```c
/*
Experiment No:      4
Date:               August 21,2019
Aim:                To study and implement different linear data structures using arrays –
                    Stack ADT, Queue ADT, and Circular Queue ADT
Problem Statement:  Use an array-based allocation to initialize a Stack, a Queue, and a Circular
                    Queue and implement the permissible operations on them. Write a menudriven
                    program in C to test these data structures.
                    The solution involving use of structure(s) to realize the mentioned data
                    structures will be preferred (but not essential).
*/
#include<stdio.h>
#include<stdlib.h>
#define MN_VAL -99
#define MX_SIZE 5
//function prototypes
int initializeStk(int *);
int initializeQ(int *);
int pushStk(int *,int *,int *);
int popStk(int *,int *,int *);
int isEmptyStk(int *,int );
int isFullStk(int *,int );
int topvalStk(int *,int *);
int dispStk(int *,int *);
int dispQ(int *,int *,int *);
int isEmptyQ(int *);
int insertQ(int *,int *,int *,int );
int isFullQ(int *);
int deleteQ(int *,int *,int *);
int insertCQ(int *,int *,int *,int );
int deleteCQ(int *,int *,int *);
int isEmptyCQ(int *);
int isFullCQ(int *,int *);
int dispCQ(int *,int *,int *);
//main
int main(){
    int a[MX_SIZE],list_type,top,choice0,ele1;
    int front,rear,ele,flag,choice1,choice2,choice3,choice01,choice02,frontCQ,rearCQ,key;
    do{
    printf("Enter the type of list\n");
    printf("1.Stack\t2.Queue\t3.Circular Queue:");
    scanf("%d",&list_type);
    switch(list_type){
      case 1:top=initializeStk(a);
        do{
          printf("The operations available are=>\n");
          printf("1.Push 2.Pop 3.isFull 4.IsEmpty 5.Top value 6.Display 0.Exit:");
          scanf("%d",&choice0);
          switch(choice0){
            case 1: printf("Enter the element?");
                    scanf("%d",&ele1);
                    flag=pushStk(a,&ele1,&top);
                    if(flag==1){
                        printf("The List is full\n");
                    }
                    dispStk(a,&top);
                    break;
            case 2:flag=popStk(a,&ele1,&top);
                   if(flag==1){
                        printf("The list is empty,pop failed\n");
                   }
                   else
                        printf("The popped element is %d\n",flag);
                        dispStk(a,&top);
                   break;
            case 3:flag=isFullStk(a,top);
```

```
                if(flag==1)
                     printf("The list is full\n");
                else
                     printf("The list is not full\n");
                break;
         case 4:flag=isEmptyStk(a,top);
                if(flag==-1)
                     printf("The list is empty\n");
                else
                     printf("The list contains some elements\n");
                break;
         case 5:flag=topvalStk(a,&top);
                if(flag!=1)
                     printf("The element at top is %d",flag);
                if(flag==1)
                     printf("The list is empty \n");
                break;
         case 6:flag=dispStk(a,&top);
                if(flag==1){
                     printf("The list is empty \n");
                }
                dispStk(a,&top);
                break;
         case 0:break;
       }
     printf("Any other operation,press 1:");
     scanf("%d",&choice1);
}while(choice1==1);
break;
  case 2:front=initializeQ(a);
         rear=front;
  do{
