Задания НИИ Фотон

2023-11-13 (b44d5a3)

Содержание

[Работа с изображениями формата BMP 2](#__RefHeading___Toc1826_3534825317)

[Задание bmp.1. Получение значений яркостей 2](#__RefHeading___Toc1828_3534825317)

[Задание bmp.2. Сохранение файла 3](#__RefHeading___Toc1830_3534825317)

[Разработка интерфейсов на Qt 4](#__RefHeading___Toc1832_3534825317)

[Задание Qt.1. Доработка примера просмотрщика изображений Qt 4](#__RefHeading___Toc1834_3534825317)

[Задание Qt.2. Реализация кодека BMP 4](#__RefHeading___Toc1836_3534825317)

[Задание Qt.3. Подсчет статистики изображения 5](#__RefHeading___Toc1838_3534825317)

[Задание Qt.4. Визуализация гистограммы изображения 5](#__RefHeading___Toc1840_3534825317)

[Задание Qt.5. Выделение областей изображения (переход на QGraphicsScene) 5](#__RefHeading___Toc1842_3534825317)

[Задание Qt.6. Открытие изображений большого объема 6](#__RefHeading___Toc1844_3534825317)

[Приложение. Проекты CMake 6](#__RefHeading___Toc1846_3534825317)

[Проект исполняемого файла 6](#__RefHeading___Toc1848_3534825317)

[Проект исполняемого файла с подключением библиотеки 8](#__RefHeading___Toc1850_3534825317)

[Проект исполняемого файла с подключением сторонней библиотеки 9](#__RefHeading___Toc1852_3534825317)

[Приложение. Менеджер пакетов Conan 11](#__RefHeading___Toc1854_3534825317)

[Установка и первичная настройка 11](#__RefHeading___Toc1856_3534825317)

[Использование conan 11](#__RefHeading___Toc1858_3534825317)

[Примеры 12](#__RefHeading___Toc1860_3534825317)

[Qt 12](#__RefHeading___Toc1862_3534825317)

[GoogleTest 14](#__RefHeading___Toc1864_3534825317)

[Приложение. Настройка IDE 17](#__RefHeading___Toc1866_3534825317)

[VS Code 17](#__RefHeading___Toc1868_3534825317)

[Приложение. Рекомендуемые ресурсы и материалы 18](#__RefHeading___Toc1870_3534825317)

[C++ 18](#__RefHeading___Toc1872_3534825317)

[Qt 18](#__RefHeading___Toc1874_3534825317)

[Conan 19](#__RefHeading___Toc1876_3534825317)

[Python 19](#__RefHeading___Toc1878_3534825317)

[Обработка изображений 19](#__RefHeading___Toc1880_3534825317)

[Алгоритмы и структуры данных 19](#__RefHeading___Toc1882_3534825317)

[Linux 19](#__RefHeading___Toc1884_3534825317)

[Git 19](#__RefHeading___Toc1886_3534825317)

# Работа с изображениями формата BMP

## Задание bmp.1. Получение значений яркостей

Написать консольную программу в процедурном стиле, которая принимает один аргумент командной строки - путь к изображению в формате BMP.

Поддерживаемая версия BMP-файлов:

* заголовок: BITMAPINFOHEADER,
* bitCount=24,
* biCompression=BI\_RGB (без сжатия).

В бесконечном цикле читает пользовательский ввод (std::cin). На вход ожидается:

* пустая строка, означающая конец ввода и выход из программы;
* строка из двух координат (x,у) - неотрицательных целых чисел через пробел, в формате <x> <y> (пример: 3 42), вывести в std::cout цвет пикселя в формате <r> <g> <b>, где <r> - код яркости в красном канале, <g> - в зеленом, <b> - в синем.

Необходимо предусмотреть ситуации:

* файл недоступен для чтения - выход с ненулевым кодом возврата;
* файл не является файлом в формате BMP - выход с ненулевым кодом возврата;
* получена ожидаемая строка - обработать запрос;
* формат строки координат некорректен - вывести сообщение об ошибке, продолжить выполнение.

При выполнении задания не использовать сторонние библиотеки кроме std. Файл можно читать в бинарном виде при помощи std::fstream с std::ios::binary.

Значение пикселей получать по запросу пользователя, то есть не следует хранить растр изображения целиком в памяти.

Для начала стоит попробовать написать программу, выводящую в std::cout блоки-описатели BMP: BITMAPFILEHEADER и BITMAPINFO в человекочитаемом виде.

