МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Программирование»

Тема: Программирование с использованием связных списков

Студент гр. 9301 _____ Судаков Е.В

Преподаватель _____ Рыжов Н.Г

Санкт-Петербург
2020

Задание

Написать программу, формирующую динамический список, содержащий координаты точек на экране, считая, что длина списка (количество точек) задана. Программа должна позволять удалять любую точку с заданным пользователем номером. Сформировать на экране изображение (до и после удаления точки), состоящее из окружностей произвольного радиуса с центрами, совпадающими с координатами точек и соединяющими эти окружности линиями.

1. Блок-схема программы(рис.1)

Рисунок 1. Блок схема программы

2. Текст программы

Файл main.cpp

```
#include <SFML/Graphics.hpp>
#include <iostream>
#include <conio.h>
#include <windows.h>
#include "linkedList.h"

#define CP 1251
#define textSize 10

void setShapes(Node *head, sf::CircleShape *shapes, int n) {
	for (int i = 0; i < n; i++) {
		shapes[i].setRadius(head->p.r);
		shapes[i].setPosition(head->p.x, head->p.y);
		shapes[i].setFillColor(sf::Color::White);
		head = head->next;
	}
}


void setText(sf::CircleShape *shapes, sf::RenderWindow &window, int n) {
//функция добавления номера круга в центр каждого круга для наглядности
	sf::Font font;
	if (!font.loadFromFile("Roboto-Regular.ttf")){ /*error...*/ }
	sf::Text text;
	text.setCharacterSize(textSize);
	text.setFont(font);
	text.setFillColor(sf::Color::Black);
	text.setStyle(sf::Text::Bold | sf::Text::Underlined);

	int r;
	char c;
	for (int i = 0; i < n; i++) {
		r = shapes[i].getRadius();
		c = i + '0';
		text.setString(c);
		sf::FloatRect textRect = text.getLocalBounds();
		text.setOrigin(textRect.left + textRect.width / 2.0f,
			textRect.top + textRect.height / 2.0f);
		text.setPosition(shapes[i].getPosition().x + r, shapes[i].getPosition().y + r);
		window.draw(text);
	}
}


void drawCircleShapes(sf::CircleShape *shapes, sf::RenderWindow &window, int n) {
	for (int i = 0; i < n; i++) {
		window.draw(shapes[i]);
	}
}


void draw(int x0, int y0, int x1, int y1, sf::VertexArray line, sf::RenderWindow &window) {
	sf::Vector2u size = window.getSize();//получение вектора текущего размера окна
	line[0].position = sf::Vector2f(x0, y0);//позиция начала линии - в центре окна
	line[1].position = sf::Vector2f(x1, y1); //конец линии - в текущих координатах xi и yi
	window.draw(line);//вывод линии в буфер
}


void connectCircles(sf::CircleShape *shapes, sf::RenderWindow &window, int n) {
```

```
//функция соединения кругов линиями
	sf::VertexArray line(sf::Lines, 2);
	int x0, y0, x1, y1, r1, r2;
	for (int i = 0; i < n - 1; i++) {
		r1 = shapes[i].getRadius();
		r2 = shapes[i+1].getRadius();
		x0 = shapes[i].getPosition().x + r1;
		y0 = shapes[i].getPosition().y + r1;
		x1 = shapes[i + 1].getPosition().x + r2;
		y1 = shapes[i + 1].getPosition().y + r2;
		std::cout << x0 << "\n";
		draw(x0, y0, x1, y1, line, window);
	}
}


int main()
{
	SetConsoleCP(CP);
	SetConsoleOutputCP(CP);

	Point p = {width / 2, height / 2, 10}; // just hardcoded first position
	Node *head = new Node;
	head->p = p;
	head->next = NULL;
	createRandomList(head, N); //создание случайного списка
	printList(head);


	sf::RenderWindow window(sf::VideoMode(width, height), "linked list");

	sf::Event event;

	sf::CircleShape shapes[N];
	sf::VertexArray lines[N-1];

	setShapes(head, shapes, N);

	drawCircleShapes(shapes, window, N);
	connectCircles(shapes, window, N);
	setText(shapes, window, N);
	window.display();

	//запроса номера элемента для удаления
	int del;
	std::cout << "\nВведите номер круга для удаления ::\n";
	std::cin >> del;
	if (del < N && del >= 0) {
		std::cout << del;
	}
	else {
		std::cout << "Нет такого номера!\n";
	}
	deleteNth(&head, del);
	printList(head);

	sf::RenderWindow newWindow(sf::VideoMode(width, height), "linked list after deletion");
	sf::CircleShape shapes1[N-1];
	sf::VertexArray lines1[N - 2];

	setShapes(head, shapes1, N-1);

	drawCircleShapes(shapes1, newWindow, N-1);
	connectCircles(shapes1, newWindow, N-1);
```

```
	setText(shapes1, newWindow, N - 1);
	newWindow.display();

	while (window.isOpen())
	{
		while (window.pollEvent(event))
		{
			if (event.type == sf::Event::Closed)
				window.close();
			if (event.type == sf::Event::KeyPressed) {
				if (event.key.code == sf::Keyboard::Escape) {
					window.close();
				}
			}
		}
	}
	return 0;
}
```

