Лабораторная работа № 7

Групповой проект

Цель задания: научиться работать в команде, развить навыки программирования, а также углубить знания в области разработки программного обеспечения.

Каждый участник группы должен создать свой класс с определенной функциональностью, затем необходимо объединить все классы в общий проект.

Защита проекта: проект представляется всеми участниками группы вместе, каждый участник должен рассказать о своем классе и его функциональности. Также каждый участник должен написать и представить документацию по своему классу, в документации должны быть описаны поля класса и разработанные методы. Содержание проекта и требования к проекту приведены в каждом варианте.

Варианты заданий лабораторной работы

Часть А (базовый уровень)

Вариант 1. «**Калькулятор целочисленных матриц**» (на 3 участника)

Описание проекта:

Создание консольного приложения, которое позволяет пользователю выполнять различные операции с матрицами, такие как сложение, вычитание, умножение, транспонирование и нахождение определителя. Каждый участник команды должен создать свой класс с определенной функциональностью, и затем необходимо объединить их в общий проект.

Распределение задач для участников:

Участник1 - Класс «Матрица»:

Реализовать класс «**Матрица»** (**Matrix**), который представляет матрицу (в виде динамического двумерного массива). Класс должен содержать следующие методы: создание матрицы заданного размера (выделение памяти под матрицу), инициализация матрицы (генерация матрицы с помощью случайных чисел из заданного диапазона, считывание матриц из файла, считывание матрицы с клавиатуры, создание нулевой матрицы, создание единичной матрицы), создание копии матрицы, вывод матрицы (на экран и в файл), удаление матрицы из памяти (деструктор). Также реализовать методы проверки: что матрица является нулевой, что матрица является единичной, что матрица является симметричной, что матрица является квадратной.

Участник2 - Класс «Матричные операции»:

Реализовать класс «**Матричные операции»** (**MatrixOperations**), который будет выполнять операции над матрицами, используя предыдущий класс Matrix (Матрица). Класс должен содержать следующие методы: сложение матриц, вычитание матриц, умножение матриц, транспонирование матрицы, нахождения определителя матрицы, нахождение минимального элемента матрицы, нахождение максимального элемента матрицы, домножение матрицы на число, нахождение суммы элементов главной диагонали матриц. Также добавить методы для проверки корректности операций (например, проверка совместимости размеров матриц, проверка на эквивалентность матриц и т.д.).

Участник 3 — «Пользовательский интерфейс» (MenuManager):

Реализовать основной класс приложения, который использует все остальные классы и реализует интерфейс для взаимодействия с пользователем. Класс должен содержать следующие методы: меню пользователя с возможностью выбора операций (создание матрицы, вывод, операции над матрицами и т.д.); обработку пользовательского ввода и вызов выбранных операций и действий над матрицами; обработку ошибок и обратную связь о выполненных операциях.

Итоговый результат:

Каждый участник должен создать свой класс с заданной функциональностью и обеспечить корректную работу класса.

Для каждого класса необходимо сделать заголовочный файл и файл реализации для класса.

По завершении работы необходимо объединить все классы в одном проекте и написать главную функцию, где демонстрируется использование всех классов и их взаимодействие, т.е. пользователи смогут выполнять различные операции с матрицами через консольный интерфейс.

Дополнительные требования:

1. Должна быть реализована документация ко всем классам и их методам, а также инструкция, объясняющая, как использовать приложение.
2. Каждый класс должен быть протестирован на корректность работы и обработку исключительных ситуаций.
3. Код должен быть чистым и хорошо документированным.
4. В проекте разрешается использование алгоритмов и контейнеров стандартной библиотеки STL.
5. Интерфейс для пользователя с меню для выбора операций должен быть простым и интуитивно понятным.
6. Для контроля процесса работы, взаимодействия участников и вклада каждого из авторов в работу используется один из двух вариантов:
   а) Разместить проект GitHub и предоставить доступ своему преподавателю для анализа истории коммитов и вносимых изменений.
   б) Один из участников создает облачный документ (например, Google-документ) и предоставляет остальным участникам и преподавателю доступ к нему. Каждый участник вносит в него свой класс (постепенно или итоговый результат — по договоренности с преподавателем).

