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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений

Студент гр. 3344 _____ Анахин Е.Д.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Анахин Е.Д.

Группа 3344

Тема работы: Обработка изображений.

Исходные данные:

- Программу требуется реализовать в виде утилиты, подобной стандартным *linux*-утилитам.
- Программа должна считать *bmp*-файл без сжатия с 24 битами на цвет
- Программа должна сохранить обработанный *bmp*-файл
- Все поля стандартных *BMP* заголовков в выходном файле должны иметь те же значения что и во входном
- Необходимо использовать *Makefile* для сборки проекта, название исполняемого файла должно быть: *cw*.

Содержание пояснительной записки:

- Содержание
- Введение
- Описание варианта работы
- Описание функций программы
- Описание структуры файлов программы
- Описание сборки проекта
- Примеры работы программы
- Заключение
- Список использованных источников

Предполагаемый объем пояснительной записки:

Не менее 30 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 15.05.2024

Дата защиты реферата: 15.05.2024

Студент _____ Анахин Е.Д.

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

Программа для обработки изображений на языке *C++*, основанная на структуре Pixel, которая хранит информацию о цвете пикселя, реализованная в виде утилиты. Программа предоставляет набор инструментов для фильтрации, рисования и преобразования изображений. Фильтры включают в себя изменение значения *RGB*-компоненты, поиск и замену самого часто встречающегося цвета. Рисование включает в себя создание разделение изображения на N*M частей с помощью линий. Преобразование включает в себя вставку части изображения в другое место, а также отзеркаливание части изображения. Операции выполняются с использованием хэш-таблицы, в которой находятся все аргументы для вызова функции. Результатом работы программы является обработанное изображение с учетом выполненных операций.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель проекта — изучение формата файла BMP и реализация утилиты на языке *C++* для работы с этим форматом. Задачи включают изучение структуры файла *BMP*, получение информацию об изображении, такую как его размеры и содержимое, обработку массива пикселей в соответствии с заданием, обработку исключительных случаев, таких как отсутствие файла или неверный формат, и сохранение итогового изображения в новый файл. Методы будут включать в себя реализацию функций для чтения и записи файлов BMP, а также функций для обработки изображений.

1. ОПИСАНИЕ ВАРИАНТА РАБОТЫ

Программа должна иметь следующую функции по обработке изображений:
Отражение заданной области. Флаг для выполнения данной операции: `--mirror`. Этот функционал определяется:
выбором оси относительно которой отражать (горизонтальная или вертикальная). Флаг `--axis`, возможные значения `x` и `y`
Координатами левого верхнего угла области. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
Координатами правого нижнего угла области. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y

Копирование заданной области. Флаг для выполнения данной операции: `--copy`. Функционал определяется:
Координатами левого верхнего угла области-источника. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
Координатами правого нижнего угла области-источника. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
Координатами левого верхнего угла области-назначения. Флаг `--dest_left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y

Заменяет все пиксели одного заданного цвета на другой цвет. Флаг для выполнения данной операции: `--color_replace`. Функционал определяется:
Цвет, который требуется заменить. Флаг `--old_color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--old_color 255.0.0` задаёт красный цвет)

Цвет на который требуется заменить. Флаг `--new_color` (работает аналогично флагу `--old_color`)

Разделяет изображение на N*M частей. Флаг для выполнения данной операции: `--split`. Реализация: провести линии заданной толщины. Функционал определяется:
Количество частей по “оси” Y. Флаг `--number_x`. На вход принимает число больше 1
Количество частей по “оси” X. Флаг `--number_y`. На вход принимает число больше 1
Толщина линии. Флаг `--thickness`. На вход принимает число больше 0
Цвет линии. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

сгруппировать в несколько файлов (например, функции обработки текста в один, функции ввода/вывода в другой). Сборка должна осуществляться при помощи *make* и *Makefile* или другой системы сборки

2. ОПИСАНИЕ ПРОГРАММЫ

2.1. Описание функций программы

В программе используется класс *Image*, которая представляет изображение и включает в себя заголовок файла *BMPHeader*, заголовок информации *BMPInfoHeader*, высоту *H* и ширину *W* изображения, а также указатель на массив пикселей *bitArr*. Также используется структура *Rgb*, которая представляет из себя 3 числа *r, g, b*, отвечающих за 3 канала пикселя.

