**CHAPTER - 1**

**INTRODUCTION**

**1.1 Aim**

To develop a graphics package that utilizes the features of the computer graphics. The main aim of this project is to graphically illustrate the working of a queue.

Open Graphics Library is a cross-language, cross-platform application programming interface for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit, to achieve hardware-accelerated rendering.

**1.2 Overview**

A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

The process to add an element into queue is called Enqueue and the process of removal of an element from queue is called Dequeue.

In computer science, a double-ended queue is an abstract data type that generalizes a queue, for which elements can be added to or removed from either the front or back. It is also often called a head-tail linked list, though properly this refers to a specific data structure implementation of dequeue.

**1.3 Outcome**

Queue is a most important data structures that is used in day to day life, queues make decision making or running a process easy as the compiler or the operating system already has an order in which it needs to runs the processes.

**CHAPTER – 2**

**DESIGN AND IMPLEMENTATION**

**2.1 Algorithm**

**SIMPLE QUEUE**

**Algorithm for insertion**

Step1: If REAR >= SIZE – 1 then  
           Write “Queue is Overflow”  
Step2: REAR = REAR + 1  
Step3: QUEUE [REAR] = X  
Step 4: If FRONT = -1 then  
           FRONT = 0

**Algorithm for delete operation**

Step 1: If FRONT = -1 then  
           Write “Queue is Underflow”  
Step 2: Return QUEUE [FRONT]  
Step 3: If FRONT = REAR then  
           FRONT = 0  
           REAR = 0  
           Else  
           FRONT = FRONT + 1

**Algorithm for insertion at rear end**

Step -1: [Check for overflow]

if(rear==MAX)

Print("Queue is Overflow”);

return;

 Step-2: [Insert element]

  else

  rear=rear+1;

  q[rear]=no;

  [Set rear and front pointer]

  if rear=0

  rear=1;

  if front=0

front=1;

Step-3: return

**Algorithm for insertion at font end**

Step-1 : [Check for the front position]   
 if(front<=1)

Print (“Cannot add item at front end”);  
return;

Step-2 : [Insert at front]  
 else  
 front=front-1;  
 q[front]=no;

 Step-3 : Return  
**Algorithm for deletion from front end**

Step-1 : [ Check for front pointer]  
 if front=0   
 print(" Queue is Underflow”);  
 return;  
Step-2: [Perform deletion]  
 else  
 no=q[front];  
 print(“Deleted element is”,no);  
 [Set front and rear pointer]   
 if front=rear  
 front=0;  
 rear=0;  
 else  
 front=front+1;  
Step-3 : Return  
**Algorithm for deletion from rear end**

Step-1 : [Check for the rear pointer]  
 if rear=0  
 print(“Cannot delete value at rear end”);  
 return;  
Step-2: [ perform deletion]  
 else  
 no=q[rear];   
 [Check for the front and rear pointer]  
 if front= rear  
 front=0;  
 rear=0;  
 else  
 rear=rear-1;  
 print(“Deleted element is”,no);  
Step-3 : Return

**2.2 Flow chart**

start

main()

Init()

Display()

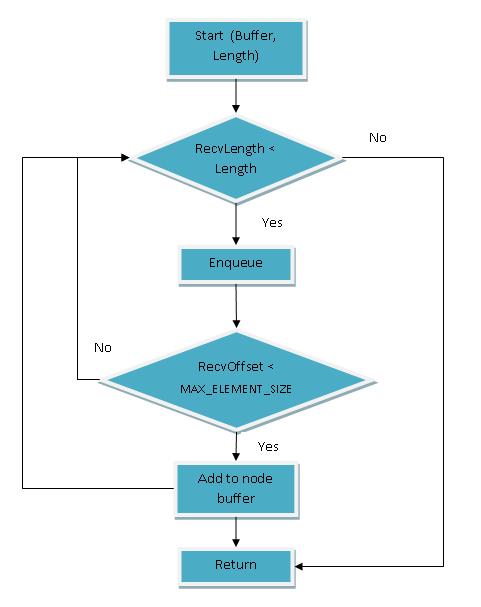
Operations()

