Government College of Engineering, Karad Programming for Problem Solving Lab

Nanekar Saurabh Rajesh

20141212

I1

**Experiment No .1**

**Title**: Implement stack and queue as an ADT using array.

**Outcome:** Students can demonstrate stack and queue operations using array.

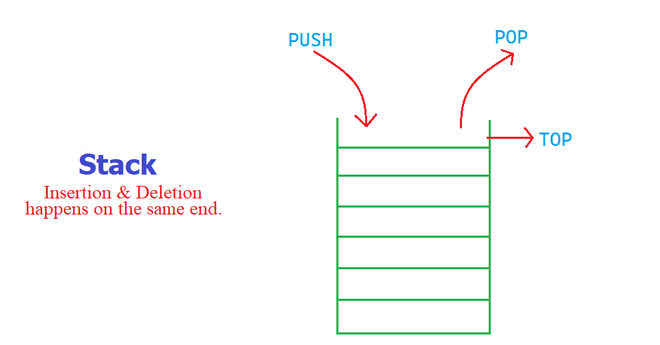
**Theory:**

**Stack ADT:**

Stack is a linear data structure that follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).

Mainly the following three basic operations are performed in the stack:

* **Push:**Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
* **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
* **Peek or Top:** Returns the top element of the stack.
* **isEmpty:**Returns true if the stack is empty, else false.



**How to understand a stack practically?**   
There are many real-life examples of a stack. Consider the simple example of plates stacked over one another in a canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time. So, it can be simply seen to follow the LIFO/FILO order.

# Stack Operations using Array

A stack can be implemented using array as follows...

Before implementing actual operations, first follow the below steps to create an empty stack.

* **Step 1 -**Include all the **header files** which are used in the program and define a constant **'SIZE'** with specific value.
* **Step 2 -**Declare all the **functions** used in stack implementation.
* **Step 3 -**Create a one dimensional array with fixed size (**int stack[SIZE]**)
* **Step 4 -**Define a integer variable **'top'** and initialize with **'-1'**. (**int top = -1**)
* **Step 5 -**In main method, display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack.

## push(value) - Inserting value into the stack

In a stack, push() is a function used to insert an element into the stack. In a stack, the new element is always inserted at **top** position. Push function takes one integer value as parameter and inserts that value into the stack. We can use the following steps to push an element on to the stack...

* **Step 1 -**Check whether **stack** is **FULL**. (**top == SIZE-1**)
* **Step 2 -**If it is **FULL**, then display **"Stack is FULL!!! Insertion is not possible!!!"** and terminate the function.
* **Step 3 -**If it is **NOT FULL**, then increment **top** value by one (**top++**) and set stack[top] to value (**stack[top] = value**).

### pop() - Delete a value from the Stack

In a stack, pop() is a function used to delete an element from the stack. In a stack, the element is always deleted from **top** position. Pop function does not take any value as parameter. We can use the following steps to pop an element from the stack...

* **Step 1 -**Check whether **stack** is **EMPTY**. (**top == -1**)
* **Step 2 -**If it is **EMPTY**, then display **"Stack is EMPTY!!! Deletion is not possible!!!"** and terminate the function.
* **Step 3 -**If it is **NOT EMPTY**, then delete **stack[top]** and decrement **top** value by one (**top--**).

### display() - Displays the elements of a Stack

We can use the following steps to display the elements of a stack...

* **Step 1 -**Check whether **stack** is **EMPTY**. (**top == -1**)
* **Step 2 -**If it is **EMPTY**, then display **"Stack is EMPTY!!!"** and terminate the function.
* **Step 3 -**If it is **NOT EMPTY**, then define a variable '**i**' and initialize with top. Display **stack[i]** value and decrement **i** value by one (**i--**).
* **Step 3 -**Repeat above step until **i** value becomes '0'.

**Time Complexities of operations on stack:**

push(), pop(), isEmpty() and peek() all take O(1) time. We do not run any loop in any of these operations.

**Applications of stack:**

* [Balancing of symbols](https://www.geeksforgeeks.org/check-for-balanced-parentheses-in-an-expression/)
* [Infix to Postfix](https://www.geeksforgeeks.org/stack-set-2-infix-to-postfix/) /Prefix conversion
* Redo-undo features at many places like editors, photoshop.
* Forward and backward feature in web browsers
* Used in many algorithms like [Tower of Hanoi,](https://www.geeksforgeeks.org/recursive-functions/)[tree traversals](https://www.geeksforgeeks.org/618/), [stock span problem](https://www.geeksforgeeks.org/the-stock-span-problem/), [histogram problem](https://www.geeksforgeeks.org/largest-rectangular-area-in-a-histogram-set-1/).

**Implementation:**   
There are two ways to implement a stack:

* Using array
* Using linked list

# Queue Data structure Using Array

A queue data structure can be implemented using one dimensional array. The queue implemented using array stores only fixed number of data values. The implementation of queue data structure using array is very simple. Just define a one dimensional array of specific size and insert or delete the values into that array by using **FIFO (First In First Out) principle** with the help of variables **'front'** and '**rear**'. Initially both '**front**' and '**rear**' are set to -1. Whenever, we want to insert a new value into the queue, increment '**rear**' value by one and then insert at that position. Whenever we want to delete a value from the queue, then delete the element which is at 'front' position and increment 'front' value by one.

# Queue Operations using Array

Queue data structure using array can be implemented as follows...  
Before we implement actual operations, first follow the below steps to create an empty queue.

* **Step 1 -**Include all the **header files** which are used in the program and define a constant **'SIZE'** with specific value.
* **Step 2 -**Declare all the **user defined functions** which are used in queue implementation.
* **Step 3 -**Create a one dimensional array with above defined SIZE (**int queue[SIZE]**)
* **Step 4 -**Define two integer variables **'front'** and '**rear**' and initialize both with **'-1'**. (**int front = -1, rear = -1**)
* **Step 5 -**Then implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on queue.

# enQueue(value) - Inserting value into the queue

In a queue data structure, enQueue() is a function used to insert a new element into the queue. In a queue, the new element is always inserted at **rear** position. The enQueue() function takes one integer value as a parameter and inserts that value into the queue. We can use the following steps to insert an element into the queue...

* **Step 1 -**Check whether **queue** is **FULL**. (**rear == SIZE-1**)
* **Step 2 -**If it is **FULL**, then display **"Queue is FULL!!! Insertion is not possible!!!"** and terminate the function.
* **Step 3 -**If it is **NOT FULL**, then increment **rear** value by one (**rear++**) and set **queue[rear]** = **value**.

# deQueue() - Deleting a value from the Queue

In a queue data structure, deQueue() is a function used to delete an element from the queue. In a queue, the element is always deleted from **front** position. The deQueue() function does not take any value as parameter. We can use the following steps to delete an element from the queue...

* **Step 1 -**Check whether **queue** is **EMPTY**. (**front == rear**)
* **Step 2 -**If it is **EMPTY**, then display **"Queue is EMPTY!!! Deletion is not possible!!!"** and terminate the function.
* **Step 3 -**If it is **NOT EMPTY**, then increment the **front** value by one (**front ++**). Then display **queue[front]** as deleted element. Then check whether both **front** and **rear** are equal (**front** == **rear**), if it **TRUE**, then set both **front** and **rear** to '**-1**' (**front** = **rear** = **-1**).

