DSPLAB Assignment - 4

1. Create a Binary tree, calculate Height , print inorder, pre and post order Traversals.

Source code:

#include <stdio.h>

#include <stdlib.h>

int count=0;

struct Node

{

int key;

struct Node \*left, \*right;

};

void inorderTraversal(struct Node\* root)

{

if (root == NULL) {

return;

}

inorderTraversal(root->left);

printf("%d ", root->key);

inorderTraversal(root->right);

}

void preorderTraversal(struct Node\* root)

{

if (root == NULL) {

return;

}

printf("%d ", root->key);

preorderTraversal(root->left);

preorderTraversal(root->right);

}

void postorderTraversal(struct Node\* root)

{

if (root == NULL)

return;

postorderTraversal(root->left);

postorderTraversal

(root->right);

printf("%d ", root->key);

}

struct Node \*new\_node(int value)

{

struct Node \*tmp = (struct node \*)malloc(sizeof(struct Node));

tmp->key = value;

tmp->left = tmp->right = NULL;

return tmp;

}

int max(int a,int b){

if(a>b)

return a;

else

return b;

}

void printvalue(struct Node\* root,int l,int h){

if(root==NULL){

printf(" null");

return;

}

if(l==1){

int i=0;

for(i=0;i<h;i++)

printf(" ");

printf("%s ",root->key);

}

else if(l >1){

printvalue(root->left,l-1,h);

printvalue(root->right,l-1,h);

}

}

void display(struct Node\* root,int h){

int i,k;

for(i=1;i<=h;i++){

for(k=i;k<h;k++)

printf(" ");

printvalue(root,i,h);

printf("\n");

}

}

int height(struct Node\* node){

if(node == NULL)

return 0;

else{

int leftheight=height(node->left);

int rightheight=height(node->right);

return max(leftheight,rightheight)+1;

}

}

struct Node\* insert\_node(struct Node\* node, int value) // inserting nodes!

{

if (node == NULL) return new\_node(value);

if (value < node->key)

{

node->left = insert\_node(node->left, value);

}

else if (value > node->key)

{

node->right = insert\_node(node->right, value);

}

return node;

}

int main()

{

int n,value,k;

int i=0,choice=0;

struct Node \*root\_node = NULL;

while (choice!=7)

{

printf("\n 1.Insert\n 2.Height\n 3.Display\n 4.Preorder\n 5.Inorder\n 6.Postorder\n 7.Exit\nEnter your choice : ");

if(scanf("%d", &choice)==1){

if(choice==1){

printf("\nEnter Number nodes in a Tree:");

scanf("%d",&n);

printf("\nEnter Nodes to Binary Tree:");

scanf("%d",&value);

root\_node = insert\_node(root\_node, value);

for(int i=1;i<n;i++)

{

scanf("%d",&value);

insert\_node(root\_node, value);

}

}

else if(choice== 2){

int r=height(root\_node);

printf("Height of the tree:");

printf("%d\n",r);

}

else if(choice== 3)

{

int r=height(root\_node);

display(root\_node,r);

printf("\n");

}

else if(choice==4) {

if(root\_node==NULL)

printf("Empty");

else

{

printf("\nThe preorder traversal is ");

preorderTraversal(root\_node);

}

}

else if(choice==5) {

if(root\_node==NULL)

printf("Empty");

else

{

printf("\nThe Inorder traversal is ");

inorderTraversal(root\_node);

}

}

else if(choice==6)

{

if(root\_node==NULL)

printf("Empty");

else{

printf("\nThe postorder traversal is ");

postorderTraversal(root\_node);

}

}

else if(choice==7)

break;

else

printf("Enter valid choice\n");

}

else{

printf("Invalid input. Only integers allowed\n");

exit(0);

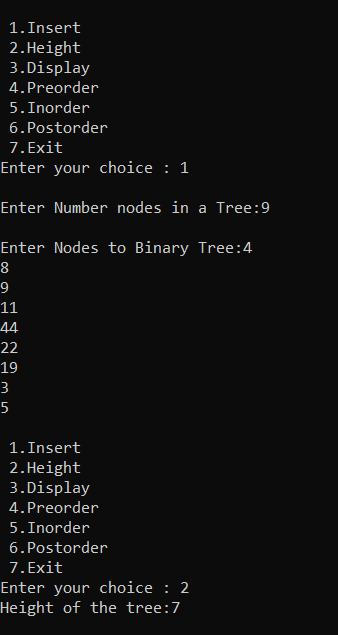
}

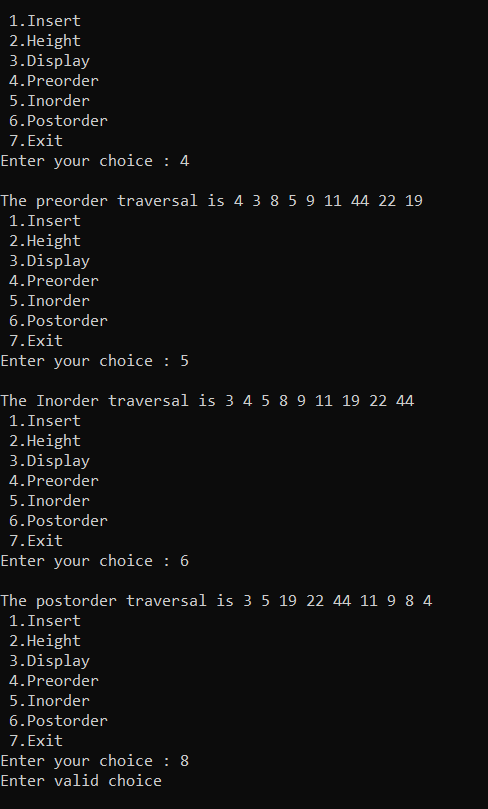
}

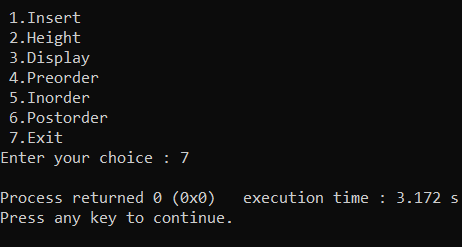
return 0;

