

Все статьи / Простые программы на C++ с библиотекой SFML

В статье мы напишем пару простейших программ на C++ и посмотрим, как происходит их компиляция.

Содержание

- Простая программа со сборкой через CMake
- Закрепим результат с помощью git
- Простая программа с библиотекой SFML
- Освоим прямоугольники и повороты

Следуйте инструкциям. Также выполните задания, указанные в тексте.

Простая программа со сборкой через CMake

CMake - это система сборки. Нам она позволит не печатать каждый раз команды компиляции с нуля, тем более что команды будут становиться всё сложнее и сложнее.

Создайте каталог и откройте его в Visual Studio Code. Создайте файл с именем `CMakeLists.txt`. Скопируйте следующий текст в файл и сохраните:

```
cmake_minimum_required(VERSION 3.8 FATAL_ERROR)

project(cpp-samples)

add_subdirectory(00)
```

Теперь создайте каталог `00` и в нём также создайте файл с именем `CMakeLists.txt`. Сохраните в нём следующий текст:

```
add_executable(00 main.cpp)
```

Затем создайте файл `main.cpp` в каталоге `00` и напишите в нём любую простую программу, например:

```
#include <iostream>

int main()
{
    int a = 0;
    int b = 0;
    std::cout << "enter two numbers:" << std::endl;
    std::cin >> a >> b;
    std::cout << "sum: " << (a + b) << std::endl;
}
```

1. Откройте терминал и введите команду `g++ 00/main.cpp -o 00.exe`. Убедитесь, что компиляция прошла успешно, затем выполните команду `00.exe`.
2. Удалите программу командой `del 00.exe`
3. Затем соберите программу с помощью CMake. Эта операция выполняется в два шага: конфигурирование + сборка.

```
cmake -G "MinGW Makefiles" .
cmake --build .
```

Вывод первой команды будет примерно таким:

```
D:\Projects\CppDistro\workspace>cmake -G "MinGW Makefiles" .
-- The C compiler identification is GNU 7.2.0
-- The CXX compiler identification is GNU 7.2.0
...
-- Configuring done
-- Generating done
-- Build files have been written to: D:/Projects/CppDistro/workspace
```

Вывод второй команды будет выглядеть так:

```
D:\Projects\CppDistro\workspace>cmake --build .
Scanning dependencies of target 00
[ 50%] Building CXX object 00/CMakeFiles/00.dir/main.cpp.obj
[100%] Linking CXX executable 00.exe
[100%] Built target 00
```

Если сборка была успешной, запустите `00\00.exe`, чтобы проверить, что программа доступна и работоспособна.

Поздравляем, теперь вы умеете собирать проекты с помощью CMake!

Закрепим результат с помощью git

Чтобы ваши достижения не потерялись, вам надо использовать систему контроля версий Git. В консоли введите следующие команды

```
git init
git add CMakeLists.txt
git add 00/CMakeLists.txt
git add 00/main.cpp
```

Ни одна из этих команд `git add` не должна была выводить сообщений. Если где-то было сообщение, значит, у вас возникла ошибка. В этом случае проверьте команду и введите её снова.

Теперь введите команду `git status` и посмотрите на её вывод. Зелёным цветом помечены файлы, добавленные под контроль версий, красным помечены остальные файлы. Если вывод команды у вас совпадает со скриншотом, двигаться идти дальше.

```
D:\Projects\CppDistro\workspace>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   00/CMakeLists.txt
        new file:   00/main.cpp
        new file:   CMakeLists.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        00/00.exe
        00/CMakeFiles/
        00/Makefile
        00/cmake_install.cmake
        CMakeCache.txt
        CMakeFiles/
        Makefile
        cmake_install.cmake
```

Пришло время зафиксировать свою первую версию! Введите команду `git commit -m "Added A+B program"`. Git зафиксирует версию и выведет краткий отчёт:

```
D:\Projects\CppDistro\workspace>git commit -m "Added A+B program"
[master (root-commit) 340778e] Added A+B program
 3 files changed, 16 insertions(+)
 create mode 100644 00/CMakeLists.txt
 create mode 100644 00/main.cpp
 create mode 100644 CMakeLists.txt
```

Теперь введите команду `git clean -fd`. О чудо! Все посторонние файлы, кроме исходного кода, теперь исчезли. Команда убрала всё, что не занесено под контроль версий.

