MINI PROJECT REPORT

ON

IMPLEMENTATION OF STACK AND QUEUE

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL, 575025

COURSE TITLE

Computer Graphics(IT254)

SUBMITTED BY:

1. PADMASALI ALEKHYA(15IT226)
2. DUDEKULA DEENA (15IT209)
3. SOWMYA.R(15IT246)

SUBMITTED TO:

Mrs.R.Priyadarshini

(Course Instructor)

CERTIFICATE

This is to certify that the mini project entitled “IMPLEMENTATION OF STACK AND QUEUE” has been presented by, Alekhya(15IT226), Sowmya.R(15IT246) and Deena(15IT209), students of second year batch(IT) , Department of Information Technology, NITK on 6th April, during the even semester of academic year 2016-17,in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in IT at NITK,Surathkal**.**

Place: Surathkal **(Signature of Course Instructor)**

Date: 06/04/2017

CONTENTS

INTRODUCTION

In this project we are going to give the OpenGL program on Stack and Queue. The implementation demonstration of Stack and Queue using the computer graphics and OpenGL .

User can choose the operation and performs the task of Either Stack or Queue. The push, pop can be perform in Stack, while insert, delete can be done on Queue. With the input from the user the task is perform and is based on the button pressed.

PROBLEM STATEMENT:

Implementing stack and queue in OpenGL by performing push and pop operations in stack and insert and delete operations in queue using buttons .

OBJECTIVES:

- To visualize push and pop, insert and delete operations in stack and queue respectively in computer graphics with OpenGL.
- Implementing certain technical concept like use of Idle Function, mouse Function based on the buttons pressed.
- To show that implementation of stack and queue is easier with OpenGL.

METHODOLOGY:

- **void glutInit(int argc, char **argv);**

Initializes GLUT and processes any command-line arguments. The command line options are dependent on the window system. glutInit() should be called before any other GLUT routine.

- **void glutInitDisplayMode(unsigned int mode);**

  Specifies a display mode for windows created when glutCreateWindow() is called. The mask argument is a bitwise Ored combination of GLUT_RGBA or GLUT_INDEX, GLUT_SINGLE or GLUT_DOUBLE, and any of the buffer-enabling flags: GLUT_DEPTH, GLUT_STENCIL, or GLUT_ACCUM. The default value is GLUT_RGBA | GLUT_SINGLE.

- **void glutInitWindowSize(int width, int height);**

  Specifies the width and height, in pixels, of your window. The initial window size is only a hint and may be overridden by other requests.

- **int glutCreateWindow(char *name);**

  Creates a window with an OpenGL context using the previously set characteristics (display mode, width, height, etc.) The string name appears in the title bar. The window is not initially displayed until glutMainLoop() is entered; consequently, no rendering should be done until then. The value returned is a unique identifier for the window. This identifier can be used for controlling and rendering to multiple windows (each with an OpenGL rendering context) from the same application.

- **void glutDisplayFunc(void (*func)(void));**

  Specifies the function func that is called whenever the contents of the window need to be redrawn. This may happen when the window is initially opened, when the window is popped and window damage is exposed, and when glutPostRedisplay() is explicitly called.

- **void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);**

  Sets the current clearing color for use in clearing color buffers in RGBA mode.

- **void glClear(GLbitfield mask);**

  Clears the specified buffers to their current clearing values. The mask argument is a bitwise logical OR combination of the values listed.

- **void glFlush(void);**
  Forces previously issued OpenGL commands to begin execution, thus guaranteeing that they complete.
- **void glutPostRedisplay(void);**
  Marks the current window as needing to be redrawn. At the next opportunity, the callback function registered by glutDisplayFunc() will be called.

- Here function"button" is used to create a button and here we are creating 4 buttons for push, pop, insert and delete operations.
  button buttonname(int x11 ,int y11,int x22,int y22,char *stringname);
  this is the syntax for creating a button with required dimensions and text on it.
  Here in class button 3 functions are declared they are, **void draw(),void togglestate(), int insidebutton(int x,int y).**

- when **draw();** function is called.,
  - **setFont(GLUT_BITMAP_TIMES_ROMAN_24);** is used to set the font style to times roman with font size 24.
  - **glColor3f(1.000, 0.647, 0.000);** is used to set colour for the text on button.
  - **drawstring(x1+10,y1+10,str);** is used to set the position of string(str) on the button.
  - **glColor3f(1.000, 0.271, 0.000);** is used to set the colour for the button.
  - **glBegin(GL_POLYGON);**is used to draw boundary of a simple convex polygon.
  - **glVertex2f(x1,y1);** is used to specify a vertex for use in describing a geometric object.
- Here **st** and **q** are references for the classes stack and queue respectively**.**
- When **togglestate()** is called state variable will toggle from 0 to 1 or 1 to 0.
- **void glutMouseFunc(void (*func)(int button, int state, int x, int y));**
  Specifies the function func that is called when a mouse button is pressed or released. The button callback parameter is GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, or GLUT_RIGHT_BUTTON. The state callback parameter is either GLUT_UP or GLUT_DOWN, depending on whether the mouse button has been released or pressed. The x and y callback parameters indicate the location (in window-relative coordinates) of the mouse pointer when the event occurred.
  - if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) is for pressing mouse button condition. In this we have 4 conditions again

-
  ```
  if(btn1.insidebutton(x,y)){
          btn1.togglestate();
          if(!st.stfull())
          st.push(itemno++);
  }
  ```

  The above condition will check whether the pressed position is inside the button btn1 or not, if the condition is true,the state of button btn1 will toggle and if the stack is not full the entered element will be pushed into the stack.

  Similarly there will be 3 conditions like above to perform the same actions mentioned above for deleting element from stack and for inserting and deleting the elements for queue.

