

## ОТЧЕТНАЯ РАБОТА НОМЕР 1.

ЯЗЫК РАЗРАБОТКИ C/C++.

ТЕМА: ПО, АНАЛИЗ ТЕКСТА С ИЗОБРАЖЕНИЙ МЕСТНОСТИ/КАРТЫ С  
ДАЛЬНЕЙШИМ ПОСТРОЕНИЕМ ГРАФИЧЕСКОЙ СОСТАВЛЯЮЩЕЙ.

2024г.

## МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ РАБОТЫ.

### 1. Установка библиотеки tgbot.

Установите сначала GIT на ваш компьютер. (x64 если у вас 64 битная система стоит)

Далее:

в терминале (командной строке) клонируем репозиторий в удобное для вас место. **git clone <https://github.com/reo7sp/tgbot-cpp.git>**

подробнее тут: <https://github.com/reo7sp/tgbot-cpp>

команда для сборки (main.cpp - ваш код):

**g++ main.cpp -o main --std=c++14 -I/usr/local/include -lTgBot -lboost\_system -lssl -lcrypto -lpthread**

запуск собранного кода: **./main**

**папка с ресурсами работы.**

<https://disk.yandex.ru/d/BZVY2CDdRBh7bA>

## 2. Определения и терминология.

**ПО (Программное обеспечение)** - совокупность программных средств, позволяющих решать определённые задачи на компьютере, включая прикладные программы, операционные системы, библиотеки и другие компоненты.

**Анализ текста** - процесс извлечения и интерпретации информации из текстовых данных с целью выявления значимых паттернов, структур и свойств.

**Изображение местности/карты** - графическое представление территории, обычно в виде карты, содержащее географические объекты, границы земельных участков, дороги и другие элементы.

**Графическая составляющая** - визуальная часть представления информации, включающая изображения, диаграммы, графики и другие графические элементы.

**Текстовый анализ изображений** - процесс обработки изображений с целью распознавания и анализа текстовой информации, включая текст, встроенный в изображения.

**Алгоритмы компьютерного зрения** - методы и процедуры, применяемые для автоматического анализа изображений и видео с целью извлечения информации о содержании изображения.

**Графическое API** - интерфейс прикладного программирования, предназначенный для взаимодействия с графическим оборудованием и выполнения операций рендеринга.

**Компьютерная графика** - область компьютерных наук, занимающаяся созданием и обработкой графических изображений с использованием программного обеспечения.

**Алгоритмы обработки изображений** - методы и процедуры для изменения и улучшения качества изображений, включая фильтрацию, сегментацию, распознавание объектов и другие техники.

**Интерфейс визуализации данных** - средства и методы представления данных в графической форме для облегчения их понимания и анализа.

### 3. Методики расчета.

#### 1) Построение гистограммы распределения цвета на картинке

Построение гистограммы распределения цвета на изображении позволяет оценить частоту встречаемости каждого цвета в изображении. Это полезно для анализа цветовых характеристик и выявления доминирующих цветовых тонов.

Использовать код на Python (main.py).

В C++:

```
#include <tgbot/tgbot.h>
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <curl/curl.h>
#include <filesystem>

void processImageAndSendGraphs(TgBot::Bot& bot, TgBot::Message::Ptr message, const std::string& filePath) {
    std::string command = "python3 main.py " + filePath;
    int result = system(command.c_str());
    if (result != 0) {
        bot.getApi().sendMessage(message->chat->id, "Error processing the image.");
        return;
    }

    // Check if the files were created successfully before attempting to send
    if (std::filesystem::exists("histograms.png") && std::filesystem::exists("dependencies.png")) {
        bot.getApi().sendPhoto(message->chat->id, TgBot::InputFile::fromFile("histograms.png", "image/png"));
        bot.getApi().sendPhoto(message->chat->id, TgBot::InputFile::fromFile("dependencies.png", "image/png"));
    } else {
        bot.getApi().sendMessage(message->chat->id, "Error: Processed images not found.");
    }
}
```

