Белорусский государственный университет

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра информационных систем управления

Волков Евгений Сергеевич

**Отчет**

**по учебной вычислительной практике**

студента 1 курса 10 группы

**Преподаватель**

*Конах Валентина Владимировна*

старший преподаватель кафедры ИСУ

Минск-2020

# **Оглавление**

[Введение 4](#_Toc40294487)

[1 HTML, CSS 5](#_Toc40294488)

[1.1 Лабораторная работа 1 5](#_Toc40294489)

[1.2 Лабораторная работа 2 7](#_Toc40294490)

[2 С++ 12](#_Toc40294491)

[2.1 Лабораторная работа 1 12](#_Toc40294492)

[Задание 1 12](#_Toc40294494)

[Задание 2 12](#_Toc40294495)

[Задание 3 13](#_Toc40294496)

[2.2 Лабораторная работа 2 17](#_Toc40294497)

[2.3 Лабораторная работа 3 22](#_Toc40294501)

[2.4 Лабораторная работа 4 22](#_Toc40294503)

[2.5 Лабораторная работа 5 23](#_Toc40294504)

[2.6 Лабораторная работа 6 26](#_Toc40294506)

[2.7 Лабораторная работа 7 29](#_Toc40294508)

[3 WinAPI 34](#_Toc40294510)

[3.1 Лабораторная работа 1 34](#_Toc40294511)

[3.1.1 Задание 1 34](#_Toc40294512)

[3.1.2 Задание 2 34](#_Toc40294513)

[3.1.3 Задание 3 35](#_Toc40294514)

[3.2 Лабораторная работа 2 37](#_Toc40294515)

[3.2.1 Задание 1 37](#_Toc40294516)

[3.2.2 Задание 2 39](#_Toc40294517)

[3.2.3 Задание 3 41](#_Toc40294518)

[3.2.4 Задание 4 43](#_Toc40294519)

[3.3 Лабораторная работа 3 46](#_Toc40294520)

[3.3.1 Задание 1 46](#_Toc40294521)

[3.3.2 Задание 2 47](#_Toc40294522)

[3.3.3 Задание 3 49](#_Toc40294523)

[3.4 Лабораторная работа 4 52](#_Toc40294525)

[3.4.1 Задание 1 52](#_Toc40294526)

[3.4.2 Задание 2 52](#_Toc40294527)

[3.4.3 Задание 3 54](#_Toc40294528)

[3.4.4 Задание 4 58](#_Toc40294529)

[3.4.5 Задание 5 58](#_Toc40294530)

[3.5 Лабораторная работа 5 58](#_Toc40294531)

[4 С# 59](#_Toc40294532)

[4.0 Лабораторная работа 0 59](#_Toc40294533)

[4.1 Лабораторная работа 1 64](#_Toc40294534)

[4.2 Лабораторная работа 2 64](#_Toc40294535)

[4.3 Лабораторная работа 3 64](#_Toc40294536)

[4.4 Лабораторная работа 4 64](#_Toc40294537)

[4.5 Лабораторная работа 5 65](#_Toc40294538)

[Заключение 66](#_Toc40294539)

[Список литературы 67](#_Toc40294540)

[Приложение А 68](#_Toc40294541)

# Введение

Я, Волков Евгений Сергеевич, студент 1 курса 10 группы факультета прикладной математики и информатики, специальности «компьютерная безопасность» проходил свою учебную практику в течение первого и второго семестров обучения.

**Цель практики:**

* закрепить теоретические знания и получить навыки их практического применения;
* ознакомление со спецификой самостоятельного изучения материала;
* практическое применение полученных знаний;
* научиться самостоятельно решать поставленные задачи, проверить на практике свою теоретическую базу;
* грамотно распределять свободное время и использовать его с пользой для себя;
* обучение самостоятельному углублению знаний.

**Постановка задач:**

* углубить теоретические и практические знания;
* выполнить ряд лабораторных работ, связанных с разработкой сайтов, Windows-приложений, консольных приложений;
* закрепить на практике изучение WinAPI, С++, C#, а также языков разметки HTML и CSS.

# 1 HTML, CSS

## 1.1 Лабораторная работа 1

**Постановка задачи.** Создать сайт "Автобиография", используя HTML.

1. Основы HTML. Структура HTML-документа. Теги, комментарии. Форматирование текста.

2. Форматы графических файлов - JPEG, GIF, PNG. Тэг <img> и его атрибуты. Размещение, выравнивание и обрамление изображений. Размеры изображений, альтернативный текст. Карты изображений. Создание графических меню.

3. Работа со списками. Неупорядоченный список, упорядоченный список, список определений. Вложенные списки. Теги <dd>, <dt>.

4. Создание гиперссылок. Изображение в качестве ссылки. Тег гипертекстовой ссылки, его атрибуты. Открытие ссылки в новом окне, ссылка на место в текущем документе, mailto: ссылки.

5. Работа с таблицами. Теги <table>, <td>, <th> и атрибуты данных тегов. Различные виды таблиц: таблица без рамки, заголовки таблиц, пустые ячейки, выравнивание и отступы в ячейках. Изображения как фон ячейки.

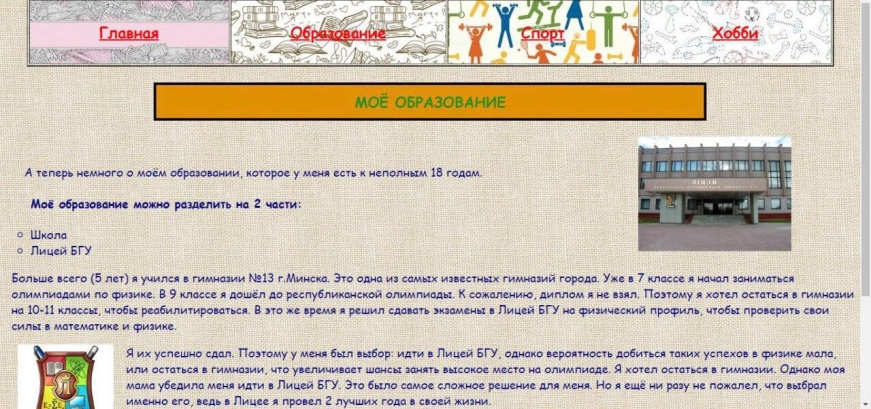
Разработать 4 HTML страницы, содержащие информацию о Вас: index.html, место рождения, место учебы, увлечения.

При разработке использовать все 5 пунктов. Включить в их состав изображения. Обеспечить переход между страницами с помощью ссылок.

**Решение**. Данная задача была реализована при помощи всех вышеперечисленных пунктов. Изображения были включены в их состав. Переход между страницами был обеспечен с помощью ссылок.



Скриншот 1.1.1. Главной страницы



Скриншот 1.1.2. Страницы «Моё образование»



Скриншот 1.1.3. Страницы «Спорт»



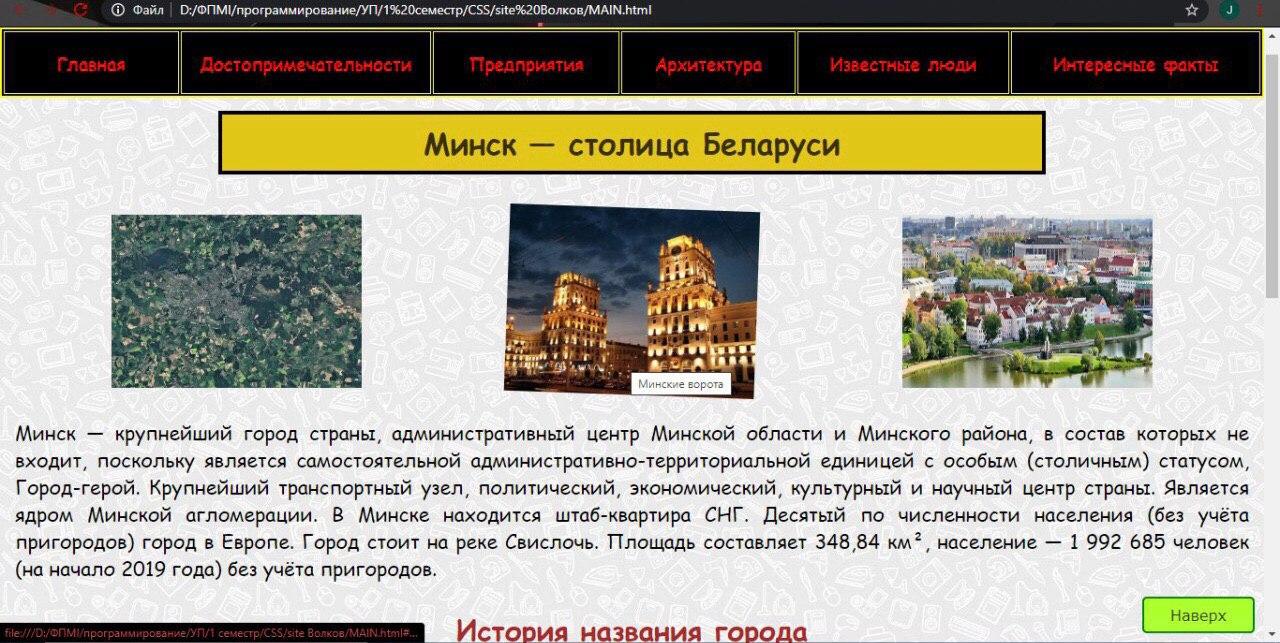
Скриншот 1.1.5. Страницы «Мои хобби»

## 1.2 Лабораторная работа 2

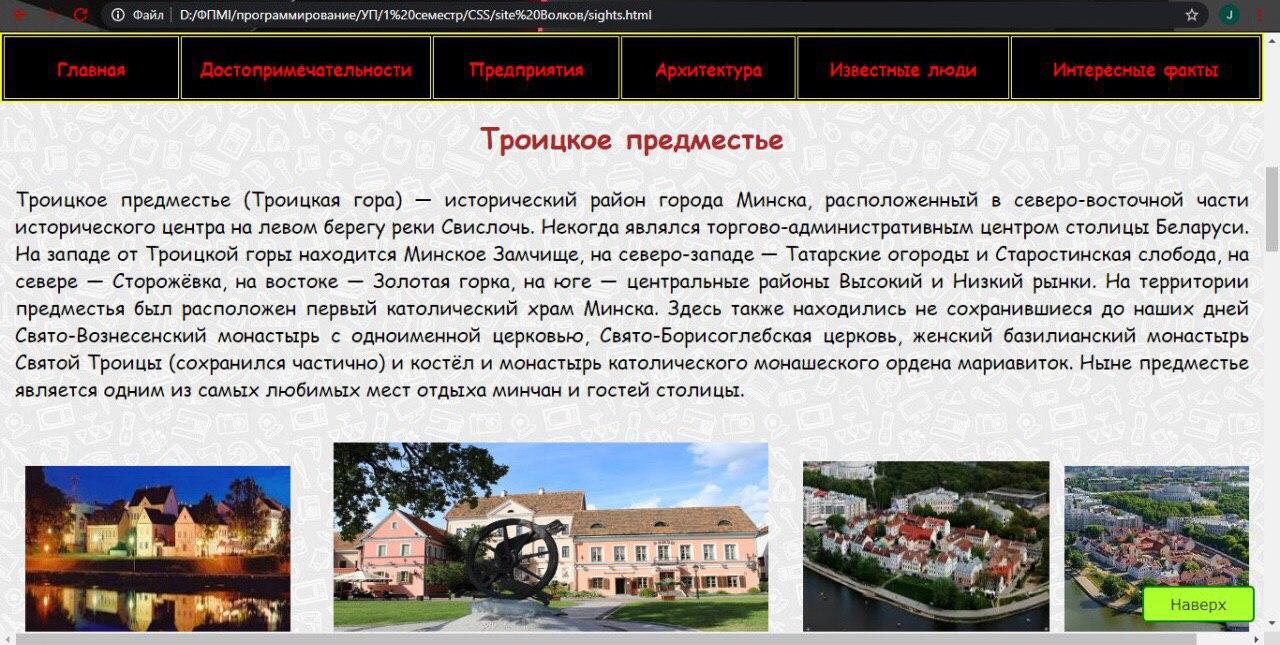
**Постановка задачи.** Моя малая Родина.

Создать не менее 6 статических HTML-страниц, в которых будет рассказано о вашей малой Родине. Нужно рассказать о вашем городе, деревне. Рассказать о природе, достопримечательностях, об основных предприятиях, выпускаемой ими продукции, о знаменитых людях и т.д. Общая структура страниц должна состоять из двух частей. В одной части будет меню. В другой - должна выводиться соответствующая информация. Титульная страница должна содержать в верхней части графическое меню, в нижней должна выводится информация. Обязательно использовать возможности CSS

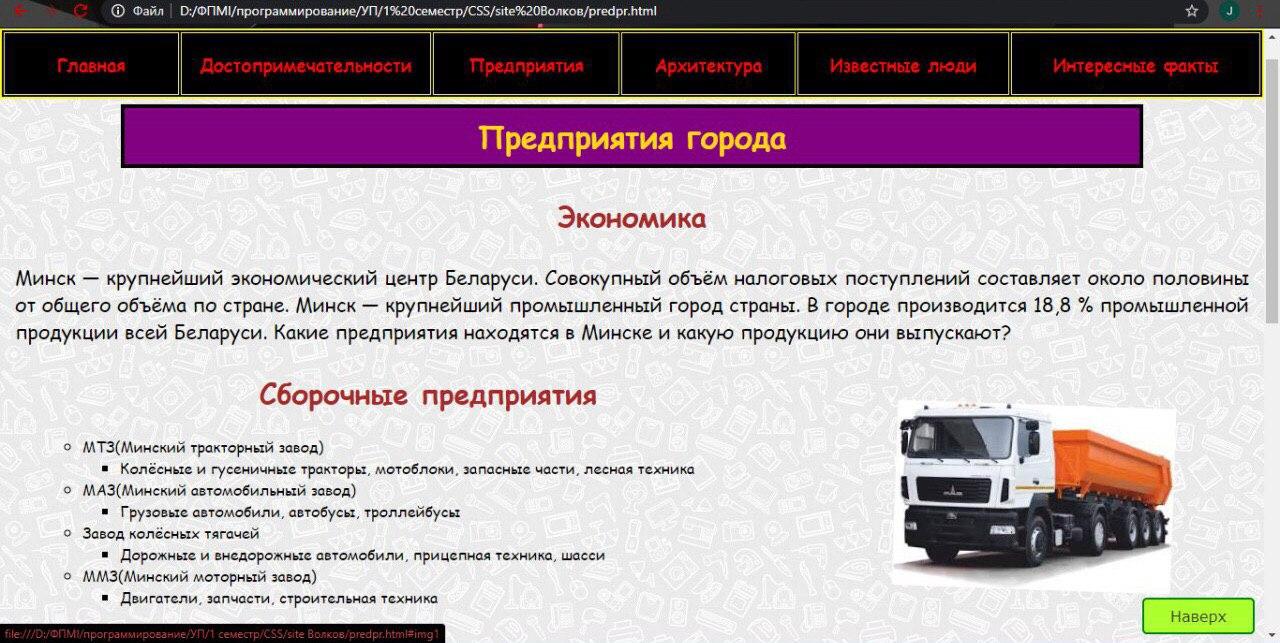
**Решение**. Данная задача была реализована при помощи средств HTML и CSS. Код работы прилагается.



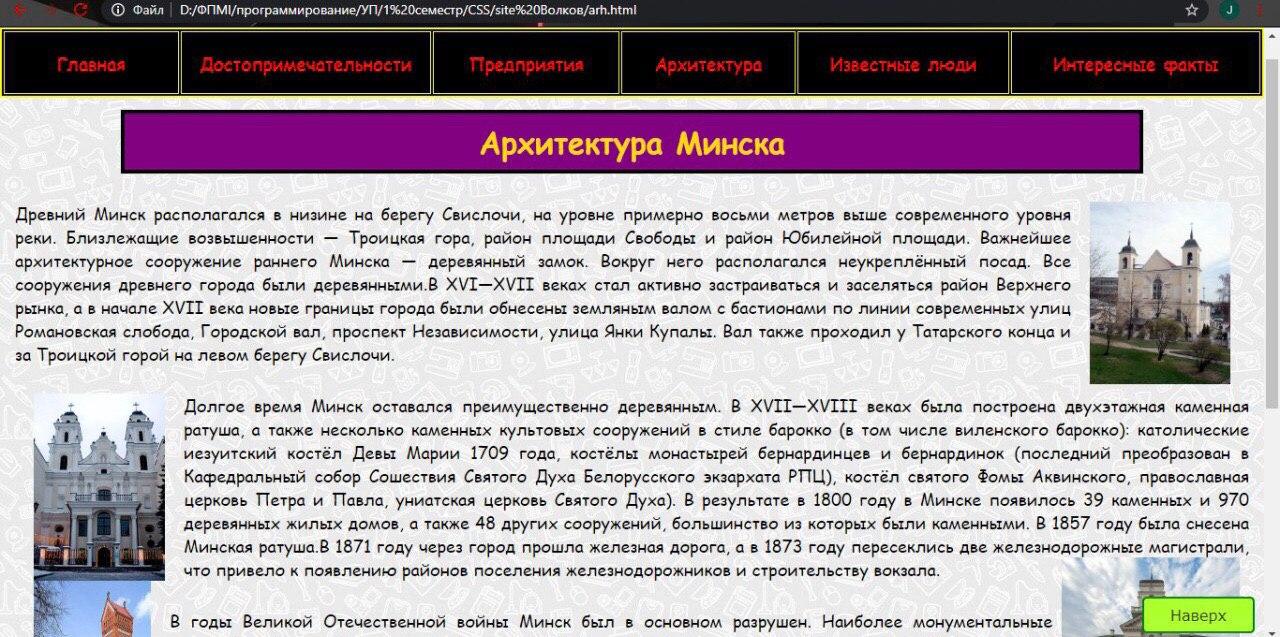
Скриншот 1.2.1. Главная страница



Скриншот 1.2.2 Страница «Достопримечательности»



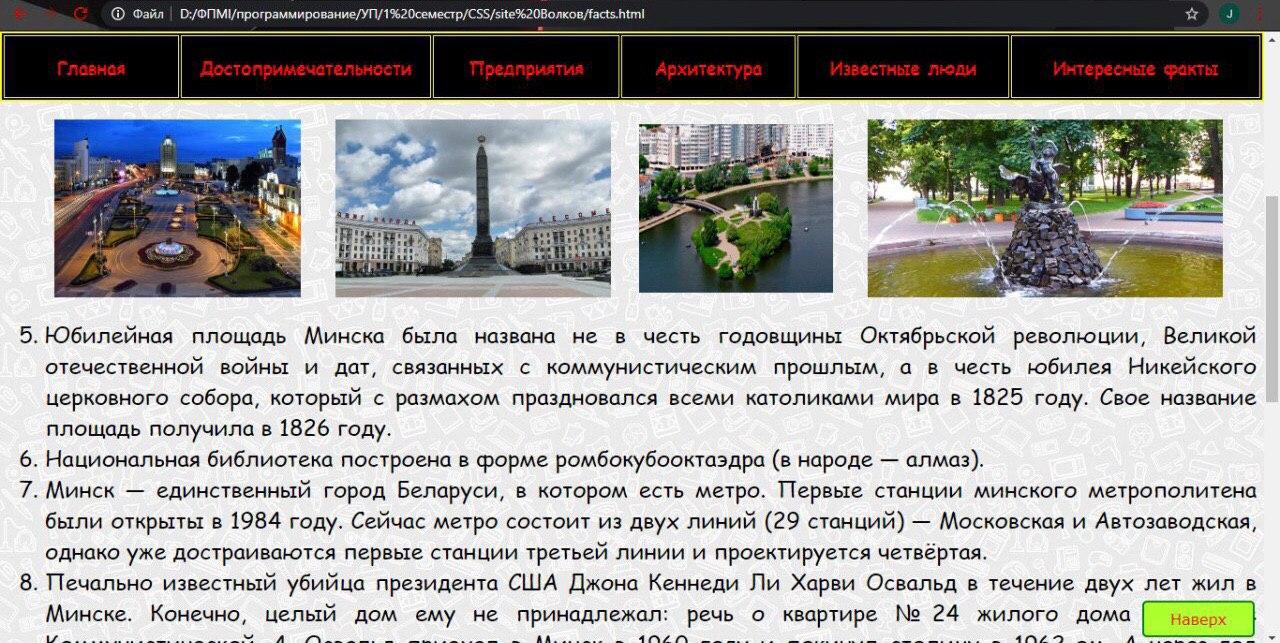
Скриншот 1.2.3. Страница «Предприятия»



