Date

Assignment - 3

Title : Construct an expression tree from the given prefix expression. eg: t--a*bcdef and transverse it using post order transversed (non-recursive) and then delete the entire tree.

Objectives:

* fundamental data storage structure used in programming.
* Combines advantages of an ordered array and a linked list.
* Searching as fast as in ordered array
* Insertion and deletion as fast as in linked list.

Theory :

Expression Trees :

When an expression is represented through a tree, it is known as an expression tree. The leaves of an expression tree are operand such as constant or variable names and all internal node contain operations.

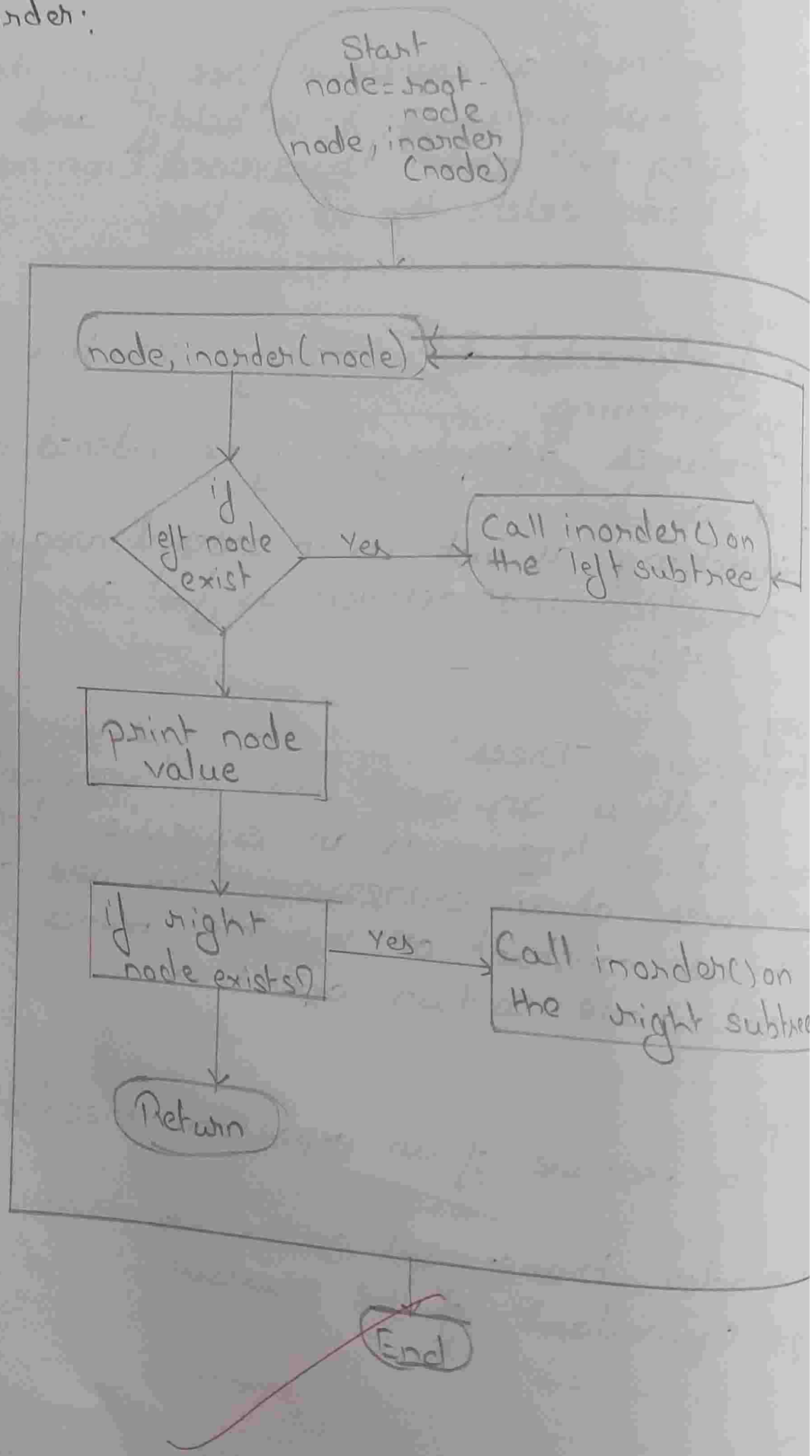
Example :

An example of an expression tree (a+b+c)+e+f

A preorder transversal on the expression three given prefix equivalent of the expression