  - if(btn==GLUT_LEFT_BUTTON && state == GLUT_UP) is for releasing mouse button. In this we have 4 conditions

    -
      ```
      if(btn1.insidebutton(x,y)){
                          btn1.togglestate();
      }
      ```

The above condition will toggle the state immediately after releasing the button mentioned in condition. Similarly other 3 conditions perform the same thing for mentioned buttons .

- **void glutPostRedisplay(void);**
  Marks the current window as needing to be redrawn. At the next opportunity, the callback function registered by glutDisplayFunc() will be called.
- **void glutIdleFunc(void (*func)(void));**
  Specifies the function func to be executed if no other events are pending. If NULL is passed in , execution of func is disabled.
- **void glutMainLoop(void);**
  Enters the GLUT processing loop, which is an infinite loop. Registered callback functions are called whenever the corresponding event occurs.
- **st.displaystack();**will call the displaystack method in stack function.
  ```
  if (st.stempty())
      drawstring(10,10,"Stack Is Empty!");
    else {
      for (i = st.top; i >= 0; i--)
          st.s[i].draw();
  ```
  The above condition will display “Stack Is Empty!” for empty stack and contents from 0 to top of stack.
- **q.displayqueue();**will call the displayqueue method in queue function

```
if(front==-1)
drawstring(260,10," No elements to display in queue");
else
   {
   for(i=front;i<=rear;i++) {
        que[i].draw(); }
   }
```

The above condition will display " No elements to display in queue" for empty queue or its content from front to rear.

In above two , draw() function is common which will draw the block of mentioned size with given element.

SYSTEM SPECIFICATIONS AND REQUIREMENTS:

Hardware requirements:

- Pentium or higher processor.
- A standard keyboard, and Microsoft compatible mouse
- VGA monitor.
- If the user wants to save the Created files a secondary storage medium can be Used.

Software requirements:

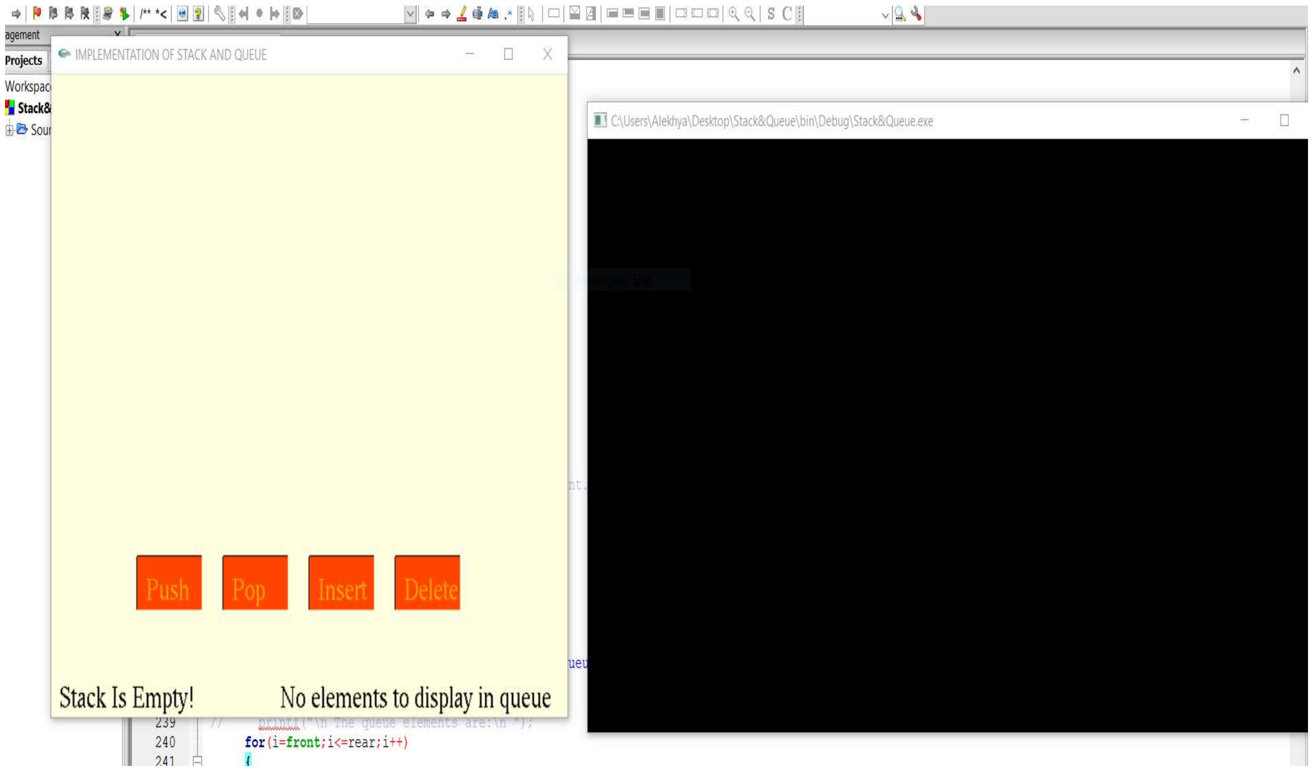
- The graphics package has been designed for OpenGL; hence the machine must have CodeBlocks.
- Glut files are required to run the code in codeblocks.
- Windows OS.

User Requirement:

- Easy to understand and should be simple.
- The built-in functions should be utilized to maximum extent.
- OpenGL library facilities should be used.