Функции и их краткое описание:

- int main(int argc, char** argv)
  Функция, в которой и вызываются все функции для проверки и преобразования изображения
- std::unordered_map(std::string, std::string) (int, char**) - функция, которая парсит флаги и преобразует их в хэш-таблицы с ключ-значением, которые являются флагами для запусками программы и их значениями
- void printHelp()
  Выводит в консоль все возможные флаги для запуска программы.
- std::string findFunctionToRun(char*, std::unordered_map<std::string, std::string>)
  Находит название функции, которую нужно будет выполнить, а также проверяет, не было ли лишних флагов передано, и не были ли забыты какие-то флаги
- bool validateArgs(std::string, std::unordered_map<std::string, std::string>)
  делает первостепенную проверку аргументов для функции, которая должна быть запущена.

- std::vector<std::vector<Pixel>> readBMP() - функция, которая используется для преобразования картинки, которая была получена аргументом –input в массив пикселей - Pixel
- void printFileInfo() - функция, которая выводит небольшую информацию о поданном файле
- void changeColors(infoHeader, bitArr, argsMap) – функция, которая меняет один цвет изображения на другой
- void copyPartOfImage(infoHeader, bitArr, argsMap) – функция, которая копирует часть изображения из одного места в другое, заданное координатами
- void mirror(infoHeader, bitArr, argsMap) – функция, которая зеркалит выделенную область по одной из осей координат
- void split(infoHeader, bitArr, argsMap) – функция, которая разделяет изображения на N*M частей, с помощью рисования на ней линий заданного цвета
- void rgbfilter(string comp_n, int comp_v)
  Применяет фильтр к изображению, изменяя значение одного из компонентов *RGB.*
- writeBMP(argsMap["output"], header, infoHeader, bitArr) – функция, которая записывает данные из обработанного массива с выполненной функцией в файл, который был передан с помощью флага --output

2.2. Описание структуры файлов программы

Программа содержит следующую структуру:

- *Makefile*: Файл для автоматизации процесса компиляции и сборки программы.

- main.cpp — файл, в котором происходит вызов остальных функций для обработки изображения

- input_functions: директория, в которой находятся все функции для того, чтобы валидировать флаги и частично обработать их
- *images_functions*: директория, в которой находятся все функции для считывания изображения, его обработки и сохранения файла

Эта структура файлов позволяет организовать код программы в логически связанные блоки, что облегчает его понимание и поддержку.

2.3. Описание сборки проекта

```
CC = g++
CPPFLAGS = -std=c++11 -O3 -w
LDFLAGS = -lstdc++ -lm
MAKEFLAGS += -j8
SRCDIRS = . images_functions input_functions
SOURCES = $(foreach dir, $(SRCDIRS), $(wildcard $(dir)/*.cpp))
OBJECTS = $(patsubst %.cpp, %.o, $(SOURCES))

%.o: %.cpp
	$(CC) $(CPPFLAGS) -c $< -o $@

cw: $(OBJECTS)
	$(CC) $(LDFLAGS) $(OBJECTS) -o cw

clean:
	rm -f $(OBJECTS) cw
```

3. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Пример 1: Функция mirror

Ввод	Вывод
--mirror –axis y –left.up 10.10 –right.down 200.200	Правильно измененная картинка

Пример 2: Функция copy

Ввод	Вывод
--copy –left.up 10.10 –right.down –100.100 –dest_left_up 20.20	Правильно измененная картинка

Пример 3: Функция color_replace

Ввод	Вывод
--color-replace –old_color – 0.0.0 –new_color 255.255.1	Правильно измененная картинка

Пример 4: Функция split

Ввод	Вывод
--split –number_x 10 –number_y 10 –thickness 20 –color 0.0.0	Правильно измененная картинка

Пример 5: Вывод информации об изображении.

Ввод	Вывод
-I input.bmp	Информация о файле Количество цветов на пиксель: 24 Размер заголовка: 32 Ширина файла: 100 Высота файла: 500

Пример 6: Проверка обработки ошибок.