Пример вызова программы:

prog.exe input.bmp

Полезные ссылки по теме:

1. [C++ Binary File I/O](https://courses.cs.vt.edu/~cs2604/fall00/binio.html)
2. [Binary VS Text Mode for File I/O Operations](https://leimao.github.io/blog/File-IO-Binary-VS-Text/)
3. [BMP file format](https://en.wikipedia.org/wiki/BMP_file_format)

Следует обратить внимание на особенности формата: каждая [строка с растром в файле должна иметь размер в байтах, кратным 4](https://en.wikipedia.org/wiki/BMP_file_format" \l "Pixel_storage). То есть ширина строки - это не ширина растра, а ближайшее большее к ширине растра значение, кратное 4.

* [как это выглядит в файле](https://learn.microsoft.com/en-us/windows/win32/medfound/image-stride)
* [как рассчитать ширину строки](https://learn.microsoft.com/en-us/windows/win32/api/wingdi/ns-wingdi-bitmapinfoheader" \l "calculating-surface-stride)

Далее следует учесть, что если значение высоты растра указано положительным, то в файле строки изображения расположены в порядке сначала нижние, потом верхние ([первый абзац](https://en.wikipedia.org/wiki/BMP_file_format" \l "Pixel_array_(bitmap_data))). Если отрицательным, то в порядке от верхних к нижним.

Проверить, что корректно выдаются значения вашей программы можно при помощи программ: ImageJ/Fiji, GIMP. Также можно пользоваться HEX-редакторами для просмотра значений байтов файла. В них также могут быть плагины для показа заголовков BMP. На Linux это, например, программа Okteta.

Срок выполнения: 1 неделя.

## Задание bmp.2. Сохранение файла

Написать консольную программу в процедурном стиле, которая принимает два аргумента командной строки:

1. путь к входному изображению в формате BMP;
2. путь к выходному файлу в формате BMP.

Поддерживаемая версия BMP-файлов:

* заголовок: BITMAPINFOHEADER,
* bitCount=24,
* biCompression=BI\_RGB (без сжатия).

Программа должна выполнить зеркальное отражение относительно вертикали исходного изображения и результат записать в выходной файл. При этом необходимо выполнять построчную обработку, то есть одновременно хранить в памяти только обрабатываемую строку.

Пример вызова программы:

prog.exe input.bmp output.bmp

Срок выполнения: 1 неделя.

# Разработка интерфейсов на Qt

## Задание Qt.1. Доработка примера просмотрщика изображений Qt

На основе [примера из документации Qt](https://doc.qt.io/qt-5/qtwidgets-widgets-imageviewer-example.html) разработать приложение для открытия файлов формата BMP (реализовывать функциональность печати/копирования/вставки не нужно).

Выполнить следующие доработки примера:

1. добавить панель инструментов, дублирующая пункты меню «File» и «View» в виде иконок;
2. добавить панель инструментов с QLabel, в которой отображаются планарные координаты изображения, над которым находится курсор мыши.

Срок выполнения: 2 недели.

## Задание Qt.2. Реализация кодека BMP

Доработать приложение так, чтобы вместо класса QImageReader использовался собственный класс чтения BMP.

Поддерживаемая версия BMP-файлов:

* заголовок: BITMAPINFOHEADER,
* bitCount=24,
* biCompression=BI\_RGB (без сжатия).

Срок выполнения: 1 неделя.

## Задание Qt.3. Подсчет статистики изображения

Добавить функциональность подсчета статистики изображения в виде пункта меню/кнопки инструментальной панели, а именно для каждого канала изображения:

1. количество пикселей;
2. математическое ожидание величин яркости;
3. среднеквадратическое отклонение величин яркости.

Рассчитанные характеристики выводить или в отдельный диалог, или панель (область) главного окна.

Рекомендация: доступ к растру осуществлять не через поле класса QImage image, а через свой промежуточный класс.

Срок выполнения: 1 неделя.

## Задание Qt.4. Визуализация гистограммы изображения

Добавить функциональность расчета и визуализации яркостной гистограммы каждого канала изображения. Яркостная гистограмма - столбчатая диаграмма, по оси X располагаются значения яркости от 0 до глубины цвета, по оси Y - количество пикселей с соответствующим значением яркости.

Гистограмму выводить или в отдельный диалог, или панель (область) главного окна.

Срок выполнения: 2 недели.