Простая программа с библиотекой SFML

Создать графическую программу немногим сложнее, чем консольную, благодаря SFML. Для начала создайте каталог `01`, и в нём создайте файл `CMakeLists.txt`, сохраните в нём текст:

```
add_executable(01 main.cpp)

set(SFML_STATIC_LIBRARIES TRUE)

find_package(Freetype)
find_package(JPEG)
find_package(SFML 2 COMPONENTS window graphics system)

target_compile_features(01 PUBLIC cxx_std_17)
target_compile_definitions(01 PRIVATE SFML_STATIC)

target_link_libraries(01 ${SFML_LIBRARIES} ${SFML_DEPENDENCIES})
```

Теперь создайте в каталоге `01` файл `main.cpp`, и аккуратно перепишите в него код:

```
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>

int main()
{
    sf::RenderWindow window(sf::VideoMode({ 800, 600 }), "title");

    window.clear();

    sf::CircleShape shape1(40);
    shape1.setPosition({ 200, 120 });
    shape1.setFillColor(sf::Color(0xFF, 0x0, 0x0));
    window.draw(shape1);

    window.display();

    sf::sleep(sf::seconds(5));
}
```

Теперь вернитесь к основному файлу `CMakeLists.txt` и добавьте строку `add_subdirectory(01)`, чтобы он выглядел так:

```
cmake_minimum_required(VERSION 3.8 FATAL_ERROR)

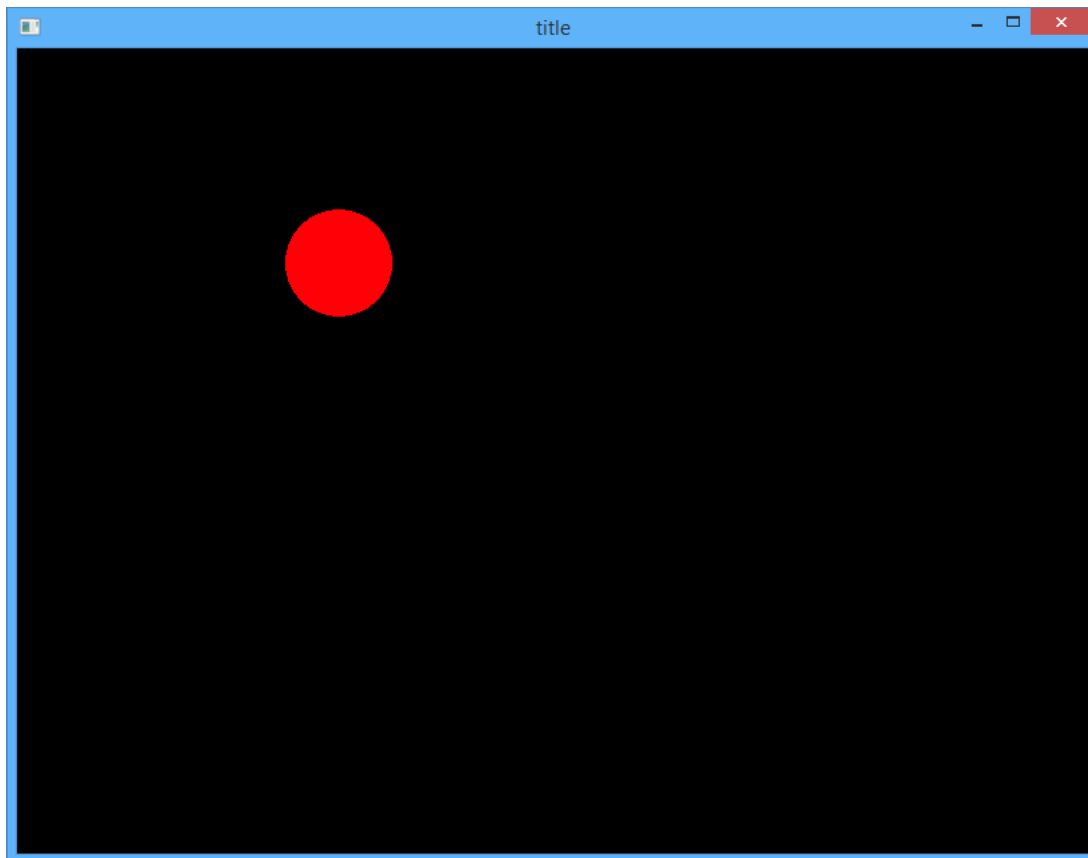
project(cpp-samples)

add_subdirectory(00)
add_subdirectory(01)
```