      printf("The operations available are=>\n");
      printf("1.Insert 2.Delete 3.isEmpty 4.isFull 5.Display 0.Exit:");
      scanf("%d",&choice01);
      switch(choice01){
        case 3:flag=isEmptyQ(&front);
               if(flag==1){
                    printf("The Queue is empty\n");
              }
              break;
        case 1:printf("Enter the element to be inserted=>");
               scanf("%d",&ele);
               flag=insertQ(a,&front,&rear,ele);
               if(flag==1){
                    printf("The List is full\n");
               }
               dispQ(a,&front,&rear);
               break;
        case 4:flag=isFullQ(&rear);
               if(flag==1){
                    printf("The Queue is full\n");
               }
               else
                    printf("The Queue has space available\n");
               break;
        case 2:flag=deleteQ(a,&front,&rear);
               if(flag==1){
                    printf("The list is empty,delete failed\n");
               }
               dispQ(a,&front,&rear);
               break;
        case 5:flag=dispQ(a,&front,&rear);
               if(flag==1){
                    printf("The list is empty \n");
               }
```

```c
                    dispQ(a,&front,&rear);
                    break;
             case 0:break;
             }
           printf("Any other operation,press 1: ");
           scanf("%d",&choice2);
     }while(choice2==1);
     break;
       case 3:frontCQ=initializeQ(a);
              rearCQ=frontCQ;
       do{
           printf("The operations available are=>\n");
           printf("1.Insert_CQ 2.Delete_CQ 3.CQ_empty 4.CQ_Full 5.Display_CQ 0.Exit:");
           scanf("%d",&choice02);
           switch(choice02){
             case 1:printf("Enter the element? ");
                    scanf("%d",&key);
                    flag=insertCQ(a,&frontCQ,&rearCQ,key);
                    if(flag==-1){
                         printf("The list is full\n");
                    }
                    else{
                         dispCQ(&frontCQ,&rearCQ,a);
                    }
                    break;
             case 2:flag=deleteCQ(a,&frontCQ,&rearCQ);
                    if(flag==-1)
                         printf("The Circular Queue is empty\n");
                    else
                         dispCQ(&frontCQ,&rearCQ,a);
                    break;
             case 3:flag=isEmptyCQ(&frontCQ);
                    if(flag==1){
                         printf("The Circular Queue is empty\n");
                    }
                    else
                         printf("The Circular queue contains elements\n");
                    break;
             case 4:flag=isFullCQ(&frontCQ,&rearCQ);
                    if(flag==1){
                         printf("The list is full\n");
                    }
                    else{
                         printf("Space Available\n");
                    }
                    break;
             case 5:flag=dispCQ(&frontCQ,&rearCQ,a);
                    if(flag==1){
                         printf("The list is empty \n");
                    }
                    dispCQ(&frontCQ,&rearCQ,a);
                    break;
             case 0:break;
         }
           printf("Any other operation,press 1:");
           scanf("%d",&choice3);
     }while(choice3==1);
      break;
     }
     printf("Want any other list press 1(Previous  progress will be lost):");
     scanf("%d",&choice1);
   }while(choice1==1);
   return 0;
