DATA STRUCTURES

DAY-3

**IMPLEMENTATION OF LINKED LIST**

#include <stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*h,\*nn,\*t,\*p;

void ibegin()

{

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

printf("enter data ");

scanf("%d",&nn->data);

nn->next=h;

h=nn;

}

void imid()

{

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

printf("enter data ");

scanf("%d",&nn->data);

int i=1,pos;

printf("\nenter insert position ");

scanf("%d",&pos);

t=h;

while(i<pos && t!=0)

{

t=t->next;

i++;

}

nn->next=t->next;

t->next=nn;

}

void iend()

{

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

printf("enter data ");

scanf("%d",&nn->data);

nn->next=0;

t=h;

while(t->next!=0)

{

t=t->next;

}

t->next=nn;

}

void ins()

{

int a;

do

{

printf("\nenter insersion type: 1-b 2-m 3-e\n");

scanf("%d",&a);

switch(a)

{

case 1: ibegin();

break;

case 2: imid();

break;

case 3: iend();

break;

default: printf("invalid choice");

}

}

while(a!=0);

}

void dbegin()

{

t=h;

h=t->next;

t->next=0;

free(t);

}

void dmid()

{

int pos,i=1;

printf("\nenter delete position ");

scanf("%d",&pos);

t=h;

while(i<pos && t!=0)

{

t=t->next;

i++;

}

while(p->next!=t)

p=p->next;

p->next=t->next;

t->next=0;

free(t);

}

void dend()

{

while(t->next!=0)

{

t=t->next;

}

while(p->next!=t)

p=p->next;

p->next=0;

free(t);

}

void del()

{

int b;

do

{

printf("\nenter delete type: 1-b 2-m 3-e\n");

scanf("%d",&b);

switch(b)

{

case 1: dbegin();

break;

case 2: dmid();

break;

case 3: dend();

break;

default: printf("invalid choice");

}

}

while(b!=0);

}

void display()

{

t=h;

while(t!=0)

{

printf("%3d",t->data);

t=t->next;

}

}

int main()

{

h=0;

int c=1;

while(c==1)

{

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

printf("enter data ");

scanf("%d",&nn->data);

nn->next=0;

if(h==0)

{

h=t=nn;

}

else

{

t->next=nn;

t=nn;

}

printf("enter 1-continue 0-stop");

scanf("%d",&c);

}

t=h;

printf("list elements are\n");

while(t!=0)

{

printf("%3d",t->data);

t=t->next;

}

int x;

do

{

printf("\nenter type: 1-insert 2-delete 3-display\n");

scanf("%d",&x);

switch(x)

{

case 1: ins();

break;

case 2: del();

break;

case 3: display();

break;

default: printf("invalid choice");