Simple queue

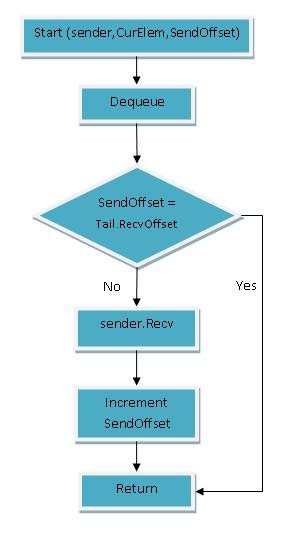
stop

De-queue

**Simple Queue**



**Double ended queue**

****

**2.3 OPEN GL API’s used description**

|  |  |
| --- | --- |
| MODE | PRIMITIVE TYPE |
| GL\_POINTS | The specified vertices are used to create a single point each. |
| GL\_LINES | The specified vertices are used to create line segments. Every two vertices specify a single and separate line segment. If the number of vertices is odd, the last one is ignored. |
| GL\_LINE\_STRIP | The specified vertices are used to create a line strip. After the first vertex, each subsequent vertex specifies the next point to which the line is extended. |
| GL\_LINE\_LOOP | This mode behaves like GL\_LINE\_STRIP, except a final line segment is drawn between the last and the first vertex specified. This is typically used to draw closed regions that might violate the rules regarding GL\_POLYGON usage. |
| GL\_TRIANGLES | The specified vertices are used to construct triangles. Every three vertices specify a new triangle. If the number of vertices is not evenly divisible by three, the extra vertices are ignored. |
| GL\_TRIANGLE\_STRIP | The specified vertices are used to create a strip of triangles. After the first three vertices are specified, each of any subsequent vertices is used with the two preceding ones to construct the next triangle. Each triplet of vertices (after the initial set) is automatically rearranged to ensure consistent winding of the triangles. |
| GL\_TRIANGLE\_FAN | The specified vertices are used to construct a triangle fan. The first vertex serves as an origin, and each vertex after the third is combined with the foregoing one and the origin. Any number of triangles may be fanned in this manner. |
| GL\_QUADS | Each set of four vertices is used to construct a quadrilateral (a four-sided polygon). If the number of vertices is not evenly divisible by four, the remaining ones are ignored. |
| GL\_QUAD\_STRIP | The specified vertices are used to construct a strip of quadrilaterals. One quadrilateral is defined for each pair of vertices after the first pair. Unlike the vertex ordering for GL\_QUADS, each pair of vertices is used in the reverse order specified to ensure consistent winding. |
| GL\_POLYGON | The specified vertices are used to construct a convex polygon. The polygon edges must not intersect. The last vertex is automatically connected to the first vertex to ensure the polygon is closed. |
| glBegin (GLenum mode) | This function is used in conjunction with glEnd to delimit the vertices of an OpenGL primitive. You can include multiple vertices sets within a single glBegin/glEnd pair, as long as they are for the same primitive type. You can also make other settings with additional OpenGL commands that affect the vertices following them. You can call only these OpenGL functions within a glBegin/glEndsequence: glVertex, glColor, glNormal, glEvalCoord, glCallList, glCallLists, glTexCoord, glEdgeFlag, and glMaterial. Note that display lists (glCallList(s)) may only contain the other functions listed here. |
| glEnd( ) | glEnd delimit the vertices that define a primitive or a group of like primitives. glBegin accepts a single argument that specifies in which of ten ways the vertices are interpreted. Taking n as an integer count starting at one, and N as the total number of vertices specified, the interpretations are as follows:  Parameters:*mode* GLenum: This value specifies the primitive to be constructed. It can be any of the values in Table |
| glColor3f(GLfloat red, GLfloat green, GLfloat blue) | glColor — set the current color Parameters red, green, blue*-* Specify new red, green, and blue values for the current color. Alpha *-* Specifie*s* a new alpha value for the current color. Included only in the four-argument glColor4 commands. |
| glVertex2f (GLfloat x, GLfloat y) | Specifies a vertex. Parameters x Specifies the x-coordinate of a vertex.*y* Specifies the y-coordinate of a vertex.The glVertex function commands are used within glBegin/glEnd pairs to specify point, line, and polygon vertices. The current color, normal, and texture coordinates are associated with the vertex when glVertex is called. When only *x* and *y* are specified, *z* defaults to 0.0 and *w* defaults to 1.0. When *x*, *y*, and *z* are specified, *w*defaults to 1.0. Invoking glVertex outside of a glBegin/glEnd pair results in undefined behavior. |
| glutBitmapCharecter(void\*font,int character)  void glutBitmapCharacter(void \*font, int character); | glutBitmapCharacter renders a bitmap character using OpenGL. font-Bitmap font to use. Character-Character to render (not confined to 8 bits). Description:Without using any display lists, glutBitmapCharacter renders the character in the named bitmap font. |
| GLUT\_BITMAP\_8\_BY\_13 | A fixed width font with every character fitting in an 8 by 13 pixel rectangle. The exact bitmaps to be used is defined by the standard X glyph bitmaps for the X font named: -misc-fixed-medium-r-normal--13-120-75-75-C-80-iso8859-1 |
| GLUT\_BITMAP\_9\_BY\_15 | A fixed width font with every character fitting in an 9 by 15 pixel rectangle. The exact bitmaps to be used is defined by the standard X glyph bitmaps for the X font named: -misc-fixed-medium-r-normal--15-140-75-75-C-90-iso8859-1 |
| GLUT\_BITMAP\_TIMES\_ROMAN\_10 | A 10-point proportional spaced Times Roman font. The exact bitmaps to be used is defined by the standard X glyph bitmaps for the X font named:-adobe-times-medium-r-normal--10-100-75-75-p-54-iso8859-1 |
| glutSwapBuffers(): | glutSwapBuffers swaps the buffers of the *current window* if double buffered. |
| glutPostRedisplay() | glutPostRedisplay marks the *current window* as needing to be redisplayed. |
| glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha) | specify clear values for the color buffers |
| GL\_COLOR\_BUFFER\_BIT | Indicates the buffers currently enabled for color writing. |
| GL\_DEPTH\_BUFFER\_BIT | Indicates the depth buffer. |
| glViewport(GL int x,GL int y,GLsizei width,GLsizei height) | set the viewport |
| glMatrixMode(GL enum mode) | Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: GL\_MODELVIEW, GL\_PROJECTION, and GL\_TEXTURE. The initial value is GL\_MODELVIEW. Additionally, if the ARB\_imaging extension is supported, GL\_COLOR is also accepted. |
| GL\_MODELVIEW | Applies subsequent matrix operations to the modelview matrix stack |
| GL\_PROJECTION | Applies subsequent matrix operations to the projection matrix stack. |
| GL\_TEXTURE- | Applies subsequent matrix operations to the texture matrix stack. |
| GL\_COLOR | Applies subsequent matrix operations to the color matrix stack. To find out which matrix stack is currently the target of all matrix operations, call glGet with argument GL\_MATRIX\_MODE. The initial value is GL\_MODELVIEW. |
| glLoadIdentity() | Replace the current matrix with the identity matrix. Description- glLoadIdentity replaces the current matrix with the identity matrix. It is semantically equivalent to calling glLoadMatrix with the identity matrix. |
| gluOrtho2D (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top) | Left, right- Specify the coordinates for the left and right vertical clipping planes.  Bottom, top- Specify the coordinates for the bottom and top horizontal clipping planes.  Description -gluOrtho2D sets up a two-dimensional orthographic viewing region. This is equivalent to calling glOrtho with near = -1 and far = 1 . |
| glRasterPos2i(GLint x, GLint y) | *x*, *y*, *z*, *w-* Specify the x, y, z, and w object coordinates (if present) for the raster position. |
| glutTimerFunc(unsigned int time,void(\*callback)(int),int value) | glutTimerFunc registers a timer callback to be triggered in a specified number of milliseconds. |