# display() - Displays the elements of a Queue

We can use the following steps to display the elements of a queue...

* **Step 1 -**Check whether **queue** is **EMPTY**. (**front == rear**)
* **Step 2 -**If it is **EMPTY**, then display **"Queue is EMPTY!!!"** and terminate the function.
* **Step 3 -**If it is **NOT EMPTY**, then define an integer variable '**i**' and set '**i** = **front+1**'.
* **Step 4 -**Display '**queue[i]**' value and increment '**i**' value by one (**i++**). Repeat the same until '**i**' value reaches to **rear** (**i** <= **rear**)

**Analysis:**



**List of similar programs: Solve any one.**

1. Program to check if a queue of first n natural number can be sorted using a stack.
2. Write a program for reversing a Stack with the help of another empty Stack.
3. Write a program for reversing a Queue using another Queue.

**Title Program: Implement Stack and queue as an ADT using array.**

**Source code of Implemented Programs:**

Code for Stack:-

//Nanekar Saurabh Rajesh\_20141212\_I1

#include<stdio.h>

int stack[100],choice,n,top=-1,x,i;

void push(void);

int pop();

void show(void);

int main()

{

printf("\n Enter the size of STACK[MAX=100]:");

scanf("%d",&n);

do

{

printf("\n\t Choose one from below options:");

printf("\n\t 1.Push\n\t 2.Pop\n\t 3.Show\n\t 4.Exit");

printf("\n Enter your Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

show();

break;

}

case 4:

{

printf("\n\t EXIT POINT ");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");

}

}

}

while(choice!=4);

return 0;

}

void push()

{

if(top==n)

{

printf("\n\tSTACK is over flow");

}

else

{

printf(" Enter a value to be pushed:");

scanf("%d",&x);

top++;

stack[top]=x;

}

}

int pop()

{

if(top==-1)

{

printf("\n\t Stack is under flow");

}

else

{

printf("\n\t The popped elements is %d",stack[top]);

top--;

}

}

void show()

{

if(top>=0)

{

printf("\n The elements in STACK \n");

for(i=top; i>=0; i--)

printf("\n%d",stack[i]);

printf("\n Press Next Choice");

}

else

{

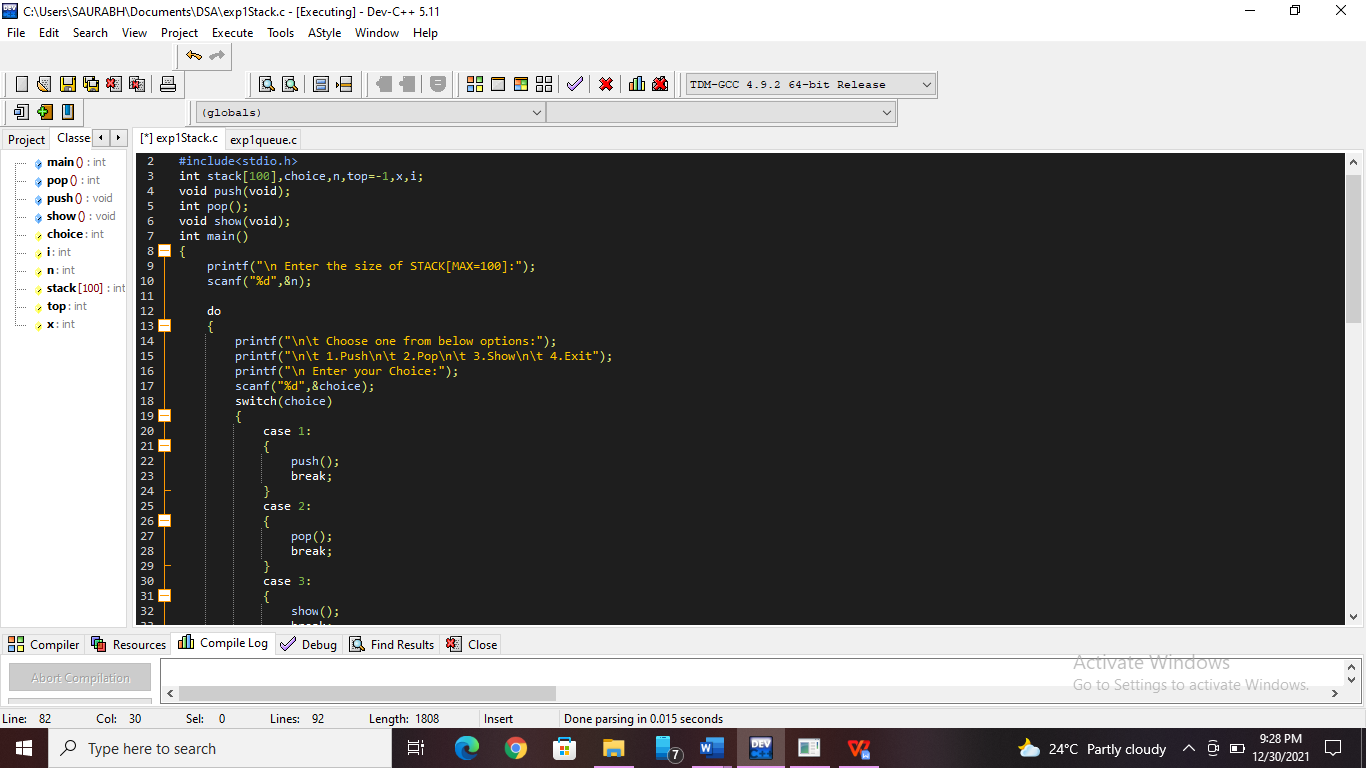
printf("\n The STACK is empty");

