Задания 03

Модульное тестирование

Частью любого проекта является набор тестов, которые запускаются после каждой сборки для проверки корректности кода. Как правило, запуск тестов выполняется автоматически после загрузки кода на центральный сервер, а результаты тестов отображаются на веб-интерфейсе сервера для ознакомления. Если тесты проходят успешно, код отправляется на проверку лидеру команды, и после этого сливается с основной веткой кода. Мы рассмотрим только первый шаг этого процесса — написание и запуск тестов.

Модульные тесты — это обычные программы на C++. Тест считается пройденным успешно, если функция `main` возвращает 0. Тест пропускается, если `main` возвращает 77 (это число досталось в наследство от старых систем). В любом другом случае тест завершается ошибкой.

Для создания тестов доступно большое количество библиотек, одной из которых является Google Test. Файлы с тестами пишутся в декларативном стиле с помощью макросов. Макросы построены на сравнении ожидаемого и актуального значений. В сравнении могут участвовать любые типы, для которых определен оператор сравнения. Пример теста приведен ниже.

```
#include <gtest/gtest.h> // заголовочный файл Google Test
#include <algorithm>
#include <cmath>

TEST(Max, Compare) {
    EXPECT_EQ(10, std::max(10,1)); // ожидается 10
    EXPECT_EQ(1, std::max(0,1)); // ожидается 1
    EXPECT_EQ(1.0/0.0, std::max(1.0/0.0,-1.0/0.0)); // Inf
    EXPECT_TRUE(std::isnan(std::max(0.0/0.0,1.0/0.0))); // NaN
    EXPECT_TRUE(std::isnan(std::max(0.0/0.0,-1.0/0.0))); // NaN
}
```

В тесте производится проверка функции `std::max`. Функция `main` в тестах отсутствует, поскольку предоставляется самой библиотекой `gtest`.

Для того чтобы Meson создал модульный тест, необходимо обернуть команду `executable` в команду `test` и подключить библиотеку `gtest`. Сами тесты, как правило, хранятся в отдельной директории, например, в `src/test` (общая структура директорий описана в предыдущем задании). Поскольку тесты модульные, то каждый из них проверяет работу отдельной функции или класса и их название совпадает с проверяемой функцией или классом. Пример конфигурации приведен ниже.

```
gtest = dependency('gtest', main: true)
test(
    'myclass',
    executable(
        'myclass_test',
        sources: ['myclass_test.cc'],
        include_directories: src,
        dependencies: [gtest]
    )
)
```

Флаг `main` позволяет использовать или не использовать встроенную функцию `main`. После этого все тесты можно запустить командой `meson test` или `ninja test`.

Задания

1. Напишите функцию `message`, которая работает аналогично функции `printf`. Первый аргумент функции — поток, куда будут выводится символы. Второй аргумент функции — шаблон строки, которая отобразится на экране, в котором символом `%` обозначены места вставки объектов. Остальные аргументы функции — это объекты, которые нужно вставить на место `%`. Функция при вызове выводит сообщение на стандартный вывод. Например,

   ```
   message(std::cout, "% + % = %\n", 'a', 2, 3.0)
   ```

   выведет на экран `a + 2 = 3`.

   - Функция должна быть реализована с помощью шаблона с произвольным количеством аргументов.
   - Напишите модульный тест, в котором проверяется корректность работы функции при несовпадении количества аргументов и количества `%`. Для сравнения актуального вывода с ожидаемым удобно использовать поток `std::stringstream`, который осуществляет вывод в строку.

2. Напишите функцию `cat`, которая соединяет произвольное количество массивов в один большой массив. Функция должна принимать произвольное количество аргументов и возвращать объект типа `std::array` (это один из новых классов стандартной библиотеки C++, который представляет собой статический массив заданного размера). Пример вызова функции:

```
std::array<float,3> vec1{1.0f,2.0f,3.0f};
std::array<float,3> vec2{4.0f,5.0f,6.0f};
std::array<float,6> r = cat(vec1, vec2); // 1 2 3 4 5 6
```

- Функция должна быть реализована с помощью шаблона с произвольным количеством аргументов в предположении, что все аргументы имеют тип `std::array<T,N>` (`T` и `N` одинаковы для всех аргументов функции).
- Напишите модульный тест для проверки корректности работы функции.
- Для выполнения задания вам может пригодиться оператор `sizeof...`.

3. Напишите функцию `tie`, которая работает также как `std::tie`, но для массивов, имеющих тип `std::array<T,N>` (реализовывать аналог `std::ignore` не нужно). Функция возвращает объект типа `Tie` (название произвольное), у которого переопределен оператор присваивания:

```
template <class T, int N, int M>
struct Tie {
  void operator=(const std::array<T,N*M>& rhs);
};
```

- Функция должна быть реализована с помощью шаблона с произвольным количеством аргументов в предположении, что все аргументы имеют тип `std::array<T,N>` (`T` и `N` одинаковы для всех аргументов функции).
- Напишите модульный тест для проверки корректности работы функции.
- Для выполнения задания вам может пригодиться класс `std::reference_wrapper`.

Пример вызова функции:

```
std::array<float,6> r{1.0f,2.0f,3.0f,4.0f,5.0f,6.0f}
std::array<float,3> vec1, vec2;
tie(vec1, vec2) = r; // (1 2 3) (4 5 6)
```