**CHAPTER – 3**

**RESULT ANALYSIS**

**3.1 snapshot**

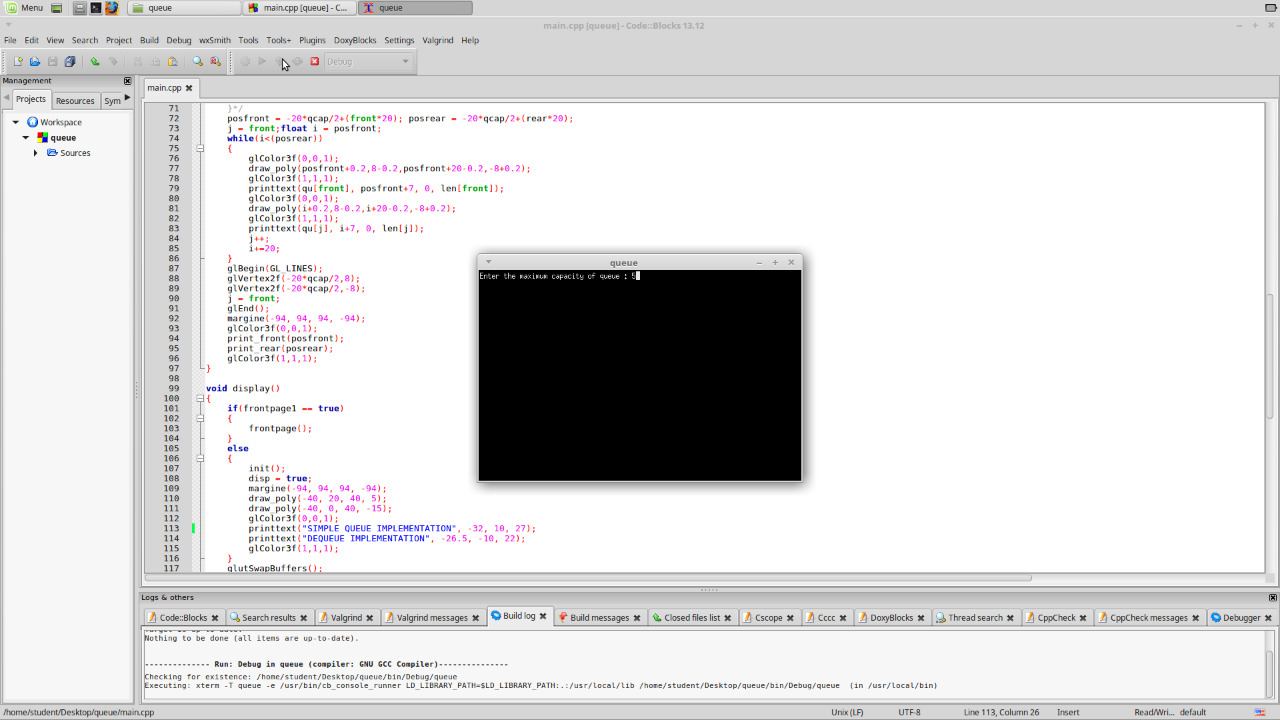


Fig1: length of the queue

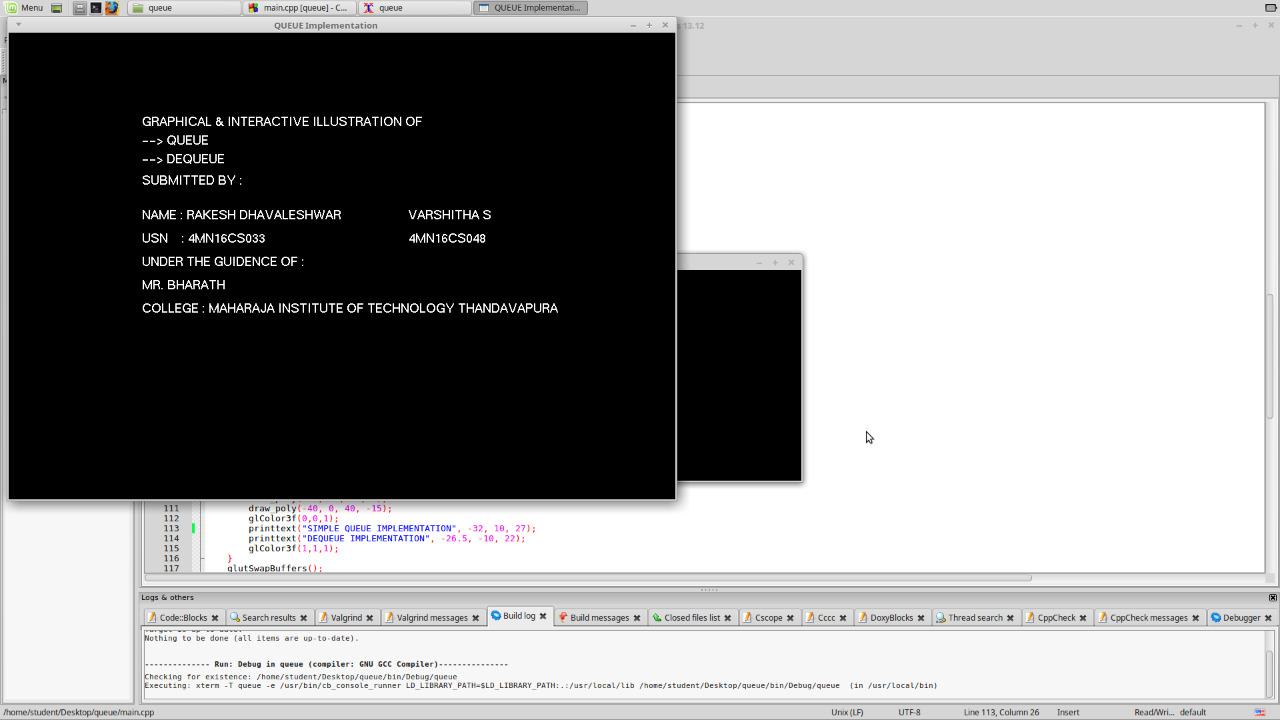


Fig2: Index

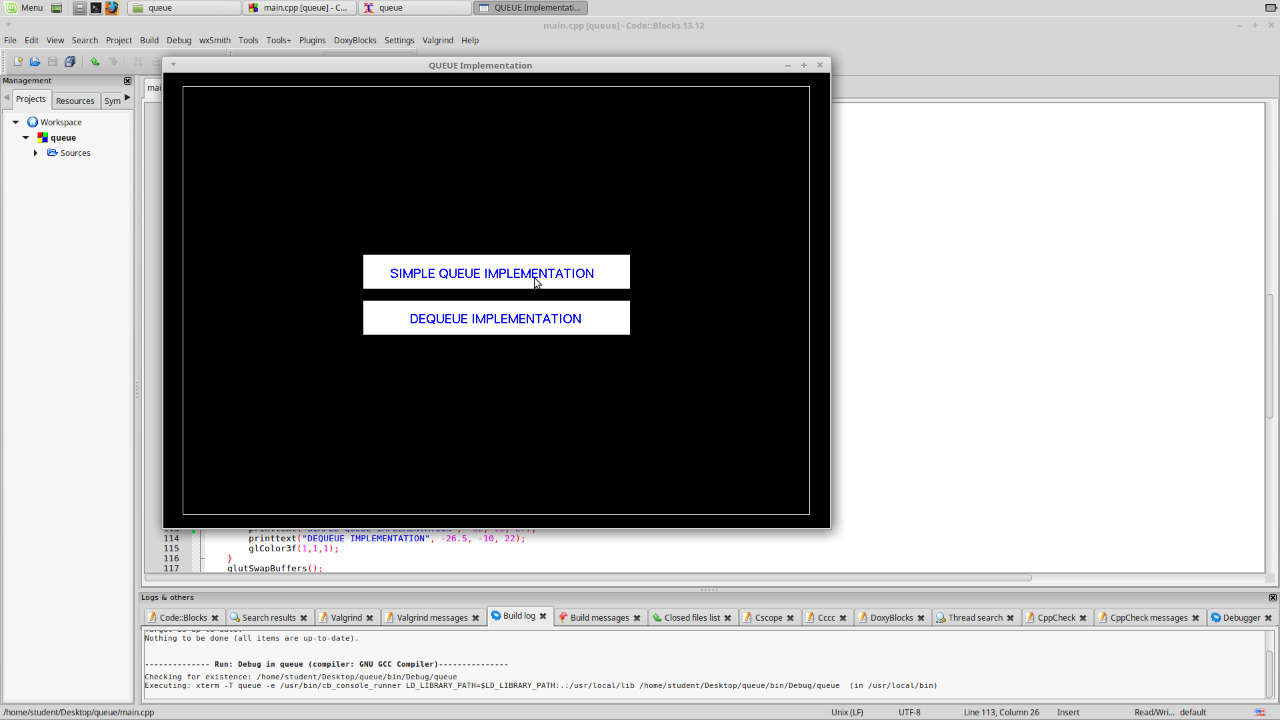
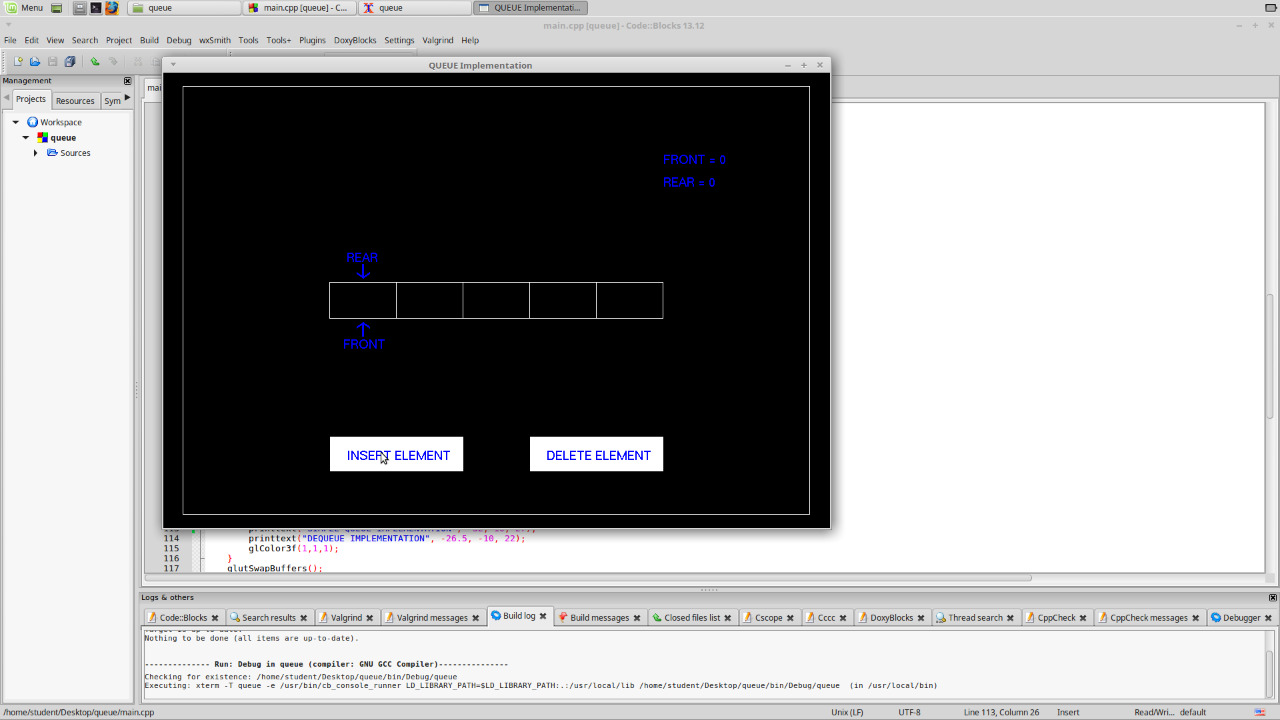


