ЛАБОРАТОРНАЯ РАБОТА №1 ПО ДИСЦИПЛИНЕ «ПРОГРАММИРОВАНИЕ НА C++». СИСТЕМА СБОРКИ BUILD, TEST AND DEPLOY

Цель работы – получить навыки использования сборочной фермы с использованием технологии разработки Continuous Integration & Continuous Delivery.

Задание

Предварительный этап

1. Создать репозиторий для выполнения лабораторных работ. Это может быть сделано на базе GitLab или GitHub. В рамках выполнения лабораторных работ дисциплины «Программирование на C++» может быть создан один репозиторий на одну лабораторную или на все лабораторные, но тогда очередная ветка предназначена для очередной лабораторной работы.
2. Изучить программный код, представленный в материалах к данной лабораторной работе. Материалы могут быть найдены рядом с файлом данного методического пособия. Необходимо иметь понимание, как работают представленные файлы формата cpp, h, yml.

Основной этап

1. Написать программный код, который печатает в консоли «Hello, World! Version 1.0.*N*», где *N* – номер релизной версии проекта в вашем репозитории.
2. Осуществить с помощью CI/CD автоматизированную сборку проекта как минимум для двух различных операционных систем. Сборка должна происходить на каждый коммит в репозиторий.

Требования к отчёту

Отчёт должен содержать:

1. Титульный лист.

2. Цель работы.

3. Демонстрация результатов проделанной.

4. Вывод.

5. Приложение 1 – Ссылка на репозиторий.

Теоретический материал

Вспомним бегло начало прошлого семестра, где мы знакомились с CMake. Есть команда разработчиков, все работают в разных операционных системах (Windows, Ubuntu, MacOS) и используют различные среды разработки (VS Code, Cline, Visual Studio) и компиляторы C++ к этим средам. Нет такого стандарта, который объясняет, что где как собирать компиляторам, поэтому спокойно может быть такое, что размерность типов данных разнится от компилятора к компилятору. Это может быть важно, когда нам необходимо развернуть приложение там, где критичен объём потребляемой памяти, но мы с такой средой не взаимодействовали в ходе разработки. На помощь приходит инструментарий CMake – эдакий переводчик между операционными системами и различными компиляторами. Именно он предоставляет инструментарий, который позволяет команде разработчиков работать сообща над одним проектом.

Написание любого проекта, приложения начинается с выбора его окружения для разработки, куда входят инструментарии и программы для разработки (например, редактор кода VS Code, компилятор gcc, средство автоматизации сборки проектов CMake). Но если разработка происходит в команде, возникают вопросы по форматированию кода, которые надо решать на берегу: делать отступы через табуляцию или пробелы, знак указателя ближе к типу данных или к имени переменной, как писать комментарии: надо их или не надо вообще, сколько много, если надо, стоит ли писать doxygen-документацию. Как следует использовать и обрабатывать исключения, приватные наследования классов...

Но есть отдельная категория вопросов, стоящих перед командой разработчиков. Как собирать проект: в чём писать код проекта (VS, CLion,

блокнот), каким компилятором собирать, на каком устройстве операционной системе запускать, как проводить и проверять тесты для разных платформ, кто их будет проверять, как обрабатывать артефакты, кто проконтролирует всё это.

Надо понимать, что все мы люди, которые устаём. Нужна железная рука, которая поможет следить за проектом. Рукой этой будет **Continuous Integration & Continuous Delivery** (CI/CD). Это практика разработки, которая предполагает использование непрерывной интеграции (CI) и непрерывной доставки (CD) в повседневной работе. CI/CD автоматически запускает сборку в фиксированном стенде после каждого изменения (коммита) проекта и проведёт тесты, а также любые метрики качества. CI/CD по итогам своей работы предоставляет отчёт и историю. Это позволяет использовать CI/CD для автоматической проверки артефактов, автоматического версионирования, проверки продуктов в нужной заказчику среде, а также автоматического deploy релизов, что облегчает жизнь для разработчиков, менеджеров проекта. Концепция CI представляет из себя дробление больших задач на мелкие и слияние этих мелких задач в основную ветку, CD – разработка ПО короткими итерациями. Пример такого подхода разработки – создание компьютерных игр. Когда игра ещё находится в «раннем доступе», команда разработчиков может представить прототип нового функционала игры в очередном обновлении и получить обратный отзыв от игроков, что следует дальше развивать, а от развития какой концепции следует отказаться.

GitHub предоставляет репозиторий хранилища (систему управления версиями) GitHub Code и автоматизацию рабочих процессов GitHub Actions, бесплатный функционал в рамках индивидуального тарифа будет достаточен в рамках выполнения курса лабораторных работ по «Программированию на C++». В своём git-репозитории добавляем описание технических процессов в определённом файле формата yml, заливаем на GitHub. Тот в свою очередь этот файл видит, он его запускает, присылает e-mail о результатах компиляции и наличии проблем.