}
//functions
int initializeStk(int stk[]){
     return -1;
}
int isEmptyStk(int stk[],int top){
     if(top==-1){
         return -1;
     }
     return 0;
}
int isFullStk(int stk[],int top){
     if(top>=MX_SIZE){
         return 1;
     }
     return 0;
}
int pushStk(int stk[],int *ele,int *top){
     if(*top>=MX_SIZE-1){
       *ele=MN_VAL;
        return 1;
     }
     *top=*top+1;
     stk[*top]=*ele;
     return 0;
}
int popStk(int stk[],int * ele,int *top){
     int key;
     if(*top==-1){
         *ele=MN_VAL;
          return 1;
     }
     key=stk[*top];
     stk[*top]=*ele;
     *top=*top-1;
return key;
}
int topvalStk(int stk[],int *top){
     if(*top==-1){
         return 1;
     }
     return stk[*top];
}
int dispStk(int a[],int *top){
     int i;
     if(*top==-1){
         return 1;
    }
    printf("The Stack is => ");
    for(i=0;i<=*top;i++){
         printf("%d|",a[i]);
     }
}
int initializeQ(int q[]){
     return -1;
}
int insertQ(int q[],int *front,int *rear,int ele){
     if(*rear==MX_SIZE-1){
         return 1;
}
     if(*front==-1){
     *front=0;
     }
     *rear=*rear+1;
     q[*rear]=ele;
return 0;
}
int deleteQ(int q[],int *front,int *rear){
     if(*front==-1||*front>*rear){
         return 1;
     }
     printf("The deleted element is %d",q[*front]);
     *front=*front+1;
}
int dispQ(int q[],int *front,int *rear){
    int i;
    if(*front==-1){
         return 1;
    }
    printf("The Queue is => ");
    for(i=*front;i<=*rear;i++){
      printf("%d|",q[i]);
    }
}
int isEmptyQ(int *front){
    if(*front==-1){
         return 1;
    }
}
int isFullQ(int *rear){
    if(*rear==MX_SIZE-1){
         return 1;
    }
}
int insertCQ(int CQ[],int *frontCQ,int *rearCQ,int key){
  if((*frontCQ==0&&*rearCQ==MX_SIZE-1)||(*rearCQ+1==*frontCQ)){
         return -1;
  }
  if(*rearCQ==-1){
         *rearCQ=0;
         *frontCQ=0;
  }
  else if(*rearCQ==MX_SIZE-1){
         *rearCQ=0;
  }
  else{
         *rearCQ=*rearCQ+1;
  }
  CQ[*rearCQ]=key;
}
int deleteCQ(int CQ[],int *frontCQ,int *rearCQ){
  int key;
  if(*frontCQ==-1){
         return -1;
  }
  key=CQ[*frontCQ];
  CQ[*frontCQ]=MN_VAL;
  if(*rearCQ==*frontCQ){
         *rearCQ=-1;
         *frontCQ=-1;
  }
  else if(*frontCQ==MX_SIZE-1){
         *frontCQ=0;
  }
  else{
         *frontCQ=*frontCQ+1;
  }
return key;
}
int isFullCQ(int *frontCQ,int *rearCQ){
  if((*frontCQ==0&&*rearCQ==MX_SIZE-1)||(*rearCQ+1==*frontCQ)) {
         return 1;
  }
return 0;
}
```

```
int isEmptyCQ(int *frontCQ){
  if(*frontCQ==-1){
        return 1;
  }
return 0;
}
int dispCQ(int *frontCQ,int *rearCQ,int q[]){
  int i;
  printf("The Circular Queue is=>");
  if(*frontCQ<=*rearCQ){
        for(i=0;i<=*rearCQ;i++){
            if(q[i]==MN_VAL){
                printf("X|");
            }
            else
                printf("%d|", q[i]);
            }
        printf("\n");
        }
  else{
        for(i=0;i<=MX_SIZE-1;i++){
            if(q[i]==MN_VAL){
                printf("X|");
            }
            else
                printf("%d|", q[i]);
        }
        printf("\nThe CQ is empty");
      }
}
/*EXECUTION TRAIL:
STACK
Enter the type of list
1.Stack 2.Queue 3.Circular Queue:1
The operations available are=>
1.Push 2.Pop 3.isFull 4.IsEmpty 5.Top value 6.Display 0.Exit:1
Enter the element?23
The Stack is => 23|Any other operation,press 1:1
The operations available are=>
1.Push 2.Pop 3.isFull 4.IsEmpty 5.Top value 6.Display 0.Exit:1
Enter the element?56
The Stack is => 23|56|Any other operation,press 1:1
