

**Name**: Jakshat Desai

**Admission number**: 16JE001854

**Semester**: 3(Monsoon)

**Branch**: B.Tech CSE

**Topic**: Demonstration of working of stacks and queues using graphic user interface

**Index**

1. Brief idea about working of project……………….3
2. Roadblocks faced while making the project………4
3. Installation of graphics.h in CodeBlocks 16.01…...5
4. Various graphics.h commands…………………….7
5. Basic idea behind animation………………………8
6. Stacks……………………………………………...9
7. Stack graphic user interface C++ code……………11
8. Screenshots………………………………………..26
9. Queues…………………………………………….28

10. Queue graphic user interface C++ code…………30

11. Screenshots……………………………………....50

12.Working of the program………………………….52

13.Resources used…………………………………...54

**Brief idea about working of the project**

The project:

* Has been made in C++.
* Uses animation to describe working of stacks and queues.
* Takes input from the user and produces an animation accordingly.

The codes to this project can be found at:

https://github.com/Jakshat-Desai/College\_projects

**Roadblocks faced while making the project**

The graphics.h library has simple functions which are easy to use and grasp. The issue was that graphics.h being an almost 20 year old library is not available in the latest versions of devcpp, codeblocks or other IDEs. It is only available in TurboC++. While TurboC++ could have been used, its old syntax along with the lack of several new libraries (such as vector) made program writing highly inconvenient. A lot of ways were present online to make graphics.h work in several versions of several different IDEs. However many of them were themselves outdated and many of the rest didn’t work properly. It took quite some time and effort to find a way which actually worked in CodeBlocks 16.01 on windows 10.

**Installation of graphics.h in Codeblocks 16.01**

Step1) Ensure that Codeblocks is installed in your computer.

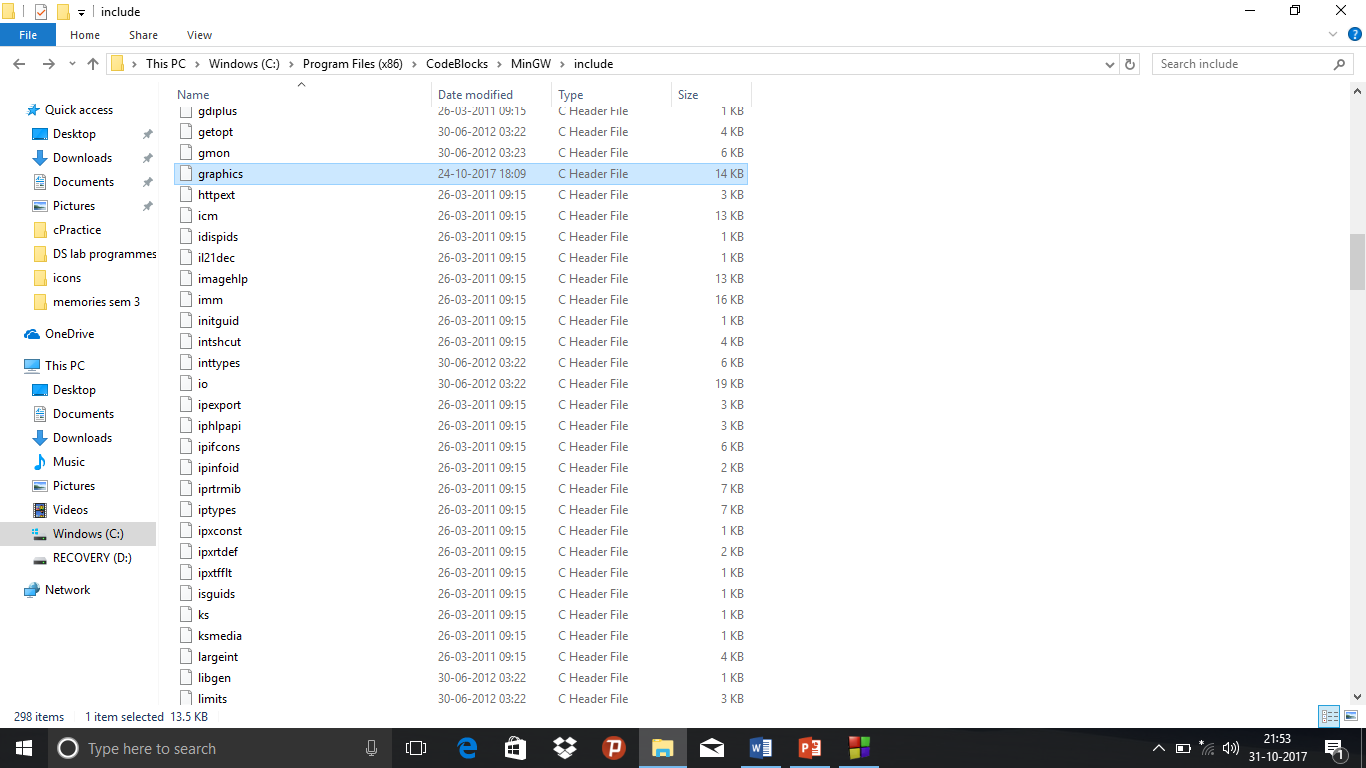
Step2)Download the contents of the following link:

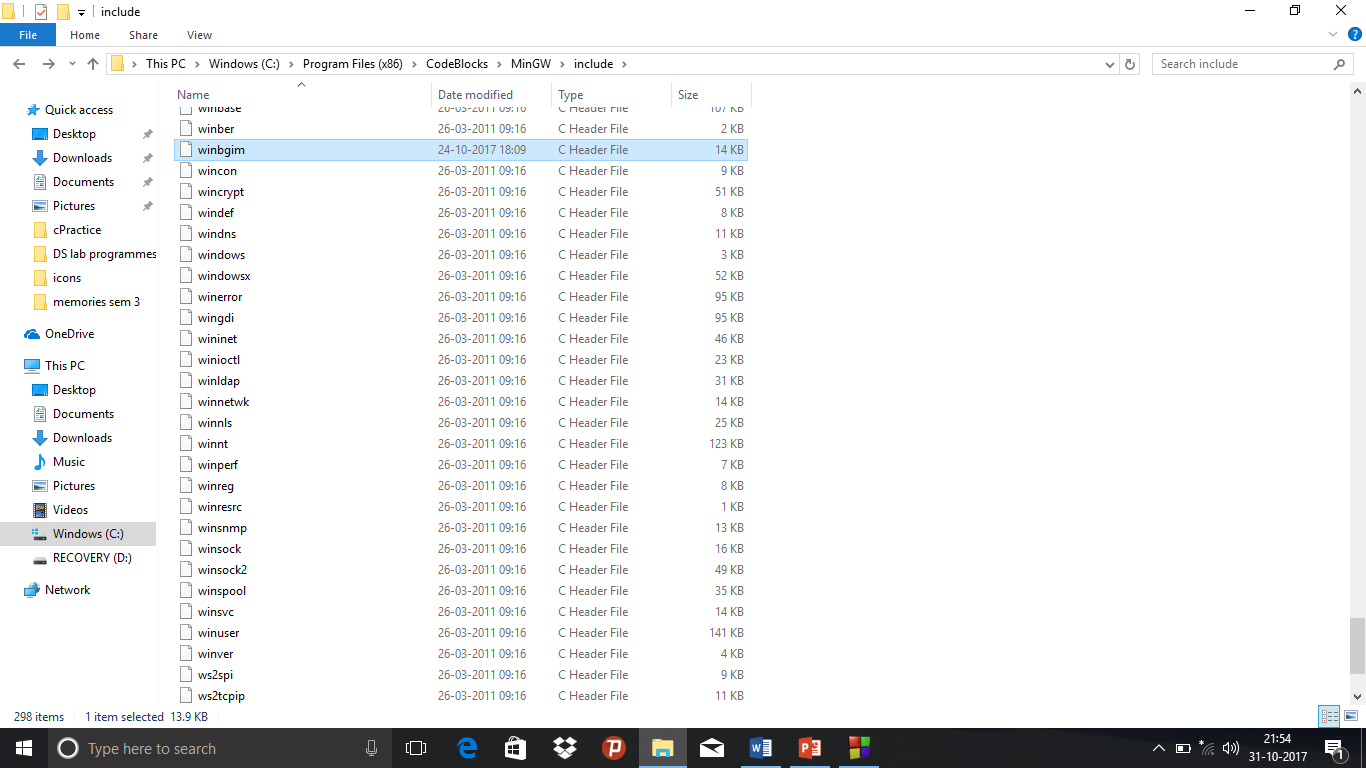
<https://drive.google.com/file/d/0ByyBxK_syBJ-ZXdOMWVsdW1fTGs/view>



Step3)From the list of downloaded items, copy the winbgim.h and graphic.h files to the include folder:

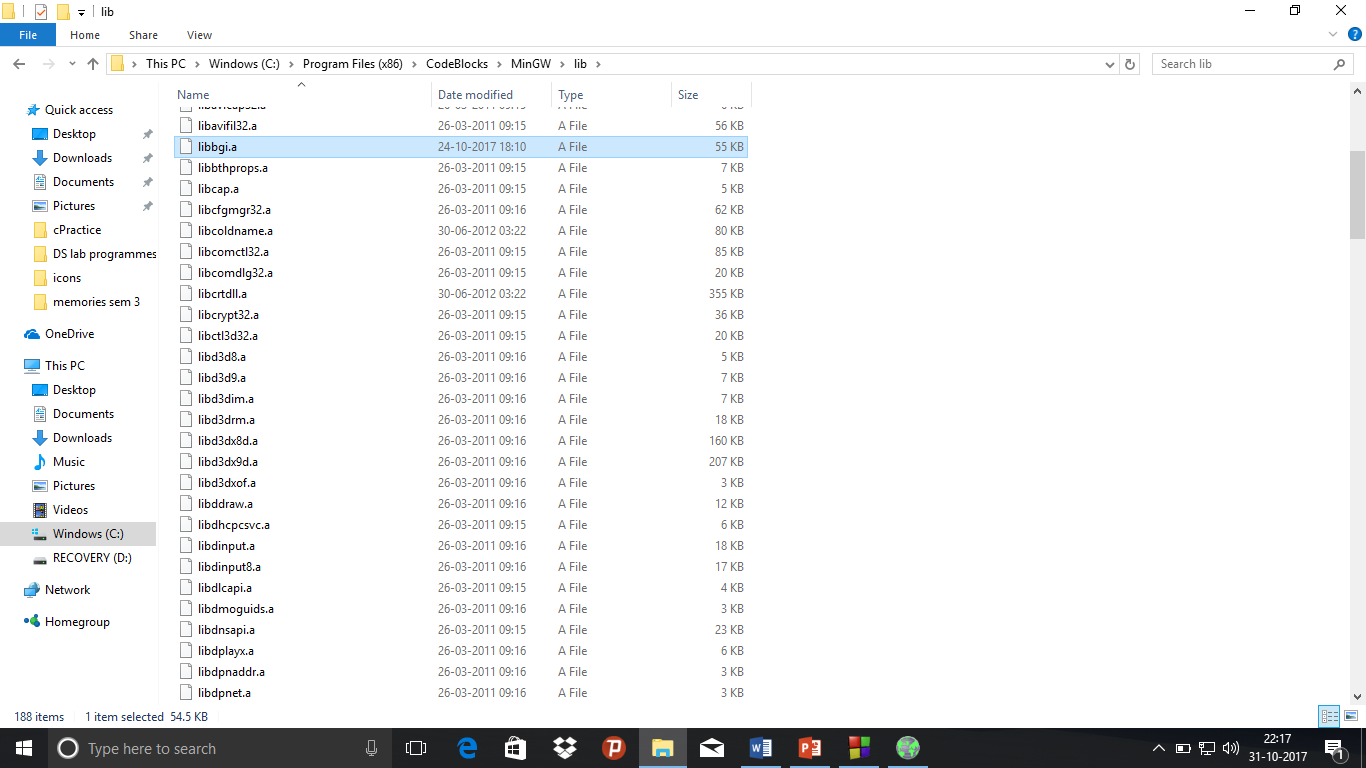
C:\Program Files (x86)\CodeBlocks\MinGW\include



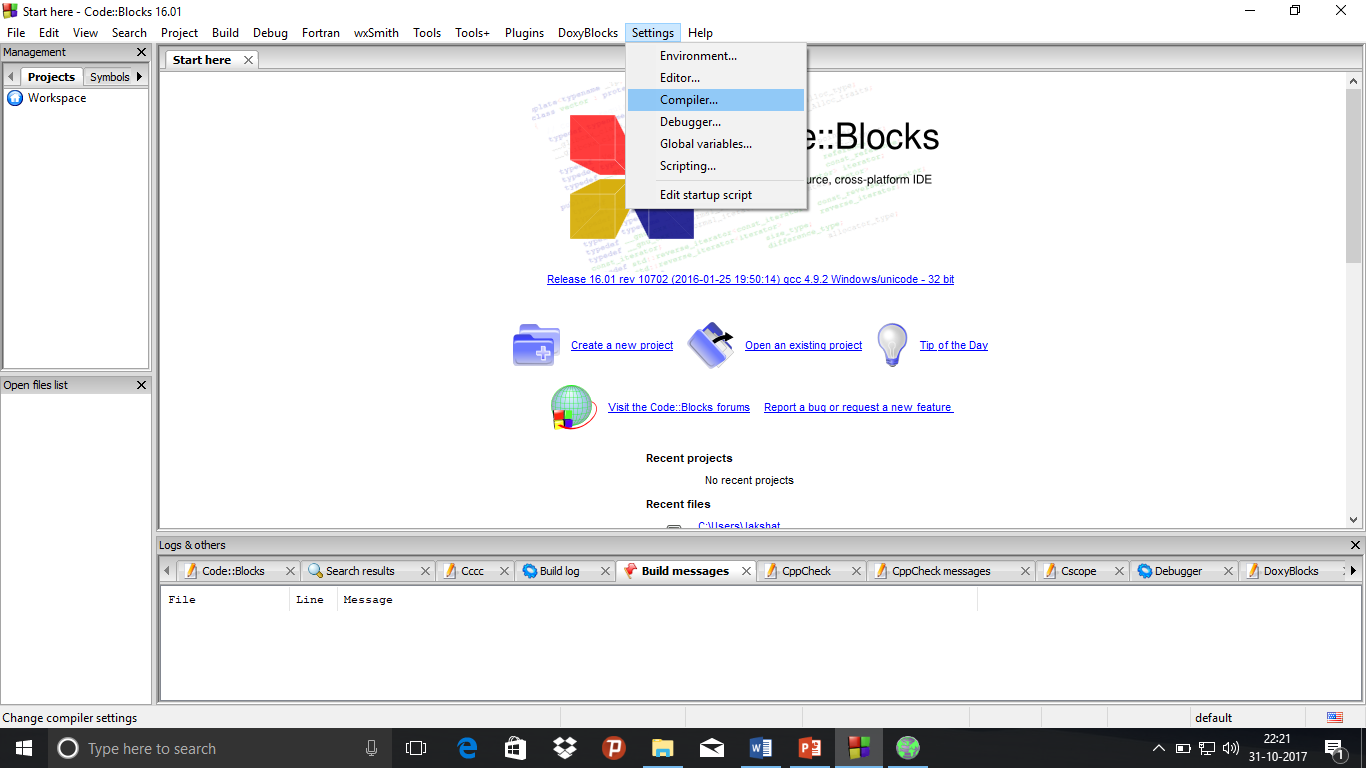


Step4)From the list of downloaded items copy the libbgi.a file to the lib folder:

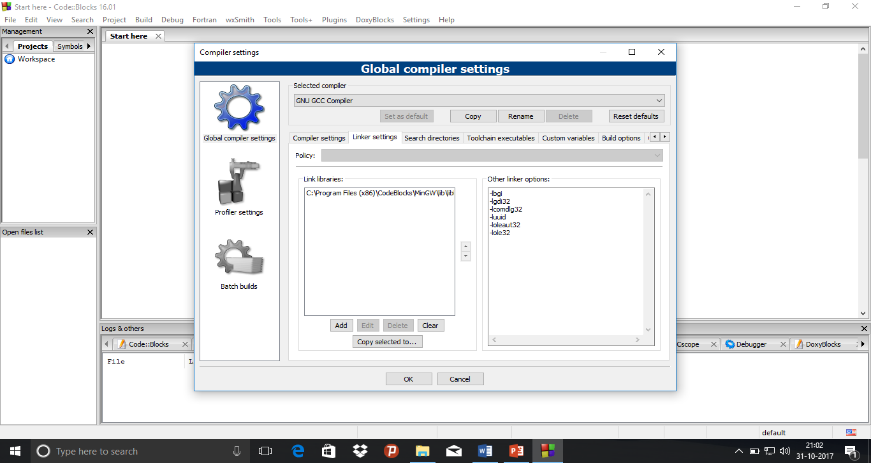
C:\Program Files (x86)\CodeBlocks\MinGW\lib



Step5)Open Codeblocks. Click on the Settings menu and then click on the Compiler option. In the dialog box that appears, click on the linker settings.



Step6)Click on the Add button and write the following address in the space provided:C:\Program Files (x86)\CodeBlocks\MinGW\lib

Step7)In the other linker options space, write the following:

-lbgi

-lgdi32 -lcomdlg32

-luuid

-loleaut32 -lole32

Step8)Click on Ok. Now graphics.h can be used in codeblocks.

**Various graphics.h commands**

a)used for creating the graphics window

* **void initgraph(int far \*graphdriver, int far \*graphmode, char far \*pathtodriver)**

1)It initializes the graphic system and loads the graphic drivers. It then puts the system in graphics mode.

2)\*graphdriver is an integer that specifies which graphics driver to be used.

3)\*graphmode specifies the initial graph mode. If graphmode is set to DETECT then initgraph sets \*graphmode to the highest resolution available for the detected driver.

4)\*pathtodriver specifies the directory path where initgraph looks for graphics drivers.

* **closegraph()** : It closes the graph.

b) used for drawing shapes and animating

* **line(int x1, int y1, int x2, int y2)** : Draws a line from the point (x1,y1) in the coordinate system to the point (x2,y2).
* **bar(int x1, int y1, int x2, int y2)** : Draws a filled rectangle with the opposite vertices as (x1,y1) and (x2,y2).
* **rectangle(int x1, int y1, int x2, int y2)** : Draws an unfilled rectangle with opposite vertices as (x1,y1) and (x2,y2).
* **outtextxy(int x, int y, char \*c)** : Displays the text stored in c at (x,y)
* **setcolor(<colorname>)** : Sets the color which the rest of the objects drawn on the graph appear in.
* **setbkcolor(<colorname>)** : Sets the background color.
* **delay(<time in milliseconds>)** : Causes a delay inexecution by the entered number of milliseconds.
* **cleardevice()** : Clears the graph

**Basic idea behind animation**

* The basic working behind the animation is that we can create an object, erase it and place it at a slightly different position, repeatedly, in a loop so as to make it seem as if the object is moving.
* The speed of the object can be controlled using the delay() function. More the delay, less the speed of the object.

**Stacks**

* Stacks are abstract data types that may or may not be bound. They are homogeneous, linear and ordered data types and are based on the LIFO(Last In First Out) principle.
* They can be implemented through arrays(static implementation) as well as queues(dynamic implementation).
* They are extremely important data structures and have several applications such as evaluation of arithmetic expressions, backtracking, etc.

Basic Stack operations

* **PUSH(<element>)** :

1)This operation inserts an element at the top of the stack.

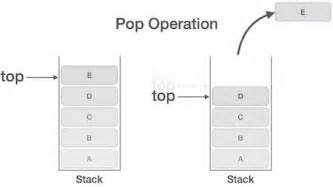
2)When the stack is full(static implementation), that is if no more elements can further be stored in the stack, if insertion is attempted then it is called an overflow condition.



* **POP()**:

1)This operation deletes the element present at the top of the stack.

2)If deletion is attempted when the stack doesn’t have a single element, i.e when the stack is empty, this is said to be an underflow condition.