SCREENSHOTS:

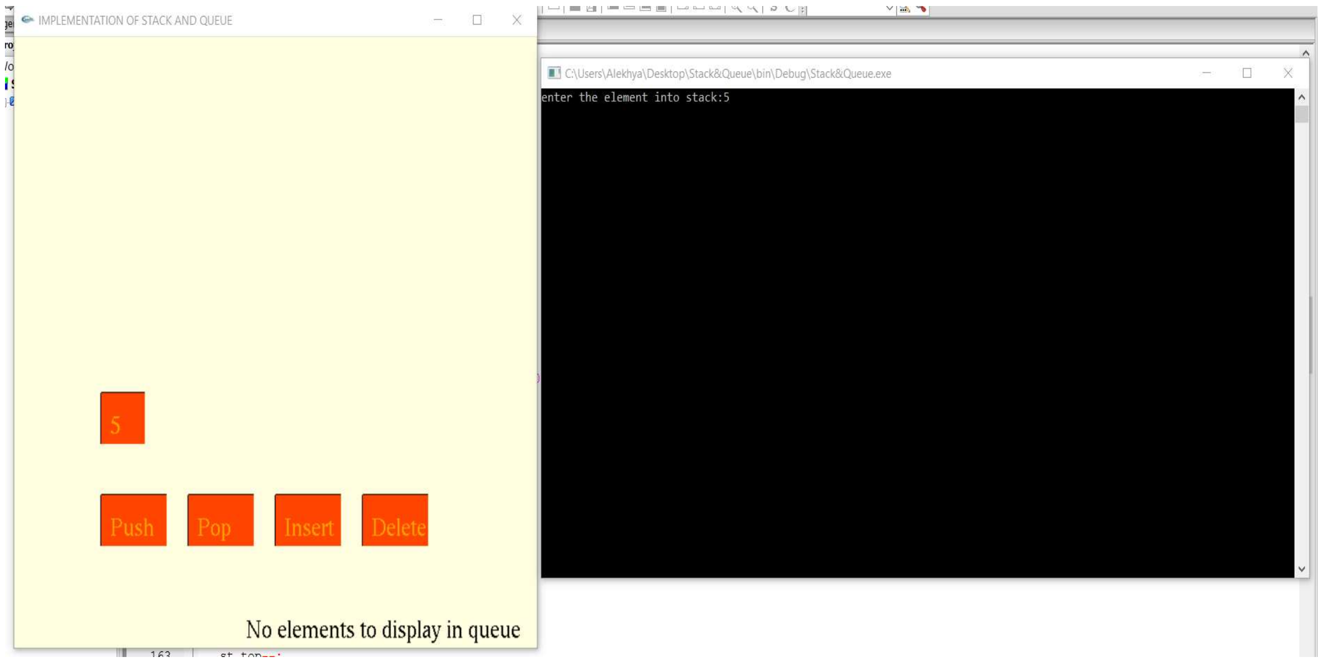
1. Screenshot displaying the windows on which we have to perform our required operations.


IMPLEMENTATION OF STACK AND QUEUE
C:\Users\Alekhya\Desktop\Stack&Queue\bin\Debug\Stack&Queue.exe
Push
Pop
Insert
Delete

```
Stack Is Empty!          No elements to display in queue
```

```
//    printf("\n The queue elements are:\n ");
      for(i=front;i<=rear;i++)
      {
```

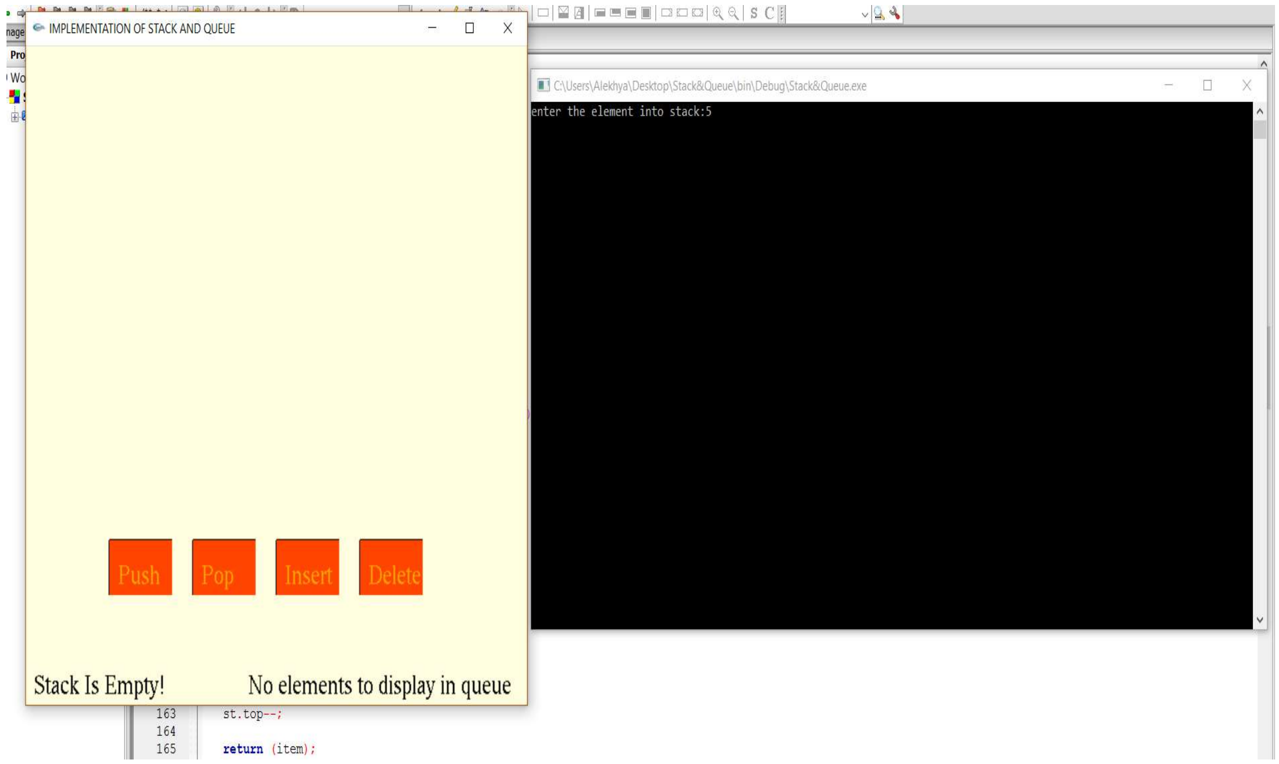
2. Screenshot displaying push operation in stack on pressing push button and entering element in output window.