Fig3: Menu

Fig4: Queue display



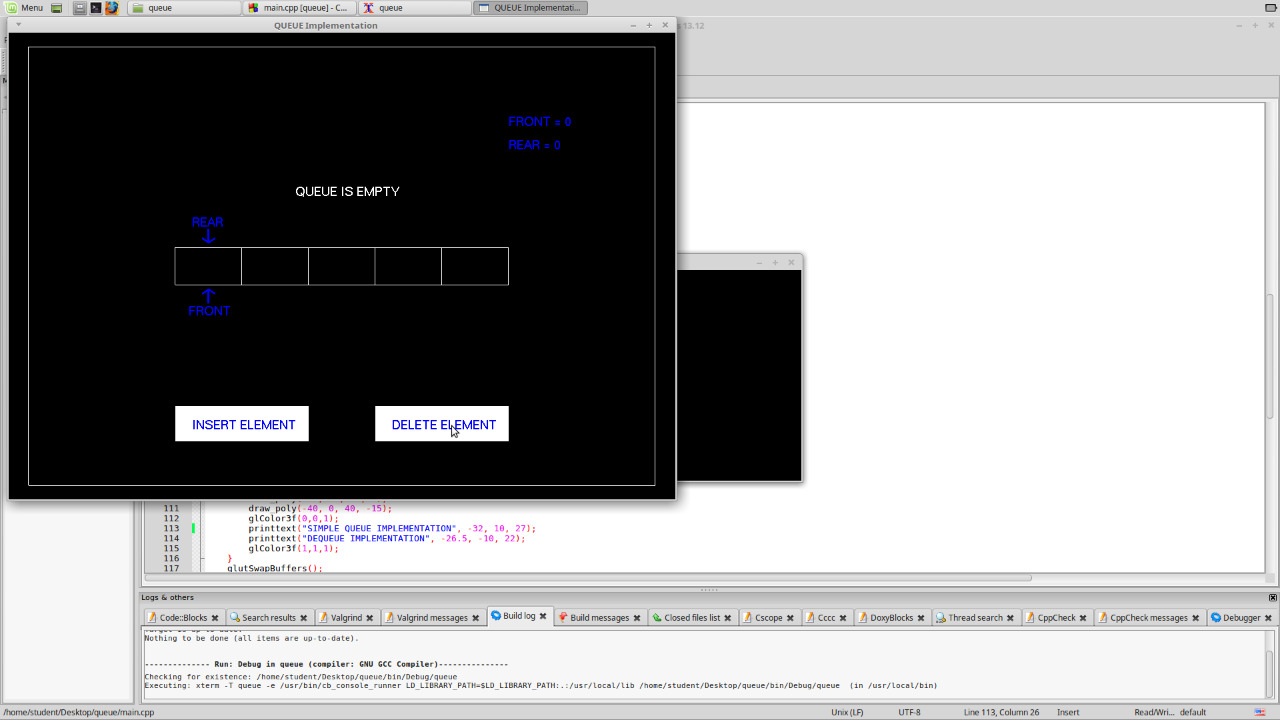
****

Fig5: Queue empty condition

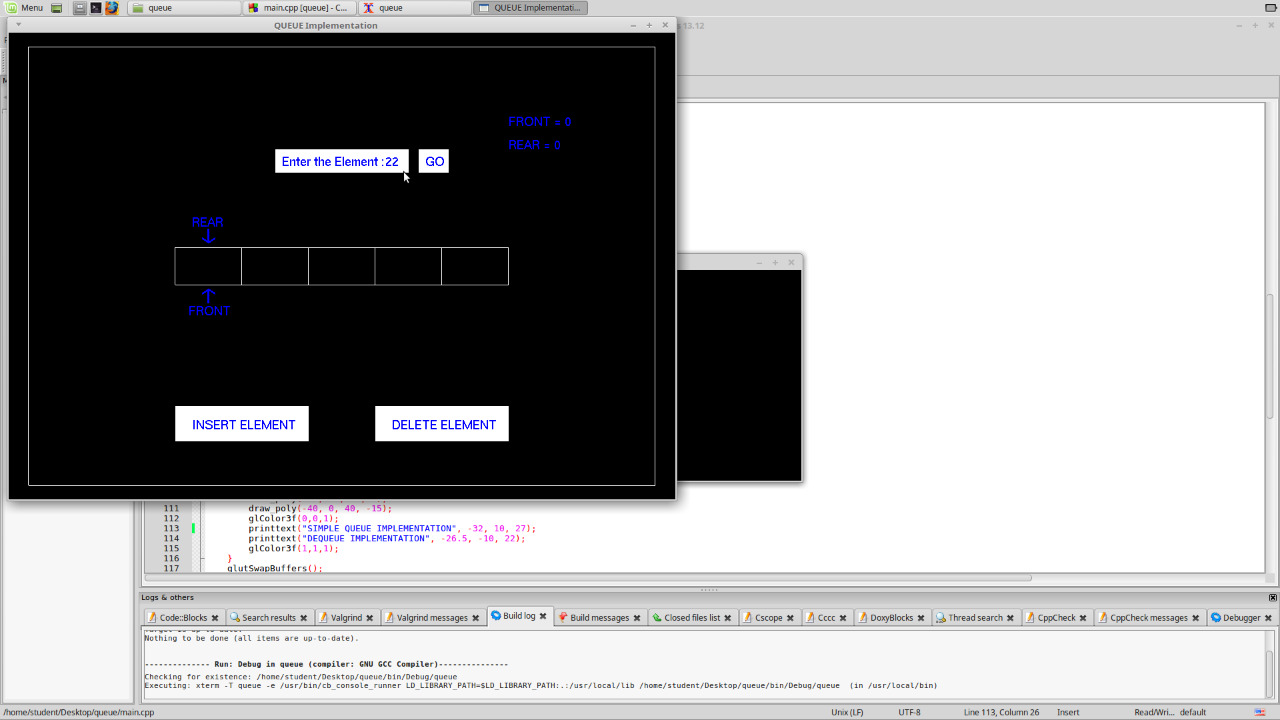


Fig6: Insert operation

**|**

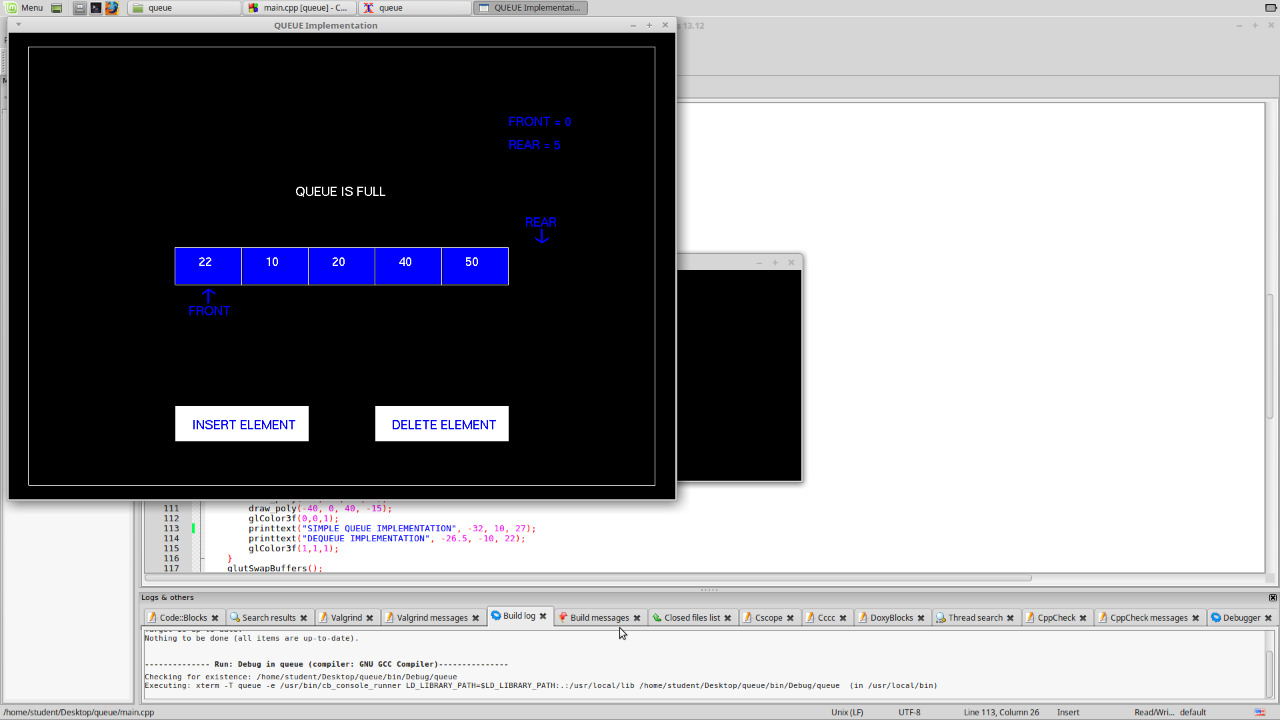


