1

#include <stdio.h>

#define compare(a,b) ((a)>(b)?1:((a)==(b)?0:-1))

int iterativeBinarySearch**(**int arr**[],** int n**,** int x**){**

int left **=** 0**,** right **=** n **-** 1**;**

**while** **(**left **<=** right**){**

int mid **=** **(**left **+** right**)** **/** 2**;**

int cmp **=** compare**(**arr**[**mid**],** x**);**

**if** **(**cmp **==** 0**)**

**return** mid**;**

**else** **if** **(**cmp **==** 1**)**

right **=** mid **-** 1**;**

**else**

left **=** mid **+** 1**;**

**}**

**return** **-**1**;**

**}**

int recursiveBinarySearch**(**int arr**[],** int left**,** int right**,** int x**){**

**if** **(**right **>=** left**){**

int mid **=** **(**left **+** right**)** **/** 2**;**

int cmp **=** compare**(**arr**[**mid**],** x**);**

**if** **(**cmp **==** 0**)**

**return** mid**;**

**else** **if** **(**cmp **==** 1**)**

**return** recursiveBinarySearch**(**arr**,** left**,** mid **-** 1**,** x**);**

**else**

**return** recursiveBinarySearch**(**arr**,** mid **+** 1**,** right**,** x**);**

**}**

**return** **-**1**;**

**}**

int main**(){**

int arr**[]** **=** **{**1**,** 3**,** 5**,** 7**,** 9**};**

int n **=** **sizeof(**arr**)** **/** **sizeof(**arr**[**0**]);**

int x **=** 9**;**

int index **=** iterativeBinarySearch**(**arr**,** n**,** x**);**

**if** **(**index **==** **-**1**)**

printf**(**"Element not found using iterative binary search\n"**);**

**else**

printf**(**"Element found at index %d using iterative binary search\n"**,** index**);**

index **=** recursiveBinarySearch**(**arr**,** 0**,** n **-** 1**,** x**);**

**if** **(**index **==** **-**1**)**

printf**(**"Element not found using recursive binary search\n"**);**

**else**

printf**(**"Element found at index %d using recursive binary search\n"**,** index**);**

**return** 0**;**

**}**

2

#include<stdio.h>

**typedef** struct**{**

int r**,**c**,**v**;**

**}** term**;**

void transpose**(**term a**[],**term t**[]){**

int rt**[**10**],**sp**[**10**];**

int i**,**j**,**numcols**=**a**[**0**].**c**,**numterms**=**a**[**0**].**v**;**

t**[**0**].**r**=**numcols**;**

t**[**0**].**v**=**numterms**;**

t**[**0**].**c**=**a**[**0**].**r**;**

**if(**numterms**>**0**){**

**for(**i**=**0**;**i**<**numcols**;**i**++)**

rt**[**i**]=**0**;**

**for(**i**=**1**;**i**<=**numterms**;**i**++)**

rt**[**a**[**i**].**c**]++;**

sp**[**0**]=**1**;**

**for(**i**=**1**;**i**<**numcols**;**i**++)**

sp**[**i**]=**sp**[**i**-**1**]+**rt**[**i**-**1**];**

**for(**i**=**1**;**i**<=**numterms**;**i**++){**

j**=**sp**[**a**[**i**].**c**]++;**

t**[**j**].**r**=**a**[**i**].**c**;**

t**[**j**].**c**=**a**[**i**].**r**;**

t**[**j**].**v**=**a**[**i**].**v**;**

**}**

**}**

**}**

int main**(){**

term a**[**10**],**t**[**10**];**

int i**;**

printf**(**"\nEnter the number of rows and columns\n"**);**

scanf**(**"%d%d"**,&**a**[**0**].**r**,&**a**[**0**].**c**);**

printf**(**"\nEnter the number of elements\n"**);**

scanf**(**"%d"**,&**a**[**0**].**v**);**

**for(**i**=**1**;**i**<=**a**[**0**].**v**;**i**++){**

printf**(**"\nEnter element no. %d row, column and value\n"**,** i**);**

scanf**(**"%d%d%d"**,&**a**[**i**].**r**,** **&**a**[**i**].**c**,** **&**a**[**i**].**v**);**