**Stack Graphic User interface C++ code**

#include<bits/stdc++.h>

#include<graphics.h>

//constant keeping track of y coordinates of the bottom of the stack

#define stckbtm 420

using namespace std;

//function to set the environment

void createStackgrap(int sze)

{

//sets colour for outer box

setcolor(LIGHTBLUE);

//creates outer box

rectangle(250,10,500,450);

//creates bar for ELEMENT BIN

bar(250,10,350,50);

//sets colour for ELEMENT BIN text

setcolor(LIGHTMAGENTA);

//writes the ELEMENT BIN text

outtextxy(255,20,"ELEMENT BIN");

//sets colour for column seperating line

setcolor(LIGHTBLUE);

//column seperating line

line(350,10,350,450);

//line for heading

line(250,50,500,50);

//creating the stack holder

setcolor(LIGHTMAGENTA);

line(280,stckbtm,280,stckbtm-sze\*20);

line(280,stckbtm,320,stckbtm);

line(320,stckbtm,320,stckbtm-sze\*20);

setcolor(MAGENTA);

line(278,stckbtm+2,278,stckbtm-sze\*20);

line(278,stckbtm+2,322,stckbtm+2);

line(322,stckbtm+2,322,stckbtm-sze\*20);

//program description and credits

setcolor(LIGHTGREEN);

outtextxy(10,200,"Stack graphic user interface");

outtextxy(10,210,"animation to explain working");

outtextxy(20,220,"of various stack operations");

outtextxy(40,240,"Created by");

outtextxy(60,250,"-JAKSHAT DESAI");

setcolor(WHITE);

}

//function for making a non moving box

void staticbox(char\* val, int top)

{

char c;

c=val[0];

//sets colour of the box

setcolor(LIGHTGREEN);

//makes the box

bar(285,top-10,315,top);

//writes the text in the box

if(c=='0')

outtextxy(300,top-10,"0");

else if(c=='1')

outtextxy(300,top-10,"1");

else if(c=='2')

outtextxy(300,top-10,"2");

else if(c=='3')

outtextxy(300,top-10,"3");

else if(c=='4')

outtextxy(300,top-10,"4");

else if(c=='5')

outtextxy(300,top-10,"5");

else if(c=='6')

outtextxy(300,top-10,"6");

else if(c=='7')

outtextxy(300,top-10,"7");

else if(c=='8')

outtextxy(300,top-10,"8");

else if(c=='9')

outtextxy(300,top-10,"9");

//sets colour to default

setcolor(WHITE);

}

//function to print all operations

void processprint(vector<char> proc, string vals)

{

int top=30;

//sets the OPERATIONS heading

outtextxy(380,top-5,"OPERATIONS:");

int i=0,j=0;

top+=25;

//prints all the previously occurred processes

for(;i<proc.size();i++)

{

char c=vals[j];

if(i==proc.size()-1)

{

//highlights the current process with red colour

setcolor(LIGHTRED);

}

if(proc[i]=='i')

{

//printing in case of push operation

if(c=='0')

outtextxy(380,top,"Push 0");

else if(c=='1')

outtextxy(380,top,"Push 1");

else if(c=='2')

outtextxy(380,top,"Push 2");

else if(c=='3')

outtextxy(380,top,"Push 3");

else if(c=='4')

outtextxy(380,top,"Push 4");

else if(c=='5')

outtextxy(380,top,"Push 5");

else if(c=='6')

outtextxy(380,top,"Push 6");

else if(c=='7')

outtextxy(380,top,"Push 7");

else if(c=='8')

outtextxy(380,top,"Push 8");

else if(c=='9')

outtextxy(380,top,"Push 9");

j++;

}

else

{

//printing in case of pop operation

outtextxy(380,top,"Pop");

}

top+=10;

//setting colour to default

setcolor(WHITE);

}

}

//popping an element

void deletebox(int top, int sze,vector<char> stk,vector<char> proc, string vals)

{

char c=\*(stk.end()-1);

int j;

for(int top1=top;top1>=50;top1--)

{

//creating the environment

createStackgrap(sze);

//printing the names of the processes being carried out

processprint(proc,vals);

//creating the new element's container box

bar(285,top1,315,top1-10);

//putting text inside the box

if(c=='0')

outtextxy(300,top1-10,"0");

else if(c=='1')

outtextxy(300,top1-10,"1");

else if(c=='2')

outtextxy(300,top1-10,"2");

else if(c=='3')

outtextxy(300,top1-10,"3");

else if(c=='4')

outtextxy(300,top1-10,"4");

else if(c=='5')

outtextxy(300,top1-10,"5");

else if(c=='6')

outtextxy(300,top1-10,"6");

else if(c=='7')

outtextxy(300,top1-10,"7");

else if(c=='8')

outtextxy(300,top1-10,"8");

else if(c=='9')

outtextxy(300,top1-10,"9");

//maintaining the previously inserted elements in stack

int k=stckbtm-5;

for(j=0;j<stk.size()-1;j++)

{

char c1=stk[j];

staticbox(&c1,k);

k-=20;

}

//sustaining the graphics on the screen for a delayed period

delay(0);

//clearing the screen

cleardevice();

}

//printing all the previously occurred processes

processprint(proc,vals);

//creating the environment

createStackgrap(sze);

//creating all the elements previously present in the stack

int k=stckbtm-5;

for(j=0;j<stk.size()-1;j++)

{

char c1=stk[j];

staticbox(&c1,k);

k-=20;

}

//sustaining the graphics after completion of operation

delay(500);

}

//function for inserting a new box

void insertbox(char\* val,int top, int sze,vector<char> stk,vector<char> proc, string vals)

{

//insertion in stack

int top1=50;

int j;

char c=\*val;

for(;top1<=top;top1++)

{

//creating the environment

createStackgrap(sze);

//printing the names of the processes

processprint(proc,vals);

//creating the new element's container box

bar(285,top1-10,315,top1);

//putting text in the box

if(c=='0')

outtextxy(300,top1-10,"0");

else if(c=='1')

outtextxy(300,top1-10,"1");

else if(c=='2')

outtextxy(300,top1-10,"2");

else if(c=='3')

outtextxy(300,top1-10,"3");

else if(c=='4')

outtextxy(300,top1-10,"4");

else if(c=='5')

outtextxy(300,top1-10,"5");

else if(c=='6')

outtextxy(300,top1-10,"6");

else if(c=='7')

outtextxy(300,top1-10,"7");

else if(c=='8')

outtextxy(300,top1-10,"8");

else if(c=='9')

outtextxy(300,top1-10,"9");

//maintaining the previously inserted elements in stack

int k=stckbtm-5;

for(j=0;j<stk.size();j++)

{

char c1=stk[j];

staticbox(&c1,k);

k-=20;

}

//sustaining the graphics on the screen for a delayed period

delay(0);

//clearing the screen

cleardevice();

}

//printing all the previously occurred processes

processprint(proc,vals);

//creating the environment

createStackgrap(sze);

//creating all the elements previously present in the stack

int k=stckbtm-5;

for(j=0;j<stk.size();j++)

{

char c1=stk[j];

staticbox(&c1,k);

k-=20;

}

//creating the box for newly inserted element

bar(285,top-10,315,top);

//putting text inside the box of the newly inserted element

if(c=='0')

outtextxy(300,top1-10,"0");

else if(c=='1')

outtextxy(300,top1-10,"1");

else if(c=='2')

outtextxy(300,top1-10,"2");

else if(c=='3')

outtextxy(300,top1-10,"3");

else if(c=='4')

outtextxy(300,top1-10,"4");

else if(c=='5')

outtextxy(300,top1-10,"5");

else if(c=='6')

outtextxy(300,top1-10,"6");

else if(c=='7')

outtextxy(300,top1-10,"7");

else if(c=='8')

outtextxy(300,top1-10,"8");

else if(c=='9')

outtextxy(300,top1-10,"9");

//checking for overflow;

if(top-20<stckbtm-5-sze\*20)

{

//code to make OVERFLOW blink in yellow colour

for(int blink=0;blink<=8;blink++)

{

if(blink%2==0)

setcolor(YELLOW);

outtextxy(265,100,"OVERFLOW");

delay(200);

setcolor(BLACK);

}

//code to make overflow element fly up the screen

stk.push\_back(c);

deletebox(top,sze,stk,proc,vals);

stk.erase(stk.begin()+stk.size()-1);

}

else

{

//sustaining the graphics on the screen

delay(500);

}

}

int main()

{

//declaring the size\_of\_stack variable and the top\_of\_stack\_y\_coordinate variable

int sze,top=stckbtm-5;

//declaring strings to store order of operations and order in which numbers are inserted

string ops,nos;

//declaring a vector to act as stack

vector<char> stk;

//declaring a vector to store previously occurred operations

vector<char> proc;

//counter variables declaration

int i=0,j=0;

//characters that take element input for insertion and decision input for choosing which action to perform

char ele,dec;

//taking inputs

cout<<"Enter size of stack:\n";

cin>>sze;

//

do

{

cout<<"Enter i to push, d to pop and any other key to proceed to the animation:\n";

cin>>dec;

//insertion or push

if(dec=='i')

{

cout<<"Enter any single digit value:\n";

cin>>ele;

nos+=ele;

ops+=dec;

}

//deletion or pop

else if(dec=='d')

{

ops+=dec;

}

else

{

break;

}

}

while(1);

//creating the graphics window

int gd=DETECT;

int gm;

initgraph(&gd, &gm, "C:\\TC\\BGI");

//setting background colour of screen

setbkcolor(BROWN);

//creating the environment

createStackgrap(sze);

processprint(proc,nos);

for(i=0;i<ops.length();i++)

{

//if insertion(push) is performed

if(ops[i]=='i')

{

proc.push\_back('i');

char c=nos[j];

j++;

//calling insertion animation to bring the box into the stack

insertbox(&c,top,sze,stk,proc,nos);

//altering the stack after checking that overflow hasnt occurred

if(top-10>stckbtm-5-(sze\*20))

{

//pushing the newest element onto the stack

stk.push\_back(c);

//updating the top of stack y coordinates

top-=20;

}

}

//if deletion(pop) has been performed

else if(ops[i]=='d')

{

proc.push\_back('d');

//checking for underflow

if(stk.size()==0)

{

//code to make UNDERFLOW blink

for(int blink=0;blink<=8;blink++)

{

if(blink%2==0)

setcolor(YELLOW);

outtextxy(265,100,"UNDERFLOW");

delay(200);

setcolor(BLACK);

}

//printing all the previously occurred processes

processprint(proc,nos);

//moving onto the next iteration

continue;

}

//making the box to be deleted fly up the screen

deletebox(top,sze,stk,proc,nos);

//popping the last element from the stack

stk.erase(stk.begin()+stk.size()-1);

//updating the top of stack y coordinates

top+=20;

}

}

//sustaining the graphics

delay(3000);

//clearing the screen

cleardevice();

//indicating the end of the animation

outtextxy(250,200,"END OF ANIMATION");

//getting a character as a signal to close the graph window

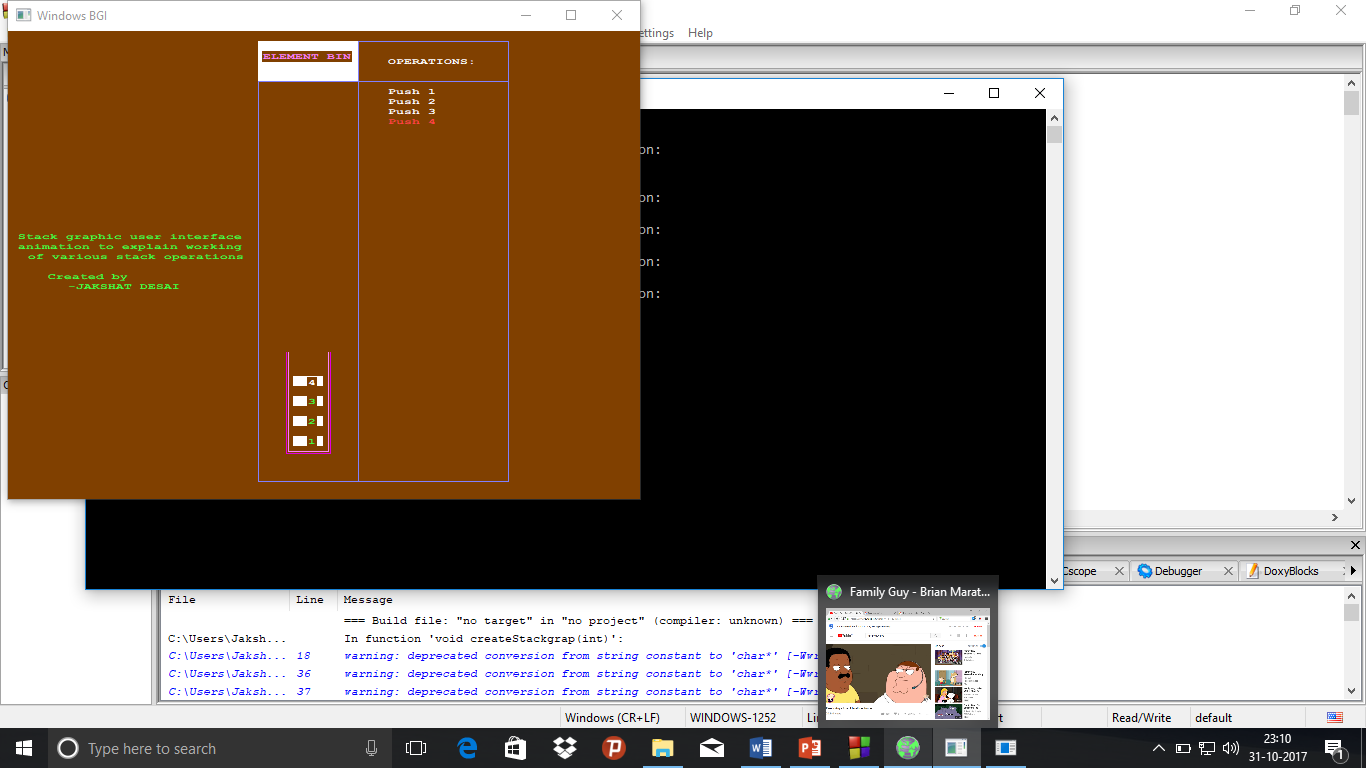
getch();

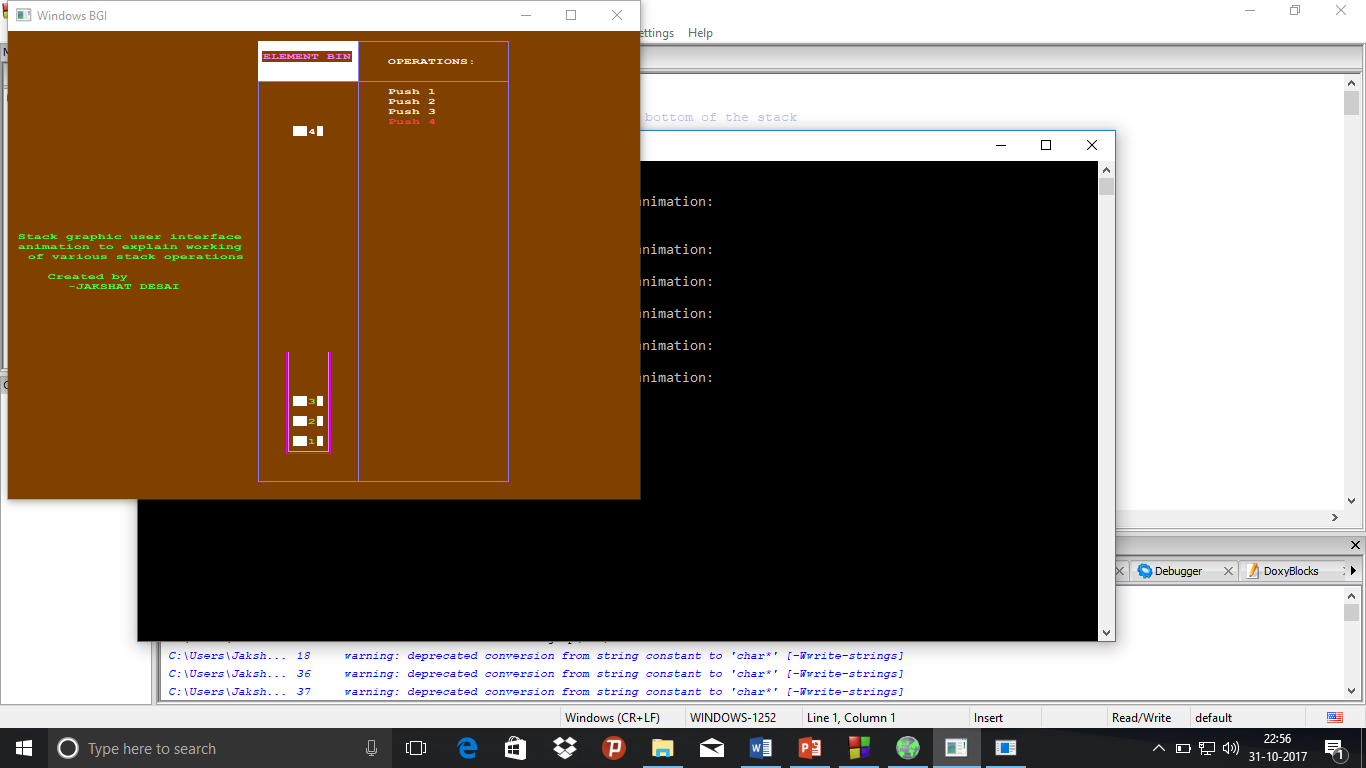
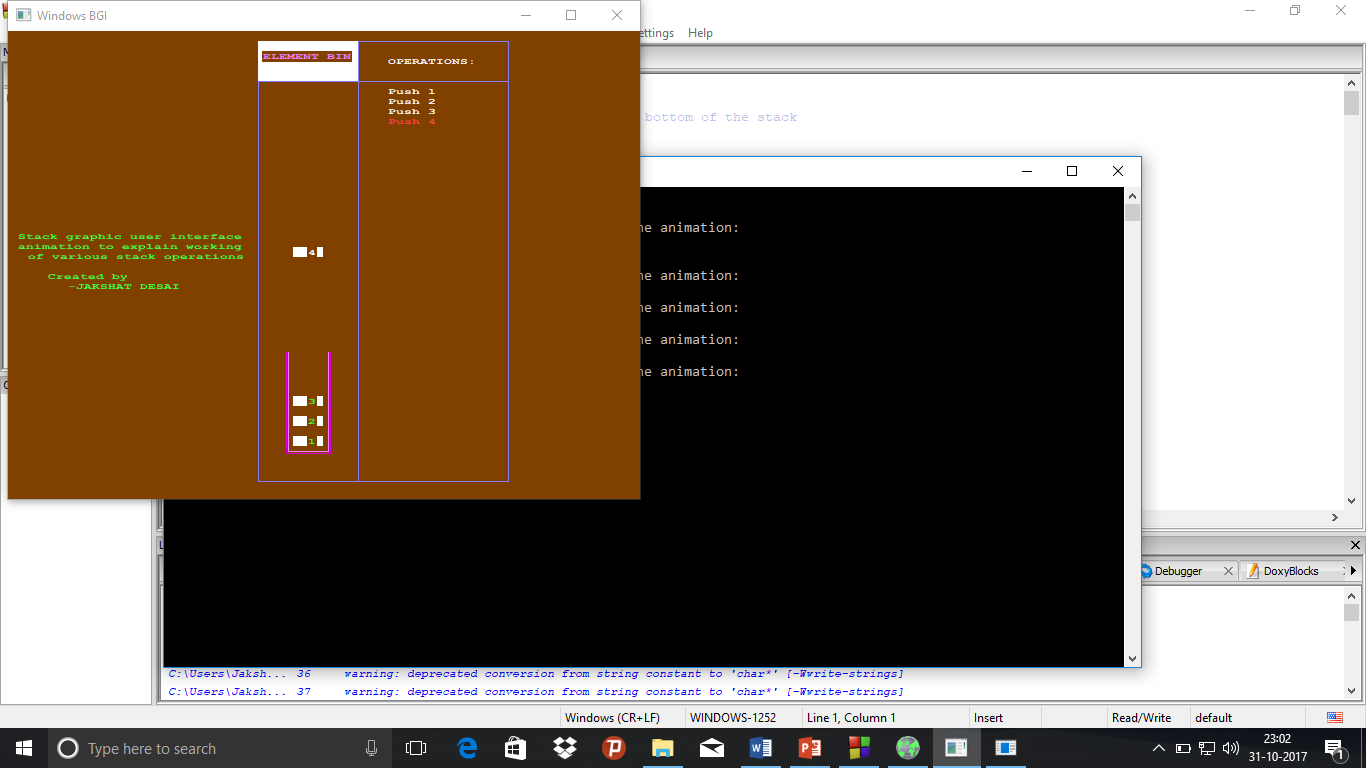
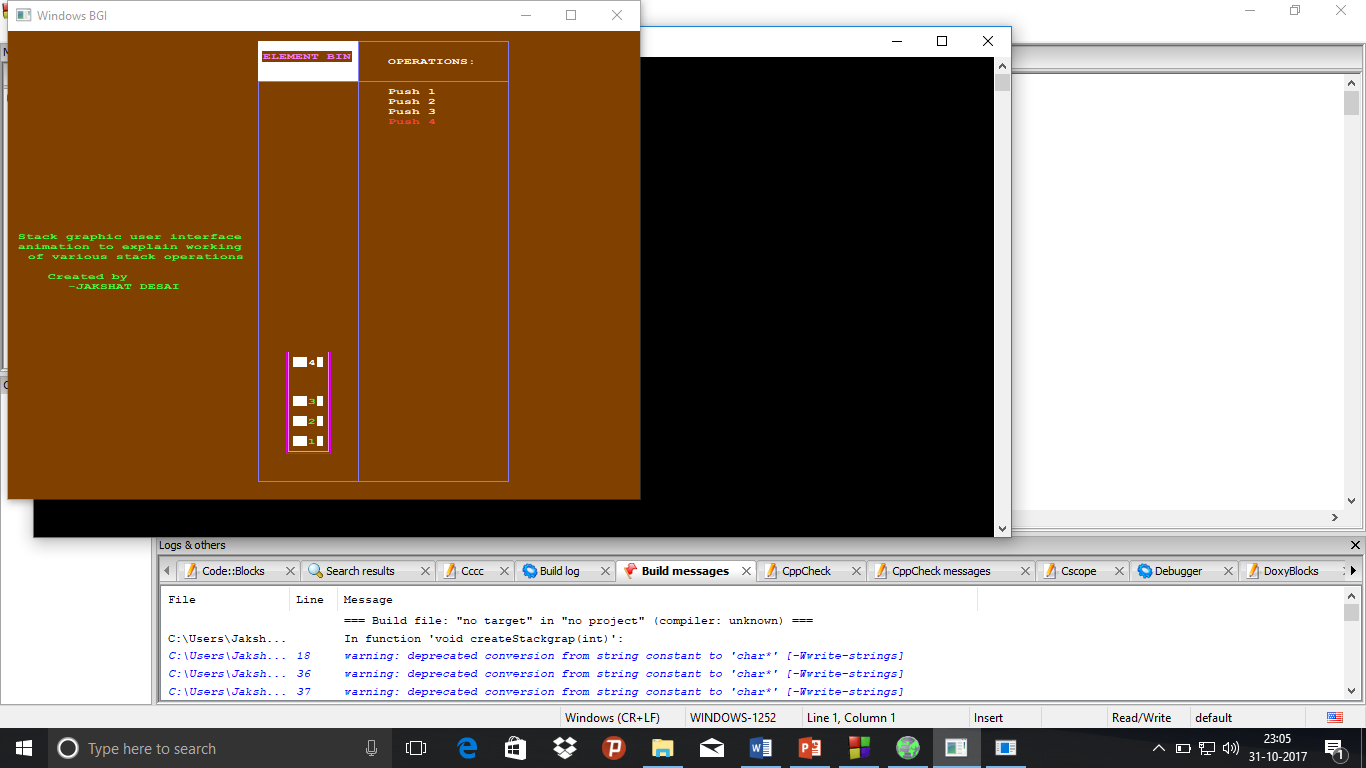
//closes the graph window

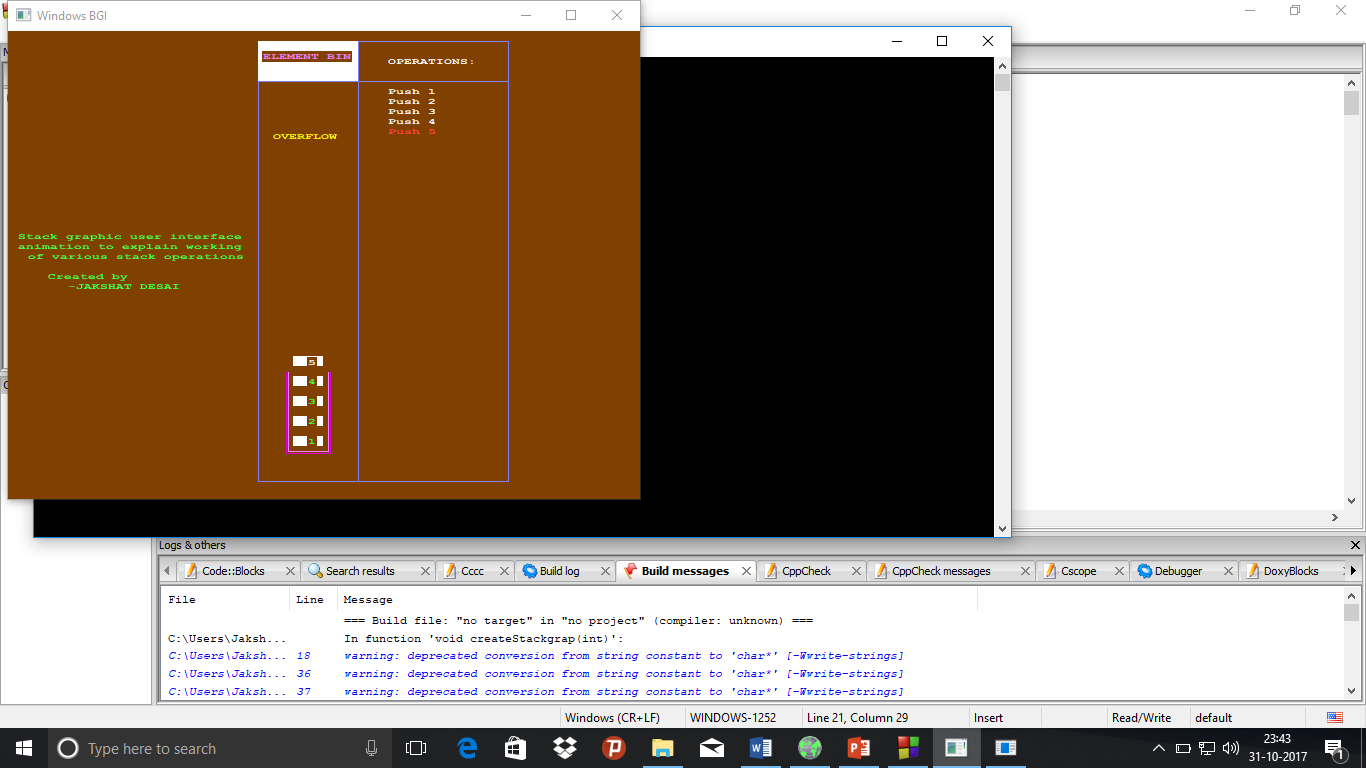
closegraph();

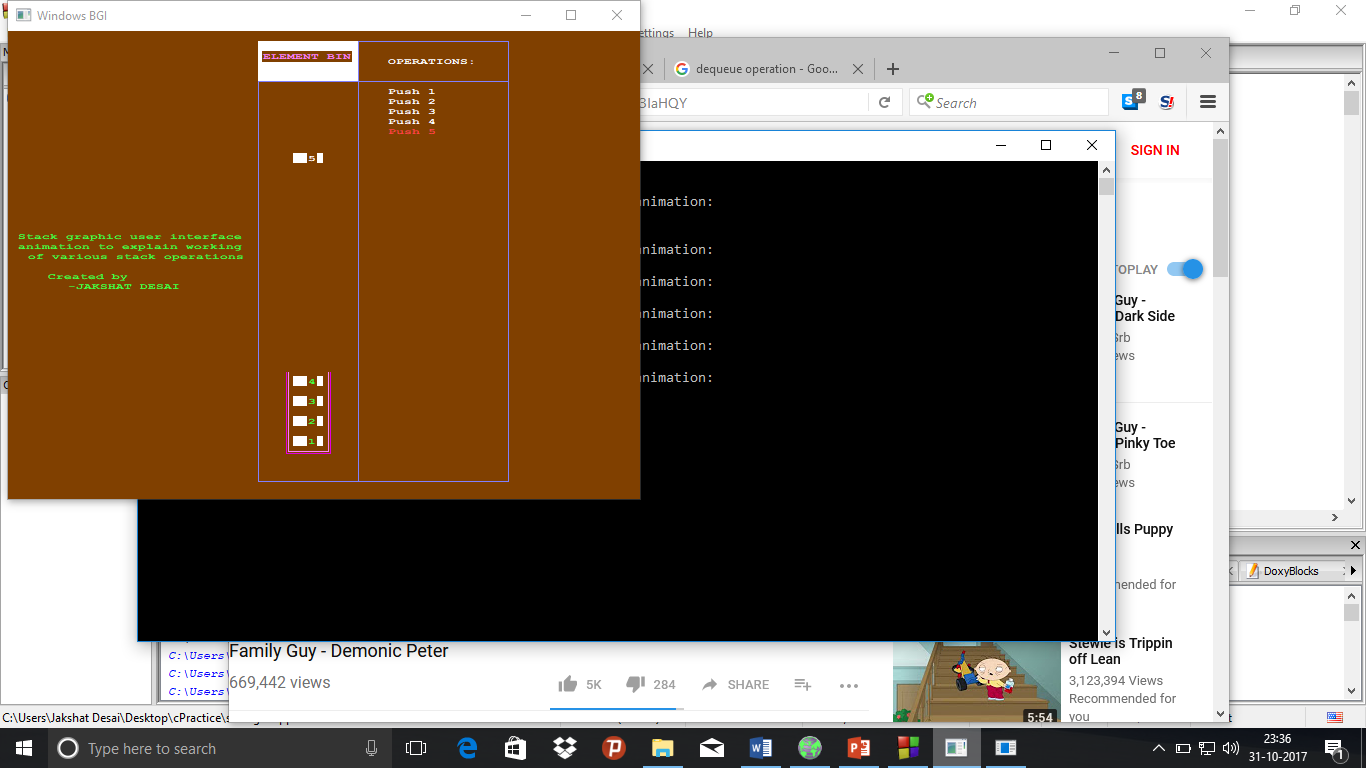
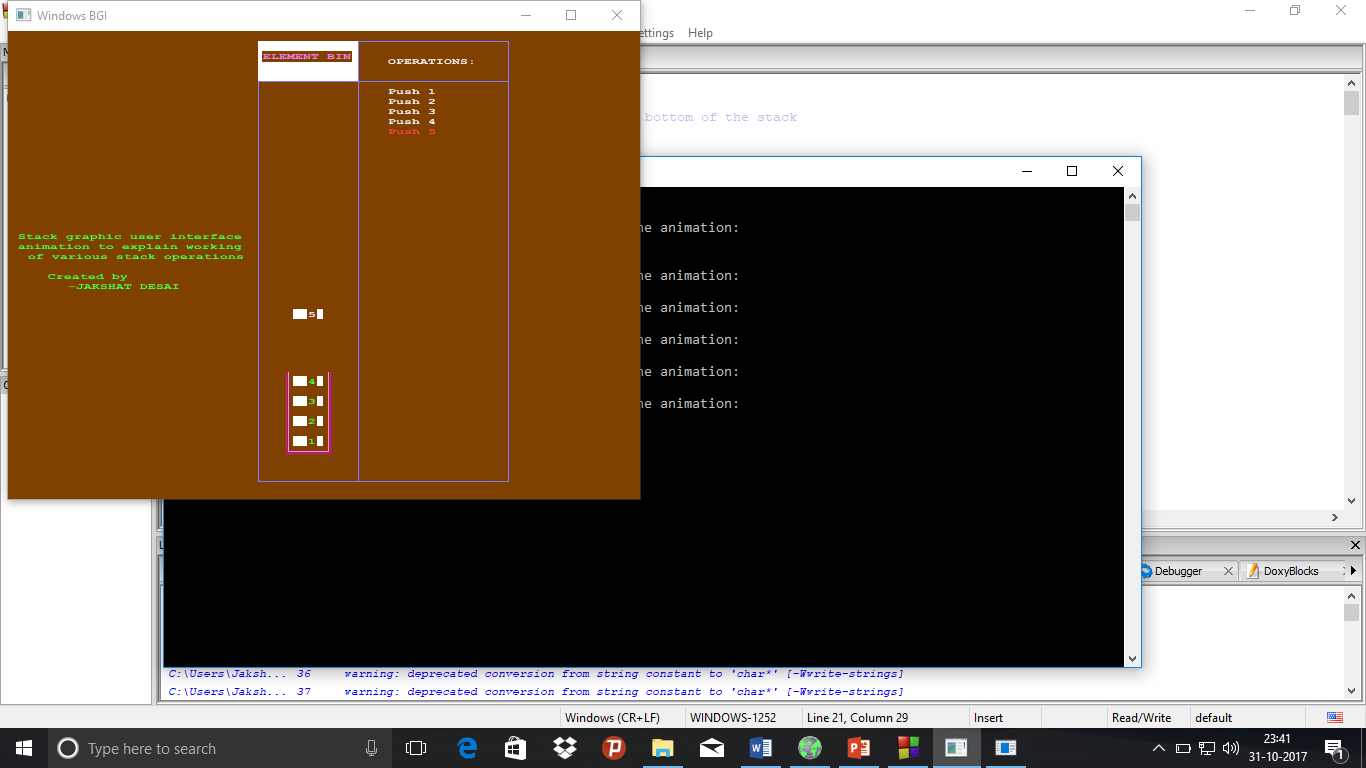
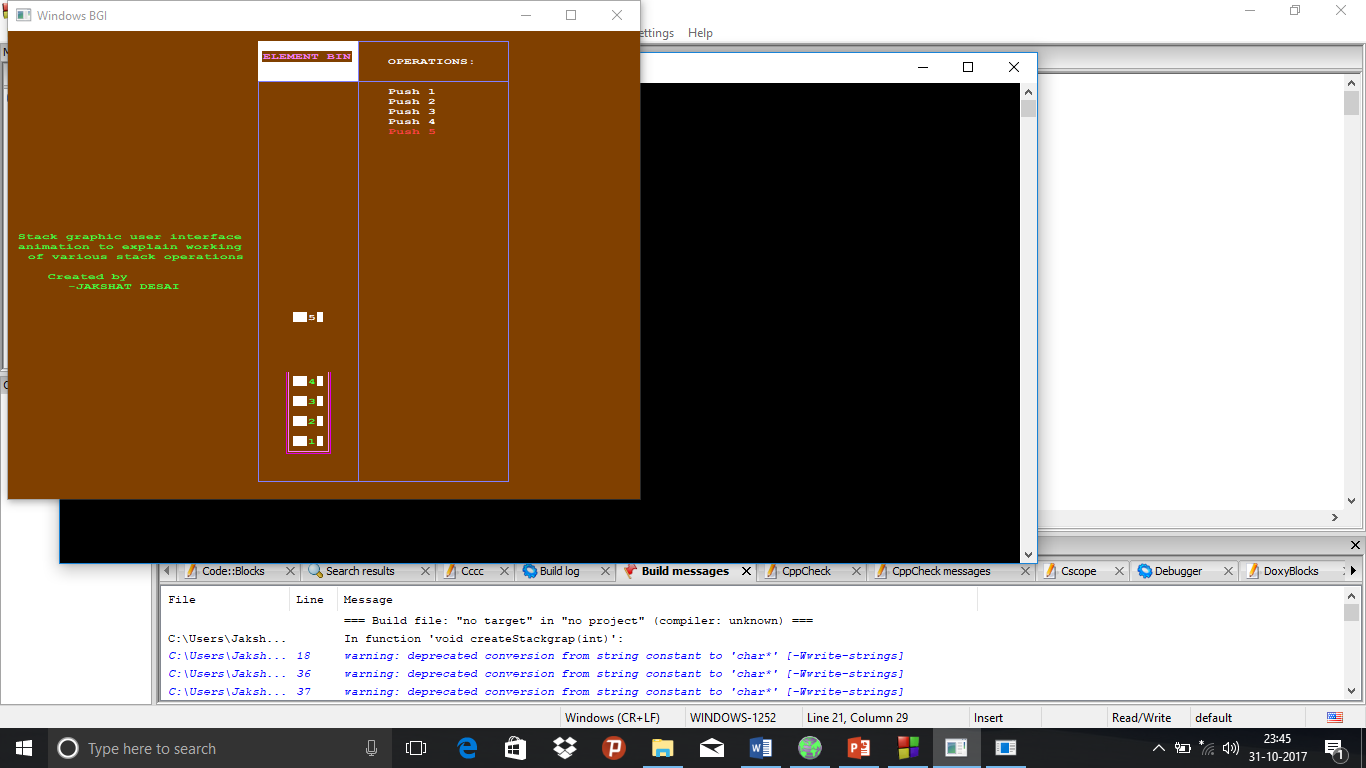
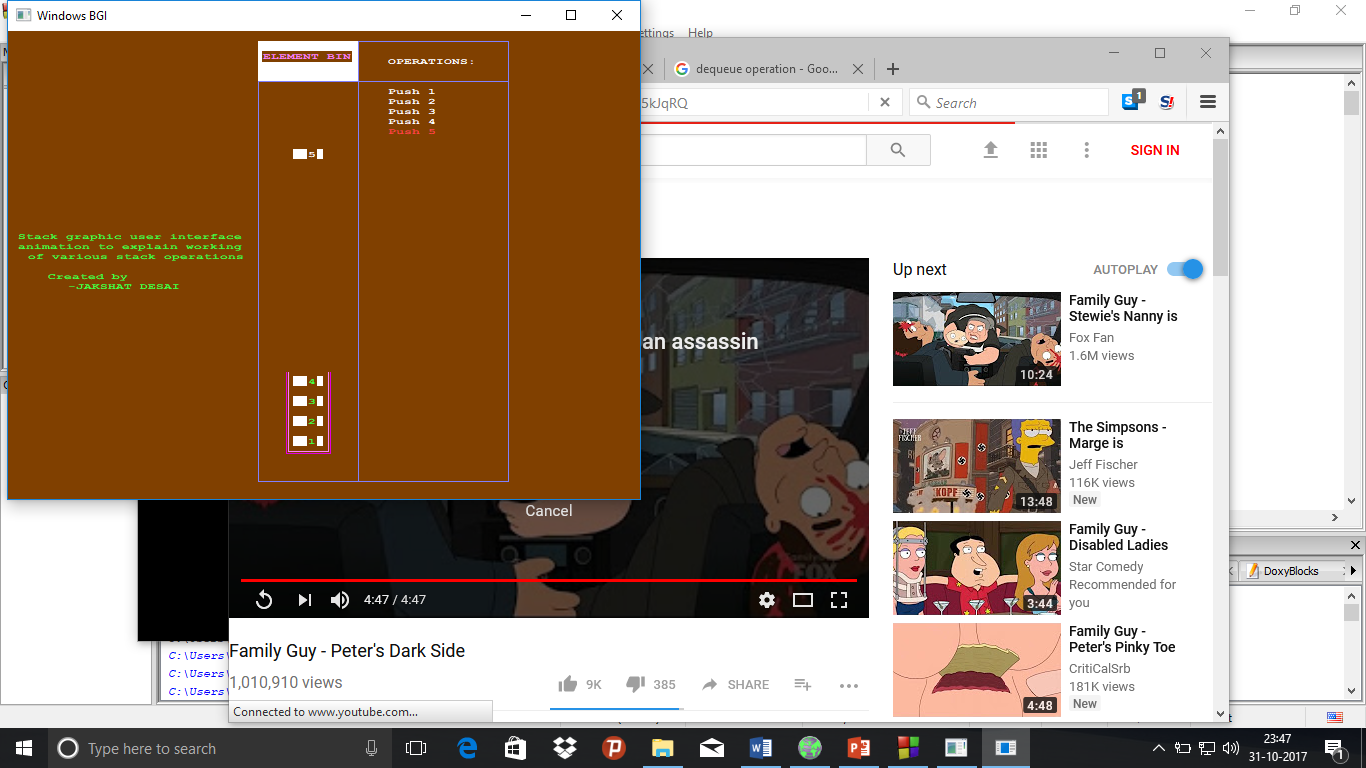
}