После этого снова запустите CMake:

```
cmake -G "MinGW Makefiles"
cmake --build .
```

Если сборка была успешной, запустите программу `01\01.exe`. На 5 секунд появится окно, а затем оно исчезнет.



Окно показывается, пока выполняется инструкция `sf::sleep(sf::seconds(5));`, а затем программа заканчивается и окно скрывается. В будущем мы научимся показывать окно столько, сколько захотим, и закрывать по нажатию на кнопку в верхнем правом углу окна.

Давайте разберём, что выполняет код:

1. Инструкция `sf::RenderWindow window(sf::VideoMode({800, 600}), "title")` объявляет переменную `window`, которая имеет тип “графическое окно”, размеры 800x600 px и незамысловатый заголовок “title”.
2. Вызов метода `window.clear()` очищает окно, а вызов `window.display()` выводит на экран то, что было нарисовано после очистки.
 - о подробнее о методах `sf::RenderWindow` вы узнаете [в документации SFML \(sfml-dev.org\)](http://sfml-dev.org).
3. Инструкция `sf::CircleShape shape1(40)` объявляет переменную `shape1` - круг размером 40px.
4. Вызовы `shape1.setPosition` и `shape1.setFillColor` задают позицию круга в пикселях и цвет в виде RGB-компонентов (Red, Green, Blue).
 - о подробнее о методах `CircleShape` вы узнаете [на странице документации \(sfml-dev.org\)](http://sfml-dev.org)

Теперь добавьте больше фигур:

```
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>

int main()
{
    sf::RenderWindow window(sf::VideoMode({ 800, 600 }), "title");

    window.clear();

    sf::CircleShape shape1(40);
    shape1.setPosition({ 200, 120 });
    shape1.setFillColor(sf::Color(0xFF, 0x0, 0x0));
    window.draw(shape1);

    sf::CircleShape shape2(48);
    shape2.setPosition({ 260, 120 });
    shape2.setFillColor(sf::Color(0x0, 0xFF, 0x0));
    window.draw(shape2);

    sf::CircleShape shape3(60);
    shape3.setPosition({ 320, 160 });
    shape3.setFillColor(sf::Color(0x0, 0x0, 0xFF));
    window.draw(shape3);

    sf::CircleShape shape4(75);
    shape4.setPosition({ 330, 220 });
    shape4.setFillColor(sf::Color(0xFF, 0xFF, 0x0));
    window.draw(shape4);

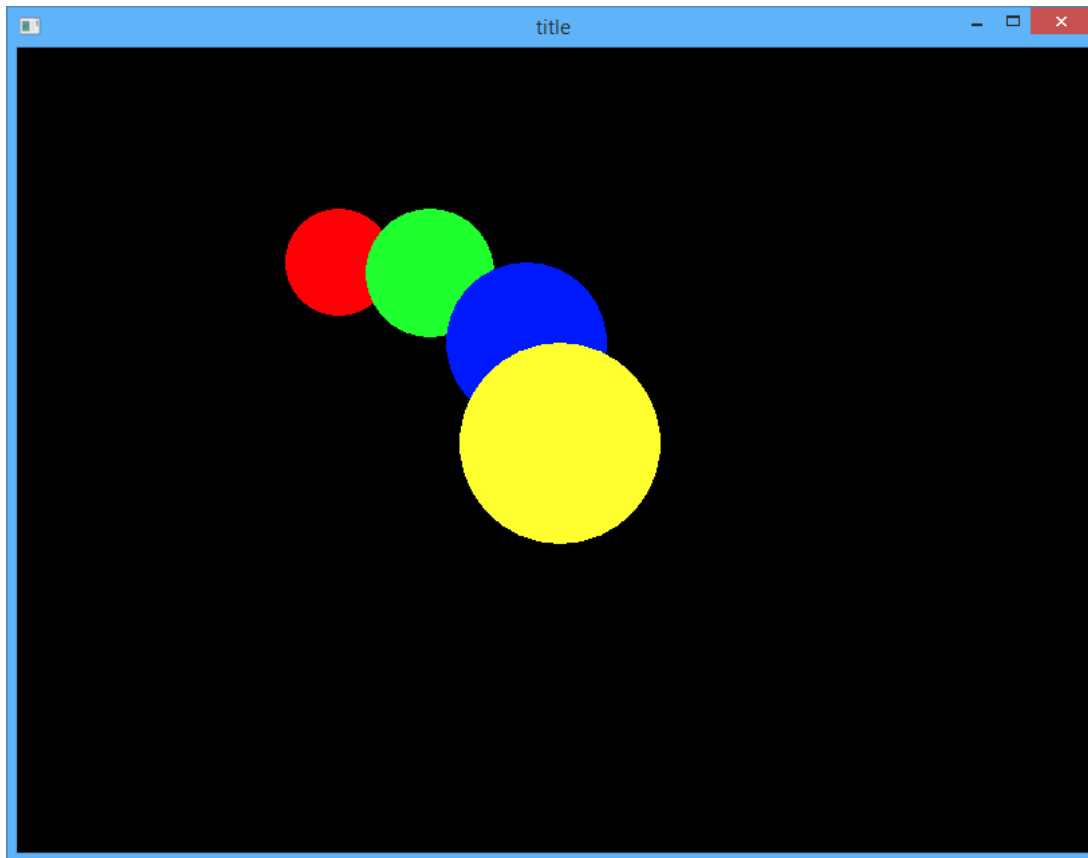
    window.display();

    sf::sleep(sf::seconds(5));
}
```

