/* A program to reserve the stack elements and then find the index of minimum valued number using array.

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0

void Push (int);
int Pop ();
void PrintStack();
void reserve ();
void MiniElement ();

    // stack data structure
int top = -1;
int stack[20] ;
int SIZE = 0 ;

int main()
{
    int temp;
    /* Inserting elements in stack */
Printf ("Enter the number of elements : ");
scanf ("%d", &SIZE);
Printf ("Enter the elements of stack : \n");
for (int i=0; i<SIZE; i++){
      scanf ("%d", &temp);
      Push (temp);
}
 Printf ("original stack\n");
 PrintStack();
 reserve ();
 Printf ("\nReserved stack\n");
 PrintStack ();
 MinElement ();

  return 0;
}
 /* Adds an element to stack and then increment top index.
  */
   void Push (int num)
  {
```

```
    if (top >= SIZE -1)
        Printf ("stack is Full... \n");
  else
    {
        top = top + 1;
        stack[top] = num;
    }
}

/* Removes top elements from stack and decrement top
   index */
int Pop()
{
    if (top == -1)
        Printf("stack is Empty ... \n");
    else
    {
        top = top - 1 ;
        return stack [top + 1];
    }
}

// Prints elements of stack void PrintStack()
{
  if (top == -1) // checks if stacks is empty
 {
        print ("stacks is empty \n");
 }
  else
  {
      for (int i = top; i >= 0; i--)
    {
        printf("%d", stack[i]);
    }
      Printf ("\n");
  }
}

  void insertAtBottom (int item)
```

```
{
  if (top == -1)
   {
     push(item);
   }
    else
   {
```

/* store the top most element of stack in top variable and recursively called insertAtBottom for rest of the stack */

```
        int top = pop();
        insertAtBottom(item);
```

/* once the item is inserted at the bottom, push the top element back to stack */

```
     push(top);
   }
}
void reverse()
{
   if (top != -1)
  {
```

/* keep on popping top element of stack in every recursive call till stack is empty */

```
    int top = POP();
    reverse();
```

/* Now, insert the top element at the bottom of stack */

```
          insertAtBottom(top);
     }
   }
```

// for finding the index of min element void MinElement()

```
  {
    int index = 0;
    for (int i = top; i >= 0; i--)
   {
      if (stack[i] < stack[index])
       {
```

```
            index = i ;
        }
    }
    Printf("\nMinimum element is present at index %d", index);
}
```

```
            index = i ;
        }
    }
    Printf("\nMinimum element is present at index %d", index);
}
```

2.
```
#include <stdio.h>
#define MAX_SIZE 100
void enqueue ();
void delete ();
void display ();

int arr_queue [MAX_SIZE];
int rear = 0, front = 0;

int main()
{
    int n;
// queue operations for equeuing elements,
    printf ("Enter the no. of queue elements: ");
    scanf ("%d", &n);
    for (int i=0; i<n; i++) {
        enqueue ();
    }
// deleting minimum value item from queue and displaying
the element of queue.
    delete ();
    return 0;
}

void enqueue ()
{
    int item;
    if (rear == MAX_SIZE)
        Print ("\nQueue Reached Max !! ");
```

```
else
{
    Printf("\nEnter the Value to be Inserted : ");
        Scanf("%d", &item);
        Printf("Inserted");
        arr-queue[rear++] = item;
    }
}
Void delete()
{
    if (front == rear)
        Printf("\nqueue is Empty !");
else
    {
        int min-element, index = 0;
        display();
        for (int i = front; i<rear; i++)
    {
        if (arr-queue[i] < arr-queue[index])
        {
            index = i;
        }
    }
    min-element = arr-queue[index];
    Printf("Deleting Minimum element\n");
    for (int i=index; i<rear; i++) {
        arr-queue[i] = arr-queue[i+1];
    }
    rear = rear-1;
    display();
  }
}
Void display()
{
    Printf("\nqueue elements :");
    for (int i=front; i<rear; i++)
        Printf("%d", arr-queue[i]);
  }
```

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

Struct node
{
    int data;
    struct node *next;
};
struct node *front;
struct node *rear;

void enqueue();
void display();

void main()
{
    int choice;
    While (1)
    {
        printf ("\nqueue operation");
        printf ("\n1.insert an element\n2. Display the
queue\no.Exit\n");
        printf ("\nEnter your choice: ");
        scanf ("%d", & choice);
        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            display();
            break;
        case 0:
            exit (0);
            break;
```

```
        default:
            printf("\nEnter Valid Choice!!\n");
        }
    }
}
Void enqueue()
{
    Struct node * ptr;
    int item;
    ptr = (struct node *) malloc (sizeof(struct node));
    if (ptr == NULL)
    {
        printf("\nOVERFLOW\n");
        return;
    }
    else
    {
        printf("\nEnter value: ");
        scanf("%d", &item);
        ptr->data = item;
        if (front == NULL) // inserting node in empty queue
        {
            front = ptr;
            rear = ptr;
            front->next = NULL;
            rear->next = NULL;
        }
        else // inserting node next to the previous node
        {
            rear->next = ptr;
            rear = ptr;
            rear->next = NULL;
        }
```

```
        printf ("Inserted!\n");
    }
}
void display()
{
    struct node * ptr;
    ptr = front;
    if (front == NULL)
        printf ("\nEmpty queue\n");
    else
    {
        printf ("\nvalues inside Queue are: ");
        While (ptr != NULL)
        {
            printf ("%d", ptr -> data);
            ptr = ptr -> next;
        }
        printf ("\n");
    }
}
```