addl.c

```
// program to add two singly linked lists and store the sum in a third linked list
#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.c

```
//program to concatenate 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;
}

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

```c
//Program to convert an infix expression to postfix
#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

```c
//Program to evaluate a postfix expression
#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");
}
```

stacksusqueues.c

```c
//Program to implement stacks using queues
#include<stdio.h>
#include<stdlib.h>
#define MAX 2



void enqueue(int *q,int *pfront,int *prear,int data)
{
if(*prear==MAX-1)
{
printf("\nQueue Overflow");
return;
}
if(*pfront==-1)
{
(*pfront)++;
}
(*prear)++;
*(q + *prear)=data;

}

int dequeue(int *q,int *pfront,int *prear)
{

int temp=*(q+*pfront);
if(*pfront==*prear)
{
*pfront=*prear=-1;
}
else{
(*pfront)++;
}
return temp;
}

void display(int *q1,int *pfront,int *prear)
{
printf("\n");
for (int i = *pfront; i <=*prear ; i++)
{
printf("%d\t",*(q1+i));
}
printf("\n");

}
int main()
```

```
{
int q1[MAX],q2[MAX],q1front=-1,q1rear=-1,q2front=-1,q2rear=-1;
int c,o,data;
do
{
printf("Enter 1 to push an element\nEnter 2 to pop an element\n");
scanf("%d",&o);
switch (o)
{
case 1:
printf("\nEnter data: ");
scanf("%d",&data);
enqueue(&q1[0],&q1front,&q1rear,data);
display(&q1[0],&q1front,&q1rear);
break;
case 2:
int a;
for(int i = 0 ; i < q1rear ; i++)
{
enqueue(&q2[0],&q2front,&q2rear,dequeue(&q1[0],&q1front,&q1rear));
}
dequeue(&q1[0],&q1front,&q1rear);
for(int i = 0 ; i <= q2rear ; i++)
{
enqueue(&q1[0],&q1front,&q1rear,dequeue(&q2[0],&q2front,&q2rear));
}
if(q1rear!=-1)
{display(&q1[0],&q1front,&q1rear);}
else{
printf("\nQueue Underflow");
}
break;
default:
printf("\nWrong choice");
break;
}
printf("\nDo you want to comtinue(1/0): ");
scanf("%d",&c);
} while (c==1);
return 0;
}
```

queuesusstacks.c

```
//Program to implement queues using stacks
#include <stdio.h>
#define MAX 3

void push(int *stack,int data,int *tos)
{
if(*tos==MAX-1)
{
printf("\nStack Overflow\n");
return;
}
(*tos)++;
*(stack+*tos)=data;
}

int pop(int *stack,int *tos)
{
int temp=*(stack+*tos);
(*tos)--;
return temp;
}

void display(int *stack,int *tos)
{
for (int i = 0; i <= *tos; i++)
{
printf("%d\t",*(stack+i));
}
}
int main()
{
int s1[MAX],s2[MAX],tos1=-1,tos2=-1;
int c,o,data;
do
{
printf("Enter 1 to enqueue\nEnter 2 to dequeue\n");
scanf("%d",&o);
switch (o)
{
case 1:
printf("\nEnter data: ");
scanf("%d",&data);
push(&s1[0],data,&tos1);
display(&s1[0],&tos1);
break;
case 2:
for(int i = 0;i<tos1;i++)
{
push(&s2[0],pop(&s1[0],&tos1),&tos2);
```

```
}
pop(&s1[0],&tos1);
for(int i = 0;i<=tos2;i++)
{
push(&s1[0],pop(&s2[0],&tos2),&tos1);
}
if(tos1!=-1)
{display(&s1[0],&tos1);}
else{printf("\nStack Underflow\n");}
break;
default:
printf("\nWrong choice");
break;
}
printf("\nDo you want to comtinue(1/0): ");
scanf("%d",&c);
} while (c==1);
return 0;
}
```

revstringusstacks.c

```
//Program to reverse a string using stacks
#include <stdio.h>
#define MAX 3

void push(int *stack,int data,int *tos)
{
if(*tos==MAX-1)
{
printf("\nStack Overflow\n");
return;
}
(*tos)++;
*(stack+*tos)=data;
}

int pop(int *stack,int *tos)
{
int temp=*(stack+*tos);
(*tos)--;
return temp;
}

void display(int *stack,int *tos)
{
for (int i = 0; i <= *tos; i++)
{
printf("%d\t",*(stack+i));
}
}
int main()
{
int s1[MAX],s2[MAX],tos1=-1,tos2=-1;
int c,o,data;
do
{
printf("Enter 1 to enqueue\nEnter 2 to dequeue\n");
scanf("%d",&o);
switch (o)
{
case 1:
printf("\nEnter data: ");
scanf("%d",&data);
push(&s1[0],data,&tos1);
display(&s1[0],&tos1);
break;
case 2:
for(int i = 0;i<tos1;i++)
{
push(&s2[0],pop(&s1[0],&tos1),&tos2);
}
```

```
pop(&s1[0],&tos1);
for(int i = 0;i<=tos2;i++)
{
push(&s1[0],pop(&s2[0],&tos2),&tos1);
}
if(tos1!=-1)
{display(&s1[0],&tos1);}
else{printf("\nStack Underflow\n");}
break;
default:
printf("\nWrong choice");
break;
}
printf("\nDo you want to comtinue(1/0): ");
scanf("%d",&c);
} while (c==1);
return 0;
}
```

circularqueues.c

```
//Program to implement circular queues
#include <stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *link;
}*front=NULL,*rear=NULL;

void enqueue(int data)
{
struct node *r;
r=(struct node *)malloc(sizeof(struct node));
r->data=data;
r->link=front;
if(rear==NULL)
{
front=rear=r;
return;
}
rear->link=r;
rear=r;
}

void dequeue()
{
if(rear==NULL)
{
printf("\nQueue Underflow\n");
}
struct node *temp=front;
if (front==rear)
{
front=NULL;
rear=NULL;
free(temp);
return;
}
front=front->link;
rear->link=front;
free(temp);
}
void display()
{
if(rear==NULL)
{
printf("\nNULL");
return;
}
struct node *temp=front;
```

```
while (temp!=rear)
{
printf("%d--->",temp->data);
temp=temp->link;
}
printf("%d",rear->data);

}
int main()
{
int c,o,data;
do
{
printf("Enter 1 to enqueue\nEnter 2 to dequeue\n");
scanf("%d",&o);
switch (o)
{
case 1:
printf("\nEnter data: ");
scanf("%d",&data);
enqueue(data);
display();
break;
case 2:
dequeue();
display();
break;
default:
printf("\nWrong choice");
break;
}
printf("\nDo you want to comtinue(1/0): ");
scanf("%d",&c);
} while (c==1);
return 0;

}
```