Домашняя работа №2

Знакомство с C++

Инструментарий и требования к работе

Работа выполняется на C++. На сервере сборка под C++20.

Задание

Напишите реализацию шаблонного класса кватерниона `Quat` с параметром `T` – тип данных для хранения частей числа (гарантируется, что в тестах `T` – вещественный тип).

Необходимо реализовать:

- операторы (список ниже).
- методы (список ниже).
- конструктор из 4 значений типа `T`: `a`, `b`, `c`, `d` (нотация `a + bi + cj + dk`)
- конструктор, создающий кватернион поворота из 3 аргументов: `T` (угол поворота), `bool` (true: угол передан в радианах, иначе: в градусах) и вектора (3 координаты, задающие ось вращения).

Нужно реализовать возможность создания объекта класса `Quat` без аргументов (все компоненты заполняются 0).

В репозитории будет `tests.cpp` для тестирования (google test) и шаблон для класса кватерниона `quat.hpp`. Файл `tests.cpp` менять нельзя. В файле `quat.hpp` нельзя менять выданное, но можно (и нужно) добавлять свои методы/операторы `Quat` и необходимые заголовочные файлы.

Информацию про кватернионы и формальное описание реализации его операций можно посмотреть здесь и здесь. Обратите внимание, что кватернион может быть представлен как комбинация скаляра и вектора из трёх компонентов. Соответственно, операции по кватерниону и скаляру/вектору следует рассматривать с этой точки зрения.

Для отправки на проверку необходимо, чтобы проходили все тесты, кроме тестов `rotation_matrix`, `matrix`, `angle`, `apply`, за которые можно получить дополнительные баллы.

Список операторов:

Оператор	**Аргументы**	**Пояснение**
`+`, `+=`, `-`, `-=`	`Quat`	Сложение/вычитание двух кватернионов (поэлементно)
`*`	`Quat`	Умножение кватернионов
`~`	`-`	Сопряжение
`==`, `!=`	`Quat`	Сравнение на (не)равенство
приведения типа к `T`	`-`	Вычисление модуля кватерниона Примечание: оператор должен быть `explicit`

В дополнение к вышеописанному должно работать умножение на скаляр и вектор (типа `T`). Это может быть реализовано как через перегрузку операторов, так и через дополнительные конструкторы.

Список методов:

Метод	Аргументы	Пояснение
`data`	`-`	Получение указателя на 4 компонента кватерниона в порядке `b`, `c`, `d`, `a` (в другой нотации: `x`, `y`, `z`, `w`).
`rotation_matrix`	`-`	Получение матрицы поворота.
`matrix`	`-`	Получение вещественного матричного представления.
`angle`	`bool`	Нахождение угла кватернионами поворота. Принимает флажок, вернуть значение в градусах или нет, аналогично конструктору из угла.
`apply`	`vector3_t`	Применение кватерниона к вектору. Поворот переданного в аргументах вектора на текущий кватернион.

Матрицы хранятся в порядке row-major.

Обратите внимание, для того, чтобы ваша программа собралась с тестами (даже если вы отжали опцию “запускать тесты с extra functions”), у вас должны быть хоть как-то реализованы все функции (пойдёт даже реализация в виде заглушки, например `return 0;`).

Репозиторий

Если что-то не работает в репозитории и вы не понимаете почему – пишем Виктории или вашему проверяющему (если есть).

Известные проблемы:

Если после первого запуска `BuildTest` вы видите сообщение `"Init repo failed"`, то пишите Виктории с указанием ошибки и ссылкой на репозиторий.

Эта ситуация возникает в случае, если по (пока неведомой нам) причине автоматически не запускается `Init` workflow при создании репозитория (ветки `main`).

В случае, когда после взятия репозитория не было ни одного успешного запуска `Init` (есть только один неуспешный), при первом `BuildTest` запуске произойдет вызов этого самого `Init` workflow и всё заработает (должно). Как это выглядит показано на рисунке 1. Когда хоть один `Init` отработал успешно, то повторно он отрабатывать не будет и все шаги с `Init` будут помечены как `skipped` (рисунок 2).

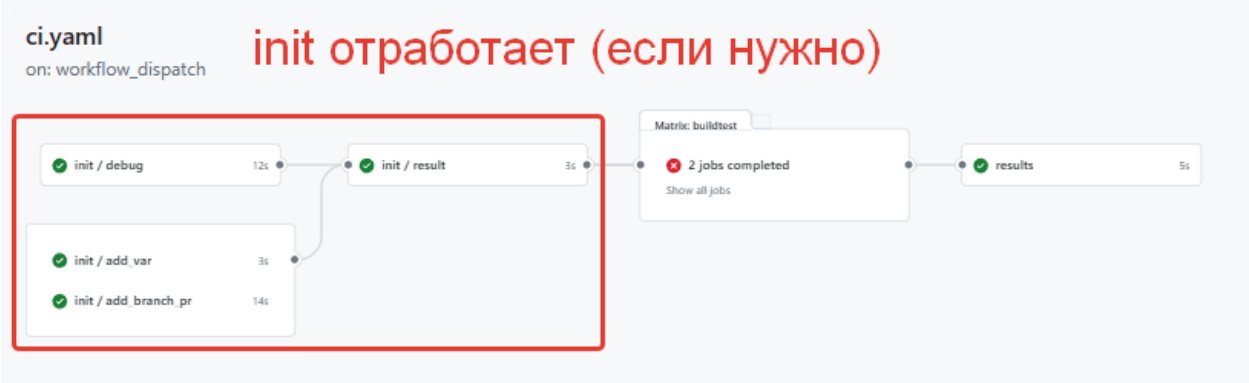

Рисунок 1 – Автоматический запуск `Init` из `BuildTest` при необходимости

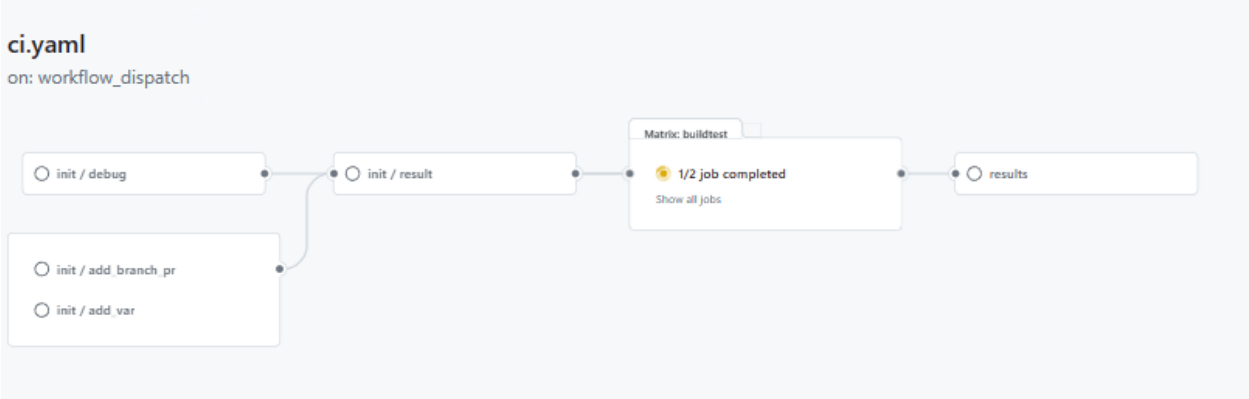

Рисунок 2 – Init из `BuildTest` не запускается, если хоть один `Init` отработал успешно