12.INFLIX TO POSTFIX

#include<stdio.h>

#include<ctype.h>

#include<string.h>

char stk[20];

int top=-1;

void push(char c)

{

stk[++top]=c;

}

char pop()

{

return(stk[top--]);

}

int priority(char c)

{

if(c=='^'|| c=='&' || c=='|')

return 3;

else if (c=='/'|| c=='\*' || c=='%')

return 2;

else if(c=='+' || c=='-')

return 1;

else

return 0;

}

main()

{

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

int i,j,l;

printf("Enter the string :");

gets(in);

l=strlen(in);

j=0;

for(i=0;i<l;i++)

{

if(isalpha(in[i]))

post[j++]=in[i];

else

{

if(in[i]=='(')

push(in[i]);

else if(in[i]==')')

while((ch=pop())!='(')

post[j++]=ch;

else

{

while(priority(in[i])<=priority(stk[top]))

post[j++]=pop();

push(in[i]);

}

}

}

while(top!=-1)

post[j++]=pop();

post[j]='\0';

printf("\n equivalent infix to postfix is:%s",post);

}

13.SINGLE LINKED LIST

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

}\*p,\*newnode,\*head=NULL,\*t;

void create()

{

int i,n,ele;

printf("Enter the number of elements to be created");

scanf("%d",&n);

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

{

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

printf("Enter the element : ");

scanf("%d",&ele);

newnode->data = ele;

newnode->next = NULL;

if(head==NULL)

{

head=newnode;

p=newnode;

}

else

{

for(p=head; p->next!= NULL; p=p->next);

p->next=newnode;

p=newnode;

}

}

}

void insert\_b()

{

int a;

printf("enter the value");

scanf("%d",&a);

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

newnode->data = a;

newnode->next = head;

head = newnode;

}

void insert\_e()

{

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

int a;

printf("enter the value");

scanf("%d",&a);

newnode->data = a;

newnode->next = NULL;

//struct node \*t = head;

while(t->next != NULL)

{

t = t->next;

}

t->next = newnode;

}

void insert\_any()

{

newnode = malloc(sizeof(struct node));

int a,i,pos;

printf("enter the a value and pos");

scanf("%d%d",&a,&pos);

newnode->data = a;

struct node \*t = head;

for(int i=2; i < pos; i++) {

if(t->next != NULL) {

t = t->next;

}

}

newnode->next = t->next;

t->next = newnode;

}

void delete\_b()

{

int ele;

p=head;

head=p->next;

ele=p->data;

free(p);

printf("The deleted element = %d",ele);

}

void delete\_e()

{

// struct node\* t = head;

int ele;

for(p=head;p->next!=NULL;p=p->next)

t=p;

t->next=NULL;

ele=p->data;

free(p);

printf("The deleted element = %d",ele);

}

void delete\_any\_position()

{

int pos,i,ele;

printf("\nEnter the Position: ");

scanf("%d",&pos);

for(p=head,i=1; i<pos; p=p->next,i++)

t=p;

t->next=p->next;

ele=p->data;

free(p);

printf("\nThe deleted element is ->:%d ",ele);

}

void display()

{

if(head==NULL)

printf("SLL is Empty");

else

for(p=head; p!=NULL; p=p->next)

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

}

void main()

{

int ch;

do

{

printf("\n1.Create\n2.insert\_b\n3.insert\_e\n4.insert\_any\n5.Display\n6.delete\_b\n7.delete\_e\n8.delete at any position\n9.exit\n");

printf("enter the choice");

scanf("%d",&ch);

switch(ch)

{

case 1:

create();

break;

case 2:

insert\_b();

break;

case 3:

insert\_e();

break;

case 4:

insert\_any();

break;

case 5:

display();

break;

case 6:

delete\_b();

break;

case 7:

delete\_e();

break;

case 8:

delete\_any\_position();

break;

case 9:

exit(0);

default:

printf("\n wrong input");

}

} while(ch>=1 && ch<=9);

}

14.TREE CREATION INORDER PREORDER POSTORDER

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

struct node

{

int data;

struct node \*l,\*r;

}\*root=NULL,\*newnode;

struct node\* create(struct node\*root,int ele)

{

if(root==NULL)

{

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

newnode->data=ele;

newnode->l=NULL;

newnode->r=NULL;

return(newnode);

}

else if(ele < root->data)

root->l=create(root->l,ele);

else if(ele > root->data)

root->r=create(root->r,ele);

return(root);

}

void inorder(struct node \*root)

{

if(root==NULL)

return;

inorder(root->l);

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

inorder(root->r);

}

void preorder(struct node \*root)

{

if(root!=NULL)

{

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

preorder(root->l);

preorder(root->r);

}

}

void postorder(struct node \*root)

{

if(root!=NULL)

{

postorder(root->l);

postorder(root->r);

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

}

}

void main()

{

int ch,ele;

do

{

printf("\n\n1.create\n2.inorder\n3.preorder\n4.postorder\n5.exit");

printf("\n\nEnter your choice");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("enter the element");

scanf("%d",&ele);

root=create(root,ele);

break;

case 2:

inorder(root);

break;

case 3:

preorder(root);

break;

case 4:

postorder(root);

break;

case 5:

exit(0);

default:

printf("\nWrong Choice!!");

}

}while(ch>=0 && ch<=5);

}

15.INSERATION SORT

#include<stdio.h>

int insertion\_sort(int A[],int n)

{

int i,j,temp;

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

{

j=i;

while(j>0 && A[j]<A[j-1])

{

temp = A[j-1];

A[j-1]=A[j];

A[j]=temp;

j--;

}

}

}

int main()

{

int i,n,A[100];

printf("\nEnter n");

scanf("%d",&n);

printf("\nEnter Array elements");

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

scanf("%d",&A[i]);

insertion\_sort(A,n);

printf("\nAfter Sorting");

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

printf("%d ",A[i]);

return 0;

}