STACK APPLICATION:1- PARANTHESIS MATCHING

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<string.h>

typedef struct Stack{
   int size;
   int top;
   int *S; //pointer pointing to integer array
}Stack;

void CREATE(Stack *);
void PUSH(Stack *,int);
void DISPLAY(Stack *st);
int POP(Stack *);
void PEEK(Stack *st);
int isEmpty(Stack *st);
 int isFull(Stack *st);
 int isBalance(char *exp);

int main(){

   Stack st;
    char exp[]="((a+b)*(a-b))";
   if(isBalance(exp)){
      printf("BALANCED\n");
   }else{
      printf("NOT BALANCED\n");
   }
```

```
    return 0;


}
int isEmpty(Stack *st){
    return st->top == -1;
 }


 int isFull(Stack *st){
    return st->top == st->size - 1;
 }


 void PUSH(Stack *st,int data){
    if(st->top==st->size-1){
        printf("\nOVERFLOW\n");
    }else{
        st->top++;
        st->S[st->top]=data;
    }


 }
 int POP(Stack *st){
    int x=0;
    if(st->top==-1){
        printf("\nSTACK UNDERFLOW\n");
    }else{
        x=st->S[st->top];
        st->top--;


    }
    return x;
```

```
}


int isBalance(char *exp){
   Stack st;
   st.size=strlen(exp);
   st.top=-1;
   st.S=(int*)malloc(st.size*sizeof(int));


   for(int i=0;i!=exp[i]!='\0';i++){
      if(exp[i]=='('){
         PUSH(&st,'(');
      }
      else if(exp[i]==')'){
         if(isEmpty(&st)){
            return false;
         }else{
            POP(&st);
         }


      }


   }
    return isEmpty(&st)?1:0;
}
```

<u>STACK APPLICATION:2 - INFIX TO POSTFIX CONVERSION</u>

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
```

```
typedef struct Stack{
   int size;
   int top;
   char *S;
}Stack;

void CREATE(Stack *st);
int isEmpty(Stack *st);
int isFull(Stack *st);
void PUSH(Stack *st,int);
char POP(Stack *st);
void DISPLAY();
int precedence(char);
int isOperator(char);
void infix_postfix(char *exp);

int main(){


   char exp[]="a+b*c";
   infix_postfix(exp);



   return 0;
}
// void CREATE(Stack *st){
//     printf("Enter the size of stack");
//     scanf("%d",&st->size);


//     st->top==-1;
//     st->S=(int*)malloc(st->size*sizeof(int));
// }

int isEmpty(Stack *st){
    return st->top==-1;

}

int isFull(Stack *st){
    return st->top==st->size-1;
}

void DISPLAY(Stack *st){
    for(int i=st->top;i>=0;i--){
        printf("%d\n",st->S[i]);
    }

}
void PUSH(Stack *st,int data){
    if(isFull(st)){
        printf("STACK OVERFLOW\n");

    }else{
        st->top++;
        st->S[st->top]=data;
    }
}

char POP(Stack *st) {
    if (isEmpty(st)) {
        printf("STACK UNDERFLOW\n");
        return '\0';
    } else {
        return st->S[st->top--];
    }
}


int precedence(char c){
    if(c=='+' || c=='-'){
        return 1;
    }else if(c=='*' ||  c=='/'){
        return 2;
    }else if(c=='^'){
        return 3;
    }else{
        return 0;
    }
}

int isOperator(char c){
    return c=='+' || c=='-'||c=='*'||c=='/'||c=='^';
}

void infix_postfix(char *exp){
    Stack st;

    int n=strlen(exp);
    st.S=(char *)malloc(n*sizeof(char));
    st.top=-1;
    char postfix[n+1];
    int j=0;
```

```
for(int i=0;exp[i]!='\0';i++){
    if (isalnum(exp[i])){
          postfix[j++]=exp[i];

    }else if(isOperator(exp[i])){
       while(!isEmpty(&st) && precedence(st.S[st.top])>=precedence(exp[i])){
        postfix[j++]=POP(&st);




       }
       PUSH(&st,exp[i]);

       }else if(exp[i]=='('){
        PUSH(&st,exp[i]);
       }
       else if(exp[i]==')'){
        while(!isEmpty(&st) && st.S[st.top]!='('){
            postfix[j++]=POP(&st);
        }
          POP(&st);
       }

    }

    while(!isEmpty(&st)){
        postfix[j++]=POP(&st);



    }
```

```
        postfix[j]='\0';
        printf("%s",postfix);



    }
```

REVERSE A STRING USING STACK

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct Node{
    int size;
    int top;
    char *S;
}Stack;

int isEmpty(Stack *st);
int isFull(Stack *st);
void CREATE(Stack *st);
int POP(Stack *st);
void PUSH(Stack *st,int data);
void DISPLAY(Stack *st);
void REVERSE(char *str);

int main(){

    Stack st;
    char str[]="amritha";
    REVERSE(str);
    return 0;
}

int isEmpty(Stack *st){
    return st->top==-1;
}
int isFull(Stack *st){
    return st->top==st->size-1;
}
void PUSH(Stack *st,int data){
    if(isFull(st)){
        printf("stack overflow\n");
    }else{
        st->top++;
        st->S[st->top]=data;
    }
}

int POP(Stack *st){
    if(isEmpty(st)){
        printf("stack underflow\n");
    }else{
         return st->S[st->top--];
    }
}

void REVERSE(char *str){
    Stack st;
    st.size=strlen(str);
    st.S=(char*)malloc(st.size*sizeof(char));
    st.top=-1;
     printf("original string = %s ",str);
    for(int i=0;str[i]!='\0';i++)