Ввод	Вывод
-r --component_name green --component_value 100.100.100 input.bmp	Use -help if you don't know how the programm works

ЗАКЛЮЧЕНИЕ

Была успешно реализована программа на языке C++ для обработки изображений в формате BMP. Программа выполняет поставленные задачи, включая чтение и запись изображений, фильтрацию, преобразование цвета, и др. Полученные результаты подтверждают успешное достижение поставленной цели. В ходе выполнения работы были приобретены навыки работы с изображениями, создания Makefile для сборки проекта, использования структур данных, работы с функциями стандартной библиотеки C++, а также оформления кода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Cplusplus. URL: https://cplusplus.com/reference/ (Дата обращения: 29.04.2024)
2. Базовые сведения к выполнению курсовой и лабораторных работ по дисциплине «программирование». Второй семестр: учеб.-метод. пособие др. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2024. 36 с. (Дата обращения: 23.04.2024)
3. Geeksforgeeks. URL: https://www.geeksforgeeks.org (Дата обращения 22.04.2024)

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

main.cpp

```
#include <unordered_map>
#include <string>

#include "input_functions/check_args_amount.h"
#include "input_functions/process_flags.h"
#include "input_functions/validate_args.h"
#include "print_functions.h"
#include "consts.h"
#include "images_functions/getFileInfo.h"
#include "images_functions/color_replace.h"
#include "images_functions/imageStructs.h"
#include "images_functions/new_file.h"
#include "images_functions/copy_part.h"
#include "images_functions/mirror.h"
#include "images_functions/split.h"

int main(int argc, char** argv) {

    if (argc == 2 && (strcmp(argv[1], "--help") == 0 ||
strcmp(argv[1], "-h") == 0)) {
        printHelp();
        return 0;
    }

    if (argc < 2) {
        std::cout << "Please enter flags\n";
        std::cout << USE_HELP;
        return 1;
    }

    std::unordered_map<std::string, std::string> argsMap =
getParams(argc, argv);
    std::string functionToCall = findFunctionToRun(argv[1],
argsMap);
    if (functionToCall == "incorrect") {
        return 1;
    }
```

```
if (functionToCall == HELP) {
    printHelp();
    return 0;
}

if (!validateArgs(functionToCall, argsMap)) {
    std::cout << "Please use flags correctly\n";
    return 1;
}

std::vector<std::vector<Pixel>> bitArr;

BMPHeader header;
BMPInfoHeader infoHeader;

try {
    bitArr = readBMP(argsMap[INPUT], header, infoHeader);
} catch(const std::runtime_error& e) {
    std::cerr << e.what();
    return 0;
}

try {
    if (functionToCall == INFO) {
        printFileInfo(infoHeader);
        return 0;
    }

    if (functionToCall == COLOR_REPLACE) {
        changeColors(infoHeader, bitArr, argsMap);
    }

    if (functionToCall == COPY) {
        copyPartOfImage(infoHeader, bitArr, argsMap);
    }

    if (functionToCall == MIRROR) {
        mirror(infoHeader, bitArr, argsMap);
    }

    if (functionToCall == SPLIT) {
        split(infoHeader, bitArr, argsMap);
    }
```

```
            writeBMP(argsMap["output"], header, infoHeader, bitArr);
        } catch (std::exception &e) {
            std::cout << e.what() << std::endl;
            std::cout << "Program didn't finished successfully\n";
        }

        return 0;
    }
```

print_functions.cpp

```
    #include "print_functions.h"

    void printHelp() {
        std::cout << "Course work for option 5.2, created by Egor
Anakhin\n\n"; // default info
        std::cout << "This program can help you transform your images\n\
    Available flags:\n\n\
    -info (use with --input) - prints info about an image \n\
    -help (this param) - shows help info\n\n\
    (for all commands below you must use --input and --output flags)\n\n\
    --mirror (use with:\n\
        --axis - takes 'x' or 'y'\n\
        --left_up and --right_down - takes X.Y where x and y are coords
by x and y)\n\n\
    --copy (use with:\n\
        --left_up and --right_down\n\
        --dest_left_up - takes X.Y where x and y are coords where image
will be copied)\n\n\
    --color_replace (use with:\n\
        --old_color - takes rrr.ggg.bbb where rrr.ggg.bbb is color in RGB
format\n\
        --new_color - same as --old_color)\n\n\
    --split (use with:\n\
        --number_x  -  takes  integer  number.  Used  to  set  amount  of
columns\n\
        --number_y - same as --numbex_x but for rows\n\
        --thickness  -  takes  integer  number.  Used  to  set  thickness  of
split lines\n\
        --color - takes rrr.ggg.bbb color to set split line color)\n";
    }

    void printFileInfo(BMPInfoHeader infoHeader) {
        std::cout << "Информация о файле\n";
```

```
        std::cout << "Количество цветов на пиксель: " << infoHeader.colorDepth;
        std::cout << "\nРазмер заголовка: " << infoHeader.headerSize;
        std::cout << "\nШирина файла: " <<  infoHeader.width;
        std::cout << "\nВысота файла: " << infoHeader.height;
        std::cout << std::endl;
    }
```