## Задание Qt.5. Выделение областей изображения (переход на QGraphicsScene)

Добавить в созданное приложение возможность интерактивного выделения прямоугольной области интереса (по аналогии с графическими редакторами).

Для этого при визуализации изображения перейти c QLabel на QGraphicsScene.

Параметры выделенной области (координаты левого верхнего угла, ширина и высота) должны отображаться на панели выделения и обновляться в ходе модификации выделения. Предусмотреть возможность отмены выделения.

При этом дальнейшие операции расчёта статистики гистограмм должны учитывать выделенную область.

Срок выполнения: 2 недели, с разбиением на этапы:

1. 1 неделя: использование QGraphicsScene;
2. 2 неделя: реализация областей выделения.

## Задание Qt.6. Открытие изображений большого объема

Доработать приложение так, чтобы обеспечить возможность визуализации очень больших изображений, не помещающихся в оперативной памяти целиком.

Для этого необходимо решить две подзадачи:

* обеспечить потайловую визуализацию, без помещения всего изображения в QImage/QPixmap или их аналог;
* обеспечить возможность чтения фрагментов изображения «на лету», чтобы избежать чтения всего изображения в оперативную память.

Срок выполнения: 4 недели, с разбиением на этапы:

1. 2я неделя: доработка класса кодека для обеспечения возможности заполнения фрагмента исходного изображения с прореживанием (берем каждую n-ную строку вместо следующей)

* Например, прочитать фрагмент изображения размером 256х256 с прореживанием 1х1 от левого верхнего угла с координатами [512,200]

1. 4я неделя: реализация визуализации исходного изображения «на лету»

# Приложение. Проекты CMake

## Проект исполняемого файла

Структура директории

./  
├── CMakeLists.txt  
└── main.cpp

Файл CMakeLists.txt. Содержит: тип проекта, список файлов исходного кода и заголовочников, флаги компиляторов.

cmake\_minimum\_required(VERSION 3.5)  
  
# Наименование проекта и исполняемого файла  
project(prog LANGUAGES CXX)  
  
# Используемый стандарт C++14  
set(CMAKE\_CXX\_STANDARD 14)  
set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)  
  
# Полезные флаги компиляции  
if (CMAKE\_CXX\_COMPILER\_ID STREQUAL "GNU") # В этом блоке дополнительные настройки для GCC (Linux)  
 set(CMAKE\_SHARED\_LINKER\_FLAGS "${CMAKE\_SHARED\_LINKER\_FLAGS} -Wl,--no-undefined")  
 add\_compile\_options(  
 -Werror=return-type  
 -Wfatal-errors  
 -Wall  
 -Wextra  
 -Wwrite-strings  
 -Winit-self  
 )  
elseif (CMAKE\_CXX\_COMPILER\_ID STREQUAL "MSVC") # В этом блоке дополнительные настройки для MSVC (Windows)  
 add\_compile\_options("/W4")  
endif()  
  
# Список файлов исходного кода и заголовочных  
add\_executable(${PROJECT\_NAME}  
 main.cpp  
)

Файл main.cpp - точка входа в программу.

#include <iostream>  
  
using namespace std;  
  
int main() {  
 cout << "Hello World!" << endl;  
 return 0;  
}

Сборка проекта (командная строка относительно директории сборки, например, в директории <корень проекта>/build/):

1. Конфигурация сборки:
   * Windows: «cmake ..»
   * Linux: «cmake -DCMAKE\_BUILD\_TYPE=Debug ..»
2. Сборка проекта: «cmake --build .»
   * Windows: «cmake --build . --config Debug»
   * Linux: «cmake --build .»

## Проект исполняемого файла с подключением библиотеки

Структура директории

./  
├── CMakeLists.txt  
├── main.cpp  
└── src/  
 ├── CMakeLists.txt  
 ├── hello.cpp  
 └── hello.h

Файл CMakeLists.txt (дополнительные флаги и комментарии опущены)

cmake\_minimum\_required(VERSION 3.5)  
  
project(prog LANGUAGES CXX)  
  
set(CMAKE\_CXX\_STANDARD 14)  
set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)  
  
# Добавление поддиректории с проектом статической библиотеки  
add\_subdirectory(src)  
  
add\_executable(${PROJECT\_NAME} main.cpp)  
  