**Screenshots**

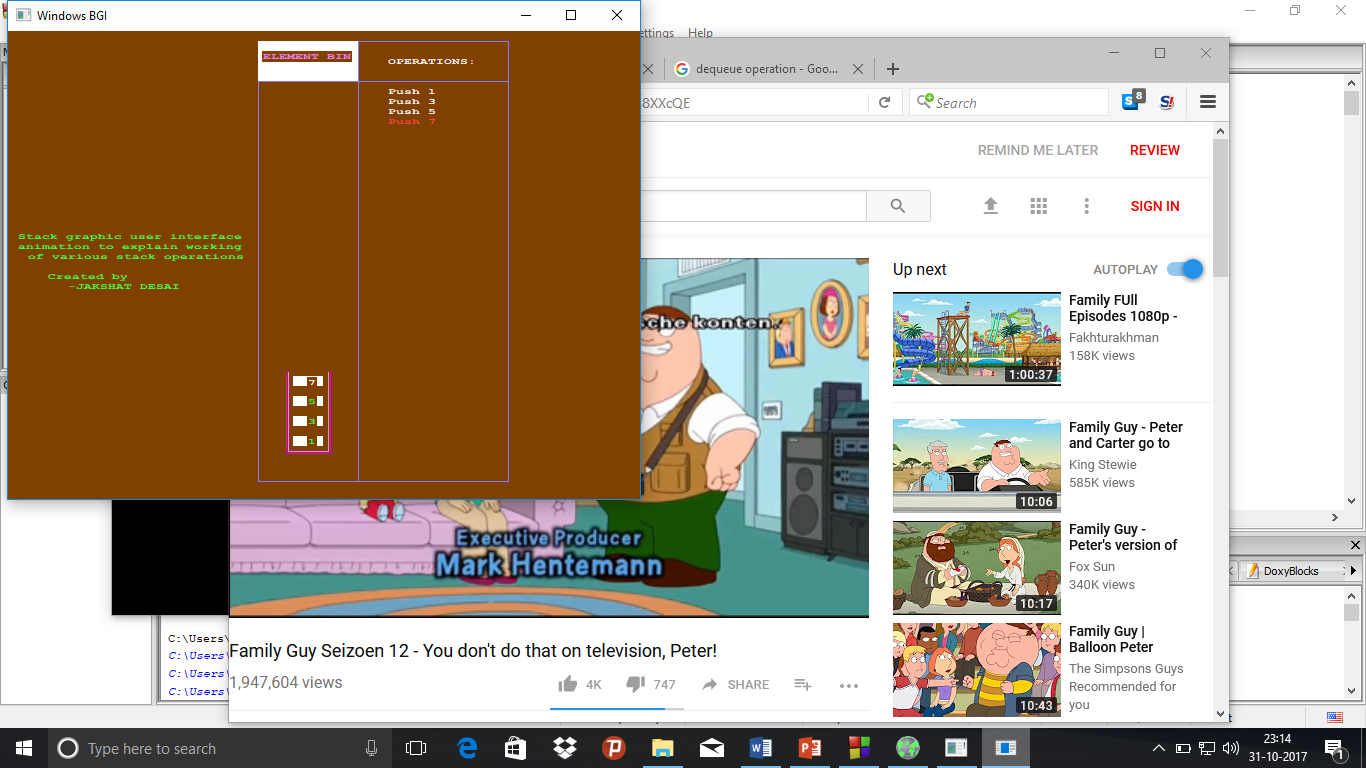
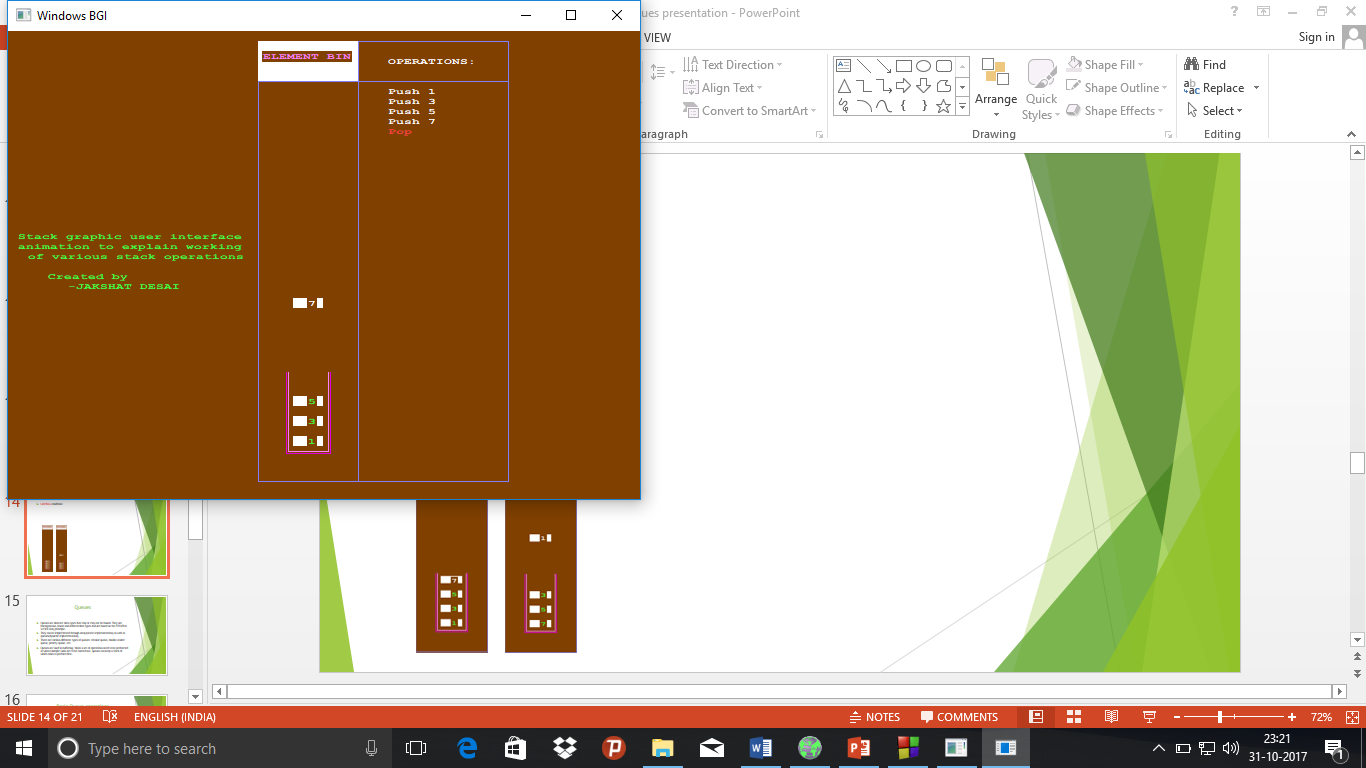
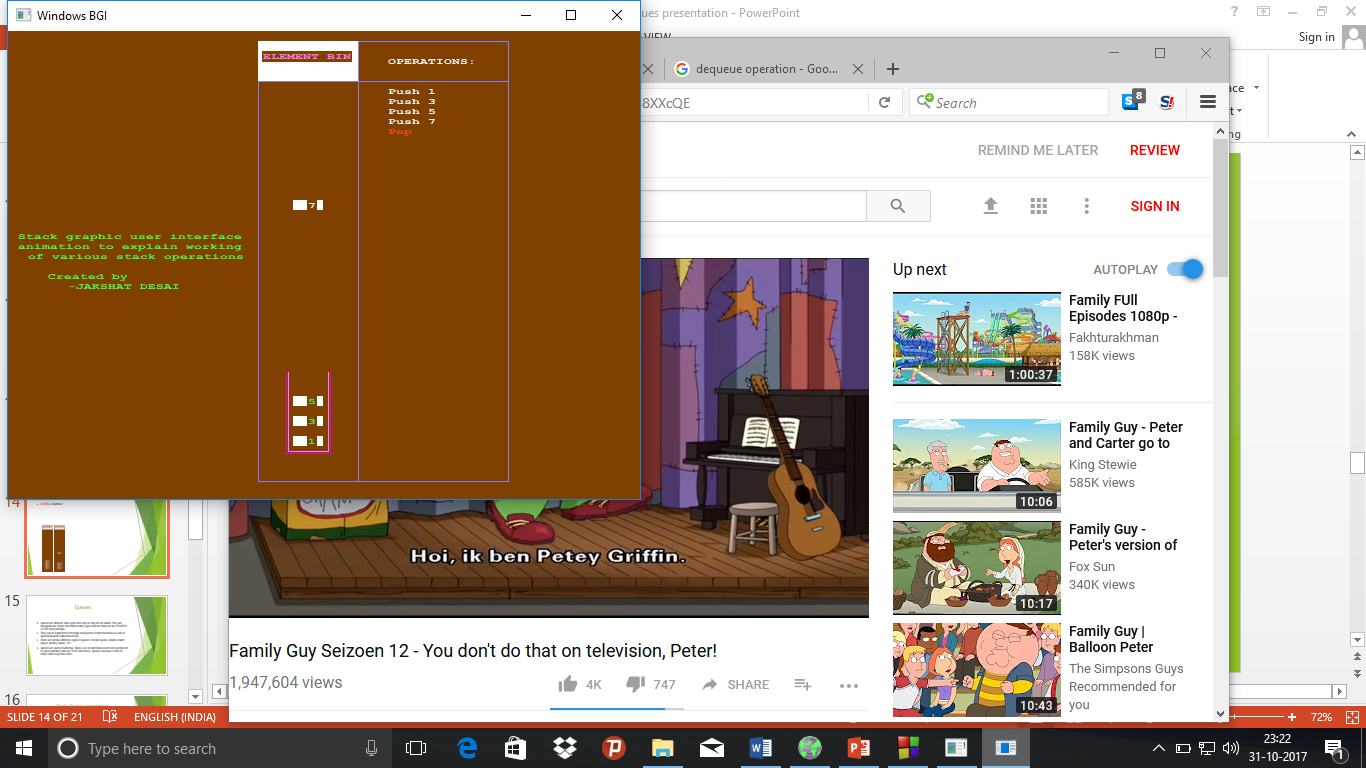
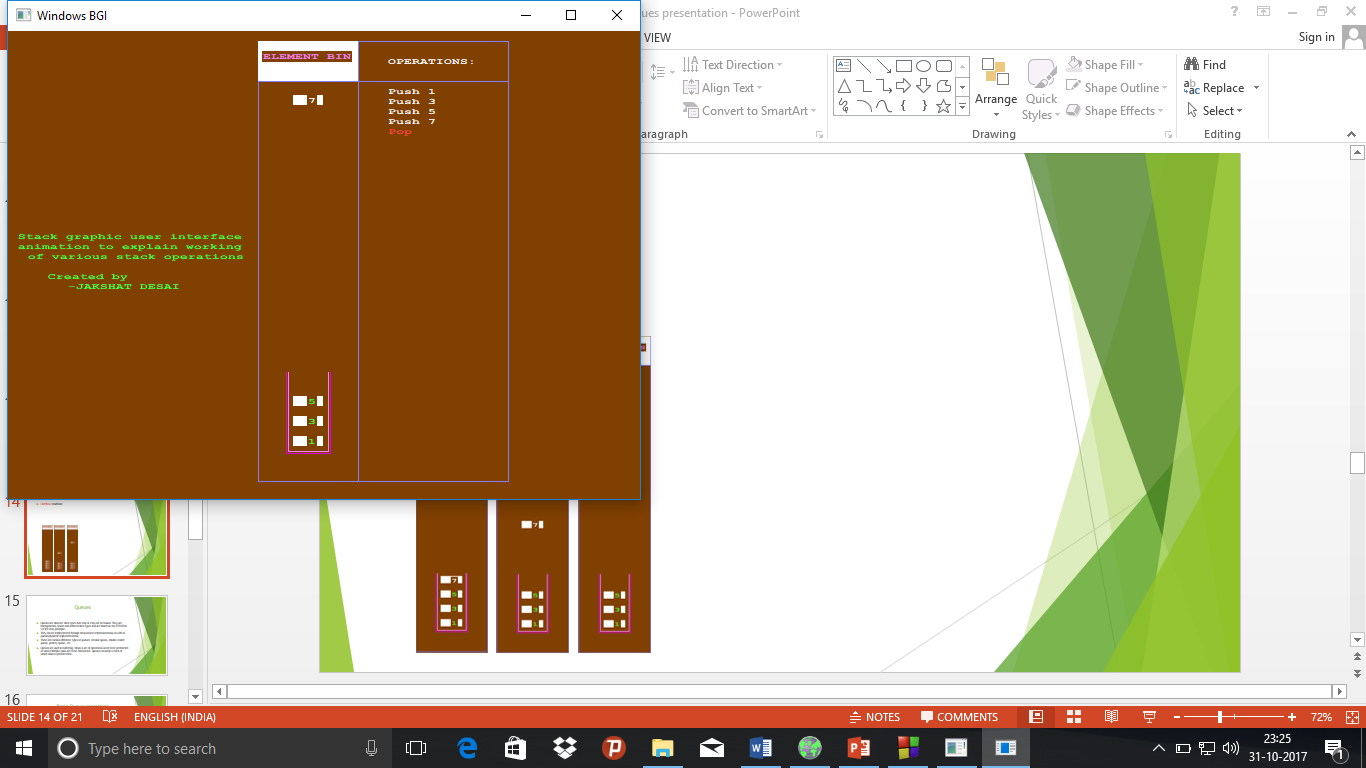
Push operation:

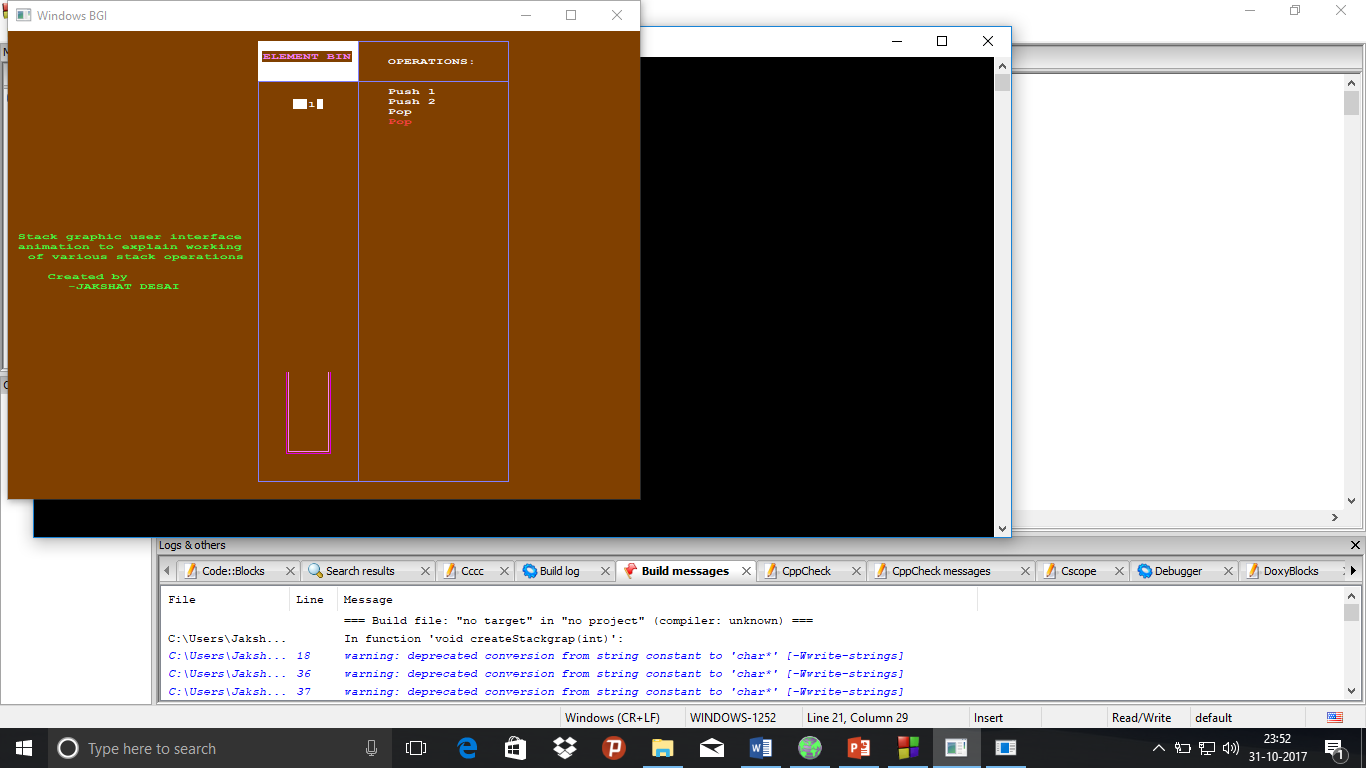
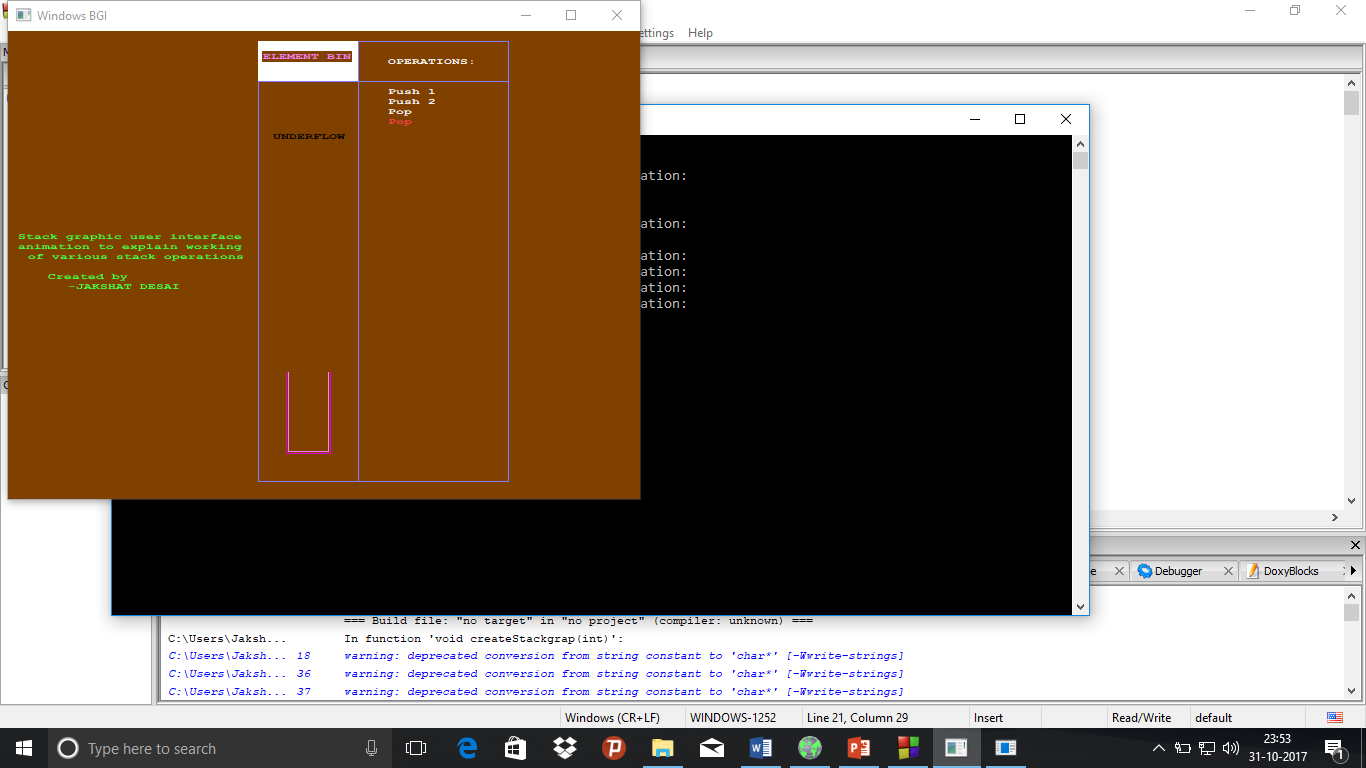


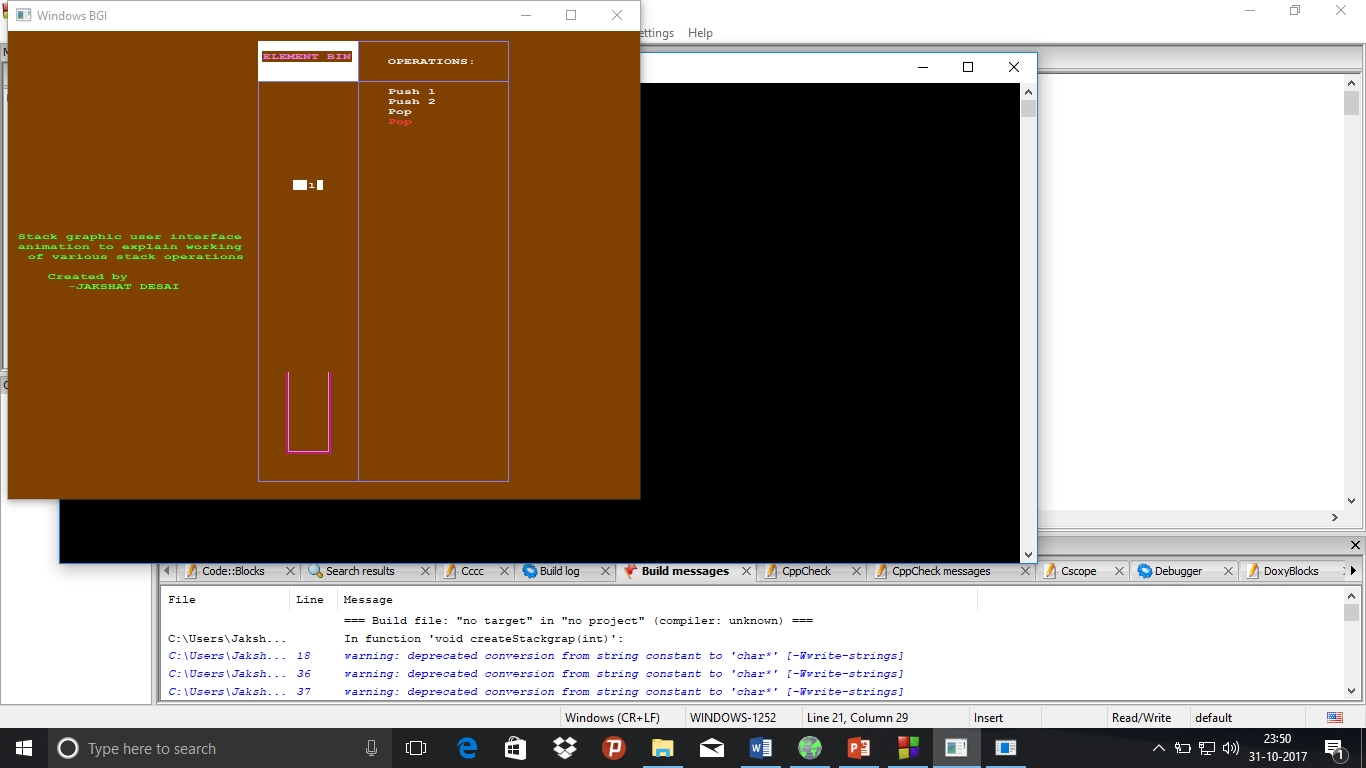
Overflow condition:

****

Pop condition:

****

Underflow condition:



**Queues**

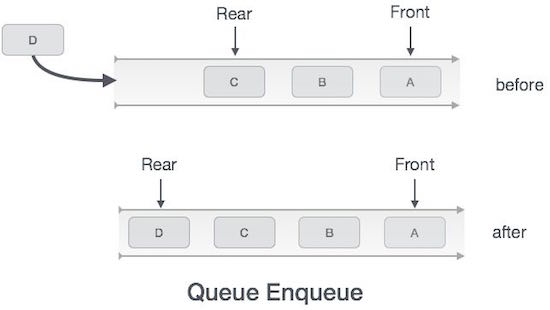
* Queues are abstract data types that may or may not be bound. They are homogeneous, linear and ordered data types and are based on the FIFO(First In First Out) principle.
* They can be implemented through arrays(static implementation) as well as queues(dynamic implementation).
* There are various different types of queues: circular queue, double-ended queue, priority queue, etc.
* Queues are used in buffering. When a set of operations need to be performed or when multiple tasks are to be carried out, queues can keep a track of which tasks to perform first.