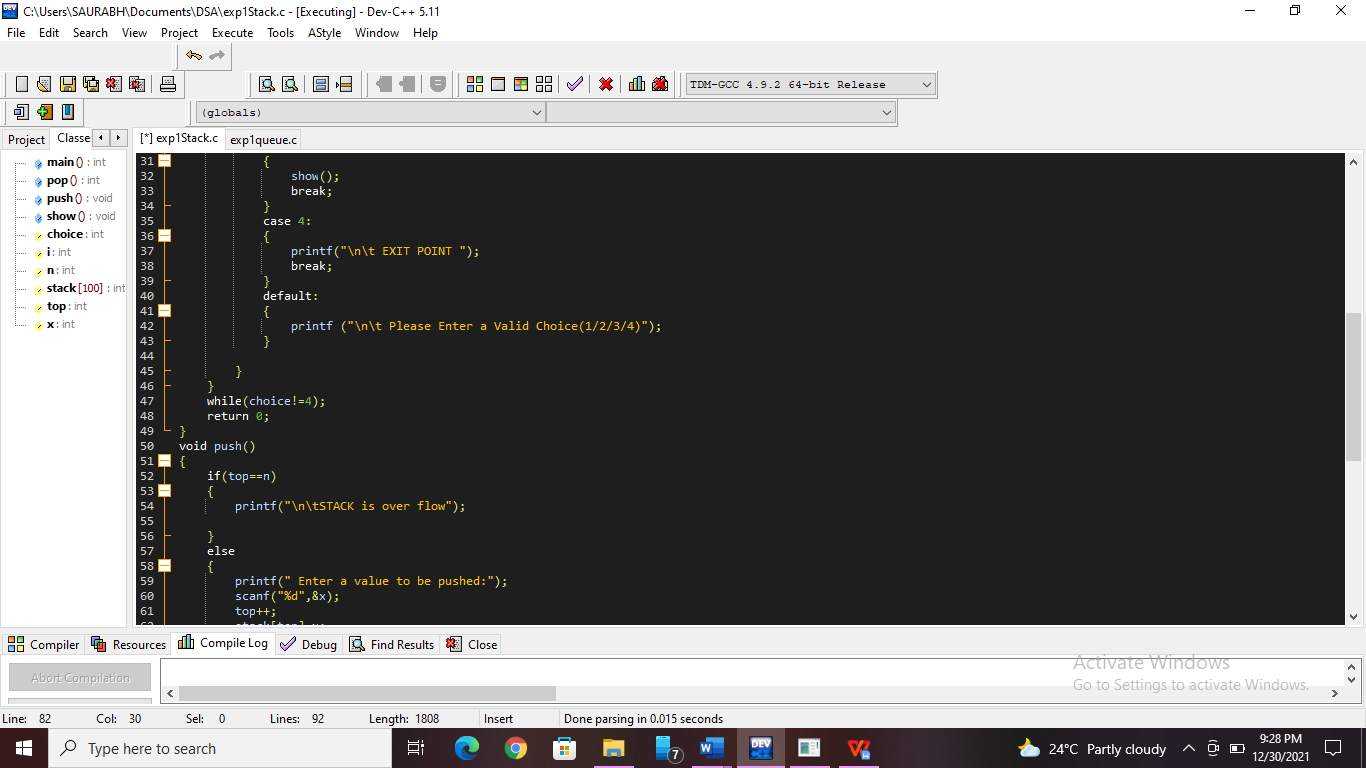
}

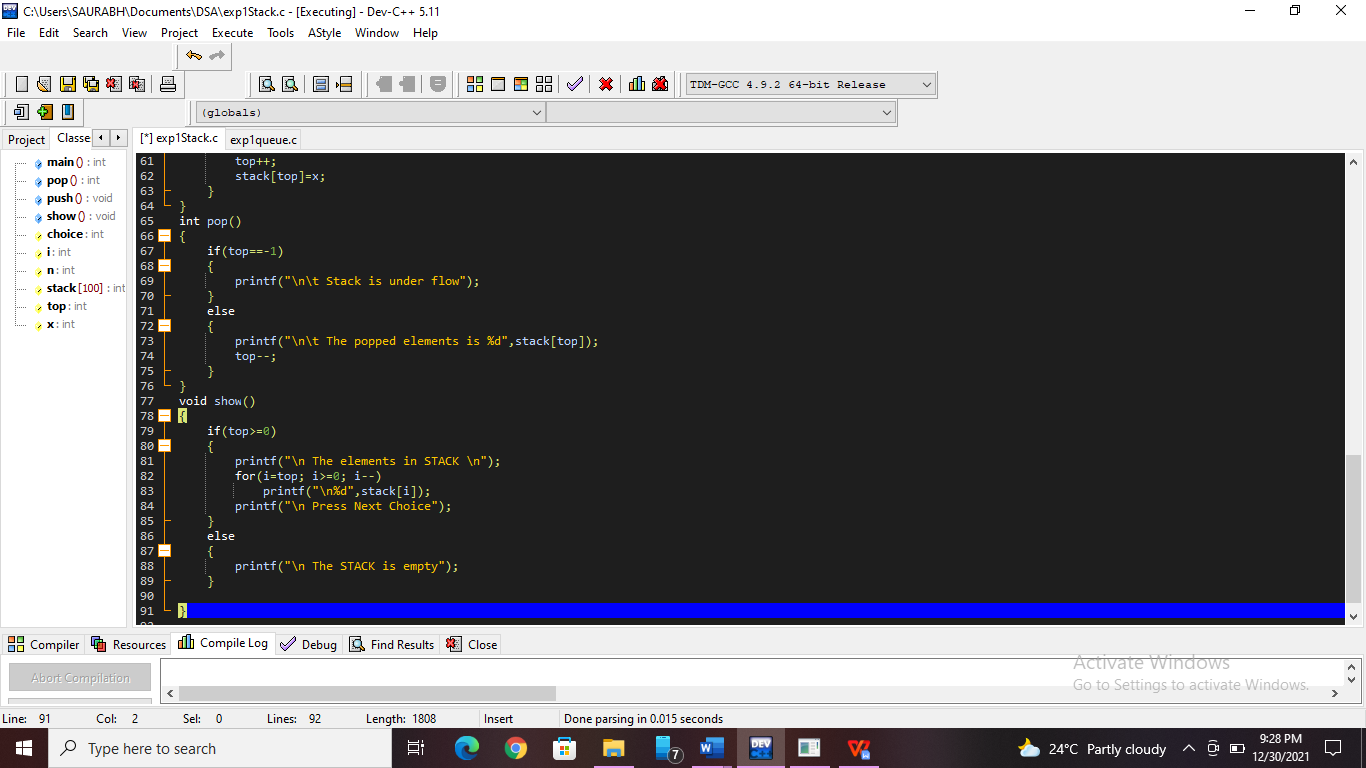
}

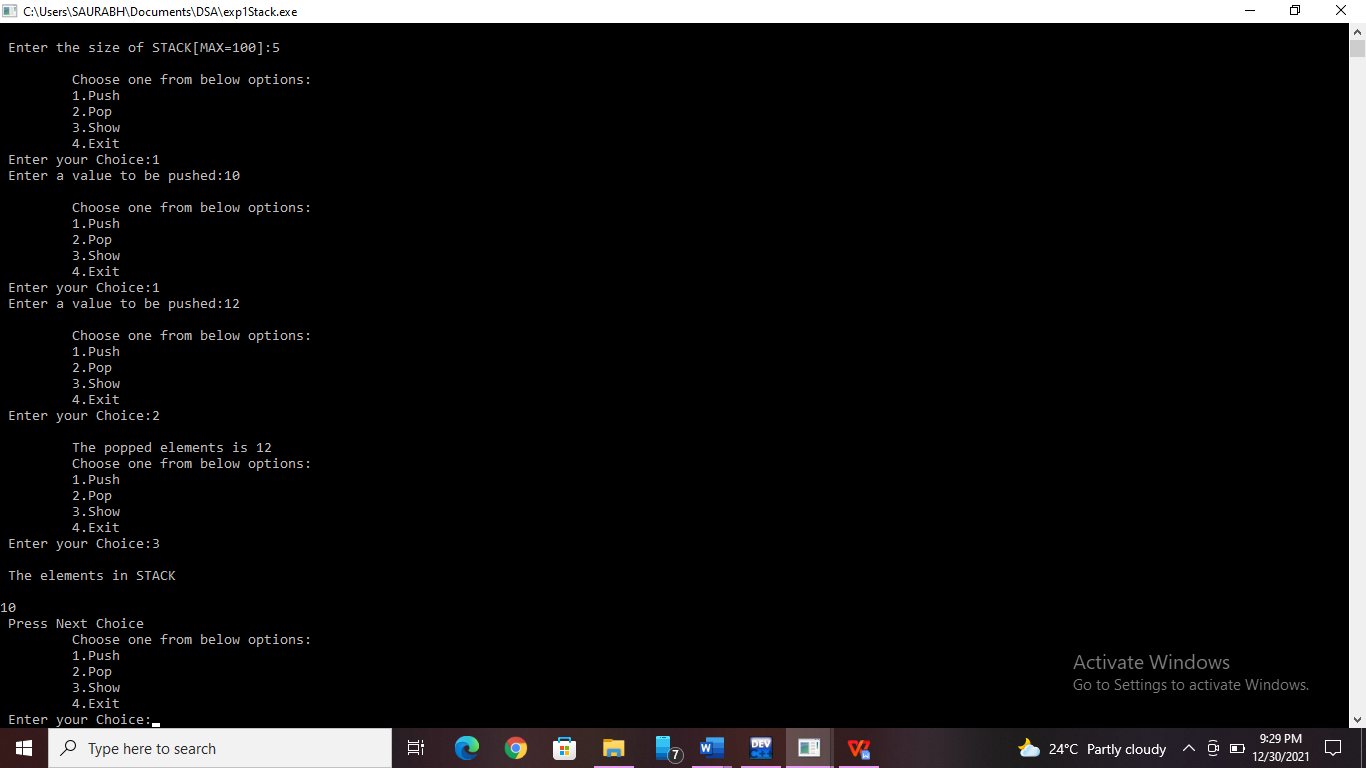
**Screenshots of Output:**

**Stack:-**









Code for Queue :-

// Nanekar Saurabh Rajesh\_20141212\_I1

#include<stdio.h>

#include<stdlib.h>

int queue[50],i,choice,n;

int front=-1,rear=-1;

void insert(void);

void delete(void);

void display(void);

int main()

{

printf("Enter the size of queue:-");

scanf("%d",&n);

do

{

printf("\nChoose choice from below:-");

printf("\n1.Insert\n2.Delete\n3.Display\n4.Exit");

printf("\nEnter your choice:-");

scanf("\n%d",&choice);

switch(choice)

{

case 1:

{

insert();

break;

}

case 2:

{

delete();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("\nEnd of process.");

break;

}

default:

{

printf("Choose valid choice from 1,2,3,4.");

}

}

}

while (choice!=4);

return 0;

}

void insert()

{

int item;

printf("\nEnter the element:-");

scanf("%d",&item);

if(rear == n-1)

{

printf("\nOverflow");

}

if(front==-1 && rear==-1)

{

front=0;

rear=0;

}

else

{

rear=rear+1;

}

queue[rear]=item;

printf("\n Value inserted.");

}

void delete()

{

int item;

if(front==-1 || front>rear)

{

printf("\nUnderflow");

}

else

{

item=queue[front];

if(front==rear)

{

front=-1;

rear=-1;

}

else

{

front=front+1;

}

printf("\nValue deleted");

}

}

void display()

{

if(front>=0 || rear>=0 )

{

printf("The Elements in Queue are:-");

for( i=front;i<=rear;i++)

printf("\n%d",queue[i]);

printf("\nEnter next choice-");