# Линковка с библиотекой  
target\_link\_libraries(${PROJECT\_NAME} prog-lib)

Файл main.cpp

#include <hello.h>  
  
int main() {  
 hello();  
}

Файл src/CMakeLists.txt

cmake\_minimum\_required(VERSION 3.8)  
  
project(prog-lib)  
  
add\_library(${PROJECT\_NAME}  
 hello.h  
 hello.cpp  
)  
  
target\_include\_directories(${PROJECT\_NAME} INTERFACE  
 $<BUILD\_INTERFACE:${CMAKE\_CURRENT\_SOURCE\_DIR}>  
 $<INSTALL\_INTERFACE:include>  
)

Файл src/hello.cpp

#include "hello.h"  
  
#include <iostream>  
  
void hello() {  
 std::cout << "Hello, world!" << std::endl;  
}

Файл src/hello.h

#pragma once  
  
void hello();

## Проект исполняемого файла с подключением сторонней библиотеки

Пример подключения библиотеки Boost::program\_options, установленную в системе.

Структура директории

./  
├── CMakeLists.txt  
└── main.cpp

Файл CMakeLists.txt (дополнительные флаги и комментарии опущены)

cmake\_minimum\_required(VERSION 3.5)  
  
project(prog LANGUAGES CXX)  
  
set(CMAKE\_CXX\_STANDARD 14)  
set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)  
  
# поиск библиотеки Boost::program\_options в системе  
find\_package(Boost REQUIRED COMPONENTS program\_options)  
  
add\_executable(${PROJECT\_NAME} main.cpp)  
  
# динамическая линковка с библиотекой Boost::program\_options  
target\_link\_libraries(${PROJECT\_NAME} Boost::program\_options)

Файл main.cpp

#include <iostream>  
  
#include <boost/program\_options.hpp>  
  
using namespace std;  
namespace po = boost::program\_options;  
  
int main(int argc, char\*\* argv) {  
 try {  
 po::options\_description desc("Allowed options");  
 desc.add\_options()("help", "produce help message")("compression", po::value<int>(), "set compression level");  
  
 po::variables\_map vm;  
 po::store(po::parse\_command\_line(argc, argv, desc), vm);  
 po::notify(vm);  
  
 if (vm.count("help")) {  
 cout << desc << endl;  
 return 1;  
 }  
  
 if (vm.count("compression")) {  
 cout << "Compression level was set to " << vm["compression"].as<int>() << endl;  
 } else {  
 cout << "Compression level was not set." << endl;  
 }  
 } catch (std::exception& e) {  
 cerr << e.what() << std::endl;  
 }  
}

# Приложение. Менеджер пакетов Conan

## Установка и первичная настройка

Установка и первое использование.

1. conan - Python3 утилита, для установки необходим pip (или pipx, если не получится через pip):

* pip install conan

1. Для использования необходимо инициализировать конфигурационный файл профиля сборки загружаемых пакетов

* conan profile detect --force
* В файле профиля будут указаны глобальные настройки сборки, которые можно переопределить в команде conan.

## Использование conan

В корне директории проекта создается файл conanfile.txt со следующим содержимым:

[requires]  
библиотека1/версия1  
библиотека2/версия2  
библиотека3/версия3  
...  
  
[generators]  
CMakeDeps  
CMakeToolchain

Conan собирает зависимости для конкретного типа сборки (Debug/Release). Для заданного <типа сборки> выполнить следующие команды в терминале (необходимо наличие компиляторов и бинарников cmake в PATH). Для Windows - [в терминале VisualStudio](https://learn.microsoft.com/en-us/visualstudio/ide/reference/command-prompt-powershell?view=vs-2022).

1. Проинициализировать зависимости conan командой из директории с conanfile.txt

* conan build . --output-folder <путь к директории сборки> --build missing --settings=build\_type=<тип сборки>
* Эта команда запустит сборку Qt5 из исходников. В отличие от сборки руками, она за тебя проставит нужные флаги компилятора, скачает и соберет зависимости и так далее. Это займет время, возможно, часы. В результате выполнения прошлой команды получили установленный Qt, который должен цепляться CMake’ом.

1. Удалить появившийся файл CMakeUserPresets.txt.
2. Запустить CMake внутри директории сборки

* cmake <путь к директории с CMakeLists.txt> -DCMAKE\_TOOLCHAIN\_FILE=conan\_toolchain.cmake
* в Linux добавить аргумент командной строки: «-DCMAKE\_BUILD\_TYPE=<тип сборки>»

1. Собрать проект командой внутри директории сборки
   * Windows: «cmake --build . --config <тип сборки>»
   * Linux: «cmake --build .»

