Gliwice, 05.05.2019

Semester: 2

Group: 2

Section: 2

COMPUTER PROGRAMMING LABORATORY

Author: Krzysztof Sauer

E-mail: sauer1999@hotmail.com

Tutor: Piotr Fabian

1. Task topic

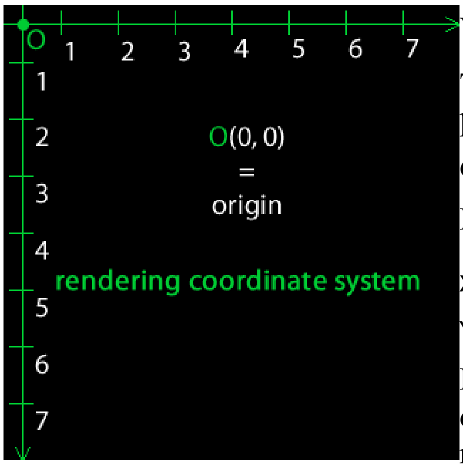
The task was to design and implement a program which would draw a chart of a function using C and/or C++ language. It could be done using Allegro library or any other library which was available.

2. Project analysis

This project was quite challenging due to many problems which occurred while writing the program. Since a chart of the function had to be displayed, I have decided to do that with the help of SFML library, since I had some experience with it already. I decided to use classes to store the information about the function in particular point (its called Function). Moreover, I have asked my tutor, dr. ing. Piotr Fabian how the function was supposed to be specified, whether it was going to be pre-compiled function or a user-defined one. It turned out that it would be good it was an user-defined function. In order to be able to do that, tutor have sent me a phaser code, which was automatically calculating the expression value. It was quite complex, so It took me a few hours to work out how does it work and what it can (and can't) do. This code (which I have moved to evaluate.h) accepted a string as an input, whereas input could be any user defined function, like sin(5)+cos(2*pi). However, the first problem that occurred was that this code accepted only values. No unknown variables were allowed. In order to fix that, I had to write a function which would substitute a variable for each value which was going to be calculated. In order to do that, each variable had to be converted to use std::string replace() function, for each occurrence of "x" (or "X") and moreover a double negative had to be replaced too, since phaser was not able to interpret that. All of that had to be performed about 1600 times. After these initial opeations, x and obtained y value was saved into a vector ArrayofVariables(1600) which was a vector of class Function, consisting of 5 variables:

x, y ,Xdisp, Ydisp, Ydisp2

where Xdisp or Ydisp stands for X or Y which was going to be displayed by SFML, and Ydisp2 is a variable which will be scaled from Ydisp, so it can actually be visible. Without it, the displayed function would be very small.



Why are X/Ydisp values needed?

The answer is that the SFML can treat them as a point (x,y) however it will not be interpreted correctly, since SFML uses different coordinate system for rendering.

Because of that, such funtions had to be implemented:

```
Xdisp = x + (Width of the window/ 2)
Ydisp = (Height of the window/ 2) - y
```

Furthermore Ydisp2 had to be scaled up, because sometimes the difference between Ydisp values were too small to be actually readable. We would only see a straight line.

2. External specification

To use the program, please run the evaluate.exe file.

You will be asked specify how many pixels on screen will be an equivalent of value 1 on the X plane. If you do not know what value you want to set, please set the recommended value of 50. It will mean that 50 pixels on screen will be an equivalent of 1 on a X-plane. Moreover every 50 pixels will be a mark on a X axis which indicates full X values (-3,-2,-1,0,1 ...), like on normal coordinate system. If you wish to change this value, please restart the program.

Later, you will be asked if you want to display the „debug values". These values are actually the values calculated at each point of the window, so if you are curious, fell free to enable this function by entering 1, [Y] or [y]. Otherwise please enter 0 or [N] or [n]

Afterwards you will be asked to specify the function. Please enter a function with X value as unknown variable eg. sin(x) + cos(x) , 2^x etc…

If nothing is displayed or the program crashed, you propably made a mistake when entering the function. Please make sure that you do not miss parenthesis. If the cause remains unknown, please check the debug values in the console for more information.

To specify a different funtion, firstly, please close the window on which the function is drowed.