Flowchart:

Inorder:



Date

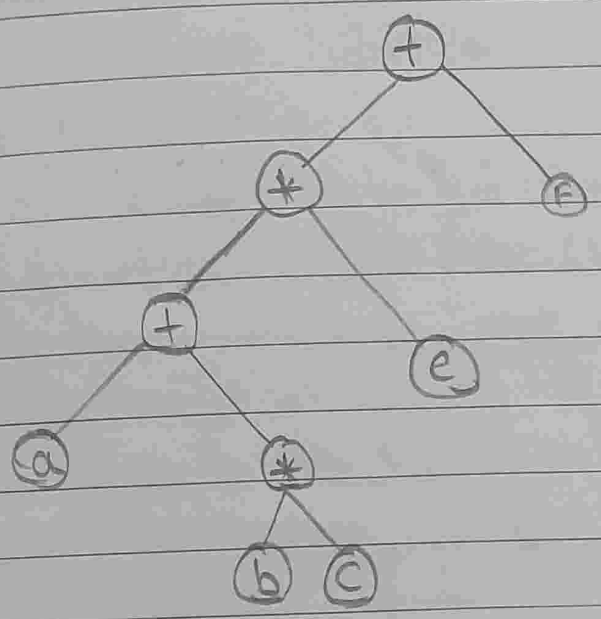
Prefix Expression:

= ++ +a* bcef

Prfix Expression:

= abc* +e*+f

Expression tree:



Algorithm:

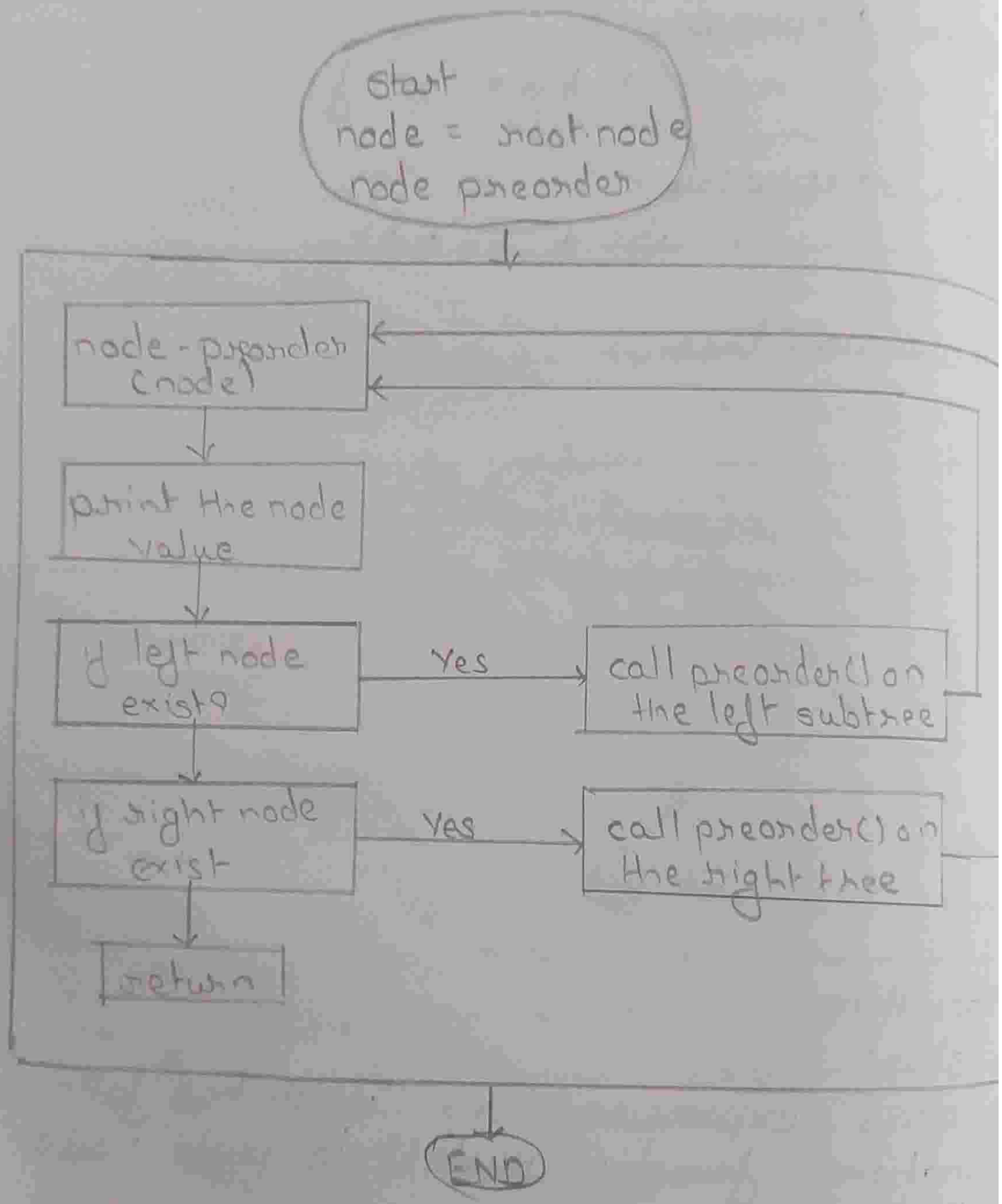
1. start

ii. class expression tree which has following functions. function push () to push node into that tree:

If stack is null
then push the node as first element.
Else
push the node and make it top

iii. function pop() to pop out nodes from the tree:

Preorder :


Start
node = root node
node preorder
node - preorder (node)
print the node value
if left node exists
Yes
call preorder() on the left subtree
if right node exist
Yes
call preorder() on the right tree
return
END

If stack is null
then print underflow
Else:
pop out the node and update top.

iv. function Insert () to insert character
If it is digit
then push it
Else if it is operator
Then pop it
Else
print "invalid expression".

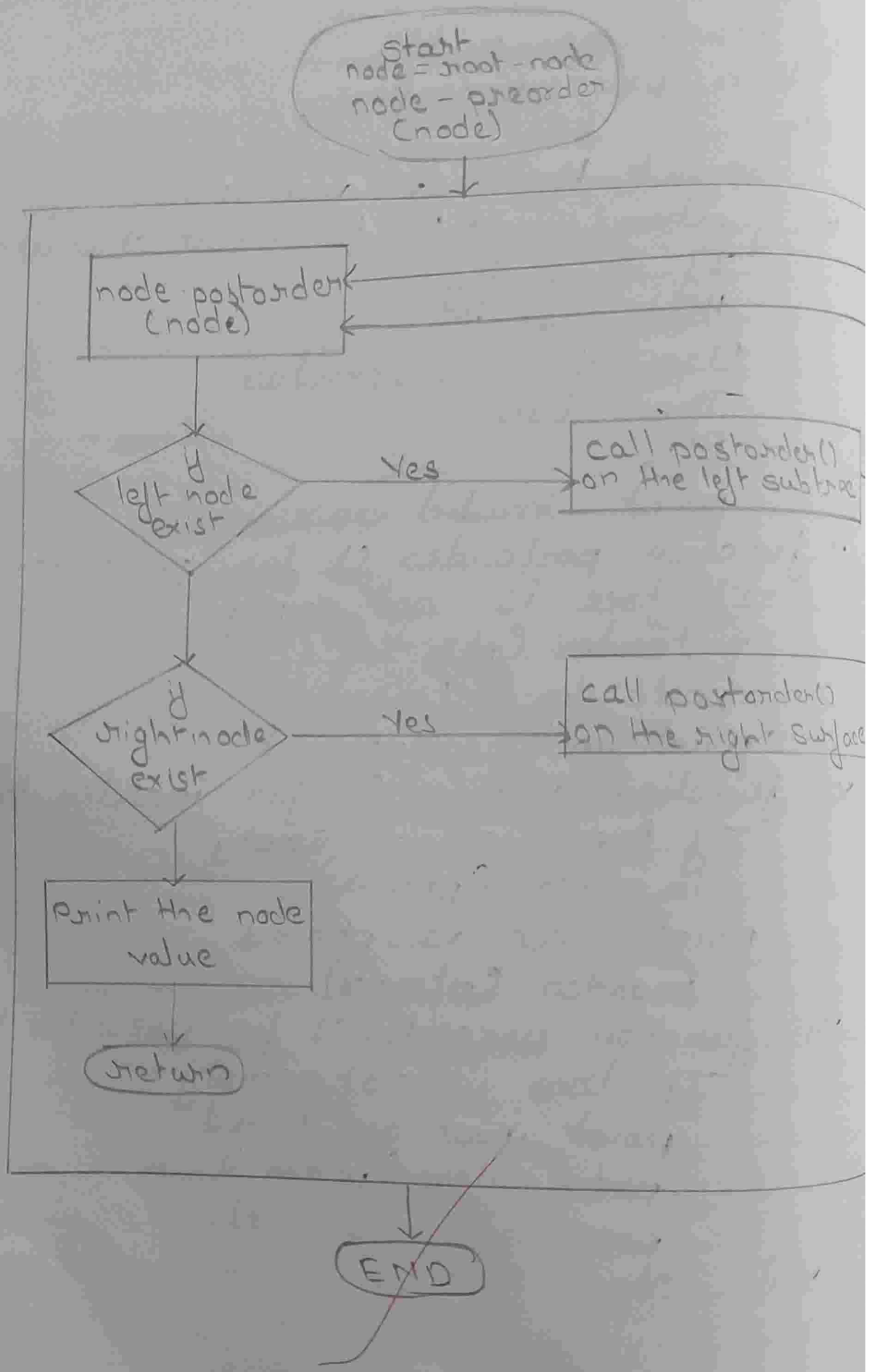
v. function postorder () for postorder transversal
If tree is not empty
postorder (ptr → l)
postorder (ptr → r)
print root as ptr → d