}

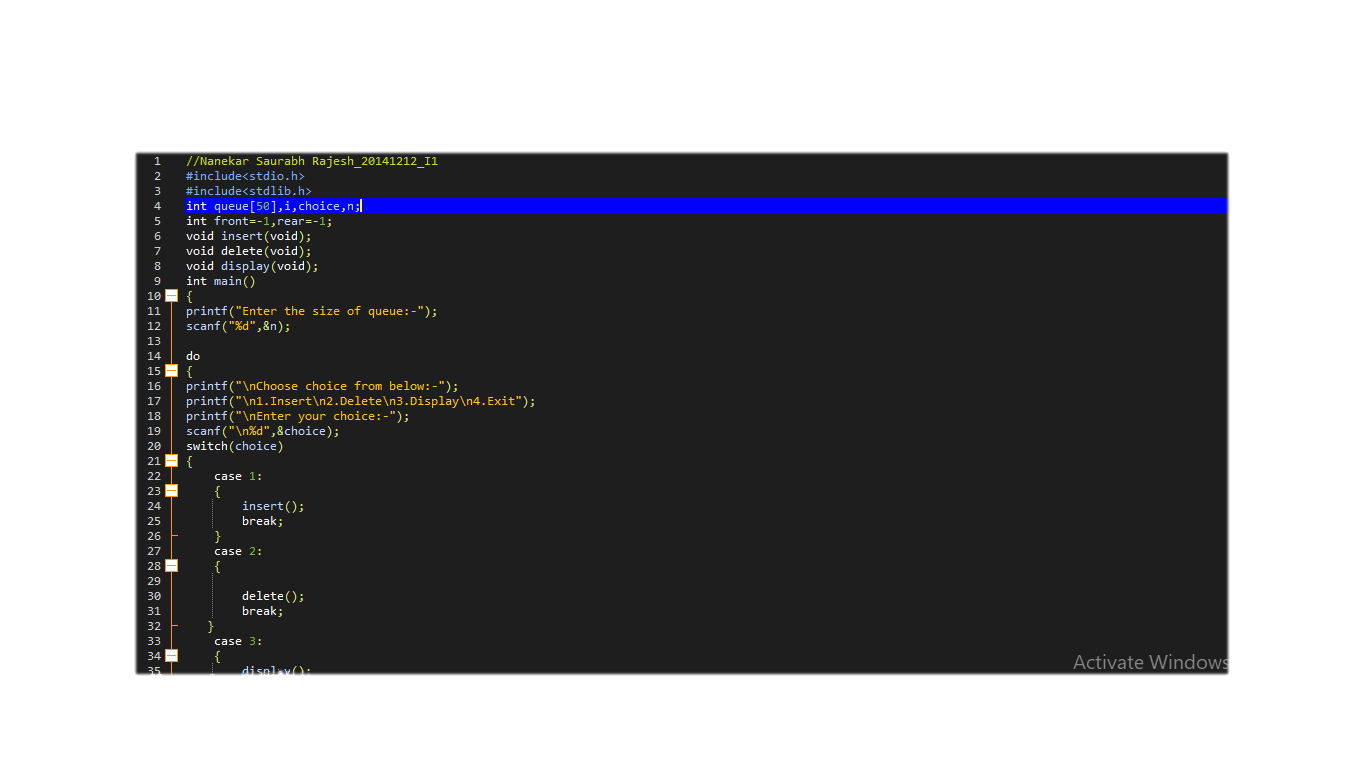
else

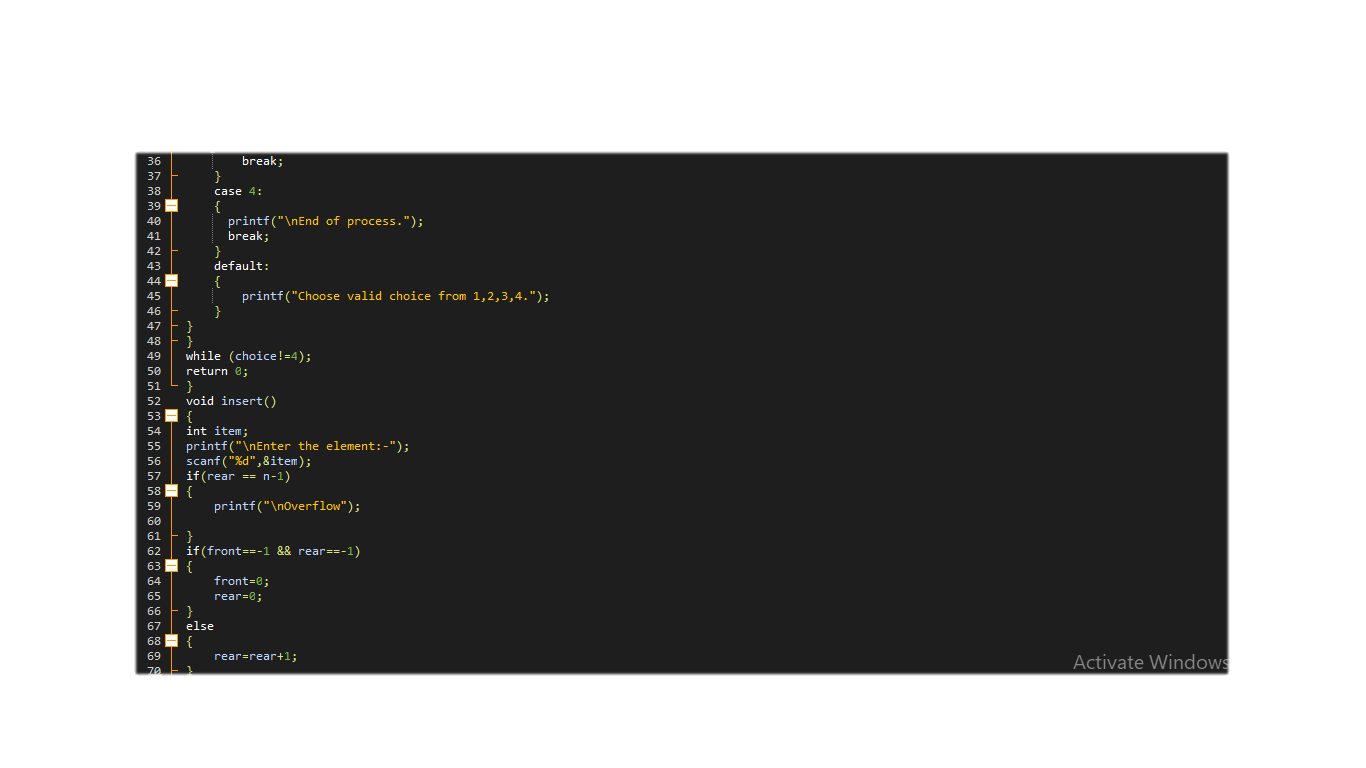
printf("\nThe Queue is empty.");

