Задания на практичеческое занятие 4

Пререквизиты

Ввод данных в программу

Все задания подразумевают, что будут реализованы алгоритмы для работы с массивами, а сами массивы будут вводиться в программы со стандартного ввода, остальные данные через параметры командной строки. Для ввода массивов со стандартного ввода удобно использовать функцию `scanf` — функцию форматированного ввода, которая принимает первым параметром строку-формат, а последующими адреса в памяти куда записывать данные, полученные со стандартного ввода. Функция синтаксически похожа на функцию `printf`, но не выводит значения переменных, а наоборот считывает и заносит значения в переменные, при этом преобразуется считанное строковое представление в тип данных, соответствующий указанному в строке-формате спецификатору.

Код программы, который считывает со стандартного ввода массив целых чисел может выглядеть так:

```
#include <stdio.h>

int main(int argc, char** argv) {
    int array[5];
    printf("Введите пять целых чисел:\n");
    for(int i = 0; i < 5; ++i) {
        scanf("%d", &array[i]);
    }
    return 0;
}
```

Обратите внимание на существенное отличие от функции `printf` — оператор `&` перед переменной массива. Этот оператор берёт адрес переменной — номер ячейки памяти с которой начинается память, выделенная под переменную справа от оператора.

При вводе данных с клавиатуры после запуска программы можно каждое значение вводить с новой строки (нажимая Enter после каждого числа) или разделить значения пробелом. Пусть исполняемый файл программы называется `program`:

```
./program
Введите пять целых чисел:
1
2
3
4
5
```

```
./program
Введите пять целых чисел:
1 2 3 4 5
```

Оба способа ввода будут равносильны.

Недостаток интерактивных программ (программ, которые просят пользователя вводить данные во время своей работы) — неудобство ручного тестирования. Для того чтобы постоянно не вводить с клавиатуры значения при тестировании и отладке программы можно воспользоваться оператором конвейера в терминале:

```
echo "1 2 3 4 5" | ./program
```

Оператор конвейера перенаправит вывод команды `echo` на стандартный ввод программе `./program` как будто данные были введены в программу с клавиатуры после запуска.

Порядок работы

Все задания выполняются в репозитории git в котором реализована компиляция и сборка программ, bash-скрипты для тестирования программ и github workflow для автоматического тестирования.

Перед началом работы необходимо создать пустой репозиторий git на github. Затем клонировать репозиторий с заготовками для выполнения заданий:

```
git clone https://github.com/practice-sibsutis/c-programming.git
```

После клонирования перейдите в папку с репозиторием:

```
cd ./c-programming
```

и настройте псевдоним для работы с вашим пустым репозиторием, который был создан в самом начале:

```
git remote add new-origin "путь к репозиторию по ssh"
```

Запуште ветку `main` в ваш репозиторий:

```
git push new-origin main
```

и переключитель на ветку `homework4`:

```
git checkout homework4
```

Запуште её тоже:

```
git push new-origin homework4
```

В ветке есть директория `src` в которой находятся пять файлов с расширением `.c` — файлы в которых следует выполнять задания. Каждое задание в отдельном файле — номер задания соответствует номеру файла в котором его следует выполнять.

Подразумевается, что скомпилированные программы будут находиться в директории `bin`. Поэтому нужно её создать, если её нет:

```
mkdir ./bin
```

а исполняемые файлы будут называться также как и файлы с исходным кодом, только без расширения. Например, `exercise1`. Тогда компиляция будет выглядеть так:

```
gcc -Wall --pedantic ./src/exercise1.c -o ./bin/exercise1
```

