# write a c program to reverse a string using stack?

#include <stdio.h> #include <string.h>

#define MAX 200 int top=-1;

int item;

char stack\_string[MAX];

/\*function to push character (item)\*/ void pushChar(char item);

/\*function to pop character (item)\*/ char popChar(void);

/\*function to check stack is empty or not\*/ int isEmpty(void);

/\*function to check stack is full or not\*/ int isFull(void);

int main()

{

char str[MAX]; int i;

printf("Input data of the string "); scanf("%[^\n]s",str);

/\*read string with spaces\*/

/\*gets(str);-can be used to read string with spaces\*/ for(i=0;i<strlen(str);i++)

pushchar(str[i]);

for(i=0;i<strlen(str);i++) str[i]=popchar();

printf("Reversed String is: %s\n",str);

return 0;

}

/\*function definition of pushChar\*/ void pushChar(char item)

{

/\*check for full\*/ if(It isFull())

{

printf("\nStack is full !\n"); return;

}

/\*increase top and push item in stack\*/ top=top+1;

stack\_string[top]=item;

}

/\*function definition of popChar\*/ char popChar()

{

if(It sEmpty())

{

printf("\nStack is empty !!!\n"); return 0;

}

/\*pop item and decrease top\*/ item = stack\_string[top]; top=top-1;

return item;

}

/\*function definition of isEmpty\*/ int isempty()

{

if(top==-1) return 1;

else

return 0;

}

/\*function definition of isFull\*/ int isFull()

{

if(top==MAX-1) return 1;

else

return 0;

}}

# write a program for Infix To Postfix Conversion Using Stack.

#include<stdio.h> char stack[50];

int top = -1;

void push(char a)

{

stack[++top] = a;

}

char pop()

{

if(top == -1)

return -1; else

return stack[top--];

}

int priority(char s)

{

if(s == '(') return 0;

if(s == '+' || s == '-') return 1;

if(s == '\*' || s == '/') return 2;

}

main()

{

char exp[50]; char \*e, s;

printf("Enter the expression :: "); scanf("%s",exp);

e = exp; while(\*e != '\0')

{

if(isalnum(\*e)) printf("%c",\*e);

else if(\*e == '(') push(\*e);

else if(\*e == ')')

{

while((s = pop()) != '(') printf("%c", s);

}

else

{

while(priority(stack[top]) >= priority(\*e)) printf("%c",pop());

push(\*e);

} e++;

}

while(top != -1)

{

printf("%c",pop());

}

}

# write a C Program to Implement Queue Using Two Stacks

/\* C program to implement queues 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 \*stack\_a; struct node \*stack\_b;

};

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

{

push(&q->stack\_a, x);

}

void dequeue(struct queue \*q)

{

int x;

if (q->stack\_a == NULL && q->stack\_b == NULL) { printf("Empty Queue");

return;

}

if (q->stack\_b == NULL) {

while (q->stack\_a != NULL) { x = pop(&q->stack\_a); push(&q->stack\_a, x);

}

}

x = pop(&q->stack\_b); 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");

}

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->stack\_a = NULL; q->stack\_b = 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 a : printf("enter the element to be added\n"); scanf("%d", &a);

enqueue(q, a); break;

case b : dequeue(q); break;

case c : display(q->stack\_a, q->stack\_b); break;

case d : exit(1); break;

default : printf("invalid\n"); break;

}

}

}

# /\* write a c program for insertion and deletion of BST.\*/

#include <stdio.h> #include <stdlib.h>

struct btnode

{

int value;

struct btnode \*left; struct btnode \*right;

}\*root = NULL, \*temp = NULL, \*t2, \*t1;

void delete1(); void insert(); void delete(); void create();

void search(struct btnode \*t);

void search1(struct btnode \*t,int data); int smallest(struct btnode \*t);

int largest(struct btnode \*t); int flag = 1;

void main()

{

int ch;

printf("\nOPERATIONS ---");

printf("\n1 - Insert the element in tree\n"); printf("2 - Delete the element from the tree\n"); printf("3 - Exit\n");

while(1)

{

printf("\nEnter your choice : "); scanf("%d", &ch);

switch (ch)

{

case a:

insert(); break;

case b:

delete(); break;

case c:

exit(0); default :

printf("Wrong choice, Please enter correct choice "); break;

}

}

}

/\* To insert a node in the tree \*/ void insert()

{

create();

if (root == NULL) root = temp;

else

search(root);

}

/\* To create a node \*/ void create()

{

int data;

printf("Enter data of node to be inserted : "); scanf("%d", &data);

temp = (struct btnode \*)malloc(1\*sizeof(struct btnode)); temp->value = data;

temp->left = temp->right = NULL;

}

/\* Function to search the appropriate position to insert the new node \*/ void search(struct btnode \*t)

{

if ((temp->value > t->value) && (t->right != NULL)) /\* value more than root node value insert at right

\*/

search(t->right);

else if ((temp->value > t->value) && (t->right == NULL))

t->right = temp;

else if ((temp->value < t->value) && (t->left != NULL)) /\* value less than root node value insert at left

\*/

search(t->left);

else if ((temp->value < t->value) && (t->left == NULL)) t->left = temp;

}

/\* To check for the deleted node \*/ void delete()

{

int data;

if (root == NULL)

{

printf("No elements in a tree to delete"); return;

}

printf("Enter the data to be deleted : "); scanf("%d", &data);

t1 = root; t2 = root;

search1(root, data);

}

/\* Search for the appropriate position to insert the new node \*/ void search1(struct btnode \*t, int data)

{

if ((data>t->value))

{

t1 = t;

search1(t->right, data);

}

else if ((data < t->value))

{

t1 = t;

search1(t->left, data);

}

else if ((data==t->value))

{

delete1(t);

}

}

/\* To delete a node \*/

void delete1(struct btnode \*t)

{

int k;

/\* To delete leaf node \*/

if ((t->left == NULL) && (t->right == NULL))

{

if (t1->left == t)

{

t1->left = NULL;

}

else

{

t1->right = NULL;

}

t = NULL;

free(t); return;

}

/\* To delete node having one left hand child \*/ else if ((t->right == NULL))

{

if (t1 == t)

{

root = t->left; t1 = root;

}

else if (t1->left == t)

{

t1->left = t->left;

}

else

{

t1->right = t->left;

}

t = NULL;

free(t); return;

}

/\* To delete node having right hand child \*/ else if (t->left == NULL)

{

if (t1 == t)

{

root = t->right; t1 = root;

}

else if (t1->right == t) t1->right = t->right;

else

t1->left = t->right;

t == NULL;

free(t); return;

}

/\* To delete node having two child \*/

else if ((t->left != NULL) && (t->right != NULL))

{

t2 = root;

if (t->right != NULL)

{

k = smallest(t->right); flag = 1;

}

else

{

k =largest(t->left); flag = 2;

}

search1(root, k); t->value = k;

}

}

/\* To find the smallest element in the right sub tree \*/ int smallest(struct btnode \*t)

{

t2 = t;

if (t->left != NULL)

{

t2 = t;

return(smallest(t->left));

}

else

return (t->value);

}

/\* To find the largest element in the left sub tree \*/ int largest(struct btnode \*t)

{

if (t->right != NULL)

{

t2 = t;

return(largest(t->right));

}

else

return(t->value);

}