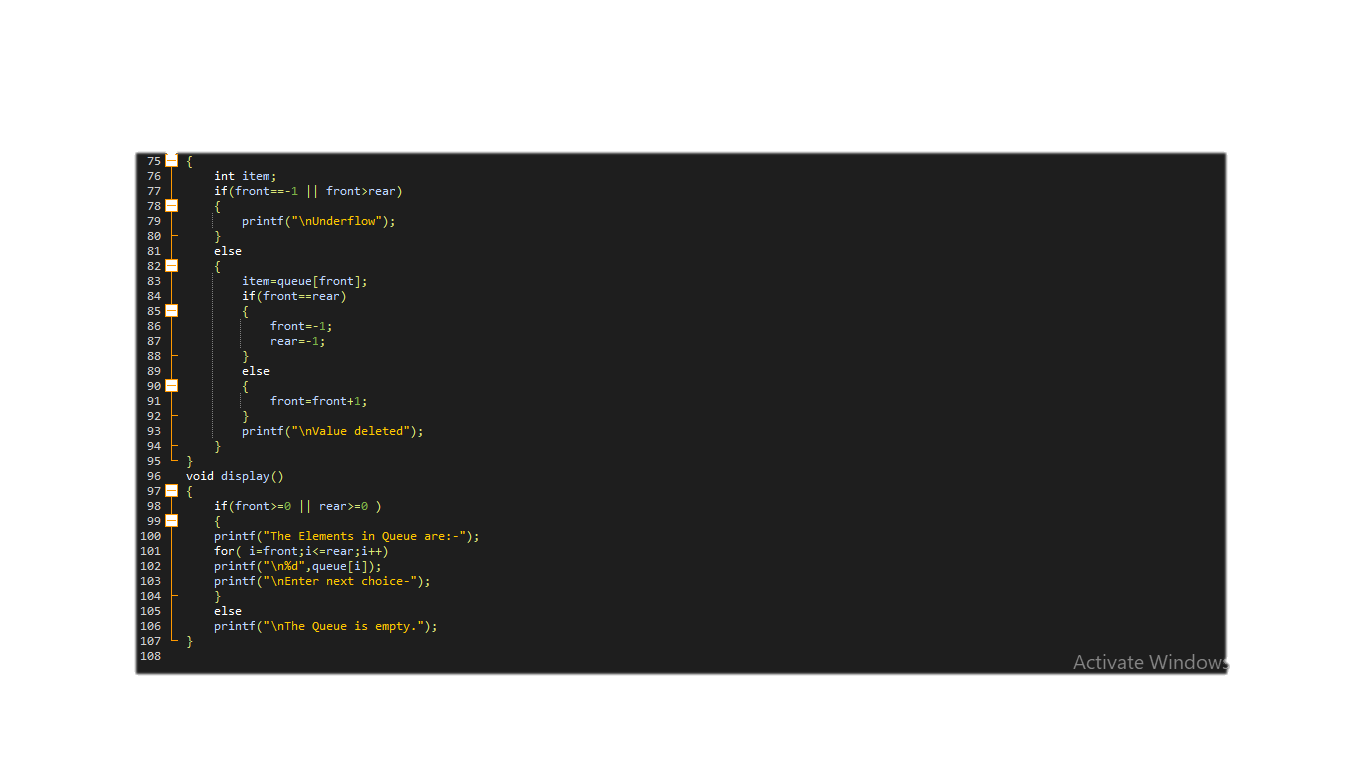
}

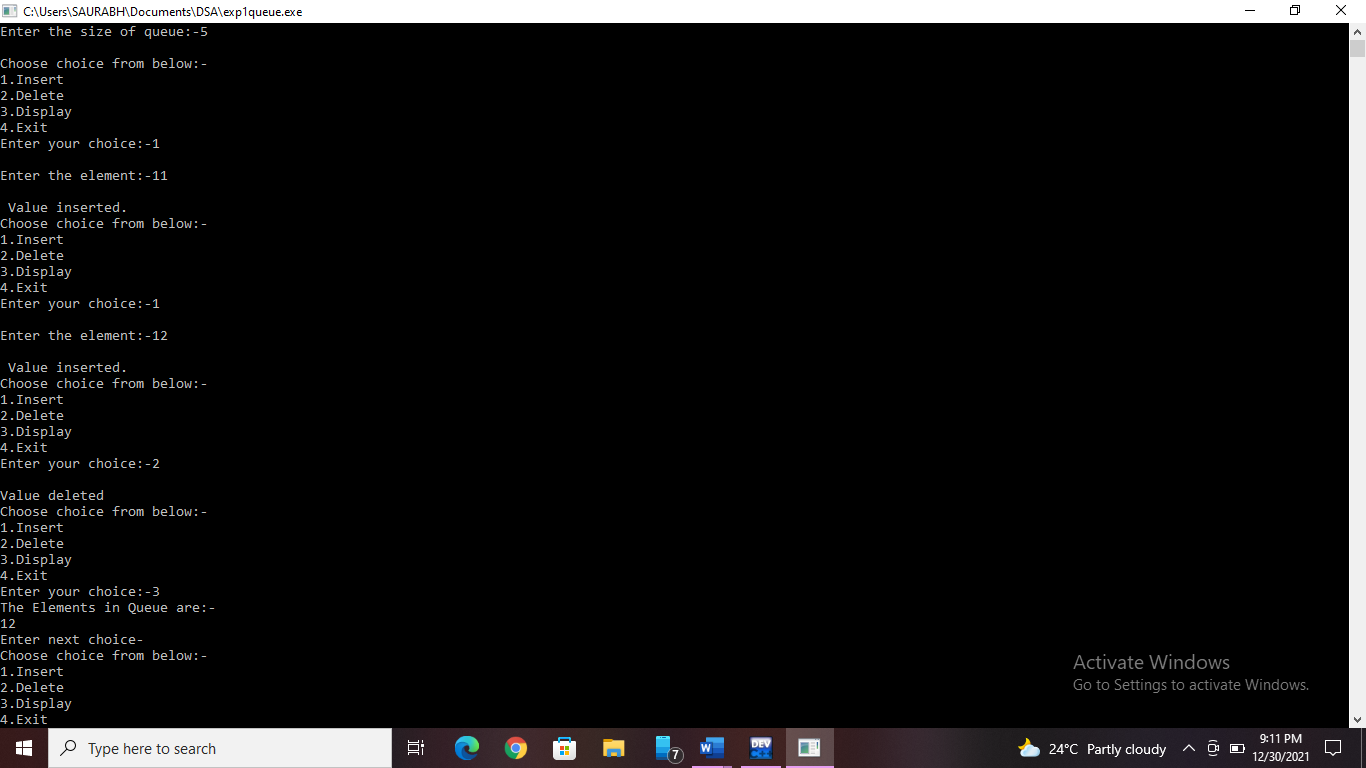
**Screenshots of Output:**

**Queue:-**









**Practice Program:**Program to check if a queue of first n natural number can be sorted using a stack.

//Nanekar Saurabh Rajesh\_20141212\_I1

#include<stdio.h>

#include<conio.h>

#include<stdbool.h>

#define MAX 100

int stack[MAX],queue[MAX],top=-1,front=-1,rear=-1;

void enqueue(int x,int n)

{

if(rear>=n-1)

return;

if(front==-1 && rear==-1)

front=0;

queue[++rear]=x;

}

void dequeue()

{

if(front==-1 && front>rear)

return;

if(front==rear)

{

front=rear=-1;

}

front++;

}

int Front()

{

if(front==-1 || front>rear)

return -1;

return queue[front];

}

bool qEmpty()

{

if(front==-1|| rear==-1 || front>rear)

return true;

else

return false;

}

void display()

{

for(int i=front;i<=rear;i++)

printf("%d\n",queue[i]);

}

void push(int x)

{

if(top>=MAX-1)

return;

stack[++top]=x;

}

void pop()

{

if(top==-1)

return;

top--;

}

int Top()

{

if(top==-1)

return -1;

return stack[top];

}

bool sEmpty()

{

if(top==-1)

return true;

else

return false;

}

bool checkSorted(int n)

{

int expected=1;

int fnt;

while(!qEmpty())

{

fnt=Front();

dequeue();

if(fnt==expected)

{

expected++;

}

else{

if(sEmpty())

push(fnt);

else if(!sEmpty() && fnt>Top())

{

return false;

}

else

push(fnt);

}

}

while(!sEmpty() && Top()==expected)

{

pop();

expected++;

}

if(expected-1==n && sEmpty())

return true;

return false;

}

int main()

{

int n;

printf("Enter size of the queue: ");

scanf("%d",&n);

for(int i=0;i<n;i++)

{

int x;

printf("Enter a value for given queue:");

scanf("%d",&x);

enqueue(x,n);

}

bool ans=checkSorted(n);

if(ans)

printf("\nGiven queue can be sorted\n");

