UNIT-3 Programs

BINARY TREE(LINKED LIST):

```
#include<iostream>
using namespace std;
template<class T>
struct node
{
        public:
        T data;
        node *left,*right;
};
template<class T>
class binarytree
{
        public:
   node<T> *root,*temp;
   binarytree()
   {       root=NULL;temp=NULL;    }
        void search(node<T>*,T);
        void insert();
        void inorder(node<T>*);
        void preorder(node<T>*);
        void postorder(node<T>*);
        void display();
};
template<class T>
void binarytree<T>::search(node<T> *p, T x)
{
        if(p!=NULL)
      {
        if(p->data==x)
        {
                temp=p;
        cout<<"\n Element "<<p->data<<" is found \n";
        }
                        search(p->left,x);
        search(p->right,x);
      }
}
template<class T>
```

```
void binarytree<T>::insert()
{
   temp=NULL; // Search function will update this variable when element is found
   if(root==NULL)
   {
      root=new node<T>;
      cout<<"\n enter value \n";
      cin>>root->data;
      root->left=NULL;
      root->right=NULL;
      cout<<"\n value inserted successfully \n ";
   }
   else
   {
      cout<<"\n Enter the parent value, where you wanted to insert the value \n";
      T x;
      cin>>x;
      node<T> *p;
      search(root,x); // this method updates temp variable, if ele is found
      if(temp==NULL)
          cout<<"\n ELement Not found \n";
      else
      {
                              p=temp;
          if(p->left!=NULL && p->right!=NULL)
                    cout<<"\nInsertion Failed "<<p->data<<" have both childs,choose other
parent\n";
          else
                                          {
                     cout<< "\n Enter 1 for left , 2 for right";
                   int ch;
                cin>>ch;
                if(ch==1)
                {
                   if(p->left!=NULL)
                     cout<<"\n Left is not free \n";
                   else
                   {
                     p->left=new node<T>;
                     p=p->left;
                     cout<<"\n enter value \n";
                     cin>>p->data;
                     p->left=NULL;
                     p->right=NULL;
```

```
                cout<<"\n value inserted successfully \n ";
              }
            }
            else if(ch==2)
            {
              if(p->right!=NULL)
              cout<<"\n Right is not free \n";
              else
              {
                p->right=new node<T>;
                p=p->right;
                cout<<"\n enter value \n";
                cin>>p->data;
                p->left=NULL;
                p->right=NULL;
                cout<<"\n value inserted successfully \n ";
              }
            }
            else
            {    cout<<"\n Not a valid input enter 1 or 2\n";return;}
          }
      }
    }

}


template<class T>
 void binarytree<T>::display()
 {
   cout<<"\n1.Inorder   2.Preorder   3.Postorder  \nEnter Your Choice:";
   int ch;
   cin>>ch;
   switch(ch)
   {
   case 1:
      inorder(root);
      break;
   case 2:
      preorder(root);
      break;
   case 3:
      postorder(root);
```

```
   }
 }
template<class T>
void binarytree<T>::inorder(node<T> *p)
{
   if(p==NULL)
      return;
   inorder(p->left);
   cout<<"   "<<p->data;
   inorder(p->right);

}
template<class T>
void binarytree<T>::preorder(node<T> *p)
{
   if(p==NULL)
      return;
   cout<<"  "<<p->data;
   preorder(p->left);
   preorder(p->right);

}
template<class T>
void binarytree<T>::postorder(node<T> *p)
{
   if(p==NULL)
      return;
   postorder(p->left);
   postorder(p->right);
   cout<<"   "<<p->data;


}
int main()
{
   binarytree<char> ob;
   int ch;
   while(1)
   {
         cout<<"\n1.INSERT  2.DISPLAY  3.EXIT \n Enter YOUR Choice";
         cin>>ch;
         switch(ch)
         {
                        case 1:
```

```
                        ob.insert();
                        break;
                        case 2:
                        ob.display();
                        break;
                        case 3:
                        return 0;
        }
        }
}
```

