---------------------------------------------------------------------- Design and simulate any data structure like stack, queue, and trees using graphics. Simulation should include all operations performed on designed data structure. Implement the same using OpenGL.

----------------------------------------------------------------------

#include<GL/glut.h>

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

void \*currentfont;

#define size 5

#define MAX 10

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\_HELVETICA\_18);

glColor3f(1.0,1.0,1.0);

drawstring(x1+10,y1+10,str);

glColor3f(0.2,0.2,0.8);

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];

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];

snprintf(str, sizeof(str), "%d", num++);

button btn(300,250+rear\*50,350,300+rear\*50,str);

if(front==0 && rear==MAX-1)

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==-1)

{

drawstring(300,10," Underflow ");

}

element=que[front];

if(front==rear)

front=rear=-1;

else

{

if(front==MAX-1)

front=0;

else

front++;

}

}

void queue::displayqueue()

{

int i;

if(front==-1)

drawstring(300,10," 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(1.0,1.0,1.0,0.0);

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("STACK AND QUEUE");

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutIdleFunc(idle);

glEnable(GL\_DEPTH\_TEST);

init();

glutMainLoop();

return 0;

}