**}**

printf**(**"\nOriginal Matrix\n"**);**

**for(**i**=**1**;**i**<=**a**[**0**].**v**;**i**++)**

printf**(**"%d\t%d\t%d\n"**,**a**[**i**].**r**,**a**[**i**].**c**,**a**[**i**].**v**);**

transpose**(**a**,**t**);**

printf**(**"\nTranspose Matrix\n"**);**

**for(**i**=**1**;**i**<=**a**[**0**].**v**;**i**++)**

printf**(**"%d\t%d\t%d\n"**,**t**[**i**].**r**,**t**[**i**].**c**,**t**[**i**].**v**);**

**return** 0**;**

**}**

3

#include<stdio.h>

#include<string.h>

int failure**[**20**];**

void fail**(**char **\***pat**)**

**{**

int i**,**j**;**

int n**=**strlen**(**pat**);**

failure**[**0**]=-**1**;**

**for(**j**=**1**;**j**<**n**;**j**++)**

**{**

i**=**failure**[**j**-**1**];**

**while((**pat**[**j**]!=**pat**[**i**+**1**])&&(**i**>**0**))**

i**=**failure**[**i**];**

**if(**pat**[**j**]==**pat**[**i**+**1**])**

failure**[**j**]=**i**+**1**;**

**else**

failure**[**j**]=-**1**;**

**}**

**}**

int match**(**char **\***string**,** char **\***pat**)**

**{**

int i**=**0**,**j**=**0**;**

int lens**=**strlen**(**string**);**

int lenp**=**strlen**(**pat**);**

**while(**i**<**lens**&&**j**<**lenp**)**

**{**

**if(**string**[**i**]==**pat**[**j**])**

**{**

i**++;**

j**++;**

**}**

**else** **if(**j**==**0**)**

i**++;**

**else**

j**=**failure**[**j**-**1**]+**1**;**

**}**

**return((**j**==**lenp**)?(**i**-**lenp**):-**1**);**

**}**

int main**(){**

int i**;**

char str**[**30**],**pat**[**20**];**

printf**(**"\nEnter a string\n"**);**

scanf**(**"%s"**,**str**);**

printf**(**"\nEnter a substring\n"**);**

scanf**(**"%s"**,**pat**);**

fail**(**sub**);**

i**=**match**(**str**,**pat**);**

**if(**i**==-**1**)**

printf**(**"\nPattern %s Not found"**,** pat**);**

**else**

printf**(**"\nPattern %sFound at position %d"**,**pat**,**i**+**1**);**

**return** 0**;**

**}**

4

#include<stdio.h>

#include<stdlib.h>

int **\***queue**;**

int front**,** rear**,** capacity**;**

int pop**(){**

front **=** **(**front**+**1**)%**capacity**;**

**return** queue**[**front**];**

**}**

int push**(**int data**){**

int **\***newQueue**,** i**;**

**if(** **(**rear**+**1**)%**capacity **==**front**%**capacity **){**

printf**(**"Extending Queue (new capacity=%d)\n"**,** 2**\***capacity**);**

newQueue **=** **(**int**\*)**malloc**(**capacity **\*** 2 **\*** **sizeof(**int**));**

**for(**i**=**1**;**front**!=**rear**;**i**++)**

newQueue**[**i**]** **=** pop**();**

rear **=** i**-**1**;**

front **=** 0**;**

capacity **\*=** 2**;**

free**(**queue**);**

queue **=** newQueue**;**

**}**

rear **=** **(**rear**+**1**)%**capacity**;**

queue**[**rear**]** **=** data**;**

**}**

void display**(){**

printf**(**"Elements:\n"**);**

**if(**front **==** rear**)**

printf**(**"Queue Empty\n"**);**

**for(**int i**=(**front**+**1**)%**capacity **;** i**!=** **(**rear**+**1**)%**capacity **;** i **=** **(**i**+**1**)%**capacity**)**