Скриншот 1.2.4. Страница «Архитектура»



Скриншот 1.2.5. Страница «Известные люди»



Скриншот 1.2.6. Страница «Интересные факты»

**Код программы.**

Общий HTML код для меню на всех страницах:

<div id="table">

<table border="2" bgcolor="black" bordercolor="yellow" width="100%" >

<tr>

<td width="14%" ><a href="MAIN.html" target="\_blank"><h3>Главная</h3></a></td>

<td width="20%" ><a href="sights.html"><h3>Достопримечательности</h3></a></td>

<td width="15%" ><a href="predpr.html"><h3>Предприятия</h3></a></td>

<td width="14%" ><a href="arh.html"><h3>Архитектура</h3></a></td>

<td width="17%" ><a href="people.html"><h3>Известные люди</h3></a></td>

<td width="20%" ><a href="facts.html"><h3>Интересные факты</h3></a></td>

</tr>

</table>

</div>

CSS код 1:

Body

{

background-image: url(fon.jpg);

}

a

{

text-decoration: none;

color:orange;

}

a:visited

{

color:red;

}

a:active

{

color:white;

}

h1

{

text-align: center;

}

img:hover

{

transform: rotate(2deg) scale(1.08);

}

#table

{

left: 0px;

right: 0px;

position: fixed;

top: 0px;

z-index: 1000;

}

hr

{

border-color: purple;

}

p

{

margin-top: 30px;

margin-left: 8px;

margin-right: 8px;

font-size: 21px;

text-align: justify;

}

h2{

font-size:30px;

text-align: center;

color: brown;

margin-top:30px;

}

.topbutton {

width:100px;

border:2px solid green;

background: GreenYellow;

text-align:center;

padding:8px;

position:fixed;

bottom:10px;

right:10px;

cursor:pointer;

color:#333;

font-family:verdana;

font-size:16px;

border-radius: 5px;

-moz-border-radius: 5px;

-webkit-border-radius: 5px;

-khtml-border-radius: 5px;

}

li{

text-align: justify;

}

CSS код для движущейся рамки:

<style>

.blok {

cursor: pointer;

}

.blok:hover {

background: yellow;

color: black;

}

.transition-width {

background: purple;

color: gold;

margin: 0 auto;

padding: 8px;

text-align: center;

max-width: 80%;

font-size: 32px;

border: 4px ridge black;

cursor: pointer;

transition: 3s linear;

}

.transition-width:hover {

max-width: 60%;

}

#text

{

margin-top: 120px;

}

#tabl

{

margin-top: 40px;

}

td{

text-align: center;

}

.thumbnail {

max-width: 100%;

}

.lightbox {

display: none;

position: fixed;

z-index: 999;

width: 100%;

height: 100%;

text-align: center;

top: 0;

left: 0;

background: gray;

}

.lightbox img {

max-width: 50%;

max-height: 50%;

margin-top: 20%;

}

.lightbox:target {

outline: none;

display: block;

}

</style>

# 2 С++

## 2.1 Лабораторная работа 1

## Сортировки

## Задание 1

**Постановка задачи.**

Разработка алгоритмов внутренней сортировки (сортировки массивов)

Дан массив целых чисел *a*[*n*], *n*>0. Разработать приложение, которое выполняет сортировку массива по возрастанию с помощью различных алгоритмов сортировки.

1. Выполнить сортировку элементов массива с помощью **«гномьей сортировки».**

Гномья сортировка (англ. **Gnome sort**) — алгоритм сортировки, похожий на сортировку вставками, но в отличие от последней перед вставкой на нужное место происходит серия обменов, как в сортировке пузырьком. Название происходит от предполагаемого поведения садовых гномов при сортировке линии садовых горшков.

Гномья сортировка основана на технике, используемой обычным голландским садовым гномом (нидерл. tuinkabouter). Это метод, которым садовый гном сортирует линию цветочных горшков. По существу, он смотрит на текущий и предыдущий садовые горшки: если они в правильном порядке, он шагает на один горшок вперёд, иначе он меняет их местами и шагает на один горшок назад. Граничные условия: если нет предыдущего горшка, он шагает вперёд; если нет следующего горшка, он закончил.

Сложность алгоритма: O(n2).

1. Сортировка слиянием (англ. **Merge sort**). Требуется O(n) дополнительной памяти; выстраиваем первую и вторую половину списка отдельно, а затем — сливаем упорядоченные списки (простое двухпутевое слияние).

Сложность алгоритма: O(n\*log n).

## Задание 2

**Постановка задачи.**

***Разработка функционала для сравнительного анализа алгоритмов сортировки вашего задания.***

Дополнить приложение, разработанное ранее (Задание1, Часть1) функционалом для сравнения производительности работы алгоритмов сортировки:

− оценить число операция сравнения и число операций обмена (перемещений) элементов (например, используйте глобальные переменные – счетчики, значение которых устанавливается в соответствующих функциях).

− время работы алгоритмов сортировки. Для оценки времени работы алгоритма можно использовать библиотеки <ctime> и <chrono>.

## **Задание** 3

**Постановка задачи.**

**Проведение эксперимента.**

1. Разработайте функции (и разместите их в отдельном файле data\_generation.cpp, подготовьте заголовочный файл) для заполнения массива тестовыми данными.
2. Для получения входных данных подготовить три варианта:

* непосредственный ввод с клавиатуры и запись в текстовый файл;
* генерирование с помощью датчика случайных чисел и запись в текстовый файл;
* создание текстового файла в текстовом редакторе (блокноте, например).

1. Проведите экспериментальное сравнение производительности работы алгоритмов сортировки (Зам. *сравнение алгоритмов должно проводится на одном и том же входе (элементы входного массива должны быть одинаковыми))* для n = 10, 100, 1 000, 10 000 входных элементов:

− элементы упорядочены по возрастанию.

− элементы упорядочены по убыванию.

− случайный набор элементов.

Для проведения эксперимента загружать данные в массив из файла исходных данных.

Результат каждой сортировки записать в выходной файл(ы).

1. Результаты экспериментов оформить на основе нескольких запусков программы в виде сводных таблиц по образцу (см. Таблица 1).
2. Оформить отчет по этой лабораторной работе в docx - формате c описанием:
   * разработанных алгоритмов сортировки,
   * фрагментами кода алгоритмов,
   * сводной таблицей (в Excel) результатов эксперимента сравнительного анализа алгоритмов.

**Результаты проделанной работы за все 3 задания:**

**Алгоритмы:**

* **Гномья сортировка**

По сути, в алгоритме сравниваются рядом стоящие элементы, если они стоят в нужном порядке, тогда мы переходим на следующий элемент массива, если нет, ты мы их переставляем и переходим на предыдущий. Нет предыдущего элемента — идём вперед, нет следующего — значит мы закончили. Изначально мы находимся на втором элементе массива.

* **Сортировка слиянием**

Процедура слияния предполагает объединение двух предварительно упорядоченных подпоследовательностей размерности n/2 в единую последовательность размерности n. Начальные элементы предварительно упорядоченных последовательностей сравниваются между собой, и из них выбирается наименьший. Соответствующий указатель перемещается на следующий элемент. Процедура повторяется до тех пор, пока не достигнут конец одной из подпоследовательностей. Оставшиеся элементы другой подпоследовательности при этом передаются в результирующую последовательность в неизменном виде.

**gnome.cpp** - гномья сортировка

void GnomeSort(int \*A, int n, long long \*k, long long \*z)

{

for (int i = 0; i + 1 < n; i++)

{

if (A[i] > A[i + 1])

{

int tmp = A[i];

A[i] = A[i + 1];

A[i + 1] = tmp;

if (i != 0)

i -= 2;

(\*z)++;

}

(\*k)++;

}

}

**merge.cpp** - сортировка слиянием

long long p = 0;

long long s = 0;

void Merge(int \*arr, int begin, int end)

{

const int nmax = 10500;

int i = begin, mid = begin + (end - begin) / 2, j = mid + 1, k = 0, d[nmax];

while (i <= mid && j <= end)

{

if (arr[i] <= arr[j])

{

d[k] = arr[i];

i++; s++;

}

else

{

d[k] = arr[j];

j++; s++; p++;

}

k++;

}

while (i <= mid)

{

d[k] = arr[i];

i++; k++;

}

while (j <= end)

{

d[k] = arr[j];

j++; k++;

}

for (i = 0; i < k; i++)

arr[begin + i] = d[i];

}

void MergeSort(int \*arr, int left, int right)

{

int temp = 0;

if(left<right)

if (right - left == 1)

{

if (arr[left] > arr[right])

{

temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

p++;

}

s++;

}

else

{

MergeSort(arr, left, left + (right - left) / 2);

MergeSort(arr, left + (right - left) / 2 + 1, right);

Merge(arr, left, right);

}

}

**data\_generation.cpp** - способы заполнения массива (рандом, консоль, файл)

void random(int\* a,int \*A, int n, int v){

int c, d;

if (v == 1){

std::cout << "vvedite granicy?" << std::endl;

std::cin >> c >> d;

for (int i = 0; i < n; i++){

a[i] = c + rand() % (d - c + 1);

A[i] = a[i];

std::cout << a[i] << " ";

}

}

if(v==2){

for (int i = 0; i < n; i++){

a[i] = i+1;

A[i] = a[i];

std::cout << a[i] << " ";

}

}

if (v == 3){

for (int i = 0; i < n; i++){

a[i] = n-i;

A[i] = a[i];

std::cout << a[i] << " ";

}

}

std::cout << std::endl;

}

void cin\_arr(int\* a, int\* A, int n)

{

for (int i = 0; i < n; i++)

{

std::cin >> a[i];

A[i] = a[i];

}

}

void fin\_arr(int\* a, int\* A, int n){

std::ifstream fin("input.txt");

if (fin.eof())

std:: cout << "empty file" << std::endl;

int i = 0;

while (!fin.eof() && (i < n)){

fin >> a[i];

A[i] = a[i];

i++;

}

}

**Таблица с результатами исследования**

****

Скриншот 2.1.3 «Таблица с результатами исследования»

## 2.2 Лабораторная работа 2

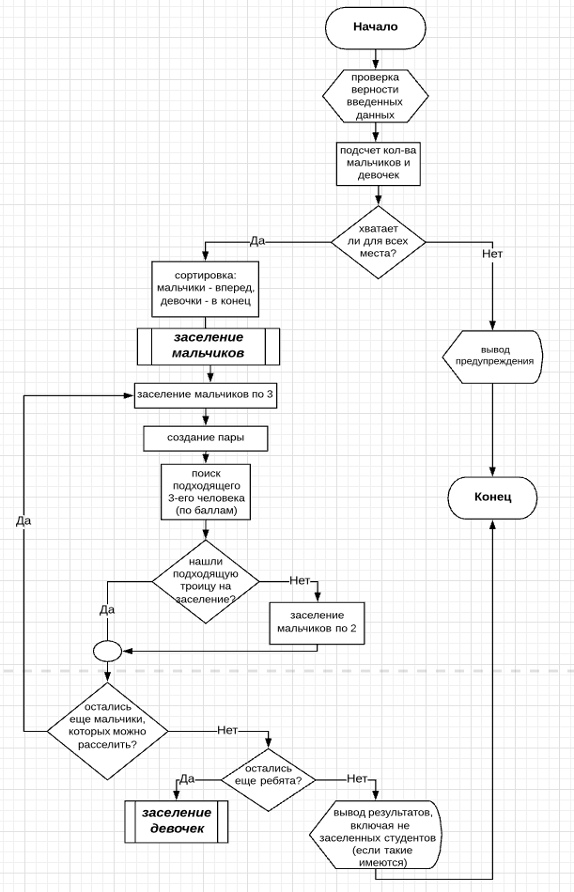
## Бинарные файлы

**Постановка задачи:**

Разработать систему заселения студенческого общежития. Необходимо подобрать студентов для проживания в одной комнате с учётом факультета, курса, специальности, индивидуальных характеристик. Информация о студенте содержит ФИО, номер студенческого, факультет, курс, специальность, спортивные интересы, хобби, номер комнаты (первоначально = 0) и хранится в бинарном файле. Все поля кроме ФИО – числовые, т.е. имеются их текстовые расшифровки в файлах-справочниках. При подборе состава комнаты нужно подобрать максимально близких по интересам людей. По возможности нужно учесть антипатии студентов друг к другу и не заселять таких стдентов вместе. Завершив расселение, нужно распечатать списки состава комнат краткие и , по запросу, с полными сведениями о студентах по всему общежитию, или для заданных комнат, а также списки тех студентов, которых не удалось распределить, если такие имеются. Кроме того, в исходный файл занести номера комнат, если студента заселели. Заранее известно, что количество студентов равно числу мест в комнатах. Комнаты есть двухместные и трёхместные.

Алгоритм.

Алгоритм представлен в виде блок-схемы, на которой изображена последовательность действий программы в зависимости от входных данных.



Скриншот 2.2.1 «Блок-схема»

## Код программы.

Данная часть программы отвечает за расселение мальчиков в комнаты по 3 (если такое предусмотрено входными данными).

Принцип следующий: для первого мальчика ищем наилучший вариант второго мальчика через подсчёт баллов (курс, факультет, специальность и т.д.). Затем для этой пары ищем наилучший вариант третьего мальчика также через подсчёт баллов (факультет, курс, специальность и т.д.). Получившуюся тройку мальчиков заселяем и выводим информацию об этом в файл. Дальше смотрим, остаслись ли у нас еще мальчики для заселения в комнаты по 3 (или остались ли такие комнаты). Если да, то проделываем такое заселение снова и снова. В итоге получаем мальчиков, заселённых в комнаты по 3.

for (int i = 0; i < vec.size() - 3 && go\_toboys>0; i += 3, go\_toboys--) {

sortCatalog(Catalog, i, vec, c\_facult, c\_specialyty, c\_curse, c\_potok, c\_group, c\_interests);

number\_find = i + 2;

number\_now = i + 2;

counter1 = 0;

counter2 = 0;

Maxcounter = 0;

for (int j = i + 2; j < vec.size(); j++) {

counter1 = 0;

counter2 = 0;

if (Catalog[i].sex != Catalog[j].sex) {

counter1 += -1000;

}

if (Catalog[i].interests != Catalog[j].interests) {

counter1 += c\_interests;

}

if (Catalog[i].facult == Catalog[j].facult) {

counter1 += c\_facult;

if (Catalog[i].specialety == Catalog[j].specialety) {

counter1 += c\_specialyty;

if (Catalog[i].curse == Catalog[j].curse) {

counter1 += c\_curse;

if (Catalog[i].potok == Catalog[j].potok) {

counter1 += c\_potok;

if (Catalog[i].group == Catalog[j].group) {

counter1 += c\_group;

}

}

}

}

else {

if (Catalog[i].curse == Catalog[j].curse) {

counter1 += c\_curse;

if (Catalog[i].potok == Catalog[j].potok) {

counter1 += c\_potok;

}

}

}

}

else {

if (Catalog[i].curse == Catalog[j].curse) {

counter1 += c\_curse;

if (Catalog[i].potok == Catalog[j].potok) {

counter1 += c\_potok;

}

}

}

if (Catalog[i + 1].sex != Catalog[j].sex) {

counter2 += -1000;

}

if (Catalog[i + 1].interests != Catalog[j].interests) {

counter2 += c\_interests;

}

if (Catalog[i + 1].facult == Catalog[j].facult) {

counter2 += c\_facult;

if (Catalog[i + 1].specialety == Catalog[j].specialety) {

counter2 += c\_specialyty;

if (Catalog[i + 1].curse == Catalog[j].curse) {

counter2 += c\_curse;

if (Catalog[i + 1].potok == Catalog[j].potok) {

counter2 += c\_potok;

if (Catalog[i + 1].group == Catalog[j].group) {

counter2 += c\_group;

}

}

}

else {

if (Catalog[i + 1].curse == Catalog[j].curse) {

counter2 += c\_curse;

if (Catalog[i + 1].potok == Catalog[j].potok) {

counter2 += c\_potok;

}

}

}

}

else {

if (Catalog[i + 1].curse == Catalog[j].curse) {

counter2 += c\_curse;

if (Catalog[i + 1].potok == Catalog[j].potok) {

counter2 += c\_potok;

}

}

}

if (Maxcounter < counter1 + counter2) {

Maxcounter = counter1 + counter2;

number\_find = number\_now;

}

number\_now++;

}

swap(vec[i + 2], vec[number\_find]);

swap(Catalog[i + 2].curse, Catalog[number\_find].curse);

swap(Catalog[i + 2].sex, Catalog[number\_find].sex);

swap(Catalog[i + 2].facult, Catalog[number\_find].facult);

swap(Catalog[i + 2].specialety, Catalog[number\_find].specialety);

swap(Catalog[i + 2].interests, Catalog[number\_find].interests);

swap(Catalog[i + 2].facult, Catalog[number\_find].facult);

swap(Catalog[i + 2].fio, Catalog[number\_find].fio);

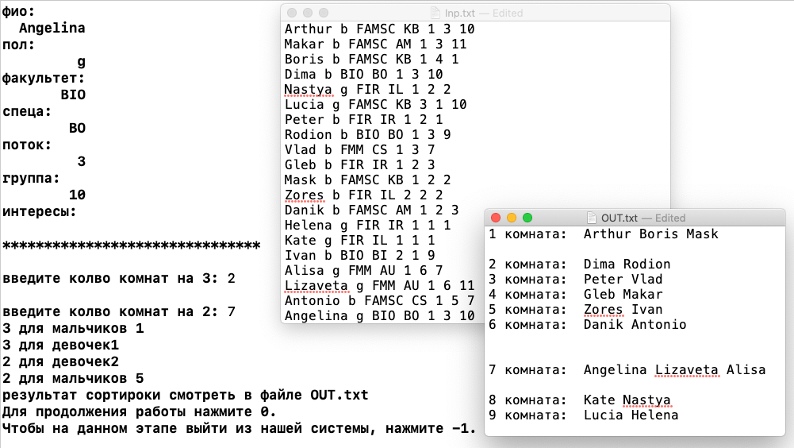
Output << rooms << " комната: " << " " << Catalog[i].fio << " " << Catalog[i + 1].fio << " " << Catalog[i + 2].fio << " \n";

rooms++;

}

**Результат выполнения.**

Результат заселения 18 студентов (13 мальчиков и 7 девочек) в 2 комнаты на 3 человека и 7 комнат на 2 человека представлен на скриншоте ниже.



Скриншот 2.2.2 «Успешное заселение»

## 2.3 Лабораторная работа 3

## Списки

**Постановка задачи:**

Задана полоска длиной 2^k клеток и шириной в одну клетку. Полоску сгибают пополам так, чтобы правая половинка оказалась под левой. Сгибание продолжают до тех пор, пока сверху находится больше одной клетки. Необходимо пронумеровать клетки таким образом, чтобы после окончания сгибания полосы номера клеток в получившейся колонки были расположены в порядке 1, 2, 3, 4, …, 2^k.

ЗАДАНИЕ НЕ ВЫПОЛНЕНО

## 2.4 Лабораторная работа 4

**Стеки и очереди**

**Постановка задачи:**

Имеется план местности, разбитой на квадраты, заданный матрицей размером NxN. Каждый квадарат имеет высоту относительно уровня моря, значение которой определяется натуральным числом. Необходимо определить маршрут каравана из позиции (XC, YC) в позицию (ХФ, УФ), при котором крутизна его подъёмов и спусков не превышает величины К. Караван может двигаться только по местности и только пораллельно осям ОХ, ОУмежду центрами квадратов. При переходе в соседний квадрат крутизна подъёма (спуска) равна модулю разности высот квадратов.

ЗАДАНИЕ НЕ ВЫПОЛНЕНО

## 2.5 Лабораторная работа 5

## Бинарное поисковое дерево

**Постановка задачи:**

Написать программу создания и обработки бинарного поискового дерева. Реализовать функции обхода, поиска и удаления по значению. Протестировать работу программы с выдачей результатов всех действий на экран.