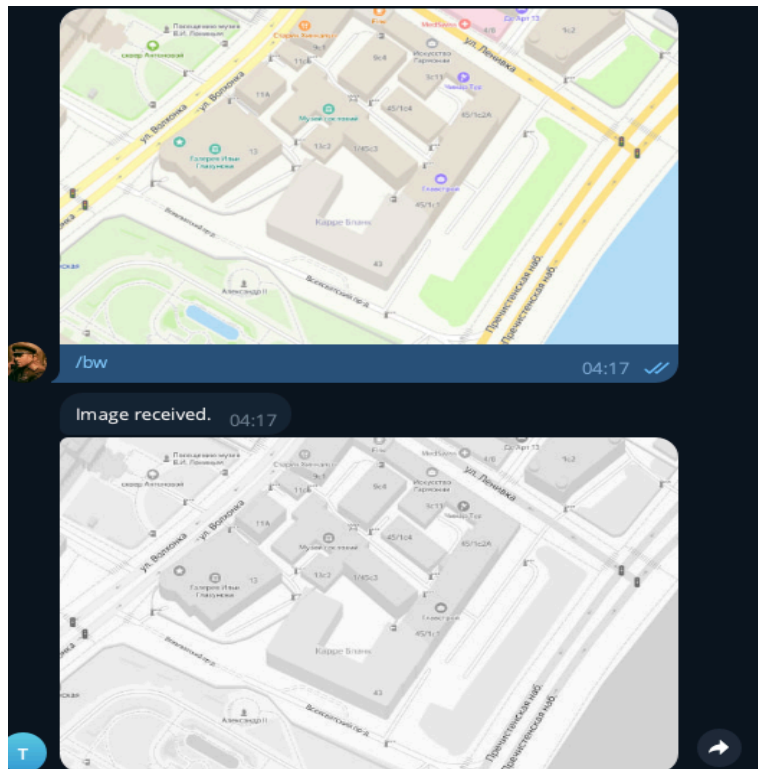
```
size_t WriteCallback(void* contents, size_t size, size_t nmemb, void* userp) {
    std::ofstream* ofs = static_cast<std::ofstream*>(userp);
    size_t totalSize = size * nmemb;
    ofs->write(static_cast<char*>(contents), totalSize);
    return totalSize;
}
```

```
bool downloadFile(const std::string& url, const std::string& filePath);
```

Ожидается что-то такое:



черно-белый фильтр на изображение:



пример в коде на C++:

развилка при загрузке и обработке:

```
if (downloadFile(fileUrl, filePath)) {
    bot.getApi().sendMessage(message->chat->id, "Image received.");
    if (!message->caption.empty() && message->caption.find("/graph") != std::string::npos) {
        processImageAndSendGraphs(bot, message, filePath, false); // Change to true if you want to convert to BW
    } else if (!message->caption.empty() && message->caption.find("/bw") != std::string::npos) {
        {
            processImageAndSendGraphs(bot, message, filePath, true);
        } else {
            bot.getApi().sendMessage(message->chat->id, "Send /graph to process this image.");
        }
    } else {
        bot.getApi().sendMessage(message->chat->id, "Failed to download image.");
    }
}
```

или:

```
if (downloadFile(fileUrl, filePath)) {
    bot.getApi().sendMessage(message->chat->id, "Image received.");
    if (!message->caption.empty() && message->caption.find("/graph") != std::string::npos) {
        // Determine processing options based on message contents
        bool convertToBW = message->caption.find("--bw") != std::string::npos;
        bool showRGB = message->caption.find("--rgb") != std::string::npos;
        bool drawCircle = message->caption.find("--circle") != std::string::npos;

        processImageAndSendGraphs(bot, message, filePath, convertToBW, showRGB, drawCircle);
    } else {
        bot.getApi().sendMessage(message->chat->id, "Send /graph to process this image with options like --bw, --rgb, or --circle.");
    }
} else {
    bot.getApi().sendMessage(message->chat->id, "Failed to download image.");
}
}
```