Файл linkedList.h

```
#ifndef LINKEDLIST_H
#define LINKEDLIST_H


constexpr auto width = 640;
constexpr auto height = 480;
constexpr auto N = 10;
constexpr auto MaxRad = 30;


#include <iostream>
#include <random>
#include <ctime>

struct Point {
	int x;
	int y;
	int r;
};

struct Node {
	Point p;
	Node *next;
};

void push(Node **head, Point p); // add in beginning of the list

Point pop(Node **head); // delete from head of the list

Node* getLast(Node *head);

void pushBack(Node *head, Point p);

Node *getOneBeforeLast(Node *head);

void popBack(Node **head);

Node *getNth(Node *head, int n);

void printList(Node *head);

void checkHead(Node *head);
```

```
void insert(Node *head, Point p, int n);

void deleteNth(Node **head, int n);

void deleteList(Node **head);

void createRandomList(Node *head, int n);

//utils
Point createRandomPoint();

#endif
```

Файл linkedList.cpp

```
#include "linkedList.h"


void checkHead(Node *head) {
	if (head == NULL) {
		std::cout << "Head is NULL from function ::" << __func__ << "\n";
		//exit(-1);
	}
}


void push(Node **head, Point p) {
	//as in stack
	Node *tmp = new Node;
	tmp->p = p;
	tmp->next = *head;
	(*head) = tmp;
}


Point pop(Node **head) {
	//as in stack
	Node *tmp = NULL;
	checkHead(*head);
	tmp = *head;
	Point val = (*head)->p;
	*head = (*head)->next;
	free(tmp);
	return val;
}


Node *getLast(Node *head) {
	if (head == NULL) {
		return NULL;
	}
	while (head->next) {
		head = head->next;
	}
	return head;
}



Node *getOneBeforeLast(Node *head) {
	checkHead(head);
	if (head->next == NULL) {
		return NULL;
	}
```

```
	while (head->next->next != NULL) {
		head = head->next;
	}
	return head;
}


void popBack(Node **head) {
	Node *tmp = getOneBeforeLast(*head);
	if (tmp == NULL) {
		free(*head);
		*head = NULL;
		return;
	}
	Node *last = tmp->next;
	tmp->next = NULL;
	free(last);
}


void pushBack(Node *head, Point p) {
	Node *tmp = new Node;
	Node *last = getLast(head);
	tmp->p = p;
	tmp->next = NULL;
	last->next = tmp;
}


Node *getNth(Node *head, int n) {
	checkHead(head);
	int cnt = 0;
	while (head && cnt < n) {
		head = head->next;
		cnt++;
	}
	return head;
}


void insert(Node *head, Point p, int n) {
	//insert node after n`th elem
	int cnt = 1;
	while (head->next && cnt < n) {
		head = head->next;
		cnt++;
	}
	Node *tmp = new Node;
	tmp->p = p;
	if (head->next) {
		tmp->next = head->next;
	}
	else {
		tmp->next = NULL; // inserting after last
	}
	head->next = tmp;
}


void deleteNth(Node **head, int n) {
	if (n == 0) {
		pop(head);
	}
	else {
```

```
			Node *prev = getNth(*head, n - 1);
			Node *tmp = prev->next;
			if (tmp) {
				prev->next = tmp->next;
				free(tmp);
			}
			else {
				std::cout << __func__ << " :: List out of range\n";
			}
		}
}



void deleteList(Node** head) {
		Node* current = *head;
		Node* next;
		while (current != NULL) {
			next = current->next;
			free(current);
			current = next;
		}
		*head = NULL;
}


void printList(Node *head) {
		while (head) {
			std::cout << " x :: "<< head->p.x << " y :: " <<head->p.y << "  R :: "<< head->p.r
<< "\n";
			head = head->next;
		}
		std::cout << "NULL" << "\n";
}


std::mt19937 gen(time(0));//Mersenne twister
std::uniform_int_distribution<> distributionX(MaxRad, width-MaxRad*2);
std::uniform_int_distribution<> distributionY(MaxRad, height-MaxRad*2);
std::uniform_int_distribution<> distributionR(15, MaxRad);


Point createRandomPoint() {
		Point p;
		p.x = distributionX(gen);
		p.y = distributionY(gen);
		p.r = distributionR(gen);
		return p;
}


void createRandomList(Node *head, int n) {
		int cnt = 1;
		Point p;
		if (!(head->p.x || head->p.y || head->p.r)) {
			head->p = createRandomPoint();
		}
		while (cnt < n) {
			p = createRandomPoint();
			pushBack(head, p);
			cnt++;
		}

}
```