Вариант 2. «**Работа с векторами на плоскости**» (на 3 участника)

Описание проекта:

Разработка консольного приложения для работы с векторами на плоскости, которое включает создание, преобразование и операции с векторами. Каждый участник команды должен создать свой класс с определенной функциональностью, и затем необходимо объединить их в общий проект. Предполагается, что вектора имеют вещественные координаты.

Распределение задач для участников:

Участник1 - Класс **«Двумерный вектор»**:

Реализовать класс «**Двумерный вектор»** (**Vector2D**), который будет представлять вектор на плоскости с координатами (x, y). Класс должен содержать следующие методы: конструкторы (по умолчанию, инициализации заданными значениями, генерация вектора с помощью случайных чисел из заданного диапазона, конструктор копирования из другого вектора, считывание вектора из заданного текстового файла, считывание вектора с клавиатуры), методы для выдачи и установки координат вектора, вывод вектора (на экран и в файл), метод для вычисления длины (модуля) вектора, проверка, что верктор нулевой, деструктор.

Участник2 - Класс **«Операции над векторами»**:

Реализовать класс «**Операции над векторами»** (**VectorOperations**), который будет выполнять операции над векторами, используя предыдущий класс Vector2D (Двумерный вектор). Класс должен содержать следующие методы: сложение векторов; вычитание векторов; умножение вектора на заданное число; нормализация вектора (приведение к единичной длине); проверка, что первый вектор больше второго по длине; сравнение, что длины векторов совпадают; вычисление скалярного произведения векторов; вычисления угла между векторами; проверка, что вектора являтся сонаправленными; проверка, что вектора являтся коллинеарными; проверка, что вектора являются равными.

Участник 3 — «Пользовательский интерфейс» (MenuManager):

Реализовать основной класс приложения, который использует все остальные классы и реализует интерфейс для взаимодействия с пользователем. Класс должен содержать следующие методы: меню пользователя с возможностью выбора действий (создание векторов, вывод, операции над векторами и т.д.); обработку пользовательского ввода и вызов выбранных операций и действий над векторами; обработку ошибок и обратную связь о выполненных операциях.

Итоговый результат:

Каждый участник должен создать свой класс с заданной функциональностью и обеспечить корректную работу класса.

Для каждого класса необходимо сделать заголовочный файл и файл реализации для класса.

По завершении работы необходимо объединить все классы в одном проекте и написать главную функцию, где демонстрируется использование всех классов и их взаимодействие, т.е. пользователи смогут выполнять различные действия с векторами через консольный интерфейс.

Дополнительные требования:

1. Должна быть реализована документация ко всем классам и их методам, а также инструкция, объясняющая, как использовать приложение.
2. Каждый класс должен быть протестирован на корректность работы и обработку исключительных ситуаций.
3. Код должен быть чистым и хорошо документированным.
4. В проекте разрешается использование алгоритмов и контейнеров стандартной библиотеки STL.
5. Интерфейс для пользователя с меню для выбора операций должен быть простым и интуитивно понятным.
6. Для контроля процесса работы, взаимодействия участников и вклада каждого из авторов в работу используется один из двух вариантов:

а) Разместить проект GitHub и предоставить доступ своему преподавателю для анализа истории коммитов и вносимых изменений.

б) Один из участников создает облачный документ (например, Google-документ) и предоставляет остальным участникам и преподавателю доступ к нему. Каждый участник вносит в него свой класс (постепенно или итоговый результат — по договоренности с преподавателем).

Вариант 3. «**Система для работы с полиномами**» (на 3 участника)

Описание проекта:

Разработать консольное приложение, которое позволяет пользователям создавать, обрабатывать и анализировать полиномы одной переменной (вида $a_nx^n+a_{n-1}x^{n-1}+...+a_1x+a_0$). Каждый участник команды должен создать свой класс с определенной функциональностью, и затем необходимо объединить их в общий проект.

Распределение задач для участников:

Участник 1 - Класс **«Полином»**:

Реализовать класс **Полином** (**Polynomial**), который будет представлять полином, т.е. хранить его коэффициенты (в виде динамического массива или вектора). Класс должен содержать следующие методы: конструкторы (по умолчанию, инициализации заданными значениями, генерация коэффициентов полинома с помощью случайных чисел из заданного диапазона, конструктор копирования другого полинома, считывание коэффициентов из заданного текстового файла, считывание коэффициентов с клавиатуры), вывод полинома на экран (в виде $a_nx^n+a_{n-1}x^{n-1}+...+a_1x+a_0$), вывод всех коэффициентов на экран, выдачи всех коэффициентов (геттер), задание коэффициентов (сеттер), выдача коэффициента при указанной степени полинома (геттер), задание коэффициента при указанной степени полинома (сеттер), сохранение всех коэффициентов полинома в файл, выдача

степени/порядка полинома (т.е его наивысшей степени), вычисление значения полинома при заданном значении x, деструктор.