измененный processImageAndSendGraphs:

```
void processImageAndSendGraphs(TgBot::Bot& bot, TgBot::Message::Ptr message, const std::string& filePath, bool
convertToBW = false, bool showRGB = false, bool drawCircle = false) {
    std::string command;

    if (showRGB) {
        ...
    } else if (convertToBW) {
        command = "python3 main.py --bw " + filePath;
    } else if (drawCircle) {
        ...
    } else {
        command = "python3 main.py " + filePath;
    }

    int result = system(command.c_str());
    if (result != 0) {
        bot.getApi().sendMessage(message->chat->id, "Error processing the image.");
        return;
    }

    // Check if the files were created successfully before attempting to send
    if (showRGB) {
        if (std::filesystem::exists("rgb_image.png")) {
            ...
        } else {
            bot.getApi().sendMessage(message->chat->id, "Error: RGB image not found.");
        }
    } else if (convertToBW) {
        if (std::filesystem::exists("bw_image.png")) {
            ...
        }
    }
}
```

```

        bot.getApi().sendPhoto(message->chat->id, TgBot::InputFile::fromFile("bw_image.png", "image/png"));
    } else {
        bot.getApi().sendMessage(message->chat->id, "Error: Processed image not found.");
    }
} else if (drawCircle) {
    if (std::filesystem::exists("image_with_circle.png")) {
        ...
    } else {
        bot.getApi().sendMessage(message->chat->id, "Error: Image with circle not found.");
    }
} else {
    if (std::filesystem::exists("histograms.png") && std::filesystem::exists("dependencies.png")) {
        bot.getApi().sendPhoto(message->chat->id, TgBot::InputFile::fromFile("histograms.png", "image/png"));
        bot.getApi().sendPhoto(message->chat->id, TgBot::InputFile::fromFile("dependencies.png", "image/png"));
    } else {
        bot.getApi().sendMessage(message->chat->id, "Error: Processed images not found.");
    }
}
}
}

```

спец команда -> `command = "python3 main.py --bw " + filePath;`

типы аргументов:

- **/bw** - черно белый формат; (название: bw\_image.png)
- **/rgb** - цветной формат; (название: rgb\_image.png)
- **/circle** - отрисовка круговой области; (название: image\_with\_circle.png)

## 2) **Нахождение максимального/минимального значения и коэф. пропускной способности;**

Максимальная точка (Peak) и минимальная точка (Valley) на графике гистограммы могут быть вычислены следующим образом:

**Максимальная точка (Peak):** пиковое значение, которое представляет наибольшую частоту цвета в изображении. Максимальная точка может быть найдена как индекс с максимальным значением в гистограмме.

**Минимальная точка (Valley):** минимальное значение, которое может представлять менее распространенные цвета. Минимальная точка может быть найдена как индекс с минимальным значением в гистограмме.

### **Формула для вычисления коэффициента пропускной способности**

**Коэффициент пропускной способности (Bandwidth)** часто используется для оценки способности системы передавать информацию и может быть вычислен как:

**$B = f_{min} - (f_{max} - 5000) / f_{max}$** , где  $f_{max}$  - максимальная точка (Peak),  $f_{min}$  - минимальная точка (Valley) на графике.

Допустимое значение -  $[0.001; 0.9]$ , если больше - тогда применение специальных средств по уменьшению коэффициента;

Эти вычисления помогают визуально и количественно оценить цветовую характеристику изображения, что может быть полезно для дальнейшего анализа и обработки данных.



### **Подготовка окружения разработки.**

1. Установка необходимых инструментов для разработки на C/C++ (*например, компиляторы, IDE - Visual Studio, CLion, или другие*).
2. Подготовка библиотек для обработки изображений и компьютерного зрения (*например, OpenCV*).
3. Подготовка библиотек для создания телеграмм бота, или реализации оконного приложения средствами WinAPI.

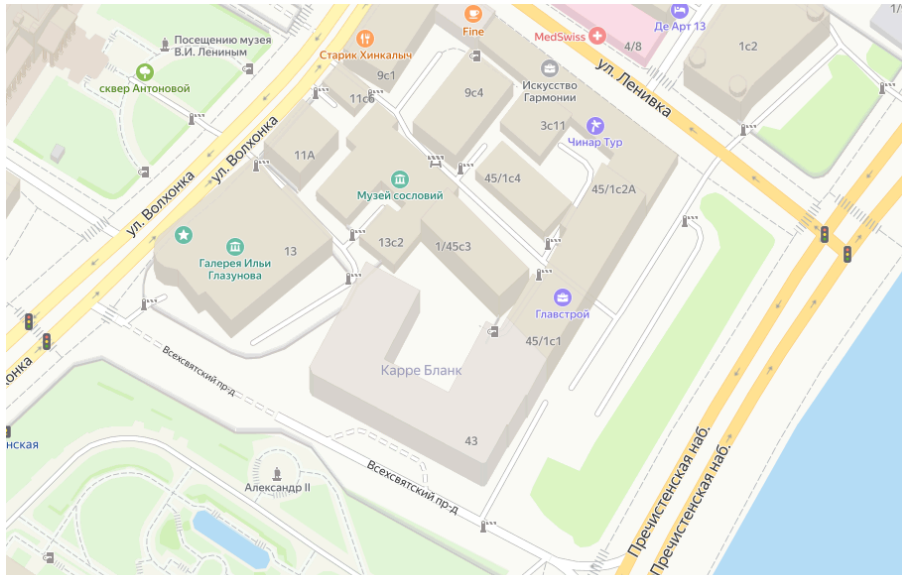
### **Требования.**

1. Соответствие заявленному заданию и методическим указаниям.
2. Использование верных подходов к реализации проекта.
3. Соответствие нормам и стандартам кода C/C++.
4. Применение ООП средств.
5. Применение АИСД средств.
6. Применение Основ языка C/C++.
7. Правильное использование всех библиотек.
8. Описание функций (docstring).
9. Код должен содержать минимум 3 функции и 1 полноценный класс с методами работы.
10. Отступы между функциями (2), между методами класса (1). Отступы между операторами, и тп.
11. Применение средств параллельного программирования (*при необходимости*)
12. Применение средств системного программирования (*при необходимости*)
13. Применение средств функционального программирования (*при необходимости*)
14. Применение WinAPI/QT (*при необходимости*)
15. Правильное оформление отчета согласно заявленным требованиям оформления.

## 1. ЗАДАНИЕ А

1.1 Генерация карты местности (можно использовать как собственные алгоритмы так и взять карту местности у компании “Яндекс”, “Google”, и другие). Карта местности должна содержать минимум 5 различных названий объектов/улиц/домов.

**Пример карты местности:**



1.2 Анализ карты местности на наличие зданий, улиц и тп. Создать таблицу с указанием всех объектов. Напоминаю минимум 5 объектов различного типа!

**Пример таблицы:**

Название/Тип объекта	Описание
Улица 1	ул. Волхонка
Улица 2	Пречистенская наб.
Магазин 1	Главстрой
Памятник 1	Александр II

*\*после построение circle создать таблицу (все объекты который попали в круг) см пункт 2.2.3*

1.3 Создать телеграмм бота через главного бота (*BotFather*). Придумать нормальное имя боту и настроить описание бота (также через *BotFather*), поставить иконку боту. Полученный токен сохранить.

1.4 Подключить библиотеку *tgbot* к проекту. Написать базовый код запуска бота с использованием созданного токена. (примеры кода в Приложении А)

1.5 Реализовать команду **/start** - приветствие бота.

- 1.6 Реализовать команду **/help** - помощь и поддержка от бота (*например вывод доступных команд для работы с ботом*)
- 1.7 Подготовить обработчики изображений/документов (*для сканирования картинки местности*), и другие команды при необходимости.
- 1.8 Сделать микровывод о работе/создании бота, о генерации карты местности и тп.

## 2. ЗАДАНИЕ Б

2.1 Используя базовый (*подготовленный*) код для работы с ботом, реализовать обработку документа (.txt), обработку изображения (.jpeg, .jpg, .png).

**Как это работает?** Вы отправляете боту документ, или картинку - бот принимает это, и обрабатывает - выводит сообщение об успешной обработке или ошибку если что-то пошло не так.