Fig7: Queue full condition

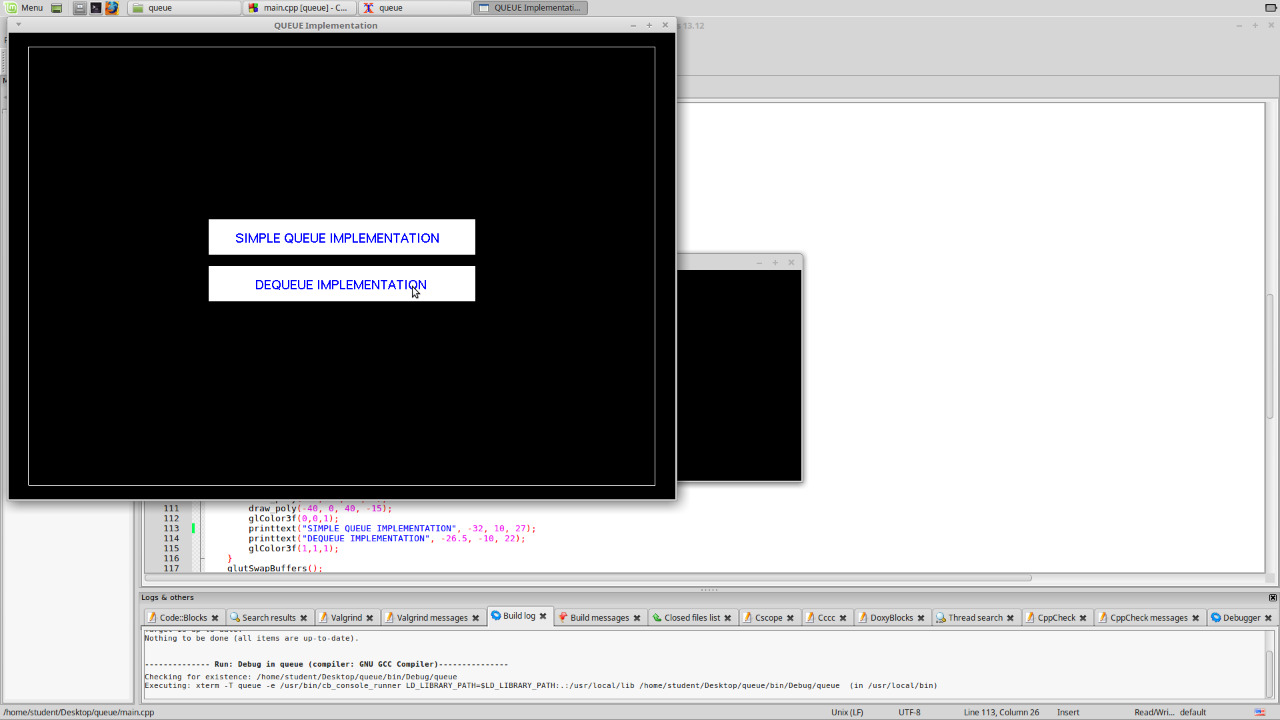


Fig8: Menu

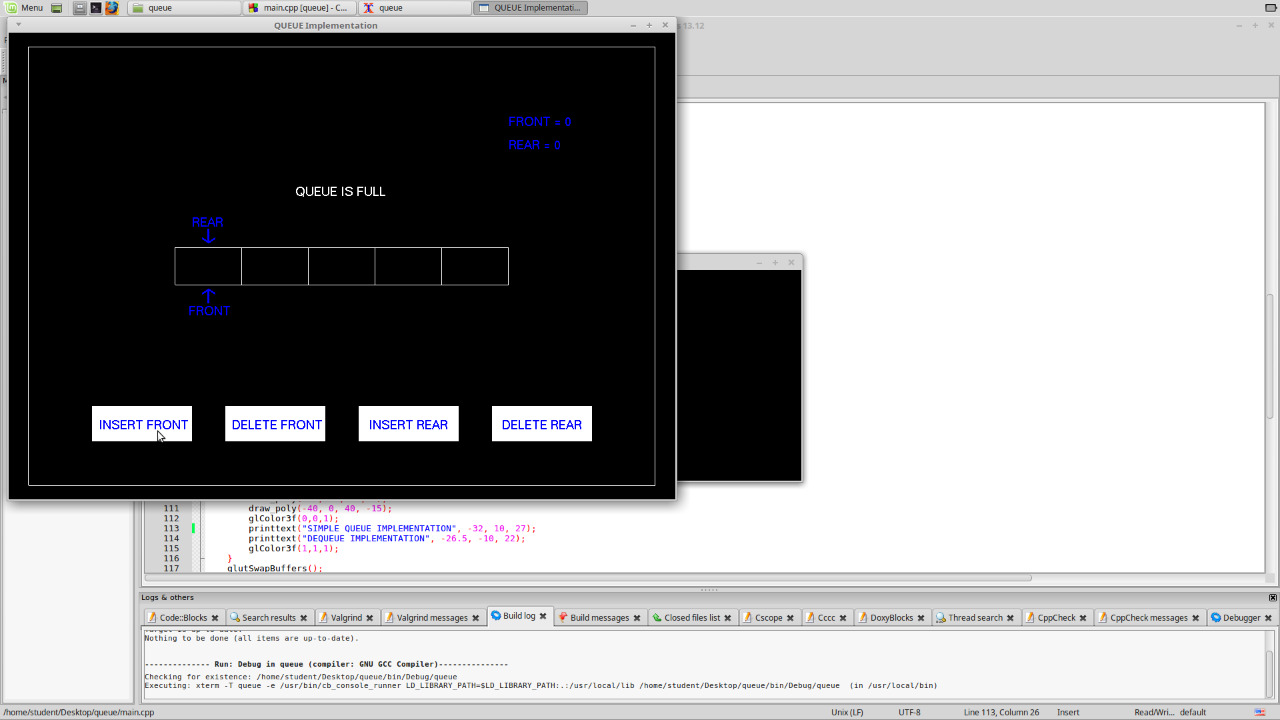


Fig9: De-queue while insert front.

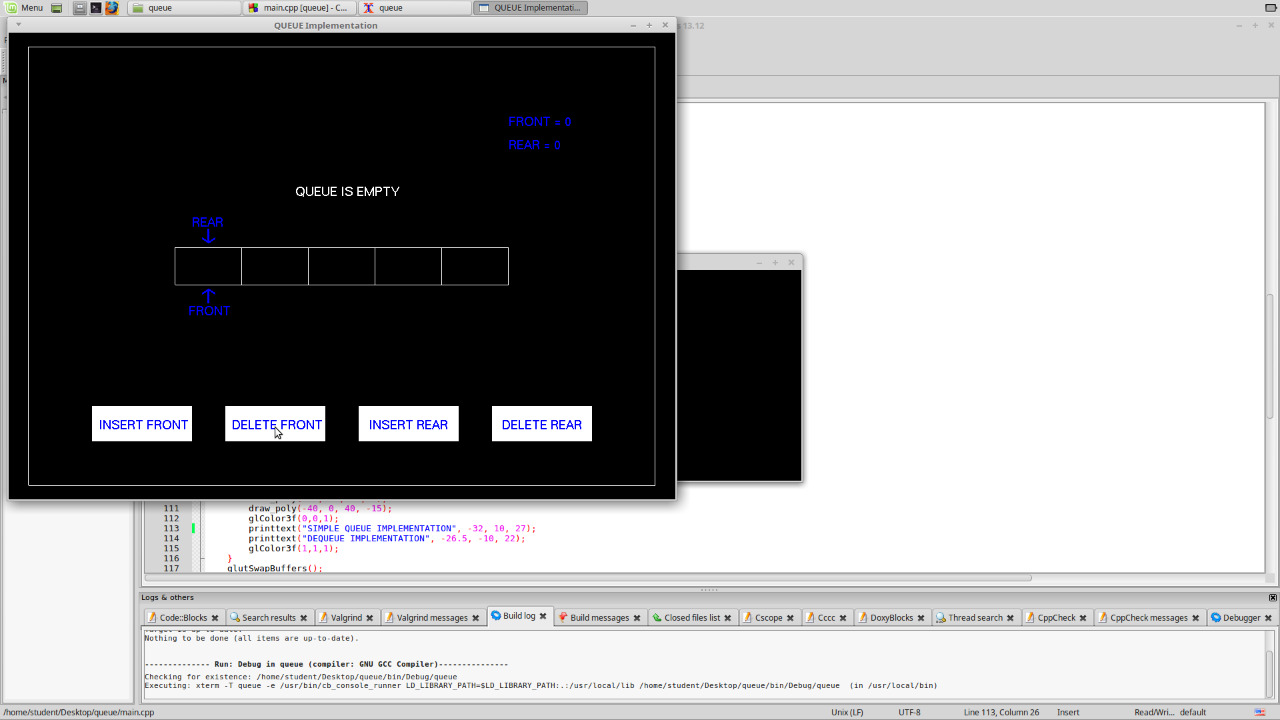
****

Fig10: De-queue while delete front.