The operations available are=>
1.Push 2.Pop 3.isFull 4.IsEmpty 5.Top value 6.Display 0.Exit:1
Enter the element?8
The Stack is => 23|56|8|Any other operation,press 1:1
The operations available are=>
1.Push 2.Pop 3.isFull 4.IsEmpty 5.Top value 6.Display 0.Exit:2
The popped element is 8
The Stack is => 23|56|Any other operation,press 1:1
The operations available are=>
1.Push 2.Pop 3.isFull 4.IsEmpty 5.Top value 6.Display 0.Exit:3
The list is not full
Any other operation,press 1:1
The operations available are=>
1.Push 2.Pop 3.isFull 4.IsEmpty 5.Top value 6.Display 0.Exit:5
The element at top is 56 Any other operation,press 1:0
Want any other list press 1(Previous  progress will be lost):1
QUEUE
Enter the type of list
1.Stack 2.Queue 3.Circular Queue:2
The operations available are=>
1.Insert 2.Delete 3.isEmpty 4.isFull 5.Display 0.Exit:1
Enter the element to be inserted=>45
The Queue is => 45|Any other operation,press 1: 1
The operations available are=>
```

```
1.Insert 2.Delete 3.isEmpty 4.isFull 5.Display 0.Exit:1
Enter the element to be inserted=>65
The Queue is => 45|65|Any other operation,press 1: 1
The operations available are=>
1.Insert 2.Delete 3.isEmpty 4.isFull 5.Display 0.Exit:2
The deleted element is 45 The Queue is => 65|Any other operation,press 1: 1
The operations available are=>
1.Insert 2.Delete 3.isEmpty 4.isFull 5.Display 0.Exit:2
The deleted element is 65 The Queue is => Any other operation,press 1: 1
The operations available are=>
1.Insert 2.Delete 3.isEmpty 4.isFull 5.Display 0.Exit:4
The Queue is not full
Any other operation,press 1: 1
The operations available are=>
1.Insert 2.Delete 3.isEmpty 4.isFull 5.Display 0.Exit:3
The Queue is empty.
Any other operation,press 1: 0
Want any other list press 1(Previous  progress will be lost):1
CIRCULAR QUEUE
Enter the type of list
1.Stack 2.Queue 3.Circular Queue:3
The operations available are=>
1.Insert_CQ 2.Delete_CQ 3.CQ_empty 4.CQ_Full 5.Display_CQ 0.Exit:1
Enter the element? 12
The Circular Queue is=>12|
Any other operation,press 1:1
The operations available are=>
1.Insert_CQ 2.Delete_CQ 3.CQ_empty 4.CQ_Full 5.Display_CQ 0.Exit:1
Enter the element? 23
The Circular Queue is=>12|23|
Any other operation,press 1:1
The operations available are=>
1.Insert_CQ 2.Delete_CQ 3.CQ_empty 4.CQ_Full 5.Display_CQ 0.Exit:1
Enter the element? 34
The Circular Queue is=>12|23|34|
Any other operation,press 1:1
The operations available are=>
1.Insert_CQ 2.Delete_CQ 3.CQ_empty 4.CQ_Full 5.Display_CQ 0.Exit:1
Enter the element? 44
The Circular Queue is=>12|23|34|44|
Any other operation,press 1:1
The operations available are=>
1.Insert_CQ 2.Delete_CQ 3.CQ_empty 4.CQ_Full 5.Display_CQ 0.Exit:1
Enter the element? 45
The Circular Queue is=>12|23|34|44|45|
Any other operation,press 1:1
The operations available are=>
1.Insert_CQ 2.Delete_CQ 3.CQ_empty 4.CQ_Full 5.Display_CQ 0.Exit:1
Enter the element? 29
The list is full
Any other operation,press 1:1
The operations available are=>
1.Insert_CQ 2.Delete_CQ 3.CQ_empty 4.CQ_Full 5.Display_CQ 0.Exit:3
The Circular queue contains elements
Any other operation,press 1:1
The operations available are=>
1.Insert_CQ 2.Delete_CQ 3.CQ_empty 4.CQ_Full 5.Display_CQ 0.Exit:2
The Circular Queue is=>X|23|34|44|45|
Any other operation,press 1:1
The operations available are=>
1.Insert_CQ 2.Delete_CQ 3.CQ_empty 4.CQ_Full 5.Display_CQ 0.Exit:2
The Circular Queue is=>X|X|34|44|45|
Any other operation,press 1:2
Want any other list press 1(Previous  progress will be lost):3
*/
```