    {
        PUSH(&st,str[i]);
    }

    printf("\nreversed string ");
    while(!isEmpty(&st)){
        printf("%c",POP(&st));
    }



}

void DISPLAY(Stack *st){

    if(isEmpty(st)){
        printf("no elements\n");
    }else{
        for(int i=st->top;i>=0;i--){
            printf("%c",st->S[i]);
        }
    }
}
```

Implementation of queue using array

```
#include<stdio.h>

#include<stdlib.h>


typedef struct Queue{

   int size;

   int front;

   int rear;

   int *Q;


}Queue;


int isEmpty(Queue *q);

int isFull(Queue *q);

void Create(Queue *q);

void Display(Queue *q);

void Enqueue(Queue *q,int data);

int Dequeue(Queue *q);


int main(){

   Queue q;

   Create(&q);

   Enqueue(&q,100);

   Enqueue(&q,200);

   Enqueue(&q,200);

   Display(&q);

   int res=Dequeue(&q);

   printf("deleted element =%d\n",res);

   Display(&q);
```

```
}

void Create(Queue *q){
   printf("Enter the size of queue");
   scanf("%d",&q->size);

   q->Q=(int *) malloc(q->size*sizeof(int));
   q->front=q->rear=-1;
}

int isEmpty(Queue *q){
   return q->front==q->rear;
}
int isFull(Queue *q){
   return q->rear==q->size-1;
}
void Enqueue(Queue *q,int data){
   if(isFull(q)){
      printf("queue is full\n");
   }else{
      q->rear++;
      q->Q[q->rear]=data;
   }
}
int Dequeue(Queue *q){
   int x=0;
   if(isEmpty(q)){
      printf("queue is empty\n");
   }else{
        q->front++;
        x=q->Q[q->front];


    }
     return x;
}

void Display(Queue *q){
    if(isEmpty(q)){
        printf("queue is empty\n");
    }else{
        for(int i=q->front+1;i<=q->rear;i++){
            printf("%d\n",q->Q[i]);
        }
    }
}
```

<u>**Simulate a Call Center Queue**</u>

Create a program to simulate a call center where incoming calls are handled on a

first-come, first-served basis. Use a queue to manage call handling and provide options to add, remove, and view calls.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct{
    int id;
    char problem[100];

}call_detail;
```

```
typedef struct Queue{
   int size;
   int front;
   int rear;
   call_detail *Q;

}Queue;

int isEmpty(Queue *q);
int isFull(Queue *q);
void Create(Queue *q);
void Display(Queue *q);
void Enqueue(Queue *q);
void Dequeue(Queue *q);

int main(){
   Queue q;
   Create(&q);

   while(1){
      int ch;
      printf("enter choice\n");
      printf("1.Add call\n2.Remove call\n3.view calls\n4.exit\n");
      scanf("%d",&ch);

      switch (ch)
      {
      case 1:
        Enqueue(&q);
        printf("\n");
```

```
            break;
             case 2:
          Dequeue(&q);
          printf("\n");

            break;
             case 3:
          Display(&q);
          printf("\n");

            break;
             case 4:
         printf("Existing the system\n");
         return 0;

            break;

        default:
        printf("invalid option");
            break;
        }
    }



    return 0;
}
int isEmpty(Queue *q){
    return q->front==q->rear;
}
```

```
int isFull(Queue *q){
   return q->rear==q->size-1;
}
void Create(Queue *q){
   printf("Enter the maximum number of call records that can hold\n");
   scanf("%d",&q->size);

   q->Q=(call_detail*)malloc(q->size*sizeof(call_detail));
   q->front=q->rear=-1;
}
void Enqueue(Queue *q){
   if(isFull(q)){
      printf("Queue is full\n");
   }else{
      q->rear++;
      printf("Enter caller ID : ");
      scanf("%d",&q->Q[q->rear].id);
      printf("Enter problem : ");
      scanf("%s",q->Q[q->rear].problem);
      printf("Call details added successfully\n\n");
   }
}

void Dequeue(Queue *q){
   int id;
   char p[100];

   if (isEmpty(q))
   {
      printf("queue is empty\n");
   }else{
   q->front++;

   id=q->Q[q->front].id;

   strcpy(p,q->Q[q->front].problem);


   printf("Removed call: Customer %d - %s\n",id,p);

   }


}

void Display(Queue *q){

   if(isEmpty(q)){

      printf("queue is empty\n");

   }else{

      for(int i=q->front+1;i<=q->rear;i++){

          printf("Customer %d - %s\n",q->Q[i].id,q->Q[i].problem);

      }

   }

}
```

<u>Print Job Scheduler</u>

Implement a print job scheduler where print requests are queued.