Как это нам поможет на практике? Изучим пример применения GitHub Actions в репозитории pcsx2, программный код которого эмулирует PlayStation 2. По ссылке https://github.com/PCSX2/pcsx2/commits/master/ вы можете видеть, как участники разработки вносят коммиты в репозиторий, при этом возле времени загрузки отображается их состояние: пройдены некоторые тесты и какое их количество. На рисунке 1 представлены некоторые коммиты в репозитории.

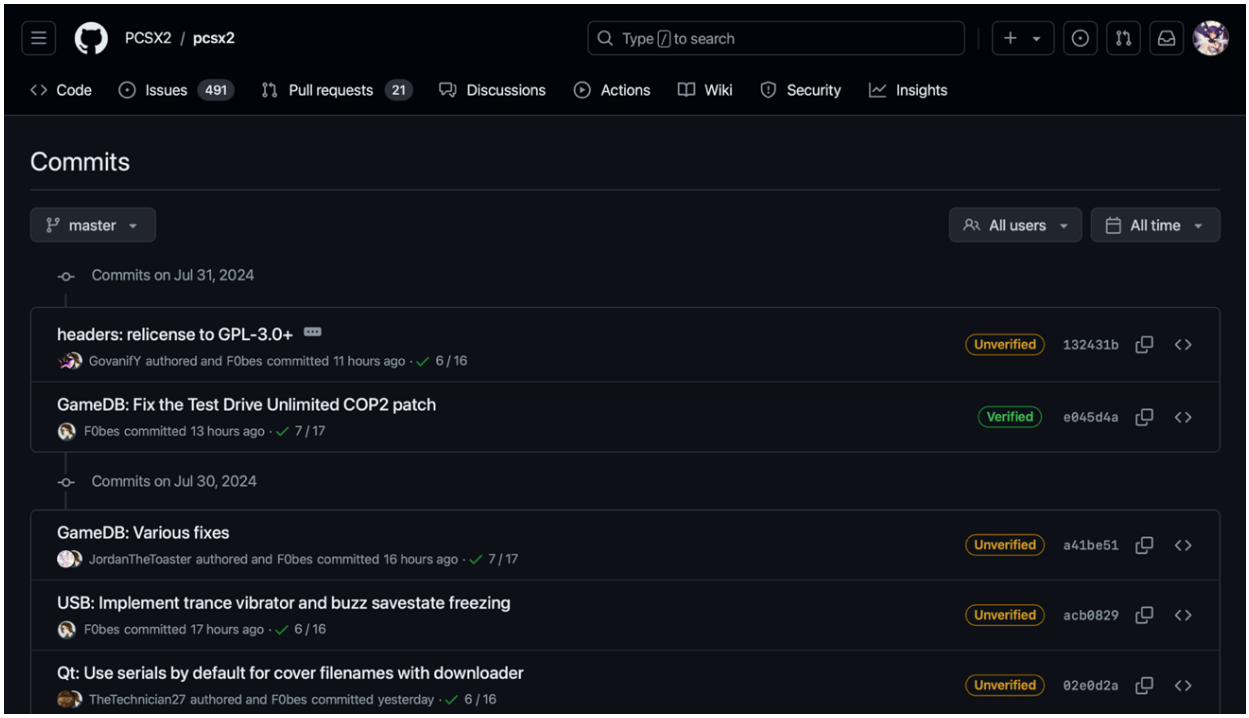

Рисунок 1. Коммиты в репозитории pcsx2

По нажатии на эту информацию открывается окно с информацией о проводимых и пропущенных тестах (см. рисунок 2). Имеется кнопка Details, по нажатии которой отобразится подробная операция о проведённых тестах.