validate_args.cpp

```
    #include "validate_args.h"

    bool validateArgs(std::string functionName, std::unordered_map<std::string, std::string> argsMap) {

        if (functionName == INFO) {
            return true;
        }
        if (functionName == MIRROR) {
            return mirrorValidation(argsMap);
        }
        if (functionName == COPY) {
            return copyValidation(argsMap);
        }
        if (functionName == COLOR_REPLACE) {
            return replaceValidation(argsMap);
        }
        if (functionName == SPLIT) {
            return splitValidation(argsMap);
        }
        std::cerr << "Function validateArgs took unknown param. Egor, fix it\n";
        return false;
    }

    bool isValidCoords(const std::string& str) {
        std::regex pattern("^\\d+\\.\\d+$");

        return std::regex_match(str, pattern);
    }

    bool isValidColor(const std::string& str) {
        std::istringstream iss(str);
        std::string token;
        int count = 0;
```

```
        while (std::getline(iss, token, '.')) {
            if (!token.empty() && token.find_first_not_of("0123456789")
== std::string::npos) {
                int value = std::stoi(token);
                if (value < 0 || value > 255) {
                    return false;
                }
            } else {
                return false;
            }
            count++;
        }

        return count == 3;
    }

    bool isValidNumber(const std::string& str) {
        try {
            int value = std::stoi(str);
            return value >= 1;
        } catch (const std::invalid_argument&) {
            return false;
        } catch (const std::out_of_range&) {
            return false;
        }
    }

    bool  mirrorValidation(std::unordered_map<std::string,  std::string>
argsMap) {
        bool isCool = true;
        if (argsMap[AXIS] != "x" and argsMap[AXIS] != "y") {
            isCool = false;
            std::cout << "Axis must be 'x' or 'y'\n";
        }
        if (!isValidCoords(argsMap[LEFT_UP])) {
            isCool = false;
            std::cout << "left_up must be two int numbers in this form:
X.Y\n";
        }
        if (!isValidCoords(argsMap[RIGHT_DOWN])) {
            isCool = false;
            std::cout << "right_down must be two int numbers in this
form: X.Y\n";
```

```
        }
        return isCool;
    }

    bool copyValidation(std::unordered_map<std::string, std::string>
argsMap) {
        bool isCool = true;
        if (!isValidCoords(argsMap[LEFT_UP])) {
            isCool = false;
            std::cout << "left_up must be two int numbers in this form:
X.Y\n";
        }
        if (!isValidCoords(argsMap[RIGHT_DOWN])) {
            isCool = false;
            std::cout << "right_down must be two int numbers in this
form: X.Y\n";
        }
        if (!isValidCoords(argsMap[RIGHT_DOWN])) {
            isCool = false;
            std::cout << "dest_left_up must be two int numbers in this
form: X.Y\n";
        }
        return isCool;
    }

    bool replaceValidation(std::unordered_map<std::string, std::string>
argsMap) {
        bool isCool = true;
        if (!isValidColor(argsMap[OLD_COLOR])) {
            isCool = false;
            std::cout << "old_color must be rrr.ggg.bbb format\n";
        }
        if (!isValidColor(argsMap[NEW_COLOR])) {
            isCool = false;
            std::cout << "new color myst be rrr.ggg.bbb format\n";
        }
        return isCool;
    }

    bool splitValidation(std::unordered_map<std::string, std::string>
argsMap) {
        bool isCool = true;
        if (!isValidNumber(argsMap[NUMBER_X])) {
            isCool = false;
```

```
            std::cout << "number_x must be integer greater than 1\n";
        }
        if (!isValidNumber(argsMap[NUMBER_Y])) {
            isCool = false;
            std::cout << "number_y must be integer greater than 1\n";
        }
        if (!isValidNumber(argsMap[THICKNESS])) {
            isCool = false;
            std::cout << "thickness must be integer greater than 1\n";
        }
        if (!isValidColor(argsMap[COLOR])) {
            isCool = false;
            std::cout << "color must be rrr.ggg.bbb format";
        }
        return isCool;
    }
```