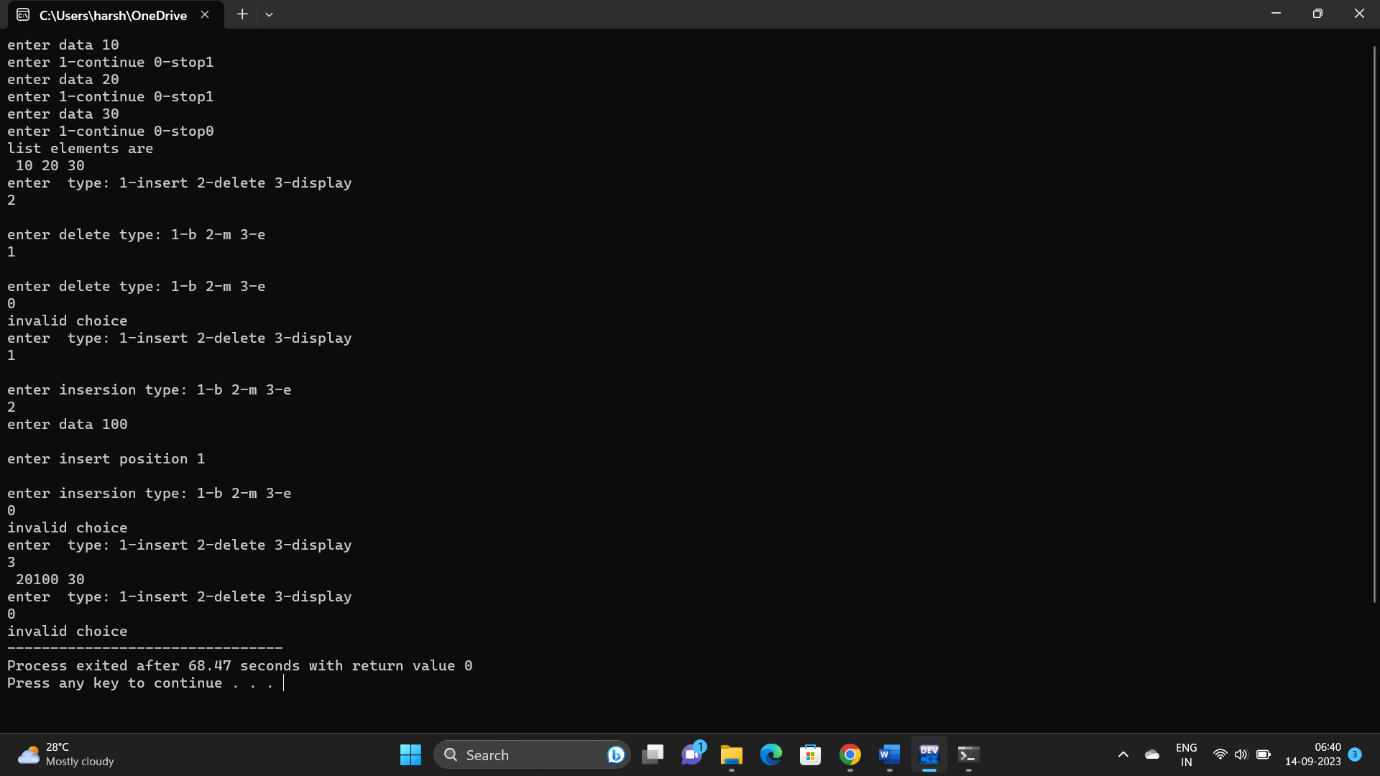
}

}

while(x!=0);

return 0;

}



**MERGING IF TWO LISTS**

#include <stdio.h>

#include<stdlib.h>

int main()

{

struct node

{

int data;

struct node \*next;

};

struct node \*h1,\*nn1,\*t1;

h1=0;

int c1=1;

printf("enter list 1 data\n");

while(c1==1)

{

nn1=(struct node\*)malloc(sizeof(struct node));

printf("enter data ");

scanf("%d",&nn1->data);

nn1->next=0;

if(h1==0)

{

h1=t1=nn1;

}

else

{

t1->next=nn1;

t1=nn1;

}

printf("enter 1-continue 0-stop");

scanf("%d",&c1);

}

t1=h1;

printf("1st list elements are");

while(t1!=0)

{

printf("%3d",t1->data);

t1=t1->next;

}

struct node \*h2,\*nn2,\*t2;

h2=0;

int c2=1;

printf("\nenter list 2 data\n");

while(c2==1)

{

nn2=(struct node\*)malloc(sizeof(struct node));

printf("enter data ");

scanf("%d",&nn2->data);

nn2->next=0;

if(h2==0)

{

h2=t2=nn2;

}

else

{

t2->next=nn2;

t2=nn2;

}

printf("enter 1-continue 0-stop");

scanf("%d",&c2);

}

t2=h2;

printf("\n2nd list elements are");

while(t2!=0)

{

printf("%3d",t2->data);

t2=t2->next;

}

t1=h1;

while(t1->next !=0)

{

t1=t1->next;