printf**(**"%d\n"**,** queue**[**i**]);**

**}**

int main**(){**

front **=** 0**;**

rear **=** 0**;**

capacity **=** 2**;**

queue **=** **(**int**\*)**malloc**(**capacity**\*** **sizeof(**int**));**

int choice **=**4**;**

int num**;**

**do{**

printf**(**"1.push\n2.pop.\n3.display\n4.exit\n"**);**

scanf**(**"%d"**,&**choice**);**

**switch(**choice**){**

**case** 1**:**scanf**(**"%d"**,&**num**);**

push**(**num**);**

**break;**

**case** 2**:**

**if(**front**==**rear**)**

printf**(**"Queue Empty\n"**);**

**else**

printf**(**"element removed is %d\n"**,**pop**());**

**break;**

**case** 3**:**

display**();**

**break;**

**case** 4**:**

**break;**

**default** **:** printf**(**"Invalid Entry\n"**);**

**}**

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

**}while(**choice**!=**4**);**

**return** 0**;**

**}**

5

#include<stdio.h>

#include<string.h>

#include<ctype.h>

#define STACKSIZE 100

int stack**[**STACKSIZE**];**

int top**=-**1**;**

int pop**(){**

**return** stack**[**top**--];**

**}**

void push**(**int n**){**

stack**[++**top**]** **=** n**;**

**}**

int result**(**int op1**,** int op2**,** char operator**){**

**switch(**operator**){**

**case** '+'**:return** op1**+**op2**;**

**case** '-'**:return** op1**-**op2**;**

**case** '\*'**:return** op1**\***op2**;**

**case** '/'**:return** op1**/**op2**;**

**case** '%'**:return** op1**%**op2**;**

**}**

**}**

int postfixEval**(**char **\***str**){**

int i**;**

int op1**,** op2**;**

**for(**i**=**0**;**i**<**strlen**(**str**);**i**++){**

**if(**isdigit**(**str**[**i**]))**

push**(**str**[**i**]-**'0'**);**

**else**

**{**

op2**=**pop**();**

op1**=**pop**();**

push**(**result**(**op1**,** op2**,** str**[**i**]));**

**}**

**}**

**return** pop**();**//since top of the stack has the answer

**}**

int main**(){**

char str**[**100**];**

printf**(**"Enter the Postfix Expression :\n"**);**

scanf**(**"%s"**,** str**);**

printf**(**"Result = %d\n"**,** postfixEval**(**str**));**

**return** 0**;**

**}**

6

#include<stdio.h>

#include<stdlib.h>

#define MAX\_STACKS 3

**typedef** struct**{**

int key**;**

**}** ele**;**

**typedef** struct stack **\***stackPtr**;**

**typedef** struct stack**{**

ele data**;**

stackPtr link**;**

**}** stack**;**

stackPtr top**[**MAX\_STACKS**];**

void push**(**int i**,** int item**){**

stackPtr temp**;**

temp**=(**stackPtr**)** malloc**(sizeof(**stack**));**

temp**->**data**.**key **=** item**;**

temp**->**link **=** top**[**i**];**

top**[**i**]** **=** temp**;**

**}**

int pop**(**int i**){**

stackPtr temp **=** top**[**i**];**

int item**;**

item **=** temp**->**data**.**key**;**

top**[**i**]** **=** temp**->**link**;**

free**(**temp**);**

printf**(**"Popped %d from stack %d\n"**,** item**,** i**);**

**}**

void display**(){**

int i**;**

stackPtr j**;**

**for(**i**=**0**;**i**<**MAX\_STACKS**;**i**++){**

printf**(**"Stack no.%d :\n"**,**i**+**1**);**

**if(**top**[**i**]** **==** **NULL)**

printf**(**"Stack Empty\n--------------------\n"**);**

**else{**

**for(**j **=** top**[**i**];** j **!=** **NULL** **;** j **=** j**->**link**)**

printf**(**"%d\n"**,**j**->**data**.**key**);**

printf**(**"\n--------------------\n"**);**

**}**

**}**

**}**

int main**(){**

int choice**=**4**,** i**,** j**;**

ele x**;**

**for(**i**=**0**;**i**<**MAX\_STACKS**;**i**++)**

top**[**i**]** **=** **NULL;**

**while(**1**){**

printf**(**"1.push\n2.pop\n3.display\n4.exit\n"**);**

printf**(**"Enter your choice\n"**);**

scanf**(**"%d"**,&**choice**);**

**switch(**choice**){**

**case** 1**:**

printf**(**"Enter the stack number(0-%d) and element to be added\n"**,**MAX\_STACKS**-**1**);**

