Karthik Jada

AP19110010488

1)C program to reverse a string using stack

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <limits.h>

// A structure to represent a stack

struct Stack

{

int top;

unsigned capacity;

char\* array;

};

// function to create a stack of given

// capacity. It initializes size of stack as 0

struct Stack\* createStack(unsigned capacity)

{

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

stack->capacity = capacity;

stack->top = -1;

stack->array = (char\*) malloc(stack->capacity \* sizeof(char));

return stack;

}

// Stack is full when top is equal to the last index

int isFull(struct Stack\* stack)

{ return stack->top == stack->capacity - 1; }

// Stack is empty when top is equal to -1

int isEmpty(struct Stack\* stack)

{ return stack->top == -1; }

// Function to add an item to stack.

// It increases top by 1

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

{

if (isFull(stack))

return;

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

}

// Function to remove an item from stack.

// It decreases top by 1

char pop(struct Stack\* stack)

{

if (isEmpty(stack))

return INT\_MIN;

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

}

// A stack based function to reverse a string

void reverse(char str[])

{

// Create a stack of capacity

//equal to length of string

int n = strlen(str);

struct Stack\* stack = createStack(n);

// Push all characters of string to stack

int i;

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

push(stack, str[i]);

// Pop all characters of string and

// put them back to str

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

str[i] = pop(stack);

}

// Driver program to test above functions

int main()

{

char str[] = "GeeksQuiz";

reverse(str);

printf("Reversed string is %s", str);

return 0;

}

C PROGRRAMME FOR INFIX TO POSTFIX CONVERSION USING STACK

include<stdio.h>

#define SIZE 50

char stack[SIZE];

int top=-1;

void push(char item)

{

if(top>=SIZE-1)

printf("\n stack oerflow,push not possible \n");

else

{

top++;

stack[top]=item;

}

}

char pop()

{

char item;

item=stack[top];

top--;

return(item);

}

int is\_operator(char symbol)

{

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

return 1;

else

return 0;

}

int precedence(char symbol)

{

if(symbol=='^')

return 3;

else if(symbol=='\*'|| symbol=='/')

return 2;

else if(symbol=='+'||symbol=='-')

return 1;

else

return 0;

}

int main()

{

char infix[SIZE],postfix[SIZE],item,temp;

int i=0,j=0;

printf("\n enter the arthimatic notation in infix notation:");

gets(infix);

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

{

item=infix[i];

if(item=='(')

{

push(item);

}

else if(item>='A' && item<='Z' || item>='a' && item<='z')

{

postfix[j]=item;

j++;

}

else if(is\_operator(item)==1)

{

temp=pop();

while(is\_operator(temp)==1 && precedence(temp)>=precedence(item))

{

postfix[j]=temp;

j++;

temp=pop();

}

push(temp);

push(item);

}

else if(item==')')

{

temp=pop();

while(temp!='(')

{

postfix[j]=temp;

j++;

temp=pop();

}

}

else

{

printf("\n invalid arthimatic expression\n");

getch();

exit(0);

}

i++;

}

while(top>-1)

{

postfix[j]=pop();

j++;

}

postfix[j]='\0';

puts(postfix);

getch();

return 0;

}

C PROGRAMME TO IMPLEMENT QUEUE USING TWO STACKS

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

void push(struct node\*\* top, int data);

int pop(struct node\*\* top);

struct queue

{

struct node \*stack1;

struct node \*stack2;

};

void enqueue(struct queue \*q, int x)

{

push(&q->stack1, x);

}

void dequeue(struct queue \*q)

{

int x;

if (q->stack1 == NULL && q->stack2 == NULL) {

printf("queue is empty");

return;

}

if (q->stack2 == NULL) {

while (q->stack1 != NULL) {

x = pop(&q->stack1);

push(&q->stack2, x);

}

}

x = pop(&q->stack2);

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

}

void push(struct node\*\* top, int data)

{

struct node\* newnode = (struct node\*) malloc(sizeof(struct node));

if (newnode == NULL) {

printf("Stack overflow \n");

return;

}

newnode->data = data;

newnode->next = (\*top);

(\*top) = newnode;

}

int pop(struct node\*\* top)

{

int buff;

struct node \*t;

if (\*top == NULL) {

printf("Stack underflow \n");

return;

}

else {

t = \*top;

buff = t->data;

\*top = t->next;

free(t);

return buff;

}

}

void display(struct node \*top1,struct node \*top2)

{

while (top1 != NULL) {

printf("%d\n", top1->data);

top1 = top1->next;

}

while (top2 != NULL) {

printf("%d\n", top2->data);

top2 = top2->next;

}

}

int main()

{

struct queue q = (struct queue)malloc(sizeof(struct queue));

int f = 0, a;

char ch = 'y';

q->stack1 = NULL;

q->stack2 = NULL;

while (ch == 'y'||ch == 'Y') {

printf("enter ur choice\n1.add to queue\n2.remove

from queue\n3.display\n4.exit\n");

scanf("%d", &f);

switch(f) {

case 1 : printf("enter the element to be added to queue\n");

scanf("%d", &a);

enqueue(q, a);

break;

case 2 : dequeue(q);

break;

case 3 : display(q->stack1, q->stack2);

break;

case 4 : exit(1);

break;

default : printf("invalid\n");

break;

}

}

}

C PROGRAMME FOR INSERTION AND DELETION OF BST

#include<stdio.h>

#include<stdlib.h>

typedef struct NodePtr

{

int value;

struct NodePtr \*left;

struct NodePtr \*right;

}Node;

Node\* createNode(int value)

{

Node node=(Node)malloc(sizeof(Node));

node->value=value;

node->left = node->right=NULL;

return node;

}

Node\* insert(Node \*root,int value)

{

if(root == NULL)

{

root=createNode(value);

}

else if(value < root->value)

{

root->left=insert(root->left,value);

}

else if(value > root->value)

{

root->right=insert(root->right,value);

}

return root;

}

Node\* findMinNode(Node \*root)

{

if(root->left)

{

root=findMinNode(root->left);

}

return root;

}

Node\* delete(Node\* root,int value)

{

if(root)

{

if(value>root->value)

{

root->right=delete(root->right,value);

}

else if(value<root->value)

{

root->left=delete(root->left,value);

}

else if(value == root->value)

{

if(root->left && root->right)

{

Node \*minNode= findMinNode(root->right);

root->value=minNode->value;

root->right=delete(root->right, minNode->value);

}

else

{

Node \*nodeToDelete=root;

if(root->left)

{

root=root->left;

}

else

{

root=root->right;

}

free(nodeToDelete);

}

}

}

return root;

}

void print(Node \*root)

{

if(root)

{

printf("\n %d",root->value);

print(root->left);

print(root->right);

}

}

int main()

{

Node \*root=NULL;

root=insert(root,30);

root=insert(root,25);

root=insert(root,50);

root=insert(root,27);

root=insert(root,40);

root=insert(root,66);

root=insert(root,5);

root=insert(root,3);

print(root);

printf("\n \n \n");

delete(root,3);

print(root);

return 0;

}