Запустите CMake, а затем программу:

```
cmake --build .
01\01.exe
```

У вас должно получиться что-то такое:



Освоим прямоугольники и повороты

SFML позволяет рисовать разные типы фигур, в том числе прямоугольники. Прямоугольным фигурам соответствует класс [sf::RectangleShape](#).

Создайте каталог `02`, и в нём создайте файл `CMakeLists.txt` и сохраните текст:

```
add_executable(02 main.cpp)

set(SFML_STATIC_LIBRARIES TRUE)

find_package(Freetype)
find_package(JPEG)
find_package(SFML 2 COMPONENTS window graphics system)

target_compile_features(02 PUBLIC cxx_std_17)
target_compile_definitions(02 PRIVATE SFML_STATIC)

target_link_libraries(02 ${SFML_LIBRARIES} ${SFML_DEPENDENCIES})
```

Теперь создайте в каталоге `01` файл `main.cpp`, и аккуратно перепишите в него код. Посмотрите на код: определите, где создаются два прямоугольника (*rectangle*) и где устанавливается поворот в градусах (*rotation*):

```

#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>

int main()
{
    sf::RenderWindow window(sf::VideoMode({ 800, 600 }), "Rectangles and circle");

    window.clear();

    sf::RectangleShape shape1;
    shape1.setSize({ 60, 20 });
    shape1.setRotation(75);
    shape1.setPosition({ 200, 120 });
    shape1.setFillColor(sf::Color(0xFF, 0x0, 0x0));
    window.draw(shape1);

    sf::CircleShape shape2(48);
    shape2.setPosition({ 260, 120 });
    shape2.setFillColor(sf::Color(0x0, 0xFF, 0x0));
    window.draw(shape2);

    sf::RectangleShape shape3;
    shape3.setSize({ 120, 15 });
    shape3.setRotation(-30);
    shape3.setPosition({ 200, 120 });
    shape3.setFillColor(sf::Color(0xFF, 0x0, 0xFF));
    window.draw(shape3);

    window.display();

    sf::sleep(sf::seconds(5));
}