process_flags.cpp

```
#include "process_flags.h"


void processArguments(int& argc, char**& argv) {
   // Проверяем, есть ли флаг --input
   bool hasInputFlag = false;
   for (int i = 1; i < argc; ++i) {
      if (std::string(argv[i]) == "--input") {
         hasInputFlag = true;
         break;
      }
   }

   // Если флаг --input отсутствует, расширяем массив и добавляем его
   if (!hasInputFlag) {
      // Создаем новый динамический массив для хранения аргументов
      std::vector<char*> newArgv(argc + 2, nullptr); // +2 для флага --input и значения

      // Копируем старые аргументы в новый массив
      for (int i = 0; i < argc; ++i) {
```

```
            newArgv[i] = argv[i];
        }

        // Добавляем новый флаг --input и его значение в предпоследнее место
        newArgv[argc - 1] = const_cast<char*>("--input");
        newArgv[argc] = argv[argc - 1]; // Предполагается, что последний аргумент - имя файла

        // Освобождаем память, выделенную для старого argv
        delete[] argv;

        // Обновляем argc и argv
        argc += 1; // Увеличиваем argc на 1, так как добавили еще один аргумент
        argv = new char*[argc];
        for (int i = 0; i < argc; ++i) {
            argv[i] = newArgv[i];
        }

        // Освобождаем память, выделенную для нового argv, если он был расширен
        if (argc > newArgv.size()) {
            for (int i = newArgv.size(); i < argc; ++i) {
                delete[] argv[i];
            }
        }
    }
}
```

flag_utils.cpp

```
#include "flag_utils.h"

std::string findFunctionToRun(std::string firstFlag, std::unordered_map<std::string, std::string> flags) {

    std::vector<std::string> mirrorFlags  = {"mirror", "axis", "left_up", "right_down", "input", "output"};
```

```
    std::vector<std::string> copyFlags = {"copy", "left_up", "right_down", "dest_left_up", "input", "output"};
    std::vector<std::string> replaceFlags = {"color_replace", "old_color", "new_color", "input", "output"};
    std::vector<std::string> splitFlags = {"split", "number_x", "number_y", "thickness", "color", "input", "output"};
    std::vector<std::string> infoFlags = {"info", "input"};
    std::vector<std::string> helpFlags = {"help"};

    if (firstFlag == "--help" || firstFlag == "-h") {
        return HELP;
    }
    if (firstFlag == "--info") {
        if (isCorrect(infoFlags, flags)) {
            return INFO;
        }
    }
    if (firstFlag == "--mirror") {
        if (isCorrect(mirrorFlags, flags)) {
            return MIRROR;
        }
    }
    if (firstFlag == "--copy") {
        if (isCorrect(copyFlags, flags)) {
            return COPY;
        }
    }
    if (firstFlag == "--color_replace") {
        if (isCorrect(replaceFlags, flags)) {
            return COLOR_REPLACE;
        }
    }
    if (firstFlag == "--split") {
        if (isCorrect(splitFlags, flags)) {
```

```
            return SPLIT;
        }
    }
    return INCORRECT;
}

std::vector<std::string> getExtraKeys(const std::unordered_map<std::string, std::string>& argsMap, const std::vector<std::string>& keys) {
    std::vector<std::string> extraKeys;
    for (const auto& kv : argsMap) {
        if (std::find(keys.begin(), keys.end(), kv.first) == keys.end()) {
            extraKeys.push_back(kv.first);
        }
    }
    return extraKeys;
}

void removeKeysFromVector(std::unordered_map<std::string, std::string>& argsMap, std::vector<std::string>& keys) {
    keys.erase(std::remove_if(keys.begin(), keys.end(),
        [&argsMap](const std::string& key) {
            return argsMap.find(key) != argsMap.end();
        }), keys.end());
}

bool isCorrect(std::vector<std::string>& flagsArr, std::unordered_map<std::string, std::string> argsMap) {

    bool isCorrect = true;

    std::vector<std::string> extraKeys = getExtraKeys(argsMap, flagsArr);

    if (extraKeys.size()) {
        isCorrect = false;
```

```
    }

    removeKeysFromVector(argsMap, flagsArr);
    if (flagsArr.size()) {
        isCorrect = false;
        for (const auto& item: extraKeys) {
            std::cout << "You have an extra flag <" << item << ">. This function doesn't
use it" << std::endl;
        }
    }

    for (const auto& flag : flagsArr) {
        std::cout << "Your function must also have <--" << flag << "> flag" << std::endl;
    }

    if (isCorrect) {
        return true;
    }

    std::cout << USE_HELP << std::endl;

    return false;
}
```