vi. function inorder () for inorder traversal:
If tree is not empty
in order (ptr → l)
print root as ptr → d
inorder (ptr → r)

vii. function preorder () for preorder traversal
If tree is not empty.
print root as ptr → d
pre-order (ptr → l)
pre-order (ptr → r)

viii) stop

Postorder:


start
node = root - node
node - preorder
(node)
node postorder
(node)
if
left node
exist
Yes
call postorder()
on the left subtree
if
right node
exist
Yes
call postorder()
on the right subtree
Print the node
value
return
END

INPUT :

```
#include <iostream>

using namespace std;

#include<string.h>

struct node

        {

        char data;

        node *left;

        node *right;

        };

class tree

{       char prefix[20];

        public: node *top;

                void expression(char []);

                void display(node *);

                void non_rec_postorder(node *);

                void del(node *);

};

class stack1

  {

        node *data[30];

        int top;

        public:

        stack1()

        {
```

```
            top=-1;

    }
            int empty()
            {
                if(top==-1)
                    return 1;
                return 0;
            }
    void push(node *p)
            {
                data[++top]=p;
            }
    node *pop()
    {
            return(data[top--]);
    }
};
void tree::expression(char prefix[])
{char c;
stack1 s;
node *t1,*t2;
int len,i;
len=strlen(prefix);
        for(i=len-1;i>=0;i--)
        {top=new node;
```

```
                top->left=NULL;
                top->right=NULL;
            if(isalpha(prefix[i]))
            {
                    top->data=prefix[i];
                s.push(top);
            }
            else if(prefix[i]=='+'||prefix[i]=='*'||prefix[i]=='-'||prefix[i]=='/')
            {
            t2=s.pop();

            t1=s.pop();
                top->data=prefix[i];
                top->left=t2;
                top->right=t1;
                s.push(top);
    }
                }
        top=s.pop();
}
void tree::display(node * root)
{
        if(root!=NULL)
        {               cout<<root->data;
            display(root->left);
```

```
                display(root->right);

        }

}
void tree::non_rec_postorder(node *top)
 {      stack1 s1,s2;   /*stack s1 is being used for flag  A NULL data
                        implies that the right subtree has not been visited */

        node *T=top;

        cout<<"\n";

        s1.push(T);

while(!s1.empty())

{

T=s1 pop();

s2.push(T),

if(T->left!=NULL)

s1.push(T->left);

if(T->right!=NULL)

s1.push(T->right);

 }

while(!s2.empty());

{

top=s2.pop();

cout<<top->data;

}}

 void tree::del(node* node)

{
```

```
if (node == NULL) return;
 /* first delete both subtrees */
del(node->left);
del(node->right);
   /* then delete the node */
 cout<<" Deleting node:"<<node->data;
 free(node);
}
int main()
{
        char expr[20];
        tree t;
        cout<<"Enter prefix  Expression: ";
        cin>>expr;
        cout<<expr;
        t.expression(expr);
  //t.display(t.top);
  //cout<<endl;
  t.non_rec_postorder(t.top);
  // t.del(t.top);
  // t.display(t.top);
   }
```

Sinhgad Institutes

Date

conclusion -

Thus, we have successfully completed Expression tree from given prefix using C++.