Rafał Kubas

1.Task topic

Assignment 106 Graphs. A graph consists of vertices and edges connecting them. Write a program that operates on graphs and draws them on the screen. Details to be discussed.

2.Project analysis

First I had to figure out what way of saving data will be the best. My first assumption was using console window to getting data from the user, vertex by vertex. But then I realised that this idea is pretty bad. It's because console window doesn't work as great as it should in my opinion with graphic libraries. Especially when I tried to use WinApi programming. There was no way to use console window corectly before windows window implementation. After short reasearch I decided to save connections of vertices as neighborhood matrix. In my opinion it wasn't hard to get data this time from a .txt file and then convert it into 2D array. When I was finished with this part. I was looking for 'friendly' in use graphic library. I was trying in few like Allegro but whole programmer society were talking that this is old fashioned and out of use nowadays. Qt, Sdl and even graphviz which was created especially for graphs problem in my opinion wasn't 'it'. Then I found sfml with tutorials that's why I decided to work on it. 'Drawing' vertices with connections was divided into 2 parts. First, we obtain randomized coordinates of vertex then, using neighborhood matrix and array with coordinates. It was enough to know where to draw lines from which point. After that process, we could draw circles which represent vertices. Last step was drawing numbers on vertices. Important fact is that, graphs here are not directed.

3.External specification

To run program there are required additional file. First one is our defined graph. There is example how it should be implemented in graph_def.txt. But for sure I will make here other example:

```
1 3
2 5
3 5
2 4
3 6
```

It is important to not create empty lines, or not put additional enter at last row. We're defining connection between 2 vertices. It is required to have space between number of vertices. Another required fire is texture.png which is simply texture used in sfml to show vertex. Last file is arial.ttf used as font for numbers on vertices.

4.Internal specification

Almost everything is commented, that's why I will include only most important parts

class graph -class of our graph, it consists of 2D dynamically allocated array and size of this array.
tab[i] = new int[length]; - constructor for array
~graph() destructor
void graph::print() examplary defining method out of class, in this case it's for showing 2D array.
f**stream graph_def** - file operator fstream to point for file
graph_def.open("graph_def.txt") defining which file will be included
getline(graph_def, line_buf); getting line from file into string buffer
istringstream iss(line_buf); starting convertion from string to int
while (getline(iss, token, ' ')) getting sign til space
buf=atoi(token.c_str()); takes string from token and convert it into int for variable buf
buf=atoi(token.c_str());
vector <sf::CircleShape> vertices; creating container for circle shapes
vector <sf::Text> numbers; creating container for numbers
vector <sf::VertexArray> connect; creating container for connections
sf::Font font; sfml function for font implementation
randposX=(int)(rand() / (RAND_MAX + 1.0) * 950); X coordinate for vertex, similar for Y coordinate but with different range (it depends od size of window)
vertices.push_back(vcircle); -adding already created object (vcicrle) into container vertices
ss.str(""); cleaning string buffer
sf::VertexArray lines(sf::Lines, 2); function which draws lines. Later there was defined position of it by implementation lines.position
Window calling is commented in source code already, there is no need to copy it here