Участник 2 - Класс **«Операции над полиномами»**:

Реализовать класс «**Операции над полиномами»** (**PolynomialOperations**), который будет выполнять операции над полиномами, используя предыдущий класс **Polynomial** (Полином). Класс должен содержать следующие методы: сложение двух полиномов; вычитание двух полиномов; умножение двух полиномов; деление двух полиномов (если возможно); вычисление производной полинома, приведение (нормировка) полинома; сравнение, что порядки полиномов равны. Также добавить методы для проверки корректности операций (например, проверка, что степень первого полинома (делимого) больше или равна степени второго, проверка на ненулевые коэффициенты полинома и т.д.).

Участник 3 — «Пользовательский интерфейс» (MenuManager):

Реализовать основной класс приложения, который использует все остальные классы и реализует интерфейс для взаимодействия с пользователем. Класс должен содержать следующие методы: меню пользователя с возможностью выбора действий (создание полиномов, вывод, операции над полиномами и т.д.); обработку пользовательского ввода и вызов выбранных операций и действий над полиномами; обработку ошибок и обратную связь о выполненных операциях.

Итоговый результат:

Каждый участник должен создать свой класс с заданной функциональностью и обеспечить корректную работу класса.

Для каждого класса необходимо сделать заголовочный файл и файл реализации для класса.

По завершении работы необходимо объединить все классы в одном проекте и написать главную функцию, где демонстрируется использование всех

классов и их взаимодействие, т.е. пользователи смогут выполнять различные действия с полиномами через консольный интерфейс.

Дополнительные требования:

1. Должна быть реализована документация ко всем классам и их методам, а также инструкция, объясняющая, как использовать приложение.
2. Каждый класс должен быть протестирован на корректность работы и обработку исключительных ситуаций.
3. Код должен быть чистым и хорошо документированным.
4. В проекте разрешается использование алгоритмов и контейнеров стандартной библиотеки STL.
5. Интерфейс для пользователя с меню для выбора операций должен быть простым и интуитивно понятным.
6. Для контроля процесса работы, взаимодействия участников и вклада каждого из авторов в работу используется один из двух вариантов:
   а) Разместить проект GitHub и предоставить доступ своему преподавателю для анализа истории коммитов и вносимых изменений.
   б) Один из участников создает облачный документ (например, Google-документ) и предоставляет остальным участникам и преподавателю доступ к нему. Каждый участник вносит в него свой класс (постепенно или итоговый результат — по договоренности с преподавателем).

Вариант 4. «Работа с множествами» (на 3 участника)

Описание проекта:

Разработать консольное приложение, которое позволяет пользователю выполнять различные операции над множествами, такие как пересечение, объединение, разность и т.д. Каждый участник команды должен создать свой

класс с определенной функциональностью, и затем необходимо объединить их в общий проект.

Распределение задач для участников:

Участник 1 - Класс **«Множество»**:

Реализовать класс **«Множество»** (**CSet**), который будет представлять множество элементов заданного типа. Предусмотрите возможность работы с множествами различных типов (например, int, float, string и т.д.). Значения во множестве не должны повторяться. Класс должен содержать следующие методы: конструкторы (по умолчанию, инициализации заданными значениями, генерация значений множества заданного размера с помощью случайных чисел из заданного диапазона, конструктор копирования другого множества, считывание множества из заданного текстового файла, заполнение множества с клавиатуры); вывод элементов множества на экран; выдача элемента множества с заданным номером (геттер); задание значения элемента множества с заданным номером (сеттер); выдача мощности множества; сохранение множества в файл; проверка, что множество пусто; проверка, есть ли во множестве элемент с заданным значением; добавление во множество нового элемента (в конец); удаление элемента с заданным значением; деструктор.