3. Описание програмы

Программа генериерует связный список, с полями координаты, которые заполняются случайно. Затем программа просит ввести пользователя номер элемента для удаления. Удаляет элемент, затем визуализрует получившийся список в новом окне.

4. Руководство пользователя

(1) Назначение программы: Программа обладает функционалом создания и визуализации связного списка

(2) Условия применения: Распространяется под лицензией MIT. Ограничения :x86-64 совместимый процессор вычислительной машины.

(3) Пуск программы : Для запуска программы дважды нажмите на исполняемый файл (linkedList.exe).

(4) Сообщения оператору: вывод рабочей сессии (рис.5)

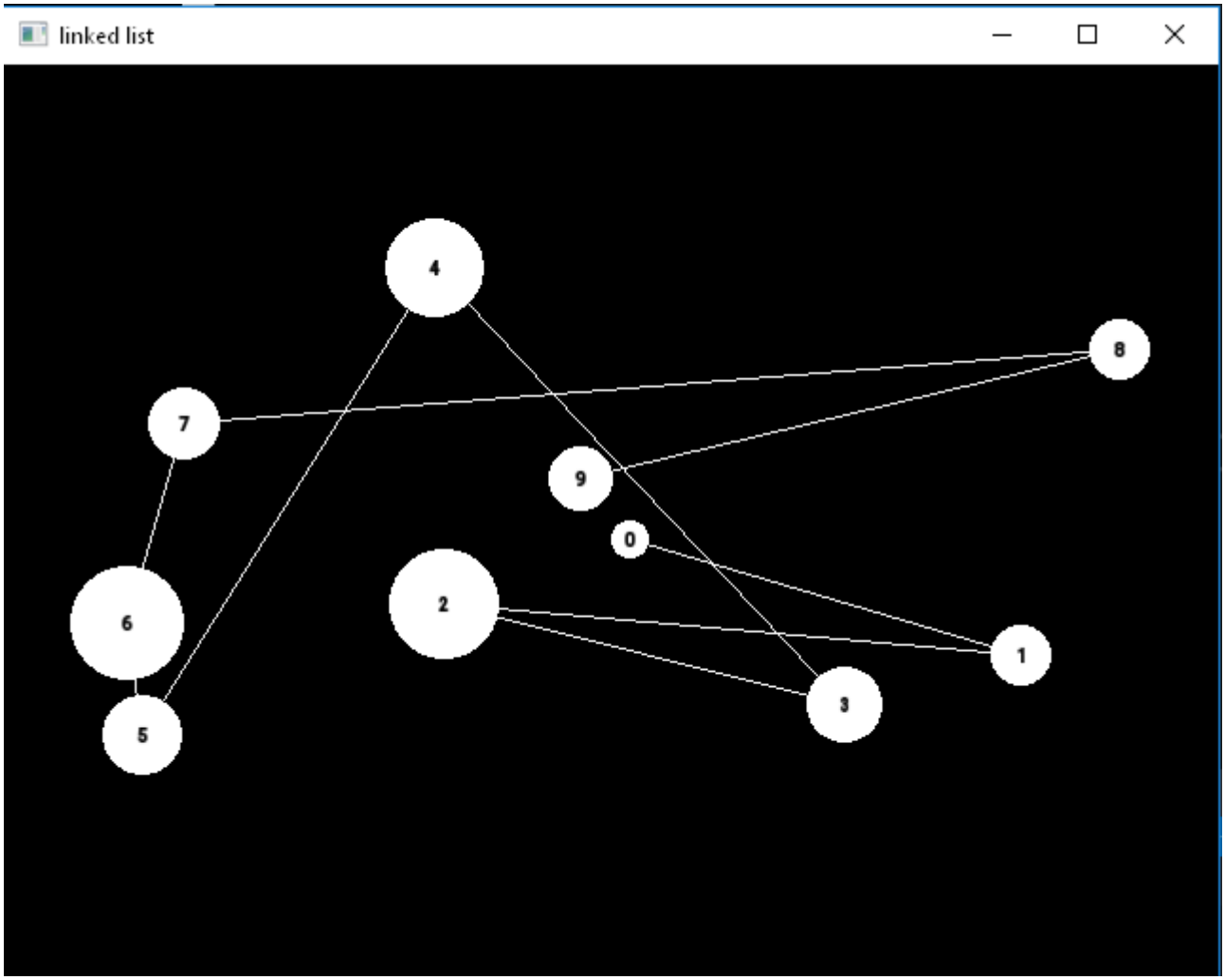

Рис 2. Список до удаления

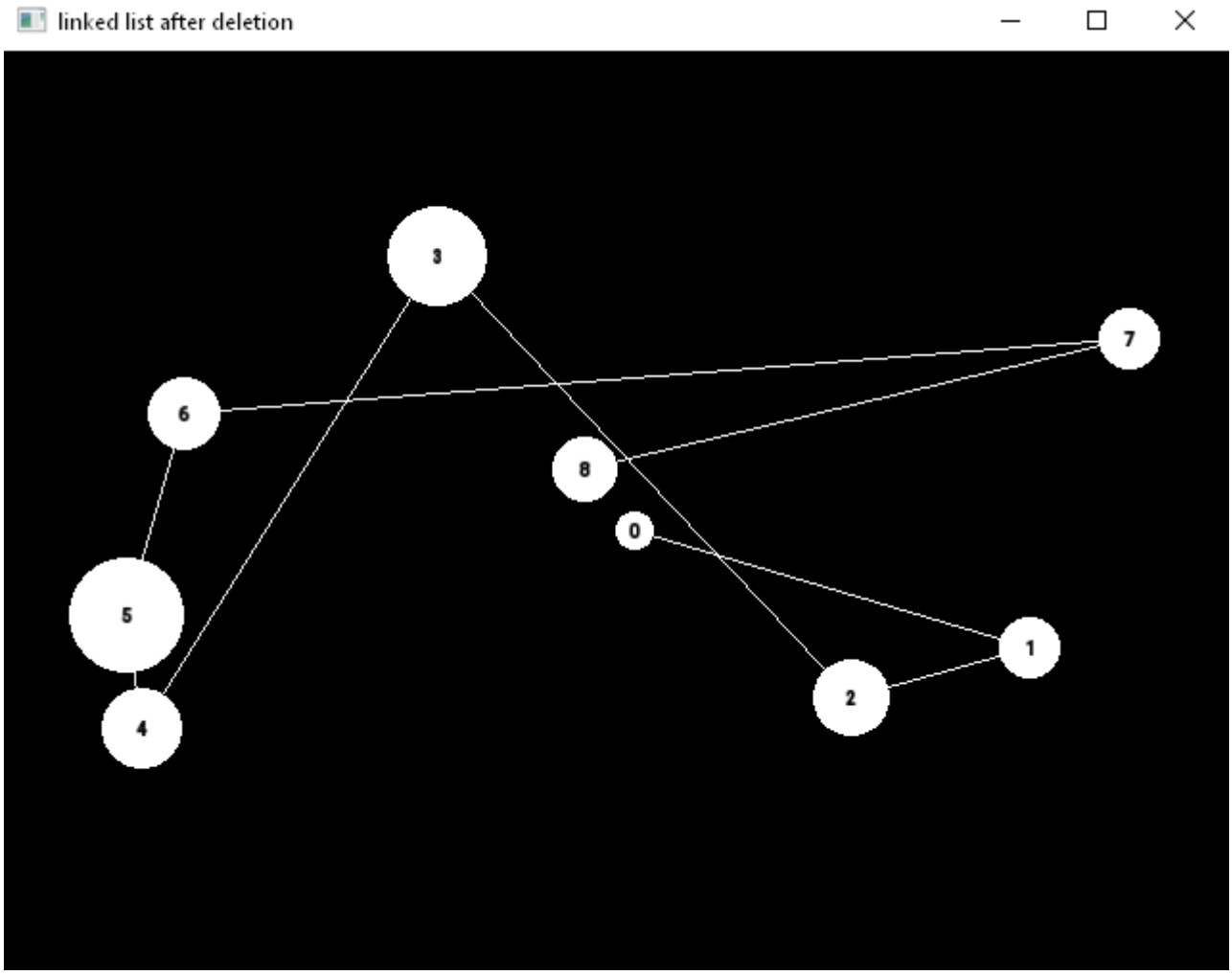

Рис. 3 Список после удаления

5. Пути дальнейшего улучшения программы

- Выбор цвета кругов и линий

- Выбор радиуса кругов

- Изменение цвета от объекта к объекту