scanf**(**"%d%d"**,&**i **,&**x**.**key**);**//x is the element to be pushed

push**(**i**,**x**.**key**);**

**break;**

**case** 2**:**

printf**(**"Enter the queue number(0-%d)\n"**,**MAX\_STACKS**-**1**);**

scanf**(**"%d"**,&**i**);**

**if(**top**[**i**]** **==** **NULL)**

printf**(**"Queue Empty\n"**);**

**else**

pop**(**i**);**

**break;**

**case** 3**:**

display**();**

**break;**

**case** 4**:**

exit**(**0**);**

**break;**

**default** **:**

printf**(**"Invalid Choice"**);**

**}**

**}**

**return** 0**;**

**}**

7

#include<stdio.h>

#include<stdlib.h>

#define MAXQUEUES 3

**typedef** struct node **\***nodePtr**;**

**typedef** struct node**{**

int data**;**

nodePtr link**;**

**}**node**;**

nodePtr front**[**MAXQUEUES**];**

nodePtr rear**[**MAXQUEUES**];**

void push**(**int i**,** int data**){**

nodePtr newNode **=** **(**nodePtr**)**malloc**(sizeof(**node**));**

newNode**->**data **=** data**;**

newNode**->**link **=NULL;**

**if(**front**[**i**]==NULL)**

front**[**i**]** **=** newNode**;**

**else**

rear**[**i**]->**link **=** newNode**;**

rear**[**i**]** **=** newNode**;**

**}**

void pop**(**int i**){**

**if(**front**[**i**]){**

nodePtr temp **=** front**[**i**];**

printf**(**"Popped : %d from Queue no.%d\n"**,** front**[**i**]->**data**,** i**);**

front**[**i**]** **=** front**[**i**]->**link**;**

free**(**temp**);**

**}**

**else**

printf**(**"Queue no.%d is EMPTY\n"**,** i**);**

**}**

void display**(**int i**){**

printf**(**"\nQueue no.%d\n"**,** i**);**

**if(**front**[**i**]){**

nodePtr temp **=** front**[**i**];**

**for(;** temp**!=NULL;** temp **=** temp**->**link**)**

printf**(**"%5d"**,** temp**->**data**);**

**}**

**else**

printf**(**"Queue %d Empty"**,** i**);**

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

**}**

int main**(){**

**for(**int i**=**0**;**i**<**MAXQUEUES**;** i**++){**

front**[**i**]** **=** **NULL;**

rear**[**i**]** **=** **NULL;**

**}**

int choice**,** i**,** data**;**

printf**(**"MENU\n1.push\n2.pop\n3.display\n4.exit\n"**);**

**do** **{**

printf**(**"choice : "**);**

scanf**(**"%d"**,** **&**choice**);**

**switch(**choice**){**

**case** 1**:**

printf**(**"Queue no(0-2) : "**);**

scanf**(**"%d"**,** **&**i**);**

printf**(**"Element : "**);**

scanf**(**"%d"**,** **&**data**);**

push**(**i**,** data**);**

**break;**

**case** 2**:**

printf**(**"Queue no(0-2) : "**);**

scanf**(**"%d"**,** **&**i**);**

pop**(**i**);**

**break;**

**case** 3**:**

printf**(**"Queue no(0-2) : "**);**

scanf**(**"%d"**,** **&**i**);**

display**(**i**);**

**break;**

**case** 4**:**

printf**(**"Exit\n"**);**

**break;**

**default:**printf**(**"Invalid\n"**);**

**}**

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

**}** **while(**choice**!=**4**);**

**return** 0**;**

**}**

8

#include<stdio.h>

#include<stdlib.h>

#define MALLOC(p,n,type) \

p=(type\*)malloc(n\*sizeof(type)); \

if(p==NULL) \

{ \

printf("insufficient memory"); \

exit(0); \

}

#define COMPARE(X,Y)(((X)==(Y))?0:((X)>(Y))?1:-1)

