->data <<" ";
     }
}
```

```
avl *   delete_node (avl * node, int n)
{
      avl * temp;
      if (node == NULL)
       {
          return node;
       }
       if (n > node → data)
          node → right = delete_node (node → right, n);
        else if (n < node → data)
            node → left = delete_node (node → left, n);
        else {
            if (node → left && node → right)
            {
               avl * temp = minimum (node → right);
               node → data = temp → data;
               node = delete_node (node_right, n);
            }
            else
            {
               avl * temp = node;
               if (node → left == NULL)
                        node = node → right;
               else if (node → right == NULL)
                        node = node → left;
               delete (temp);
            }
      }
```

```
int bf = balancing_factor(node);
if (bf > 1 && balancing_factor(node -> left) >= 0)
        return left_rotation (node);
if (bf > 1 && balancing_factor( node -> left) < 0)
        return left_right_rotation (node);
if (bf < -1 && balancing_factor (node -> right) <= 0)
        return right_left_rotation (node);
if (bf < -1 && balancing_factor (node -> right) > 0)
        return right_rotation (node);
return node;
}

avl * minimum (avl * node)
{
    if (node1 == NULL)
        return node1;
    if (node1 -> left != NULL)
        return minimum (node1 -> left);
    else
        return node1;
}
```

Sample Output

Enter your choice
1. Insert 2. Inorder 3. PreOrder 4. PostOrder 5. Delete.
1
Enter the element to be inserted
5
Enter your choice
1
Enter the element to be inserted
3
Enter your choice
1
Enter the element to be inserted
7
Enter your choice
1
Enter the element to be inserted.
4
Enter your choice
1
Enter the element to be inserted
2
Enter your choice
2
Inorder traversal
2 3 4 5 7
Enter your choice
3
PreOrder traversal
5 3 2 4 7

Enter your choice

4

Post order traversal

2 4 3 7 5

Enter your choice

6

Enter the element to be deleted.

2

Enter your choice

2

Inorder traversal

3 4 5 7

Enter your choice

0

# WEEK-12

19-5P1

Aim:- Write a program for implementation of BFS, DFS for a given graph.

Description

Graph is a non-linear data structure consisting of nodes, edges. Nodes are referred as vertices and edges are lines connecting 2 vertices.

BFS Breadth first search

BFS is a vertex based technique for finding a shortest path in graph. It uses queue data structure.

DFS Depth first search

DFS is an edge based technique which uses stack data structure.

Program

```
#include <bits/stdc++.h>
using namespace std;
void add_edge(vector<int> adj[], int a, int b)
{
    adj[a].push_back(b);
}
```

```
void bfs(int n, vector<int> arr[], bool visit[])
{
    queue<int> q;
    q.push(n);
    visit[n] = true;
    while(!q.empty())
    {
        cout << q.front() << " ";
        int u = q.front();
        q.pop();
        int i;
        for(i=0; i<arr[u].size(); i++)
        {
            if(!visit[arr[u][i]])
            {
                q.push(arr[u][i]);
                visit[arr[u][i]] = true;
            }
        }
    }
}
void dfs(int n, vector<int> arr[], bool visit[])
{
    stack<int> q;
    q.push(n);
    visit[n] = true;
```

```
    while(!q.empty())
    {
        cout << q.top() << " ";
        int u = q.top();
        q.pop();
        int i;
        for(i=0;i< arr[u].size(); i++)
        {
            if(!visit[arr[u][i]])
            {
                q.push(arr[u][i]);
                visit[arr[u][i]] = true;
            }
        }
    }
}
int main()
{
    int n, i, ch, a, b;
    cout << "No. of edges \n";
    cin >> n;
    vector <int> arr[n];
    bool visit[n];
    for(i=0; i<n; i++)
        visit[i] = false;
```

```
cout << " Enter your choice \n 1. add an edge \t
2. bfs traversal \t 3. dfs traversal \t 0. Exit \n";
cin >> ch;
while ( ch != 0 )
{
        switch (ch)
        {
            case 0 : exit (1);
            case 1 : cout << " Data a, b \n";
                     cin >> a>>b;
                     add_edge (arr, a, b);
                     break;
           case 2 : for (i = 0; i < n; i++)
                     visit [i] = false ;
                     cout << " edge choice \n";
                     cin >> ch;      cout << "BFS traversal \n";
                     bfs (ch, arr, visit );
                     break;
           case 3 : for (i = 0 ; i < n; i++)
                      visit [i] = false ;
                     cout << " Edge choice \n",
                     cin >> ch;   cout << " DFS traversal \n";
                     dfs ( ch, arr, visit );
                     break;
```

```
            default : cout << " Enter proper choice \n";
        }
        cout << " \n Enter your choice \n";
        cin >> ch;
    }
}
```

Sample Output

```
No. of edges
3
Enter your choice
1. add an edge   2. bfs traversal   3. dfs traversal   0. Exit.
1
data a,b
0
2
Enter your choice
1
data a,b
2
1
Enter your choice
1
data a,b
1
2
Enter your choice
1
```

```
data a,b
1
0
Enter your choice
2
edge choice
0
BFS traversal
0 2 1
Enter your choice
2
Edge choice
1
BFS traversal
1 2 0
Enter your choice
3
Edge choice
2
DFS traversal
2 1 0
Enter your choice
0
```