```

Теперь вернитесь к основному файлу `CMakeLists.txt` и добавьте строку `add_subdirectory(01)`, чтобы он выглядел так:

```

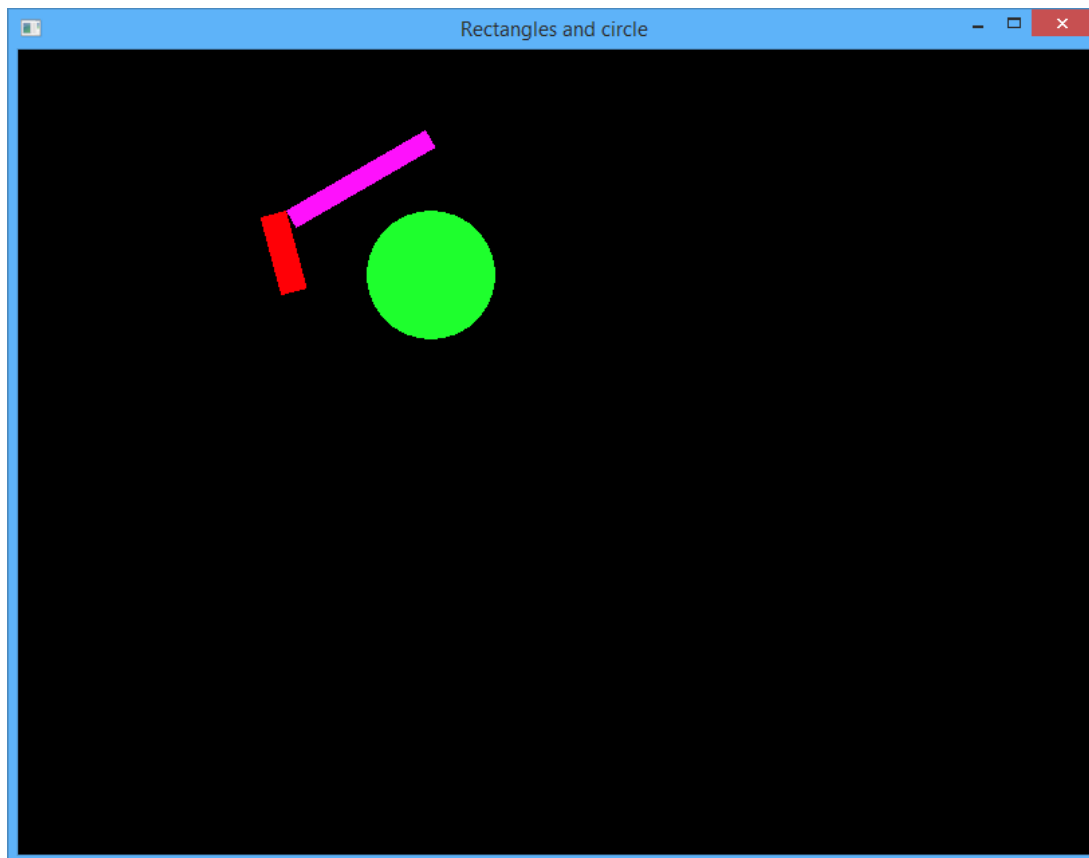
cmake_minimum_required(VERSION 3.8 FATAL_ERROR)

project(cpp-samples)

add_subdirectory(00)
add_subdirectory(01)

```

Соберите через CMake и запустите `02\02.exe`. Результат будет таким:



Теперь вам надо зафиксировать успех! В терминале выполните команды:

```
git init
git add CMakeLists.txt
git add 01/CMakeLists.txt
git add 01/main.cpp
git add 02/CMakeLists.txt
git add 02/main.cpp
git status
git commit -m "Added rectangles example"
```

Введите команду `git log`. Она покажет вам журнал изменений; этот журнал показывает все ранее зафиксированные версии. Выйти из просмотра лога можно с помощью горячей клавиши `Q`.

Задание: нарисуйте фигурами свои инициалы в английской транскрипции. Например, если вас зовут "Fadeev Victor Andreevich", напишите фигурами три буквы: 'FVA'. Перед тем, как написать код, нарисуйте эти буквы на клетчатой бумаге - так вы сможете измерить координаты, на которых должны располагаться буквы, и примерно оценить углы поворота. Нарисованные инициалы должны быть читаемые и аккуратные. Когда вы сделаете задание, зафиксируйте его с помощью git.