IMPLEMENTATION OF STACK AND QUEUE
C:\Users\Alekhya\Desktop\Stack&Queue\bin\Debug\Stack&Queue.exe
5
Push
Pop
Insert
Delete

```
enter the element into stack:5
```

```
No elements to display in queue
```

3. Screenshot displaying pop operation in stack by pressing pop button.



Stack Is Empty!
No elements to display in queue

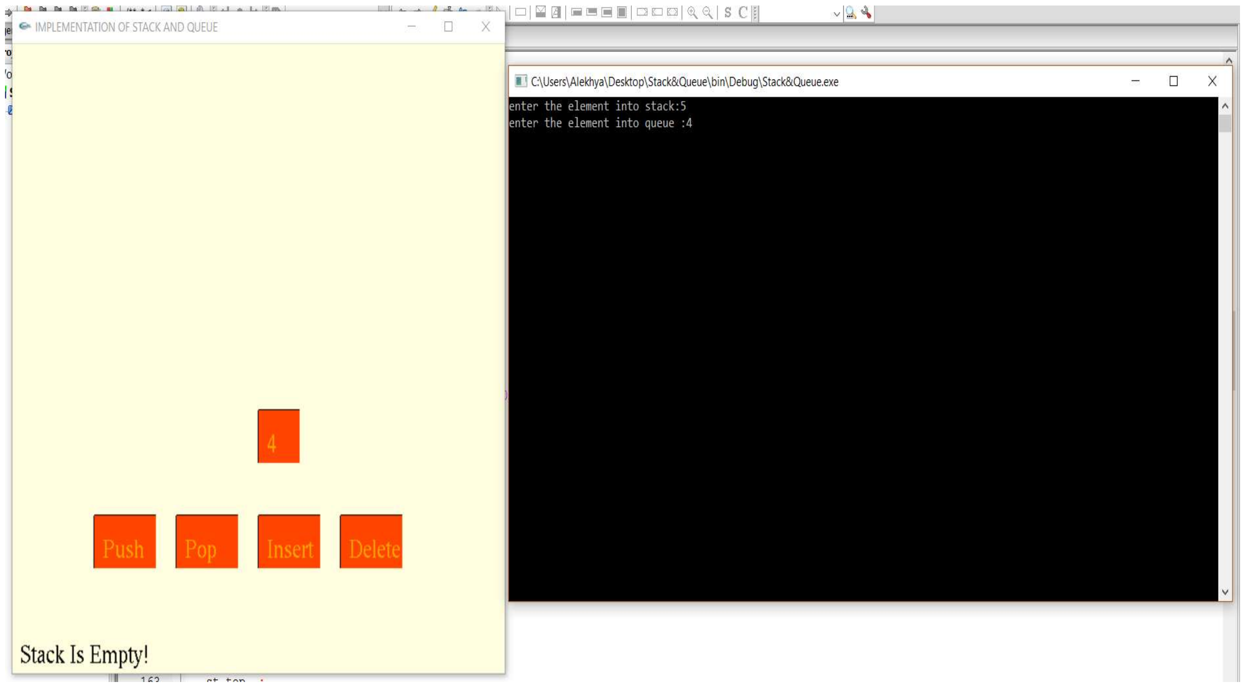
C:\Users\Alekhya\Desktop\Stack&Queue\bin\Debug\Stack&Queue.exe

```
enter the element into stack:5
```

```
st.top--;

return (item);
```