}

Output:







2.Convert infix to postfix , prefix , valid infix expression:

Source Code:

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

struct Stack {

int top;

int maxSize;

int\* array;

};

struct Stack\* create(int max)

{

struct Stack\* stack = (struct Stack\*)malloc(sizeof(struct Stack));

stack->maxSize = max;

stack->top = -1;

stack->array = (int\*)malloc(stack->maxSize \* sizeof(int));

return stack;

}

int isFull(struct Stack\* stack)

{

if(stack->top == stack->maxSize - 1){

printf("Will not be able to push maxSize reached\n");

}

return stack->top == stack->maxSize - 1;

}

int isEmpty(struct Stack\* stack)

{

return stack->top == -1;

}

void push(struct Stack\* stack, int item)

{

if (isFull(stack))

return;

stack->array[++stack->top] = item;

}

int pop(struct Stack\* stack)

{

if (isEmpty(stack))

return INT\_MIN;

return stack->array[stack->top--];

}

int peek(struct Stack\* stack)

{

if (isEmpty(stack))

return INT\_MIN;

return stack->array[stack->top];

}

int checkIfOperand(char ch)

{

return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');

}

int precedence(char ch)

{

{

switch (ch)

{

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

}

int covertInfixToPostfix(char\* expression)

{

int i, j;

struct Stack\* stack = create(strlen(expression));

if(!stack)

return -1 ;

for (i = 0, j = -1; expression[i]; ++i)

{

if (checkIfOperand(expression[i]))

expression[++j] = expression[i];

else if (expression[i] == '(')

push(stack, expression[i]);

else if (expression[i] == ')')

{

while (!isEmpty(stack) && peek(stack) != '(')

expression[++j] = pop(stack);

if (!isEmpty(stack) && peek(stack) != '(')

return -1; // invalid expression

else

pop(stack); }

else

{

while (!isEmpty(stack) && precedence(expression[i]) <= precedence(peek(stack)))

expression[++j] = pop(stack);

push(stack, expression[i]);

}

}

while (!isEmpty(stack))

expression[++j] = pop(stack);

expression[++j] = '\0';

printf( "%s", expression);

}

int getPostfix(char\* expression)

{

int i, j;

struct Stack\* stack = create(strlen(expression));

if(!stack)

return -1 ;

for (i = 0, j = -1; expression[i]; ++i)

{

if (checkIfOperand(expression[i]))

expression[++j] = expression[i];

else if (expression[i] == '(')

push(stack, expression[i]);

else if (expression[i] == ')')

{

while (!isEmpty(stack) && peek(stack) != '(')

expression[++j] = pop(stack);

if (!isEmpty(stack) && peek(stack) != '(')

return -1;

else

pop(stack);

}

else

{

while (!isEmpty(stack) && precedence(expression[i]) <= precedence(peek(stack)))

expression[++j] = pop(stack);

push(stack, expression[i]);

}

}

while (!isEmpty(stack))

expression[++j] = pop(stack);

expression[++j] = '\0';

}

void reverse(char \*exp)

{

int size = strlen(exp);

int j = size, i=0;

char temp[size];

temp[j--]='\0';

while(exp[i]!='\0')

{

temp[j] = exp[i];

j--;

i++;

}

strcpy(exp,temp);

}

void brackets(char\* exp){

int i = 0;

while(exp[i]!='\0')

{

if(exp[i]=='(')

exp[i]=')';

else if(exp[i]==')')

exp[i]='(';

i++;

}

}

void InfixtoPrefix(char \*exp){

int size = strlen(exp);

reverse(exp);

brackets(exp);

getPostfix(exp);

reverse(exp);

}

//Checking for operator

int isOperator (char symbol)

{

if (symbol == '+' || symbol == '-' || symbol == '\*' || symbol == '/'

|| symbol == '^')

{

return 1;

}

return 0;

}

int validate(char expr[])

{

int n = strlen(expr);

if(isOperator(expr[0]) || isOperator(expr[n-1]) )

{

return 0;

}

//return if you find two consecutive operator in the expression

for(int i=0; i<n-1; i++)

{

if(isOperator(expr[i]) && isOperator(expr[i+1]) )

{

return 0;

}

}

return 1;

}ss

int main()

{

char infix[MAX];

printf("\nEnter the expression :\n");

gets(infix);

if(validate(infix))

{

printf("infix expression is valid\n");

printf("The postfix is: ");

covertInfixToPostfix(infix);

printf("\n");

InfixtoPrefix(infix);

printf("The prefix is: ");

printf("%s\n",infix);

return 0;

}

else{

printf("infix expression is not valid");

}

}

Output:

