DSA ASSIGNMENT - 5

Maddineni Chandrahas
AP19110010536
CSE-F

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

```
/*write a c program to reverse a string using stack*/
#include <stdio.h>
#include <string.h>
#define MAX 100
int up=-1;
int item;
char stack_string[MAX];

void pushChar (char item);
char popChar (void);
int isEmpty (void);
int isFull (void);
int
main ()
{
  char str[MAX];
  int c;
  printf ("Input a string: ");
  scanf ("%[^\n]s", str);
  for (c = 0; c < strlen (str); c++)
    pushChar (str[c]);
  for (c = 0; c < strlen (str); c++)
    str[c] = popChar ();
printf ("Reversed String is: %s\n", str);
  return 0;
}
void
pushChar (char item)
{
  if (isFull ())
    {
      printf ("\nStack is FULL !!!\n");
      return;
    }
  up = up + 1;
  stack_string[up] = item;
}
char
popChar ()
{
```

```
  if (isEmpty ())
    {
      printf ("\nStack is EMPTY!!!\n");
      return 0;
    }
  item = stack_string[up];
  up = up - 1;
  return item;
}
int
isEmpty ()
{
  if (up == -1)
    return 1;
  else
    return 0;
}
int
isFull ()
{
  if (up == MAX - 1)
    return 1;
  else
    return 0;
}
```

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

```
/*8write a program for Infix To Postfix Conversion Using Stack*/
#include<stdio.h>
char stack[20];
int up = -1;
void push(char c)
{
stack[++up] = c;
}
char pop()
{
if(up == -1)
return -1;
```

```
else
return stack[up--];
}
int priority(char c)
{
if(c == '(')
return 0;
if(c == '+' || c == '-')
return 1;
if(c == '*' || c == '/')
return 2;
}
main()
{
char exp[20];
char *e, c;
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((c = pop()) != '(')
printf("%c", c);
}
else
{
while(priority(stack[up]) >= priority(*e))
printf("%c",pop());
push(*e);
}
e++;
}
while(up != -1)
{
printf("%c",pop());
}
}
```

3.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** up, int data);
int pop(struct node** up);
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 the 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** up, int data)
{
struct node* newnode = (struct node*) malloc(sizeof(struct node));
if (newnode == NULL) {
```

```
printf("Stack overflow \n");
return;
}
newnode->data = data;
newnode->next = (*up);
(*up) = newnode;
}
int pop(struct node** up)
{
int buff;
struct node *t;
if (*up == NULL) {
printf("Stack underflow \n");
}
else {
t = *up;
buff = t->data;
*up = t->next;
free(t);
return buff;
}}
void display(struct node *up1,struct node *up2)
{
while (up1 != NULL) {
printf("%d\n", up1->data);
up1 = up1->next;
}
while (up2 != NULL) {
printf("%d\n", up2->data);
up2 = up2->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 your choice\n1.add an element to queue\n2.remove an element from
queue\n3.display\n4.exit\n");
scanf("%d", &f);
```

```
switch(f) {
case 1 : printf("enter the element to be added\n");
scanf("%d", &a);
enqueue(q, a);
break;
case 2 : dequeue(q);
break;
case 3 : display(q->stack_a, q->stack_b);
break;
case 4 : exit(1);
break;
default : printf("invalid input\n");
break;
}}}
```

---

4. write a c program for insertion and deletion of BST.

```
/* 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 c;
printf("\nOPERATIONS ---");
printf("\n1 - Insert an element into tree\n");
```

```
printf("2 - Delete an element from the tree\n");
printf("3 - Exit\n");
while(1)
{
printf("\nEnter your choice of selection : ");
scanf("%d", &c);
switch (c)
{
case 1:
insert();
break;
case 2:
delete();
break;
case 3:
printf("Thank you for using this program😃");
exit(0);
default :
printf("Incorrect input ");
break;
}}}
void insert()
{
create();
if (root == NULL)
root = temp;
else
search(root);
}
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;
}
void search(struct btnode *t)
{
if ((temp->value > t->value) && (t->right != NULL))
search(t->right);
else if ((temp->value > t->value) && (t->right == NULL))
```

```
t->right = temp;
else if ((temp->value < t->value) && (t->left != NULL))
search(t->left);
else if ((temp->value < t->value) && (t->left == NULL))
t->left = temp;
}
void delete()
{
int data;
if (root == NULL)
{
printf("No elements ar recorded in a tree to delete");
return;
}
printf("Enter the data to be deleted : ");
scanf("%d", &data);
t1 = root;
t2 = root;
search1(root, data);
}
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);
}}
void delete1(struct btnode *t)
{
int k;
if ((t->left == NULL) && (t->right == NULL))
{
if (t1->left == t)
{
```

```
t1->left = NULL;
}
else
{
t1->right = NULL;
}
t = NULL;
free(t);
return;
}
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;
}
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;
```

```
}
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;
}}
int smallest(struct btnode *t)
{
t2 = t;
if (t->left != NULL)
{
t2 = t;
return(smallest(t->left));
}
else
return (t->value);
}

int largest(struct btnode *t)
{
if (t->right != NULL)
{
t2 = t;
return(largest(t->right));
}
else
return(t->value);
}
```

**********THE END**********