}

t1->next=h2;

t1=h1;

printf("\n merged list elements are");

while(t1!=0)

{

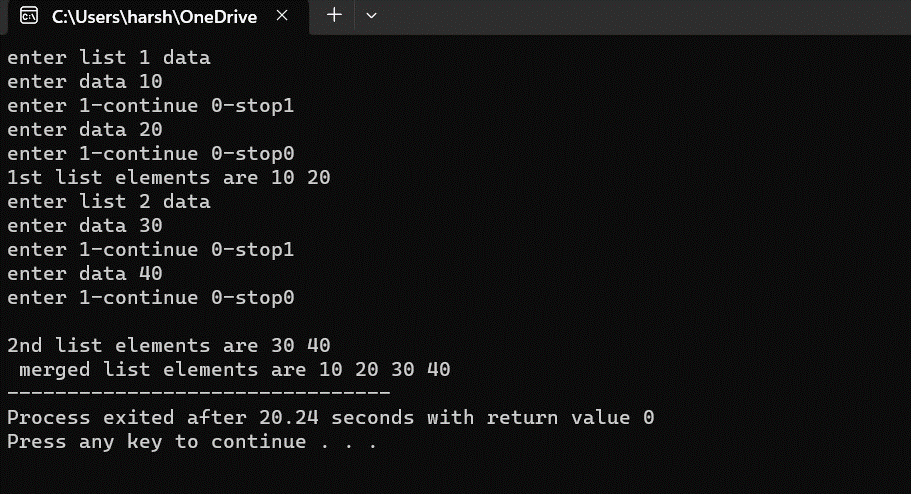
printf("%3d",t1->data);

t1=t1->next;

}

return 0;

}



**STACK IMPLEMENTATION**

#include<stdio.h>

#include<stdlib.h>

int s[10];

int t=-1,n=10;

void push()

{

int x;

printf("enter data");

scanf("%d",&x);

if(t>=n-1)

{

printf("stack is full\n");

}

else

{

t++;

s[t]=x;

}

}

void pop()

{

int item;

if(t==-1)

printf("stack is empty to pop\n");

else

{

item=s[t];

t--;

}

printf("popped element is %2d\n",item);

}

void peek()

{

if(t==-1)

printf("stack is empty\n");

else

printf("peek element is %2d\n",s[t]);

}

void display()

{

int i;

for(i=t;i>=0;i--)

printf("%3d",s[i]);

}

int main()

{

int choice;

do

{

printf("\nenter choice:\n 1-push 2-pop 3-peek 4-display \n");

scanf("%d",&choice);

switch(choice)

{

case 1: push();

break;

case 2: pop();

break;

case 3: peek();

break;

case 4: display();

break;