Участник 2 - Класс **«Операции над множествами»**:

Реализовать класс «**Операции над множествами»** (SetOperations), который будет выполнять операции над множествами, используя предыдущий класс C**Set** (Множество). Класс должен содержать следующие методы: объединение двух множеств; пересечение двух множеств; разность первого и второго множеств; симметрическая разность двух множеств; дополнение множества; проверка, что мощность первого множества больше мощности второго; проверка, что мощности двух множеств равны; проверка, что первое множество является подмножеством второго. Также добавить методы для проверки корректности операций.

Участник 3 — «Пользовательский интерфейс» (MenuManager):

Реализовать основной класс приложения, который использует все остальные классы и реализует интерфейс для взаимодействия с пользователем. Класс должен содержать следующие методы: меню пользователя с возможностью выбора действий (создание множеств, вывод, операции над множествами и т.д.); обработку пользовательского ввода и вызов выбранных операций и действий над множествами; обработку ошибок и обратную связь о выполненных операциях.

Итоговый результат:

Каждый участник должен создать свой класс с заданной функциональностью и обеспечить корректную работу класса. Для каждого класса необходимо сделать заголовочный файл и файл реализации для класса.

По завершении работы необходимо объединить все классы в одном проекте и написать главную функцию, где демонстрируется использование всех классов и их взаимодействие, т.е. пользователи смогут выполнять различные действия с множествами через консольный интерфейс.

Дополнительные требования:

1. Должна быть реализована документация ко всем классам и их методам, а также инструкция, объясняющая, как использовать приложение.
2. Каждый класс должен быть протестирован на корректность работы и обработку исключительных ситуаций.
3. Код должен быть чистым и хорошо документированным.
4. В проекте разрешается использование алгоритмов и контейнеров стандартной библиотеки STL.
5. Интерфейс для пользователя с меню для выбора операций должен быть простым и интуитивно понятным.
6. Для контроля процесса работы, взаимодействия участников и вклада каждого из авторов в работу используется один из двух вариантов:

а) Разместить проект GitHub и предоставить доступ своему преподавателю для анализа истории коммитов и вносимых изменений.

б) Один из участников создает облачный документ (например, Google-документ) и предоставляет остальным участникам и преподавателю доступ к нему. Каждый участник вносит в него свой класс (постепенно или итоговый результат — по договоренности с преподавателем).

Часть В (уровень повышенной сложности)

Вариант 5. «Игра "Крестики-Нолики"» (на 3 участника):

Разработать консольную версию игры "Крестики-Нолики". Игра должна поддерживать режимы игры для двух игроков и против компьютера. На экране отобразить поле 3x3, в котором отображаются крестики и нолики. Игрок задает клетку поля с помощью ввода номера строки и столбца. Реализовать оптимальную стратегию компьютера и возможность выбора очередности первого хода.

Каждый участник команды должен создать свой класс с определенной функциональностью, и затем необходимо объединить их в общий проект.

Распределение задач для участников:

Участник 1 — Класс **«Игровое поле»** :

Реализовать класс **«**Игровое поле**»** (**Board**), который будет хранить состояние игрового поля. Класс должен содержать следующие методы: конструктор (задание стартового поля); отображение поля; проверки корректности хода в указанное поле (занято поле или нет), проверка выполнения условия победы или ничьей после сделанного хода; деструктор.

Участник 2 - Класс **«Компьютерный игрок»**:

Реализовать класс «**Компьютерный игрок»** (**ComputerPlayer**), который будет представлять компьютерного игрока. Можно сделать два уровня

сложности: первый — начинающий игрок, второй — продвинутый. Класс должен содержать следующие методы: конструктор (по умолчанию, инициализации), алгоритм для выбора хода (для начинающего — случайный ход, для продвинутого — оптимальный алгоритм, например, переборный или минимакс), деструктор.

Участник 3 — Класс «**Игра» (Game)**:

Реализовать основной класс приложения, который управляет ходом игры и использует два предыдущих класса. Также в классе должна храниться информация об игроках (кто играет, имена игроков). Класс должен содержать следующие методы: начало игры, завершение игры, проверка состояния игры (использует метод проверки выполнения условия победы или ничьей из класса **Board**), меню с вариантами выбора режима игры (человек против человека или человек против компьютера), возможностью перезапуститься после окончания игры; метод управления очередностью ходов между игроками; обработка ввода от пользователей (проверяйте, чтобы вводимые координаты были в пределах поля). Обеспечьте информативный вывод (инструкции и сообщения о ходе игры).

