addl.c

// program to add two singly linked lists

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*link;

}\*p;

void create(struct node \*\*a) // Function to create a singly linked list

{

struct node \*temp;

int num;

printf("\nEnter data: ");

scanf("%d", &num);

if (\*a == NULL)

{

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

temp->data = num;

temp->link = NULL;

\*a = temp;

}

}

void display(struct node \*b)

{

struct node \*temp;

temp = b;

while (temp->link != NULL)

{

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

temp = temp->link;

}

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

}

void append(struct node \*c) // Function to add a node at the end of a singly linked list

{

struct node \*temp, \*r;

int num;

printf("\nEnter data: ");

scanf("%d", &num);

temp = c;

while (temp->link != NULL)

{

temp = temp->link;

}

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

r->data = num;

r->link = NULL;

temp->link = r;

}

void add(struct node \*a, struct node \*b)

{

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

struct node \*list1, \*list2, \*sum,\*r;

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

r->data=0;

r->link=NULL;

p->link=r;

p->data=0;

list1 = a;

list2 = b;

sum = p;

while (list1 != NULL)

{

sum->data=0;

sum->data = list1->data + list2->data;

if(sum->link!=NULL)

sum=sum->link;

list1=list1->link;

list2=list2->link;

}

display(p);

}

int main()

{

struct node \*q = NULL, \*m = NULL, \*p = NULL;

printf("\nlist 1\n");

create(&q);

printf("\nlist 2\n");

create(&m);

printf("\nlist 1 is: ");

display(q);

printf("\nlist 2 is: ");

display(m);

printf("\nappend at list 1\n");

append(q);

printf("\nappend at list 2\n");

append(m);

printf("\nlist 1\n");

display(q);

printf("\nlist 2\n");

display(m);

printf("\nSum\n");

add(q,m);

return 0;

}

concatenate

//prog to concatenate two linked lists

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*link;

};

void create(struct node \*\*a) //Function to create a singly linked list

{

struct node \*temp;

int num;

printf("\nEnter data: ");

scanf("%d",&num);

if(\*a==NULL)

{

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

temp->data=num;

temp->link=NULL;

\*a=temp;

}

}

void display(struct node \*b)

{

struct node \*temp;

temp=b;

while(temp->link!=NULL)

{

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

temp=temp->link;

}

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

}

void append(struct node \*c) //Function to add a node at the end of a singly linked list

{

struct node \*temp,\*r;

int num;

printf("\nEnter data: ");

scanf("%d",&num);

temp=c;

while(temp->link!=NULL)

{

temp=temp->link;

}

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

r->data=num;

r->link=NULL;

temp->link=r;

}

void concatenate(struct node \*\*a,struct node \*\*b)

{

struct node \*temp,\*t;

temp=\*a;

while(temp->link!=NULL)

{

temp=temp->link;

}

temp->link=\*b;

}

int main()

{

struct node \*q=NULL,\*m=NULL;

printf("\nlist 1\n");

create(&q);

printf("\nlist 2\n");

create(&m);

printf("\nlist 1:");

display(q);

printf("\nlist 2:");

display(m);

printf("\nappend at list 1\n");

append(q);

printf("\nappend at list 2\n");

append(m);

printf("\nlist 1:");

display(q);

printf("\nlist 2:");

display(m);

printf("\nConcatenate list 1 and 2:");

concatenate(&q,&m);

display(q);

printf("\n");

return 0;

}

singly2.c

//program to create two singly linked lists

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*link;

};

void create(struct node \*\*a) //Function to create a singly linked list

{

struct node \*temp;

int num;

printf("\nEnter data: ");

scanf("%d",&num);

if(\*a==NULL)

{

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

temp->data=num;

temp->link=NULL;

\*a=temp;

}

}

void display(struct node \*b)

{

struct node \*temp;

temp=b;

while(temp->link!=NULL)

{

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

temp=temp->link;

}

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

}

void append(struct node \*c) //Function to add a node at the end of a singly linked list

{

struct node \*temp,\*r;

int num;

printf("\nEnter data: ");

scanf("%d",&num);

temp=c;

while(temp->link!=NULL)

{

temp=temp->link;

}

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

r->data=num;

r->link=NULL;

temp->link=r;

}

int main()