**Решение поставленной задачи.**

**Код программы**

struct TreeItem {

int Info;

TreeItem\* Father;

TreeItem\* LSon;

TreeItem\* RSon;

TreeItem() { LSon = RSon = NULL; }

};

TreeItem\* Root = NULL;

bool find(TreeItem\* R, int a, TreeItem\*& t)

{

if (R == NULL){

t = NULL;

return false;

}

t = R;

for (;;){

if (t->Info == a)

return true;

if (t->Info > a){

if (t->LSon == NULL)

return false;

t = t->LSon;

}

else{

if (t->RSon == NULL)

return false;

t = t->RSon;

}

}

}

bool Insert(TreeItem\*& R, int info){

TreeItem\* s, \* q;

if (find(R, info, s))

return false;

q = new TreeItem;

q->Info = info;

if (s == NULL){

R = q;

q->Father = NULL;

}

else{

q->Father = s;

if (s->Info < info)

s->RSon = q;

else

s->LSon = q;

}

return true;

}

void print(TreeItem\*& s){

cout << s->Info << " ";

}

// обратный обход дерева

void post\_order\_walk(TreeItem\* s, void function(TreeItem\*& s)){

if (s != NULL){

post\_order\_walk(s->LSon, function);

post\_order\_walk(s->RSon, function);

function(s);

}

}

// обратный обход дерева

void pre\_order\_walk(TreeItem\* s, void function(TreeItem\*& s)){

if (s != NULL){

function(s);

pre\_order\_walk(s->LSon, function);

pre\_order\_walk(s->RSon, function);

}

}

// обратный обход дерева

void in\_order\_walk(TreeItem\* s, void function(TreeItem\*& s)){

if (s != NULL){

in\_order\_walk(s->LSon, function);

function(s);

in\_order\_walk(s->RSon, function);

}

}

// удаление листа дерева

void EraseItem(TreeItem\*& s){

if (s->Father != NULL){

if ((s->Father)->LSon == s)

(s->Father)->LSon = NULL;

else

(s->Father)->RSon = NULL;

}

else

Root = NULL; //если лист-корень

delete s;

s = NULL;

}

void post\_order(TreeItem\* R, void function(TreeItem\*& s)){

if (R == NULL){

cout << "empty tree" << endl;

return;

}

post\_order\_walk(R, function);

}

void pre\_order(TreeItem\* R, void function(TreeItem\*& s)){

if (R == NULL){

cout << "empty tree" << endl;

return;

}

pre\_order\_walk(R, function);

}

void in\_order(TreeItem\* R, void function(TreeItem\*& s)){

if (R == NULL){

cout << "empty tree" << endl;

return;

}

in\_order\_walk(R, function);

}

// удаление вершины, имеющей не более одного сына

void EraseNode(TreeItem\*& s){

TreeItem\* q;

if (s->LSon != NULL)

q = s->LSon;

else

q = s->RSon;

if (q != NULL)

q->Father = s->Father;

if (s->Father == NULL)

Root = q;

else

if ((s->Father)->LSon == s)

(s->Father)->LSon = q;

else

(s->Father)->RSon = q;

delete s; s = NULL;

}

// удаление вершины по значению

bool Erase(TreeItem\*& R, int info){

TreeItem\* s, \* q;

if (!find(R, info, s))

return false;

else{

if ((s->LSon != NULL) && (s->RSon != NULL)){

q = s->RSon;

while (q->LSon != NULL)

q = q->LSon;

s->Info = q->Info;

EraseNode(q);

}

else

EraseNode(s);

return true;

}

}

void Destroy(TreeItem\*& R)

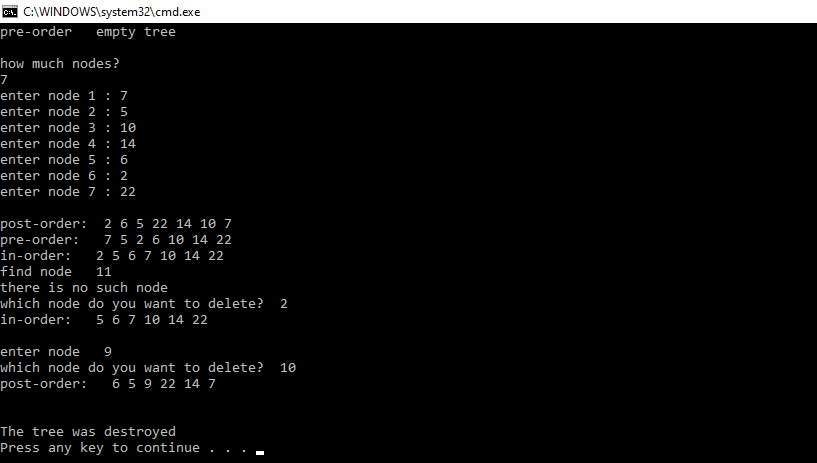
// уничтожение дерева

{

post\_order(R, EraseItem);

R = NULL;

}

**Результат выполнения.**

Скриншот 2.5.1 «Работа с БПД»

## 2.6 Лабораторная работа 6

## Шаблоны функций

**Постановка задачи:**

Разработать шаблон функции для обработки массивов с элементами простого арифметического типа и массивов, элементы которых – строки типа char\*. Продемонстрировать использование этого шаблона для конкретных динамических массивов типа int, double, char\*. Размерности массивов вводить с клавиатуры, значения элементов массивов генерировать случайным образом.

Разработать шаблон функции, позволяющей подсчитать количество различных элементов в последовательности.

**Решение поставленной задачи.**

**Код программы**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <string>

#include <cstring>

#include <time.h>

#include <typeinfo>

#pragma warning (disable:4996)

using namespace std;

const int nmax = 500;

template<typename T>

int count(T a[], int k)

{

int s = 0;

bool is = true;

for (int i = 0; i < k; i++)

{

is = true;

for (int j = 0; j < i; j++)

if (a[i] == a[j])

{

is = false;

break;

}

if (is == true)

s++;

}

return s;

}

template<>

int count(char\*\* arr, int n)

{

char\*\* mas = new char\* [n];

int s = 0;

bool f;

for (int i = 0; i < n; i++)

{

f = true;

for (int j = 0; j < s; j++)

{

if (!strcmp(arr[i], mas[j]))

f = false;

}

if (f == true)

{

mas[s] = new char[20];

strcpy(mas[s++], arr[i]);

}

}

return s;

}

void rand(char\*& v)

{

const char alpha[] = "bzqDUW507";

int size = 1 + rand() % 5;

char\* p = new char[size + 1];

v = p;

while (size > 0)

{

size--;

\*p = alpha[rand() % 11];

p++;

}

\*p = '\0';

}

int main()

{

srand(time(NULL));

//////////////INT

int\* arrint = new int[nmax];

cout << "Enter the size of the Array of INT ";

int n; cin >> n;

cout << endl << "Array INT:" << endl;

for (int i = 0; i < n; i++)

{

arrint[i] = -5 + rand() % 14;

cout << arrint[i] << " ";

}

cout << endl << "number of different elements = " << count(arrint, n) << endl

<< "==============================================================\n";

delete[] arrint;

//////////////DOUBLE

double\* arrdouble = new double[nmax];

cout << endl << "Enter the size of the Array of DOUBLE ";

cin >> n;

cout << endl << "Array DOUBLE:" << endl;

for (int i = 0; i < n; i++)

{

arrdouble[i] = (-5 + rand() % 20) / 10.0;

cout << arrdouble[i] << " ";

}

cout << endl << "number of different elements = " << count(arrdouble, n) << endl

<< "==============================================================\n";

delete[] arrdouble;

/////////////////CHAR

char\* arrchar = new char[nmax];

cout << endl << "Enter the size of the Array of CHAR ";

cin >> n;

cout << endl << "Array CHAR:" << endl;

for (int i = 0; i < n; i++)

{

arrchar[i] = char('a' + rand() % ('z' - 'a'));

cout << arrchar[i] << " ";

}

cout << endl << "number of different elements = " << count(arrchar, n) << endl

<< "==============================================================\n";

delete[] arrchar;

//////////////////////CHAR\*

///////////////////Вводите число элементов не меньше 25, чтобы генерировались различные строки

cout << endl << "Enter the size of the Array of CHAR\* ";

cin >> n;

cout << endl << "Array CHAR\*:" << endl;

char\*\* mas = new char\* [n];

for (int i = 0; i < n; i++)

{

mas[i] = new char[10];

rand(mas[i]);

cout << mas[i] << " ";

}

cout << endl << "number of different elements = " << count(mas, n) << endl

<< "==============================================================\n\n";

for (int i = 0; i < n; i++)

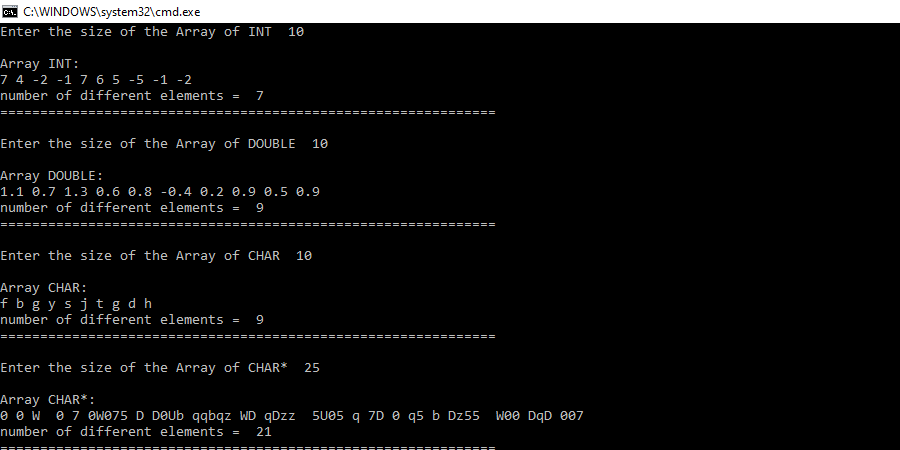
delete[] mas[i];

delete[] mas;

return 0;

}

**Результат выполнения.**



Скриншот 2.6.1 «Нахождение различных элементов в массивах»

## 2.7 Лабораторная работа 7

**Шаблоны классов**

**Постановка задачи:**

Реализовать шаблон класса “нестандартный дек” для целых типов и char\*.

**Решение поставленной задачи.**

**Код программы**

template<class TInfo>

class TDeque

{

protected:

struct TDequeItem{

TInfo info;

TDequeItem\* next;

TDequeItem\* prev;

TDequeItem() :next(NULL), prev(NULL) {};

};

TDequeItem\* first, \* last;

int size;

void erase();

void clone(const TDeque&);

void DeleteItem(TDequeItem\*);

void\* PtrByIndex(unsigned) const;

public:

TDeque();

TDeque(const TDeque&);

virtual ~TDeque();

void InsertFirst(TInfo);

void InsertLast(TInfo);

bool DeleteFirst();

bool DeleteLast();

const TDeque& operator=(const TDeque&);

const TInfo& GetByIndex(unsigned) const;

void SetByIndex(TInfo, unsigned);

void func(void func(TInfo&));

void func(void func(TInfo)) const;

};

//удаление очереди

template<class TInfo>

void TDeque<TInfo>::erase(){

while (DeleteLast());

size = 0;

}

//клонирование очереди

template<class TInfo>

void TDeque<TInfo>::clone(const TDeque& c){

TDequeItem\* q = c.first;

for (int i = 0; i < c.size; i++, q = q->next)

InsertLast(q->info);

size = c.size;

}

// удаление элемента по указателю

template<class TInfo>

void TDeque<TInfo>::DeleteItem(TDequeItem\* c){

(c->next)->prev = c->prev;

(c->prev)->next = c->next;

delete c;

size--;

}

//удаление элемента по указателю для Char\*

template<>

void TDeque<char\*>::DeleteItem(TDequeItem\* c){

(c->next)->prev = c->prev;

(c->prev)->next = c->next;

delete[] c;

delete c;

size--;

}

//получение указателя на элемент с заданным индексом

template<class TInfo>

void\* TDeque<TInfo>::PtrByIndex(unsigned n) const{

if (size == 0)

throw "Error! DEQUE is empty!";

if (n < 0 || n >= size)

return NULL;

TDequeItem\* q = first;

for (int i = 0; i < n; i++)

q = q->next;

return q;

}

// Конструктор без параметров

template<class TInfo>

TDeque<TInfo>::TDeque() : first(NULL), last(NULL), size(0) {}

// Конструктор копирования

template<class TInfo>

TDeque<TInfo>::TDeque(const TDeque& c) : first(NULL), last(NULL), size(0) { clone(c); }

// Деструктор

template<class TInfo>

TDeque<TInfo>::~TDeque() { erase(); }

// вставка элемента в начало очереди

template<class TInfo>

void TDeque<TInfo>::InsertFirst(TInfo c){

TDequeItem\* q = new TDequeItem;

q->info = c;

if (size == 0)

first = last = q;

else{

q->next = first;// новый элемент теперь указывает на первый элемент

first->prev = q;//новый элемент стал предыдущим для первого

first = q;//первым элементом теперь делаем новый элемент

}

size++;

}

// вставка элемента в начало очереди для char\*

template<>

void TDeque<char\*>::InsertFirst(char\* c){

TDequeItem\* q = new TDequeItem;

q->info = new char[strlen(c) + 1];

strcpy(q->info, c);

if (size == 0)

first = last = q;

else{

q->next = first;

first->prev = q;

first = q;

}

size++;

}

// Вставка элемента в конец очереди

template<class TInfo>

void TDeque<TInfo>::InsertLast(TInfo c){

TDequeItem\* q = new TDequeItem;

q->info = c;

if (size == 0)

first = last = q;

else{

q->prev = last;

last->next = q;

last = q;

}

size++;

}

// Вставка элемента в конец очереди для char\*

template<>

void TDeque<char\*>::InsertLast(char\* c){

TDequeItem\* q = new TDequeItem;

q->info = new char[strlen(c) + 1];

strcpy(q->info, c);

if (size == 0)

first = last = q;

else{

q->prev = last;

last->next = q;

last = q;

}

size++;

}

// Удаление элемента из начала очереди (голова)

template <typename TInfo>

bool TDeque <TInfo>::DeleteFirst(){

if (size == 0)

return false;

TDequeItem\* q = first;

if (size > 1)

first = first->next;

delete q;

--size;

first->prev = nullptr;

if (size == 0){

first = nullptr;

last = nullptr;

}

return true;

}

//Удаление элемента из конца очереди (хвост)

template <typename TInfo>

bool TDeque <TInfo>::DeleteLast(){

if (size == 0)

return false;

TDequeItem\* q = last;

last = last->prev;

delete q;

--size;

if (last)

last->next = nullptr;

if (size == 0){

first = nullptr;

last = nullptr;

}

return true;

}

// перегрузка оператора присваивания

template<class TInfo>

const TDeque<TInfo>& TDeque<TInfo>::operator=(const TDeque& c){

if (this == &c)

return \*this;

erase();

clone(c);

return \*this;

}

// Получение элемента очереди по индексу

template<class TInfo>

const TInfo& TDeque<TInfo>::GetByIndex(unsigned n) const{

if (size == 0)

throw "Error! DEQUE is empty!\n";

if (n < 0 || n >= (unsigned)size)

throw "Error: wrong index!\n";

TDequeItem\* q = (TDequeItem\*)PtrByIndex(n);

return q->info;

}

// вставка элемента в очередь по индексу

template<class TInfo>

void TDeque<TInfo>::SetByIndex(TInfo i, unsigned n){

if (size == 0)

throw "Error! DEQUE is empty!\n";

if (n < 0 || n >= (unsigned)size)

throw "Error: wrong index!\n";

TDequeItem\* q = (TDequeItem\*)PtrByIndex(n);

q->info = i;

}

// вставка элемента в очередь по индексу для Char\*

template<>

void TDeque<char\*>::SetByIndex(char\* i, unsigned n){

if (size == 0)

throw "Error! DEQUE is empty!\n";

if (n < 0 || n >= (unsigned)size)

throw "Error: wrong index!\n";

TDequeItem\* q = (TDequeItem\*)PtrByIndex(n);

delete[] q->info;

q->info = new char[strlen(i) + 1];

strcpy(q->info, i);

}

// просмотр очереди с изменением элементов

template<class TInfo>

void TDeque<TInfo>::func(void func(TInfo&)){

if (size == 0)

throw "Error! DEQUE is empty!\n";

TDequeItem\* q = first;

for (int i = 0; i < size; i++, q = q->next)

func(q->info);

}

// просмотр очереди без изменения элементов

template<class TInfo>

void TDeque<TInfo>::func(void func(TInfo)) const{

if (size == 0)

throw "Error! DEQUE is empty!\n";

TDequeItem\* q = first;

for (int i = 0; i < size; i++, q = q->next)

func(q->info);

}

// вывод очереди в поток

template<typename TInfo>

void Print(TInfo in){

cout << in << endl;

}

// функция просмотра очереди с изменением элементов

template<typename TInfo>

void Change(TInfo& a){

cout << a \* 10 << endl;

}

// функция просмотра очереди с изменением элементов для char\*

template<>

void Change(char\*& a){

char\* str = new char[strlen(a) + 5];

strcpy(str, a);

strcat((char\*)str, "+13");

a = new char[strlen(str) + 1];

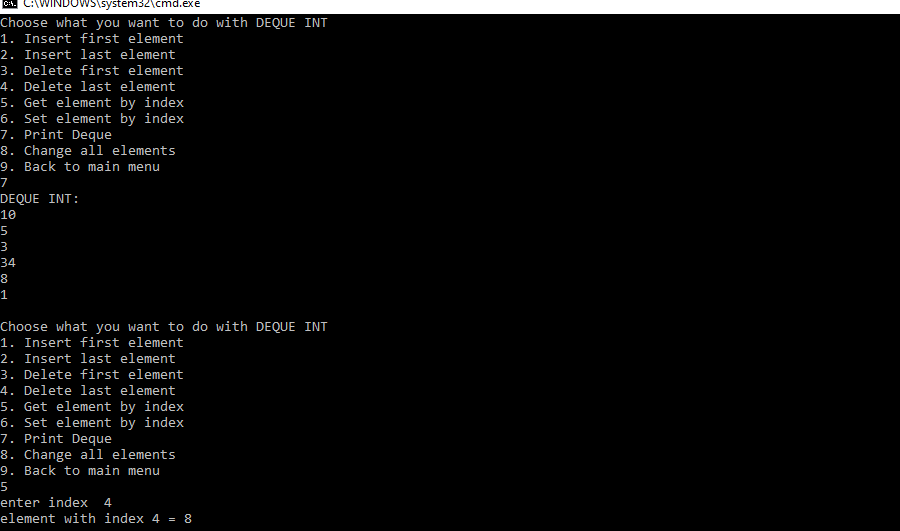
strcpy(a, str);

cout << a << endl;

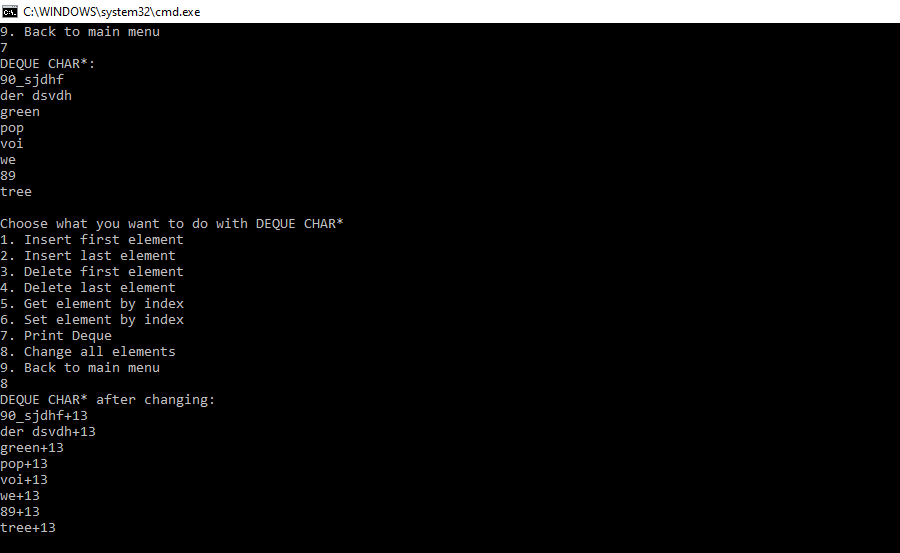
delete[] str;