Итоговый результат:

Каждый участник должен создать свой класс с заданной функциональностью и обеспечить корректную работу класса.

Для каждого класса необходимо сделать заголовочный файл и файл реализации для класса.

По завершении работы необходимо объединить все классы в одном проекте и написать главную функцию, где демонстрируется использование всех классов и их взаимодействие, т.е. выполняется игра.

Дополнительные требования:

1. Должна быть реализована документация ко всем классам и их методам, а также инструкция, объясняющая, как использовать приложение.

2. Каждый класс должен быть протестирован на корректность работы и обработку исключительных ситуаций.
3. Код должен быть чистым и хорошо документированным.
4. В проекте разрешается использование алгоритмов и контейнеров стандартной библиотеки STL.
5. Интерфейс для пользователя с меню для выбора операций должен быть простым и интуитивно понятным.
6. Для контроля процесса работы, взаимодействия участников и вклада каждого из авторов в работу используется один из двух вариантов:
   а) Разместить проект GitHub и предоставить доступ своему преподавателю для анализа истории коммитов и вносимых изменений.
   б) Один из участников создает облачный документ (например, Google-документ) и предоставляет остальным участникам и преподавателю доступ к нему. Каждый участник вносит в него свой класс (постепенно или итоговый результат — по договоренности с преподавателем).

Вариант 6. «Игра “Города”» (на 4 участника):

Разработать консольное приложение для игры "Города", в которой два игрока по очереди называют города, начинающиеся на последнюю букву предыдущего названного города. Игра должна поддерживать режимы игры для двух игроков и против компьютера. При игре с компьютером реализовать возможность выбора очередности первого хода. Правила игры следующие:

1. Первым элементом цепочки может быть любой город,
2. Каждый следующий начинается с последней буквы предыдущего города (если название заканчивается на ь, ъ, ы - то берётся предпоследняя буква),
3. В цепочке не должно быть двух одинаковых городов.

Каждый участник команды должен создать свой класс с определенной функциональностью, и затем необходимо объединить их в общий проект.

Распределение задач для участников:

Участник 1 - Класс **«Города»** и список возможных городов:

Создать список возможных городов (хранить как поле класса «**Города**» или как отдельный список/вектор строк). Заполнить список из текстового файла.

Реализовать класс **«Города»** (**CityList**), который будет хранить список названных городов и проверять на уникальность. Класс должен содержать следующие методы: конструктор; добавление нового города в список; проверка, был ли город ранее назван (проверка на уникальность); проверка, что добавляемый город реальный (есть в списке возможных городов); выдача последнего названного города; вывод текущего состояния списка городов; деструктор.

Участник 2 - Класс **«Компьютерный игрок»** и Класс **«Игрок»**:

Реализовать класс «**Компьютерный игрок**» (**ComputerPlayer**), который будет представлять компьютерного игрока. Класс должен содержать следующие методы: конструктор, алгоритм выбора города (случайным образом из списка доступных городов или каким-то другим алгоритмом), деструктор.

Реализовать класс «**Игрок**» (**Player**), который будет представлять игрока. Этот класс должен содержать: имя игрока. Класс должен содержать следующие методы: конструктор (по умолчанию, инициализации), выдача имени, задание имени, деструктор. Можно добавить ведение статистики игр для каждого игрока (количество побед, ничьих и т.д.).

Участник 3 — «Интерфейс пользователя»:

Реализовать «**Интерфейс пользователя**» (**UserInterface**), который будет отвечать за взаимодействие с пользователем. Класс должен содержать следующие методы: для отображения текста на экране (например, инструкции, сообщения о ходе игры и вызов списка названных городов как

метода соответствующего класса), ввод данных от игрока (имя и текущий город), вывод меню с вариантами выбора режима игры (человек против человека или человек против компьютера), метод для отображения результатов игры (победитель).

Участник 4 — «Игра» **(Game)**:

Реализовать основной класс приложения, который управляет игровой логикой и использует предыдущие классы. В классе должна храниться информация об игроках (кто играет, имена игроков, сколько сделано ходов). Класс должен содержать следующие методы: начало игры, завершение игры, проверка состояния игры, вызов меню (из предыдущего класса); метод остановки игры; метод управления очередностью ходов между игроками; получения следующей буквы из последнего названного города (вызывает метод соответствующего класса), проверка правильности названного города текущим игроком (начинается ли он на нужную букву, а проверки на повтор и реальность необходимо вызывать как методы класса «**Города**»).