BINARY TREE(VECTOR-BASED):

```
#include<iostream>
#include<cmath>
using namespace std;
template<class T>
class binarytree
{
        public:
   T *V,C;
        int l,ms,temp;
   binarytree()
   {
                cout<<"\nEnter Maximum No of levels:";
                cin>>l;
                ms=pow(2,l)-1;
                V=new T [ms];  //space created for maximum elements in l levels
                cout<<"\nenter global constant to intialize the vector";
                cin>>C;
                for(int i=0;i<ms;i++)
                        V[i]=C;
        }
        void search(int key, int i);//key is the element to search and i is the index of the root
node
        void insert();
        void inorder(int i);  //i is index of the root node
        void preorder(int i);
        void postorder(int i);
        void display();
};
template<class T>
void binarytree<T>::search(int key,int i)
{
```

```
        if(V[i]!=C)
    {
        if(V[i]==key)
        {
                temp=i;
        cout<<"\n Element "<<V[i]<<" is found \n";
                                return;
        }
        if(2*i+1 < ms) search(key,2*i+1);
        if(2*i+2 < ms) search(key,2*i+2);
    }

}
template<class T>
void binarytree<T>::insert()
{
  temp=-1; // Search function will update this variable when element is found
  if(V[0]==C)
  {
    cout<<"\n enter value \n";
    cin>>V[0];
    cout<<"\n value inserted successfully \n ";
  }
  else
  {
    cout<<"\n Enter the parent value, where you wanted to insert the value \n";
    T x;
    int p;
    cin>>x;
    search(x,0); // this method updates temp variable, if ele is found
    if(temp==-1)
        cout<<"\n Element Not found \n";
    else
    {
                cout<<endl<<temp;
                        p=temp;
        if(V[2*p+1]!=C && V[2*p+2]!=C)
                cout<<"\nInsertion Failed "<<V[p]<<" have both childs,choose other
parent\n";
        else
                                {
                  cout<< "\n Enter 1 for left , 2 for right";
                int ch;
            cin>>ch;
```

```
            if(ch==1)
            {
               if(V[2*p+1]!=C)
                 cout<<"\n Left is not free \n";
               else
               {
                 cout<<"\n enter value \n";
                 cin>>V[2*p+1];
                 cout<<"\n value inserted successfully \n ";
               }
            }
            else if(ch==2)
            {
               if(V[2*p+2]!=C)
                 cout<<"\n Right is not free \n";
               else
               {
                 cout<<"\n enter value \n";
                 cin>>V[2*p+2];
                 cout<<"\n value inserted successfully \n ";
               }
            }
            else
            {    cout<<"\n Not a valid input enter 1 or 2\n";return;}
         }
      }
   }

}


template<class T>
 void binarytree<T>::display()
 {
   cout<<"\n1.Inorder   2.Preorder   3.Postorder  \nEnter Your Choice:";
   int ch;
   cin>>ch;
   switch(ch)
   {
   case 1:
      inorder(0);
      break;
   case 2:
      preorder(0);
```

```
      break;
   case 3:
      postorder(0);

   }
 }
template<class T>
void binarytree<T>::inorder(int i)
{
   if(V[i]!=C)
          {
             if(2*i+1 < ms) inorder(2*i+1);
          cout<<"   "<<V[i];
          if(2*i+2 < ms) inorder(2*i+2);
          }
}
template<class T>
void binarytree<T>::preorder(int i)
{
   if(V[i]==C)
      return;
   cout<<"  "<<V[i];
   if(2*i+1 < ms) preorder(2*i+1);
   if(2*i+2 < ms) preorder(2*i+2);


}
template<class T>
void binarytree<T>::postorder(int i)
{
   if(V[i]==C)
      return;
   if(2*i+1 < ms) postorder(2*i+1);
          if(2*i+1 < ms) postorder(2*i+2);
          cout<<"   "<<V[i];


}
int main()
{
   binarytree<char> ob;
   int ch;
   while(1)
   {
```

```
        cout<<"\n1.INSERT  2.DISPLAY  3.EXIT \n Enter YOUR Choice";
        cin>>ch;
        switch(ch)
        {
                        case 1:
                        ob.insert();
                        break;
                        case 2:
                        ob.display();
                        break;
                        case 3:
                        return 0;
        }
        }
}
```

HEIGHT OF BINARY TREE:

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

template<class T>
struct node
{
        public:
        T data;
        node *left,*right;
};

template<class T>
class binarytree
{
        public:
   node<T> *root,*temp;
   binarytree()
   {       root=NULL;temp=NULL;    }
        void search(node<T>*,T);
        void insert();
        int height(node<T>*);
};

template<class T>
```

```
void binarytree<T>::search(node<T> *p, T x)
{
        if(p!=NULL)
    {
        if(p->data==x)
        {
                temp=p;
        cout<<"\n Element "<<p->data<<" is found \n";
        }
                        search(p->left,x);
        search(p->right,x);
    }
}
template<class T>
void binarytree<T>::insert()
{
  temp=NULL; // Search function will update this variable when element is found
  if(root==NULL)
  {
     root=new node<T>;
     cout<<"\n enter value \n";
     cin>>root->data;
     root->left=NULL;
     root->right=NULL;
     cout<<"\n value inserted successfully \n ";
  }
  else
  {
     cout<<"\n Enter the parent value, where you wanted to insert the value \n";
     T x;
     cin>>x;
     node<T> *p;
     search(root,x); // this method updates temp variable, if ele is found
     if(temp==NULL)
        cout<<"\n ELement Not found \n";
     else
     {
                        p=temp;
        if(p->left!=NULL && p->right!=NULL)
                cout<<"\nInsertion Failed "<<p->data<<" have both childs,choose other
parent\n";
        else
                                {
                 cout<< "\n Enter 1 for left , 2 for right";
```