# }

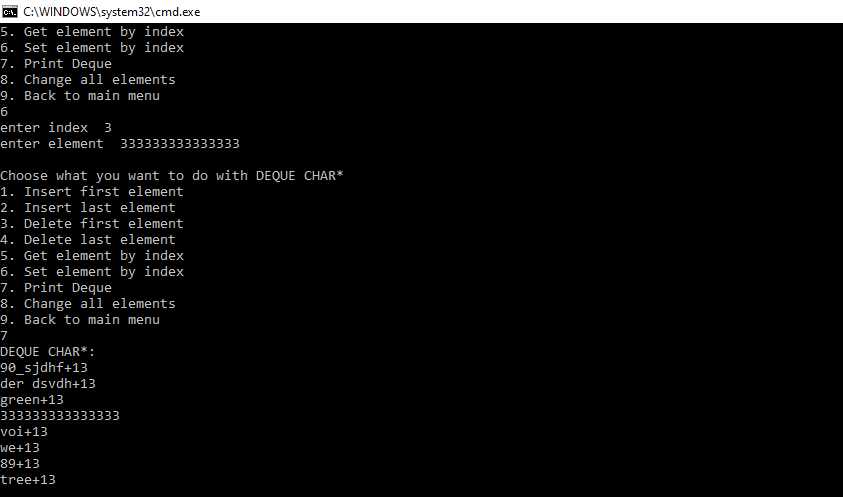
**Результаты выполнения.**



Скриншот 2.7.1 «Нахождение элемента по индексу (для int)»



Скриншот 2.7.2 «Изменение всех элементов дека (для char\*)»



Скриншот 2.7.3 «Вставка элемента по индексу(для char\*)»

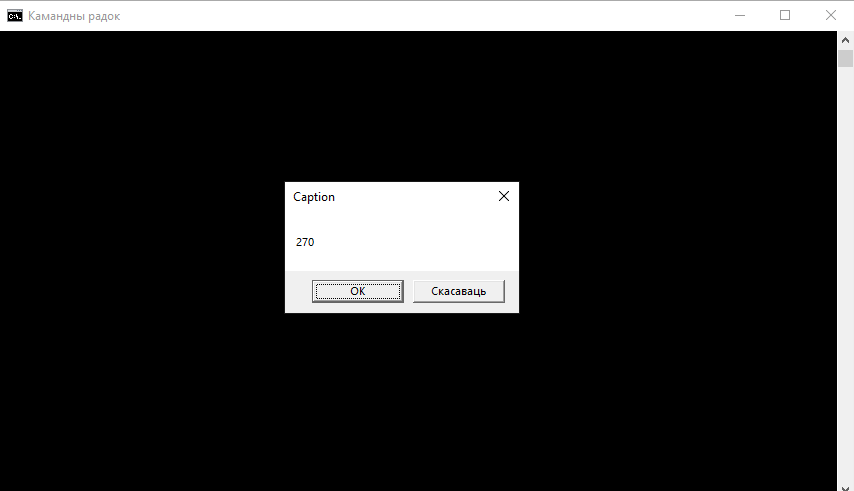
# 3 WinAPI

## 3.1 Лабораторная работа 1

## 3.1.1 Задание 1

**Постановка задачи.**

* В файле <winuser.h> найдите константы (MB\_), задающие кнопки в диалоговом окне. Используйте их.
* Учимся выводить информацию через окно сообщений:  
  задайте целочисленный параметр в командной строке. Выведите его с помощью функции MessageBox() (sprintf()).

**Скриншоты результатов работы. **

Скриншот Лабораторной работы 3.1.1

## 3.1.2 Задание 2

**Постановка задачи**.

В файле <winuser.h> найдите константы (IDC\_), которые являются идентификаторами стандартных курсоров. Используйте их.

Второй параметр функции ShowWindow(). Константы с префиксом SW\_.

Нарисуйте красным цветом прямоугольник, границы которого находятся на расстоянии 20 пикселей от границ клиентской области окна. Нарисуйте синим цветом эллипс, вписанный в клиентскую область окна. Нарисуйте красным цветом эллипс, вписанный в прямоугольник. Сделайте заливку внутреннего эллипса зеленым цветом. Проведите две диагонали для клиентской области.

## 3.1.3 Задание 3

**Постановка задачи**. Вывести текст (названия углов) в четырех углах клиентской области, а в центре – свою фамилию.

**Решение задачи.**

**Алгоритм.** Рисуем красный квадрат на расстоянии 20 пикселей от границ. Потом рисуем синий эллипс, вписанный в клиентскую область, и зеленый эллипс, вписанный в красный прямоугольник. Рисуем белые линии из противоположных углов. Делаем в углах надписи, а в центре пишем, кто автор.



Скриншот Лабораторной работы 3.1.3

**Фрагмент кода.**

case WM\_PAINT:

{

hDC = BeginPaint(hwnd, &PS);

SetBkMode(hDC, TRANSPARENT);

SetTextColor(hDC, RGB(200, 240, 200));

GetClientRect(hwnd, &Rect);

SelectObject(hDC, PRed);

SelectObject(hDC, BRed); //красный квадрат

Rectangle(hDC, Rect.left + 20, Rect.top + 20, Rect.right - 20, Rect.bottom - 20);

SelectObject(hDC, PBlue);

SelectObject(hDC, BBlue); //синий эллипс

Ellipse(hDC, Rect.left, Rect.top, Rect.right, Rect.bottom);

SelectObject(hDC, PRed);

SelectObject(hDC, BGreen); //зелёный эллипс

Ellipse(hDC, Rect.left + 20, Rect.top + 20, Rect.right - 20, Rect.bottom - 20);

SelectObject(hDC, GetStockObject(WHITE\_PEN));

MoveToEx(hDC, Rect.left, Rect.top, NULL);

LineTo(hDC, Rect.right, Rect.bottom); //первая диагональ

MoveToEx(hDC, Rect.right, Rect.top, NULL);

LineTo(hDC, Rect.left, Rect.bottom); //вторая диагональ

//надписи

DrawText(hDC, L" Left top", 9, &Rect, DT\_SINGLELINE | DT\_TOP | DT\_LEFT);

DrawText(hDC, L"Right top", 10, &Rect, DT\_SINGLELINE | DT\_TOP | DT\_RIGHT);

DrawText(hDC, L"Right bottom", 13, &Rect, DT\_SINGLELINE | DT\_BOTTOM | DT\_RIGHT);

DrawText(hDC, L" Left bottom", 12, &Rect, DT\_SINGLELINE | DT\_BOTTOM | DT\_LEFT);

SetTextColor(hDC, RGB(0, 0, 0));

DrawText(hDC, L"VOLKOV EVGENIJ", 15, &Rect, DT\_SINGLELINE | DT\_VCENTER | DT\_CENTER);

DeleteObject(BRed);

DeleteObject(BGreen);

DeleteObject(BBlue);

DeleteObject(PRed);

DeleteObject(PGreen);

DeleteObject(PBlue);

EndPaint(hwnd, &PS);

break;

}

## 

## 3.2 Лабораторная работа 2

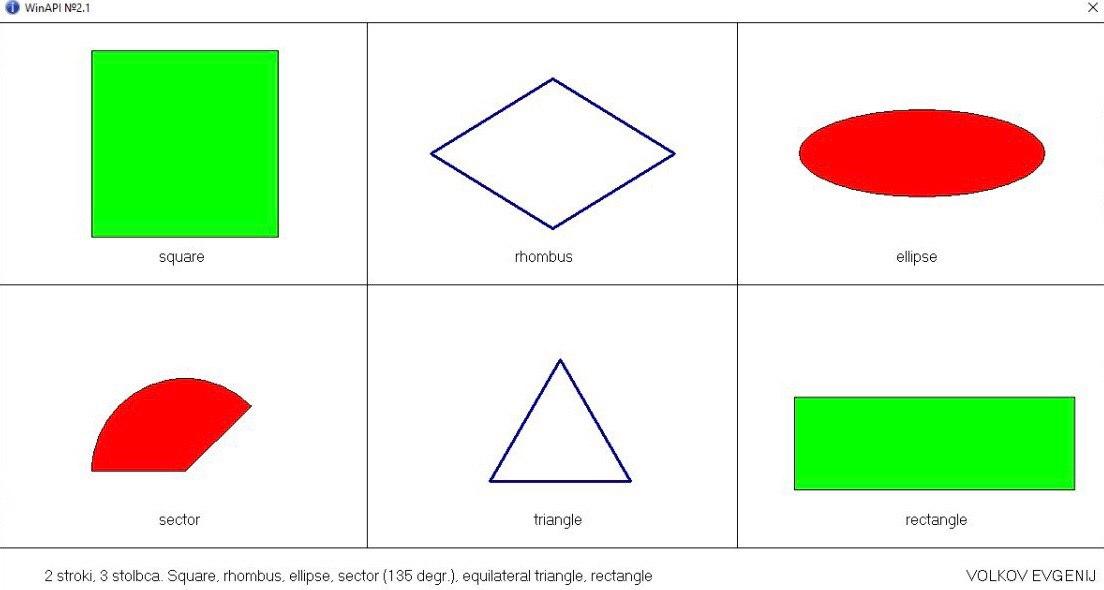
## 3.2.1 Задание 1

**Постановка задачи.**

Разбить клиентскую область на 6 частей (2 строки, 3 столбца) и вывести в них квадрат, ромб, эллипс, сектор (135 градусов), равносторонний треугольник, прямоугольник. Для построения треугольника и ромба использовать отрезки. Сделать подписи под каждым рисунком. Использовать параметры пера и кисти. В правом нижнем углу экрана написать свою фамилию и инициалы.

**Решение задачи.**

**Алгоритм.** Делим клиентскую область на 6 частей с помощью рисования линий. Затем в зависимости от её размеров рисуем фигуры и подписываем их.



Скриншот Лабораторной работы 3.2.1

**Фрагмент кода.**

case WM\_PAINT:

{

HPEN hBlue = CreatePen(PS\_DOT, 3, RGB(0, 0, 128));

hdc = BeginPaint(hwnd, &ps);

GetClientRect(hwnd, &clientRect);

hBlackPen = CreatePen(PS\_SOLID, 1, RGB(0, 0, 0));

hRedBrush = CreateSolidBrush(RGB(255, 0, 0));

hGreenBrush = CreateSolidBrush(RGB(0, 255, 0));

// разделение окна на части для рисования

MoveToEx(hdc,0,0, NULL);

LineTo(hdc, clientRect.right, 0);

MoveToEx(hdc, clientRect.left, (clientRect.bottom - 50) / 2, NULL);

LineTo(hdc, clientRect.right, (clientRect.bottom - 50) / 2);

MoveToEx(hdc, clientRect.left, clientRect.bottom -50, NULL);

LineTo(hdc, clientRect.right, clientRect.bottom -50);

MoveToEx(hdc, clientRect.right / 3, clientRect.top, NULL);

LineTo(hdc, clientRect.right / 3, clientRect.bottom-50);

MoveToEx(hdc, clientRect.right\*2 / 3, clientRect.top, NULL);

LineTo(hdc, clientRect.right\*2 / 3, clientRect.bottom-50);

//квадрат

SelectObject(hdc, hBlackPen);

SelectObject(hdc, hGreenBrush);

Rectangle(hdc, 100, 30, 300, 230);

//ромб

SelectObject(hdc, hBlue);

MoveToEx(hdc, clientRect.right / 2, clientRect.top + 60, NULL);

LineTo(hdc, clientRect.right / 2 + 130, (clientRect.bottom-50) / 4);

LineTo(hdc, clientRect.right / 2, (clientRect.bottom - 50) / 2-60);

LineTo(hdc, clientRect.right / 2 -130, (clientRect.bottom - 50) / 4);

LineTo(hdc, clientRect.right / 2, clientRect.top + 60);

//эллипс

SelectObject(hdc, hRedBrush);

SelectObject(hdc, hBlackPen);

Ellipse(hdc, clientRect.right\*13/18, (clientRect.bottom-50)/6, clientRect.right\*17/18, (clientRect.bottom - 50) / 3);

//сектор

SelectObject(hdc, hBlackPen);

Pie(hdc,100, 380, 300, 580, 271, 410, 100, 480);

//треугольник

SelectObject(hdc, hBlue);

MoveToEx(hdc, 600, 360, NULL);

LineTo(hdc, 525, 490);

LineTo(hdc, 675, 490);

LineTo(hdc, 600, 360);

//прямоугольник

SelectObject(hdc, hGreenBrush);

SelectObject(hdc, hBlackPen);

Rectangle(hdc, 850, 400, 1150, 500);

SelectObject(hdc, NewFont);

TextOut(hdc, clientRect.right \* 1 / 6 - 25, (clientRect.bottom - 50) / 2 - 40, "square", 6);

TextOut(hdc, clientRect.right / 2 -40, (clientRect.bottom - 50) / 2 - 40, "rhombus", 7);

TextOut(hdc, clientRect.right \* 2 / 3 +170, (clientRect.bottom - 50) / 2 - 40, "ellipse", 7);

TextOut(hdc, clientRect.right \* 1 / 6 - 25, clientRect.bottom - 90, "sector", 6);

TextOut(hdc, clientRect.right / 2 - 20, clientRect.bottom - 90, "triangle", 8);

TextOut(hdc, clientRect.right \* 2 / 3 + 180, clientRect.bottom - 90, "rectangle", 9);

TextOut(hdc, clientRect.left + 50, clientRect.bottom - 30, "2 stroki, 3 stolbca. Square, rhombus, ellipse, sector (135 degr.), equilateral triangle, rectangle",100);

TextOut(hdc, clientRect.right - 150, clientRect.bottom - 30, "VOLKOV EVGENIJ", 15);

DeleteObject(hBlackPen);

DeleteObject(hGreenBrush);

DeleteObject(hRedBrush);

EndPaint(hwnd, &ps);

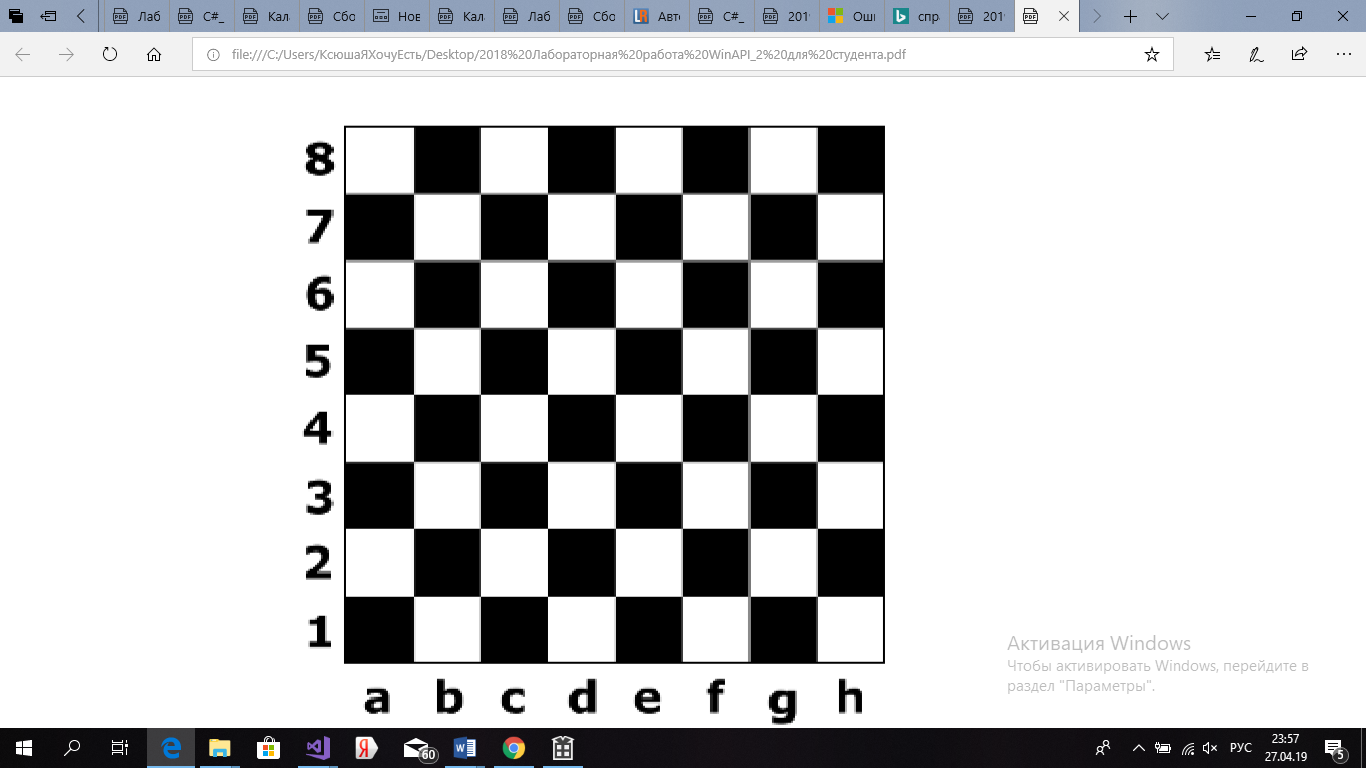
break;

}

## 3.2.2 Задание 2

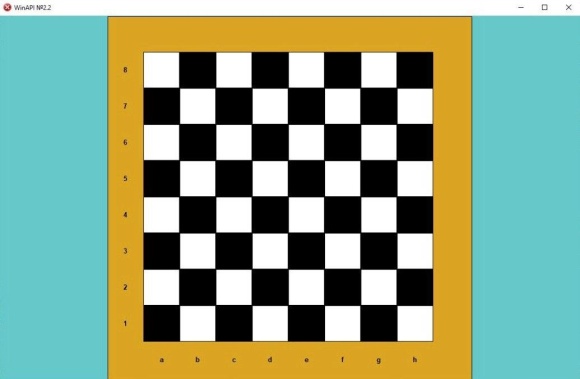
**Постановка задачи.**

Нарисовать шахматную доску:



**Решение задачи.**

**Алгоритм.** Выбираем меньшую из сторон клиентской области и делаем её стороной шахматной доски. Делим доску на 100 квадратов. Средние 64 раскрашиваем, как шахматную доску (если в сумме их номера – чётное число, то чёрная клетка, если нечётное число – белая). Слева в квадратах пишем цифры (от 1 до 8), а снизу буквы (от a до h).