WARNING: Do not specify tangent funcions since it will crash the program. Please use alternative function tg(x)=sin(x)/cos(x), ctg(x)=cos(x)/sin(x). The cause of the crash is unknown.

3. Internal specification

This program has been written on top of the project which evaluated a single function value, which has been sent to me. It is due to fact that for some unknown reason, Visual Studio did not let me to compile the phaser if it was copied to a different project. Therefore I was forced to write this program inside this project.

Basically the program consisted of a class:

```
const int WINX = 1600; //Width of the rendered window
const int WINY = 1000; //Height of the rendered window


class Function
{
public:
	double x, y, Xdisp, Ydisp, Ydisp2;
	double FXdisp(double);
	double FYdisp(double);
	Function();
	~Function();
};
```

```
Please note that FXdisp and FYdisp methods were not actually used. They were remains of a
previous concept of program.
```

And a few functions:

1) A function which replaced all occurances of string with another string

```
void findAndReplaceAll(std::string & input, std::string toSearch, std::string replaceStr)
{
	// Get the first occurrence
	size_t pos = input.find(toSearch);
	// Repeat till end is reached
	while (pos != std::string::npos)
	{
		// Replace this occurrence of Sub String
		input.replace(pos, toSearch.size(), replaceStr);
		// Get the next occurrence from the current position
		pos = input.find(toSearch, pos + replaceStr.size());
	}
}
```

2) The function `evaluate(TempConversion, &ArrayOfVariables[q].y, &a)`, which is a function which computed the value of specified equation at a X point (the TempConversion value) and later was saved to ArrayOfVariables[q].y, where this value was later converted to SFML displaying standard.

3) The part of program which evaluated the value of function at certain X point:

```
//These variables were required by phaser function
	int a, ec;
	char expr[1024];
//These values were required for the SFML library
	int Xshift = WINX / 2;
	int shrink = 50;
	int size = (WINX);
	…


		// Replace X's in string with values and putting them into array

			for (double j2 = ((-WINX / 2)); j2 < (WINX / 2); j2++)

			{
				int q;
				double p = j2 / shrink; // 1 on xplane becomes wide 50 [px]

			// Q is j2 shifted by negative values of X axis
			// Otherwise, the reference index would be negative
				q = (j2 + ((Xshift)));

			// converts Xvalue to string
				std::string xval = std::to_string(p);
			// converts Equation to string
				std::string equation(expr);

			// Here the X'es are replaced with values
				findAndReplaceAll(equation, "x", xval);
				findAndReplaceAll(equation, "X", xval);

			// Here double negative is replaced with plus sign
				replacedoublenegative(equation);
				if ((j2) > (WINX / 2))
					break;

			// Here the string is converted into C-string so it can be used by
			   pharser
```

```
                char TempConversion[1024];
                strcpy(TempConversion, equation.c_str());
                ec = evaluate(TempConversion, &ArrayOfVariables[q].y, &a);

         // The value from phaser is written into the class. Here also, the
            Xdisp and Ydisp values are calculated.
                ArrayOfVariables[q].x = (j2);
                ArrayOfVariables[q].y = ArrayOfVariables[q].y;
                ArrayOfVariables[q].Xdisp = (ArrayOfVariables[q].x + Xshift);
                ArrayOfVariables[q].Ydisp = ((WINY / 2) - ArrayOfVariables[q].y);
                if (expr[0] == 0)
                       break;
         // Noting minimum and maximum values (will be used later on)
                if ((ArrayOfVariables[q].Ydisp > Ymax))
                       Ymax = ArrayOfVariables[q].Ydisp;
         // Tries to remove infinite values to avoid errors in scaling
                if (ArrayOfVariables[q].Ydisp == INFINITY)
                       Ymax = 0;
                if (ArrayOfVariables[q].Ydisp == (-INFINITY))
                       Ymin = WINY;
                if ((ArrayOfVariables[q].Ydisp < Ymin))
                       Ymin = ArrayOfVariables[q].Ydisp;
         }
```