Итоговый результат:

Каждый участник должен создать свой класс с заданной функциональностью и обеспечить корректную работу класса.

Для каждого класса необходимо сделать заголовочный файл и файл реализации для класса.

По завершении работы необходимо объединить все классы в одном проекте и написать главную функцию, где демонстрируется использование всех классов и их взаимодействие, т.е. выполняется игра.

Дополнительные требования:

1. Должна быть реализована документация ко всем классам и их методам, а также инструкция, объясняющая, как использовать приложение.
2. Каждый класс должен быть протестирован на корректность работы и обработку исключительных ситуаций.
3. Код должен быть чистым и хорошо документированным.

4. В проекте разрешается использование алгоритмов и контейнеров стандартной библиотеки STL.
5. Интерфейс для пользователя с меню для выбора операций должен быть простым и интуитивно понятным.
6. Для контроля процесса работы, взаимодействия участников и вклада каждого из авторов в работу используется один из двух вариантов:
   а) Разместить проект GitHub и предоставить доступ своему преподавателю для анализа истории коммитов и вносимых изменений.
   б) Один из участников создает облачный документ (например, Google-документ) и предоставляет остальным участникам и преподавателю доступ к нему. Каждый участник вносит в него свой класс (постепенно или итоговый результат — по договоренности с преподавателем).

Вариант 7. «**Игра "Морской бой"**» (на 3 участника):

Разработать консольную версию игры "Морской бой". Игра должна поддерживать режимы игры для двух игроков и против компьютера. Игрок задает клетку поля с помощью ввода номера строки и столбца. Реализовать оптимальную стратегию компьютерного игрока и возможность выбора очередности первого хода.

Каждый участник команды должен создать свой класс с определенной функциональностью, и затем необходимо объединить их в общий проект.

Распределение задач для участников:

Участник 1 — Классы **«Корабль»** и **«Игровое поле»** :

Реализовать класс **«Корабль»** (**Ship**), который будет определять свойства и методы для кораблей. Поля класса: имя, размер, координаты, направление (вертикальное/горизонтальное). Класс должен содержать следующие методы: конструктор (размер, координаты), проверка состояния корабля

(живой/ранен/потоплен), определение направления и размера (если необходимо); деструктор.

Реализовать класс **«Игровое поле»** (**Board**), который будет хранить информацию о текущем состоянии игрового поля и методы для его визуализации. Класс должен содержать следующие методы: создание и отображение игрового поля; обработка попаданий и промахов; деструктор.

Участник 2 — Класс **«Игрок»**:

Реализовать класс «**Игрок»** (**Player**), который будет представлять игрока. Этот класс должен содержать: имя игрока, человек/компьютер, размещение кораблей (т.е. его версию игрового поля — объект класса Board), версия игрового поля противника. Класс должен содержать следующие методы: конструктор, размещение кораблей на поле (ручное или автоматическое для компьютерного игрока), метод атаки противника (ручной ввод) алгоритм атаки компьютерного игрока (искусственный интеллект), ведение статистики игрока (количество потопленных кораблей и т.д.), деструктор.

Участник 3 — Класс «**Игра» (Game)**:

Реализовать основной класс приложения, который управляет игровой логикой, взаимодействием между игроками и использует два предыдущих класса. Класс должен содержать следующие методы: инициализация игры (установка размеров поля, количество кораблей), хранение информации об игроках (кто играет, очередность), запуск игрового цикла (повтор ходов), обработка ходов игроков, вывод информации игрокам о ходе игры, проверка состояния игры, завершение игры, вывод итоговой информации.

Итоговый результат:

Каждый участник должен создать свой класс с заданной функциональностью и обеспечить корректную работу класса.

Для каждого класса необходимо сделать заголовочный файл и файл реализации для класса.

По завершении работы необходимо объединить все классы в одном проекте и написать главную функцию, где демонстрируется использование всех классов и их взаимодействие, т.е. выполняется игра.

Дополнительные требования:

1. Должна быть реализована документация ко всем классам и их методам, а также инструкция, объясняющая, как использовать приложение.
2. Каждый класс должен быть протестирован на корректность работы и обработку исключительных ситуаций.
3. Код должен быть чистым и хорошо документированным.
4. В проекте разрешается использование алгоритмов и контейнеров стандартной библиотеки STL.
5. Интерфейс для пользователя с меню для выбора операций должен быть простым и интуитивно понятным.
6. Для контроля процесса работы, взаимодействия участников и вклада каждого из авторов в работу используется один из двух вариантов:
   а) Разместить проект GitHub и предоставить доступ своему преподавателю для анализа истории коммитов и вносимых изменений.
   б) Один из участников создает облачный документ (например, Google-документ) и предоставляет остальным участникам и преподавателю доступ к нему. Каждый участник вносит в него свой класс (постепенно или итоговый результат — по договоренности с преподавателем).

Вариант 8. «**Скобочный калькулятор**» (на 4 участника):

Разработать консольное приложение на C++, выполняющее расчет математических выражений с использованием круглых скобок, чисел (в т.ч. и вещественных), знаков математических операторов (+, -, *, /, ^ (степень)) и функций (sin, cos, tan, cotan, exp, ln). В качестве входных данных вводится

символьная строка, содержащая математическое выражение. Программа вычисляет значение введенного выражения и выводит его на экран. В случае если выражение задано неверно, программа выводит сообщение об ошибке.

Каждый участник команды должен создать свой класс с определенной функциональностью, и затем необходимо объединить их в общий проект.

Распределение задач для участников:

Участник 1 — Класс «Допустимые функции»:

Реализовать класс «Допустимые функции» (FunctionsRegistry), который служит для хранения и регистрации математических функций, которые могут быть использованы в выражениях (например, sin, cos, exp и т.д.). Класс должен содержать следующие методы: проверка, что функция относится к числу допустимых; добавление новых функций; вызов функции по имени.

Участник 2 — Класс «Выражение»:

Реализовать класс «Выражение» (Expression), который будет хранить выражение и работать с ним. Выражение необходимо хранить в удобном для обработки формате (например, дерево разбора, ассоциативный массив или постфиксную нотацию). Класс должен содержать следующие методы: конструктор инициализации; вывод выражения на экран; вычисление значения выражения с использованием допустимых операторов и функций (например, сложение, вычитание, умножение, деление, степень, sin, cos, tan и т.д. из предыдущего класса); деструктор.

Участник 3 — Класс «Парсер выражений»:

Реализовать класс «Парсер выражений» (ExpressionParser), который выполняет разбор строкового выражения. Он принимает входную строку, разбирает ее на части и каждую часть заносит в выражение (объект класса Expression). Предусматривает обработку ошибок, например, некорректные выражения или несоответствие скобок.

Участник 4 — Класс «Калькулятор»:

Реализовать класс «Калькулятор» (Calculator), который объединяет все предыдущие классы и предоставляет интерфейс для взаимодействия с

пользователем. Он принимает входные данные (строку, содержащую выражение) от пользователя, передает их в класс ExpressionParser, вызывает полученное выражение (класс Expression) для вычисления результата и выводит его.

Итоговый результат:

Каждый участник должен создать свой класс с заданной функциональностью и обеспечить корректную работу класса. Для каждого класса необходимо сделать заголовочный файл и файл реализации для класса.

По завершении работы необходимо объединить все классы в одном проекте и написать главную функцию, где демонстрируется использование всех классов и их взаимодействие, т.е. реализуется работа консольного скобочного калькулятора.

Дополнительные требования:

1. Должна быть реализована документация ко всем классам и их методам, а также инструкция, объясняющая, как использовать приложение.
2. Каждый класс должен быть протестирован на корректность работы и обработку исключительных ситуаций.
3. Код должен быть чистым и хорошо документированным.
4. В проекте разрешается использование алгоритмов и контейнеров стандартной библиотеки STL.
5. Интерфейс для пользователя с меню для выбора операций должен быть простым и интуитивно понятным.
6. Для контроля процесса работы, взаимодействия участников и вклада каждого из авторов в работу используется один из двух вариантов:
   а) Разместить проект GitHub и предоставить доступ своему преподавателю для анализа истории коммитов и вносимых изменений.
   б) Один из участников создает облачный документ (например, Google-документ) и предоставляет остальным участникам и преподавателю доступ к нему. Каждый участник вносит в него

свой класс (постепенно или итоговый результат — по договоренности с преподавателем).