Скриншот Лабораторной работы 3.2.2

**Фрагмент кода**

// Функция рисования шахматной доски

void DrawChess(HDC hdc, const RECT& rect)

{

// Создание кистей

HBRUSH board = CreateSolidBrush(RGB(218, 165, 32)); // цвет фона шахматной доски

HBRUSH white\_kl = CreateSolidBrush(RGB(255, 255, 255)); // цвет белой клетки

HBRUSH black\_kl = CreateSolidBrush(RGB(0, 0, 0)); // цвет чёрной клетки

int width = rect.right - rect.left + 1;

int height = rect.bottom - rect.top + 1;

int sx = 0; int sy = 0; // кооринаты левого верхнего угла доски

if (width > height)

{

sx = (width - height) / 2;

width = height;

}

else

{

sy = (height - width) / 2;

height = width;

}

int cell\_width = width / 10; // ширина одной клетки

int cell\_height = height / 10; // высота одной клетки

// Отрисовка фона доски

SelectObject(hdc, board);

Rectangle(hdc, sx + 1, sy + 1, sx + width - 2, sy + height - 2);

// Отрисовка всех клеток доски

for (int i = 1; i < 9; ++i)

for (int j = 1; j < 9; ++j)

{

// Выбор цвета клетки

if ((i + j) % 2 != 0)

SelectObject(hdc, black\_kl);

else

SelectObject(hdc, white\_kl);

// Отрисовка одной клеточки

int x = sx + i \* cell\_width;

int y = sy + j \* cell\_height;

Rectangle(hdc, x, y, x + cell\_width, y + cell\_height);

}

// Установка прозрачного фона у текста

SetBkMode(hdc, TRANSPARENT);

RECT rect\_for\_letter;

char letter;

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

{

// Горизонтальные a-h

letter = i + 'a';

rect\_for\_letter.left = sx + (i + 1) \* cell\_width;

rect\_for\_letter.right = rect\_for\_letter.left + cell\_width;

rect\_for\_letter.top = sy + 9 \* cell\_height;

rect\_for\_letter.bottom = rect\_for\_letter.top + cell\_height;

DrawText(hdc, &letter, 1, &rect\_for\_letter, DT\_CENTER | DT\_VCENTER | DT\_SINGLELINE);

// Вертикальные 1-8

letter = i + '1';

rect\_for\_letter.left = sx;

rect\_for\_letter.right = rect\_for\_letter.left + cell\_width;

rect\_for\_letter.top = sy + (8 - i) \* cell\_height;

rect\_for\_letter.bottom = rect\_for\_letter.top + cell\_height;

DrawText(hdc, &letter, 1, &rect\_for\_letter, DT\_CENTER | DT\_VCENTER | DT\_SINGLELINE);

}

DeleteObject(board);

DeleteObject(white\_kl);

DeleteObject(black\_kl);

}

### 

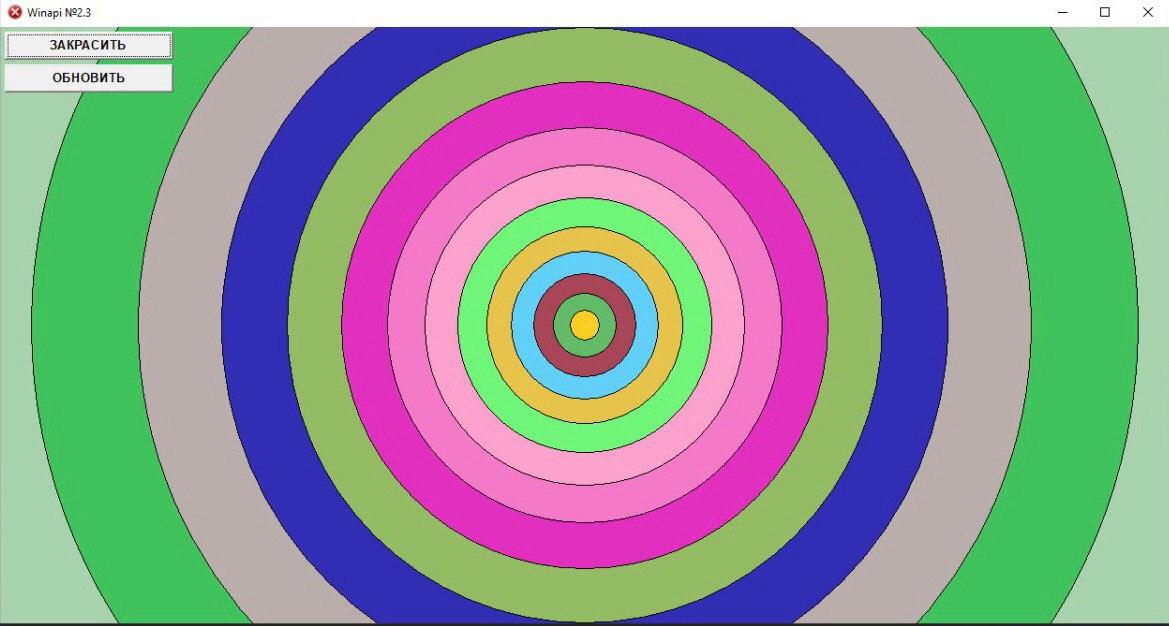
## 3.2.3 Задание 3

**Постановка задачи.**

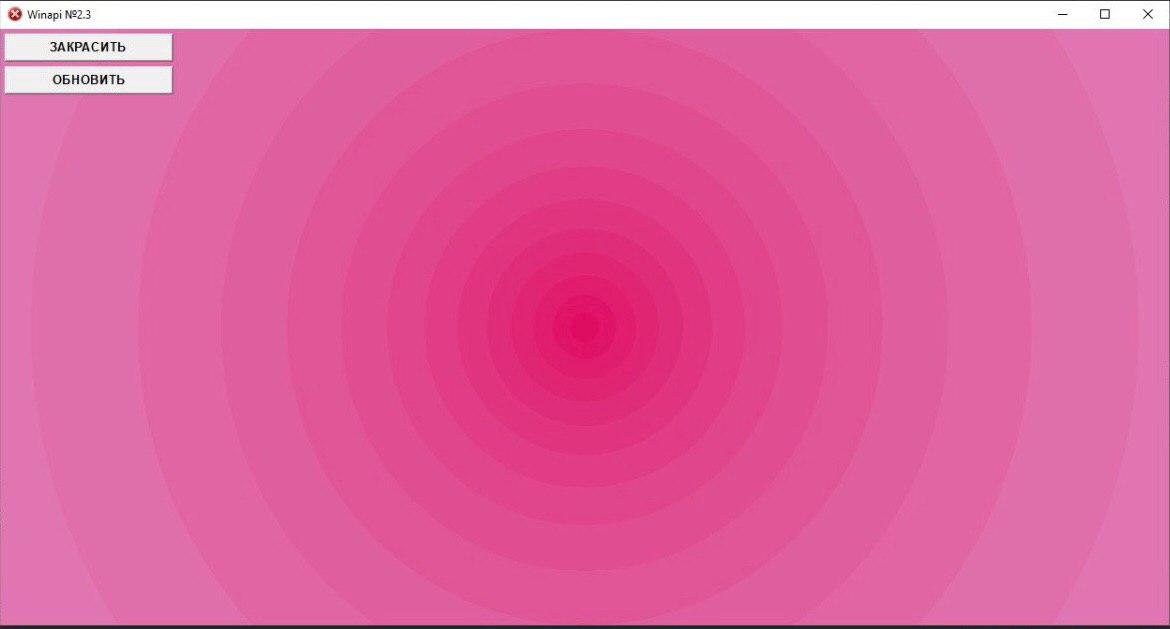
Разработать приложение, в котором в окне изобразить несколько вложенных окружностей разных цветов с общим центром и две кнопки. При нажатии кнопки ЗАКРАСИТЬ сделать заливку окружностей контрастными цветами. При нажатии кнопки ОБНОВИТЬ убрать границы и используя близкие по значениям цвета для заливки, получить эффект плавного перехода.

**Решение задачи.**

**Алгоритм.** Рисуем 19 окружностей и закрашиваем их в случайный цвет при нажатии на кнопку ЗАКРАСИТЬ. При нажатии на кнопку ОБНОВИТЬ создаём 2 случайных цвета и делаем плавный переход между ними, добавляя к каждой составляющей цвета (Red, Green, Blue) каждой следующей окружности её номер.



Скриншот Лабораторной работы 3.2.3(1)



Скриншот Лабораторной работы 3.2.3(2)

**Фрагмент кода**

case WM\_PAINT:

hdc = BeginPaint(hwnd, &ps);

RECT rect, recttemp;

int rght, bttm;

GetClientRect(hwnd, &rect);

rght = rect.right;

bttm = rect.bottom;

for (int i = 1; i < 20; i++)

{

HBRUSH Brush = CreateSolidBrush(RGB(rand() % 255, rand() % 255, rand() % 255));

SelectObject(hdc, Brush);

recttemp.top = bttm \* 10 / i;

recttemp.bottom = bttm - recttemp.top;

int lft;

lft = (rght - (bttm - 2 \* recttemp.top)) / 2;

recttemp.right = rght - lft;

recttemp.left = lft;

Ellipse(hdc, recttemp.left, recttemp.top, recttemp.right, recttemp.bottom);

DeleteObject(Brush);

}

// рисуем окружности случайным цветом

if (K == 1)

{

rght = rect.right;

bttm = rect.bottom;

for (int i = 1; i < 20; i++)

{

HBRUSH Brush = CreateSolidBrush(RGB(rand() % 255, rand() %255, rand() % 255));

SelectObject(hdc, Brush);

recttemp.top = bttm \* 10 / i;

recttemp.bottom = bttm - recttemp.top;

int lft;

lft = (rght - (bttm - 2 \* recttemp.top)) / 2;

recttemp.right = rght - lft;

recttemp.left = lft;

Ellipse(hdc, recttemp.left, recttemp.top, recttemp.right, recttemp.bottom);

DeleteObject(Brush);

}

}

// плавный переход от самой большой окружности к самой маленькой

if (K == 2)

{

rght = rect.right;

bttm = rect.bottom;

COLORREF COLOR1 = RGB(rand() % 255, rand() % 255, rand() % 255);

COLORREF COLOR2 = RGB(rand() % 255, rand() % 255, rand() % 255);

int R1 = GetRValue(COLOR1);

int R2 = GetRValue(COLOR2);

int G1 = GetGValue(COLOR1);

int G2 = GetGValue(COLOR2);

int B1 = GetBValue(COLOR1);

int B2 = GetBValue(COLOR2);

int R, G, B;

R = (R2 - R1) / 12;

G = (G2 - G1) / 12;

B = (B2 - B1) / 12;

for (int i = 1; i < 20; i++)

{

HBRUSH Brush = CreateSolidBrush(RGB(R1 + i \* R, G1 + i \* G, B1 + i \* B));

HPEN Pen = CreatePen(PS\_SOLID, 1, RGB(R1 + i \* R, G1 + i \* G, B1 + i \* B));

SelectObject(hdc, Brush);

SelectObject(hdc, Pen);

recttemp.top = bttm \* 10 / i;

recttemp.bottom = bttm - recttemp.top;

int lft;

lft = (rght - (bttm - 2 \* recttemp.top)) / 2;

recttemp.right = rght - lft;

recttemp.left = lft;

Ellipse(hdc, recttemp.left, recttemp.top, recttemp.right, recttemp.bottom);

DeleteObject(Brush);

DeleteObject(Pen);

}

}

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case 9:

K = 1;

InvalidateRect(hwnd, NULL, TRUE);

break;

case 10:

K = 2;

InvalidateRect(hwnd, NULL, TRUE);

break;

}

EndPaint(hwnd, &ps);

break;

## 3.2.4 Задание 4

**Постановка задачи.**

Танграм - это головоломка, которая представляет из себя квадрат, разрезанный на 7 частей определенным образом. Цель игры заключается в том, чтобы собирать из деталей танграма фигуры людей, животных, птиц, цифр, предметов…

*Правила игры танграм:*

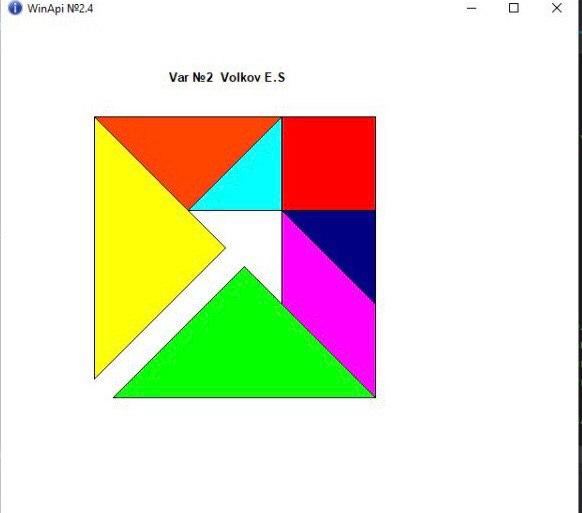
* В собранную фигуру должны входить все семь частей.
* Части не должны налегать друг на друга.
* Части должны примыкать друг к другу.

Необходимо нарисовать фигуру, соответствующую вашему варианту, над рисунком по центру окна написать:

Вариант № 2 Волков Е.С.

**Решение задачи.**

**Алгоритм.** Рисуем желтый треугольник, потом идем вверх и рисуем оранжевый треугольник. Затем голубой треугольник, красный квадрат, синий треугольник и фиолетовый параллелограмм. Оставшуюся часть не трогаем. Это и будет белая стрелочка.



Скриншот Лабораторной работы 3.2.4

**Фрагмент кода**

void Draw(HDC hdc, const RECT& rectClient)

{

HPEN hPen = CreatePen(PS\_SOLID, 1, RGB(0, 0, 0)); // чёрная граница

HBRUSH

hRed = CreateSolidBrush(RGB(255, 0, 0)), //для красного квадрата

hOrange = CreateSolidBrush(RGB(255, 69, 0)), // для оранжевого треугольника

hYellow = CreateSolidBrush(RGB(255, 255, 0)), // для жёлтого треугольника

hGreen = CreateSolidBrush(RGB(0, 255, 0)), // для зелёного треугольника

hLightBlue = CreateSolidBrush(RGB(0, 255, 255)), // для голубого треугольника

hBlue = CreateSolidBrush(RGB(0, 0, 128)), // для синего треугольника

hPurple = CreateSolidBrush(RGB(255, 0, 255)); // для фиолетового параллелограмма

POINT poly[4];

//жёлтый треугольник

SelectObject(hdc, hPen);

poly[0].x = 100;

poly[0].y = 100;

poly[1].x = 100;

poly[1].y = 380;

poly[2].x = 240;

poly[2].y = 240;

SelectObject(hdc, hYellow);

Polygon(hdc, poly, 3);

//оранжевый треугольник

SelectObject(hdc, hPen);

poly[0].x = 100;

poly[0].y = 100;

poly[1].x = 300;

poly[1].y = 100;

poly[2].x = 200;

poly[2].y = 200;

SelectObject(hdc, hOrange);

Polygon(hdc, poly, 3);

//голубой треугольник

SelectObject(hdc, hPen);

poly[0].x = 200;

poly[0].y = 200;

poly[1].x = 300;

poly[1].y = 100;

poly[2].x = 300;

poly[2].y = 200;

SelectObject(hdc, hLightBlue);

Polygon(hdc, poly, 3);

//красный квадрат

SelectObject(hdc, hPen);

poly[0].x = 300;

poly[0].y = 100;

poly[1].x = 400;

poly[1].y = 100;

poly[2].x = 400;

poly[2].y = 200;

poly[3].x = 300;

poly[3].y = 200;

SelectObject(hdc, hRed);

Polygon(hdc, poly, 4);

//синий треугольник

SelectObject(hdc, hPen);

poly[0].x = 300;

poly[0].y = 200;

poly[1].x = 400;

poly[1].y = 200;

poly[2].x = 400;

poly[2].y = 300;

SelectObject(hdc, hBlue);

Polygon(hdc, poly, 3);

//фиолетовый параллелограмм

SelectObject(hdc, hPen);

poly[0].x = 300;

poly[0].y = 200;

poly[1].x = 400;

poly[1].y = 300;

poly[2].x = 400;

poly[2].y = 400;

poly[3].x = 300;

poly[3].y = 300;

SelectObject(hdc, hPurple);

Polygon(hdc, poly, 4);

//зелёный треугольник

SelectObject(hdc, hPen);

poly[0].x = 400;

poly[0].y = 400;

poly[1].x = 120;

poly[1].y = 400;

poly[2].x = 260;

poly[2].y = 260;

SelectObject(hdc, hGreen);

Polygon(hdc, poly, 4);

DeleteObject(hRed);

DeleteObject(hOrange);

DeleteObject(hYellow);

DeleteObject(hGreen);

DeleteObject(hLightBlue);

DeleteObject(hBlue);

DeleteObject(hPurple);

DeleteObject(hPen);

}

## 3.3 Лабораторная работа 3

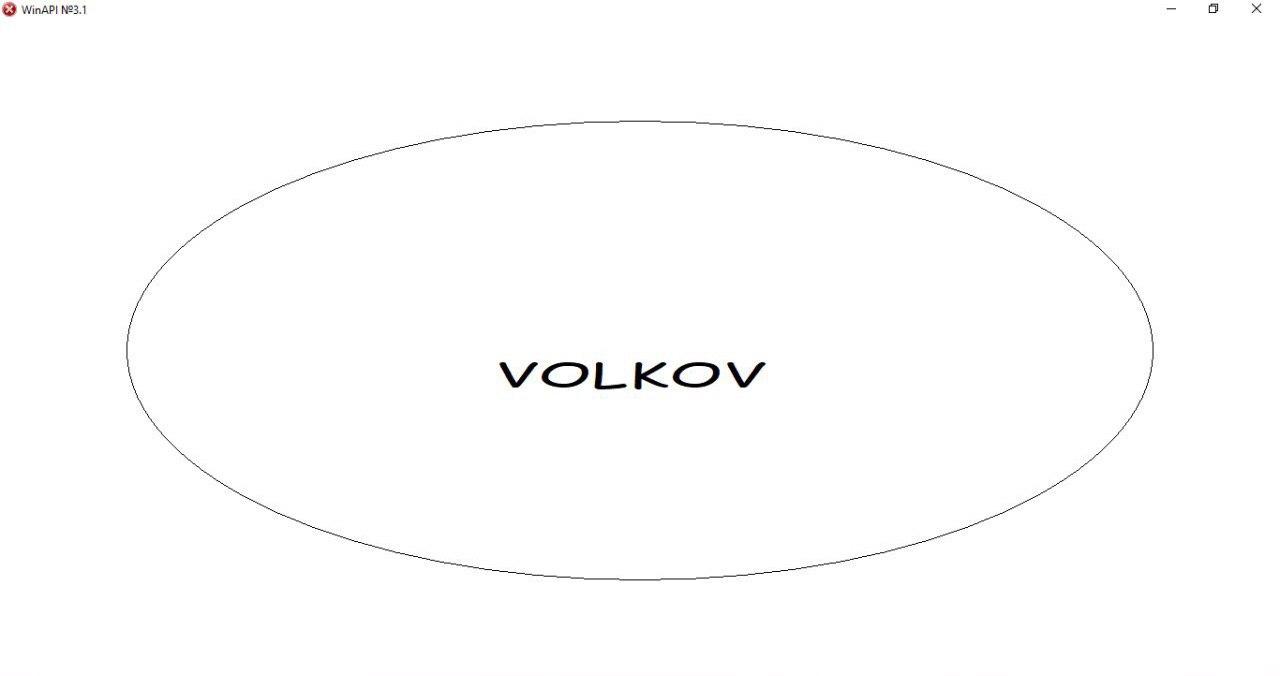
## 3.3.1 Задание 1

**Постановка задачи.**

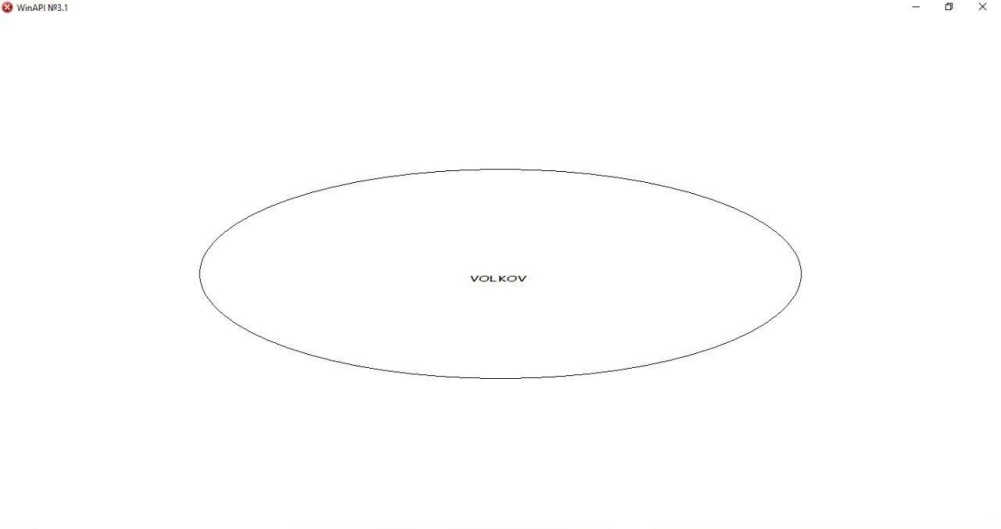
Написать программу, строящую эллипс в центре окна. В центре – ваша фамилия. При нажатии на левую кнопку мыши размеры эллипса и текста уменьшаются, а при нажатии на правую кнопку — увеличиваются.

**Решение задачи.**

**Алгоритм.** Изначально рисуем эллипс и надпись в нём. При нажатии на левую кнопку мыши умножаем размеры эллипса на 0,95, а размеры текста на 0,9 (т.е. картинка уменьшается). При нажатии на правую кнопку мыши умножаем на 1,05 размеры эллипса и на 1,1 размеры текста, что обеспечивает увеличение картинки.