Basic Queue operations

* **ENQUEUE(<element>)** :

1)This operation inserts an element in front of the queue.

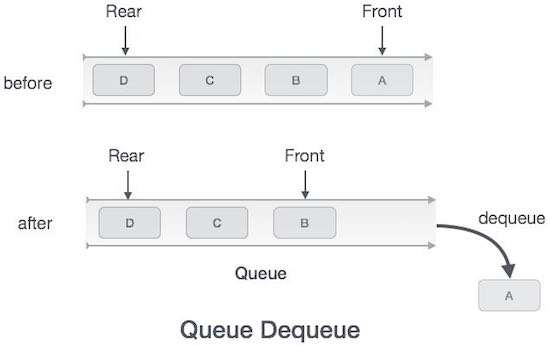
2)When the queue is full(static implementation), that is if no more elements can further be stored in the stack, if insertion is attempted then it is called an overflow condition.



* **DEQUEUE()**:

1)This operation deletes the element present at the top of the stack.

2)If deletion is attempted when the stack doesn’t have a single element, i.e. when the stack is empty, this is said to be an underflow condition.



**Queue Graphic User interface C++ code**

#include<bits/stdc++.h>

#include<graphics.h>

//constant keeping track of y coordinates of the bottom of the queue

#define queuebtm 300

using namespace std;

//function to set the environment

void createQueuegrap(int sze)

{

//setting background colour of screen

setbkcolor(BROWN);

//sets colour for outer box

setcolor(LIGHTBLUE);

//creates outer box

rectangle(250,10,500,450);

//creates bar for ELEMENT BIN

bar(250,10,350,50);

//sets colour for ELEMENT BIN text

setcolor(LIGHTMAGENTA);

//writes the ELEMENT BIN text

outtextxy(255,20,"ELEMENT BIN");

//creates bar for TO PROCESS

bar(250,410,350,450);

//sets colour for TO PROCESS text

setcolor(LIGHTGREEN);

//writes the TO PROCESS text

outtextxy(260,420,"TO PROCESS");

//sets colour for column seperating line

setcolor(LIGHTBLUE);

//column seperating line

line(350,10,350,450);

//line for heading

line(250,50,500,50);

//creating the queue holder

setcolor(LIGHTMAGENTA);

line(280,queuebtm,280,queuebtm-sze\*20);

line(320,queuebtm,320,queuebtm-sze\*20);

setcolor(MAGENTA);

line(278,queuebtm+2,278,queuebtm-sze\*20);

line(322,queuebtm+2,322,queuebtm-sze\*20);

//program description and credits

setcolor(LIGHTGREEN);

outtextxy(10,200,"Queue graphic user interface");

outtextxy(10,210,"animation to explain working");

outtextxy(20,220,"of various queue operations");

outtextxy(40,240,"Created by");

outtextxy(60,250,"-JAKSHAT DESAI");

setcolor(WHITE);

}

//function for making a non moving box

void staticbox(char\* val, int top)

{

char c;

c=val[0];

//sets colour of the box

setcolor(LIGHTGREEN);

//makes the box

bar(285,top-10,315,top);

//writes the text in the box

if(c=='0')

outtextxy(300,top-10,"0");

else if(c=='1')

outtextxy(300,top-10,"1");

else if(c=='2')

outtextxy(300,top-10,"2");

else if(c=='3')

outtextxy(300,top-10,"3");

else if(c=='4')

outtextxy(300,top-10,"4");

else if(c=='5')

outtextxy(300,top-10,"5");

else if(c=='6')

outtextxy(300,top-10,"6");

else if(c=='7')

outtextxy(300,top-10,"7");

else if(c=='8')

outtextxy(300,top-10,"8");

else if(c=='9')

outtextxy(300,top-10,"9");

//sets colour to default

setcolor(WHITE);

}

//function to print all operations

void processprint(vector<char> proc, string vals)

{

int top=30;

//sets the OPERATIONS heading

outtextxy(380,top-5,"OPERATIONS:");

int i=0,j=0;

top+=25;

//prints all the previously occurred processes

for(;i<proc.size();i++)

{

char c=vals[j];

if(i==proc.size()-1)

{

//highlights the current process

setcolor(LIGHTRED);

}

if(proc[i]=='i')

{

//printing in case of enqueue operation

if(c=='0')

outtextxy(380,top,"Enqueue 0");

else if(c=='1')

outtextxy(380,top,"Enqueue 1");

else if(c=='2')

outtextxy(380,top,"Enqueue 2");

else if(c=='3')

outtextxy(380,top,"Enqueue 3");

else if(c=='4')

outtextxy(380,top,"Enqueue 4");

else if(c=='5')

outtextxy(380,top,"Enqueue 5");

else if(c=='6')

outtextxy(380,top,"Enqueue 6");

else if(c=='7')

outtextxy(380,top,"Enqueue 7");

else if(c=='8')

outtextxy(380,top,"Enqueue 8");

else if(c=='9')

outtextxy(380,top,"Enqueue 9");

j++;

}

else

{

//printing in case of dequeue operation

outtextxy(380,top,"Dequeue");

}

top+=10;

//setting colour to default

setcolor(WHITE);

}

}

//removing an element

void deletebox(int top, int sze,vector<char> que,vector<char> proc, string vals)

{

char c=\*(que.begin());

int j;

for(int top1=queuebtm-5;top1<=410;top1++)

{

//creating the environment

createQueuegrap(sze);

//printing the names of the processes being carried out

processprint(proc,vals);

//creating the new element's container box

bar(285,top1,315,top1-10);

//putting text inside the box

if(c=='0')

outtextxy(300,top1-10,"0");

else if(c=='1')

outtextxy(300,top1-10,"1");

else if(c=='2')

outtextxy(300,top1-10,"2");

else if(c=='3')

outtextxy(300,top1-10,"3");

else if(c=='4')

outtextxy(300,top1-10,"4");

else if(c=='5')

outtextxy(300,top1-10,"5");

else if(c=='6')

outtextxy(300,top1-10,"6");

else if(c=='7')

outtextxy(300,top1-10,"7");

else if(c=='8')

outtextxy(300,top1-10,"8");

else if(c=='9')

outtextxy(300,top1-10,"9");

//maintaining the previously inserted elements in stack

int k=queuebtm-25;

for(j=1;j<que.size();j++)

{

char c1=que[j];

staticbox(&c1,k);

k-=20;

}

//sustaining the graphics on the screen for a delayed period

delay(0);

//clearing the screen

cleardevice();

}

//printing all the previously occurred processes

processprint(proc,vals);

//creating the environment

createQueuegrap(sze);

//creating all the elements previously present in the stack

//and moving the remaining elements down one by one

int k=queuebtm-5;

for(j=1;j<que.size();j++)

{

char c1=que[j];

for(int i=k-20;i<=k;i++)

{

processprint(proc,vals);

createQueuegrap(sze);

int l=queuebtm-25;

//loop taking care of displaying the non moving elements

for(int x=1;x<que.size();x++)

{

processprint(proc,vals);

createQueuegrap(sze);

char c2=que[x];

if(x>j)

{

staticbox(&c2,l);

}

if(x<j)

{

staticbox(&c2,l+20);

}

l-=20;

}

//moving element

staticbox(&c1,i);

delay(20);

cleardevice();

}

k-=20;

}

//prints all the previously occurred processes

processprint(proc,vals);

//creating the environment

createQueuegrap(sze);

//displaying the final queue

k=queuebtm-5;

for(j=1;j<que.size();j++)

{

char c1=que[j];

staticbox(&c1,k);

k-=20;

}

//sustaining the graphics after completion of operation

delay(500);

}

void boxflyup(int top, int sze,vector<char> que,vector<char> proc, string vals)

{

char c=\*(que.end()-1);

int j;

for(int top1=top;top1>=50;top1--)

{

//creating the environment

createQueuegrap(sze);

//printing the names of the processes being carried out

processprint(proc,vals);

//creating the new element's container box

bar(285,top1,315,top1-10);

//putting text inside the box

if(c=='0')

outtextxy(300,top1-10,"0");

else if(c=='1')

outtextxy(300,top1-10,"1");

else if(c=='2')

outtextxy(300,top1-10,"2");

else if(c=='3')

outtextxy(300,top1-10,"3");

else if(c=='4')

outtextxy(300,top1-10,"4");

else if(c=='5')

outtextxy(300,top1-10,"5");

else if(c=='6')

outtextxy(300,top1-10,"6");

else if(c=='7')

outtextxy(300,top1-10,"7");

else if(c=='8')

outtextxy(300,top1-10,"8");

else if(c=='9')

outtextxy(300,top1-10,"9");

//maintaining the previously inserted elements in queue

int k=queuebtm-5;

for(j=0;j<que.size()-1;j++)

{

char c1=que[j];

staticbox(&c1,k);

k-=20;

}

//sustaining the graphics on the screen for a delayed period

delay(0);

//clearing the screen

cleardevice();

}

//printing all the previously occurred processes

processprint(proc,vals);

//creating the environment

createQueuegrap(sze);

//creating all the elements previously present in the queue

int k=queuebtm-5;

for(j=0;j<que.size()-1;j++)

{

char c1=que[j];

staticbox(&c1,k);

k-=20;

}

//sustaining the graphics after completion of operation

delay(500);

}

//function for inserting a new box

void insertbox(char\* val,int top, int sze,vector<char> que,vector<char> proc, string vals)

{

//insertion in queue

int top1=50;

int j;

char c=\*val;

for(;top1<=top;top1++)

{

//creating the environment

createQueuegrap(sze);

//printing the names of the processes

processprint(proc,vals);

//creating the new element's container box

bar(285,top1-10,315,top1);

//putting text in the box

if(c=='0')

outtextxy(300,top1-10,"0");

else if(c=='1')

outtextxy(300,top1-10,"1");

else if(c=='2')

outtextxy(300,top1-10,"2");

else if(c=='3')

outtextxy(300,top1-10,"3");

else if(c=='4')

outtextxy(300,top1-10,"4");

else if(c=='5')

outtextxy(300,top1-10,"5");

else if(c=='6')

outtextxy(300,top1-10,"6");

else if(c=='7')

outtextxy(300,top1-10,"7");

else if(c=='8')

outtextxy(300,top1-10,"8");

else if(c=='9')

outtextxy(300,top1-10,"9");

//maintaining the previously inserted elements in queue

int k=queuebtm-5;

for(j=0;j<que.size();j++)

{

char c1=que[j];

staticbox(&c1,k);

k-=20;

}

//sustaining the graphics on the screen for a delayed period

delay(0);

//clearing the screen

cleardevice();

}

//printing all the previously occurred processes

processprint(proc,vals);

//creating the environment

createQueuegrap(sze);

//creating all the elements previously present in the queue

int k=queuebtm-5;

for(j=0;j<que.size();j++)

{

char c1=que[j];

staticbox(&c1,k);

k-=20;

}

//creating the box for newly inserted element

bar(285,top-10,315,top);

//putting text inside the box of the newly inserted element

if(c=='0')

outtextxy(300,top1-10,"0");

else if(c=='1')

outtextxy(300,top1-10,"1");

else if(c=='2')

outtextxy(300,top1-10,"2");

else if(c=='3')

outtextxy(300,top1-10,"3");

else if(c=='4')

outtextxy(300,top1-10,"4");

else if(c=='5')

outtextxy(300,top1-10,"5");

else if(c=='6')

outtextxy(300,top1-10,"6");

else if(c=='7')

outtextxy(300,top1-10,"7");

else if(c=='8')

outtextxy(300,top1-10,"8");

else if(c=='9')

outtextxy(300,top1-10,"9");

//checking for overflow;

if(top-20<queuebtm-5-sze\*20)

{

//code to make OVERFLOW blink in yellow colour

for(int blink=0;blink<=8;blink++)

{

if(blink%2==0)

setcolor(YELLOW);

outtextxy(265,100,"OVERFLOW");

delay(200);

setcolor(BLACK);

}

//code to make overflow element fly up the screen

que.push\_back(c);

boxflyup(top,sze,que,proc,vals);

que.erase(que.begin()+que.size()-1);

}

else

{

//sustaining the graphics on the screen

delay(500);

}

}

int main()

{

//declaring the size\_of\_queue variable and the front pointer\_of\_queue\_y\_coordinate variable

int sze,top=queuebtm-5;

//declaring strings to store order of operations and order in which numbers are inserted

string ops,nos;

//declaring a vector to act as queue

vector<char> que;

//declaring a vector to store previously occurred operations

vector<char> proc;

//counter variables declaration

int i=0,j=0;

//characters that take element input for insertion and decision input for choosing which action to perform

char ele,dec;

//taking inputs

cout<<"Enter size of queue:\n";

cin>>sze;

do

{

cout<<"Enter i to enqueue, d to dequeue and any other key to proceed to the animation:\n";

cin>>dec;

//insertion or enqueue

if(dec=='i')

{

cout<<"Enter any single digit value:\n";

cin>>ele;

nos+=ele;

ops+=dec;

}

//deletion or dequeue

else if(dec=='d')

{

ops+=dec;

}

else

{

break;

}

}

while(1);

//creating the graphics window

int gd=DETECT;

int gm;

initgraph(&gd, &gm, "C:\\TC\\BGI");

//creating the environment

createQueuegrap(sze);

processprint(proc,nos);

for(i=0;i<ops.length();i++)

{

//if insertion(enqueue) is performed

if(ops[i]=='i')

{

proc.push\_back('i');

char c=nos[j];

j++;

//calling insertion animation to bring the box into the queue

insertbox(&c,top,sze,que,proc,nos);

//altering the queue after checking that overflow hasnt occurred

if(top-10>queuebtm-5-(sze\*20))

{

//inserting the newest element into the queue

que.push\_back(c);

//updating the front pointer of queue y coordinates

top-=20;

}

}

//if deletion(dequeue) has been performed

else if(ops[i]=='d')

{

proc.push\_back('d');

//checking for empty list

if(que.size()==0)

{

//code to make EMPTY LIST blink

for(int blink=0;blink<=8;blink++)

{

if(blink%2==0)

setcolor(YELLOW);

outtextxy(257,100,"EMPTY QUEUE");

delay(200);

setcolor(BLACK);

}

//printing all the previously occurred processes

processprint(proc,nos);

//moving onto the next iteration

continue;

}

//making the box to be deleted fly up the screen

deletebox(top,sze,que,proc,nos);

//dequeueing the element from the queue

que.erase(que.begin());

//updating the front pointer of queue y coordinates

top+=20;

}

}

//sustaining the graphics

delay(3000);

//clearing the screen

cleardevice();

//indicating the end of the animation

outtextxy(250,200,"END OF ANIMATION");