4) Part of program rescaling the function which was going to be rendered:

```
//checks if difference between Ymin and Ymax is in reasonable range. If it was always
  true, it would
double deltaY = abs(Ymax-Ymin);
if (deltaY <= WINY)
{
//checks how much value changed (for Ydisp above X-axis)
                     double deltaplus = (WINY/2)/(Ymin-(WINY / 2));
//checks how much value chandeg (for Ydisp below X-axis)
                     double deltaminus = (WINY / 2)/(Ymax -(WINY / 2));

//sometimes the scaling worked for positive values while for y<0 the scaling did not
work. These 4 functioncs make sure that this does not happen.
if (deltaminus >= WINY)
       deltaminus = -deltaplus;

if (deltaminus <= 0)
       deltaminus = -deltaplus;

if (deltaplus >= WINY)
       deltaplus = -deltaminus;

if (deltaplus <= 0)
       deltaplus = -deltaminus;

       for (int j2 = 0; j2 < WINX; j2++)
       {

       //Amplifies the value upwards
       if (ArrayOfVariables[j2].Ydisp > (WINY / 2))
              {
              double delta = (((WINY / 2) - ArrayOfVariables[j2].Ydisp)*9)/10;
ArrayOfVariables[j2].Ydisp2 = ArrayOfVariables[j2].Ydisp + delta * deltaplus;
              }

       //Copies the value if it’s at y=0
		if (ArrayOfVariables[j2].Ydisp == (WINY / 2))
	ArrayOfVariables[j2].Ydisp2 = ArrayOfVariables[j2].Ydisp;

		//Amplifies the value downwards
		if (ArrayOfVariables[j2].Ydisp < (WINY / 2))
			{
		double delta = ((ArrayOfVariables[j2].Ydisp - (WINY / 2)) * 9) / 10;
		ArrayOfVariables[j2].Ydisp2 = ArrayOfVariables[j2].Ydisp + delta * deltaminus;
			}

		}

	}
```

5) The part of the program which task was to render the results:

```
//Create the main window on which the function will be displayed
sf::RenderWindow window(sf::VideoMode(WINX, WINY), "Chart creator, version: 0.2 Alpha");

// Create a function line.
sf::VertexArray function(sf::LineStrip, size);
for (int i = 0; i < size; i++)
{
  function[i].position = sf::Vector2f(ArrayOfVariables[i].Xdisp, ArrayOfVariables[i].Ydisp2);
  function[i].color = sf::Color::Red;
}
//Create markers on a X-axis
sf::VertexArray xmark(sf::Lines, 2*(WINX / shrink));
for (int i = 0, j = 0,k = 0; j < (WINX / shrink)*2; i++, j=j+2,k++)
{
  xmark[j].position   = sf::Vector2f(0 + (j*shrink)/2, (WINY / 2) - 5);
  xmark[j+1].position = sf::Vector2f(0 + (j*shrink)/2, (WINY / 2) + 5);
}

//Create X axis line
sf::Vertex linex[] =
{
  sf::Vertex(sf::Vector2f(0,(WINY / 2)),sf::Color::White),
  sf::Vertex(sf::Vector2f(WINX, (WINY / 2)),sf::Color::White),
};

//Create Y axis line
sf::Vertex liney[] =
{
sf::Vertex(sf::Vector2f((WINX / 2), 0),sf::Color::White),
sf::Vertex(sf::Vector2f((WINX / 2), WINY),sf::Color::White),
};

	// main loop of the program, which is running while window is opened
	while (window.isOpen())
	{
		sf::Event event;
		//Event catcher
		while (window.pollEvent(event))
			{
				//Close the rendered window if X button is pressed
				if (event.type == sf::Event::Closed)
					window.close();
			}
	//Resets the window from previous frame
		window.clear(sf::Color::Black);
	//draws stuff in the window
```

```
        window.draw(function);
        window.draw(linex, 2, sf::Lines);
        window.draw(liney, 2, sf::Lines);
        window.draw(xmark);
        window.display();
}
```

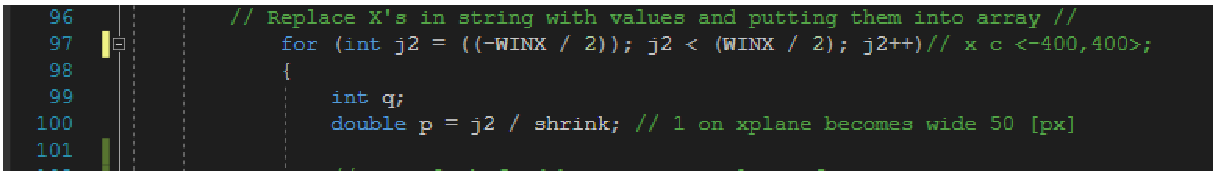
4. *Source code*

The source code has been included along with this file (actually, entire project was included). The main code of the program is included in Source.cpp where the main code of the program is, and evaluate.h where the code of the phaser is.