## Примеры

### Qt

#### Структура проекта

./  
├── CMakeLists.txt  
├── conanfile.txt  
└── main.cpp

#### Содержание файлов

Файл conanfile.txt

[requires]  
qt/5.15.10  
  
[generators]  
CMakeDeps  
CMakeToolchain

Файл CMakeLists.txt

cmake\_minimum\_required (VERSION 3.8)  
project (helloworld )  
find\_package(Qt5 COMPONENTS Widgets REQUIRED)  
add\_executable(helloworld main.cpp)  
target\_link\_libraries(helloworld  
 Qt5::Widgets  
 Qt5::Gui  
 Qt5::QWindowsIntegrationPlugin)

Файл main.cpp

#include <QApplication>  
#include <QDialog>  
#include <QLabel>  
  
#ifdef WIN32  
#include <QtPlugin>  
Q\_IMPORT\_PLUGIN(QWindowsIntegrationPlugin);  
#endif  
  
int main(int argc, char\* argv[]) {  
 QApplication app(argc, argv);  
 QDialog\* dialog = new QDialog;  
 QLabel\* label = new QLabel(dialog);  
 label->setText("<font color=red>Hello, World!</font>");  
 dialog->show();  
 return app.exec();  
}

#### Последовательность действий

Qt будем собирать и линковать статически, чтобы полученный exe-файл был «самодостаточным».

1. Команда сборки Qt (директория сборки: ./build/, тип сборки: Debug):

* conan install . --output-folder=build --build=missing -o:b=Qt/\*:shared=false -o:h=Qt/\*:shared=false --settings=compiler.runtime=static --settings=build\_type=Debug

1. Команда конфигурирования CMake из директории сборки:
   * Windows
   * cmake ..
   * Linux
   * cmake -DCMAKE\_BUILD\_TYPE=Debug ..
2. Команда сборки из директории сборки
   * Windows
   * cmake --build . --config Debug
   * Linux
   * cmake --build .

### GoogleTest

#### Структура проекта

./  
├── CMakeLists.txt  
├── conanfile.txt  
├── main.cpp  
├── src/  
│   ├── CMakeLists.txt  
│   ├── square.cpp  
│   └── square.h  
└── test/  
 ├── CMakeLists.txt  
 └── square\_test.cpp

#### Содержание файлов

Файл ./test/square\_test.cpp

#include "gtest/gtest.h"  
  
#include <square.h>  
  
TEST(squaretests, basic) {  
 EXPECT\_DOUBLE\_EQ(9., square(3.));  
 EXPECT\_DOUBLE\_EQ(25., square(5.));  
}

Файл ./test/CMakeLists.txt

cmake\_minimum\_required(VERSION 3.8)  
  
project(prog-test)  
  
find\_package(GTest REQUIRED)  
add\_executable(${PROJECT\_NAME}  
 square\_test.cpp  
)  
  
target\_link\_libraries(${PROJECT\_NAME}  
 gtest::gtest  
 prog-lib  
)  
  
add\_test(NAME ${PROJECT\_NAME} COMMAND ${PROJECT\_NAME})

Файл ./main.cpp

#include <iostream>  
  
#include <square.h>  
  
int main() {  
 std::cout << square(5) << std::endl;  
}

Файл ./conanfile.txt

[requires]  
gtest/1.8.1  
  
[generators]  
CMakeDeps  
CMakeToolchain

Файл ./src/square.cpp

#include "square.h"  
  
double square(double x) {  
 return x\*x;  
}

Файл ./src/CMakeLists.txt

cmake\_minimum\_required(VERSION 3.8)  
  
project(prog-lib)  
  
add\_library(${PROJECT\_NAME}  
 square.h  
 square.cpp  
)  
  
target\_include\_directories(${PROJECT\_NAME} INTERFACE  
 $<BUILD\_INTERFACE:${CMAKE\_CURRENT\_SOURCE\_DIR}>  
 $<INSTALL\_INTERFACE:include>  
)

Файл ./src/square.h

#pragma once  
  
double square(double x);