check_args_amount.cpp

```
#include "check_args_amount.h"

struct option long_options[] = {
        {"input", required_argument, 0, 'i'},
        {"output", required_argument, 0, 'o'},
        {"info", no_argument, 0, 0},
        {"help", no_argument, 0, 'h'},
        {"mirror", no_argument, 0, 0},
        {"axis", required_argument, 0, 0},
        {"left_up", required_argument, 0, 0},
```

```
        {"right_down", required_argument, 0, 0},
        {"copy", no_argument, 0, 0},
        {"dest_left_up", required_argument, 0, 0},
        {"color_replace", no_argument, 0, 0},
        {"old_color", required_argument, 0, 0},
        {"new_color", required_argument, 0, 0},
        {"split", no_argument, 0, 0},
        {"number_x", required_argument, 0, 0},
        {"number_y", required_argument, 0, 0},
        {"thickness", required_argument, 0, 0},
        {"color", required_argument, 0, 0},
        {0, 0, 0, 0},
};

std::unordered_map<std::string, std::string> getParams(int argc, char** argv) {
    int opt;
    std::string option_name;
    std::unordered_map<std::string, std::string> argsMap;

    int option_index = 0;
    bool exit_check = 0;
    opterr = 0;   // устанавливается, чтобы не было выводов в консоль при getopt_long

    while ((opt = getopt_long(argc, argv, "i:o:h", long_options, &option_index)) != -1) {
        if (opt == '?') {
            std::cerr << "Unknown argument: " << argv[optind - 1] << std::endl;
            exit_check = true;
            continue;
        }

        if (opt == 'i') {
            option_name = "input";
```

```
        } else if (opt == 'o') {
            option_name = "output";
        } else {
            option_name = long_options[option_index].name;
        }

        if (optarg) {
            argsMap.insert(std::make_pair(option_name, optarg));
        } else {
            argsMap.insert(std::make_pair(option_name, ""));
        }
    }

    if (argsMap.find("input") == argsMap.end()) {
        if (optind < argc) {
            argsMap.insert(std::make_pair("input", argv[argc - 1]));
        } else {
            std::cerr << "Missing input file." << std::endl;
            exit_check = true;
        }
    }

    if (exit_check) {
        std::cout << LOST_PARAMS;
        std::cout << USE_HELP;
        exit(0);
    }
    return argsMap;
}
```

color_replcae.cpp

```
#include "color_replace.h"

void changeColors(BMPInfoHeader infoHeader, std::vector<std::vector<Pixel>>
&bytesARR, std::unordered_map<std::string, std::string> argsMap) {
```

```
    int width = infoHeader.width;
    int height = infoHeader.height;

    Pixel old_color_rgb = parsePixel(argsMap["old_color"]);
    Pixel new_color_rgb = parsePixel(argsMap["new_color"]);

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            if (old_color_rgb == bytesARR[i][j]) {
                bytesARR[i][j] = new_color_rgb;
            }
        }
    }
}
```

copy_part.cpp

```
#include "copy_part.h"

void copyPartOfImage(BMPInfoHeader infoHeader, std::vector<std::vector<Pixel>>
&bytesARR, std::unordered_map<std::string, std::string> argsMap) {
    int x_start;
    int x_end;
    int y_start;
    int y_end;
    int copy_x;
    int copy_y;

    try {
        getCoords(argsMap["left_up"], &x_start, &y_start);
        getCoords(argsMap["right_down"], &x_end, &y_end);
        getCoords(argsMap["dest_left_up"], &copy_x, &copy_y);
    } catch (const std::exception& e) {
        std::cerr << "Error while running programm:\n" << e.what() << std::endl;
    }
```

```
    makeCooler(x_start, y_start, x_end, y_end, infoHeader);

    std::vector<std::vector<Pixel>> copyedArr;

    for (int i = infoHeader.height - y_end; i < infoHeader.height - y_start; i++) {
        std::vector<Pixel> copyLine;
        for (int j = x_start; j < x_end; j++) {
            copyLine.push_back(bytesARR[i][j]);
        }
        copyedArr.push_back(copyLine);
    }

    for (int i = copy_y; i < copy_y + y_end - y_start; i++) {
        for (int j = copy_x; j < copy_x + x_end - x_start; j++) {
            if (i < bytesARR.size() and j < bytesARR[i].size()) {
                bytesARR[i][j] = copyedArr[i - copy_y][j - copy_x];
            }
        }
    }
}
```