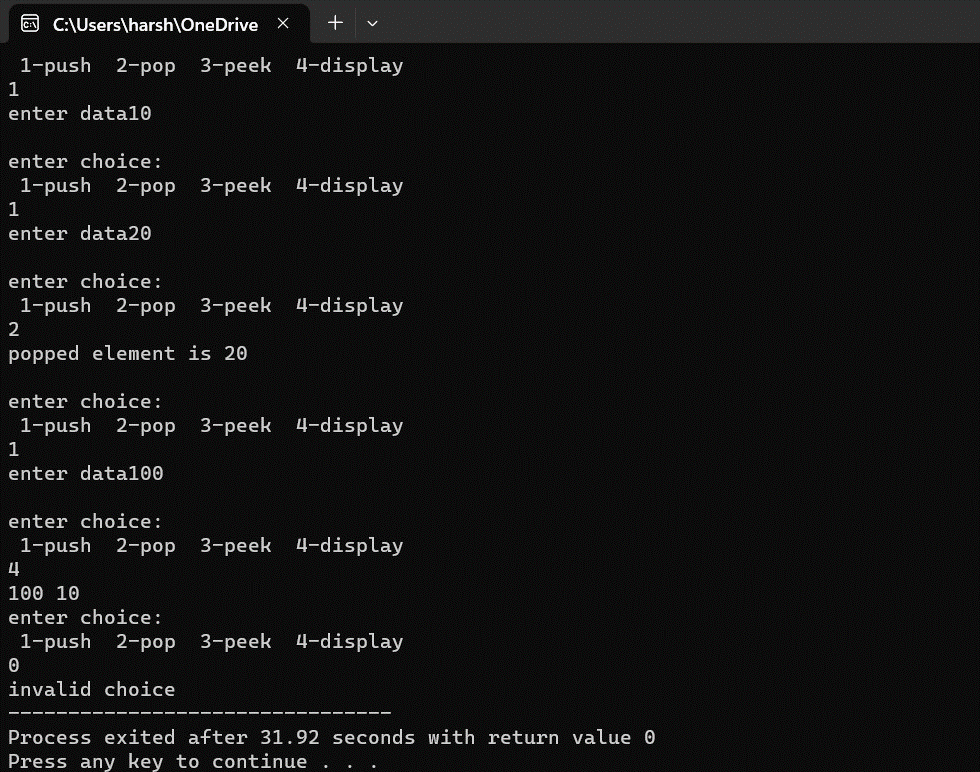
default: printf("invalid choice");

}

}

while (choice!=0);

}



**QUEUE IMPLEMENTATION**

#include<stdio.h>

int q[5],f=-1,r=-1,n=5;

void enqueue()

{

int x;

printf("enter enqueue value");

scanf("%d",&x);

if(r>=n-1)

printf("queue is full");

else if(f==-1 &&r==-1)

{

f++;

r++;

q[r]=x;

}

else

{

r++;

q[r]=x;

}

}

void dequeue()

{

if(r==-1 &&f==-1)

printf("queue is empty to dequeue");

else if(f==r)

f=r=-1;

else

{

printf("dequeued element is %d",q[f]);

f++;

}

}

void display()

{

int i;

for(i=f;i<=r;i++)

printf("%3d",q[i]);

}

int main()

{

int choice;

do

{

printf("\nenter choice:\n 1-enque 2-deque 3-display \n");

scanf("%d",&choice);

switch(choice)

{

case 1: enqueue();

break;

case 2: dequeue();

break;

case 3: display();

break;