struct node

**{**

int coeff**;**

int expon**;**

struct node **\***link**;**

**};**

**typedef** struct node **\***NODE**;**

NODE attach**(**int coeff**,**int expon**,**NODE head**)**

**{**

NODE temp**,**cur**;**

MALLOC**(**temp**,**1**,**struct node**);**

temp**->**coeff**=**coeff**;**

temp**->**expon**=**expon**;**

cur**=**head**->**link**;**

**while(**cur**->**link**!=**head**)**

**{**cur**=**cur**->**link**;**

**}**

cur**->**link**=**temp**;**

temp**->**link**=**head**;**

**return** head**;**

**}**

NODE read\_poly**(**NODE head**)**

**{**

int i**=**1**;**

int coeff**;**

int expon**;**

printf**(**"enter the coefficient -999 to end the polynomial"**);**

**while(**1**)**

**{**

printf**(**"enter the %d term \n"**,**i**++);**

printf**(**"Coeff="**);**

scanf**(**"%d"**,&**coeff**);**

**if(**coeff**==-**999**)break;**

printf**(**"pow x="**);**

scanf**(**"%d"**,&**expon**);**

head**=**attach**(**coeff**,**expon**,**head**);**

**}**

**return** head**;**

**}**

NODE poly\_add**(**NODE head1**,**NODE head2**,**NODE head3**)**

**{**

NODE a**,**b**;**

int coeff**;**

a**=**head1**->**link**;**

b**=**head2**->**link**;**

**while(**a**!=**head1 **&&** b**!=**head2**)**

**{**

**switch(**COMPARE**(**a**->**expon**,**b**->**expon**))**

**{**

**case** 0**:**coeff**=**a**->**coeff**+**b**->**coeff**;**

**if(**coeff**!=**0**)**head3**=**attach**(**coeff**,**a**->**expon**,**head3**);**

a**=**a**->**link**;**

b**=**b**->**link**;**

**break;**

**case** 1**:**head3**=**attach**(**a**->**coeff**,**a**->**expon**,**head3**);**

a**=**a**->**link**;**

**break;**

**default:**head3**=**attach**(**b**->**coeff**,**b**->**expon**,**head3**);**

b**=**b**->**link**;**

**}**

**}**

**while(**a**!=**head1**)**

**{**

head3**=**attach**(**a**->**coeff**,**a**->**expon**,**head3**);**

a**=**a**->**link**;**

**}**

**while(**b**!=**head2**)**

**{**

head3**=**attach**(**b**->**coeff**,**b**->**expon**,**head3**);**

b**=**b**->**link**;**

**}**

**return** head3**;**

**}**

void display**(**NODE head**)**

**{**NODE temp**;**

**if(**head**->**link**==**head**)**

**{**

printf**(**"polynomial does not exists"**);**

**return;}**

temp**=**head**->**link**;**

**while(**temp**!=**head**)**

**{**

**if(**temp**->**coeff**<**0**)**

printf**(**"%2dX^%2d"**,**temp**->**coeff**,**temp**->**expon**);**

**else**

printf**(**"+%2dx^%2d"**,**temp**->**coeff**,**temp**->**expon**);**

temp**=**temp**->**link**;**

**}**

**}**

void main**()**

**{**

NODE head1**,**head2**,**head3**;**

MALLOC**(**head1**,**1**,**struct node**);**

MALLOC**(**head2**,**1**,**struct node**);**

MALLOC**(**head3**,**1**,**struct node**);**

head1**->**link**=**head1**;**

head2**->**link**=**head2**;**

head3**->**link**=**head3**;**

printf**(**"enter the first polynomail"**);**

head1**=**read\_poly**(**head1**);**

printf**(**"enter the second polynomail"**);**

head2**=**read\_poly**(**head2**);**

head3**=**poly\_add**(**head1**,**head2**,**head3**);**

printf**(**"Polynomail 1:"**);**

display**(**head1**);**

printf**(**"Polynomail 2:"**);**

display**(**head2**);**

printf**(**"Polynomail 3:"**);**

display**(**head3**);**

**}**

9

#include<stdio.h>

#include<stdlib.h>

**typedef** struct node **\***nodePtr**;**

**typedef** struct node **{**

nodePtr llink**;**

int data**;**

nodePtr rlink**;**

**}**node**;**

nodePtr head**;**

void dinsert**(){**

int n**;**

nodePtr temp**;**

printf**(**"Enter the info for the new node : "**);**

scanf**(**"%d"**,** **&**n**);**