getFileInfo.cpp

```
#include "getFileInfo.h"

std::vector < std::vector < Pixel >> readBMP(const std::string & filename,
BMPHeader & header, BMPInfoHeader & infoHeader) {
  std::ifstream file(filename, std::ios::binary);
  if (!file.is_open()) {
    throw std::runtime_error("Error opening file: no such file\n");
  }

  file.read(reinterpret_cast < char * > ( & header), sizeof(header));
  file.read(reinterpret_cast < char * > ( & infoHeader), sizeof(infoHeader));

  if (header.bitmapSignatureBytes != 0x4D42) { // 'BM'
```

```
    throw std::runtime_error("The file is not a bitmap image\n");
  }

  if (infoHeader.colorDepth != 24) {
    throw std::runtime_error("Unsupported color depth. Only 24-bit BMP files are supported.\n");
  }

  if (infoHeader.compressionMethos != 0) {
    throw std::runtime_error("Unsupported compression method. Only uncompressed BMP files are supported.\n");
  }

  int padding = (4 - (infoHeader.width * sizeof(Pixel)) % 4) % 4;

  std::vector < std::vector < Pixel >> pixels(abs(infoHeader.height), std::vector < Pixel > (infoHeader.width));

  file.seekg(header.pixelDataOffset, std::ios::beg);

  for (int i = 0; i < abs(infoHeader.height); i++) {
    for (int j = 0; j < infoHeader.width; j++) {
      Pixel pixel;
      file.read(reinterpret_cast < char * > ( & pixel), sizeof(pixel));
      pixels[i][j] = pixel;
    }
    file.seekg(padding, std::ios::cur);
  }

  file.close();
  return pixels;
}
```

mirror.cpp

```
#include "mirror.h"
```

```
void mirror(BMPInfoHeader infoHeader, std::vector<std::vector<Pixel>> &bytesARR, std::unordered_map<std::string, std::string> argsMap) {
    int width = infoHeader.width;
    int height = infoHeader.height;
    std::string axis = argsMap["axis"];

    int x_start;
    int x_end;
    int y_start;
    int y_end;

    try {
        getCoords(argsMap["left_up"], &x_start, &y_start);
        getCoords(argsMap["right_down"], &x_end, &y_end);
    } catch (const std::exception& e) {
        std::cerr << "Error while running programm:\n" << e.what() << std::endl;
    }

    if (axis == "x") {
        reverseX(width, height, bytesARR, x_start, y_start, x_end, y_end);
    } else if (axis == "y") {
        reverseY(width, height, bytesARR, x_start, y_start, x_end, y_end);
    } else {
        throw std::invalid_argument("Axis can only be x or y");
    }
}

void reverseX(int width, int height, std::vector<std::vector<Pixel>> &bytesARR, int x_start, int y_start, int x_end, int y_end) {
    for (int y = y_start; y <= y_end; ++y) {
        for (int x = x_start; x <= (x_start + x_end) / 2; ++x) {
            int mirror_x = x_end - (x - x_start);
            std::swap(bytesARR[y][x], bytesARR[y][mirror_x]);
```

```
        }
    }
}

void reverseY(int width, int height, std::vector<std::vector<Pixel>> &bytesARR, int x_start, int y_start, int x_end, int y_end) {
    for (int y = y_start; y <= (y_start + y_end) / 2; ++y) {
        int mirror_y = y_end - (y - y_start);
        for (int x = x_start; x <= x_end; ++x) {
            std::swap(bytesARR[y][x], bytesARR[mirror_y][x]);
        }
    }
}
```