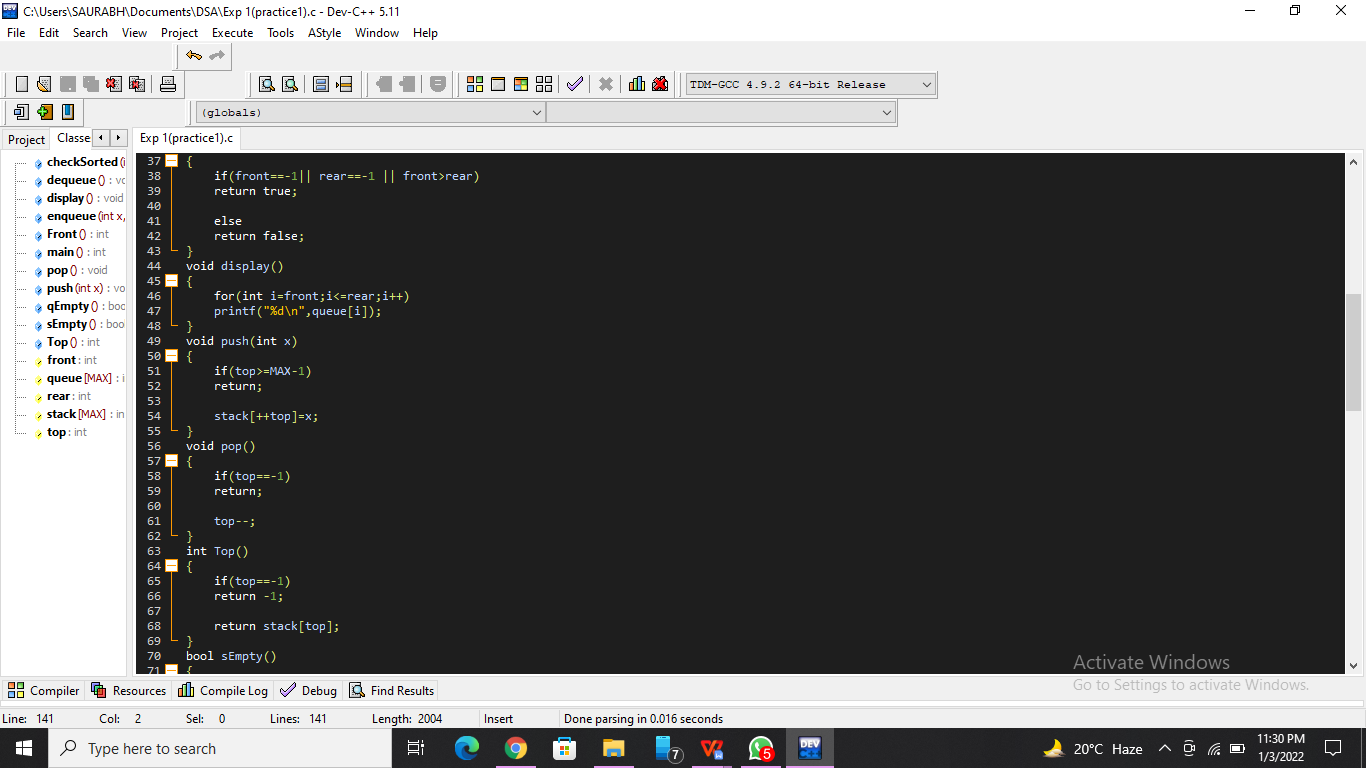
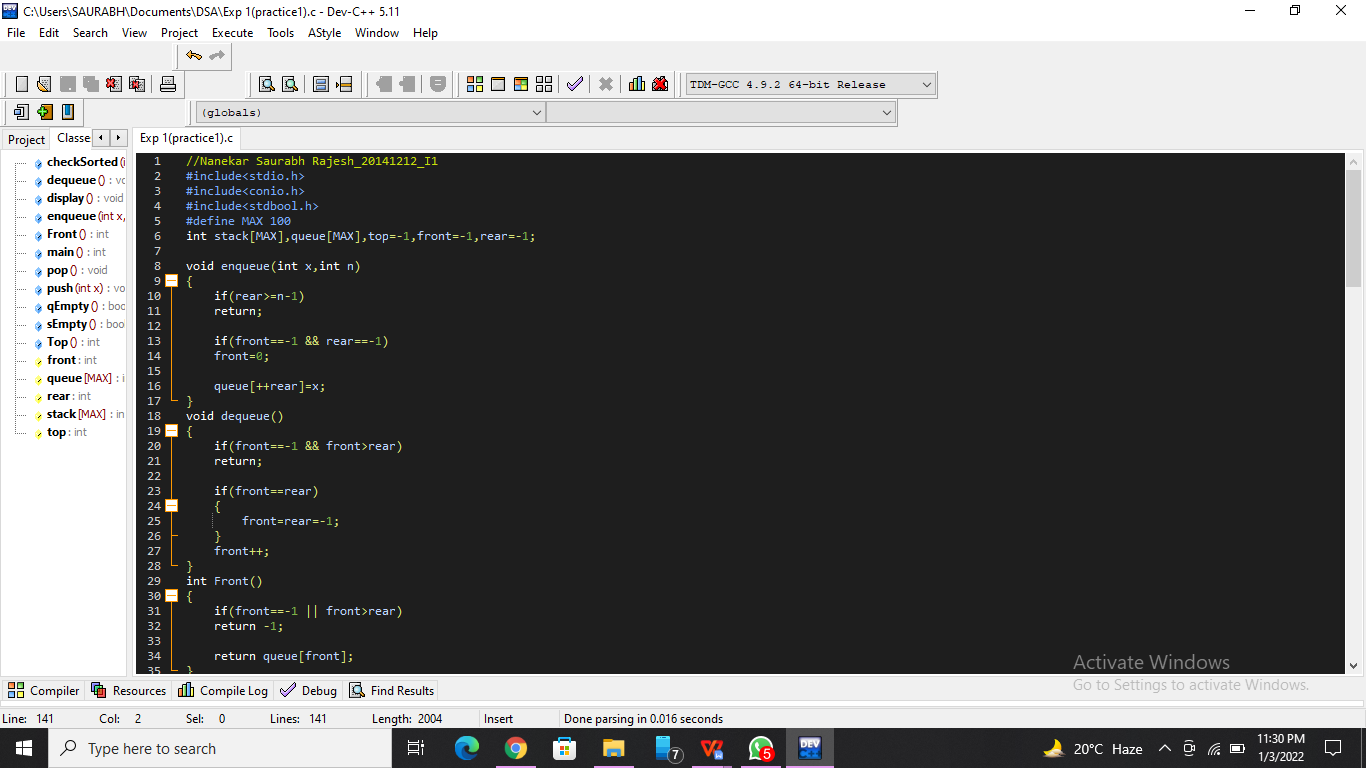
else

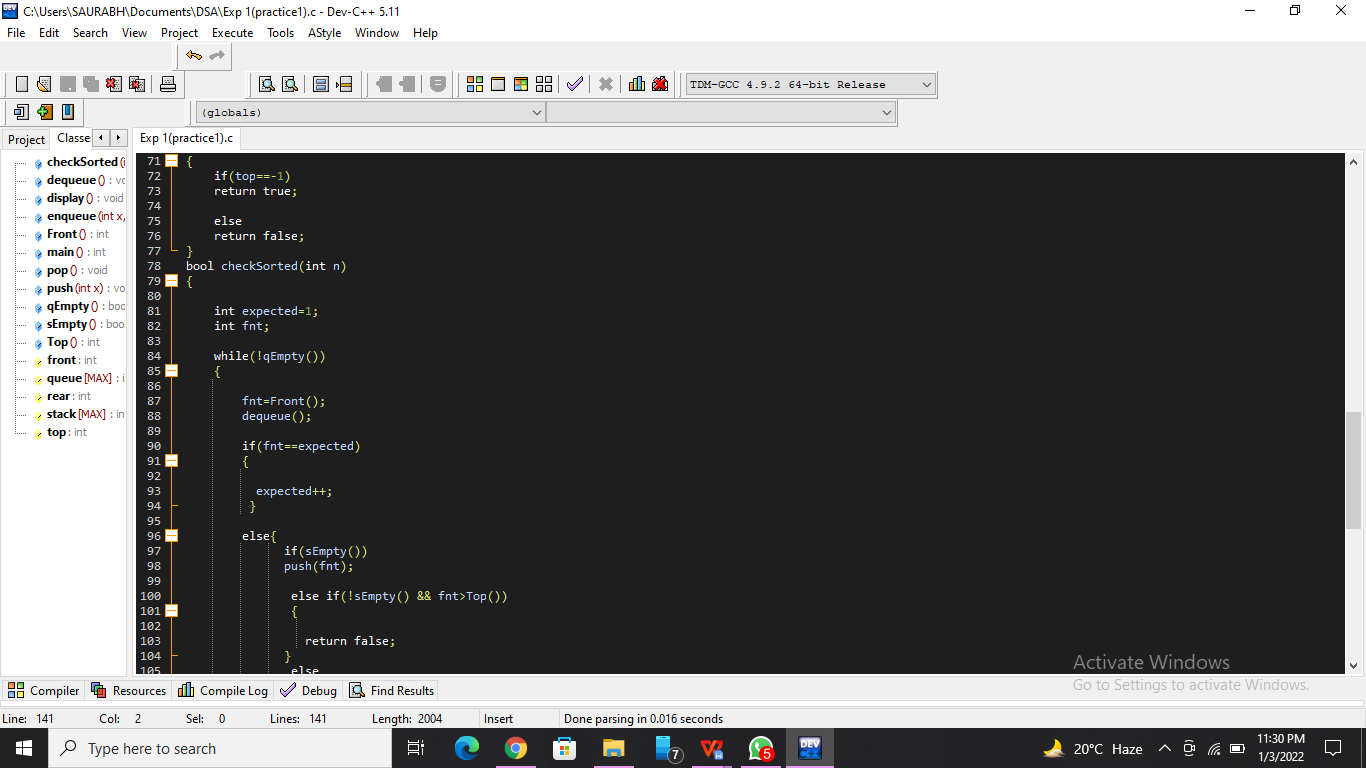
printf("\nGiven queue cannot be sorted\n");