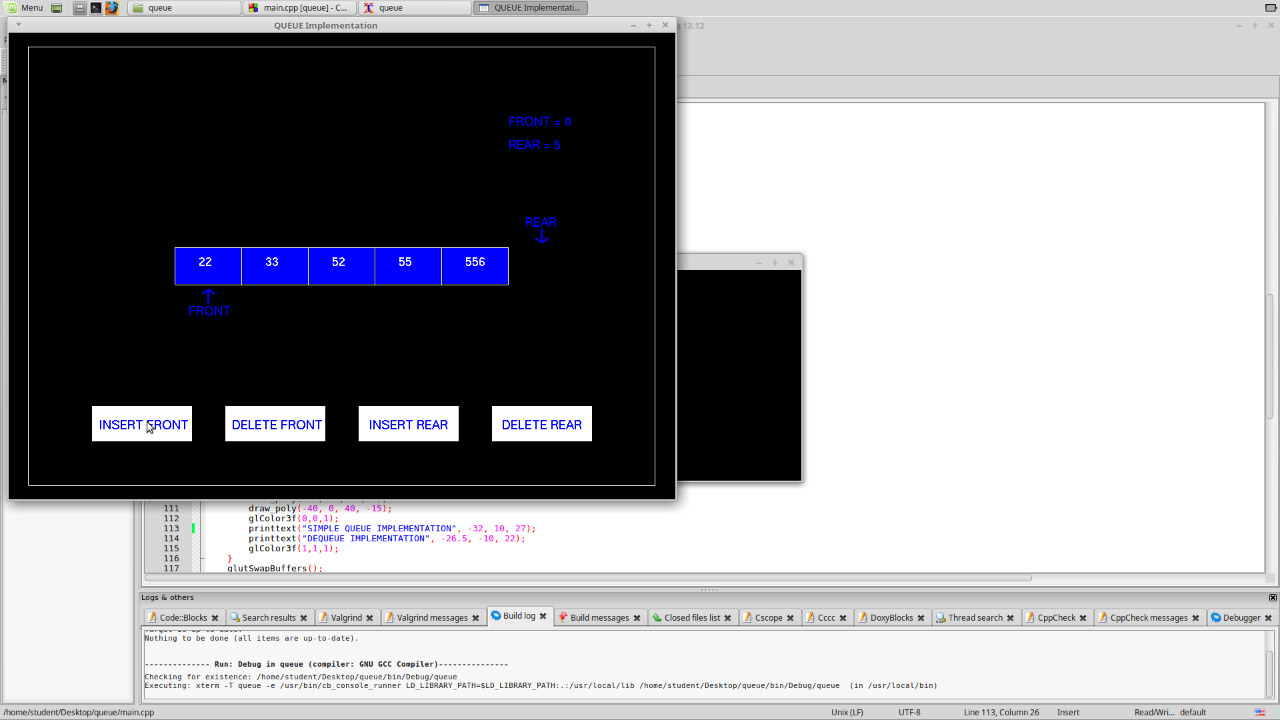
****

Fig11: De-queue after insert front.

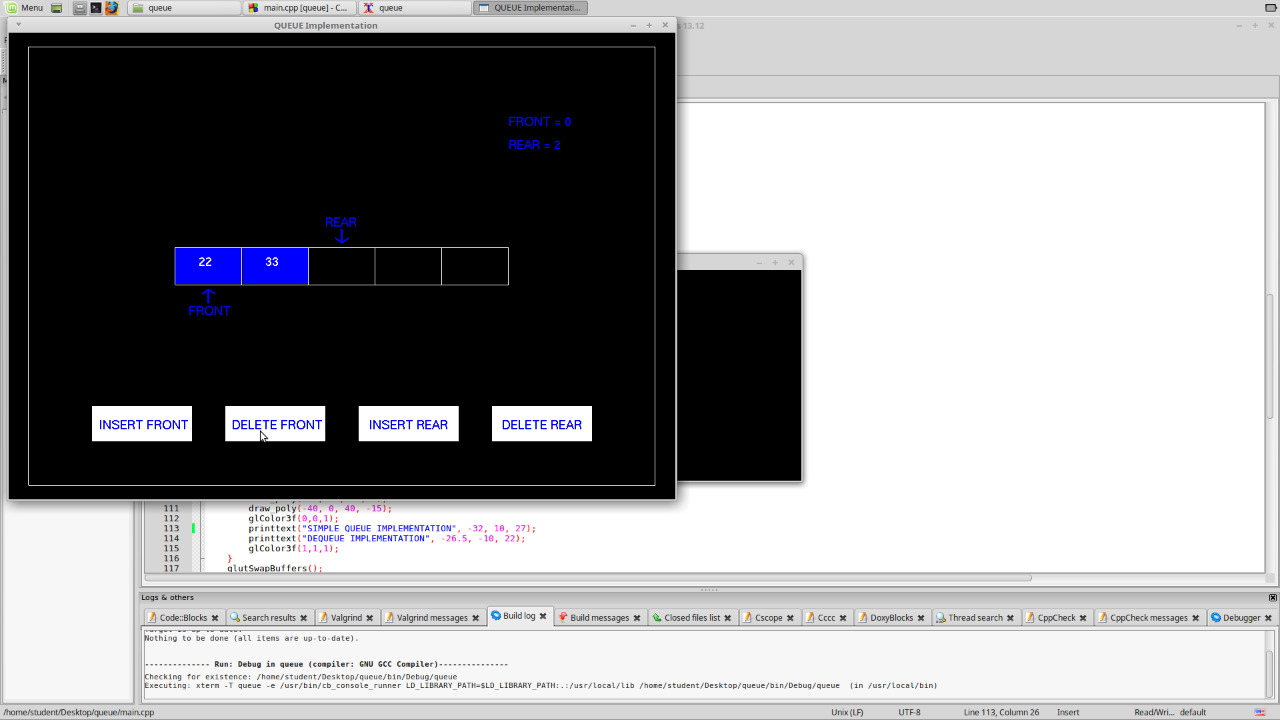


Fig12: De-queue after delete front.

**3.2 discussion**

On running the project, we’ll get an introduction page which will give the information of students and guide. On this page press 1 to continue and give the values for number of processes, arrival time of the process, burst time of the process and time quantum for the processes.

After giving the values the page will be redirected to the menu page which shows the table of values entered and the operations to be performed.

There are two options namely simple queue & double ended queue.  
SimpleQueue:  
 A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first. The difference between [stacks](https://www.geeksforgeeks.org/stack-data-structure/)and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

De-Queue:  
 Deque or Double Ended Queue is a generalized version of Queue data structure that allows insert and delete at both ends.

Operations on Deque:

* insertFront(): Adds an item at the front of Deque.
* insertLast(): Adds an item at the rear of Deque.
* deleteFront(): Deletes an item from front of Deque.
* deleteLast(): Deletes an item from rear of Deque.

**CHAPTER-4**

**CONCLUSION AND FUTURE WORK**

**4.1 conclusion**

Using OpenGL, the operations of queue will be implemented like insert operation, delete operation, multi-insert and multi-delete operations are represented graphically and This project shows a graphical view of simple &de-queue for better understanding it holds a best approach of graphics programming Open GL supports enormous flexibility in the design and the use of OpenGL graphics program.