Allow users to add new print jobs, cancel a specific job, and print jobs in the order they were added.

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>


typedef struct{
   int id;

   char job[100];


}job_detail;


typedef struct Queue{

   int size;

   int front;

   int rear;

   job_detail *Q;


}Queue;


int isEmpty(Queue *q);

int isFull(Queue *q);

void Create(Queue *q);

void Display(Queue *q);

void Enqueue(Queue *q);

void Dequeue(Queue *q);


int main(){

   Queue q;

   Create(&q);


   while(1){

      int ch;

      printf("enter choice\n");

      printf("1.Add new\n2.Remove specific job\n3.view all jobs\n4.exit\n");

      scanf("%d",&ch);


      switch (ch)
   {
   case 1:
     Enqueue(&q);
     printf("\n");

       break;
        case 2:
     Dequeue(&q);
     printf("\n");

       break;
        case 3:
     Display(&q);
     printf("\n");

       break;
        case 4:
    printf("Existing the system\n");
    return 0;

       break;

   default:
   printf("invalid option");
       break;
   }
}




return 0;
```

```
}


int isEmpty(Queue *q){
    return q->front==q->rear;
}
int isFull(Queue *q){
    return q->rear==q->size-1;
}
void Create(Queue *q){
    printf("Enter the maximum number of record\n");
    scanf("%d",&q->size);


    q->Q=(job_detail*)malloc(q->size*sizeof(job_detail));
    q->front=q->rear=-1;
}
void Enqueue(Queue *q){
    if(isFull(q)){
        printf("job list is full\n");
    }else{
        q->rear++;
        printf("Enter job ID : ");
        scanf("%d",&q->Q[q->rear].id);
        printf("Enter job name : ");
        scanf("%s",q->Q[q->rear].job);
        printf("job details added successfully\n\n");
    }
}
void Dequeue(Queue *q){
    int id;
    char p[100];
    if (isEmpty(q))
    {
        printf("job list is empty\n");
    }else{




    q->front++;
    id=q->Q[q->front].id;
    strcpy(p,q->Q[q->front].job);

    printf("Removed job: job id %d - %s\n",id,p);
    }
}

void Display(Queue *q){
    if(isEmpty(q)){
        printf("queue is empty\n");
    }else{
        for(int i=q->front+1;i<=q->rear;i++){
            printf("Customer %d - %s\n",q->Q[i].id,q->Q[i].job);
        }
    }
}
```

<u>**Design a Ticketing System**</u>

Simulate a ticketing system where people join a queue to buy tickets.

Implement functionality for people to join the queue, buy tickets, and display the queue's current state.

```
#include<stdio.h>
#include<stdlib.h>
```

```
#include<string.h>

typedef struct{
    int id;
    char name[100];

}call_detail;

typedef struct Queue{
    int size;
    int front;
    int rear;
    call_detail *Q;

}Queue;

int isEmpty(Queue *q);
int isFull(Queue *q);
void Create(Queue *q);
void Display(Queue *q);
void Enqueue(Queue *q);
void Dequeue(Queue *q);

int main(){
    Queue q;
    Create(&q);

    while(1){
        int ch;
        printf("enter choice\n");
        printf("1.Join Queue\n2.Buy Ticket\n3.view Queue\n4.exit\n");
    scanf("%d",&ch);

    switch (ch)
    {
    case 1:
      Enqueue(&q);
      printf("\n");

        break;
         case 2:
      Dequeue(&q);
      printf("\n");

        break;
         case 3:
      Display(&q);
      printf("\n");

        break;
         case 4:
     printf("Existing the system\n");
     return 0;

        break;

    default:
    printf("invalid option");
        break;
    }
}
```

```
    return 0;
}
int isEmpty(Queue *q){
    return q->front==q->rear;
}
int isFull(Queue *q){
    return q->rear==q->size-1;
}
void Create(Queue *q){
    printf("Enter the maximum number of person can enter into queue\n");
    scanf("%d",&q->size);

    q->Q=(call_detail*)malloc(q->size*sizeof(call_detail));
    q->front=q->rear=-1;
}
void Enqueue(Queue *q){
    if(isFull(q)){
        printf("Ticket is full\n");
    }else{
        q->rear++;
        printf("Enter caller ID : ");
        scanf("%d",&q->Q[q->rear].id);
        printf("Enter Name : ");
        scanf("%s",q->Q[q->rear].name);
        printf(" added into queue successfully\n\n");
    }
}

void Dequeue(Queue *q){
    int id;
    char p[100];

    if (isEmpty(q))
    {
        printf("queue is empty\n");
    }else{



    q->front++;
    id=q->Q[q->front].id;
    strcpy(p,q->Q[q->front].name);

    printf("Ticket booked: Customer ID: %d - %s\n",id,p);
    }

}
void Display(Queue *q){
    if(isEmpty(q)){
        printf("queue is empty\n");
    }else{
        for(int i=q->front+1;i<=q->rear;i++){
            printf("Customer %d - %s\n",q->Q[i].id,q->Q[i].name);
        }
    }
}
```