Лабораторная работа № 1: «Основы программирования на высокоуровневых языках»

Общие требования

1. Проект на git'е.
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 (CMake, qmake и прочие, работающие "поверх" Makefile).
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

Предполагается инкрементальное (поочередное) выполнение пунктов работы для более подробного ознакомления с концепциями высокоуровневых языков. То есть, каждый следующий пункт выполняется поверх предыдущего, а в конце должна получиться программа, удовлетворяющая всем пунктам. Для опытных программистов допускается выполнение всех пунктов сразу, в одной программе.

1. **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.

2. **Память**. Модифицировать программу, заменив все функции управления памятью из языка C (`malloc`/`calloc`/`realloc`/`free`/`strdup`/...) на операторы языка C++ (`new`/`delete`).

3. **Исключения**. Модифицировать программу, реализовав механизм обработки ошибок при помощи исключений (`exception`). Использование кодов возврата недопустимо.

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

5. **Ссылки**. Обеспечить передачу параметров в функции посредствам ссылок (&) или константных ссылок (`const &`) где это целесообразно. Передача параметров по указателю или по значению допустимо только при наличии весомого обоснования, почему нельзя заменить тип на ссылку.
6. **Ввод-вывод**. Модифицировать программу, полностью реализовав ввод-вывод при помощи библиотеки `<iostream>`. Использование функций ввода-вывода из языка C (`stdio`) недопустимо.

Дополнительные задачи:

7. **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.
8. **Ranges (только для самых опытных)**. Вместо стандартных алгоритмов использовать библиотеку `<ranges>` (аналог list comprehension/generator в python, или stream в java). Для выполнения операций использовать алгоритмы над множествами (см. https://en.cppreference.com/w/cpp/language/constraints.html). Для решения непредусмотренных библиотекой задач использовать `std::generator`.

Вариант № 21: «Решатель Кроссвордов»

Реализовать функцию разгадывания кроссвордов.

На вход функции передаётся шаблон искомого слова. В шаблоне искомого слове неизвестные буквы представлены символом «?». В качестве результата вернуть массив слов, подходящих под шаблон. Если в шаблоне присутствуют недопустимые символы — вернуть ошибку. Список всех известных слов допускается разместить в коде или вынести во внешний файл. Опционально — реализовать на русском языке.

Например, если в качестве шаблона передаётся слово «?ppl?», нужно вернуть строки «apple» и «apply» (в зависимости от набора известных программе слов).

Реализовать 2 перегрузки описанной функции:

1. Принимает на вход и возвращает строки в виде указателя (`const char*`).
2. Принимает на вход и возвращает строки в виде экземпляра std::string, для представления массива используется std::vector.

* Тип возвращаемого из каждой перегрузки значения должен совпадать по стилю с типом передаваемого. Если передаётся тип данных из языка C++ (string, vector, …), то и возвращать необходимо тип из C++. Если передаётся сырой тип из C (char*, int*, …), то вернуть также нужно тип из C.