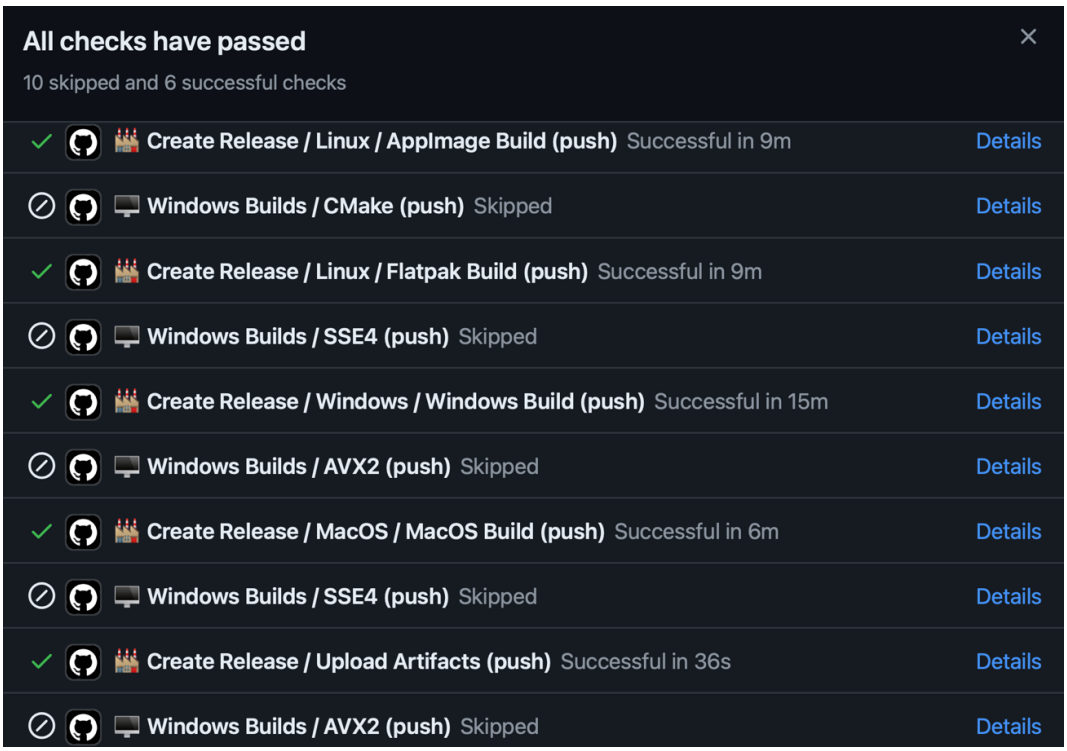

Рисунок 2. Результаты тестирования коммита 132431b репозитория pcsx2

GitHub Actions позволяет создавать исполняемые приложения на основе исходного кода проекта в результате исполнения yml-файла. Так, на рисунке 3 представлены релизы, созданные автоматически с помощью CI/CD. В описании содержится информация, оставленная при создании коммита.

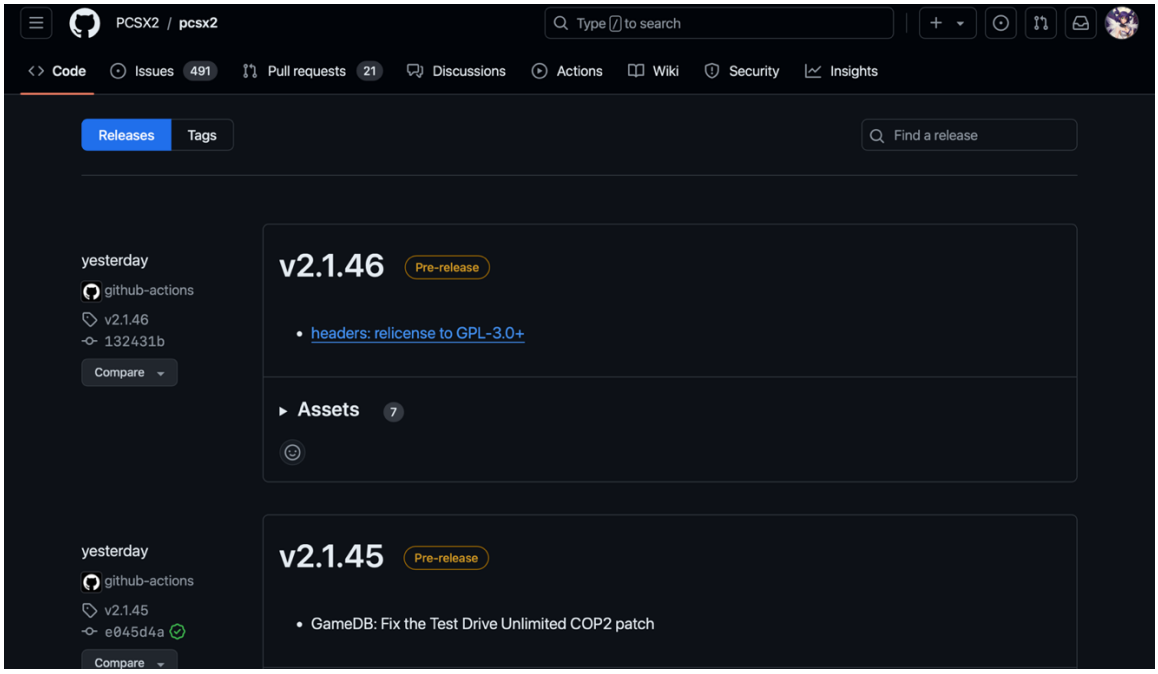

Рисунок 3. Релизы в репозитории pcsx2

Если мы откроем какой-либо из релизов, нам покажется, как на рисунке 4, интерфейс наборов с файлами для установки приложения для конкретного семейства операционных систем (Linux, Mac OS, Windows), а также исходный код приложения для самостоятельной его компиляции.