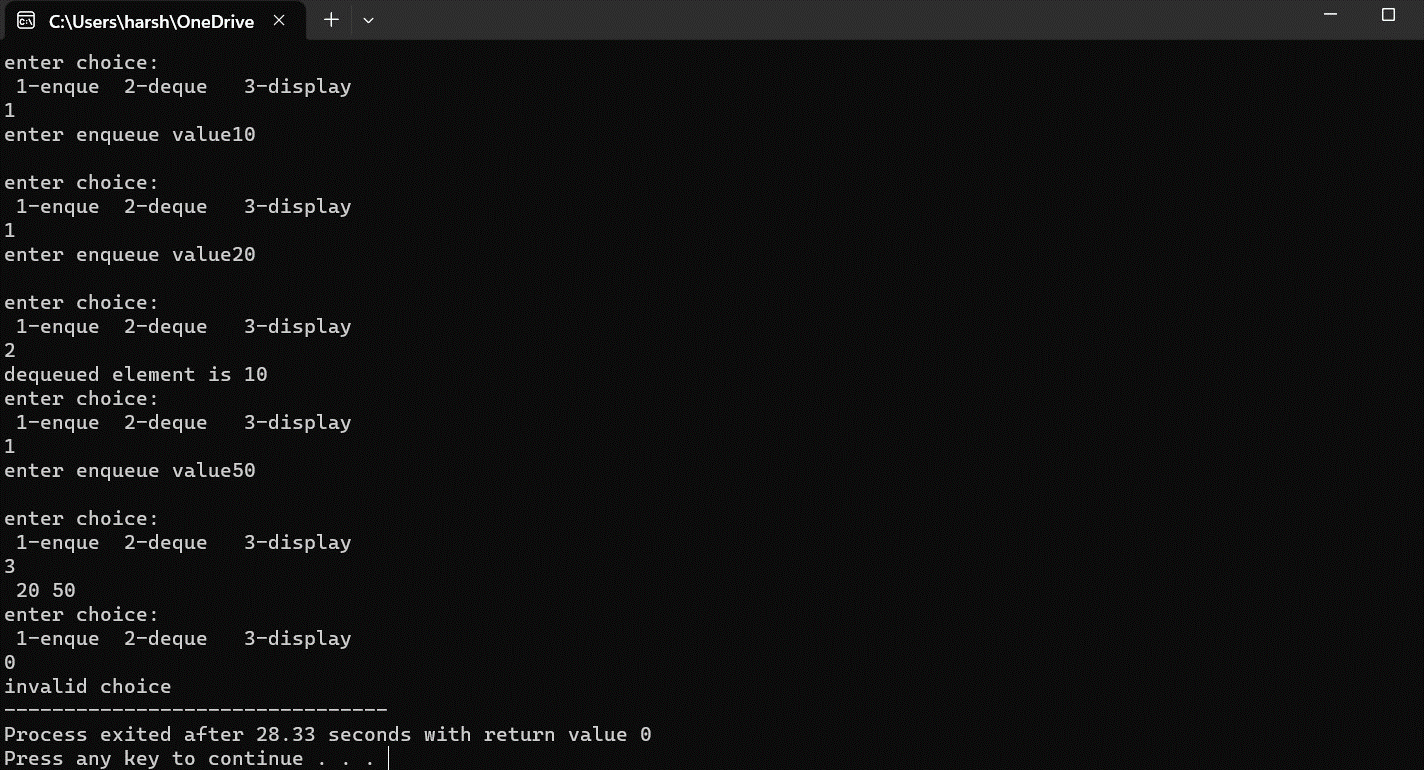
default: printf("invalid choice");

}

}

while (choice!=0);

}



**EVALUATION OF POSTFIX EXPRESION**

#include<stdio.h>

#include<string.h>

#include<ctype.h>

char post[50];

float s[50];

int t=-1;

void push(float c)

{

t++;

s[t]=c;

}

float pop()

{

float x;

x=s[t];

t--;

return x;

}

int main()

{

float v1,v2;

int i;

printf("enter postfix expression\n");

scanf("%s",&post);

for(i=0;post[i]!='\0';i++)

{

if(isdigit(post[i]))

{

push(post[i]-'0');

}

else

{

v1=pop();

v2=pop();

switch(post[i])

{

case '+':

push(v2+v1);

break;

case '-':

push(v2-v1);

break;

case '\*':

push(v2\*v1);

break;

case '/':

push((float)v2/v1);

break;

}

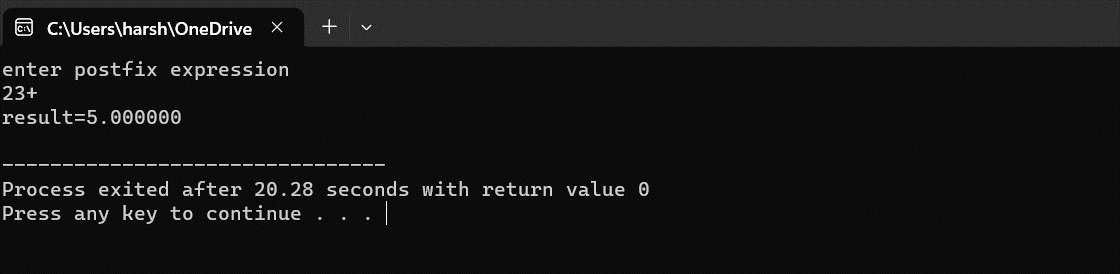
}

}

printf("result=%f\n",s[t]);

return 0;

