DSA CLASS ASSIGNMENT -5

N.Saikeerthi

AP19110010543

1). write a c program to reverse a string using stack?

#include<stdio.h>

#include<string.h>

#define SIZE 90

int top = -1;

char stack[SIZE];

char push(char x)

{

if(top == (SIZE-1))

printf("Stack Overflow\n");

Else

stack[++top] =x;

}

char pop()

{

if(top == -1)

printf("Stack Underflow\n");

else{

return stack[top--];

}

}

int main()

{

char str[90];

int n;

printf("Enter the string : " );

gets(str);

for(n=0;n<strlen(str);n++)

push(str[n]);

for(n=0;n<strlen(str);n++)

str[n]=pop();

printf("Reversed string is : ");

puts(str);

}

2) write a program for Infix To Postfix Conversion Using Stack. #include<stdio.h>

char stack[20];

int top = -1;

void push(char x)

{

stack[++top] = x;

}

char pop()

{

if(top == -1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x == '(')

return 0;

if(x == '+' || x == '-')

return 1;

if(x == '\*' || x == '/')

return 2; }

main()

{

char exp[20];

char \*e, x;

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((x = pop()) != '(')

printf("%c", x);

}

else

{

while(priority(stack[top]) >= priority(\*e))

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

push(\*e);

}

e++;

}

while(top != -1)

{

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

}

}

3​) write a C Program to Implement Queue Using Two Stacks

#include <stdio.h>

#include <stdlib.h>

struct sNode {

int data;

struct sNode\* next;

};

void push(struct sNode\*\* top\_ref, int new\_data);

int pop(struct sNode\*\* top\_ref);

struct queue {

struct sNode\* stack1;

struct sNode\* stack2;

};

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

{

push(&q->stack1, x);

}

int deQueue(struct queue\* q)

{

int x;

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

printf("Q is empty");

getchar();

exit(0);

}

if (q->stack2 == NULL) {

while (q->stack1 != NULL) {

x = pop(&q->stack1);

push(&q->stack2, x);

}

}

x = pop(&q->stack2);

return x;

}

void push(struct sNode\*\* top\_ref, int new\_data)

{

structsNode\*new\_node=(structsNode\*)malloc(sizeof(structsNode));

if (new\_node == NULL)

{

printf("Stack overflow \n");

getchar();

exit(0);

}

new\_node->data = new\_data;

new\_node->next = (\*top\_ref);

(\*top\_ref) = new\_node;

}

int pop(struct sNode\*\* top\_ref)

{

int res;

struct sNode\* top;

if (\*top\_ref == NULL)

{

printf("Stack underflow \n");

getchar();

exit(0);

}

else

{

top = \*top\_ref;

res = top->data;

\*top\_ref = top->next;

free(top);

return res;

}

}

int main()

{

struct queue\* q = (struct queue\*)malloc(sizeof(struct queue)); q->stack1 = NULL;

q->stack2 = NULL;

enQueue(q, 1);

enQueue(q, 2);

enQueue(q, 3);

printf("%d ", deQueue(q));

printf("%d ", deQueue(q));

printf("%d ", deQueue(q));

return 0;

}

4​) write a c program for insertion and deletion of BST

Insertion in BST

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node\* left;

struct node\* right;

};

struct node \*newNode(int item)

{

struct node \*temp = (struct node \*)malloc(sizeof(struct node)); temp->data = item;

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

return temp;

}

struct node\* insert(struct node \*node, int value)

{

if (node == NULL) return newNode(value);

if (value < node->data)

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

else if (value > node->data)

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

return node;

}

void inorder(struct node\* root){

if(root == NULL) return;

inorder(root->left);

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

inorder(root->right); } void main()

{

struct node \*root = NULL;

root = insert(root, 50);

insert(root, 30);

insert(root, 20);

insert(root, 40);

insert(root, 70);

printf("\ninorder traversal \n");

inorder(root);

}

Deletion in bst: #

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*right\_child;

struct node \*left\_child;

};

struct node\* search(struct node \*root, int x)

{

if(root==NULL || root->data==x)

return root;

else if(x>root->data)

return search(root->right\_child, x);

else

return search(root->left\_child,x);

}

struct node\* find\_minimum(struct node \*root)

{

if(root == NULL)

return NULL;

else if(root->left\_child != NULL)

return find\_minimum(root->left\_child);

return root;

}

//function to create a node

struct node\* new\_node(int x)

{

struct node \*p;

p = malloc(sizeof(struct node));

p->data = x;

p->left\_child = NULL;

p->right\_child = NULL;

return p;

}

struct node\* insert(struct node \*root, int x)

{

if(root==NULL)

return new\_node(x);

else if(x>root->data)

root->right\_child = insert(root->right\_child, x);

else

root->left\_child = insert(root->left\_child,x);

return root;

}

struct node\* delete(struct node \*root, int x)

{

if(root==NULL)

return NULL;

if (x>root->data)

root->right\_child = delete(root->right\_child, x);

else if(x<root->data)

root->left\_child = delete(root->left\_child, x);

else

{

if(root->left\_child==NULL && root->right\_child==NULL)

{

free(root);

return NULL;

}

else if(root->left\_child==NULL || root->right\_child==NULL) {

struct node \*temp;

if(root->left\_child==NULL)

temp = root->right\_child;

else

temp = root->left\_child;

free(root);

return temp;

}

Else

{

struct node \*temp = find\_minimum(root->right\_child); root->data = temp->data;

root->right\_child = delete(root->right\_child, temp->data); }

}

return root;

}

void inorder(struct node \*root)

{

if(root!=NULL)

{

inorder(root->left\_child);

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

inorder(root->right\_child);

}

}

int main()

{

struct node \*root;

root = new\_node(20);

insert(root,6);

insert(root,3);

insert(root,17);

insert(root,8);

insert(root,31);

insert(root,27);

insert(root,42);

inorder(root);

printf("\n");

root = delete(root, 1);

root = delete(root, 40);

root = delete(root, 9);

inorder(root);

printf("\n");

return 0;

}