Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
Факультет Електроніки
Кафедра мікроелектроніки

ЗВІТ

Про виконання лабораторної роботи №4

з дисципліни: «Алгоритми та структури даних-2»

Використання списків для збереження та відображення графічної інформації.

Виконавець:

Студент 3-го курсу _____ А. С. Мнацаканов
(підпис)

Перевірив: _____ Д. Д. Татарчук
(підпис)

2021

Мета роботи – навчитись використовувати списки для тимчасового збереження великих об'ємів інформації..

Завдання

Написати програму, що виконує наступні дії:

1) Зчитує дані із файлу отриманого в першій лабораторній роботі та зберігає їх у пам'яті у вигляді структури заданої у таблиці 5 відповідно до варіанту завдань.

2) Визначає максимальний та мінімальний елементи списку.

3) Розраховує масштабні коефіцієнти для відображення графіку функції на екрані. Масштабні коефіцієнти повинні бути розраховані таким чином, щоб графік функції займав весь екран, як по вертикалі так і по горизонталі.

4) Відображати графік функції на екрані. Крім графіку на екрані повинні відображатись осі координат, масштабна сітка та значення по осях X та Y.

Виконання роботи

Код на C++

```
#include <cmath>
#include <stdlib.h>
#include <vector>
#include <iostream>
#include <thread>
#include <chrono>
#include <algorithm>
#include <fstream>
```

```
#include "sgl/sgl"

#define BLOCK_SIZE 1024

#include <GL/glut.h>
#include <stddef.h>
#include <math.h>


class list{
public:
    list(){
        data = (float*)malloc(BLOCK_SIZE*sizeof(float));
        size = 0;
        reserve = BLOCK_SIZE;
    }

    ~list(){
        free(data);
    }

    void push(float f){
        if(size == reserve){
            data = (float*)realloc(data, (reserve+BLOCK_SIZE)*
               sizeof(float));
            reserve+=BLOCK_SIZE;
            return push(f);
        }
        data[size] = f;
        size+=1;
    }

    float operator[](size_t index){
        return data[index];
```

```
    }

    size_t get_size(){
        return size;
    }

    float min(){
        float res = data[0];
        for(size_t i=1; i<size; ++i){
            if(res > data[i])
                res = data[i];
        }
        return res;
    }

    float max(){
        float res = data[0];
        for(size_t i=1; i<size; ++i){
            if(res < data[i])
                res = data[i];
        }
        return res;
    }

private:
    float* data;
    size_t size;
    size_t reserve;
};

std::ostream& operator<<(std::ostream& out, list& l){
    for(size_t i=0; i<l.get_size(); ++i){
        out << l[i] << " ";
    }
    out << std::endl;
```

```
    return out;
}

list x;
list y;

void displayX(){
    float yratio = glutGet(GLUT_WINDOW_HEIGHT)/(y.max()-y.min());
    float xratio = glutGet(GLUT_WINDOW_WIDTH)/(x.max()-x.min());
    glBegin(GL_LINES);
        glVertex2f(0, (0-y.min())*yratio);
        glVertex2f(glutGet(GLUT_WINDOW_WIDTH), (0-y.min())*yratio)
           ;
    glEnd();
    for(float i = x.min(); i<x.max();i+=(x.max()-x.min())/20){
        glBegin(GL_LINES);
            glVertex2f((float(i)-x.min())*xratio, 0);
            glVertex2f((float(i)-x.min())*xratio, glutGet(
               GLUT_WINDOW_HEIGHT));
        glEnd();
    }
}

void displayY(){
    float xratio = glutGet(GLUT_WINDOW_WIDTH)/(x.max()-x.min());
    float yratio = glutGet(GLUT_WINDOW_HEIGHT)/(y.max()-y.min());
    glLineWidth(1.0f);
    glBegin(GL_LINES);
        glVertex2f((0-x.min())*xratio, 0);
        glVertex2f((0-x.min())*xratio, glutGet(GLUT_WINDOW_HEIGHT)
           );
    glEnd();
    for(float i = y.min(); i<y.max();i+=(y.max()-y.min())/20){
        glBegin(GL_LINES);
            glVertex2f(0, (float(i)-y.min())*yratio);
```

```
            glVertex2f(glutGet(GLUT_WINDOW_WIDTH), (float(i)-y.min
               ())*yratio);
        glEnd();
    }
}

void display_function(){
    float xratio = glutGet(GLUT_WINDOW_WIDTH)/(x.max()-x.min());
    float yratio = glutGet(GLUT_WINDOW_HEIGHT)/(y.max()-y.min());
    glBegin(GL_LINE_STRIP);
        for(size_t i=0; i<x.get_size();++i){
            glVertex2f((x[i]-x.min())*xratio, (y[i]-y.min())*
               yratio);
        }
    glEnd();
}

void display(){
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glColor3f(0.0f, 0.0f, 0.0f);
    glLineWidth(0.5f);
    displayX();
    displayY();
    glLineWidth(2.0f);
    display_function();
    glutSwapBuffers();
}

void init()
{
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    gluOrtho2D(0, glutGet(GLUT_WINDOW_WIDTH), 0, glutGet(
```

```
        GLUT_WINDOW_HEIGHT));

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glClearColor(1.0f, 1.0f, 1.0f, 0.0f);
}


void idle() {
    glutPostRedisplay();
}

int main(int argc, char *argv[]){
    std::ifstream in("data.txt");
    while(!in.eof()){
        float temp_x, temp_y;
        in >> temp_x >> temp_y;
        x.push(temp_x);
        y.push(temp_y);
    }

    std::cout << x << y << std::endl;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowSize(800, 600);
    glutCreateWindow("2D Drawing Tool");

    glutDisplayFunc(display);
    glutIdleFunc(idle);

    init();
    glutMainLoop();
    return 0;
}
```