Скриншот Лабораторной работы 3.3.1(1)



Скриншот Лабораторной работы 3.3.1(2)

**Фрагмент кода**

case WM\_PAINT:

{

HFONT hFont, Font1;

PAINTSTRUCT ps;

HDC hDC;

hDC = BeginPaint(hwnd, &ps);

hFont = CreateFont(tex/2, tex / 3, 0, 0, FW\_NORMAL, FALSE, FALSE, FALSE, DEFAULT\_CHARSET, OUT\_DEFAULT\_PRECIS, CLIP\_DEFAULT\_PRECIS, DEFAULT\_QUALITY, VARIABLE\_PITCH, "Comic Sans MS");

Font1 = (HFONT)SelectObject(hDC, hFont);

Ellipse(hDC, x+50, y+50, rect.right-x-50, rect.bottom - y-50);

SetBkMode(hDC, TRANSPARENT);

TextOut(hDC, rect.right/2-1.5\*tex, rect.bottom/2, "VOLKOV", 7);

SelectObject(hDC, Font1);

DeleteObject(hFont);

DeleteObject(Font1);

EndPaint(hwnd, &ps);

break;

}

case WM\_LBUTTONDOWN:

tex \*= 0.9;

x \*= 1.05;

y \*= 1.05;

InvalidateRect(hwnd, NULL, true);

break;

case WM\_RBUTTONDOWN:

tex \*= 1.1;

x \*= 0.95;

y \*= 0.95;

InvalidateRect(hwnd, NULL, true);

break;

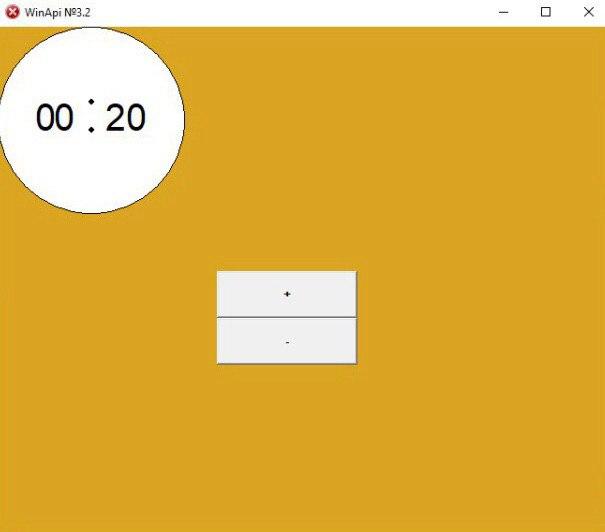
## 3.3.2 Задание 2

**Постановка задачи.**

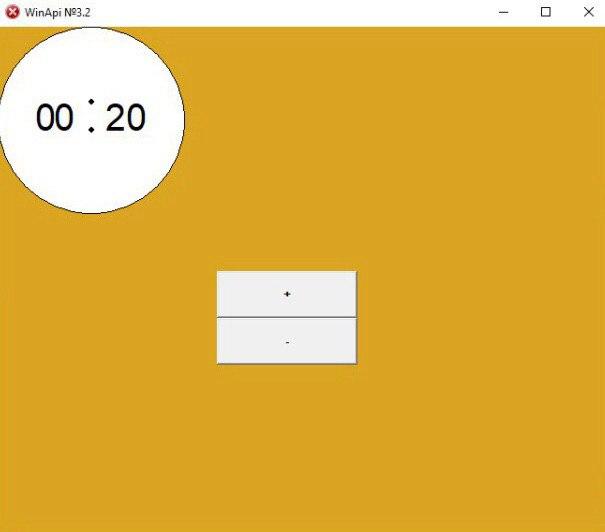
Построить электронный секундомер. Окружность в левом нижнем углу окна. Запускается секундомер клавишей “Enter”, его можно остановить (клавиша «-») и снова запустить (клавиша «+»)

**Решение задачи.**

**Алгоритм.** Рисуем в левом верхнем углу окружность. Создаём 2 кнопки. При срабатывании секундомера запоминаем количество минут и секунд. Если секунд 60, то увеличиваем минуты и обнуляем секунды. Прорисовываем цифры.



Скриншот Лабораторной работы 3.3.2(1)



Скриншот Лабораторной работы 3.3.2(2)

**Фрагмент кода**

//создание кнопок + и -

CreateWindow("BUTTON", "+", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, rect.right-150, rect.bottom-100, 150, 50, hWnd, (HMENU)33, hInstance, NULL);

CreateWindow("BUTTON", "-", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, rect.right - 150, rect.bottom-50, 150, 50, hWnd, (HMENU)66, hInstance, NULL);

//прорисовка секундомера

case WM\_CREATE:

SetTimer(hWnd, 1, 1000, NULL);

KillTimer(hWnd, 1);

InvalidateRect(hWnd, NULL, TRUE);

break;

case WM\_KEYDOWN:

if (wParam == 13)

{

SetTimer(hWnd, 1, 1000, NULL);

InvalidateRect(hWnd, NULL, TRUE);

break;

}

break;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case 33:

SetTimer(hWnd, 1, 1000, NULL);

InvalidateRect(hWnd, NULL, TRUE);

break;

case 66:

KillTimer(hWnd, 1);

InvalidateRect(hWnd, NULL, TRUE);

break;

}

break;

case WM\_TIMER:

{

Sec++;

InvalidateRect(hWnd, NULL, TRUE);

if (Sec >= 60)

{

Min++;

Sec = 0;

}

break;

}

case WM\_PAINT:

if (time)

{

char c[30];

HBRUSH Hbr = CreateSolidBrush(RGB(0,0,0));

HFONT NewFont = CreateFontW(45, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL);

hdc = BeginPaint(hWnd, &ps);

Ellipse(hdc,0, 0, 200, 200);

SelectObject(hdc, Hbr);

Ellipse(hdc,97, 77,103,83);

Ellipse(hdc, 97, 107, 103, 113);

SelectObject(hdc, NewFont);

SetBkMode(hdc, TRANSPARENT);

if (Sec < 10)

{

sprintf(c, "%d", 0);

TextOut(hdc, rect.left + 115, rect.top + 75, c, 2);

sprintf(c, "%d", Sec);

TextOut(hdc, rect.left + 135, rect.top + 75, c, 2);

}

else

{

sprintf(c, "%d", Sec);

TextOut(hdc, rect.left + 115, rect.top + 75, c, 2);

}

if (Min < 10)

{

sprintf(c, "%d", 0);

TextOut(hdc, rect.left + 40, rect.top + 75, c, 2);

sprintf(c, "%d", Min);

TextOut(hdc, rect.left + 60, rect.top + 75, c, 2);

}

else

{

sprintf(c, "%d", Min);

TextOut(hdc, rect.left + 40, rect.top + 75, c, 2);

}

DeleteObject(Hbr);

DeleteObject(NewFont);

EndPaint(hWnd, &ps);

}

break;

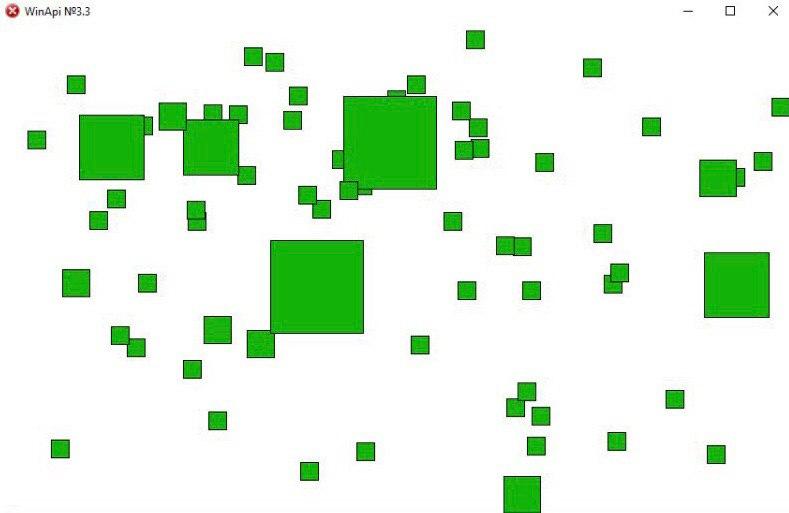
## 3.3.3 Задание 3

**Постановка задачи.**

Программа «квадратики» - на месте щелчка левой клавишей мыши появлятся квадратик начального размера. При повторных щелчках внутри квадратика его сторона увеличивается на фиксированную величину, вне квадратика – рисуем новый. Обеспечить отображение всех квадратиков при сворачивании/разворачивании окна (хранить координаты щелчков мыши.)

**Решение задачи.**

**Алгоритм.** Создаём вектор наших квадратиков, в котором храним координаты (по х и у) и сторону. При очередном нажатии левой кнопкой мыши определяем координаты, куда мы нажали, а потом пробегаемся по вектору. Если наши координаты оказались внутри какого-то квадрата, то увеличиваем его размер, если нет такого квадрата в векторе, то создаём новый и записываем в вектор. С каждым новым нажатием случайно меняем цвет всех квадратиков.

****

## Скриншот Лабораторная работа 3.3.3

**Фрагмент кода**

struct Square

{

int x, //координата х квадратика

y, //координата у квадратика

r; //сторона квадратика

Square() : r(10) {}

Square(int X, int Y) : r(10), x(X), y(Y) {}

};

vector<Square> Sqr; //храним координаты квадратиков и их размеры в векторе

//проверка на то, куда мы тыкаем курсором

void check(int x, int y)

{

bool f = false;

for (int i = 0; i < Sqr.size(); ++i)

if (!(Sqr[i].x + Sqr[i].r < x || Sqr[i].y + Sqr[i].r < y || Sqr[i].x - Sqr[i].r > x || Sqr[i].y - Sqr[i].r > y))

{

Sqr[i].r += 5; //увеличиваем квадратик

f = true;

}

if (!f)

Sqr.push\_back(Square(x, y));

}

…

case WM\_PAINT:

{

srand(time(NULL));

hdc = BeginPaint(hwnd, &ps);

HBRUSH hbr = CreateSolidBrush(RGB(rand() % 255, rand() % 255, rand() % 255));

SelectObject(hdc, hbr);

for (int i = 0; i < Sqr.size(); ++i)

Rectangle(hdc, Sqr[i].x - Sqr[i].r, Sqr[i].y - Sqr[i].r, Sqr[i].x + Sqr[i].r, Sqr[i].y + Sqr[i].r);

DeleteObject(hbr);

EndPaint(hwnd, &ps);

}

break;

## 3.4 Лабораторная работа 4

## 3.4.1 Задание 1

**Постановка задачи.**

Автобус движется по клиентской области справо налево, когда уходит за область, то разворачивается и возвращается. Дополнительные эффекты:

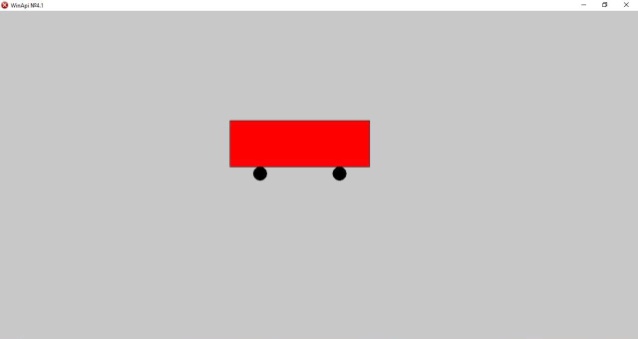
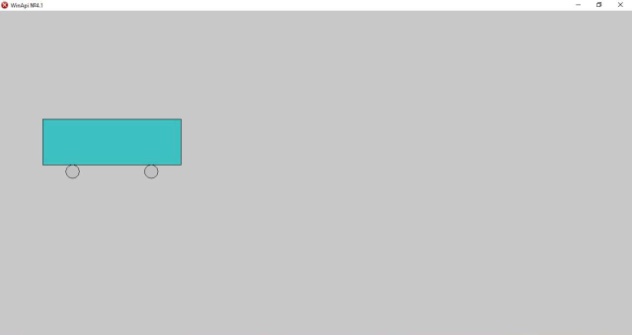
* движение с ускорением/замедлением;
* изменение цвета при приближении к краю;
* движение по наклонной;

Обрабатывать сообщения:

* WM\_CREATE – создать таймер;
* WM\_PAINT – нарисовать объект в текущей позиции;
* WM\_TIMER – изменить текущую позицию объект и сообщить о необходимости обновить клиентскую область.

**Решение задачи.**

**Алгоритм.** Изначально рисуем красный автобус с черными колёсами. При приближении к левой границе плавно меняем цвет. При достижении левого края меняем направление движения. При приближении к правому краю плавно меняем цвет на исходный (красный) и также меняем направление движения, если достигли правого края. На правом скриншоте показано начальное положение, на левом скриншоте автобус приближается к левой границе.



Скриншоты Лабораторная работа 3.4.1

**Фрагмент кода**

case WM\_CREATE:

{

SetTimer(hWnd, 1, 30, NULL);

x = rect.right / 2 - 50;

InvalidateRect(hWnd, NULL, TRUE);

break;

}

case WM\_TIMER:

if (left)

i += 2;

else

i -= 2;

InvalidateRect(hWnd, NULL, TRUE);

break;

case WM\_PAINT:

{

if (x - i == rect.left)

left = false;

if (x - i + 300 == rect.right)

left = true;

if (x - i < 100)

j++;

if (x - i+300>rect.right-100)

j--;

HBRUSH hR = CreateSolidBrush(RGB(255-2\*j, 0+2\*j, 0+2\*j));

HBRUSH hB = CreateSolidBrush(RGB(0+2\*j, 0+2\*j, 0+2\*j));

hdc = BeginPaint(hWnd, &ps);

SelectObject(hdc, hR);

Rectangle(hdc, x-i, rect.bottom / 3, x-i + 300, rect.bottom / 3 + 100);

SelectObject(hdc, hB);

Ellipse(hdc, x-i +50, rect.bottom / 3 + 99, x-i+80, rect.bottom / 3 + 129);

Ellipse(hdc, x-i + 220, rect.bottom / 3 + 99, x-i + 250, rect.bottom / 3 + 129);

EndPaint(hWnd, &ps);

DeleteObject(hB);

DeleteObject(hR);

break;

}

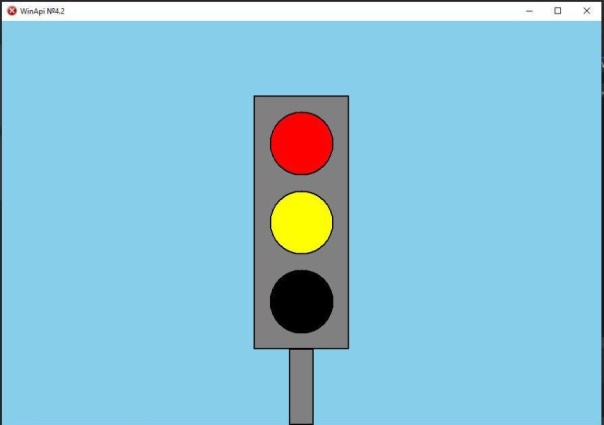
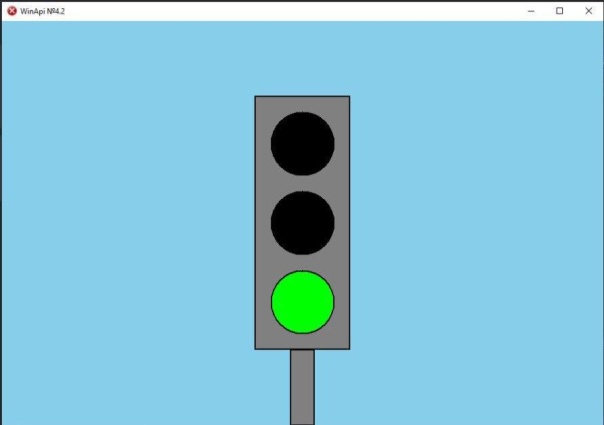
## 3.4.2 Задание 2

**Постановка задачи.**

Светофор. Обеспечить переключение цветов.

**Решение задачи.**

**Алгоритм.** Мысленно разбиваем время на 20 секунд. С первой по седьмую горит зелёный цвет, на восьмой и десятой секунде зелёный цвет мигает, т.е. ничего не горит, а на девятой и одиннадцатой зелёный горит. С 12 по 13 секунду горит жёлтый, а потом с 14 по 18 горит только красный. На 19 секунде к нему присоединяется желтый. С 20 секунды время обнуляется, процесс повторяется снова. Осталось нарисовать соответветствующие кружочки.



Скриншот Лабораторной работы 3.4.2

**Фрагмент кода**

int i = 0; //для светофора

int k; //для рисования светофора

bool yellow = false;

bool green = false;

bool red = false;

LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

HDC hdc;

PAINTSTRUCT ps;

RECT rect;

GetClientRect(hWnd, &rect);

switch (uMsg)

{

case WM\_TIMER:

{

i++;

if (i > 20)

i = 1;

red = false;

yellow = false;

green = false;

if (i <= 7 || i == 9 || i ==11)

green = true;

if (i == 12 || i == 13 || i == 19 || i == 20)

yellow = true;

if (i >= 14 && i <= 20)

red = true;

RedrawWindow(hWnd, NULL, NULL, RDW\_INVALIDATE | RDW\_ERASE);

break;

}

case WM\_PAINT:

{

hdc = BeginPaint(hWnd, &ps);

SetBkMode(hdc, TRANSPARENT);

HBRUSH hBlack = CreateSolidBrush(RGB(0, 0, 0));

HBRUSH hYellow = CreateSolidBrush(RGB(255, 255, 0));

HBRUSH hRed = CreateSolidBrush(RGB(255, 0, 0));

HBRUSH hGreen = CreateSolidBrush(RGB(0, 255, 0));

HPEN pBlack = CreatePen(PS\_SOLID, 2, RGB(0, 0, 0));

HBRUSH hGray = CreateSolidBrush(RGB(128, 128, 128));

SelectObject(hdc, pBlack);

SelectObject(hdc, hGray);

if (rect.right > rect.bottom)

k = rect.bottom;

else

k = rect.right;

k /= 25;

//основа светофора

Rectangle(hdc, rect.right / 2 - 3 \* k, rect.bottom / 2 - 8 \* k, rect.right / 2 + 3 \* k, rect.bottom / 2 + 8 \* k);

Rectangle(hdc, rect.right / 2 - 3 \* k + rect.right / 2 / 5, rect.bottom / 2 + 8 \* k , rect.right / 2 + 3 \* k - rect.right / 2 / 5, rect.bottom);

SelectObject(hdc, hBlack);

//черные кружочки внутри светофора

Ellipse(hdc, rect.right / 2 - 2 \* k, rect.bottom / 2 - 7 \* k , rect.right / 2 + 2 \* k, rect.bottom / 2 - 3 \* k);

Ellipse(hdc, rect.right / 2 - 2 \* k, rect.bottom / 2 - 2 \* k , rect.right / 2 + 2 \* k, rect.bottom / 2 + 2 \* k);

Ellipse(hdc, rect.right / 2 - 2 \* k, rect.bottom / 2 + 3 \* k , rect.right / 2 + 2 \* k, rect.bottom / 2 + 7 \* k);

if (yellow == true)

{

//жёлтый кружочек

SelectObject(hdc, hYellow);

Ellipse(hdc, rect.right / 2 - 2 \* k, rect.bottom / 2 - 2 \* k, rect.right / 2 + 2 \* k, rect.bottom / 2 + 2 \* k);

}

if (red == true)

{

//красный кружочек

SelectObject(hdc, hRed);

Ellipse(hdc, rect.right / 2 - 2 \* k, rect.bottom / 2 - 7 \* k, rect.right / 2 + 2 \* k, rect.bottom / 2 - 3 \* k);

}

if(green == true)

{

//зелёный кружочек

SelectObject(hdc, hGreen);

Ellipse(hdc, rect.right / 2 - 2 \* k, rect.bottom / 2 + 3 \* k, rect.right / 2 + 2 \* k, rect.bottom / 2 + 7 \* k);

}

DeleteObject(hBlack);

DeleteObject(pBlack);

DeleteObject(hGray);

DeleteObject(hRed);

DeleteObject(hGreen);

DeleteObject(hYellow);

EndPaint(hWnd, &ps);

break;