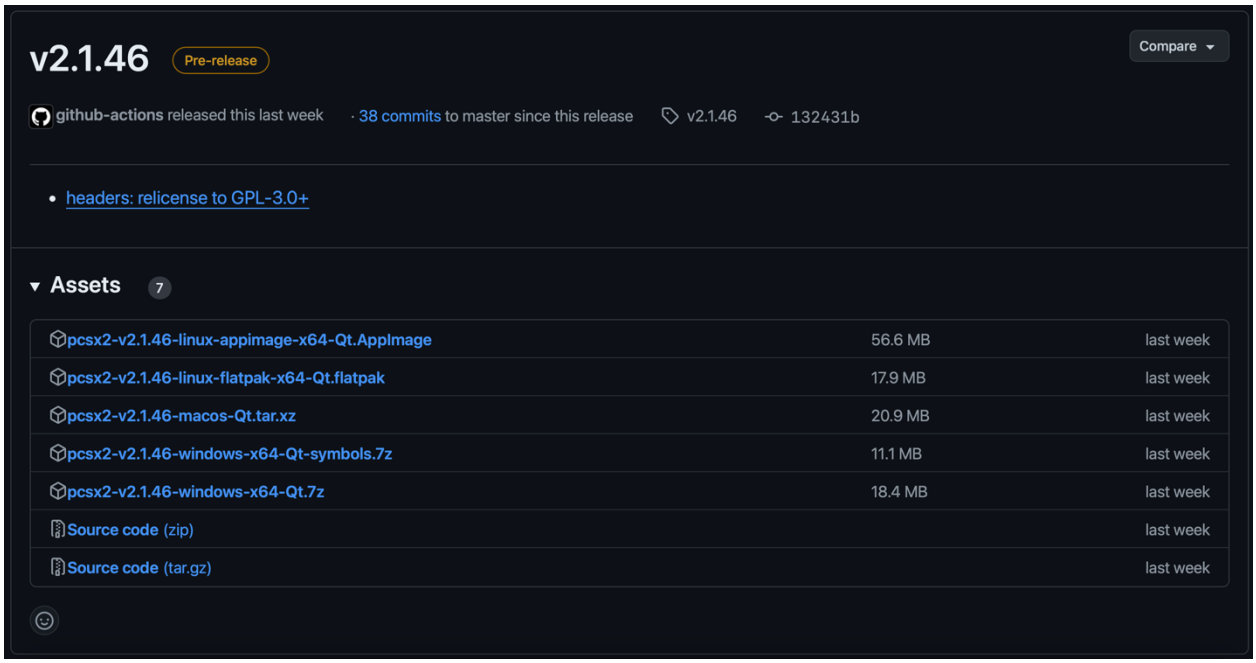

Рисунок 4. Набор архивов для установки pcsx2 на различных операционных системах

Неподалёку от данного методического пособия находится архивированный каталог с программным кодом, который необходимо загрузить на свой репозиторий. В каталоге находятся main.cpp, lib.cpp и lib.h файлы. В этом же каталоге находится .github/workflows/release.yml. По указанному пути GitHub отслеживает действия для запуска, которые прописываются в файле формата yml, при этом неочевидных требований к имени файла нет.

Первым делом нужно решить, как запускать экшн. В секции ‘on’ мы определяем поведение системы: если произошёл push, то активными будут ветки main и feature/github_actions. Во время push будет происходить сборка (build) проекта. В yml-файле можно в параметре runs-on указать систему для

сборки. В рамках выполнения курса лабораторных работ мы будем применять мощности GitHub (информация о возможных конфигурациях: https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions#choosing-github-hosted-runners). Permissions write-all предоставляет Actions права на запись в репозиторий – не забудьте её прописать. Обратите внимание, что следует быть внимательней с GitHub Token, не записывая его в файлах репозитория явно, иначе последуют компрометация доступа к вашему репозиторию. Применение secrets.GITHUB_TOKEN соответствует нормам безопасности.

Применяйте .gitignore или .git/info/exclude, чтобы сообщить информацию о путях файлов, которые создаются на основе программного кода. Стремитесь содержать в репозитории только ту важную часть проекта (программный код), которая позволит скомпилировать ваше приложение.