

/* 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 MinElement ();
```

```
// 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;
```

```
}
```

```
/* Add 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 element 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 stack is empty
    {
        print ("stack 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 call 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 elements 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("Minimum 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 operation for enqueueing 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 <stdio.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\n0.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");
```

```
    }
```

```
}
```



```

Management
Projects Resources
Workspace

Start here Set5Q1.c
25 printf("Enter the number of elements: ");
26 scanf("%d", &SIZE);
27 printf("Enter the elements of stack:\n");
28 for(int i=0; i<SIZE; i++){
29     scanf("%d", &temp);
30     push(temp);
31 }
32 printf("Original Stack\n");
33 printStack();
34 reverse();
35 printf("\nReversed Stack\n");
36 printStack();
37 MinElement();
38 return 0;
39 }
40
41 /*
42  Adds an element to stack and then increment top index
43  */
44 void push(int num)
45 {
46     if (top >= SIZE-1)
47         printf("Stack is Full...\n");
48     else
49     {
50         top = top + 1;
51         stack[top] = num;
52     }
53 }
54
55 /*
56  Removes top element from stack and decrement top index
57  */
58 int pop()
59 {
60     if (top == -1)
61         printf("Stack is Empty...\n");
62     else
63     {
64         top = top - 1;
65         return stack[top + 1];
66     }
67 }
68
69
88 if (top == -1)
89 {
90     push(item);
91 }
92 else
93 {
94     /* Store the top most element of stack in top variable and
95     recursively call insertAtBottom for rest of the stack */
96     int top = pop();
97     insertAtBottom(item);
98
99     /* Once the item is inserted at the bottom, push the
100    top element back to stack */
101    push(top);
102 }
103 }
104
105 void reverse()
106 {
107     if (top != -1)
108     {
109         /* keep on popping top element of stack in
110         every recursive call till stack is empty */
111         int top = pop();
112         reverse();
113
114         /* Now, insert the top element at the bottom of stack */
115         insertAtBottom(top);
116     }
117 }
118
119 // for finding the index of min element
120 void MinElement()
121 {
122     int index = 0;
123     for (int i = top; i >= 0; i--)
124     {
125         if (stack[i] < stack[index])
126         {
127             index = i;
128         }
129     }
130     printf("\nMinimum element is present at index %d", index);
131 }

```

/home/crypticani/Downloads/Telegram Desktop/Set5Q1

– □ ×

Enter the number of elements: 5

Enter the elements of stack:

23 54 5 43 76

Original Stack

76 43 5 54 23

Reversed Stack

23 54 5 43 76

Minimum element is present at index 2

Process returned 0 (0x0) execution time : 53.119 s

Press ENTER to continue.

■

Set5Q2.c - Code::Blocks 20.03

File Edit View Search Project Build Debug wxSmith

<global> main():int

Management Start here Set5Q2.c

Projects Resources Workspace