4. Screenshot displaying enqueue operation by pressing insert button and by entering element in output window.

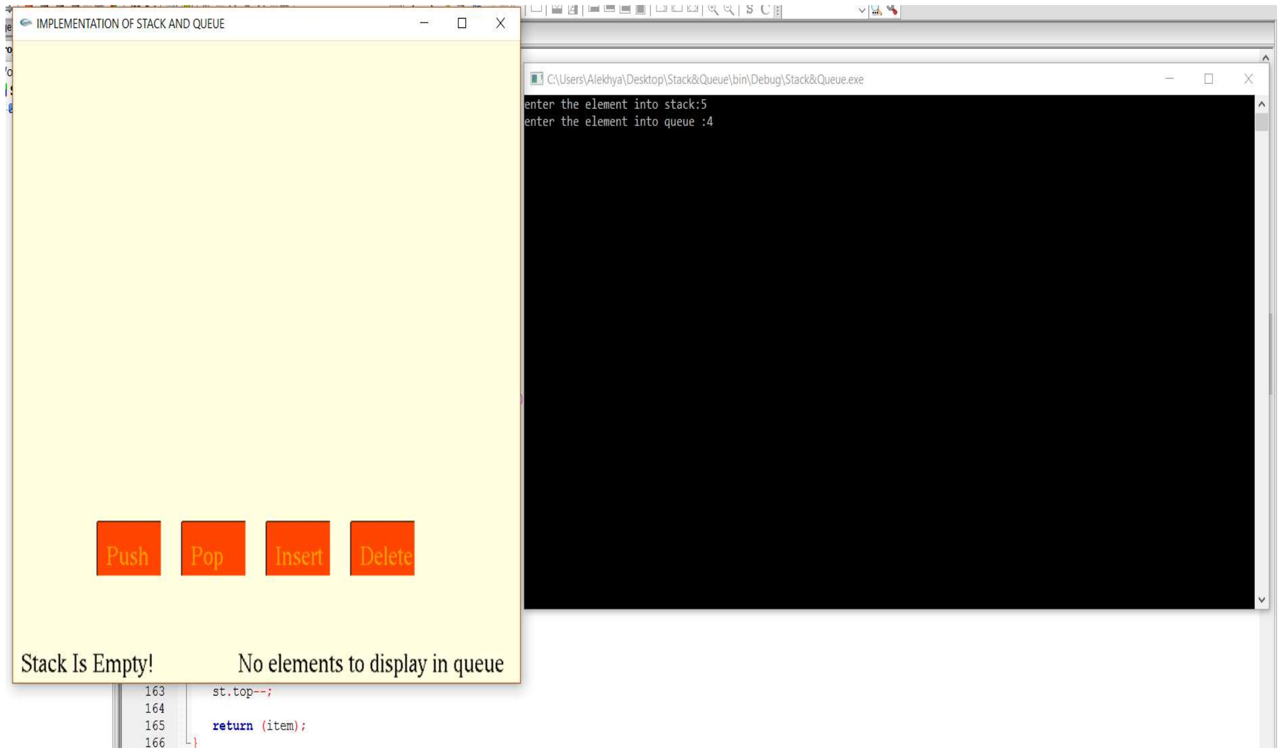


Stack Is Empty!

C:\Users\Alekhya\Desktop\Stack&Queue\bin\Debug\Stack&Queue.exe

```
enter the element into stack:5
enter the element into queue :4
```

5. Screenshot displaying dequeue operation in stack by pressing delete button.


IMPLEMENTATION OF STACK AND QUEUE
Push
Pop
Insert
Delete
C:\Users\Alekhya\Desktop\Stack&Queue\bin\Debug\Stack&Queue.exe

Stack Is Empty!

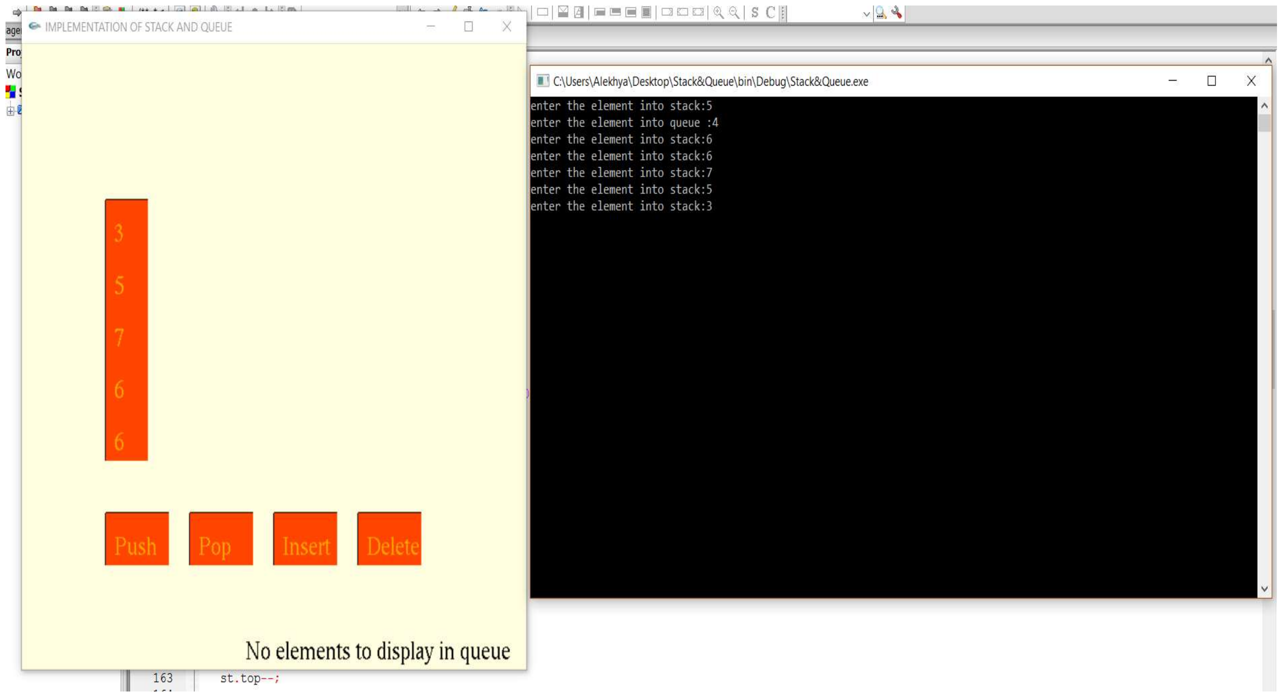
No elements to display in queue

```
enter the element into stack:5
enter the element into queue :4
```

```
st.top--;

return (item);
}
```

