Общие требования:

1) Проект на git'e.

2) Наличие интерактивного диалогового интерфейса для проверки корректности разработанной программы.

3) Корректное завершение программы, как в случае штатного выхода, так и в случае невосстановимых ошибок (без утечек и без использования функций мгновенного завершения программы exit, abort, std::terminate и пр.).

4) Логичная и удобная структура проекта, где каждая единица (файл/библиотека) обладает своей единой зоной ответственности (каждый класс в своих файлах .h и .cpp, диалоговые функции и main в своих).

5) Наличие средств автосборки проекта (желательно CMake, qmake и прочие, работающие “поверх” Makefile; использование самописного Makefile нежелательно, но допустимо).

6) Статический анализ кода, встроенный инструментарий в IDE (пр. VS2019: Analyze->Run Code Analysis, см. также Project -> Properties -> Configuration Properties -> Code Analysis -> Microsoft -> Active Rules) или внешние инструменты (Sonarqube + extensions, Clang Static Analyzer и д.р.) (обязательно знакомство с инструментом, исправление всех замечаний или обоснование в комментарии почему конкретное замечание исправить нельзя).

7) Динамический анализ на утечки памяти, встроенный инструментарий в IDE / библиотеки (Пр., VS2019) или внешние инструменты (valgrind, Deleaker и т.п.). Отсутствие утечек памяти и прочих замечаний анализатора.

8) Не "кривой", не избыточный, поддерживаемый и расширяемый код (разумная декомпозиция, DRY, корректное использование заголовочных файлов и т.п.).

9) Стандарт языка C++20 или C++23.

10) Сдача работ проводится только в интегрированной среде разработки (IDE) или редакторе с настроенным LSP.

Порядок выполнения работы:

Реализация. Реализовать указанные в индивидуальном задании функции на языке C. Для проверки функций реализовать простую диалоговую программу, позволяющую вводить значения для обработки функцией и выводить результаты обработки.

Память. Модифицировать программу, заменив все функции управления памятью из языка C (malloc/calloc/realloc/free/strdup/...) на операторы языка C++ (new/delete). Использование функций управления памятью из языка C после выполнения пункта недопустимо.

Исключения. Модифицировать программу, реализовов механизм обработки ошибок при помощи исключений (exception). Использование кодов возврата после выполнения пункта недопустимо.

Перегрузки. Модифицировать программу, добавив перегрузки реализованной функции, указанные в задании. Разработанные перегрузки должны использовать то же имя что и основная функция.

Ссылки. Обеспечить передачу параметров в функции посредствам ссылок (&) или константных ссылок (const &) где это возможно. Передача параметров по указателю или по значению допустимо только при наличии обоснования почему нельзя заменить тип на ссылку.

Ввод-вывод. Модифицировать программу, полностью реализовав ввод-вывод при помощи библиотеки <iostream>. Использование функций ввода-вывода из языка C (stdio) после выполнения пункта недопустимо.

Алгоритмы. Модифицировать программу, реализовав обработку входных данных на основе функций библиотеки <algorithm>. Заменить все циклы внутри функции на примитивы библиотеки алгоритмов. Например, для поиска использовать функцию std::find, для копирования массива — std::copy и т.д.

Примечание: для получения зачёта по лабораторной работе в финальной версии программы должны быть выполнены ВСЕ вышеперечисленные пункты одновременно.

Дополнительные задачи:

Тестирование. Разработать модульные тесты для реализованных функций. Тесты должны покрывать 100% строк указанных в варианте функций (если 100% покрыть невозможно это необходимо обосновать) и не менее 80% кода в целом. Для тестирования использовать фреймворк тестирования Catch2 или GoogleTest.

Документация. Добавить документацию к разработанным функциям в формате doxygen. Выполнить сборку документации.

Open-source. Загрузить разработанную программу в публичный репозиторий на GitHub, GitLab или другой git хостинг. Приложить к проекту подходящую лицензию (LICENSE). Описать проект в README файле.

Вариант 17

Реализовать набор функций для работы с путями файловой системы.

a. Функция join — принимает на вход любой путь в качестве первого аргумента и относительный путь в качестве второго, и возвращает их конкатенацию. Например, join("/tmp", "abc") должен вернуть "/tmp/abc". Если переданный вторым аргументом путь не является относительным — вернуть ошибку.

b. Функция absolute — принимает на вход относительный путь (относительно текущего рабочего каталога) и возвращает абсолютный. Если на вход передан абсолютный путь — вернуть его без изменений.

c. Функция relative — принимает на вход абсолютный путь и возвращает относительный (относительно текущего рабочего каталога). Если на вход передан относительный путь — вернуть его без изменений. Если текущий рабочего каталога не является частью переданного пути, дополнить его необходимым количеством переходов в родительский каталог (..).

d. Функция relativize — принимает на вход два абсолютных пути и возвращает расположение второго пути относительно первого. То есть выполняет операцию аналогичную relative, но работает относительно произвольного пути. Если любой из переданных на вход путей - относительный — вернуть ошибку.

Разделительный символ необходимо выбрать исходя из используемой операционной системы (linux, macos - «/», windows - «\») при помощи макросов (#ifdef __linux__, #ifdef __APPLE__, #ifdef _WIN32 и пр.)

Реализовать 2 перегрузки каждой описанных функций:

1. Принимает на вход строки в виде указателя (const char*).

2. Принимает на вход строки в виде экземпляра std::string.