Файл ./CMakeLists.txt

cmake\_minimum\_required (VERSION 3.8)  
  
project(prog)  
  
set(CMAKE\_CXX\_STANDARD 14)  
set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)  
  
if (CMAKE\_CXX\_COMPILER\_ID STREQUAL "GNU")  
 set(CMAKE\_SHARED\_LINKER\_FLAGS "${CMAKE\_SHARED\_LINKER\_FLAGS} -Wl,--no-undefined")  
 add\_compile\_options(  
 -Werror=return-type  
 -Wfatal-errors  
 -Wall  
 -Wextra  
 -Wwrite-strings  
 -Winit-self  
 )  
elseif (CMAKE\_CXX\_COMPILER\_ID STREQUAL "MSVC")  
 add\_compile\_options("/W4")  
endif()  
  
add\_subdirectory(src)  
add\_executable (${PROJECT\_NAME}  
 main.cpp  
)  
  
target\_link\_libraries(${PROJECT\_NAME} prog-lib)  
  
install(TARGETS ${PROJECT\_NAME} DESTINATION ${CMAKE\_INSTALL\_PREFIX})  
  
option(BUILD\_TESTS "Build the tests" ON)  
if(BUILD\_TESTS)  
 enable\_testing()  
 add\_subdirectory(test)  
endif()

# Приложение. Настройка IDE

## VS Code

**Важно**: VS Code должен запускаться из терминала разработчика.

Плагины:

* «CMakeTools» Microsoft;
* «C/C++» Microsoft.

Файл .vscode/settings.json из корня проекта позволяет указывать настройки плагинов. В частности, пригодятся способы:

* указания директории сборки CMake;
* указания параметров первичной команды cmake (построение кэша сборки).

Пример файла .vscode/settings.json;

{  
 "cmake.buildDirectory": "${workspaceFolder}"/build, // то есть путь к директории сборки: build  
 "cmake.configureArgs": [  
 "-DCMAKE\_POLICY\_DEFAULT\_CMP0091=NEW", // указание флага сборки проекта для линковки с Conan, может быть решением соответствующего сообщения об ошибке  
 "-DBUILD\_TESTS=ON", // Включение тестов для проекта с GoogleTest  
 "-DCMAKE\_TOOLCHAIN\_FILE=conan\_toolchain.cmake", // использование тулчейна conan  
 ]  
}

Работа с проектом в VS Code:

1. Открыть директорию в code
2. Ctrl+Shift+P: cmake configure
3. Ctrl+Shift+P: cmake build
4. Ctrl+Shift+P: cmake set debug target, выбрать цель тестов
5. Ctrl+Shift+P: cmake run without...

Чтобы отлаживать нужно выбрать отлаживаемую цель и через Ctrl+Shift+P: cmake debug

Чтобы передать аргументы командной строки при запуске нужно [почитать про файл](https://vector-of-bool.github.io/docs/vscode-cmake-tools/debugging.html) .vscode/launch.json

# Приложение. Рекомендуемые ресурсы и материалы

## C++

1. Курсы
   1. Программирование на языке C++: https://stepik.org/course/7
   2. Программирование на языке C++ (продолжение): https://stepik.org/course/3206
      * не обязателен к прохождению раздел 5: Многопоточность, библиотеки и метапрограммирование
   3. Основы С++: https://academy.yandex.ru/handbook/cpp/article/introduction
2. Книга
   1. Бьярне Страуструп «Язык программирования C++. Краткий курс»

### Qt

1. М. Шлее «Qt 5.10. Профессиональное программирование на С++»
2. Официальная документация https://doc.qt.io/qt-5.15/classes.html

### Conan

1. https://docs.conan.io/2/tutorial.html, а именно раздел Consuming packages:
   * Build a simple CMake project using Conan;
   * Using build tools as Conan packages;
   * Building for multiple configurations: Release, Debug, Static and Shared.
2. https://youtu.be/X4bPsRAXHRs?t=435 до 12й минуты

## Python

1. Программирование на Python: https://stepik.org/67

## Обработка изображений

1. Курсы
   1. Обзорный курс (Python): https://stepik.org/course/1280
2. Книги
   1. Гонсалес Р., Вудс Р. «Цифровая обработка изображений»
   2. Прэтт Э. «Цифровая обработка изображений»

## Алгоритмы и структуры данных

1. Алгоритмы: теория и практика. Методы: https://stepik.org/course/217
2. Алгоритмы: теория и практика. Структуры данных: https://stepik.org/course/1547
3. Онлайн задачники
   1. https://leetcode.com/
   2. https://www.hackerrank.com/
   3. https://acm.timus.ru/problemset.aspx

## Linux

1. Введение в Linux: https://stepik.org/course/73

## Git

1. Основы Git: https://stepik.org/3145