```
                int ch;
            cin>>ch;
            if(ch==1)
            {
              if(p->left!=NULL)
                cout<<"\n Left is not free \n";
              else
              {
                p->left=new node<T>;
                p=p->left;
                cout<<"\n enter value \n";
                cin>>p->data;
                p->left=NULL;
                p->right=NULL;
                cout<<"\n value inserted successfully \n ";
              }
            }
            else if(ch==2)
            {
              if(p->right!=NULL)
              cout<<"\n Right is not free \n";
              else
              {
                p->right=new node<T>;
                p=p->right;
                cout<<"\n enter value \n";
                cin>>p->data;
                p->left=NULL;
                p->right=NULL;
                cout<<"\n value inserted successfully \n ";
              }
            }
            else
            {    cout<<"\n Not a valid input enter 1 or 2\n";return;}
          }
      }
    }

}



// Recursive function to calculate height of given binary tree
template<class T>
```

```
int binarytree<T>::height(node<T>* root)
{
   // Base case: empty tree has height 0
   if (root == NULL)
      return 0;

   // recur for left and right subtree and consider maximum depth
   return 1 + max(height(root->left), height(root->right));
}

int main()
{
   binarytree<char> ob;
   int ch;
   while(1)
   {
         cout<<"\n1.INSERT  2.HEIGHT  3.EXIT \n Enter YOUR Choice: ";
         cin>>ch;
         switch(ch)
         {
                        case 1:
                        ob.insert();
                        break;
                        case 2:
                        cout<<ob.height(ob.root);
                        break;
                        case 3:
                        return 0;
         }
         }
}
```

ALGORITHMS FOR TREE TRAVERSAL TECHNIQUES:

Inorder Traversal:

Algorithm Inorder(tree)

1. Traverse the left subtree, i.e., call Inorder(left-subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right-subtree)

Preorder Traversal:

Algorithm Preorder(tree)

1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder(left-subtree)
3. Traverse the right subtree, i.e., call Preorder(right-subtree)

Postorder Traversal:

Algorithm Postorder(tree)

1. Traverse the left subtree, i.e., call Postorder(left-subtree)
2. Traverse the right subtree, i.e., call Postorder(right-subtree)
3. Visit the root.

Evaluating an Arithmetic Expression

```
Algorithm evaluateExpression(T, p):
if p is an internal node then
        x ← evaluateExpression(T, p.left())
        y ← evaluateExpression(T, p.right())
        Let ◦ be the operator associated with p return x ◦ y
else
        return the value stored at p
```

Printing an Arithmetic Expression

```
Algorithm printExpression(T, p):
if p.isExternal() then
        print the value stored at p
else
        print “(”
        printExpression(T, p.left())
        print the operator stored at p
        printExpression(T, p.right())
        print “)”
```

ALGORITHMS FOR PATTERN MATCHING TECHNIQUES:

BRUTE FORCE:

```
Algorithm BruteForceMatch(T,P):
    Input: Strings T (text) with n characters and P (pattern) with m
    characters
    Output: Starting index of the first substring of T matching P,
    or an indication that P is not a substring of T
    for i ← 0 to n−m {for each candidate index in T} do
        j ← 0
    while (j < m and T[i+ j] = P[ j]) do
        j ← j +1
    if j = m then
        return i
    return "There is no substring of T matching P."
```

BOYER MOORE:

```
Algorithm BMMatch(T,P):
    Input: Strings T (text) with n characters and P (pattern) with m
    characters
    Output: Starting index of the first substring of T matching P,
    or an indication that P is not a substring of T
    compute function last
    i ← m−1
    j ← m−1
    repeat
        if P[ j] = T[i] then
            if j = 0 then
                return i {a match!}
            else
                i ← i−1
                j ← j −1
        else
            i ← i+m−min(j,1+last(T[i])) {jump step}
            j ← m−1
```

```
    until i > n−1
    return “There is no substring of T matching P.”
```

KNUTH-MORRIS-PRATT Algorithm:

```
Algorithm KMPMatch(T,P):
    Input: Strings T (text) with n characters and P (pattern) with m
    characters
    Output: Starting index of the first substring of T matching P, or
    an indication that P is not a substring of T
    f ← KMPFailureFunction(P) {construct the failure function f for
    P}
    i ← 0
    j ← 0
    while i < n do
        if P[ j] = T[i] then
            if j = m−1 then
                return i−m+1 {a match!}
            i ← i+1
            j← j +1
        else if j > 0 {no match, but we have advanced in P} then
            j ← f(j −1) { j indexes just after prefix of P that
        must match}
        else
            i ← i+1
    return “There is no substring of T matching P.”
```

Constructing the KMP Failure Function:

```
Algorithm KMPFailureFunction(P):
    Input: String P (pattern) with m characters
    Output: The failure function f for P, which maps j to the length
    of the longest prefix of P that is a suffix of P[1.. j]
    i ← 1
    j ← 0
    f(0) ← 0
    while i < m do
```

```
            if P[ j] = P[i] then
                {we have matched j +1 characters}
                f(i) ← j +1
                i ← i+1
                j ← j +1
            else if j > 0 then
                { j indexes just after a prefix of P that must
    match}
                j ← f(j −1)
            else
                {we have no match here}
                f(i) ← 0
                i ← i+1
```

TRIES:

```
#include<iostream>
#include<stdlib.h>
#include<cstring>
using namespace std;
class trie;//forward declaration
class node
{
    node *child[58];
    //Can accecpt integers,Capital alphabets and small alphabets
    bool end;
    //used to mark the end of particular word
    friend class trie;
    public:
        node()
        {
            for(int i=0;i<58;++i)
            child[i]=NULL;
            end=false;
            //initializing all child elements to NULL
        }
};
```

```
class trie
{
	node *root;
	public :
		trie()
		{
			root=new node;
			root->end=false;
		}
		void insert();//insert new word into Trie
		bool search(string);//search for word in Trie if present return true
else false
		void display();//we can access the root directly so we use Driver
		void displayDriver(node *, char *, int);//displays every word or
charcter in Trie
};

void trie::insert()
{
	node *t=root;
	string data;
	int i=0,c;
	cout<<"Enter the string to be inserted... \n";
	getchar();
	cin>>data;
	//checking how much of the new string is already present
	while(t->child[data[i]-'A']!=NULL&&data[i]!='\0')
	{
		t=t->child[data[i]-'A'];
		++i;
	}
   //if string to be inserted is already present we don't go to below while loop
	//if only a part of it there then we add the remaining
	while(i<strlen(&data[0]))
	{
		node *temp=new node;
		t->child[data[i]-'A']=temp;
```

```
                ++i;
                temp->end=false;
                t=temp;
        }
        //atlast after adding new string we mark the end node's end as true
        t->end=true;
}

bool trie::search(string data)
{
        node *t=root;
        bool b=false,flag=false;
        int c,j=0;
        int i=strlen(&data[0]);
        while(t->child[data[j]-'A']!=NULL&&i!=0)
        {
                --i;
                t=t->child[data[j]-'A'];
                b=t->end;
                ++j;
        }
        if(i==0&&b)//checking whether we matched string's end
        //present in Trie with end of search string
        {
                flag=true;
        }
        return flag;
}
void trie::display()
{
        int l=0;
        char str[100];
        displayDriver(root,str,l);
}
void trie::displayDriver(node *r,char *str,int l)
{
        if(r->end==true)//checking for end of word
```

```
	{
		str[l]='\0';
		cout<<str<<"\n";
		//we stop calling once end is found
	}
	int i;
	for(i=0;i<58;i++)
	{
		if(r->child[i])
		{
			str[l]=i+'A';
			displayDriver(r->child[i],str,l+1);//recursively calling
		}
	}
}
int main()
{
	trie t;
	node p;
	int n;
	do
	{
		cout<<"\n1.Insert \n2.Search \n3.Display \n4.Exit\n";
		cout<<"enter choice\n";
		cin>>n;
		switch(n)
		{
			case 1:t.insert();break;
			case 2:
			{
				string data;
				cout<<"enter the string to be searched...\n";
				cin>>data;
				bool b=t.search(data);
				if(b==false)
					cout<<"Not Found"<<endl;
				else
```

```
                                cout<<"Found"<<endl;
                        break;
                    }
                    case 3:t.display();break;
                    case 4:
                        break;
                    default:
                        cout<<"wrong choice"<<endl;
                        break;
                }
        }while(n!=4);
        return 0;
}
```