}

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return (DefWindowProc(hWnd, uMsg, wParam, lParam));

}

return 0;

}

## 3.4.3 Задание 3

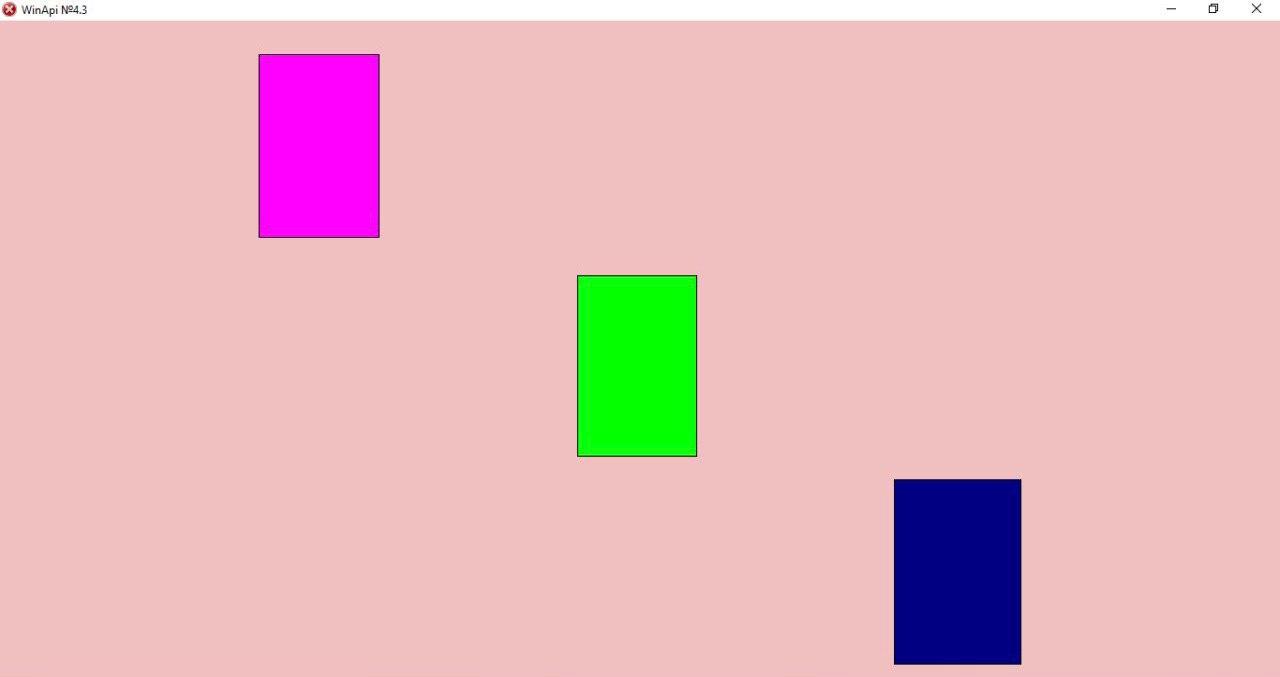
**Постановка задачи.**

В клиентской области расположены три квадрата разных цветов. Щелчок левой кнопкой мыши по кругу приводит к его «миганию» (увеличение/уменьшение диаметра). Щелчок правой кнопкой мыши по кругу останавливает мигание. Сообщения:

* WM\_CREATE – создать таймер;
* WM\_PAINT – нарисовать три круга в соответствующем состоянии;
* WM\_TIMER – изменить состояния кругов и сообщить о необходимости обновить клиентскую область;
* WM\_LBUTTONDOWN – добавить соответствующий круг к «мигающим» (функция GetPixel() позволит определить цвет пикселя, на который попал курсор мыши, и, следовательно, узнать, в область какого круга попал курсор мыши);
* WM\_RBUTTONDOWN – удалить соответствующий круг из «мигающих».

**Решение задачи.**

**Алгоритм.** Изначально рисуем 3 прямоугольника. При нажатии левой кнопкой мыши определяем координаты. Проверяем на то, попадают ли они в один из квадратов. Если да, то активируем мигание этой фигуры, если нет, то ничего не происходит. Если произошло нажатие правой кнопкой мыши, то также проверяем координаты на попадание в фигуры. Если попали, то завершаем мигание. Мигание реализовано таким образом: сперва фигура 5 раз увеличивается, а потом 10 раз уменьшается, потом 10 раз увеличивается и т.д.

****

Скриншот Лабораторной работы 3.4.3

**Фрагмент кода**

bool timer = false;

int kv\_1 = 0; //для изменения размеров квадратиков

int kv\_2 = 0; //

int kv\_3 = 0; //

bool s\_1 = false, //определяем, в какую сторону мигают квадратики

s\_2 = false, //

s\_3 = false; //

bool kvadrat\_1 = false, //определяем, какой квадратик мигает

kvadrat\_2 = false, //

kvadrat\_3 = false; //

auto FUCHSIA = RGB(255, 0, 255); //цвет квадратика

auto LIME = RGB(0, 255, 0); //

auto NAVY = RGB(0, 0, 128); //

case WM\_LBUTTONDOWN:

{

hdc = GetDC(hWnd);

int xm = LOWORD(lParam); //координаты тыка левой мышки

int ym = HIWORD(lParam);

//определяем, куда тыкнули

if (FUCHSIA == GetPixel(hdc, xm, ym)){

kvadrat\_1 = true;

}

if (NAVY == GetPixel(hdc, xm, ym)){

kvadrat\_2 = true;

}

if (LIME == GetPixel(hdc, xm, ym)) {

kvadrat\_3 = true;

}

if (timer == false) {

SetTimer(hWnd, timer1, 30, NULL);

timer = true;

}

ReleaseDC(hWnd, hdc);

RedrawWindow(hWnd, NULL, NULL, RDW\_INVALIDATE | RDW\_ERASE);

break;

}

case WM\_RBUTTONDOWN:

{

hdc = GetDC(hWnd);

int xm = LOWORD(lParam); //координаты тыка правой мышки

int ym = HIWORD(lParam);

//определяем, куда тыкнули

if (FUCHSIA == GetPixel(hdc, xm, ym)) {

kvadrat\_1 = false;

}

if (NAVY == GetPixel(hdc, xm, ym)){

kvadrat\_2 = false;

}

if (LIME == GetPixel(hdc, xm, ym)) {

kvadrat\_3 = false;

}

if (FUCHSIA == false && LIME == false && NAVY == false){

timer = false;

KillTimer(hWnd, timer1);

}

ReleaseDC(hWnd, hdc);

RedrawWindow(hWnd, NULL, NULL, RDW\_INVALIDATE | RDW\_ERASE);

break;

}

case WM\_TIMER:

{

if (kvadrat\_1 == true){

if (kv\_1 > 5)

s\_1 = true;

else if (kv\_1 < -5)

s\_1 = false;

if (s\_1)

kv\_1--;

else

kv\_1++;

}

if (kvadrat\_2 == true)

{

if (kv\_2 > 5)

s\_2 = true;

else if (kv\_2 < -5)

s\_2 = false;

if (s\_2)

kv\_2--;

else

kv\_2++;

}

if (kvadrat\_3 == true)

{

if (kv\_3 > 5)

s\_3 = true;

else if (kv\_3 < -5)

s\_3 = false;

if (s\_3)

kv\_3--;

else

kv\_3++;

}

RedrawWindow(hWnd, NULL, NULL, RDW\_INVALIDATE | RDW\_ERASE);

break;

}

case WM\_PAINT:

{

hdc = BeginPaint(hWnd, &ps);

SetBkMode(hdc, TRANSPARENT);

HPEN Black = CreatePen(PS\_SOLID, 2, RGB(0, 0, 0));

HBRUSH fuchsia = CreateSolidBrush(RGB(255, 0, 255));

HBRUSH navy = CreateSolidBrush(RGB(0, 0, 128));

HBRUSH lime = CreateSolidBrush(RGB(0, 255, 0));

double x;

if (rect.bottom >= rect.right)

x = rect.right / 3.5;

else

x = rect.bottom / 3.5;

SelectObject(hdc, fuchsia);

Rectangle(hdc, rect.right / 5 - kv\_1, rect.bottom / 3 - x - kv\_1, rect.right / 10 \* 3 + kv\_1, rect.bottom / 3 + kv\_1);

SelectObject(hdc, navy);

Rectangle(hdc, rect.right / 10 \* 7 - kv\_2, rect.bottom - x - kv\_2 - 15, rect.right / 5 \* 4 + kv\_2, rect.bottom + kv\_2 - 15);

SelectObject(hdc, lime);

Rectangle(hdc, rect.right / 20 \* 9- kv\_3, rect.bottom / 3 \* 2 - x - kv\_3, rect.right / 20 \* 11 + kv\_3, rect.bottom / 3 \* 2 + kv\_3);

DeleteObject(fuchsia);

DeleteObject(navy);

DeleteObject(lime);

DeleteObject(Black);

EndPaint(hWnd, &ps);

break;

}

### 

## 3.4.4 Задание 4

**Постановка задачи**.

Нарисовать график функции y=f(x) на отрезке [a,b]. График масштабируется в зависимости от размеров клиентской области. Щелчок по графику приводит к отображению значений (x,y). Проверить для трёх различных функций.

ЗАДАНИЕ НЕ ВЫПОЛНЕНО

## 3.4.5 Задание 5

**Постановка задачи.**

Шарик катится по синусоиде туда и обратно. Дополнительно показать вращение шарика, добавив положение радиуса шарика.

ЗАДАНИЕ НЕ ВЫПОЛНЕНО

## 3.5 Лабораторная работа 5

**Постановка задачи.**

1. Разобрать пример из книги Щупака (начиная со страницы 350), в котором нужно разработать приложение «Телефонная книга» и реализовать его.
2. Для выбора файла использовать стандартные диалоги выбора файла.
3. В отдельном диалоговом окне нарисовать диаграмму, отображающую количество телефонов в записной книжке, принадлежащих различным мобильным операторам. Выбор цвета столбиков выполнить через стандартный диалог выбора цвета. Выбор шрифта выполнить через стандартный диалог выбора шрифта.
4. Интерфейс проектировать с помощью редактора ресурсов.

ЗАДАНИЕ НЕ ВЫПОЛНЕНО

# 4 С#

## 4.0 Лабораторная работа 0

**Постановка задачи:**

Разработайте приложение Windows Forms при помощи MS Visual Studio C#. При реализации интерфейса следуйте рекомендациям стандарта CUI (Common User Interface).

Программа должна иметь главное меню, строку состояния и клиентскую область (область

рисования). Главное меню приложения обязательно должно содержать следующие команды (в формате Подменю/Команда):

– File/Open - выбор файла

– File/Quit - завершение приложения после подтверждения пользователя

– Help/About - вывод диалогового окна которое печатает условие задачи и информацию о

разработчике (ФИО студента, группа, курс, факультет, ВУЗ)

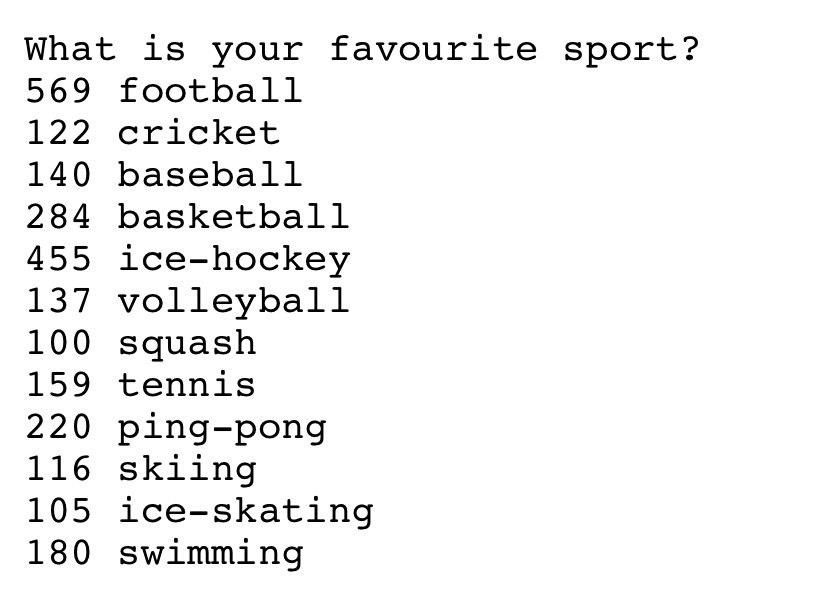
– (команды) - команды необходимые для решения вашей задачи, например, View/Fonts для выбора шрифта, View/Colors для настройки цвета и т.п.

*Постройте диаграмму опроса в сети отображающую процентное соотношение*

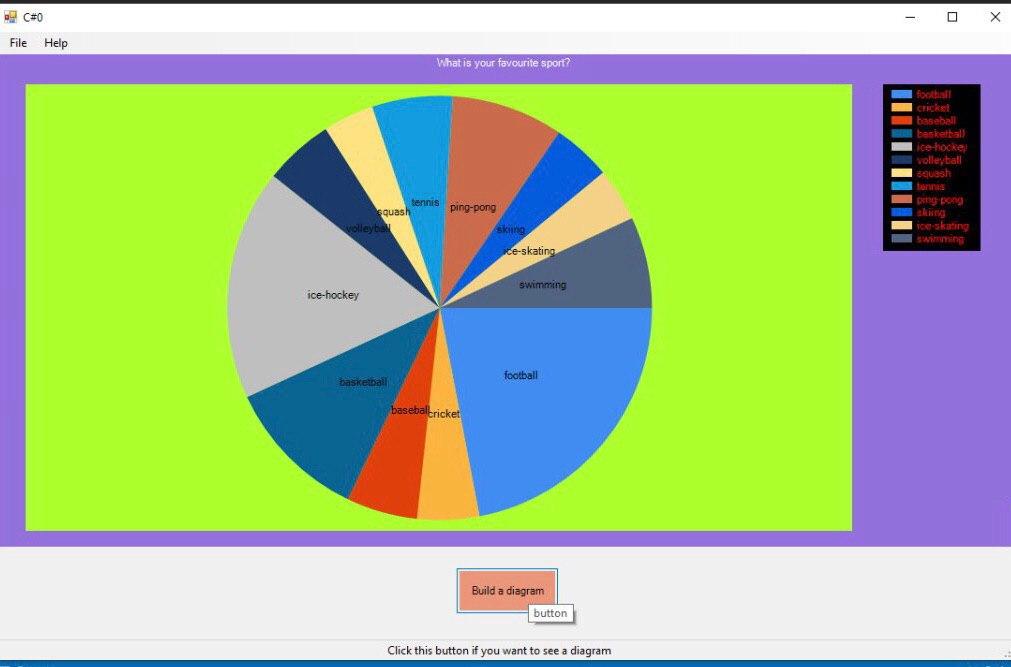
*проголосовавших. Входные данные хранятся в текстовом файле. Первая строка файла содержит наименование опроса. Каждая последующая строка файла содержит количество проголосовавших, затем, через пробел и до конца строки текст, соответствующий*

*опции опроса. Для выбора файла входных даных использовать стандартный диалог.*

**Текстовый файл “sports.txt”**

****

**Решение задачи.**



Скриншот 4.0. «Диаграмма»

**Фрагмент кода.**

**//**обработка закрытия окна

private void AppWindow\_FormClosing(Object sender, FormClosingEventArgs e)

{

if (MessageBox.Show("Do you want to quit?", Text, MessageBoxButtons.OKCancel, MessageBoxIcon.Question) != DialogResult.OK)

e.Cancel = true;

}

//меню

private void BuildMenu()

{

mnuFile.Text = "&File";

mnuFile.MouseEnter += (o, s) => toolStripStatusLabel.Text = "You should open file to build a diagram";

mnuFile.MouseLeave += (o, s) => toolStripStatusLabel.Text = "";

mnuMain.Items.Add(mnuFile);

mnuHelp.Text = "&Help";

mnuHelp.MouseEnter += (o, s) => toolStripStatusLabel.Text = "Read information how to use my program";

mnuHelp.MouseLeave += (o, s) => toolStripStatusLabel.Text = "";

mnuMain.Items.Add(mnuHelp);

mnuFileOpenFile.Text = "&Open";

mnuFile.DropDownItems.Add(mnuFileOpenFile);

mnuFileOpenFile.Click += (o, s) => OnFileOpen();

mnuFileOpenFile.MouseEnter += (o, s) => toolStripStatusLabel.Text = "Open file";

mnuFileOpenFile.MouseLeave += (o, s) => toolStripStatusLabel.Text = "";

mnuFile.DropDownItems.Add(new ToolStripSeparator());

mnuFileExit.Text = "&Exit";

mnuFile.DropDownItems.Add(mnuFileExit);

mnuFileExit.Click += (o, s) => Application.Exit();

mnuFileExit.MouseEnter += (o, s) => toolStripStatusLabel.Text = "Goodbye";

mnuFileExit.MouseLeave += (o, s) => toolStripStatusLabel.Text = "";

mnuHelpAbout.Text = "&About my program";

mnuHelp.DropDownItems.Add(mnuHelpAbout);

mnuHelpAbout.Click += (o, s) => OnHelpAbout();

mnuHelpAbout.MouseEnter += (o, s) => toolStripStatusLabel.Text = "Information about program";

mnuHelpAbout.MouseLeave += (o, s) => toolStripStatusLabel.Text = "";

Controls.Add(mnuMain);

MainMenuStrip = mnuMain;

}

//кнопка

private void BuildButton()

{

btnClickMe.Text = "Build a diagram";

btnClickMe.SetBounds(20, 20, 110, 50);

btnClickMe.Click += (o, s) => onButtonClicked();

btnClickMe.MouseEnter += (o, s) => toolStripStatusLabel.Text = "Click this button if you want to see a diagram";

btnClickMe.MouseLeave += (o, s) => toolStripStatusLabel.Text = "";

toolTip.SetToolTip(btnClickMe as System.Windows.Forms.Control, "button");

Controls.Add(btnClickMe);

}

//обработка файла

private void OnFileOpen()

{

using (OpenFileDialog openFileDialog = new OpenFileDialog())

{

openFileDialog.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.Personal);

openFileDialog.Filter = "Text files (\*.txt) |\*.txt|All files (\*.\*) |\*.\*";

openFileDialog.FilterIndex = 0;

openFileDialog.DefaultExt = "txt";

openFileDialog.FileName = "\*.txt\*";

openFileDialog.RestoreDirectory = true;

openFileDialog.Multiselect = false;

openFileDialog.ShowDialog();

String sString;

StreamReader sreader = new StreamReader(openFileDialog.FileName, System.Text.Encoding.GetEncoding(1251));

ChartName = sreader.ReadLine();

while ((sString = sreader.ReadLine()) != null)

{

int x = sString.IndexOf(' ');

DataElement Name = new DataElement(sString.Substring(x + 1), Convert.ToInt32(sString.Substring(0, x)));

DataList.Add(Name);

}

sreader.Close();

}

}

//построение диаграммы

private void BuildChart(String title)

{

System.Windows.Forms.DataVisualization.Charting.ChartArea chartArea1 = new System.Windows.Forms.DataVisualization.Charting.ChartArea();

System.Windows.Forms.DataVisualization.Charting.Legend legend1 = new System.Windows.Forms.DataVisualization.Charting.Legend();

System.Windows.Forms.DataVisualization.Charting.Series series1 = new System.Windows.Forms.DataVisualization.Charting.Series();

System.Windows.Forms.DataVisualization.Charting.Title title1 = new System.Windows.Forms.DataVisualization.Charting.Title();

this.MainChart = new System.Windows.Forms.DataVisualization.Charting.Chart();

((System.ComponentModel.ISupportInitialize)(this.MainChart)).BeginInit();

this.MainChart.BackgroundImageLayout = System.Windows.Forms.ImageLayout.None;

this.MainChart.ForeColor = Color.Yellow;

this.MainChart.BackColor = Color.MediumPurple;

chartArea1.BackColor = Color.GreenYellow;

series1.Color = Color.Black;

legend1.ForeColor = Color.Red;

legend1.BackColor = Color.Black;

chartArea1.Name = "ChartArea1";

this.MainChart.ChartAreas.Add(chartArea1);

legend1.Name = "Legend1";

this.MainChart.Legends.Add(legend1);

this.MainChart.Location = new System.Drawing.Point(0, 10);

this.MainChart.Name = "MainChart";

series1.ChartArea = "ChartArea1";

series1.ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Pie;

series1.Legend = "Legend1";

series1.Name = "Opros";

series1.YValuesPerPoint = 2;

this.MainChart.Series.Add(series1);

this.MainChart.Size = new System.Drawing.Size(ClientSize.Width, ClientSize.Height \* 9 / 11);

this.MainChart.TabIndex = 0;

this.MainChart.Text = "Chart";

title1.Name = "title1";

title1.Text = title;

title1.ForeColor = Color.White;

title1.Alignment = ContentAlignment.TopCenter;

this.MainChart.Titles.Add(title1);

this.Controls.Add(this.MainChart);

this.Name = "AppWindow";

((System.ComponentModel.ISupportInitialize)(this.MainChart)).EndInit();

this.ResumeLayout(false);

}

private System.Windows.Forms.DataVisualization.Charting.Chart MainChart;

//окно ABOUT

private void OnHelpAbout()

{

MessageBox.Show("Volkov Evgenij, group 10, CS#0\n" +

"First, choose file using buttons FILE and OPEN.\n" +

"Find your file on a computer and open it.\n" +

"Then click on button BUILD A DIAGRAM and see it.\n" +

"GOOD LUCK!!", "About program", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

## 4.1 Лабораторная работа 1

**Постановка задачи.**

Перекодировка текстового файла, пользователь выбирает несовпадающие кодировки входного и выходного файлов.