6. Screenshot displaying that the stack is full i.e., it will take only max elements(5).

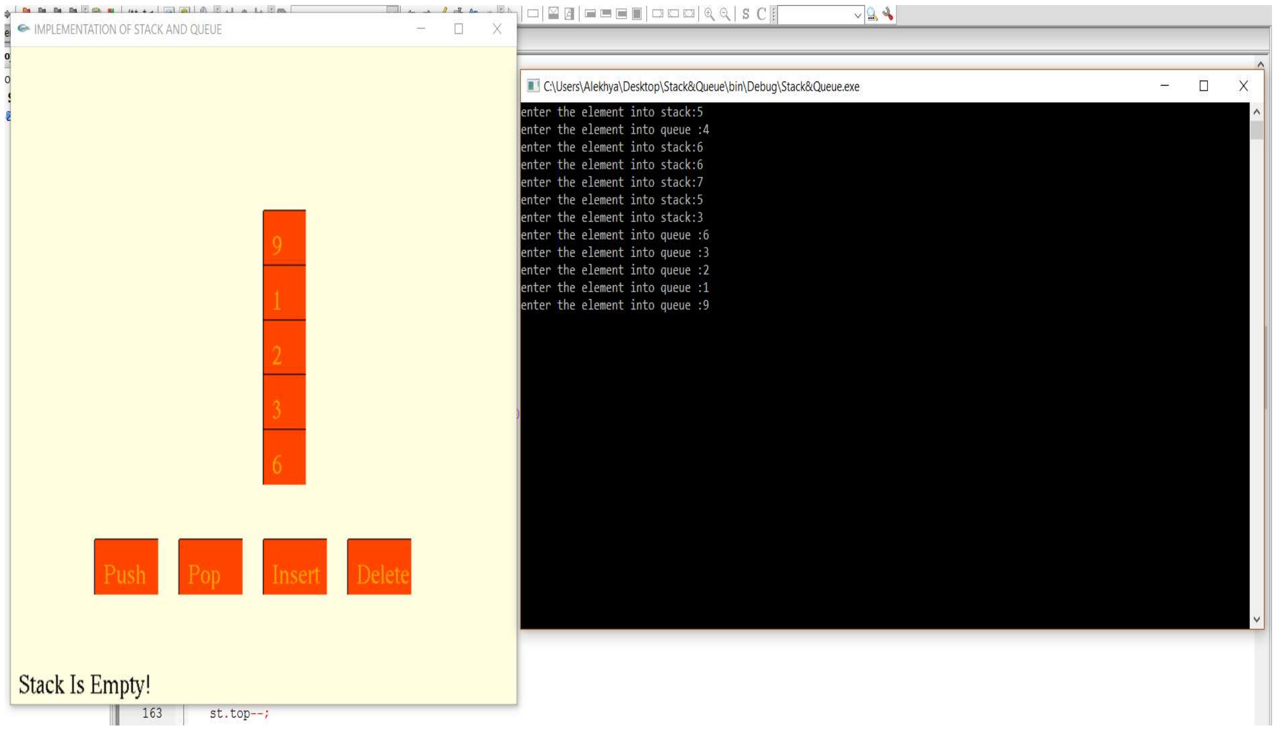

IMPLEMENTATION OF STACK AND QUEUE
3
5
7
6
6
Push
Pop
Insert
Delete
C:\Users\Alekhya\Desktop\Stack&Queue\bin\Debug\Stack&Queue.exe

No elements to display in queue

```
enter the element into stack:5
enter the element into queue :4
enter the element into stack:6
enter the element into stack:6
enter the element into stack:7
enter the element into stack:5
enter the element into stack:3
```

```
st.top--;
```

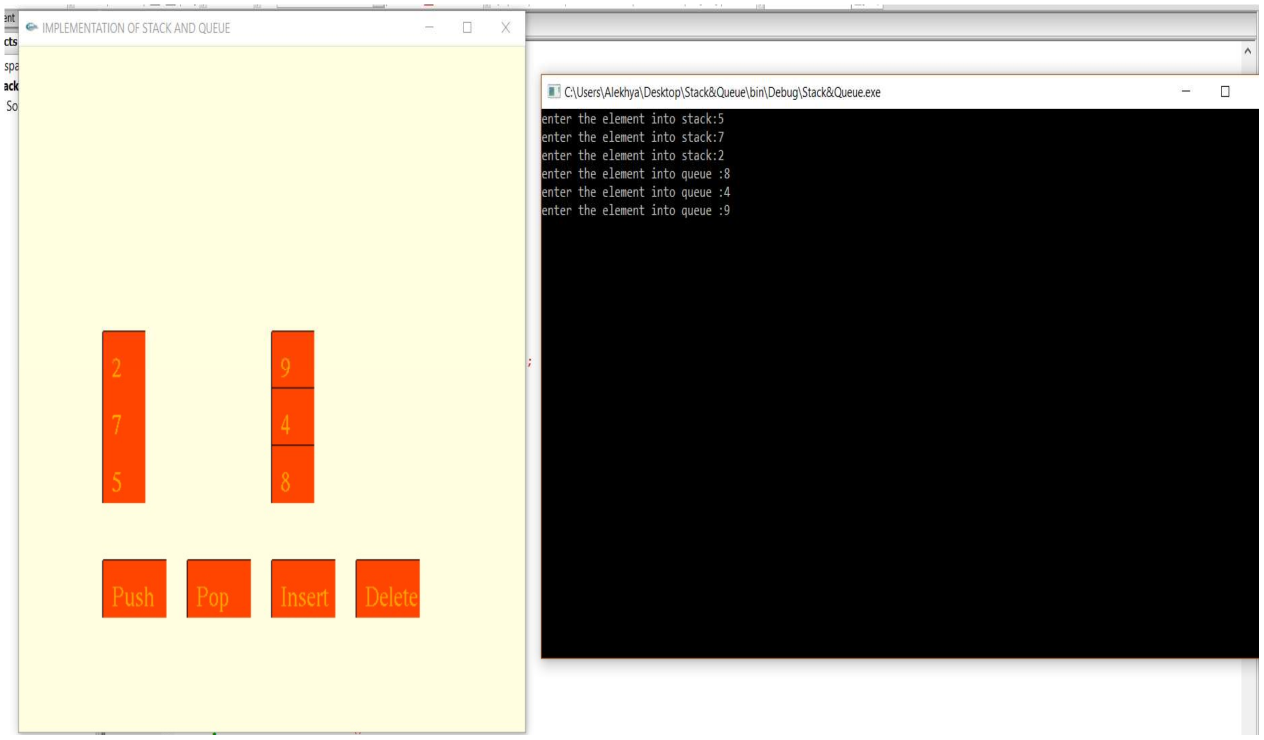
7. Screenshot displaying that the queue is full i.e., it will take only max elements(5).