{

struct node \*q=NULL,\*m=NULL;

printf("\nlist 1\n");

create(&q);

printf("\nlist 2\n");

create(&m);

display(q);

display(m);

printf("\nappend at list 1\n");

append(q);

printf("\nappend at list 2\n");

append(m);

printf("\nlist 1\n");

display(q);

printf("\nlist 2\n");

display(m);

return 0;

}

reverse.c

//program to reverse a singly linked lists

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*link;

};

void create(struct node \*\*a) //Function to create a singly linked list

{

struct node \*temp;

int num;

printf("\nEnter data: ");

scanf("%d",&num);

if(\*a==NULL)

{

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

temp->data=num;

temp->link=NULL;

\*a=temp;

}

}

void display(struct node \*b)

{

struct node \*temp;

temp=b;

while(temp->link!=NULL)

{

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

temp=temp->link;

}

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

}

void append(struct node \*\*c) //Function to add a node at the end of a singly linked list

{

struct node \*temp,\*r;

int num;

printf("\nEnter data: ");

scanf("%d",&num);

temp=\*c;

while(temp->link!=NULL)

{

temp=temp->link;

}

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

r->data=num;

r->link=NULL;

temp->link=r;

}

void reverse(struct node \*\*r)

{

struct node \*prev,\*middle,\*next;

prev=\*r;

middle=\*r;

next=\*r;

middle=middle->link;

next=next->link->link;

while(middle!=NULL)

{

middle->link=prev;

if (prev==\*r)

{prev->link=NULL;}

prev=middle;

middle=next;

if(next!=NULL)

{next=next->link;}

}

\*r=prev;

}

int main()

{

struct node \*q=NULL;

printf("\nlist 1\n");

create(&q);

display(q);

printf("\nlist 1\n");

display(q);

append(&q);

printf("\n REVERSE LIST 1\n");

if(q->link!=NULL)

{reverse(&q);}

display(q);

printf("\n");

return 0;

}

infixtopostfix.c

#include <stdio.h>

#include<string.h>

#include<ctype.h>

#define MAX 9

char stack[MAX];

int tos=-1;

void push(char a)

{

if(tos==MAX-1)

{

printf("\nStack Overflow");

return;

}

tos++;

stack[tos]=a;

}

void pop()

{

char a;

if(tos==-1)

{

printf("\nStack Underflow");

return;

}

a=stack[tos];

tos--;

printf("%c",a);

}

int precedence(char a)

{

int precedence;

if(a=='$')

{

precedence=3;

}

else if(a=='\*' || a=='/')

{

precedence=2;

}

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

{

precedence=1;

}

else if(a=='(')

{

precedence=0;

}

return precedence;

}

int main()

{

char expr[20];

printf("Enter expr: ");

scanf("%s",expr);

int n= strlen(expr);

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

{

if(isdigit(expr[i]))

{

printf("%c",expr[i]);

}

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

{

push(expr[i]);

}

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

{

while(stack[tos]!='(')

{pop();}

tos--;

}

else

{

if(tos==-1 || precedence(expr[i])>precedence(stack[tos]))

{

push(expr[i]);

}

else

{

while(precedence(expr[i])<=precedence(stack[tos]) && tos!=-1)

{

pop();

}

push(expr[i]);

}

}

}

while(tos!=-1)

{

pop();

}

return 0;

}

postfixtoinfix.c

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <math.h>

#define MAX 9

int stack[MAX],tos=-1;

void push(int a)

{

if(tos==MAX-1)

{

printf("\nStack overflow");

}

tos++;

stack[tos]=a;

}

int pop()

{

if(tos==-1)

{

printf("\nStack underflow");

}

int item=stack[tos];

tos--;

return item;

}

int main()

{

int n,op2,op1,ans;

char eval[100];

printf("Enter the size of the postfix expression: ");

scanf("%d",&n);

printf("\nEnter the postfix expression: ");

scanf("%s",eval);

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

{

if(isdigit(eval[i])!=0)

{

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

}

else

{

op2=pop();

op1=pop();

if(eval[i]=='+')

{

ans=op1+op2;

push(ans);

}

else if(eval[i]=='-')

{

ans=op1-op2;

push(ans);

}

else if(eval[i]=='\*')

{

ans=op1\*op2;

push(ans);

}

else if(eval[i]=='/')

{

ans=op1/op2;

push(ans);

}

else if(eval[i]=='$')

{

ans=pow(op1,op2);

push(ans);

}

}

}

ans=pop();

printf("%d",ans);

printf("\n");

}