Допустимые кодировки: koi8-r, windows-1251, utf-8, ibm-866.

ЗАДАНИЕ НЕ ВЫПОЛНЕНО

## 4.2 Лабораторная работа 2

**Постановка задачи.**

Копирование с шифрованием бинарного входного файла в выбранный каталог (отличный от исходного) блоками по 512 байт фиксированным ключом длиной 128 байт. Ключ создать случайным образом и выводить в файл по команде пользователя.

ЗАДАНИЕ НЕ ВЫПОЛНЕНО

## 4.3 Лабораторная работа 3

**Постановка задачи.**

Сортировка исходного текстового файла, реализовать алгоритм чётно-нечётной сортировки слиянием Бэтчера.

ЗАДАНИЕ НЕ ВЫПОЛНЕНО

## 4.4 Лабораторная работа 4

**Постановка задачи.**

Разделение изображения на части. Программа предлагает выбрать файл изображения, отображает его в клиентской области окна и спрашивает пользователя на сколько частей нужно разделить изображение по горизонтали (H) и на сколько частей по вертикали (V). Полученные H\*V изображений сохраняются в файлы типа GIF.

ЗАДАНИЕ НЕ ВЫПОЛНЕНО

## 4.5 Лабораторная работа 5

**Постановка задачи.**

Создание HTML-карты. Программа предлагает выбрать файл изображения, отображает его в клиентской области окна. Пользователь может определять непересекающиеся прямоугольные области на изображении, с каждой областью связывает название и URL- адрес. После нанесения разметки программа сохраняет разметку в текстовый файл в виде HTML-карты (тег <map>, оставшиеся неразмеченными части изображения помечать атрибутом nohref).

ЗАДАНИЕ НЕ ВЫПОЛНЕНО

# Заключение

За время учебной практики я научился разрабатывать сайты с базовым функционалом, используя современные средства. Так же я научился разрабатывать базовые Windows-приложения на языке C++, используя средства WinApi. Выучил основы работы с языком C#, а также WindowsForms. Улучшил свои умения в нахождении необходимой информации, а также стал более самостоятельным при возникновении технических трудностей.

Думаю, что опыт, полученный мной на данной практике, несомненно, будет полезен в дальнейшем при построении своей профессиональной деятельности в сфере программирования, а особенно в сфере компьютерной безопасности.

# Список литературы

1. В. В. Мержевич. HTML и CSS на примерах. / В. В. Мержевич. – СПб: BHV-Санкт-Петербург, 2005.
2. Т. А. Павловская. С#. Программирование на языке высокого уровня: Учебник для вузов. /Т. А. Павловская. - СПб: Питер, 2012.
3. Дженнифер Грин, Эндрю Стиллмен. Изучаем C#. / Дженнифер Грин, Эндрю Стиллмен. – СПб: Питер, 2014.
4. Лафоре Р., «Объектно-ориентированное программирование в C++, 4-е издание». Санкт-Петербург, «Питер», 2004
5. Е.Л. Романов. «Практикум по программированию на С++». Санкт- Петербург, «БХВ-Петербург»,2004.
6. Страуструп, Б. Программирование: принципы и практика с использованием С++, 2-е издание: Пер. с англ. – М.: ООО "И.Д. Вильяме", 2016 – 1328 с.
7. Шилдг Г. Полный справочник по С#: Пер. с англ. — М.: Издательский дом "Вильяме", 2004 – 752 с.
8. https://habr.com – Электронный ресурс (дата доступа: 06.05.20, 20:11).
9. http://.cyberforum.ru – Электронный ресурс (дата доступа: 10.05.20, 14:00).
10. http://stackoverflow.com – Электронный ресурс (дата доступа: 12.05.20, 17:40).
11. https://www.wikipedia.org – Электронный ресурс (дата доступа: 12.05.20, 17:40).

# Приложение А

# 5 Вложение

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра информационных систем управления

Волков Евгений Сергеевич

**РЕФЕРАТ**

**на тему «Отладка программ»**

студента 1 курса 10а группы

**Преподаватель**

*Конах Валентина Владимировна*

старший преподаватель

кафедры ИСУ

Минск-2020

# **Оглавление**

[Введение 71](#_Toc39776954)

[Ошибки 72](#_Toc39776955)

[Отладчики 74](#_Toc39776956)

[Точка останова 76](#_Toc39776957)

[Инструменты отладки 76](#_Toc39776958)

[Инструменты, снижающие потребность в отладке 77](#_Toc39776959)

[Проблемы отладки 77](#_Toc39776960)

[Выводы 79](#_Toc39776961)

[Список использованной литературы 80](#_Toc39776962)

# 

# Введение

В современном мире программирование вышло на новый уровень. Его возможности позволяют решать любые задачи в любой сфере: медицина, спорт, бизнес, безопасность, транспорт, страхование, промышленность и т.д. И в течение последних 50 лет программисты успешно справляются с решением таких задач. Однако хватит ли просто навыков программировать, чтобы грамотно и быстро решить поставленную задачу? Нет, не хватит. Каждый уважающий себя программист умеет анализировать свой код, искать в нём ошибки, исправлять их и оптимизировать программу. Для этого и нужна отладка программ.

Как известно, создание программы происходит в несколько этапов (анализ, проектирование, программирование, тестирование, отладка и т.д.). Отладка и тестирование являются наиболее важными и трудоёмкими этапами. Именно на них устраняются все логические и синтаксические ошибки в создаваемой программе.

*Отладка (debugging)* — этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки.

*Тестирование* – это процесс выполнения программы на некотором наборе данных, для которого заранее известен результат применения или известны правила поведения этих программ. Таким образом, отладку можно представить в виде многократного повторения трех процессов: тестирования, в результате которого может быть констатировано наличие ошибки, поиска места ошибки в программе и редактирования программы и документации с целью устранения обнаруженной ошибки. Таким образом, все вышесказанное можно представить формулой:

Отладка = Тестирование + Поиск ошибок + Редактирование.

# 

# Ошибки

Знание на высоком уровне языка программирования, не является гарантией, что ошибок в коде не будет. По этой причине каждый программист должен уметь отлаживать свою программу. Рассмотрим существующие типы ошибок:

*Синтаксические ошибки* — ошибки, которые компилятор может отловить при сборке проекта. Хотя уже сейчас такого типа ошибки компилятор отлавливает ещё до сборки проекта.

*Логические ошибки* являются следствием реализации неправильного алгоритма и проявляются при выполнении программы. Наличие логических ошибок обычно не приводит к выдаче пользователю каких-либо сообщений или прекращению работы всего приложения, однако программа будет работать некорректно и выдавать неправильные результаты.

*Ошибки компиляции* – это предупреждения компилятора, которые после запуска программы стали ошибками, т.к. не были исправлены.

*Динамические ошибки или ошибки времени выполнения (runtime errors)*– это такие ошибки, которые проявляются во время выполнения программы. Причиной их могут быть как ошибки в программе, так и неправильные действия пользователя, неверные данные и т.д.

*Арифметические ошибки* возникают, когда компьютер не может справиться с проблемами, такими как "Деление на ноль", или ведущими к бесконечному результату.

*Ошибки ресурса* возникают, когда значение переменной переполняет максимально допустимое значение. Переполнение буфера, использование неинициализированной переменной, нарушение прав доступа и переполнение стека - примеры некоторых распространенных ошибок.

*Ошибки взаимодействия* могут возникнуть в связи с несоответствием программного обеспечения с аппаратным интерфейсом или интерфейсом прикладного программирования.

В качестве примера рассмотрим следующую программу. После запуска мы получим сообщение о такой ошибке:

***int main(){***

***…***

***return 0 /// забыли поставить точку с запятой***

***}***

Ошибка 1 error C2143: синтаксическая ошибка: отсутствие «;» перед «}» d:\c++\project1\main.cpp 15 1 main.

В сообщении указываются такие данные:

* код ошибки — C2143
* тип ошибки — синтаксическая ошибка
* суть ошибки — отсутствие ";" перед "}"
* адрес исполняемого файла — d:\c++\project1\main.cpp
* номер строки — 15
* столбец — 1
* имя файла — main

На данный период времени синтаксические ошибки не являются серьезной проблемой, так как еще до запуска программы, компилятор находит их и сообщает об этом программисту. Другое дело логические ошибки, которые компилятор не может поймать, а найти их бывает сложно с первого раза.

Пример логической ошибки:

***int counter = 1;***

***while (counter = 1) // ОШИБКА***

***{***

***counter--;***

***}***

Ошибка заключается в том, что вместо знака сравнения (==) мы используем знак присваивания (=). По этой причине мы оказываемся в бесконечном цикле. Для того, чтобы увидеть эту ошибку, можно слегка изменить код таким образом, добавив вывод на экран значение нашей переменной counter:

***int counter = 1;***

***while (counter = 1) // ОШИБКА***

***{***

***counter--;***

***cout << counter << endl; // вывод на экран значения переменной counter***

***}***

Благодаря дополнению в этом коде мы видим, что у нас бесконечно выводится ноль, что не дает программе завершиться.

# Отладчики

В современных программных системах отладка осуществляется часто с использованием специальных программных средств, называемых *отладчиками*. Эти средства позволяют **исследовать внутреннее поведение программы.**

*Программа-отладчик* обычно обеспечивает следующие возможности:

* пошаговое исполнение программы с остановкой после каждой команды (оператора);
* просмотр текущего значения любой переменной или нахождение значения любого выражения, в том числе, с использованием стандартных функций; при необходимости можно установить новое значение переменной;
* установку в программе "контрольных точек", т.е. точек, в которых программа временно прекращает свое выполнение, так что можно оценить промежуточные результаты, и др.

Есть различные подходы и средства отладки; в качестве основного средства используются встроенные программные отладчики, которые включают в себя пользовательский интерфейс для пошагового выполнения программы: оператор за оператором, функция за функцией, с остановками на некоторых строках исходного кода или при достижении определённого условия.

Кроме использования встроенных программных отладчиков существуют и другие методы отладок программы: использование внутрисхемного эмулятора, отладка при помощи внешних программных отладчиков и отлаживаемым устройством с записанным в память программ двоичным кодом программы.

*Внутрисхемный эмулятор* с отображением переменных языка программирования на дисплее компьютера оказывает значительную помощь при отладке программ непосредственно на разрабатываемой аппаратуре. Этот метод отладки предоставляет наиболее удобную среду, когда можно непосредственно в отлаживаемом устройстве останавливать программу, контролировать выполнение программы непосредственно по исходному тексту программы, состояние внешних портов и внутренних переменных, как входящих в состав микросхемы, так и объявленных при написании исходного текста программы. До недавнего времени внутрисхемный эмулятор являлся отдельным устройством, подключаемым к разрабатываемой плате.

*Встроенный программный отладчик*, входящий в состав интегрированной среды программирования значительно упрощает процесс отладки программы. Но в тоже время она значительно замедляется.

*Внешний программный отладчик*. В некоторых случаях используется не интегрированная среда программирования, а отдельный транслятор с выбранного языка программирования. Так как объектные форматы различных трансляторов несколько отличаются друг от друга, то в качестве входного файла используется загрузочный модуль в двоичном формате.

Отладка программ заключается в проверке правильности работы программы и аппаратуры. Программа, не содержащая синтаксических ошибок, тем не менее, может содержать логические ошибки, не позволяющие программе выполнять заложенные в ней функции. Логические ошибки могут быть связаны с алгоритмом программы или с неправильным пониманием работы аппаратуры, подключённой к портам микроконтроллера.

Встроенный в состав интегрированной среды программирования отладчик позволяет отладить те участки кода программы, которые не зависят от работы аппаратуры, не входящей в состав микросхемы микроконтроллера. Обычно это относится к вычислению математических выражений или преобразованию форматов представления данных.

Для отладки программ обычно применяют три способа:

1. Пошаговая отладка программ с заходом в подпрограммы.
2. Пошаговая отладка программ с выполнением подпрограммы как одного оператора.
3. Выполнение программы до точки останова.

# Точка останова

В процессе отладки программы используются так называемые точки останова. *Точка останова (англ. breakpoint)* — это преднамеренное прерывание выполнения программы, при котором выполняется вызов отладчика (одновременно с этим, программа сама может использовать точки останова для своих нужд). После перехода к отладчику, программист может исследовать состояние программы (логи, состояние памяти, регистров процессора, стека и т. п.), с тем чтобы определить, правильно ли ведёт себя программа. После остановки в отладчике, программа может быть завершена либо продолжена с того же места где произошла остановка.

На практике, точка остановки определяется как одно или несколько условий, при которых происходит прерывание программы. Наиболее часто используется условие остановки при переходе управления к указанной инструкции программы (instruction breakpoint). Другое условие остановки — операция чтения, записи или изменения указанной ячейки или диапазона ячеек памяти (data breakpoint или watchpoint).

Многие процессоры имеют аппаратную поддержку точек останова (обычно, только для instruction breakpoint и watchpoint). При отсутствии такой аппаратной поддержки отладчики используют программные точки останова.

# Инструменты отладки

Отладчик представляет из себя программный инструмент, позволяющий программисту наблюдать за выполнением исследуемой программы, останавливать и перезапускать её, прогонять в замедленном темпе, изменять значения в памяти и даже, в некоторых случаях, возвращать назад по времени.

Также полезными инструментами в руках программиста могут оказаться:

* *Профилировщики*. Они позволят определить, сколько времени выполняется тот или иной участок кода. Анализ покрытия позволяет выявить неисполняемые участки кода.
* *API логгеры* позволяют отследить взаимодействие программы и Windows API при помощи записи сообщений Windows в лог.
* *Дизассемблеры* позволяют посмотреть ассемблерный код исполняемого файла.
* *Снифферы* помогут отследить сетевой трафик, генерируемый программой
* *Снифферы аппаратных интерфейсов* позволяют увидеть данные, которыми обмениваются система и устройство.

Использование языков программирования высокого уровня обычно упрощает отладку, если такие языки содержат, например, средства обработки исключений, сильно облегчающие поиск источника проблемы. В низкоуровневых языках ошибки могут приводить к незаметным проблемам — например, повреждениям памяти и утечкам памяти. Тогда бывает довольно трудно определить, что стало первоначальной причиной ошибки. В этих случаях могут потребоваться сложные приёмы и средства отладки.

# Инструменты, снижающие потребность в отладке

Другое направление — сделать, чтобы отладка нужна была как можно реже. Для этого применяются:

* *Контрактное программирование* — чтобы программист подтверждал другим путём, что ему на выходе нужно именно такое поведение программы. В языках, в которых контрактного программирования нет, используется самопроверка программы в ключевых точках.
* *Модульное тестирование* — проверка поведения программы по частям.
* *Статический анализ кода* — проверка кода на стандартные ошибки «по недосмотру».
* *Высокая культура программирования*, в частности, паттерны проектирования, соглашения об именовании и прозрачное поведение отдельных блоков кода — чтобы объявить себе и другим, каким образом должна вести себя та или иная функция.
* Широкое *использование* проверенных *внешних библиотек*.

# Проблемы отладки

1. Оптимизация кода.

Первое, что приходит на ум при отладке, это вопрос: «Мой код слишком сложный?» Иногда мы находим решение проблемы и понимаем, что оно слишком сложное для реализации. Настолько сложное, на самом деле, что может быть проще решить эту проблему по-другому. Нужно спросить себя: «Есть ли более простое решение?». Часто, когда мы напишем плохой код, у нас появляется представление о том, как должен выглядеть хороший код.

Хитрость заключается в умении определить, пытаетесь ли вы решить исходную задачу или найти конкретное решение. Если это решение, то вполне возможно, что ваши проблемы не связаны с исходной задачей вообще — может быть, вы слишком много думаете о задаче или неправильно к ней подходите.

Люди часто не хотят избавляться от плохого кода, который они написали, и не переписывают его. Одна из причин — это то, что написанный код кажется завершенной работой, и избавляясь от него, вы как бы движетесь в обратном направлении. Но при отладке, переписывание кода может показаться более привлекательным, потому что вы экономите время при отладке, потратив немного больше времени на кодирование. Хитрость заключается в том, чтобы избежать удаления кода, чтобы не начинать всю программу снова (если только она не вся насквозь ужасна). Переписывать надо только те части, которые действительно в этом нуждаются.

Второй проект, вероятно, будет понятнее и с меньшим количеством ошибок, чем первый, и, возможно, не придется возвращаться и переписывать код, потому что вы можете выяснить, как он должен работать.

1. Сперва найдите маленькие проблемы, чтобы потом найти большие.

Одним из преимуществ выделения кода в функции является то, что вы можете отдельно проверить эти функции. Это означает, что иногда можно избежать отладки больших проблем, вызванных ошибками в простых первоначальных функциях. Нет ничего хуже, чем написать правильный код, дать ему, по-вашему мнению, работающую функцию (или класс), только для того, чтобы узнать, что она не работает должным образом. Такое модульное тестирование требует определенной дисциплины и понимания того, что с вашим кодом может пойти не так.

# 

# Выводы

1. Основные типы ошибок, возникающих при отладке программ: синтаксические, логические и динамические.

2. Отладка программ позволяет выявлять и исправлять ошибки для корректной работы программы.

3. Основным средством при отладке является отладчик, который представляет из себя программный инструмент, позволяющий наблюдать за выполнением исследуемой программы, останавливать и перезапускать её, изменять значения в памяти и т.д.

4. При отладке важно помнить следующее: возникающие затруднения следует четко **разделять и устранять строго поочередно**, **не нужно считать причиной ошибок машину**, так как современные машины и трансляторы обладают чрезвычайно высокой надежностью.

5. Чтобы реже использовать отладку нужно вначале создать алгоритм решения конкретной задачи, предусмотреть исключительные ситуации и только потом приступать непосредственно к написанию кода и тестированию.

# Список использованной литературы

1. Д. Ван Тассел. Стиль, разработка, эффективность, отладка и испытание программ. - М.: Мир, 1985.
2. Тестирование программного обеспечения. К: ДиаСофт, 2000 - 544с.
3. Microsoft Visual C++ и MFC. Программирование для Windows 95 и Windows NT. М.: ДИАЛОГ-МИФИ, 1996. - 288 с. - (Библиотека системного программиста; Т. 24)
4. [https://works.doklad.ru](https://works.doklad.ru/) — Электронный ресурс (дата доступа: 16.11.19, 16:20)
5. https://www.youtube.com — Электронный ресурс (дата доступа: 16.11.19, 16:50)
6. https://wikipedia.org/wiki/Отладка\_программы — Электронный ресурс (дата доступа: 17.11.19, 11:10)
7. [https://cinref.ru](https://cinref.ru/) — Электронный ресурс (дата доступа: 17.11.19, 12:00)
8. https://www.examen.ru — Электронный ресурс (дата доступа: 17.11.19, 12:50)
9. https://studfile.net — Электронный ресурс (дата доступа: 17.11.19, 13:20)
10. https://juice-health.ru — Электронный ресурс (дата доступа: 17.11.19, 17:30).