This representation of stack operation is user friendly and it will help to the beginners to understand the more easily and graphical representation approach is very useful for learning a concept.

**4.2 future enhancement**

As we all are aware of de-queue disadvantage in future we plan to make a working opengl model for circular queue which would help the future generation to understand closely the differences between simple, de-queue & circular queue.

As we end this stage of the project, we would like to recommend other researches in this

field or related to:

* Add visual effects to the program which will get more attention for the user when

Implementing an algorithm.

* Add sound effects to the program to make it more attractive.
* Use different types of queues algorithms like priority queue and others and add it to this program.

**Appendix:**

Singlequeue

void insert()

{

if(rear == qcap)

{

printtext("QUEUE IS FULL", -14, 30, 13);

ins = false;

}

else

{

draw\_poly(-20, 50, 20, 40);

glColor3f(0,0,1);

printtext("Enter the Element : ", -18, 43, 20);

glColor3f(1,1,1);

draw\_poly(23, 50, 32, 40);

glColor3f(0,0,1);

printtext("GO", 25, 43, 2);

printtext(inp, 13, 43, linp);

glColor3f(0,0,1);

if(go == true)

{

glColor3f(0,0,0);

draw\_poly(-20, 50, 32, 40);

glColor3f(0,0,1);

draw\_poly(p, q, r, s);

glColor3f(1,1,1);

printtext(inp, p+6, q-8, linp);

glColor3f(0,0,1);

if(instime == false)

{

instime = true;

timer1(0);

}

if(r <= (20+posrear))

{

strcpy(qu[rear++],inp);

len[rear-1] = linp;

linp = 0;

strcpy(inp,"");

instime = false;

go = false;

ins = false;

p = 100;

r = 120;

glutPostRedisplay();

}

}

}

glColor3f(1,1,1);

}

void delet()

{

if(front == rear)

{

printtext("QUEUE IS EMPTY", -14, 30, 14);

del = false;

}

else

{

glColor3f(0,0,0);

draw\_poly(posfront,8,posfront+20-0.2,-8+0.2);

glColor3f(0,0,1);

draw\_poly(p, q, r, s);

glColor3f(1,1,1);

printtext(qu[front], p+6, q-8, len[front]);

glColor3f(0,0,1);

if(p == 100)

{

p = posfront;

r = p+20;

}

if(deltime == false)

{

deltime = true;

timer2(0);

}

if(r <= -100)

{

front++;

deltime = false;

del = false;

p = 100;

r = 120;

glutPostRedisplay();

}

}

if(front == qcap && rear == qcap)

front = rear = 0;

glColor3f(1,1,1);

}

Double queue

void insertrear()

{

if(rear == qcap)

{

printtext("QUEUE IS FULL", -14, 30, 13);

inr = false;

}

else

{

draw\_poly(-20, 50, 20, 40);

glColor3f(0,0,1);

printtext("Enter the Element : ", -18, 43, 20);

glColor3f(1,1,1);

draw\_poly(23, 50, 32, 40);

glColor3f(0,0,1);

printtext("GO", 25, 43, 2);

printtext(inp, 13, 43, linp);

glColor3f(0,0,1);

if(go == true)

{

glColor3f(0,0,0);

draw\_poly(-20, 50, 32, 40);

glColor3f(0,0,1);

draw\_poly(p, q, r, s);

glColor3f(1,1,1);

printtext(inp, p+6, q-8, linp);

glColor3f(0,0,1);

if(instime == false)

{

instime = true;

timer1(0);

}

if(r <= (20+posrear))

{

strcpy(qu[rear++],inp);

len[rear-1] = linp;

linp = 0;

strcpy(inp,"");

instime = false;

go = false;

inr = false;

p = 100;

r = 120;

glutPostRedisplay();

}

}

}

glColor3f(1,1,1);

}

void deleterear()

{

if(front == rear)

{

printtext("QUEUE IS EMPTY", -14, 30, 14);

dr = false;

}

else

{

glColor3f(0,0,0);

draw\_poly(posrear-20,8,posrear-0.2,-8+0.2);

glColor3f(0,0,1);

draw\_poly(p, q, r, s);

glColor3f(1,1,1);

printtext(qu[rear-1], p+6, q-8, len[rear-1]);

glColor3f(0,0,1);

if(p == 100)

{

p = posrear-20;

r = p+20;

}

if(deltime == false)

{

deltime = true;

timer3(0);

}

if(r >= 119)

{

rear--;

deltime = false;

dr = false;

p = 100;

r = 120;

glutPostRedisplay();

}

}

glColor3f(1,1,1);

}

void insertfront()

{

if(front == 0)

{

printtext("QUEUE IS FULL", -14, 30, 13);

inf = false;

}

else

{

draw\_poly(-20, 50, 20, 40);

glColor3f(0,0,1);

printtext("Enter the Element : ", -18, 43, 20);

glColor3f(1,1,1);

draw\_poly(23, 50, 32, 40);

glColor3f(0,0,1);

printtext("GO", 25, 43, 2);

printtext(inp, 13, 43, linp);

glColor3f(0,0,1);

if(go == true)

{

glColor3f(0,0,0);

draw\_poly(-20, 50, 32, 40);

glColor3f(0,0,1);

if(p == 100)

{

p = -120;

r = p+20;

}

draw\_poly(p, q, r, s);

glColor3f(1,1,1);

printtext(inp, p+6, q-8, linp);

glColor3f(0,0,1);

if(instime == false)

{

instime = true;

timer4(0);

}

if(r >= (posfront))

{

strcpy(qu[front-1],inp);

len[front-1] = linp;

front--;

linp = 0;

strcpy(inp,"");

instime = false;

go = false;

inf = false;

p = 100;

r = 120;

glutPostRedisplay();

}

}

glColor3f(1,1,1);

}

}

void deletefront()

{

if(front == rear)

{

printtext("QUEUE IS EMPTY", -14, 30, 14);

df = false;

}

else

{

glColor3f(0,0,0);

draw\_poly(posfront,8,posfront+20-0.2,-8+0.2);

glColor3f(0,0,1);

draw\_poly(p, q, r, s);

glColor3f(1,1,1);

printtext(qu[front], p+6, q-8, len[front]);

glColor3f(0,0,1);

if(p == 100)

{

p = posfront;

r = p+20;

}

if(deltime == false)

{

deltime = true;

timer2(0);

}

if(r <= -100)

{

front++;

deltime = false;

df = false;

p = 100;

r = 120;

glutPostRedisplay();

}

}

if(front == qcap && rear == qcap)

front = rear = 0;

glColor3f(1,1,1);

}

Queue

#include</home/student/Desktop/project cg/req.cpp>

#include</home/student/Desktop/project cg/singlequeue.cpp>

#include</home/student/Desktop/project cg/doublequeue.cpp>

int main(int argc, char\*\*argv)

{

cout<<"Enter the maximum capacity of queue : ";

cin>>qcap;

glutInit(&argc,argv);

glutInitDisplayMode( GLUT\_RGBA | GLUT\_DOUBLE );

glutInitWindowPosition(10,10);

glutInitWindowSize(1000,700);

glutCreateWindow("QUEUE Implementation");

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutKeyboardFunc(keyboard);

glutMouseFunc(mouse);

glutMainLoop();

}

void queueinit()

{

init();

glBegin(GL\_LINES);

glVertex2f(0-20\*qcap/2, 8);

glVertex2f(0+20\*qcap/2, 8);

glVertex2f(0-20\*qcap/2, -8);

glVertex2f(0+20\*qcap/2, -8);

for(int i=0-20\*qcap/2;i<=0+20\*qcap/2;i+=20)

{

glVertex2f(i, 8);

glVertex2f(i, -8);

}

/\*if(front!=rear)

{

draw\_poly(posfront,8,posfront+20,-8);

glColor3f(1,1,1);

printtext(qu[j], posfront+6, 0, len[j]);

j++;

}

//int

for(int i=posfront;i<posrear;i+=20,j++)

{

glColor3f(0,0,1);

draw\_poly(i,8,i+20-0.2,-8+0.2);

glColor3f(1,1,1);

printtext(qu[j], i+6, 0, len[j]);