//getting a character as a signal to close the graph window

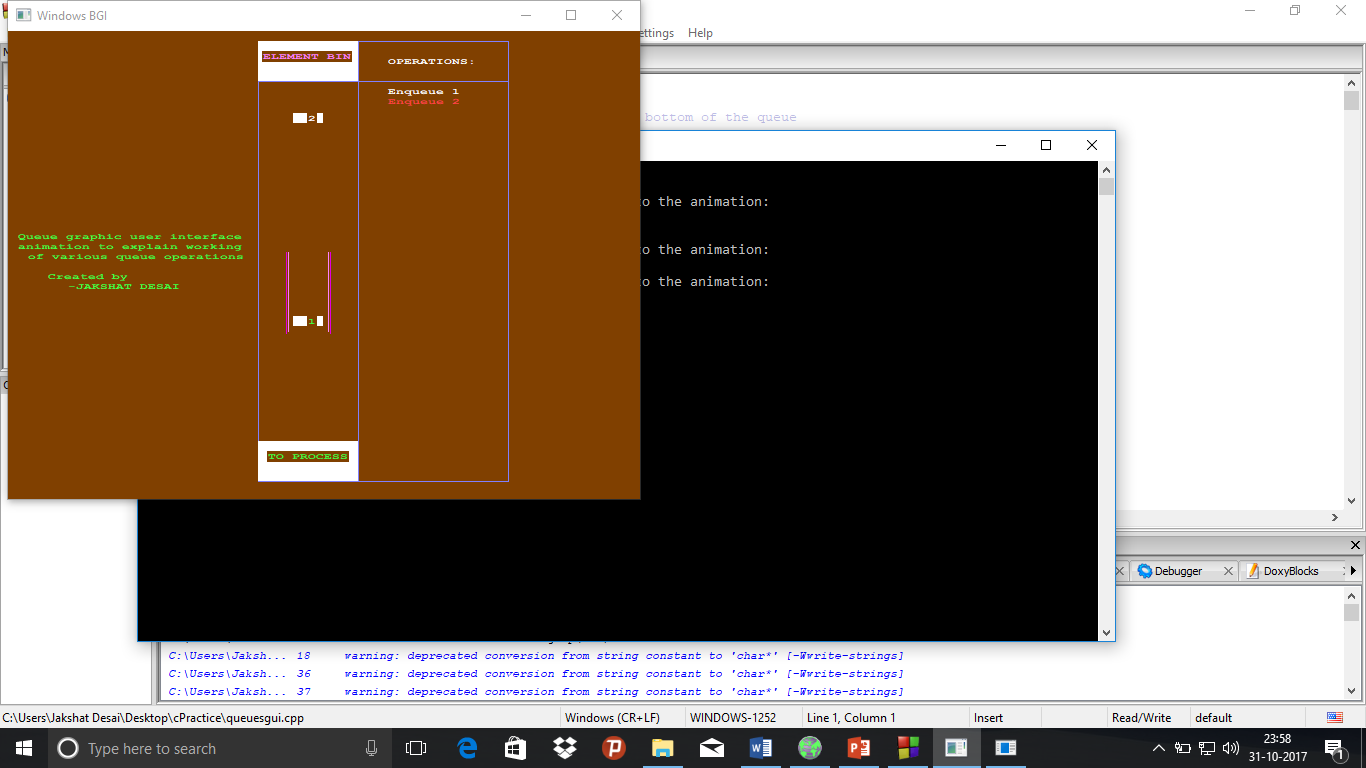
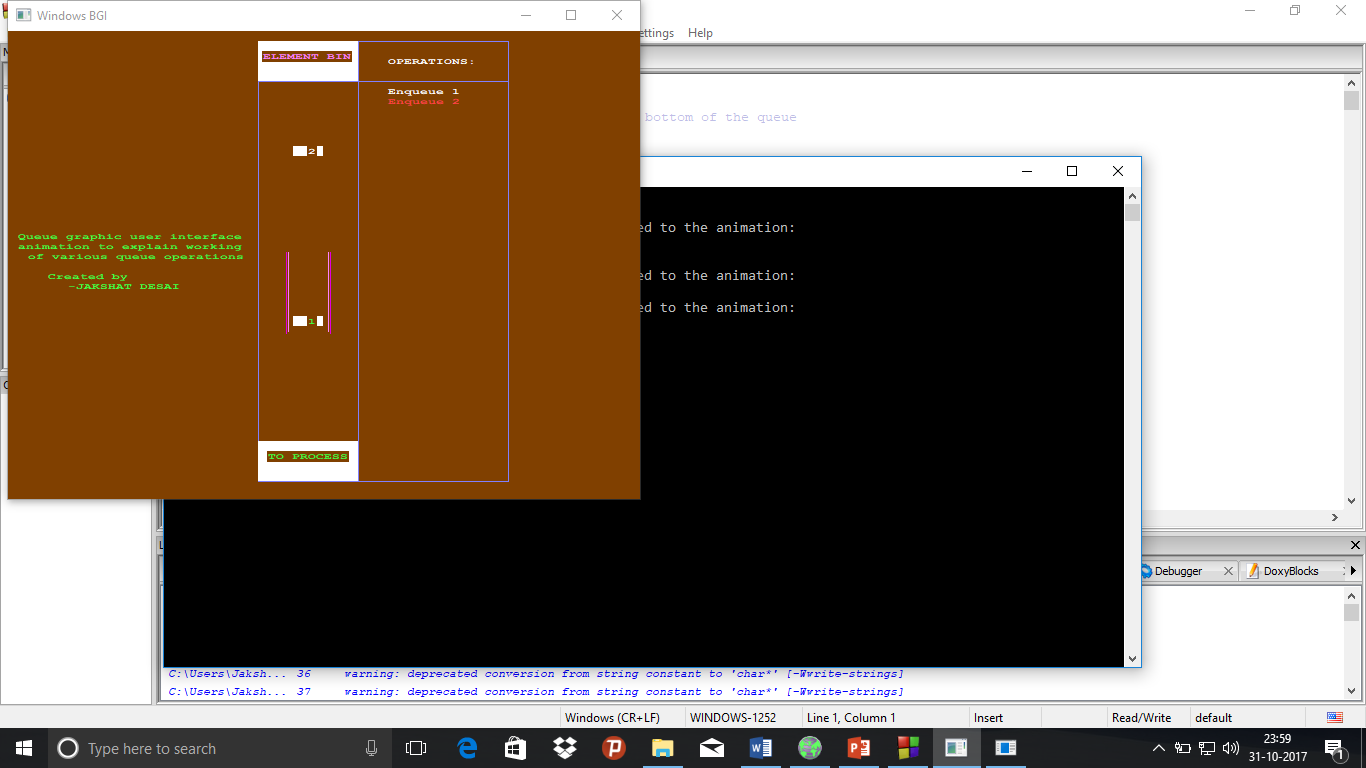
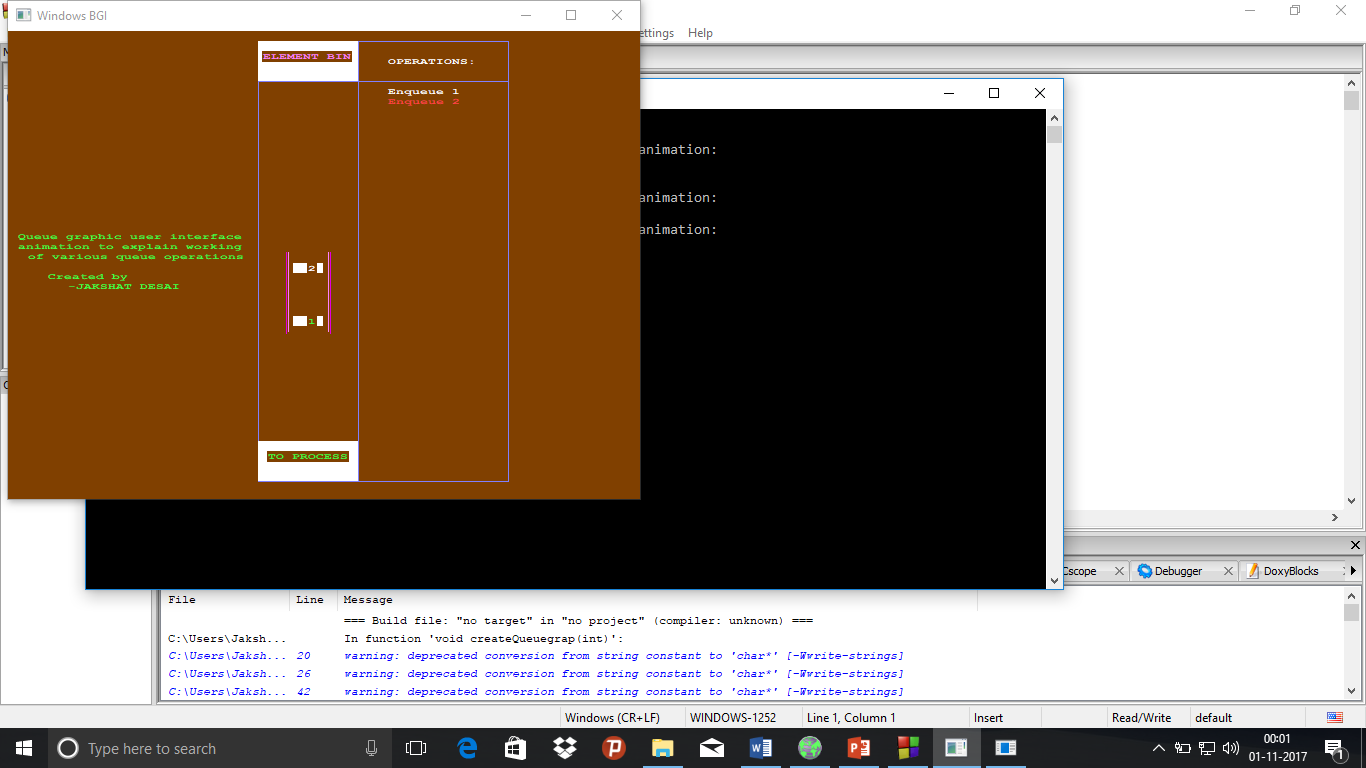
getch();

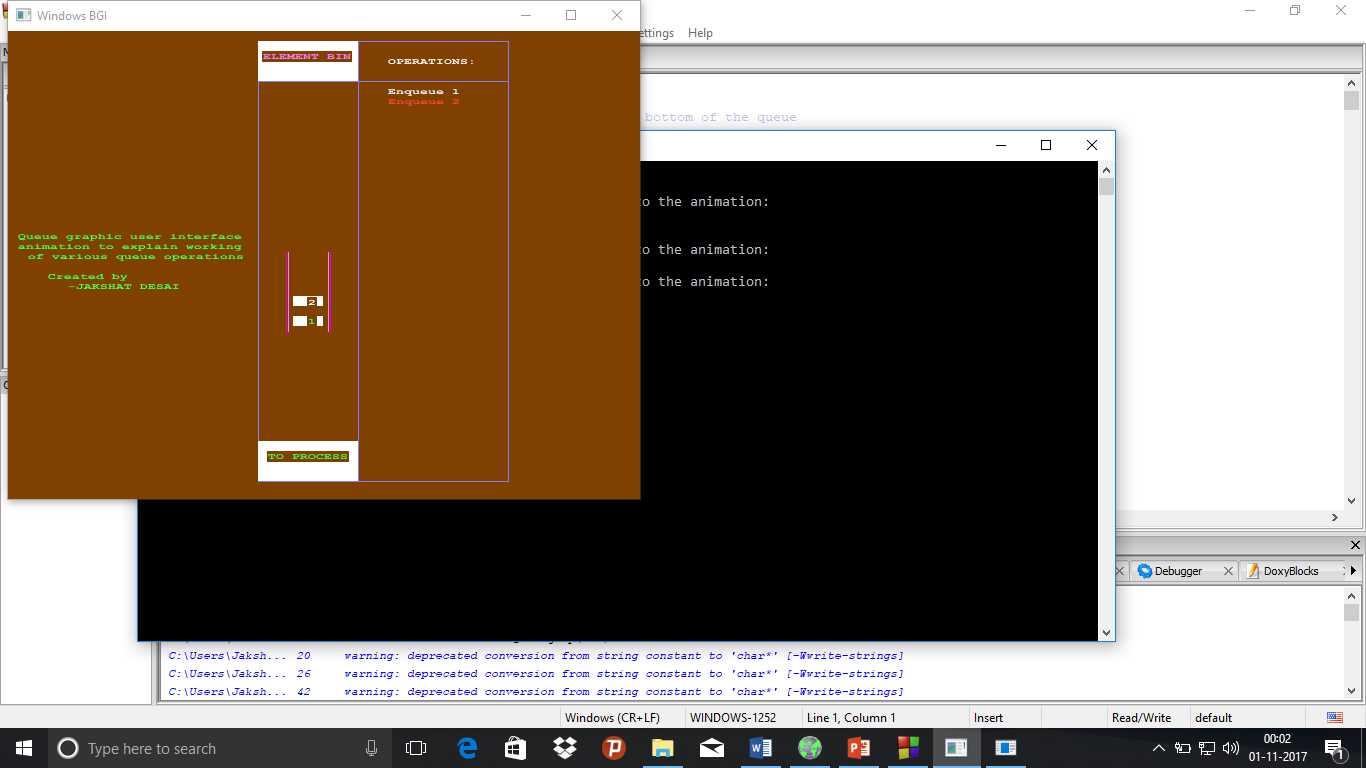
//closes the graph window

closegraph();

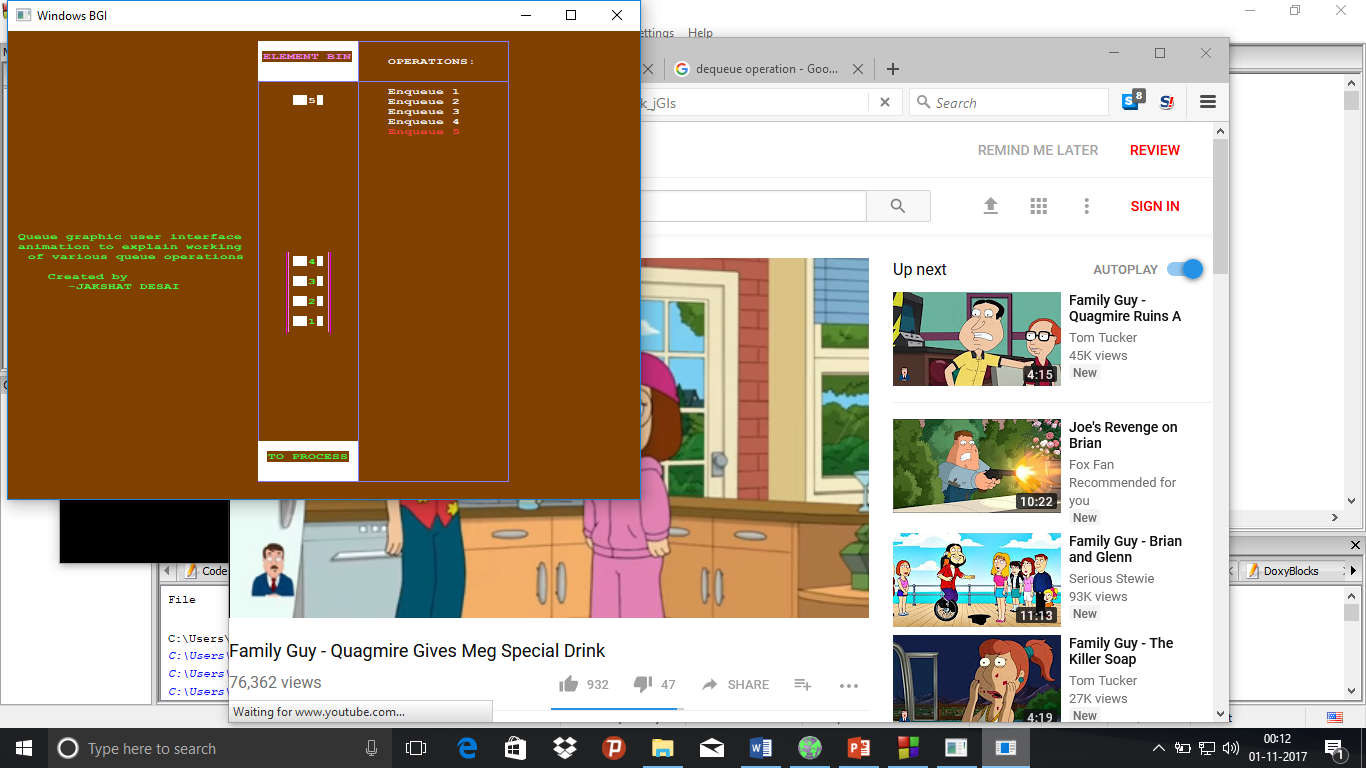
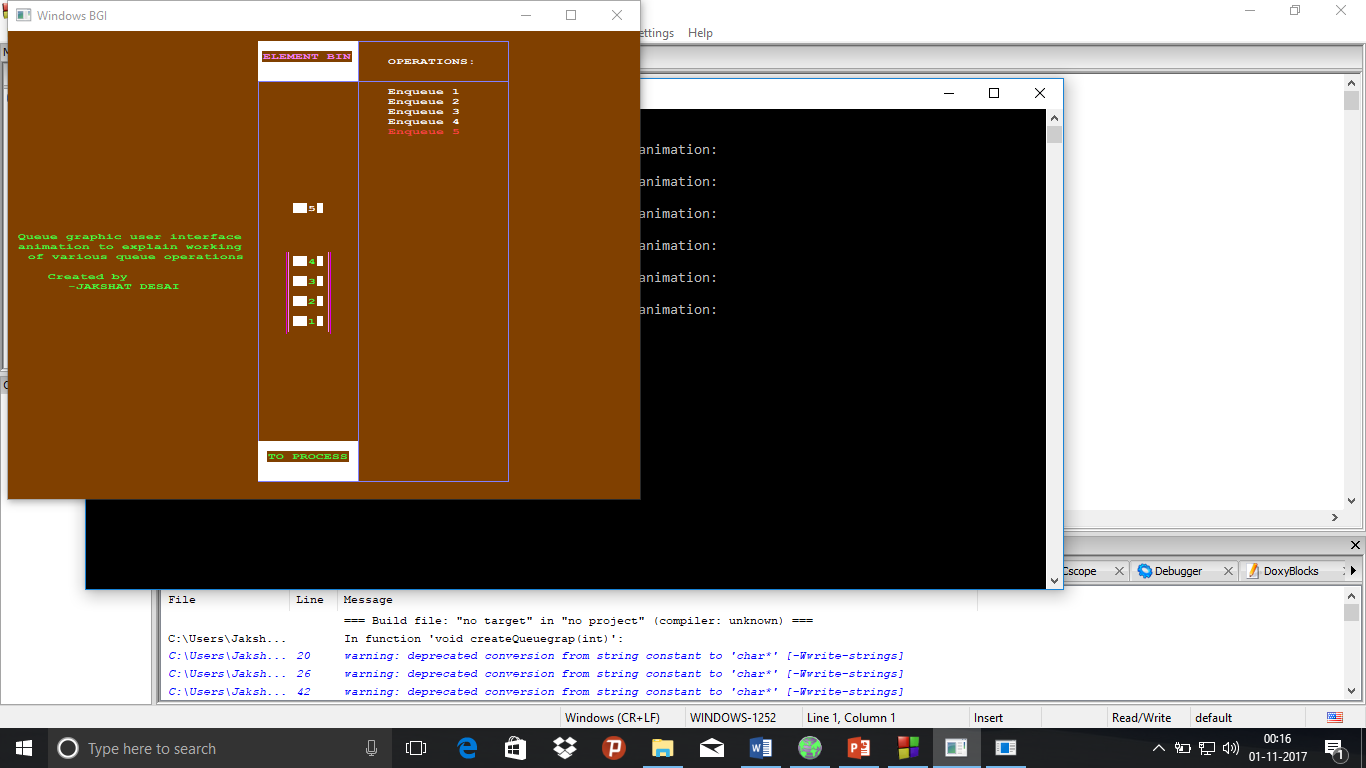
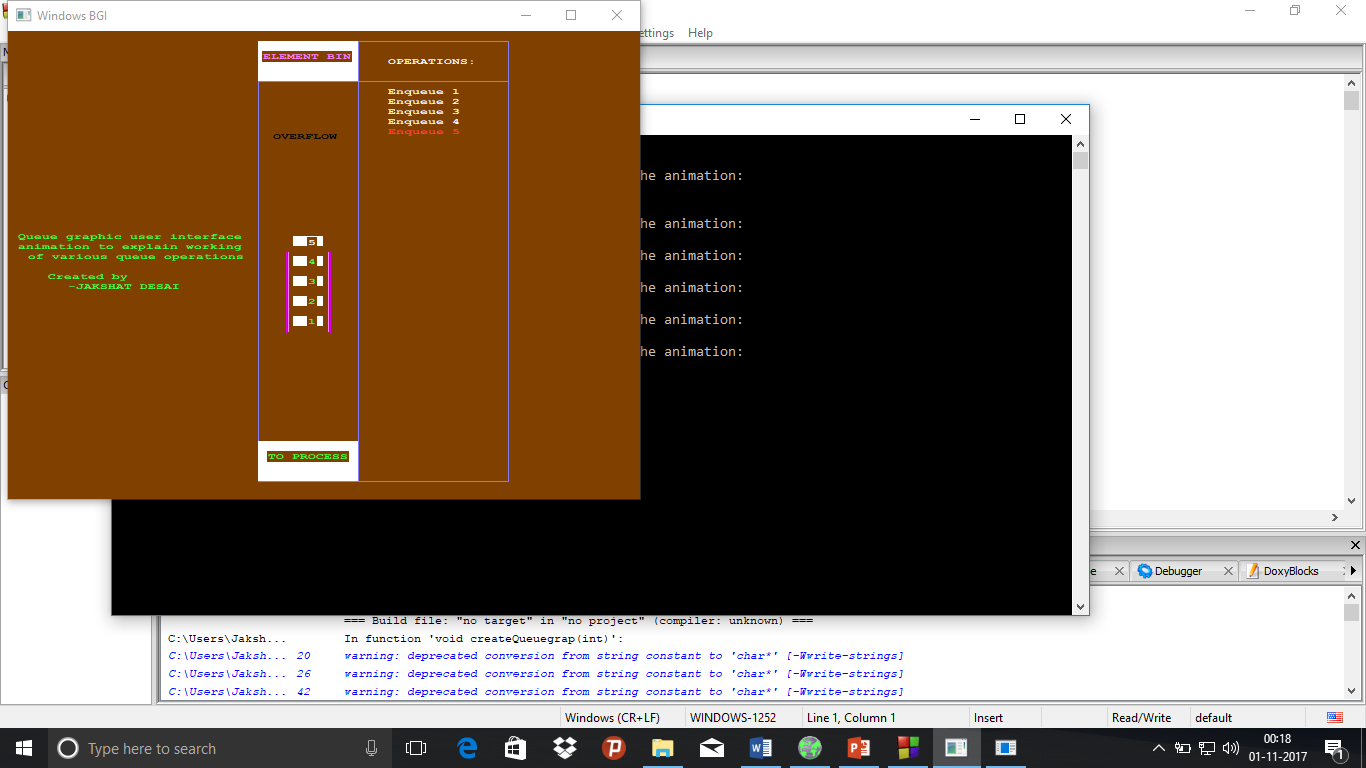
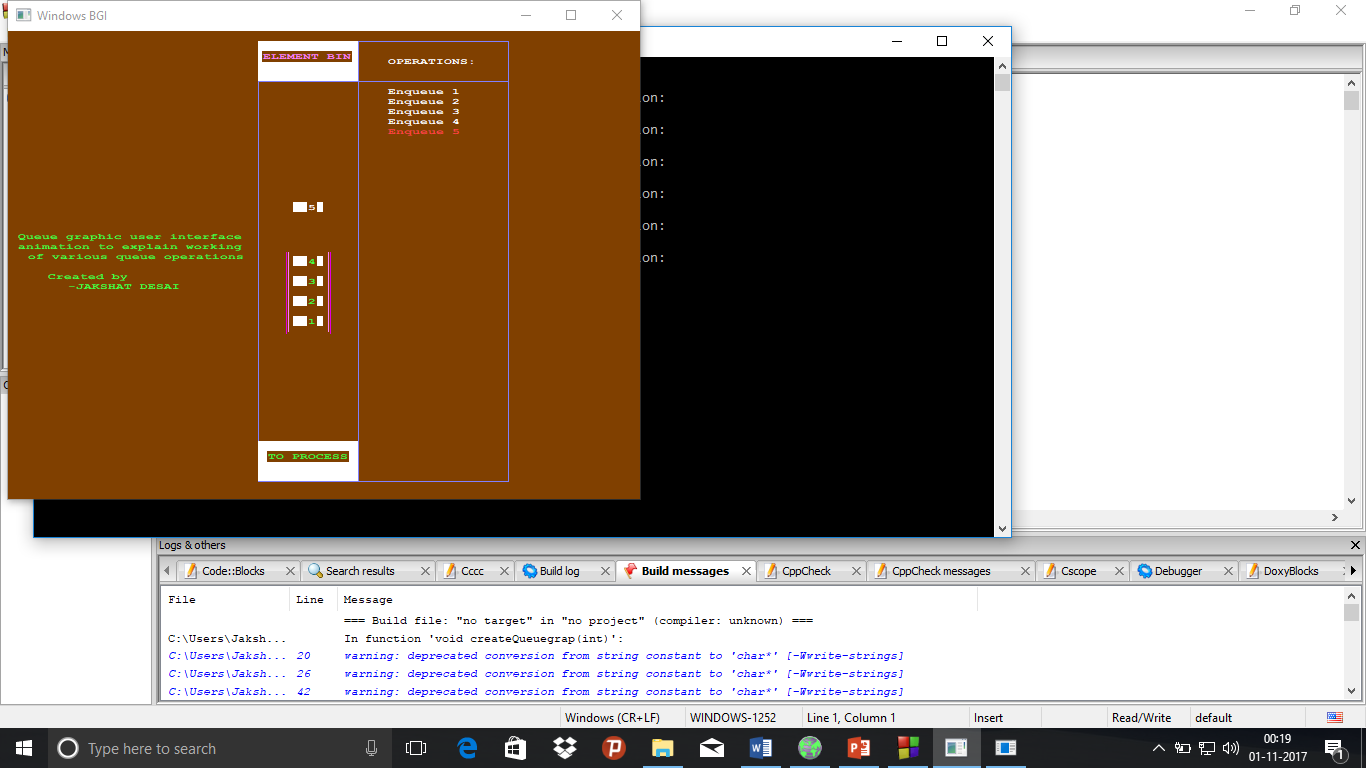
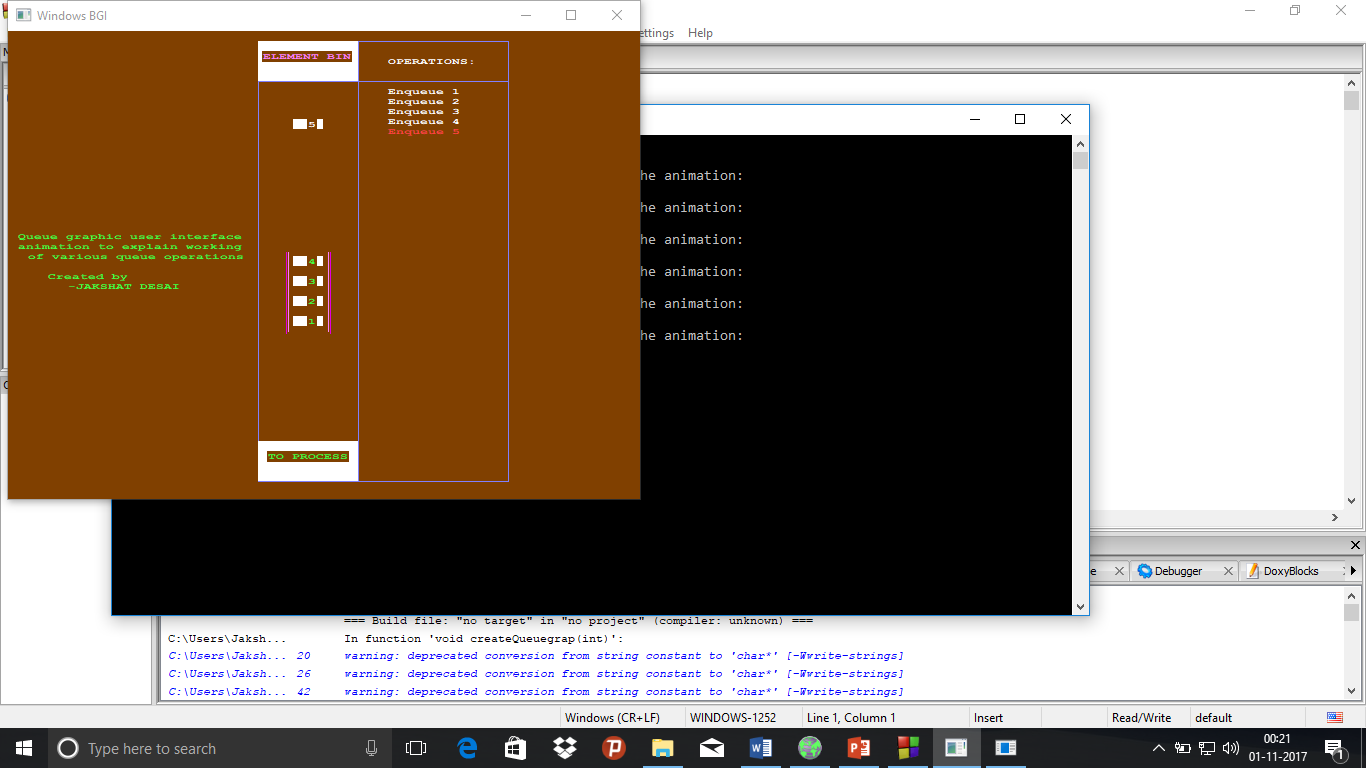
}