2.2 Написать возможности при отправке картинки с текстом сделать:

2.2.1 Если текст содержит **/bw** (*то сделать фильтр на картинку черно-белый*) - без таблицы;

2.2.2 Если текст содержит **/rgb** (*то сделать цветной фильтр*) - без таблицы;

2.2.3 Если текст содержит **/circle** (*то нарисовать круговую область полупрозрачную - охватывающую объекты местности*) - без таблицы;

2.2.4 Если текста нет или он пустой тогда - стандартная обработка изображения с вывод таблицы.

2.3 Сформулируйте микровывод.

### 3. ЗАДАНИЕ В

3.1 Если в тексте под изображением будет /graph - построить графики распределения цвета на картинке (*гистограмма частот полевых параметров/объектов*).

(см *Методика Расчета*)

3.2 Совместно с графиком распределение цвета - По формулам вычислить максимальную точку, минимальную точку на графике. Вычислить коэффициент пропускной способности (*допустимое значение от 0.001 до 0.9, если выходит за эти пределы, тогда выбрать другое изображение - более качественное*).

(см *Методика Расчета*)

3.3 Проверить код на соответствие всем заявленным требованиям, и наличие ошибок.

3.4 Сформулируйте микровывод.

**ПОСЛЕ ВСЕЙ РАБОТЫ СФОРМУЛИРУЙТЕ ОБЩЕЕ-ТЕЗИСНОЕ  
ЗАКЛЮЧЕНИЕ.**

**ОБЯЗАТЕЛЬНО СОСТАВЬТЕ ОТЧЕТ О ПРОДЕЛАННОЙ РАБОТЕ СОГЛАСНО  
ТРЕБОВАНИЯМ ОТЧЕТНОЙ РАБОТЫ.**

## ПРИЛОЖЕНИЕ А.

### ПРИМЕР КОДА - СОЗДАНИЕ ТЕЛЕГРАММ БОТА И ОБРАБОТКА /START.

```
#include <tgbot/tgbot.h>
#include <iostream>

int main() {
    TgBot::Bot bot("..");
    bot.getEvents().onCommand("start", [&bot](TgBot::Message::Ptr message) {
        bot.getApi().sendMessage(message->chat->id, "Hello! I am your map bot.");
    });

    bot.getEvents().onCommand("help", [&bot](TgBot::Message::Ptr message) {
        bot.getApi().sendMessage(message->chat->id, "Available commands:\n/start - Greet the bot\n/help - Get help");
    });

    try {
        std::cout << "Bot username: " << bot.getApi().getMe()->username.c_str() << std::endl;
        TgBot::TgLongPoll longPoll(bot);
        while (true) {
            std::cout << "Polling..." << std::endl;
            longPoll.start();
        }
    } catch (TgBot::TgException& e) {
        std::cerr << "Error: " << e.what() << std::endl;
    }

    return 0;
}
```

### ПРИМЕР КОДА - ДЛЯ ОБРАБОТЧИКА ЗАГРУЗЧИКА ИЗОБРАЖЕНИЙ.

```
bot.getEvents().onAnyMessage([&bot](TgBot::Message::Ptr message) {
    if (message->photo.size() > 0) {
        std::string fileId = message->photo.back()->fileId;
        TgBot::File::Ptr file = bot.getApi().getFile(fileId);
        std::string filePath = "downloads/" + file->filePath;
        std::string fileUrl = "https://api.telegram.org/file/bot" + bot.getToken() + "/" + file->filePath;

        // Ensure the downloads directory exists
        std::filesystem::create_directories("downloads");

        if (downloadFile(fileUrl, filePath)) {
            bot.getApi().sendMessage(message->chat->id, "Image received.");
            if (!message->caption.empty() && message->caption.find("/graph") != std::string::npos) {
                processImageAndSendGraphs(bot, message, filePath, false); // Change to true if you want to convert
to BW
            } else if (!message->caption.empty() && message->caption.find("/bw") != std::string::npos) {
                processImageAndSendGraphs(bot, message, filePath, true);
            } else {
                bot.getApi().sendMessage(message->chat->id, "Send /graph to process this image.");
            }
        } else {
            bot.getApi().sendMessage(message->chat->id, "Failed to download image.");
        }
    }
});
```