IMPLEMENTATION OF STACK AND QUEUE
9
1
2
3
6
Push
Pop
Insert
Delete
Stack Is Empty!
C:\Users\Alekhya\Desktop\Stack&Queue\bin\Debug\Stack&Queue.exe

```
enter the element into stack:5
enter the element into queue :4
enter the element into stack:6
enter the element into stack:6
enter the element into stack:7
enter the element into stack:5
enter the element into stack:3
enter the element into queue :6
enter the element into queue :3
enter the element into queue :2
enter the element into queue :1
enter the element into queue :9
```

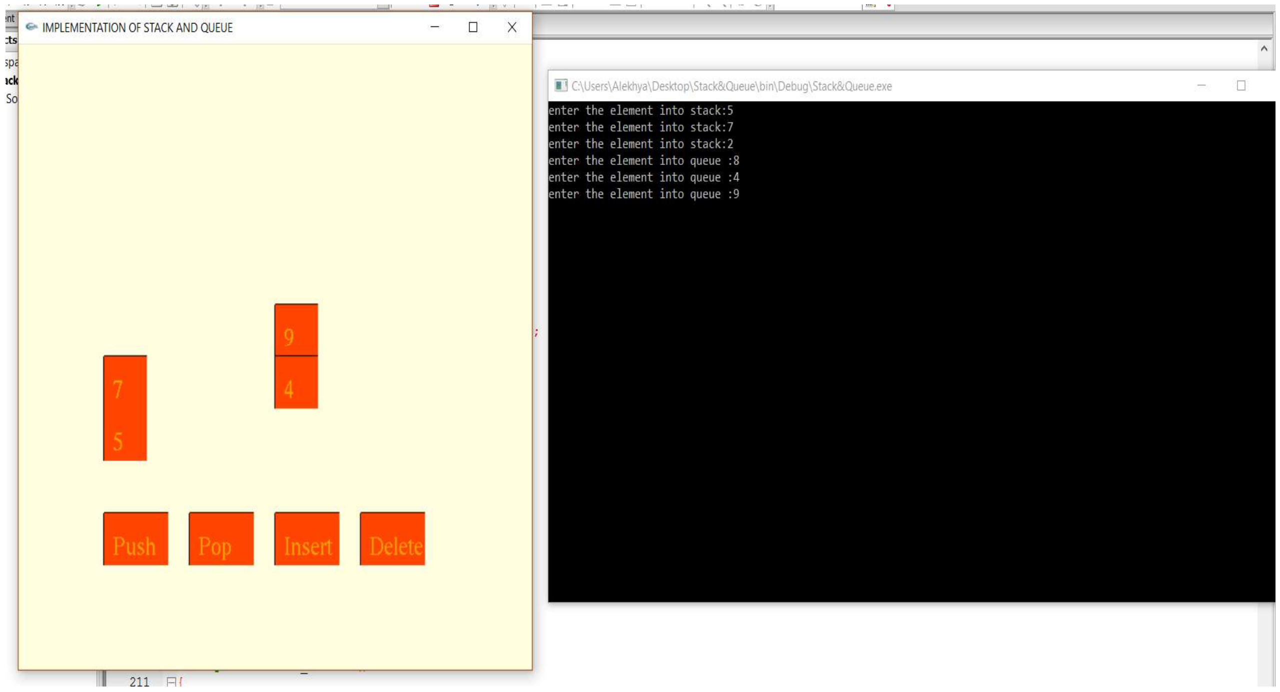
```
st.top--;
```

8. Screenshot displaying how stack and queue are taking the elements when we press push and insert buttons respectively.


IMPLEMENTATION OF STACK AND QUEUE
2
7
5
9
4
8
Push
Pop
Insert
Delete
C:\Users\Alekhya\Desktop\Stack&Queue\bin\Debug\Stack&Queue.exe

```
enter the element into stack:5
enter the element into stack:7
enter the element into stack:2
enter the element into queue :8
enter the element into queue :4
enter the element into queue :9
```

9. Screenshot displaying how stack and queue are deleting the elements when we press pop and delete buttons respectively i.e.,in stack , last element will be deleted and in queue, first element will be deleted.


IMPLEMENTATION OF STACK AND QUEUE
7
5
9
4
Push
Pop
Insert
Delete
C:\Users\Alekhya\Desktop\Stack&Queue\bin\Debug\Stack&Queue.exe

```
enter the element into stack:5
enter the element into stack:7
enter the element into stack:2
enter the element into queue :8
enter the element into queue :4
enter the element into queue :9
```

FUTURE WORKS:

This is a very easy project to understand as we would have studied the working of stack and queue in our data structure classes. So, in future we can implement and understand other data structures easily with the help of Computer Graphics using OpenGL.

CONCLUSION

An attempt has been made to develop an OpenGL package which meets necessary requirements of the user successfully. Since it is user friendly, it enables the user to interact efficiently and easily.

The development of the mini project has given us a good exposure to OpenGL by which we have learnt some of the technique which help in development of animated pictures, gaming.

Hence it is helpful for us even to take up this field as our career too and develop some other features in OpenGL and provide as a token of contribution to the graphics world.

REFERENCES:

- http://www.geeksforgeeks.org/implement-stack-using-queue/
- http://stackoverflow.com/questions/688276/implement-stack-using-two-queues
- http://www.opengl-tutorial.org/beginners-tutorials/
- http://www.opengl-tutorial.org/