temp**=(**nodePtr**)**malloc**(sizeof(**node**));**

temp**->**data**=**n**;**

temp**->**llink **=** head**;**

temp**->**rlink **=** head**->**rlink**;**

head**->**rlink**->** llink **=** temp**;**

head**->**rlink **=** temp**;**

**}**

void ddelete**(){**

nodePtr temp**=**head**->**rlink**;**

**if** **(**head**->**rlink **==** head**)**

printf**(**"Deletion of head node not permitted.\n"**);**

**else**

**{**

head**->**rlink **=** temp**->**rlink**;**

temp**->**rlink**->**llink **=** head**;**

printf**(**"removing node with data %d\n"**,**temp**->**data**);**

free**(**temp**);**

**}**

**}**

void displayRight**(){**

nodePtr temp**;**

**if** **(**head**->**rlink **==** head**)**

printf**(**"Empty list.\n"**);**

**else**

**{**

**for(**temp**=**head**->**rlink**;** temp**->**rlink **!=** head**;** temp **=** temp**->**rlink**)**

printf**(**"%d\t"**,** temp**->**data**);**

printf**(**"%d\t"**,** temp**->**data**);**

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

**}**

**}**

void displayLeft**(){**

nodePtr temp**;**

**if** **(**head**->**llink **==** head**)**

printf**(**"Empty list.\n"**);**

**else**

**{**

**for(**temp**=**head**->**llink**;** temp**->**llink **!=** head**;** temp **=** temp**->**llink**)**

printf**(**"%d\t"**,** temp**->**data**);**

printf**(**"%d\t"**,** temp**->**data**);**

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

**}**

**}**

int main**(){**

unsigned int choice**;**

head**=(**nodePtr**)**malloc**(sizeof(**node**));**

head**->**rlink**=**head**;**

head**->**llink**=**head**;**

**while(**1**){**

printf**(**"1:insert a node in DLL \n2:delete a node from DLL \n3:display the DLL forward\n4:display the DLL reverse\n5:exit\n"**);**

scanf**(**"%u"**,** **&**choice**);**

**switch(**choice**){**

**case** 1**:** dinsert**();**

**break;**

**case** 2**:** ddelete**();**

**break;**

**case** 3**:** displayLeft**();**

**break;**

**case** 4**:** displayRight**();**

**break;**

**case** 5**:** exit**(**0**);**

**break;**

**default:** printf**(**"Invalid choice... try again\n"**);**

**}**

**}**

**return** 0**;**

**}**

10

#include<stdio.h>

#include<stdlib.h>

#define MAX\_ELEMENTS 25

int heap**[**MAX\_ELEMENTS**];**

int n **=** 0**;**

void push**(**int item**){**

int i**;**

i**=** **++**n**;**

**while((**i**!=**1**)** **&&** **(** item **>** heap**[**i**/**2**]))** **{**

heap**[**i**]** **=** heap**[**i**/**2**];**

i **=** i**/**2**;**

**}**

heap**[**i**]** **=** item**;**

**}**

void pop**(){**

int item**;**

int temp**;**

int parent**,** child**;**

**if(**n**==**0**)**

printf**(**"heap is empty\n"**);**

**else{**

item **=** heap**[**1**];**

temp **=** heap**[**n**--];**

parent **=** 1**;**

child **=** 2**;**

**while(**child **<=** n**){**

**if(**child **<** n **&&** **(**heap**[**child**]** **<** heap**[**child**+**1**]))**

child**++;**

**if(**temp **>=** heap**[**child**])**

**break;**

heap**[**parent**]** **=** heap**[**child**];**

parent **=** child**;**

child **\*=** 2**;**

**}**

heap**[**parent**]** **=** temp**;**

printf**(**"Element removed from heap is %d\n"**,** item**);**

**}**

**}**

void display**(){**

int i**;**

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

printf**(**"%d\t"**,** heap**[**i**]);**

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

**}**

int main**(){**

unsigned int choice**;**

int x**;**

**while(**1**){**

printf**(**"1:insert a node to heap \n2:delete a node from heap \n3:display the max heap\n4:exit\n"**);**