5. *Testing*

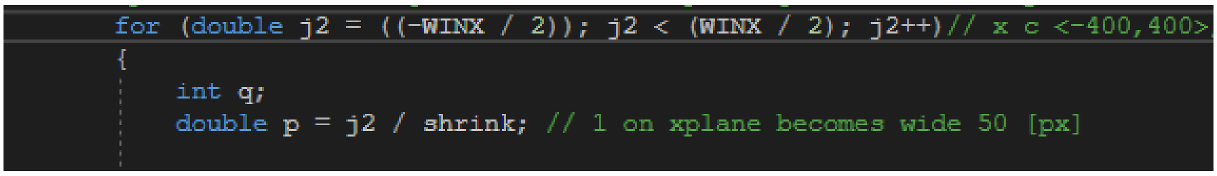
Many trials were made in order to make the program work correctly. At the very beginning my concept was in fact completly different than on the final result. Most of the errors were a result of tiredness or lack of knowledge, but in fact all of the mistakes were fixed on spot. The only part which gave me the problem was the scaling function, which at the very beginnig did not even want to work, since sometimes the Ydelta was infinite, what resulted in a infinite scale. However after implementing the functions which checked if some values are infinite, the problem was mostly fixed. Apart form that, there was only a problem with shrinking the rendered value, as all functions were square. The solution was actually fairly simple. It turned out that the j2 value had to be initalised as a double instead of int. Other minor errors were fixed by trial and error method.

Before:



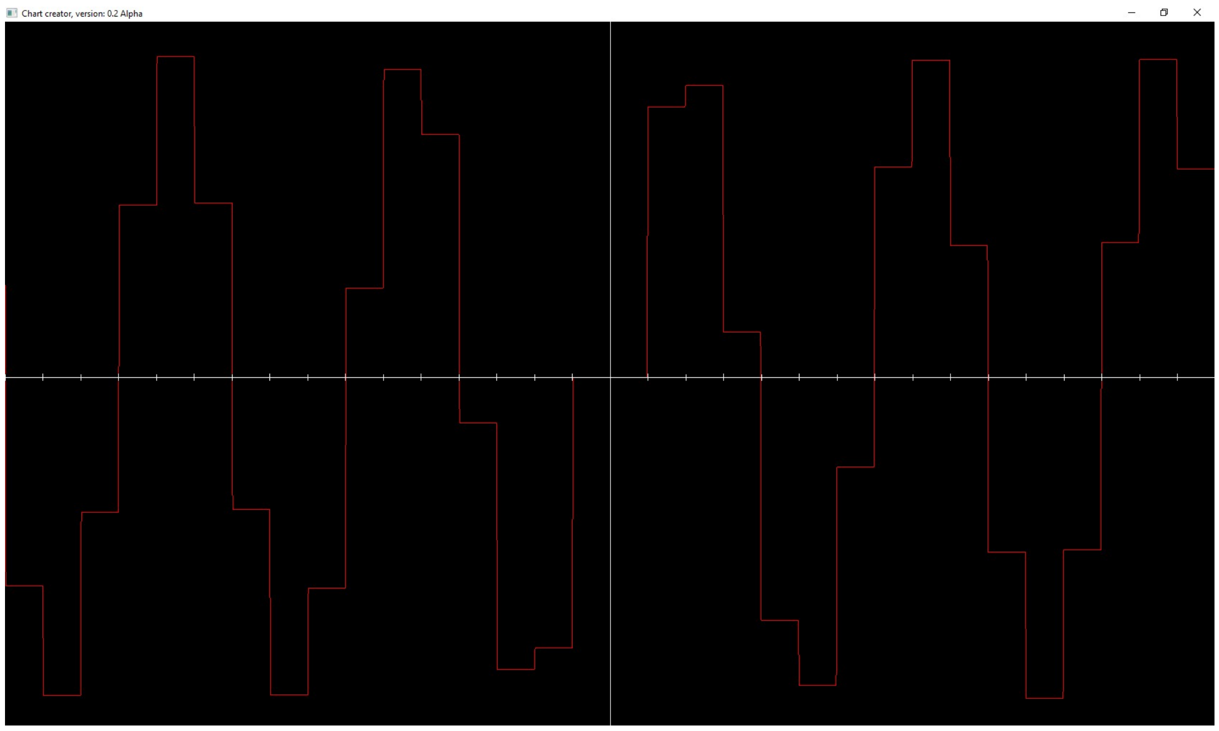
```
    // Replace X's in string with values and putting them into array //
        for (int j2 = ((-WINX / 2)); j2 < (WINX / 2); j2++)// x c <-400,400>;
        {
            int q;
            double p = j2 / shrink; // 1 on xplane becomes wide 50 [px]
```