5.Source Code

```
#include <iostream>
#include <string.h>
#include <string>
#include <stdio.h>
#include <sstream>
#include <stdlib.h>
#include <fstream>
#include <cstdlib>
#include <cstdio>
#include <ctime>
```

```
#include <SFML/Graphics.hpp>
#include <vector>


#define C4996
using namespace std;

class graph //class of our graph
{
private:
   int size;
public:
   int **tab; //matrix neighbours
      graph(int length) //constructor
      {
      tab=new int *[length];
      for(int i=0; i<length; i++)
         tab[i] = new int[length];
         size=length;
      }
     ~graph() //destructor
      {
         for(int i=0; i<size; i++)
            delete[] tab[i];

            delete[]tab;

      }
      //Few methods
      void print(); //showing matrix of neighbours
      void set_zero(); //setting all entrances as 0
      void show_direction(); //better version of print();
      void edge_set(int x, int y);
      int degree(int vx); //showing degree of vertex (not used)
};
      void graph::print()
          {
            for(int i=0; i<size; i++)
               {
                  for(int j=0; j<size; j++)
                     {
                        cout<<tab[i][j] ;
                     }
```

```
                cout<<"\n";
            }

        }
    void graph::set_zero()
        {
            for(int i=0; i<size; i++)
                {
                    for(int j=0; j<size; j++)
                        {
                            tab[i][j]=0;
                        }
                }

        }
    void graph::edge_set(int x, int y)
    {
            tab[x][y]=1;
    }
    int graph::degree(int vx)
    {
        int deg=0;
        for(int i=0; i<size; i++)
            deg+=tab[vx-1][i];

                return deg;
    }


      void graph::show_direction()
         {
            for(int i=0; i<size; i++)
                {
                    cout<<i+1<<":  ";
                    for(int j=0; j<size; j++)
                        {

                            if(tab[i][j])
                                cout<<j+1<<" ";
                        }
                        cout<<endl;
                }

        }
```

```
int main()
{      //FILE OPERATORS
       fstream graph_def;
       graph_def.open("graph_def.txt");
       //INT FOR ROW COUNTING
       int LineNumb=0;
       //String to get vertices
       string line_buf;
       //Getting number of connection one by one
       string token;
       int buf;

       //Using highest number to determine number of vertices
       int VecNumb=0;
       int guard=0;
   if(graph_def.good())
   {
          //COUNTING LINE NUMBER
       while(!graph_def.eof()) //while file is not ended
       {
                getline(graph_def, line_buf); //getting line from file
                LineNumb++; //increasing integer which represents number of
line
                istringstream iss(line_buf);
                        while (getline(iss, token, ' '))
                        {
                           buf=atoi(token.c_str());
                              if(buf==0)
                                 {
                                 guard=1;  //variable used to check if number of
vertex starts with 0 or with 1
                                 }
                              if(buf>VecNumb)
                                 VecNumb=buf; //We know what is the number pf
vertices

                        }
       }
       //Creating object in class
          graph basic(VecNumb);
          basic.set_zero();
```

```
    //Rewind of file
       graph_def.seekg(0);

    int counter=0;
    int whichvertex;

     while(!graph_def.eof())
    {
                getline(graph_def, line_buf);
                istringstream iss(line_buf);
                            while (getline(iss, token, ' '))
                            {
                               buf=atoi(token.c_str());
                                  if(counter==0)
                               {
                                     whichvertex=buf;
                                     counter++;

                                  }

                            switch(guard)
                                     {case 0:
                                        basic.edge_set(whichvertex-1, buf-1);

                                        break;
                                      case 1:
                                        basic.edge_set(whichvertex, buf);

                                        break;
                                     }
                            }
          counter=0;

    }
    basic.show_direction();

    basic.print();
//GRAPHIC
    //texture for vertix
    sf::Texture vtexture;
    vtexture.loadFromFile("texture.png");
    //Coordinates of vertix
    int randposX;
    int randposY;
       srand(time(NULL));
```

```
//VERTEX


vector <sf::CircleShape> vertices; //creating container for circle shapes
vector <sf::Text> numbers;   //creating container for numbers
vector <sf::VertexArray> connect;  //creating container for connections
stringstream ss;   //string buffer
      int **positionstab=new int*[3];
         for(int i=0; i<2; i++)
            positionstab[i]=new int[VecNumb+1];  //creating array to save
positions of vertices

//loading font for numbers
sf::Font font;
      font.loadFromFile("arial.ttf"); //loading
      sf::Text mytext;



      for(int i=0; i<VecNumb; i++)
         {   randposX=(int)(rand() / (RAND_MAX + 1.0) * 950);
            randposY=(int)(rand() / (RAND_MAX + 1.0) * 750);
               positionstab[0][i]=randposX;
               positionstab[1][i]=randposY;



         //DRAWING VERTICES
         //parameters for circles
         sf::CircleShape vcircle(120);
         vcircle.setTexture(&vtexture);
         vcircle.setScale(0.1, 0.1);
         vcircle.setPosition(randposX, randposY);
         vertices.push_back(vcircle);
         //DRAWING NUMBERS ON VERTICES

               ss<<i+1;
                  //Parameters for text
                  mytext.setFont(font);
                  mytext.setCharacterSize(15);
                  mytext.setStyle(sf::Text::Bold);
                  mytext.setColor(sf::Color::Black);
                  mytext.setPosition(randposX+5, randposY+5);
                  mytext.setString(ss.str());
                  numbers.push_back(mytext);
                        ss.str("");
```

```
   }
        for(int i=0; i<VecNumb; i++)
       cout<<positionstab[0][i]<<" "<<positionstab[1][i]<<endl;

       sf::VertexArray lines(sf::Lines, 2);

           for(int i=0; i<VecNumb; i++)
              {
                 for(int j=0; j<VecNumb; j++)
                    {
                       if(basic.tab[i][j] && j!=i)
                       {
                          lines[0].position = sf::Vector2f((positionstab[0][i]+5),
(positionstab[1][i]+5));
                          lines[1].position = sf::Vector2f((positionstab[0][j]+5),
(positionstab[1][j]+5));
                          connect.push_back(lines);
                             //creating connections
                       }


                    }
              }



 sf::RenderWindow window(sf::VideoMode(1000, 800), "My window");

   // run the program as long as the window is open
   while (window.isOpen())
   {
      // check all the window's events that were triggered since the last iteration
of the loop
      sf::Event event;
      while (window.pollEvent(event))
      {
         // "close requested" event: we close the window
         if (event.type == sf::Event::Closed)

            window.close();

      }

      // clear the window with black color
```

```
        window.clear(sf::Color(62, 62, 62));

          for(int i=0; i<LineNumb-1; i++)
        {
            window.draw(connect[i]); //drawing connections
        }

        for(int i=0; i<VecNumb; i++)
        {
            window.draw(vertices[i]); //drawing vertices
            window.draw(numbers[i]); //drawing numbers

        }


        window.display();
    }
    exit(0);

    //Free memory
for(int i=0; i<VecNumb; i++)
        {
            delete[] positionstab[i];
        }
            delete[]positionstab;

        connect.clear();
        numbers.clear();
        vertices.clear();
        graph_def.close();






    }
    else
    {
     cout<<"FILE DOESN'T WORK";
    }

return 0;
}
```
6.Testing

Testing was based on many examples, self invented graphs. There is nothing hard to draw graph therefore testing here was nothing special. First I was testing if data from graph_def is saved correctly and if it's correctly put in 2D array. When I noticed everything was correct. Next testing was after using sfml 2.0 library. I will here few screens of results. They can be found in folder testing, jpg with numbers from 1 to 4. From simple graphs to more complicated. I won't include it here

7.Conclusions

Programm basically is finished. It draws graphs as it's supposed. But there are many other things which can be implemented to increase speed of drawing + other defining lines as special class not using sfml basic function. There could be also something like drag and drop vertex using mouse (few events in sfml) but I decided that it is not that important. I really became interested in sfml library and decided to work on it for a while to learn something new in c++. It's like new point of view in my programming course. That's why I decided to end with these graphs and try to write little game like pacman or something similiar.