}\*/

posfront = -20\*qcap/2+(front\*20); posrear = -20\*qcap/2+(rear\*20);

j = front;float i = posfront;

while(i<(posrear))

{

glColor3f(0,0,1);

draw\_poly(posfront,8,posfront+20-0.2,-8+0.2);

glColor3f(1,1,1);

printtext(qu[front], posfront+7, 0, len[front]);

glColor3f(0,0,1);

draw\_poly(i,8,i+20-0.2,-8+0.2);

glColor3f(1,1,1);

printtext(qu[j], i+7, 0, len[j]);

j++;

i+=20;

}

glBegin(GL\_LINES);

glVertex2f(-20\*qcap/2,8);

glVertex2f(-20\*qcap/2,-8);

j = front;

glEnd();

margine(-94, 94, 94, -94);

glColor3f(0,0,1);

print\_front(posfront);

print\_rear(posrear);

glColor3f(1,1,1);

}

void display()

{

disp = true;

margine(-94, 94, 94, -94);

draw\_poly(-40, 20, 40, 5);

draw\_poly(-40, 0, 40, -15);

glColor3f(0,0,1);

printtext("NORMAL QUEUE IMPLEMENTATION", -32, 10, 27);

printtext("DEQUEUE IMPLEMENTATION", -26.5, -10, 22);

glColor3f(1,1,1);

glutSwapBuffers();

}

void queue()

{

queueinit();

draw\_poly(-50, -60, -10, -75);

draw\_poly(10, -60, 50, -75);

glColor3f(0,0,1);

printtext("INSERT ELEMENT", -45, -70, 14);

printtext("DELETE ELEMENT", 15, -70, 14);

glColor3f(1,1,1);

if(ins == true)

{

insert();

}

else if(del == true)

{

delet();

}

glutSwapBuffers();

}

void dequeue()

{

queueinit();

draw\_poly(-75, -60, -45, -75);

draw\_poly(-35, -60, -5, -75);

draw\_poly(5, -60, 35, -75);

draw\_poly(45, -60, 75, -75);

glColor3f(0,0,1);

printtext("INSERT FRONT", -73, -70, 12);

printtext("DELETE FRONT", -33, -70, 12);

printtext("INSERT REAR", 8, -70, 12);

printtext("DELETE REAR", 48, -70, 12);

glColor3f(1,1,1);

if(inr == true)

{

insertrear();

}

else if(dr == true)

{

deleterear();

}

else if(inf == true)

{

insertfront();

}

else if(df == true)

{

deletefront();

}

glutSwapBuffers();

}

Req

#include<GL/gl.h>

#include<GL/glu.h>

#include<GL/glut.h>

#include<math.h>

#include<cmath>

#include<iostream>

#include<string.h>

using namespace std;

int qcap, kkey = 32, front = 0, rear = 0, linp = 0, len[50], j = 0;

bool disp = false, que = false, dque = false, ins = false, del = false, go = false, instime = false, deltime = false, inr = false, inf = false, dr = false, df = false;

float x, y, p = 100, q = 8, r = 120, s = -8, a = -20, b = 1000, c = 19.65, d, posfront, posrear;

char inp[100], qu[100][100];

void queue();

void dequeue();

void display();

void insertrear();

void insertfront();

void deletefront();

void deleterear();

void init()

{

glClearColor(0,0,0,1);

glClear(GL\_COLOR\_BUFFER\_BIT);

}

void margine(float p, float q, float r, float s)

{

glBegin(GL\_LINE\_LOOP);

glVertex2f(p, q);

glVertex2f(p, s);

glVertex2f(r, s);

glVertex2f(r, q);

glEnd();

}

void reshape(int w, int h)

{

glViewport(0,0,w,h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-100,100,-100,100);

glMatrixMode(GL\_MODELVIEW);

}

void draw\_poly(float a, float b, float c, float d)

{

glBegin(GL\_POLYGON);

glVertex2f(a,b);

glVertex2f(a,d);

glVertex2f(c,d);

glVertex2f(c,b);

glEnd();

}

void mouse(int button, int state, int mousex, int mousey)

{

if(button==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN)

{

x = (float(mousex) - 500)/5;

y = (700 - float(mousey) - 350)/3.5;

}

if(disp == true)

{

if(x >= -40 && x <= 40 && y >= 5 && y <= 20)

{

disp = false;

que = true;

glutDisplayFunc(queue);

}

else if(x >= -40 && x <= 40 && y >= -15 && y <= 0)

{

disp = false;

dque = true;

glutDisplayFunc(dequeue);

}

}

if(que == true)

{

if(x >= -50 && x <= -10 && y >= -75 && y <= -60)

{

ins = true;

}

else if(x >= 10 && x <= 50 && y >= -75 && y <= -60)

{

del = true;

}

else if(ins == true)

{

if(x >=23 && x <= 32 && y >= 40 && y <= 50)

{

go = true;

}

}

}

if(dque == true)

{

draw\_poly(-75, -60, -45, -75);//if

draw\_poly(-35, -60, -5, -75);//df

draw\_poly(5, -60, 35, -75);//ir

draw\_poly(45, -60, 75, -75);//dr

if(x >= -75 && x <= -45 && y >= -75 && y <= -60)

{

inf = true;

}

else if(x >= -35 && x <= -5 && y >= -75 && y <= -60)

{

df = true;

}

else if(x >= 5 && x <= 35 && y >= -75 && y <= -60)

{

inr = true;

}

else if(x >= 45 && x <= 75 && y >= -75 && y <= -60)

{

dr = true;

}

else if(inr == true || inf == true)

{

if(x >=23 && x <= 32 && y >= 40 && y <= 50)

{

go = true;

}

}

}

glutPostRedisplay();

}

void keyboard(unsigned char key, int x, int y)

{

kkey = (int)key;

if(que == true || dque == true)

{

if(ins == true || inr == true || inf == true)

{

if(isdigit(key) || key == '-')

{

inp[linp] = char(kkey);

linp++;

}

else if(kkey == 8)

{

inp[--linp] = '\0';

}

}

}

glutPostRedisplay();

}

void printtext(char s[], float x, float y, int l)

{

glRasterPos2i(x,y);

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

{

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, s[i]);

}

}

void print\_front(int x)

{

glLineWidth(3);

glBegin(GL\_LINES);

glVertex2f(x+10, -10);

glVertex2f(x+10, -16);

glVertex2f(x+10, -10);

glVertex2f(x+8,-12);

glVertex2f(x+10, -10);

glVertex2f(x+12,-12);

glEnd();

glLineWidth(1);

printtext("FRONT", x+4, -21, 5);

printtext("FRONT = ", 50, 60, 8);

if(front == -1)

{

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, 45);

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, 49);

}

else

{

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, front + 48);

}

}

void print\_rear(int x)

{

glLineWidth(3);

glBegin(GL\_LINES);

glVertex2f(x+10, 10);

glVertex2f(x+10, 16);

glVertex2f(x+10, 10);

glVertex2f(x+8,12);

glVertex2f(x+10, 10);

glVertex2f(x+12,12);

glEnd();

glLineWidth(1);

printtext("REAR", x+5, 17, 4);

printtext("REAR = ", 50, 50, 7);

if(rear == -1)

{

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, 45);

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, 49);

}

else

{

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, rear + 48);

}

}

void timer1(int)

{

if(instime == true)

{

glutPostRedisplay();

glutTimerFunc(1000/60,timer1,0);

if(r > (20+posrear))

{

r-=1;

p = r-20;

}

else

{

instime = false;

return;

}

}

}

void timer2(int)

{

if(deltime == true)

{

glutPostRedisplay();

glutTimerFunc(1000/60,timer2,0);

if(r > -100)

{

r-=1;

p = r-20;

}

else

{

deltime = false;

return;

}

}

}

void timer3(int)

{

if(deltime == true)

{

glutPostRedisplay();

glutTimerFunc(1000/60,timer3,0);

if(r < 119)

{

r+=1;

p = r-20;

}

else

{

deltime = false;

return;

}

}

}

void timer4(int)

{

if(instime == true)

{

glutPostRedisplay();

glutTimerFunc(1000/60,timer4,0);

if(r < (posfront))

{

r+=1;

p = r-20;

}

else

{

instime = false;

return;

}

}

}