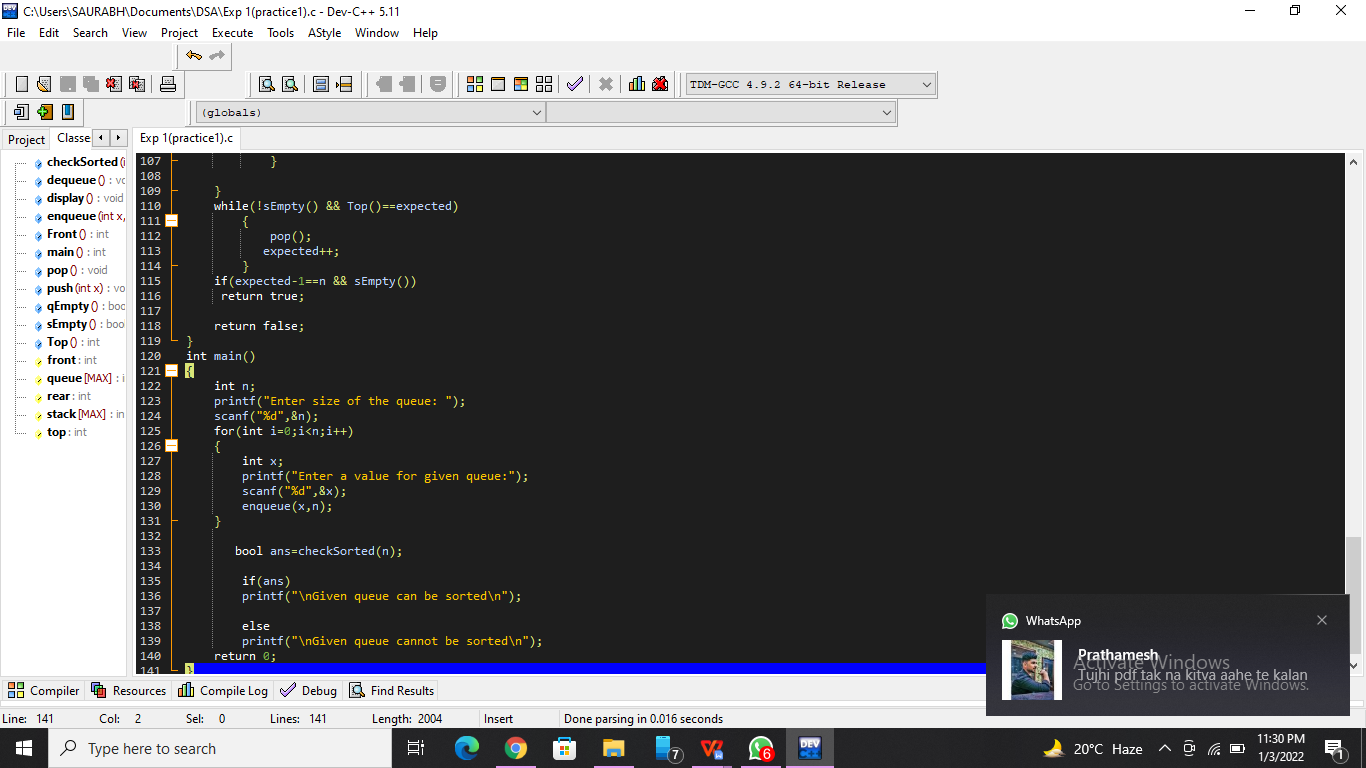
return 0;

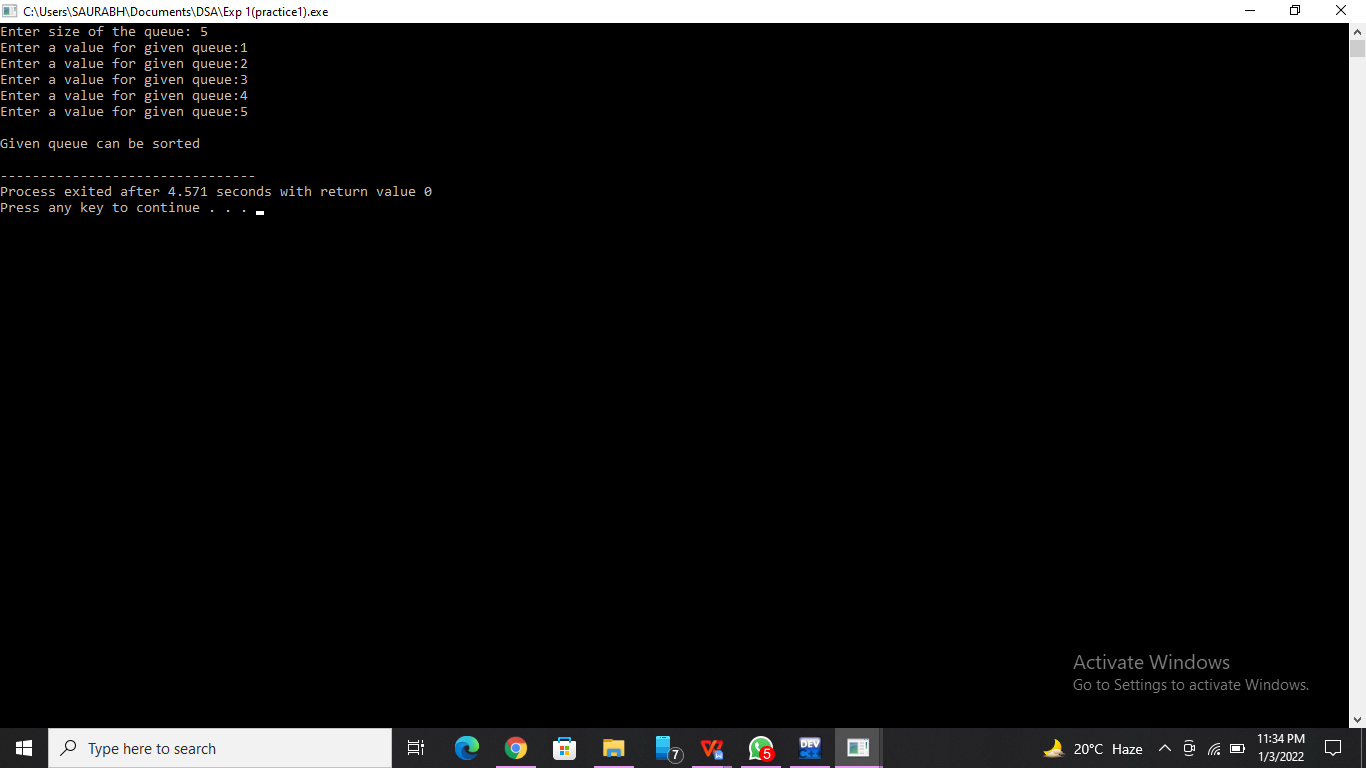
}

**Screen shots of Practice Program:-**

****

****

****

****

**List of sample questions for oral examination:**

1. Explain why Stack is a recursive data structure
2. Define Stack
3. Why and when should I use Stack or Queue data structures instead of Arrays/Lists?
4. Why Are Stacks Useful?
5. Implement a Queue using two Stacks.

**Conclusion:**

Use a queue when you want to get things out in the order that you put them in. Use a stack when you want to get things out in the reverse order than you put them in.