C Set5Q1.c U × C Set5Q2.c U C Set5Q3.c U ···

C > CA1 > C Set5Q1.c > main()

```c

#include <stdio.h>

#define TRUE 1
#define FALSE 0

void push(int);
int pop();
void printStack();
void reverse();
void MinElement();

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

int main()
{
    int temp;
    /* Inserting elements in stack */
    printf("Enter the number of elements
    (not more than 20): ");
    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();
    reverse();
    printf("\nReversed Stack\n");
    printStack();
    MinElement();
    return 0;
}
```

C Set5Q1.c U ×

C > CA1 > C Set5Q1.c > main()

```c
/*
 Checks if Stack is Full or not
*/
int isFull()
{
    if (top >= SIZE - 1)
        return TRUE;
    else
        return FALSE;
}

/*
 Checks if Stack is Empty or not
*/
int isEmpty()
{
    if (top == -1)
        return TRUE;
    else
        return FALSE;
}

/*
 Adds an element to stack and then increment
 top index
*/
void push(int num)
{
    if (isFull())
        printf("Stack is Full...\n");
    else
    {
        top = top + 1;
        arr_stack[top] = num;
    }
}

```

Set5Q1.c U | Set5Q2.c U | Set5Q3.c U

C > CA1 > Set5Q1.c > main()

```c
int pop()
{
    if (isEmpty())
        printf("Stack is Empty...\n");
    else
    {
        top = top - 1;
        return arr_stack[top + 1];
    }
}

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

void insertAtBottom(int item)
{
    if (isEmpty())
    {
        push(item);
    }
    else
    {
        /* Store the top most element of
```

Set5Q1.c U

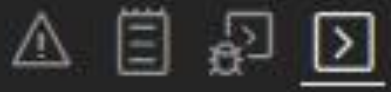
C > CA1 > Set5Q1.c > reverse()

```c
        bottom, push the
        top element back to stack */
        push(top);
    }
}

void reverse()
{
    if (!isEmpty())
    {
        /* 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 (arr_stack[i] < arr_stack[index])
        {
            index = i;
        }
    }
    printf("\nMinimum element is present at
    index %d and is %d(th) element from the
    top.\n", index, top-index+1);
}
```




```
Enter the number of elements (not more than 20): 5
Enter the elements of stack:
32 45 67 12 56
Original Stack
56 12 67 45 32

Reversed Stack
32 45 67 12 56

Minimum element is present at index 1 and is 4(th)
element from the top.
[crypticani@fedora DSA_Programs]$
```

Set5Q2.c - DSA_Programs - Visual Studio Code

File Edit Selection View Go Run Terminal Help

Set5Q1.c U | Set5Q2.c U | Set5Q3.c U

C > CA1 > Set5Q2.c > main()

```c
#include <stdio.h>

#define MAX_SIZE 100

void insert();
void delete_min();
void display();

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

int main()
{
    int n;
// queue operations for enqueuing elements,
    printf("Enter the no of queue elements: ");
    scanf("%d", &n);
    for(int i=0; i<n; i++){
        insert();
    }
    // deleting minimum valued item from queue and
    displaying the elements of queue
    delete_min();
    return 0;
}

void insert()
{
    int item;
    if (rear == MAX_SIZE)
        printf("\n## Queue Reached Max!!");
    else
    {
        printf("\nEnter The Value to be Inserted :
        ");
        scanf("%d", &item);
        printf("# Position : %d , Inserted Value
        : %d ", rear + 1, item);
        arr_queue[rear++] = item;
    }
}

void delete_min()
{
    if (front == rear)
        printf("\n## Queue 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("\nMinimum element is %d.\n",
        min_element);
        printf("Deleting it\n");

        for(int i=index; i<rear; i++){
            arr_queue[i] = arr_queue[i+1];
        }
        rear = rear - 1;
        display();
    }
}

void display()
{
    printf("\n## Queue Size : %d \n", rear);
    for (int i = front; i < rear; i++)
        printf("%d ", arr_queue[i]);
}
```

```
Enter the no of queue elements: 5

Enter The Value to be Inserted : 32
# Position : 1 , Inserted Value  : 32
Enter The Value to be Inserted : 45
# Position : 2 , Inserted Value  : 45
Enter The Value to be Inserted : 67
# Position : 3 , Inserted Value  : 67
Enter The Value to be Inserted : 12
# Position : 4 , Inserted Value  : 12
Enter The Value to be Inserted : 56
# Position : 5 , Inserted Value  : 56
## Queue Size : 5
32 45 67 12 56
Minimum element is 12.
Deleting it

## Queue Size : 4
32 45 67 56 [crypticani@fedora DSA_Programs]$
```

main* | 0 0 | gcc - Build and debug active file (DSA_Programs) | Ln 53, Col 14 | Spaces: 4 | UTF-8 | LF | C | Linux

Set5Q1.c U | Set5Q2.c U | Set5Q3.c U

C > CA1 > Set5Q3.c > display()

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

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

void insert();
void display();

void main()
{
    int choice;
    while(1)
    {
        printf("\nQueue operation");
        printf("\n1.insert an element\n2.Display the
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            insert();
            break;
        case 2:
            display();
            break;
        case 0:
            exit(0);
            break;
        default:
            printf("\nEnter valid choice!!\n");
        }
    }
```

Set5Q3.c U

C > CA1 > Set5Q3.c > display()

```c
        }
    }
}

void insert()
{
    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
        {
            front = ptr;
            rear = ptr;
            front->next = NULL;
            rear->next = NULL;
        }
        else //inserting node next to the previous n
        {
            rear->next = ptr;
            rear = ptr;
            rear->next = NULL;
        }
        printf("Inserted!\n");
    }
}

void display()
```

Set5Q3.c U

C > CA1 > Set5Q3.c > display()

```c
        {
            front = ptr;
            rear = ptr;
            front->next = NULL;
            rear->next = NULL;
        }
        else //inserting node next to the previous n
        {
            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");
    }
}
```

cppdbg: Set5Q3

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

Enter your choice: 1

Enter value: 43
Inserted!

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

Enter your choice: 1

Enter value: 46
Inserted!

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

Enter your choice: 2

Values inside Queue are: 43 46

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

Enter your choice: 0
[crypticani@fedora DSA_Programs]$
```