APPENDIX(CODE):

```
#include<Windows.h>
#include<GL/glut.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void *currentfont;
#define size 5
#define MAX 5
class button
{
    int x1,y1,x2,y2;
    int state;
    char str[10];

public:
    button()
    {

    }
    button(int x11,int y11,int x22,int y22,char *str1)
    {
            x1=x11;
            y1=y11;
            x2=x22;
            y2=y22;
            state=1;
            strcpy(str,str1);
    }
    void draw();
    void togglestate();
    int insidebutton(int x,int y);
};
class stack
{
   button s[size];
   int top;
public:
  stack()
  {
      top=-1;
  }
  int stfull();
  button pop();
  void push(int item);
  int stempty();
  void displaystack();
};
stack st;
class queue
{
button que[MAX];
int front,rear;
public:
queue()
{
    front=-1;
    rear=-1;
}
void displayqueue();
void insert_element();
void delete_element();
};
queue q;
void setFont(void *font)
{
    currentfont=font;
}
void drawstring(float x,float y,char *string)
{
        char *c;
    glRasterPos2f(x,y);

    for(c=string;*c!='\0';c++)
    {       glColor3f(0.0,0.0,0.0);
            glutBitmapCharacter(currentfont,*c);
    }
}
```

```
void button::draw()
{
    setFont(GLUT_BITMAP_TIMES_ROMAN_24);
    glColor3f(1.000, 0.647, 0.000);
            drawstring(x1+10,y1+10,str);
    glColor3f(1.000, 0.271, 0.000);
    glBegin(GL_POLYGON);
    glVertex2f(x1,y1);
    glVertex2f(x2,y1);
    glVertex2f(x2,y2);
    glVertex2f(x1,y2);
    glEnd();
    if(state==0)
    {
    glColor3f(0,0,0);
    glBegin(GL_LINES);
    glVertex2f(x1,y1);
    glVertex2f(x2,y1);
    glVertex2f(x2,y1);
    glVertex2f(x2,y2);
    glEnd();
    }
    else if(state==1)
    {
            glColor3f(0,0,0);
            glBegin(GL_LINES);
            glVertex2f(x1,y1);
            glVertex2f(x1,y2);
            glVertex2f(x1,y2);
            glVertex2f(x2,y2);
            glEnd();
    }
}
void button::togglestate(void)
{
    /*if(state==1)
            state=0;
    else if(state==0)
            state=1;*/
    state=(state==1)?0:1;
}
```

```
int button::insidebutton(int x,int y)
{
    if(x>x1&&x<x2&&y>y1&&y<y2)
            return 1;
    else return 0;
}
button btn1(100,100,175,150,"Push");
button btn2(200,100,275,150,"Pop");
button btn3(300,100,375,150,"Insert");
button btn4(400,100,475,150,"Delete");

//
// Stack functions start
//
int stack::stfull() {
   if (st.top >= size-1)
      return 1;
   else
      return 0;
}

void stack::push(int item) {
    char str[10];
    printf("enter the element into stack:");
    scanf("%d",&item);
    snprintf(str, sizeof(str), "%d", item);
   button btn(100,250+st.top*50,150,300+st.top*50,str);
    st.top++;

   st.s[st.top] = btn;

}

int stack::stempty() {
   if (st.top == -1)
      return 1;
   else
      return 0;
}

button stack::pop() {
   button item;
   item = st.s[st.top];
   st.top--;

   return (item);
}

void stack::displaystack() {
   int i;
   if (st.stempty())
      drawstring(10,10,"Stack Is Empty!");
   else {
      for (i = st.top; i >= 0; i--)
       st.s[i].draw();
   }
}
//
//stack functions end
//
//
// queue function starts
//
void queue::insert_element()
{
  static int num=0;
  char str[10];
  printf("enter the element into queue :");
     scanf("%d",&num);
     snprintf(str, sizeof(str), "%d", num++);
     button btn(300,250+rear*50,350,300+rear*50,str);
  if(front==0 && rear==4)
    drawstring(10,10," Queue OverFlow Occured");
  else if(front==-1&&rear==-1)
  {
      front=rear=0;
      que[rear]=btn;

  }
  else if(rear==MAX-1 && front!=0)
  {
    rear=0;
    que[rear]=btn;
  }
  else
  {
     rear++;
     que[rear]=btn;
  }
}
void queue::delete_element()
{
  button element;
  if(front>rear)
  {
     drawstring(260,10," Underflow");

  }
  element=que[front];
  if(front==rear)
     front=rear=-1;
  else
  {
    if(front==MAX-1)
      front=0;
    else
      front++;
  //     printf("\n The deleted element is: %s",element.str);
  }

}

void queue::displayqueue()
{
    int i;
    if(front==-1)
      drawstring(260,10," No elements to display in queue");
    else
     {
 //     printf("\n The queue elements are:\n ");
      for(i=front;i<=rear;i++)
      {
que[i].draw();
```

```
      }
    }
}
//
// queue function ends
//
void displaystacknqueue()
{
st.displaystack();
q.displayqueue();
}
void display()
{
    glClearColor(1.000, 1.000, 0.878,0.000);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);


btn1.draw();
btn2.draw();
btn3.draw();
btn4.draw();
displaystacknqueue();
glFlush();
glutSwapBuffers();
//glutPostRedisplay();
}
void mouse(int btn, int state, int x, int y)
{static int itemno=0;
    y=600-y;
    if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
    if(btn1.insidebutton(x,y))
    {
            btn1.togglestate();
            if(!st.stfull())
            st.push(itemno++);
    }
    if(btn2.insidebutton(x,y))
            {
                    btn2.togglestate();
                    if(!st.stempty())
```

```
                                        st.pop();

                }
        if(btn3.insidebutton(x,y))
                {
                        btn3.togglestate();
                        q.insert_element();
                }
        if(btn4.insidebutton(x,y))
                {
                        btn4.togglestate();
                        q.delete_element();
                }
        }
        if(btn==GLUT_LEFT_BUTTON && state == GLUT_UP)
        {
                if(btn1.insidebutton(x,y))
                        {
                                btn1.togglestate();
                        }
                if(btn2.insidebutton(x,y))
                                        {
                                                btn2.togglestate();
                                        }
                if(btn3.insidebutton(x,y))
                                        {
                                                btn3.togglestate();
                                        }
                if(btn4.insidebutton(x,y))
                                        {
                                                btn4.togglestate();
                                        }
        }
        glutPostRedisplay();
    }
    void init()
    {
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0,600,0,600);
        glMatrixMode(GL_MODELVIEW);
```

```
}
void idle()
{
   glutPostRedisplay();
}
int main(int argc, char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(600,600);
glutCreateWindow("IMPLEMENTATION OF STACK AND QUEUE");
glutDisplayFunc(display);
glutMouseFunc(mouse);
glutIdleFunc(idle);
glEnable(GL_DEPTH_TEST);
init();
glutMainLoop();
return 0;
}
```