After:

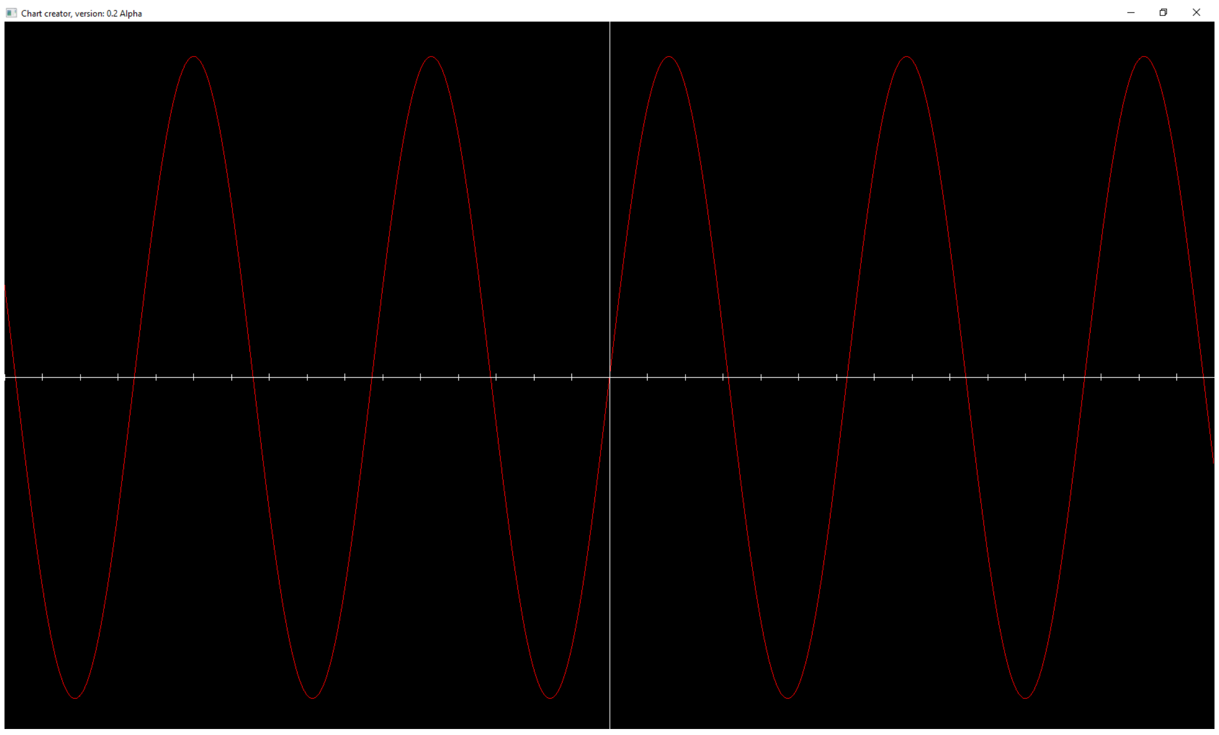


```
for (double j2 = ((-WINX / 2)); j2 < (WINX / 2); j2++)// x c <-400,400>
{
    int q;
    double p = j2 / shrink; // 1 on xplane becomes wide 50 [px]
```

Before:



After:



Simple, yet catastrophical mistake.