}



**INFIX TO POSTFIX EXPRESION**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

char s[50],in[50],post[50];

int t=-1;

void push(char);

char pop();

int empty();

void topost();

void print();

int pre(char);

int main()

{

printf("enter infix expression\n");

gets(in);

topost();

print();

return 0;

}

void topost()

{

int i,j=0;

char sym,nxt;

for(i=0;i<strlen(in);i++)

{

sym=in[i];

switch(sym)

{

case '(':

push(sym);

break;

case ')':

while((nxt=pop())!='(')

post[j++]=nxt;

break;

case '+':

case '-':

case '\*':

case '/':

case '^':

while(!empty() && pre(s[t])>=pre(sym))

post[j++]=pop();

push(sym);

break;

default:

post[j++]=sym;

}

}

while(!empty())

post[j++]=pop();

post[j++]='\0';

}

int pre(char sym)

{

switch(sym)

{

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

case '%':

return 4;

default:

return 0;

}

}

void print()

{

int i=0;

printf("postfix expression is\n");

while(post[i])

{

printf("%c",post[i++]);

}

printf("\n");

}

void push(char c)

{

if(t>=50-1)

printf("stack is full\n");

else

{

t++;

s[t]=c;

}

}

char pop()

{

int c;

if(t==-1)

{

printf("stack empty");

}

else

{

c=s[t];

t--;

return c;

}

}

int empty()

{

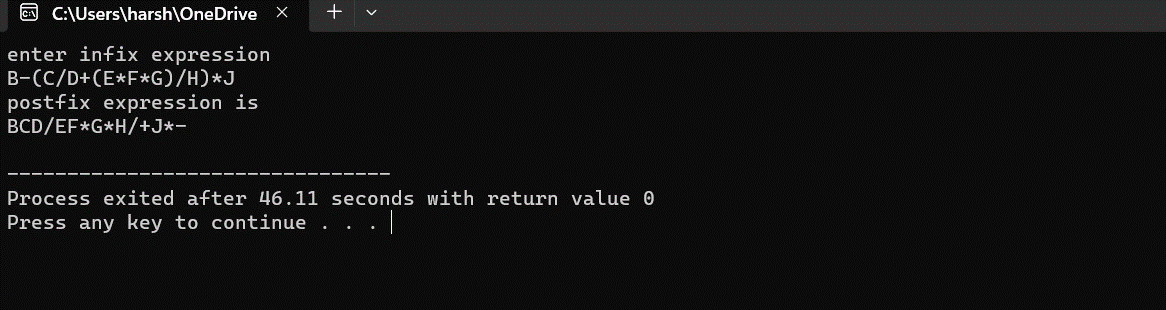
if(t==-1)

return 1;

else

return 0;

}



TREE TRAVERSAL

#include <stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node\* l;

struct node\* r;

};

void inorder(struct node\* root){

if(root==NULL){

return;

}

inorder(root->l);

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

inorder(root->r);

}

void postorder(struct node\* root){

if(root==NULL){

return;

}

postorder(root->l);

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

postorder(root->r);

}

void preorder(struct node\* root){

if(root==NULL){

return;

}

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

preorder(root->l);

preorder(root->r);

}

struct node \*create()

{

int x;

struct node \*nn;

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

printf("enter data (-1 for no node)");

scanf("%d",&x);

if(x==-1)

return 0;

nn->data=x;

printf("enter left child of %d ",x);

nn->l=create();

printf("enter right child of %d ",x);

nn->r=create();

return nn;

}

int main(){

int a;

struct node\* root;

root=create();

printf("enter the traversal type inorder->1 preorder->2 postorder->3:");

scanf("%d",&a);

switch(a){

case 1:inorder(root);

break;

case 2:preorder(root);

break;

case 3:postorder(root);

break;

}

return 0;

}