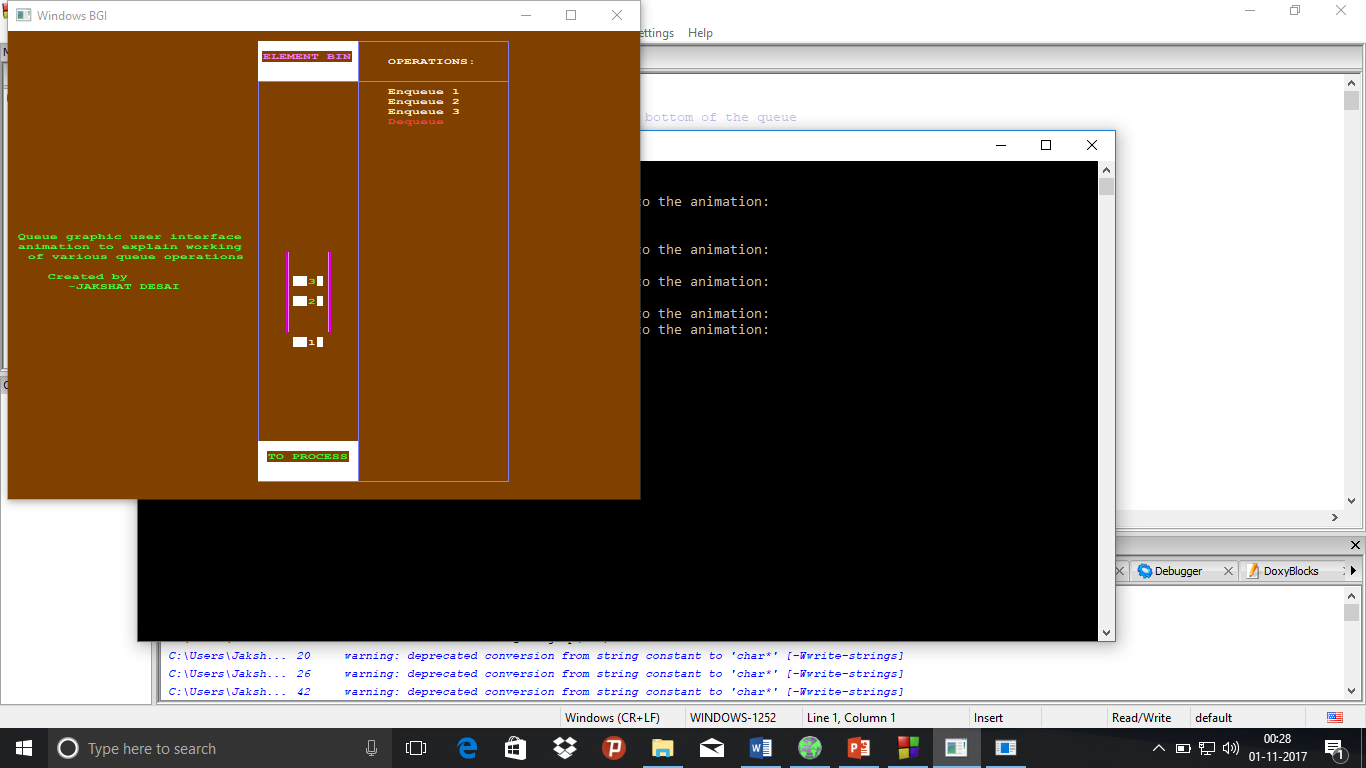
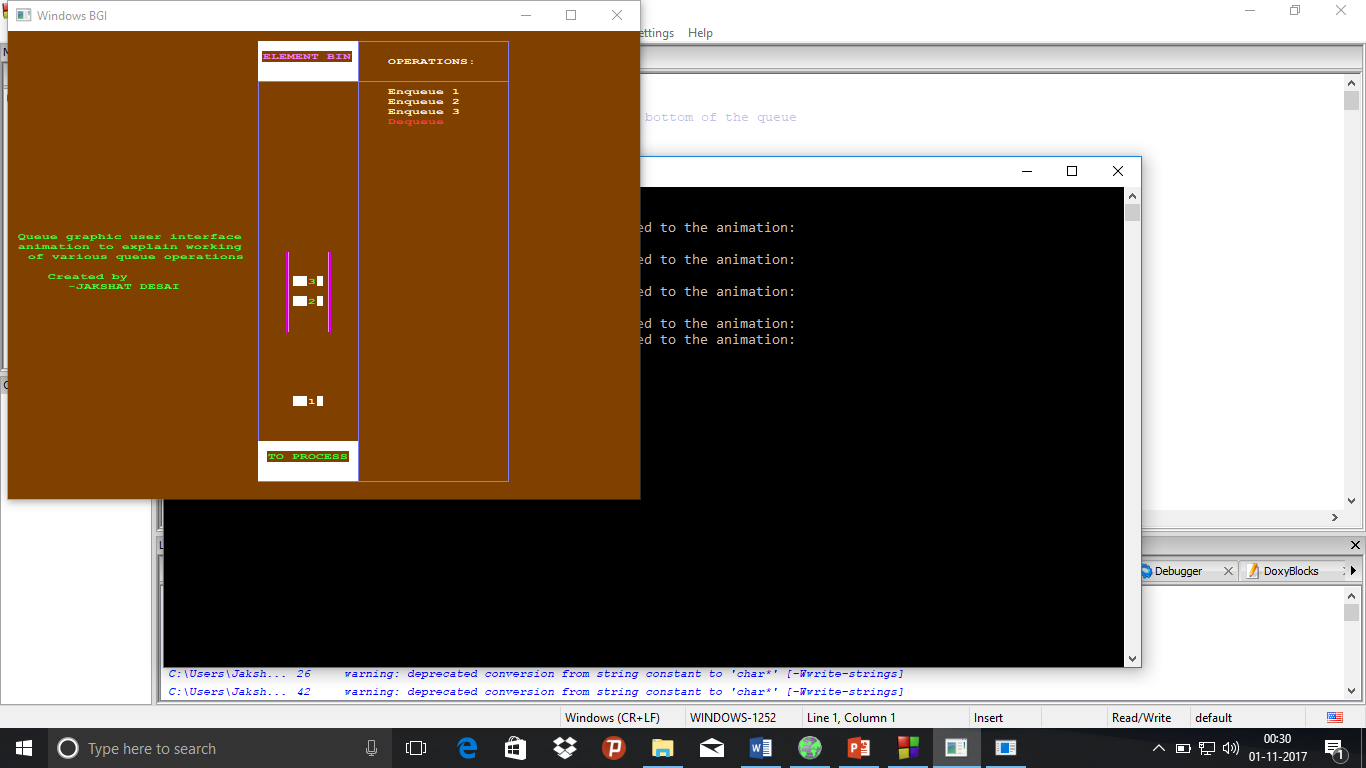
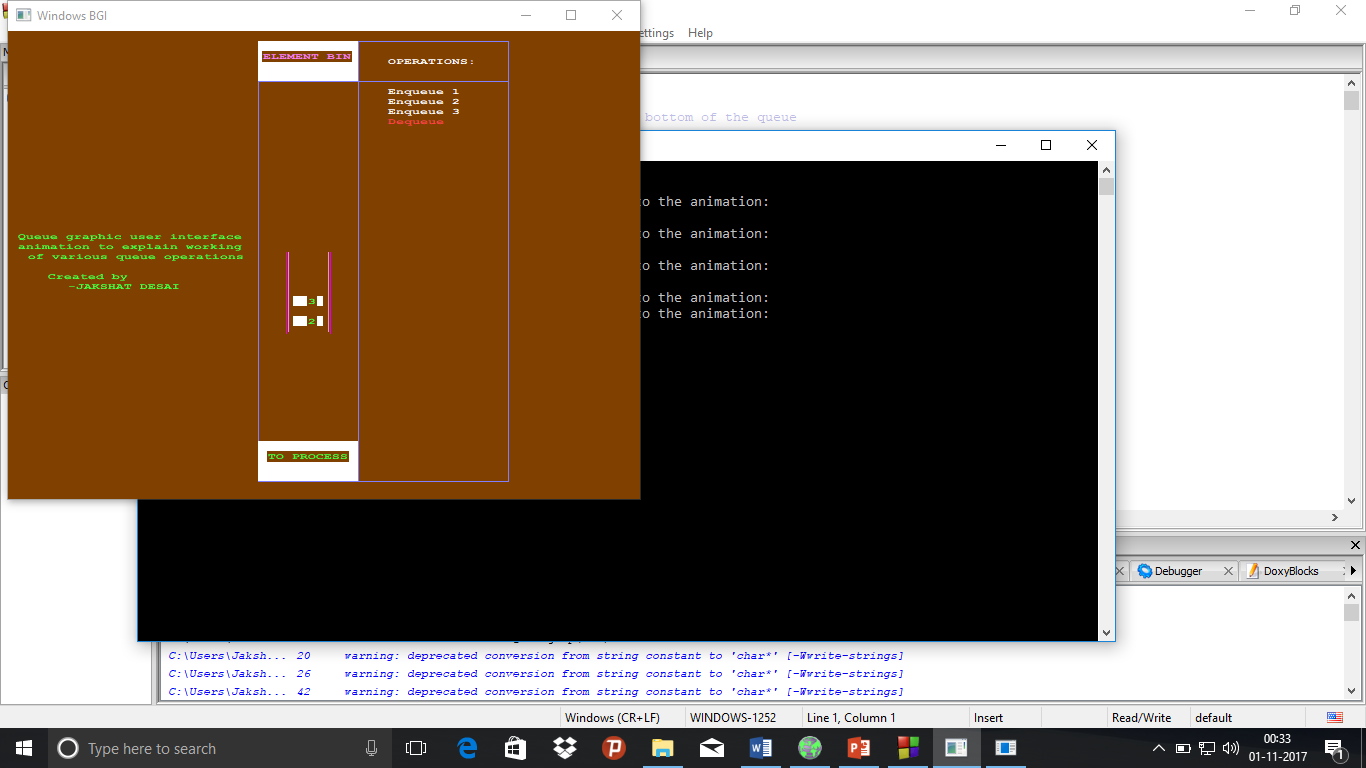
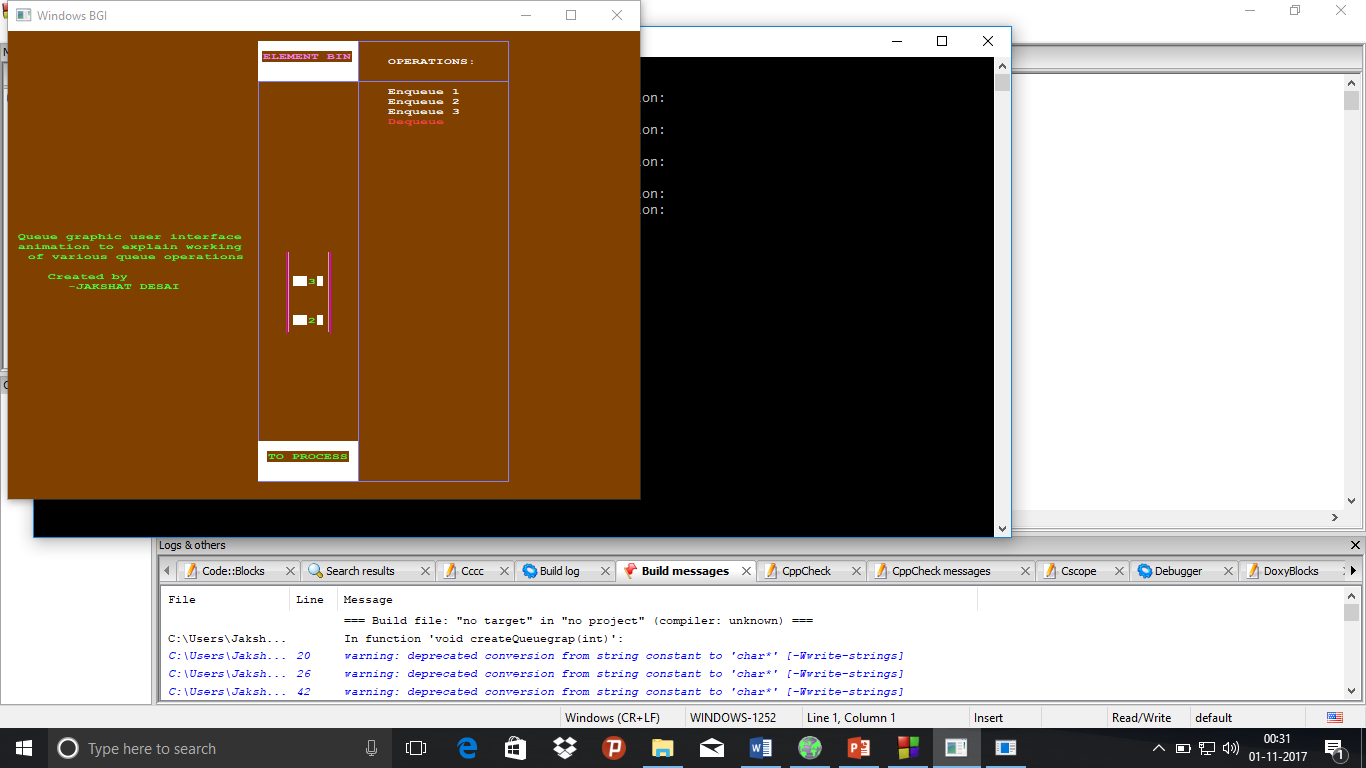
**Screenshots**