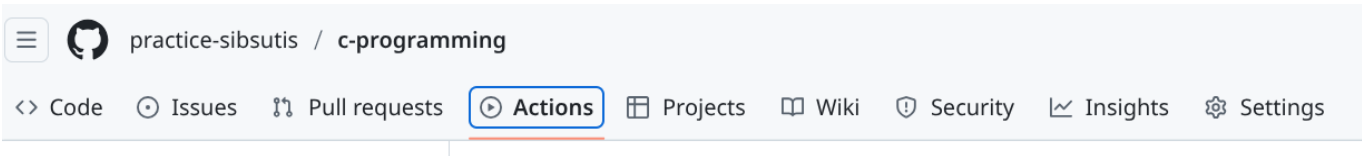
А запуск:

```
./bin/exercise1
```

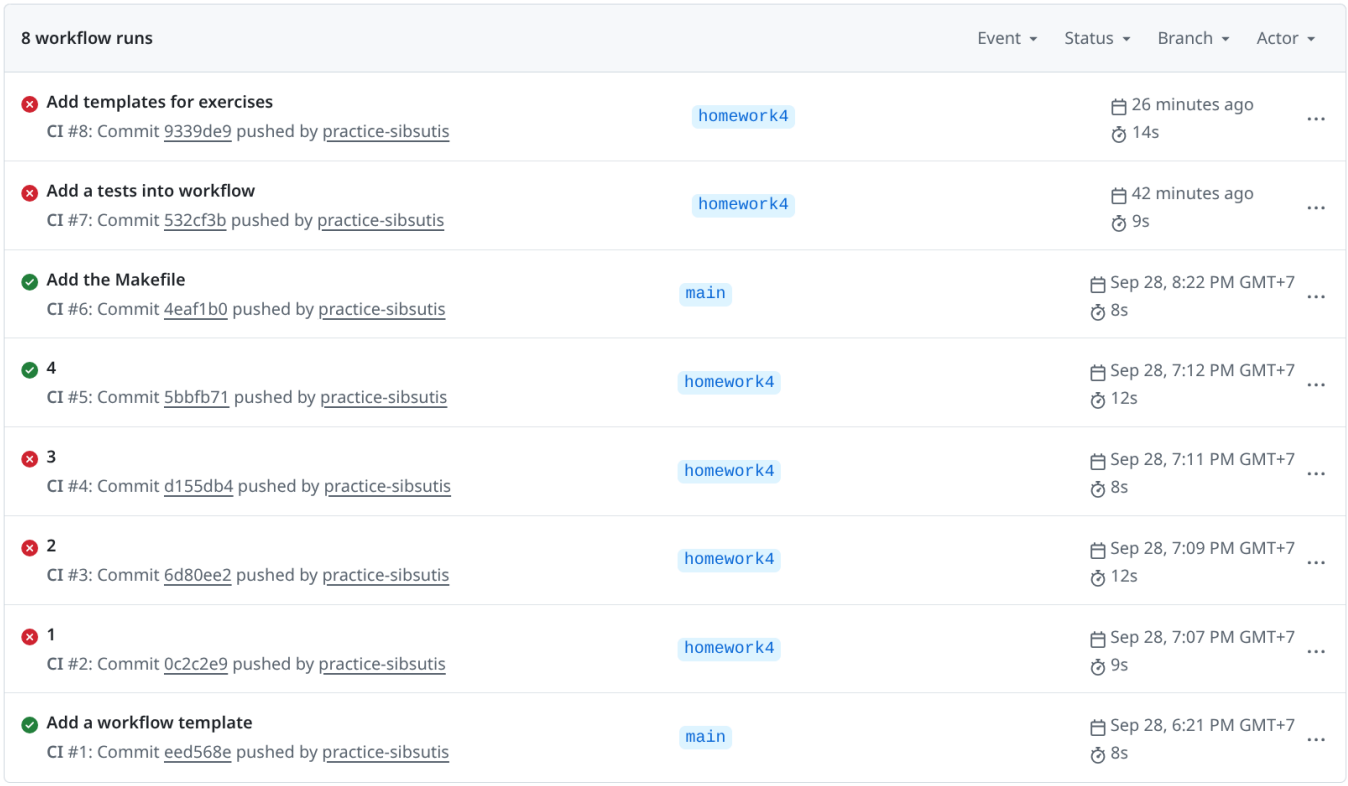
После выполнения заданий, выполните пуш ветки homework4 в ваш репозиторий:

```
git push new-origin homework4
```

и проверьте, что тесты проходят. Для этого перейдите на вкладку `Actions` в репозитории на github:

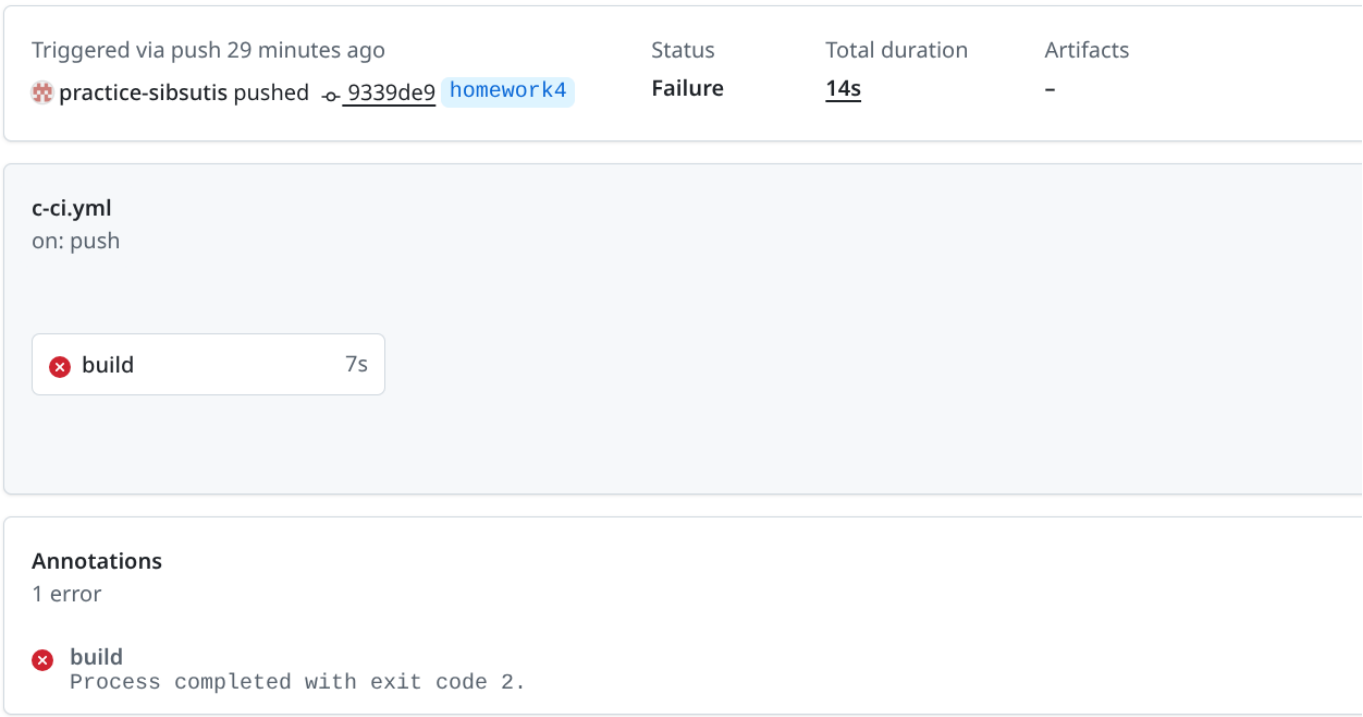


И посмотрите на последний (верхний) коммит в ветку homework4

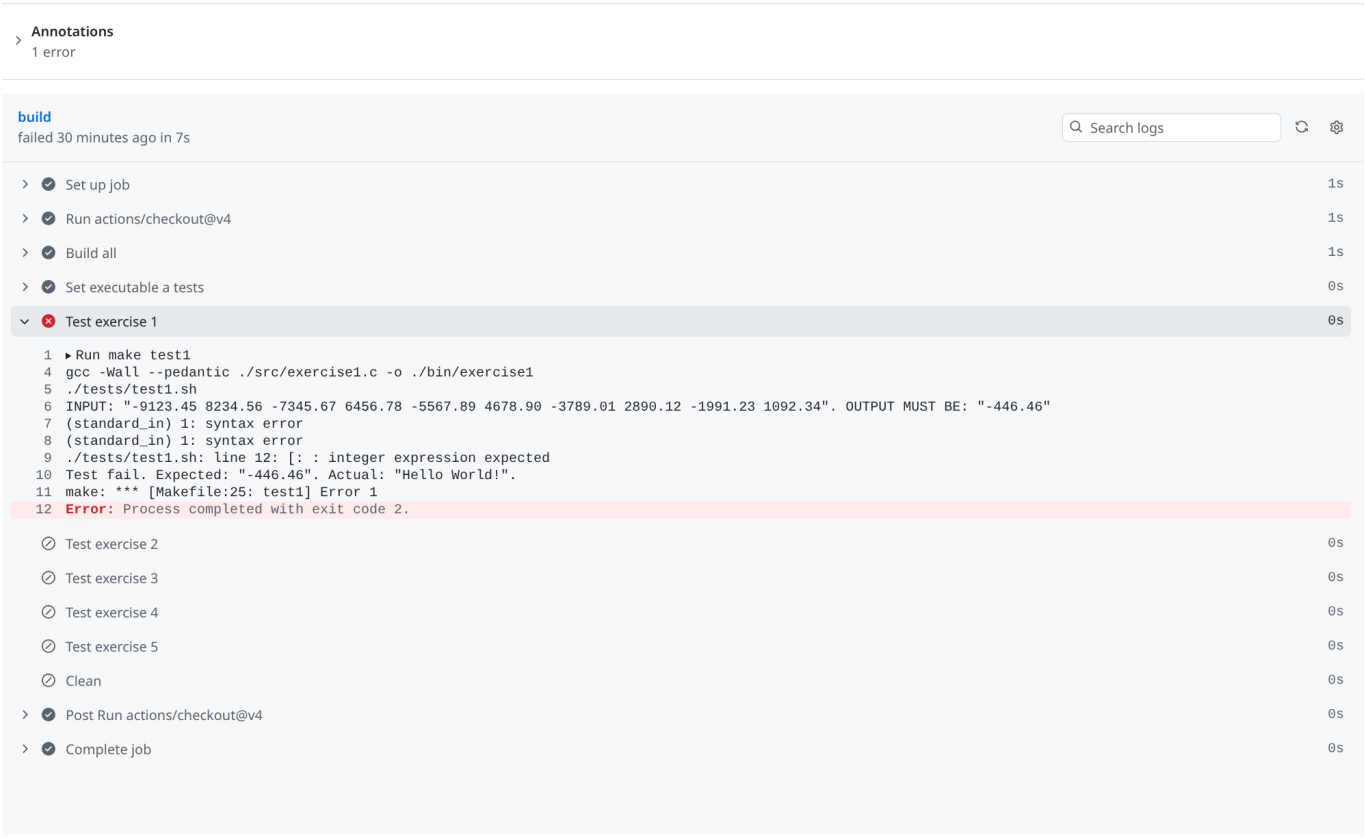


Он должен быть отмечен зелёной галочкой, если все тесты были пройдены успешно. Красным крестиком, если наоборот.

В случае неудачи нажмите на коммит, который не проходит тест:



Нажмите на `build`, чтобы перейти в окно просмотра результата выполнения задач:



Проанализируйте результат выполнения задач, исправьте ошибки, сделайте коммит с исправлениями и запуште ветку homework4 повторно и посмотрите результат.

Задания

1. Дан массив из десяти вещественных чисел. Вывести на стандартный вывод среднее арифметическое (точность — два знака после запятой). Например:

```
echo "1 2 3 4 5 6 7 8 9 10" | ./exercise1
5.5
```

2. Дан массив из десяти вещественных чисел. Вывести на стандартный вывод Инвертированный массив. Например:

```
echo "1 2 3 4 5 6 7 8 9 10" | ./exercise2
10 9 8 7 6 5 4 3 2 1
```

3. Дан массив из десяти вещественных чисел и целое число K. Вывести на стандартный вывод массив циклически сдвинутый на K позиций (если K положительное, то сдвигать вправо, если K отрицательное сдвигать влево). Например:

```
echo "1 2 3 4 5 6 7 8 9 10" | ./exercise3 1
10 1 2 3 4 5 6 7 8 9
```

```
echo "1 2 3 4 5 6 7 8 9 10" | ./exercise3 -1
2 3 4 5 6 7 8 9 10 1
```

4. Дан массив из десяти вещественных чисел и два числа: вещественное X и целое число K. Вывести на стандартный вывод массив, где на позиции K стоит X, а элементы массива начиная с K, сдвинуты вправо, крайний правый элемент при этом исчезает. Например:

```
echo "1 2 3 4 5 6 7 8 9 10" | ./exercise4 1 5
1 2 3 4 5 1 6 7 8 9
```

5. Дано два двумерных массива 10x3 и 3x10, представляющих собой матрицы. Реализуйте алгоритм умножения матриц и выведите результат одной строкой по строкам.