```
1 #include <stdio.h>
2
3 #define MAX_SIZE 100
4
5 void enqueue();
6 void delete();
7 void display();
8
9 int arr_queue[MAX_SIZE];
10 int rear = 0, front = 0;
11
12 int main()
13 {
14     int n;
15     // queue operations for enqueueing elements,
16     printf("Enter the no of queue elements: ");
17     scanf("%d", &n);
18     for(int i=0; i<n; i++){
19         enqueue();
20     }
21     // deleting minimum valued item from queue and displaying the elements of queue
22     delete();
23     return 0;
24 }
25
26 void enqueue()
27 {
28     int item;
29     if (rear == MAX_SIZE)
30         printf("\nQueue Reached Max!!");
31     else
32     {
33         printf("\nEnter The Value to be Inserted : ");
34         scanf("%d", &item);
35         printf("Inserted");
36         arr_queue[rear++] = item;
37     }
38 }
39
40 void delete()
41 {
42     if (front == rear)
43         printf("\nQueue is Empty!");
44     else
45     {
46         int min_element, index = 0;
47         display();
48         for (int i = front; i < rear; i++)
49         {
50             if (arr_queue[i] < arr_queue[index])
51             {
52                 index = i;
53             }
54         }
55         min_element = arr_queue[index];
56         printf("Deleting Minimum element\n");
57         for(int i=index; i<rear; i++){
58             arr_queue[i] = arr_queue[i+1];
59         }
60         rear = rear - 1;
61         display();
62     }
63 }
64
65 void display()
66 {
67     printf("\nQueue elements: ");
68     for (int i = front; i < rear; i++)
69         printf("%d ", arr_queue[i]);
70 }
```

Enter the no of queue elements: 5
Enter The Value to be Inserted : 12
Inserted
Enter The Value to be Inserted : 43
Inserted
Enter The Value to be Inserted : 34
Inserted
Enter The Value to be Inserted : 54
Inserted
Enter The Value to be Inserted : 32
Inserted
Queue elements: 12 43 34 54 32 Deleting Minimum element
Queue elements: 43 34 54 32
Process returned 0 (0x0) execution time : 160.845 s
Press ENTER to continue.

```
if (rear == MAX_SIZE)
    printf("\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]);
}
```

Set5Q3.c - Code::Blocks 20.03

File Edit View Search Project Build Debug wxSmith Tools Tools+ P

<global>

Management

Projects Resources

Workspace

Start here Set5Q3.c

```
4 struct node
5 {
6     int data;
7     struct node *next;
8 };
9 struct node *front;
10 struct node *rear;
11
12 void enqueue();
13 void display();
14
15 void main()
16 {
17     int choice;
18     while(1)
19     {
20         printf("\nQueue operation");
21         printf("\n1.insert an element\n2.Display the queue\n0.Exit\n");
22         printf("\nEnter your choice: ");
23         scanf("%d", &choice);
24         switch (choice)
25         {
26             case 1:
27                 enqueue();
28                 break;
29             case 2:
30                 display();
31                 break;
32             case 0:
33                 exit(0);
34                 break;
35             default:
36                 printf("\nEnter valid choice!!\n");
37         }
38     }
39 }
40
41 void enqueue()
42 {
43     struct node *ptr;
44     int item;
45
46     ptr = (struct node *)malloc(sizeof(struct node));
```

Queue operation
1.insert an element
2.Display the queue
0.Exit

Enter your choice: 1

Enter value: 32
Inserted!

Queue operation
1.insert an element
2.Display the queue
0.Exit

Enter your choice: 1

Enter value: 34
Inserted!

Queue operation
1.insert an element
2.Display the queue
0.Exit

Enter your choice: 2

Values inside Queue are: 32 34

```
47 if (ptr == NULL)
48 {
49     printf("\nOVERFLOW\n");
50     return;
51 }
52 else
53 {
54     printf("\nEnter value: ");
55     scanf("%d", &item);
56     ptr->data = item;
57     if (front == NULL) //inserting node in empty queue
58     {
59         front = ptr;
60         rear = ptr;
61         front->next = NULL;
62         rear->next = NULL;
63     }
64     else //inserting node next to the previous node
65     {
66         rear->next = ptr;
67         rear = ptr;
68         rear->next = NULL;
69     }
70     printf("Inserted!\n");
71 }
72 }
73
74 void display()
75 {
76     struct node *ptr;
77     ptr = front;
78     if (front == NULL)
79         printf("\nEmpty queue\n");
80     else
81     {
82         printf("\nValues inside Queue are: ");
83         while (ptr != NULL)
84         {
85             printf("%d ", ptr->data);
86             ptr = ptr->next;
87         }
88         printf("\n");
89     }
90 }
```

Line 1, Col 1, Pos 0

Insert

Read/Write default

UTF-8

C/C++

Unix (LF)

/home/crypticani/Downloads/Telegram Desktop/Set5Q3.c