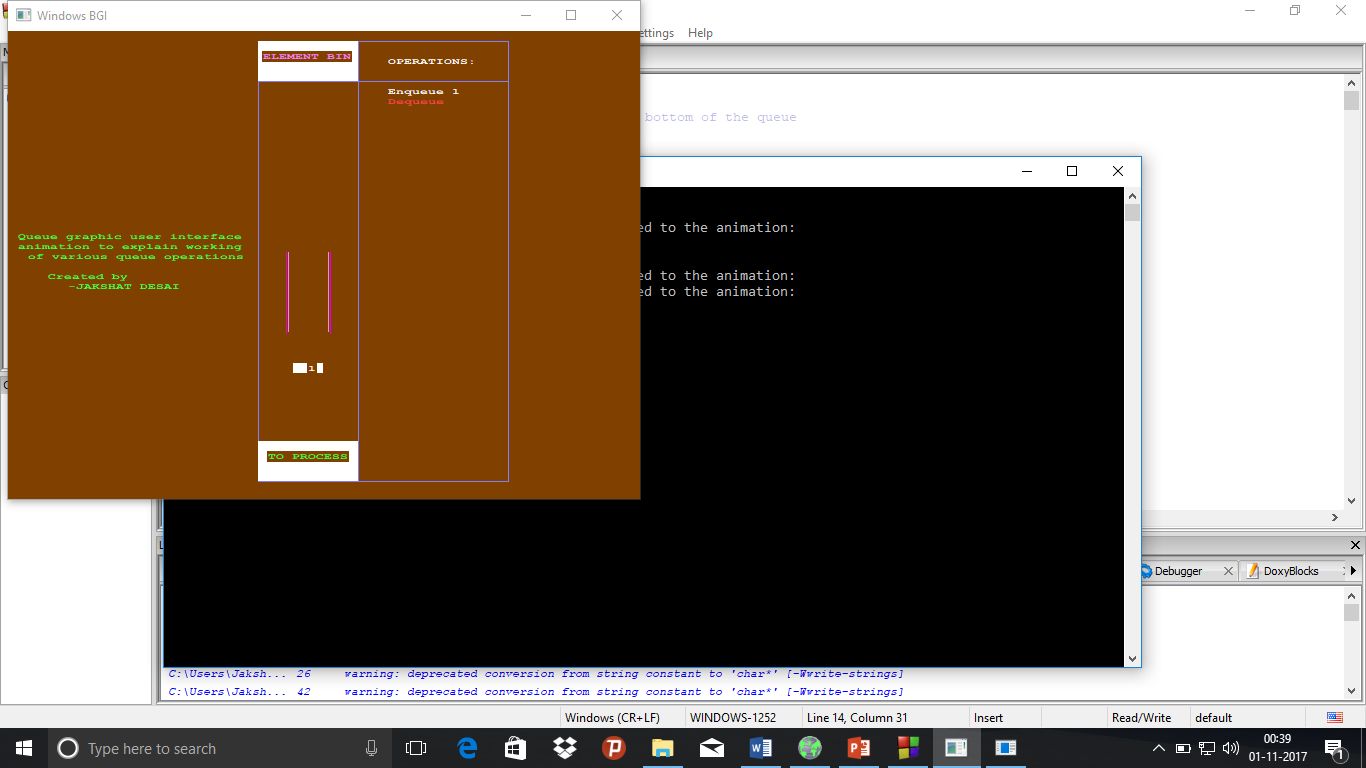
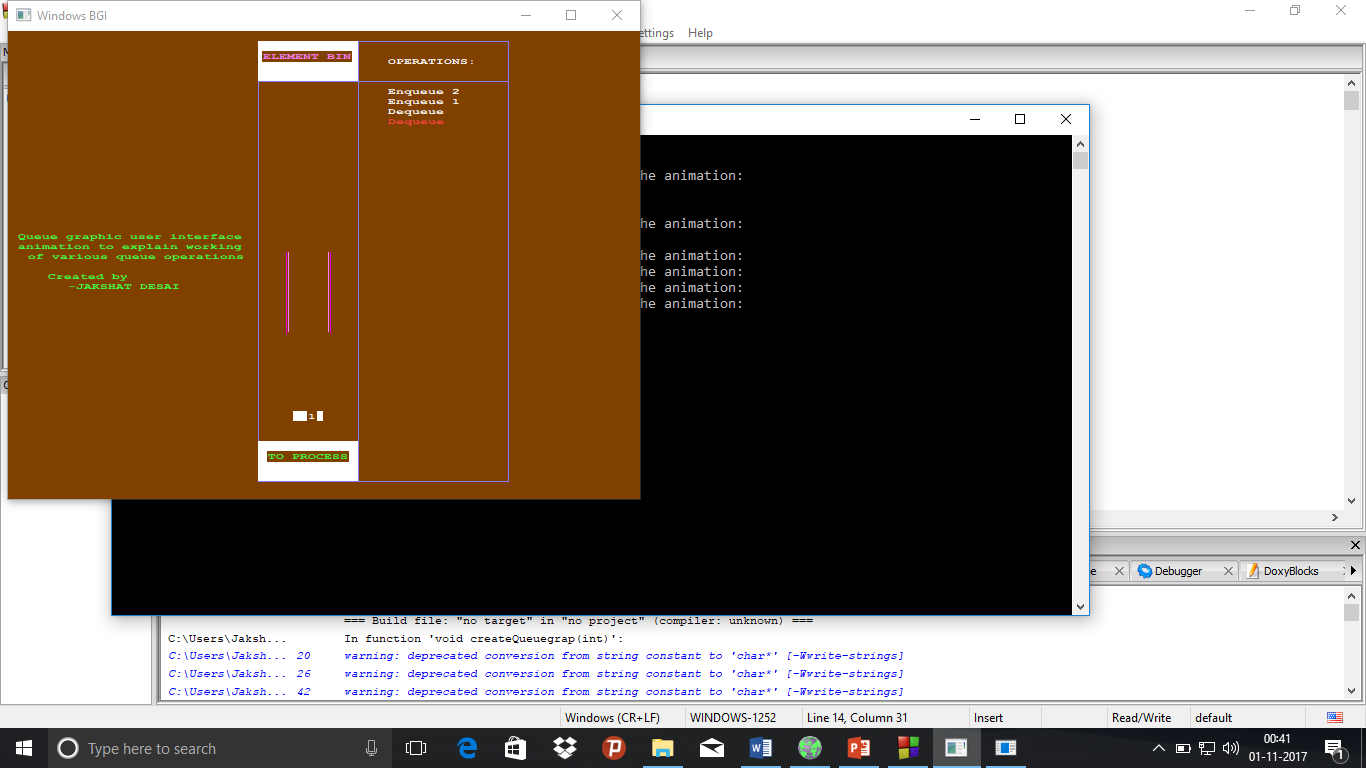
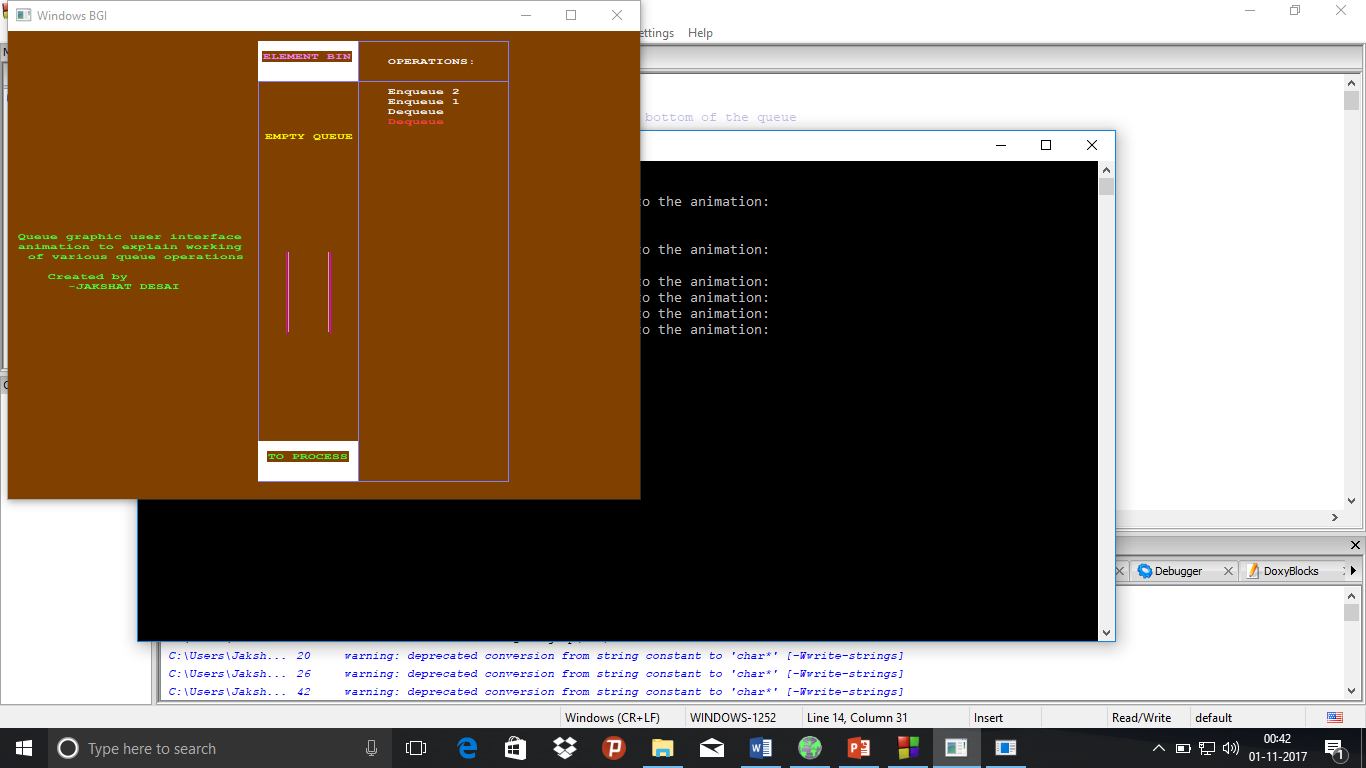
Enqueue operation:



Overflow condition:

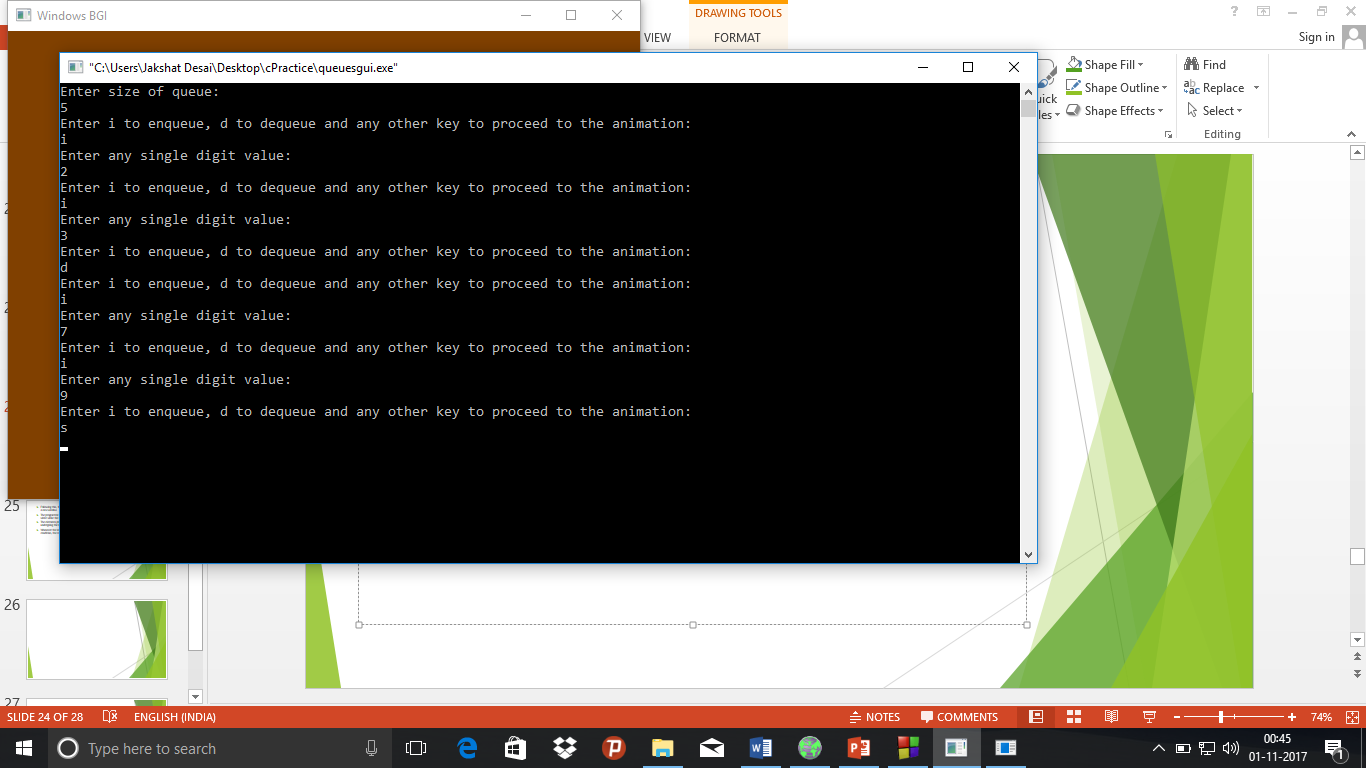
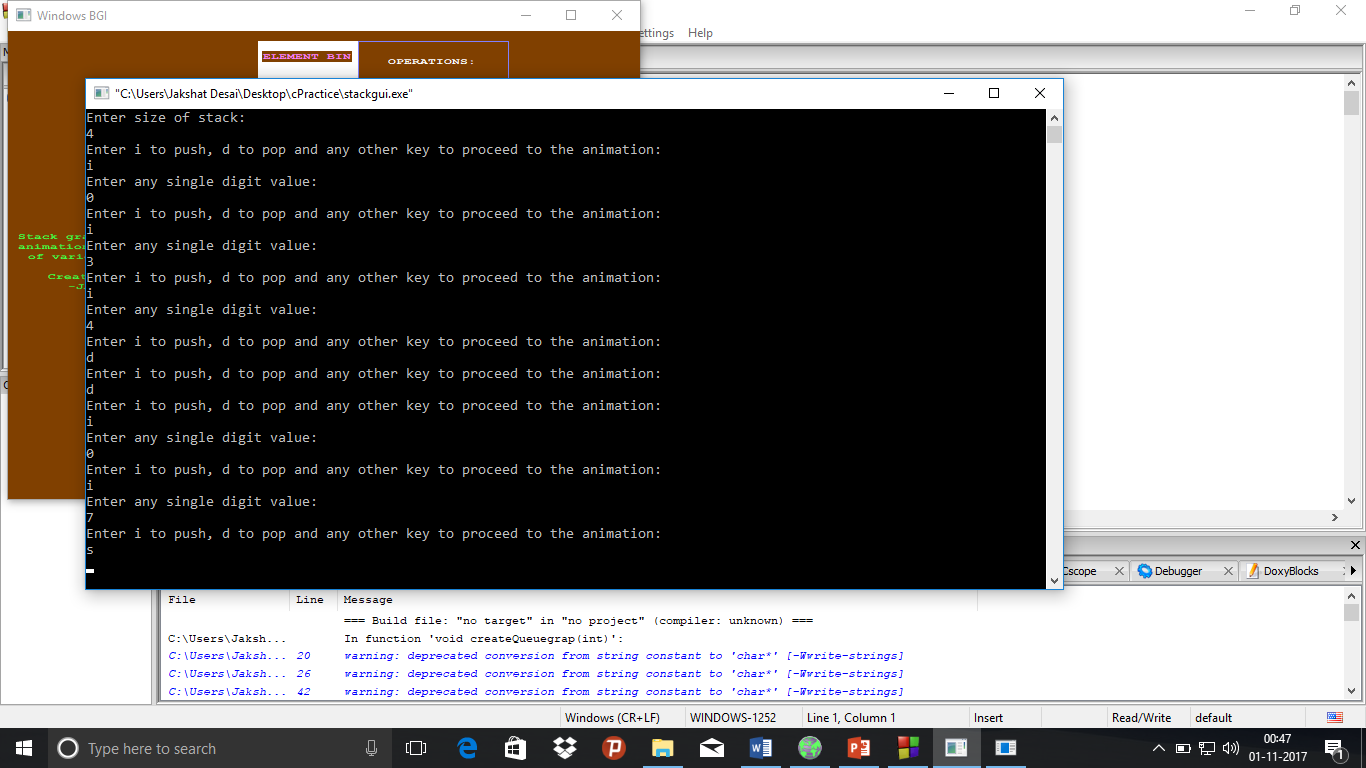


Dequeue operation:

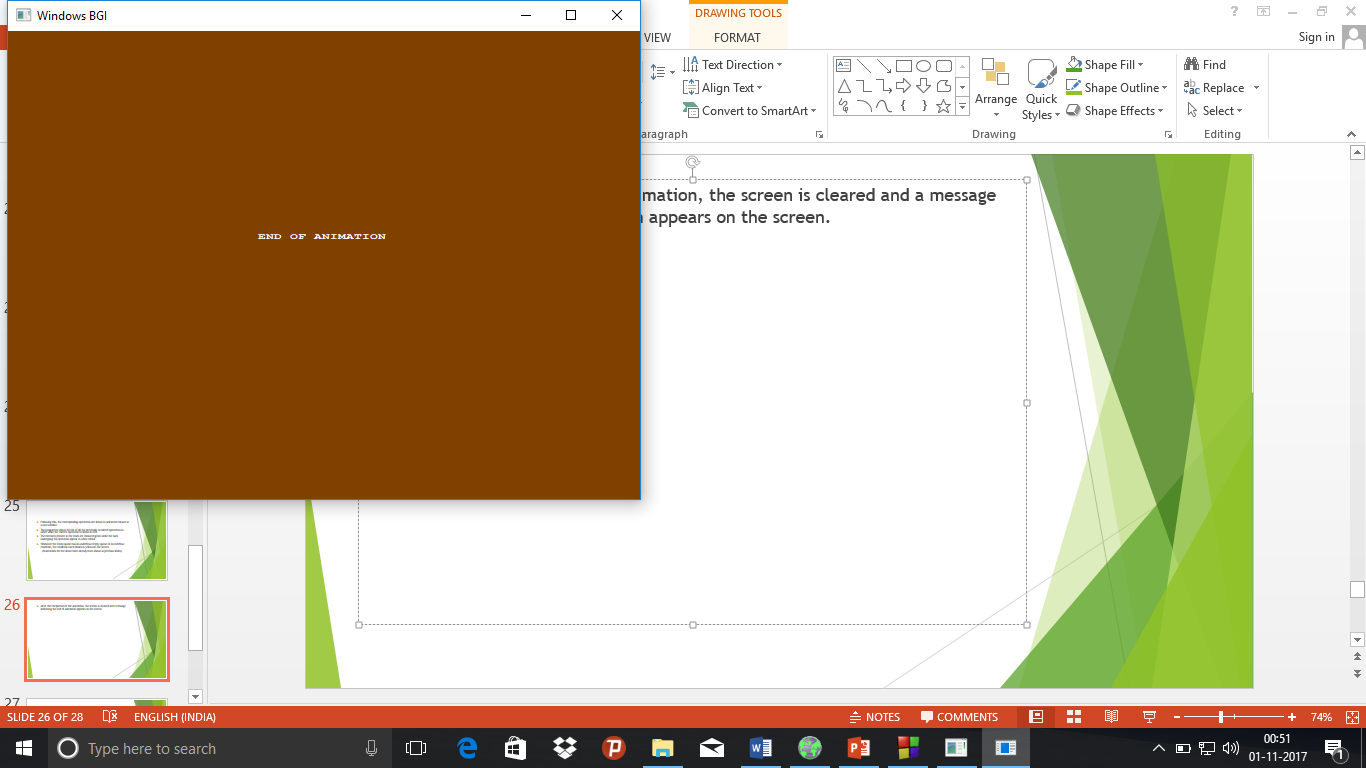
Empty queue/Underflow condition:

**Working of the programme**

* The programme(for both stack and queue) first asks the user for the size of the stack/queue.
* After this, the programme asks the user to enter the set of operations to be performed on the stack/queue.



* Following this, the corresponding operations are shown in animated manner in a new window.
* The programme shows the list of all the previously occurred operations in white while the current operation in shown in red.
* The elements present in the stack are coloured green while the ones undergoing the operation appear in white colour.
* Whenever the stack/queue has an underflow/empty queue or an overflow condition, the condition name blinks in yellow on the screen.
* After the completion of the animation, the screen is cleared and a message indicating the end of animation appears on the screen.



**Resources used**

* graphics.h header file, winbgim.h header file and libbgi.a
* Codeblocks 16.01
* Youtube:

1)Installing graphics in Codeblocks:

a)(Video tutorial)Best Programming Channel:

https://www.youtube.com/watch?v=Xkgg63TH6Og

b)(Download link)

https://drive.google.com/file/d/0ByyBxK\_syBJ-ZXdOMWVsdW1fTGs/view

2)Using graphics in C++:

VCR Games(Video tutorials): https://www.youtube.com/watch?v=cK9pmJdTK9Y&list=PL5UFsTza4wWSNhe0xuO6ELw7ORU-UHNDO

\* \* \* \* \* \*