scanf**(**"%u"**,** **&**choice**);**

**switch(**choice**){**

**case** 1**:** **if(**n **==** MAX\_ELEMENTS**){**

printf**(**"Heap is full\n"**);**

exit**(**1**);**

**}**

printf**(**"Enter the element to be added to heap\n"**);**

scanf**(**"%d"**,&**x**);**//x is the element to be pushed

push**(**x**);**

**break;**

**case** 2**:** pop**();**

**break;**

**case** 3**:** display**();**

**break;**

**case** 4**:** exit**(**0**);**

**break;**

**default:** printf**(**"Invalid choice... try again\n"**);**

**}**

**}**

**return** 0**;**

**}**

11

#include<stdio.h>

#include<stdlib.h>

**typedef** struct node**\*** treeptr**;**

**typedef** struct node**{**

int data**;**

treeptr left**;**

treeptr right**;**

**}**node**;**

treeptr createNode**(**int value**){**

treeptr newNode **=** malloc**(sizeof(**struct node**));**

newNode**->**data **=** value**;**

newNode**->**left **=** **NULL;**

newNode**->**right **=** **NULL;**

**return** newNode**;**

**}**

treeptr insert**(**treeptr root**,** int data**){**

**if** **(**root **==** **NULL)** **return** createNode**(**data**);**

**if** **(**data **<** root**->**data**)**

root**->**left **=** insert**(**root**->**left**,** data**);**

**else** **if** **(**data **>** root**->**data**)**

root**->**right **=** insert**(**root**->**right**,** data**);**

**return** root**;**

**}**

void search**(**treeptr root**,** int data**){**

**if** **(**root **==** **NULL){**

printf**(**"key not found\n"**);**

**return;**

**}**

**else** **if** **(**data **==** root**->**data**)**

printf**(**"key found in the BST\n"**);**

**else** **if** **(**data **<** root**->**data**)**

search**(**root**->**left**,** data**);**

**else** **if** **(**data **>** root**->**data**)**

search**(**root**->**right**,** data**);**

**}**

void inorder**(**treeptr root**){**

**if(**root **==** **NULL)**

**return;**

inorder**(**root**->**left**);**

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

inorder**(**root**->**right**);**

**}**

int main**(){**

treeptr root **=** **NULL;**

int key**;**

char ch**=**'y'**;**

**while** **(**ch **==** 'y'**){**

printf**(**"Enter a key to insert in BST\n"**);**

scanf**(**"%d"**,** **&**key**);**

getchar**();**

root **=** insert**(**root**,** key**);**

printf**(**"do you wish to enter another key into BST (y/n)\n"**);**

scanf**(**"%c"**,** **&**ch**);**

**}**

printf**(**"Keys in inorder traversal\n"**);**

inorder**(**root**);**

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

printf**(**"Enter the search Key\n"**);**

scanf**(**"%d"**,** **&**key**);**

search**(**root**,** key**);**

**}**

12

#include<stdio.h>

#include<stdlib.h>

#define TRUE 1

#define FALSE 0

**typedef** struct node**{**

struct node **\***link**;**

int vertex**;**

**}**node**;**

node **\***G**[**20**];**

int visited**[**20**],** n**;**

void insert**(**int vi**,**int vj**){**

node **\***p**,\***q**;**

q**=(**node**\*)**malloc**(sizeof(**node**));**

q**->**vertex**=**vj**;**

q**->**link**=NULL;**

**if(**G**[**vi**]==NULL)**

G**[**vi**]=**q**;**

**else{**

**for(**p**=**G**[**vi**];**p**->**link**!=NULL;** p**=**p**->**link**);**

p**->**link**=**q**;**

**}**

**}**

void read\_graph**(){**

int i**,**vi**,**vj**,**no\_of\_edges**;**

printf**(**"Enter number of vertices:"**);**

scanf**(**"%d"**,&**n**);**

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

G**[**i**]=NULL;**

printf**(**"Enter number of edges \n"**);**

scanf**(**"%d"**,&**no\_of\_edges**);**

**for(**i**=**0**;**i**<**no\_of\_edges**;**i**++){**

printf**(**"Enter an edge(u v):"**);**

scanf**(**"%d%d"**,&**vi**,&**vj**);**

insert**(**vi**,**vj**);**

**}**

**}**

void DFS**(**int i**){**

node **\***p**;**

printf**(**"%5d"**,**i**);**

visited**[**i**]=**TRUE**;**

**for(**p**=**G**[**i**];**p**;** p**=**p**->**link**){**

**if(!**visited**[**p**->**vertex**])**

DFS**(**p**->**vertex**);**

**}**

**}**

int main**(){**

int i**;**

read\_graph**();**

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

visited**[**i**]=**FALSE**;**

printf**(**"\nNodes visited in DFS order\n"**);**

DFS**(**1**);**

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

**return** 0**;**

**}**