new_file.cpp

```
#include "new_file.h"

void writeBMP(const std::string& filename, const BMPHeader& header, const BMPInfoHeader& infoHeader, const std::vector<std::vector<Pixel>>& bitArr) {
    std::ofstream file(filename, std::ios::binary);
    if (!file) {
        throw std::runtime_error("Cannot open file for writing");
    }

    file.write(reinterpret_cast<const char*>(&header), sizeof(header));

    file.write(reinterpret_cast<const char*>(&infoHeader), sizeof(infoHeader));

    for (const auto& row : bitArr) {
        for (const auto& pixel : row) {
            file.write(reinterpret_cast<const char*>(&pixel), sizeof(Pixel));
        }
        for (int i = 0; i < (4 - (infoHeader.width * sizeof(Pixel)) % 4) % 4; ++i) {
            char zero = 0;
            file.write(&zero, 1);
```

```
        }
    }

    file.close();
}
```

parsing_functions.cpp

```
#include "parsing_fucntions.h"

Pixel parsePixel(const std::string& colorString) {
    std::istringstream iss(colorString);
    std::string token;
    Pixel pixel;

    if (std::getline(iss, token, '.')) {
        pixel.red = std::stoi(token);
    }

    if (std::getline(iss, token, '.')) {
        pixel.green = std::stoi(token);
    }

    if (std::getline(iss, token, '.')) {
        pixel.blue = std::stoi(token);
    }

    return pixel;
}


bool operator==(const Pixel& old_color, const Pixel& current_color) {
    return old_color.blue == current_color.blue and old_color.green == current_color.green and old_color.red == current_color.red;
}
```

```
void getCoords(const std::string& str, int* x, int* y) {
    if (x == nullptr || y == nullptr) {
        return;
    }

    size_t dotPos = str.find('.');
    if (dotPos == std::string::npos) {
        throw std::invalid_argument(COORDS_EXCEPTION);
        return;
    }

    try {
        *x = std::stoi(str.substr(0, dotPos));
        *y = std::stoi(str.substr(dotPos + 1));
    } catch (const std::invalid_argument&) {
        throw std::invalid_argument("Invalid format: unable to convert substrings to integers");
    } catch (const std::out_of_range&) {
        throw std::out_of_range("Integer value out of range");
    }
}

void makeCooler(int& x_start, int& y_start, int& x_end, int& y_end, BMPInfoHeader infoHeader) {
    if (x_start < 0) {
        x_start = 0;
    }
    if (y_start < 0) {
        y_start = 0;
    }
    if (x_end > infoHeader.width) {
        x_end = infoHeader.width;
    }
```

```
    if (y_end > infoHeader.height) {
        y_end = infoHeader.height;
    }
}
```

split.cpp

```
#include "split.h"

void split(BMPInfoHeader infoHeader, std::vector<std::vector<Pixel>> &bytesARR,
std::unordered_map<std::string, std::string> argsMap) {
    int width = infoHeader.width;
    int height = infoHeader.height;

    int x_amount = std::stoi(argsMap["number_x"]);
    int y_amount = std::stoi(argsMap["number_y"]);
    int thickness = std::stoi(argsMap["thickness"]);
    Pixel color = parsePixel(argsMap["color"]);

    if (x_amount <= 0 || y_amount <= 0) {
        throw std::invalid_argument("Amounts must be greater than zero");
    }

    int part_width = (width + x_amount - 1) / x_amount;
    int part_height = (height + y_amount - 1) / y_amount;

    for (int i = 1; i < x_amount; ++i) {
        int x = i * part_width - thickness / 2;
        for (int t = 0; t < thickness; ++t) {
            for (int y = 0; y < height; ++y) {
                if (x - thickness / 2 + t >= 0 && x - thickness / 2 + t < width) {
                    bytesARR[y][x - thickness / 2 + t] = color;
                }
                if (x + thickness / 2 + t >= 0 && x + thickness / 2 + t < width) {
                    bytesARR[y][x + thickness / 2 + t] = color;
                }
```

```
            }
        }
    }

    for (int i = 1; i < y_amount; ++i) {
        int y = i * part_height - thickness / 2;
        for (int t = 0; t < thickness; ++t) {
            for (int x = 0; x < width; ++x) {
                if (y - thickness / 2 + t >= 0 && y - thickness / 2 + t < height) {
                    bytesARR[y - thickness / 2 + t][x] = color;
                }
                if (y + thickness / 2 + t >= 0 && y + thickness / 2 + t < height) {
                    bytesARR[y + thickness / 2 + t][x] = color;
                }
            }
        }
    }
}
```