# Введение

В эпоху цифровизации и быстро развивающихся технологий, глубокие знания в области информатики и программирования становятся неотъемлемым элементом во многих сферах жизни. Одними из основных структур данных, используемых в программировании, являются стеки и двусвязные списки.

Они играют ключевую роль в реализации многих алгоритмов и обеспечивают эффективное управление памятью, что делает их неотъемлемой частью большинства языков программирования. Несмотря на их кажущуюся простоту, понимание их структуры и механизмов работы требует глубоких знаний и практического опыта.

Цель данной работы - глубоко изучить теоретические аспекты стеков и двусвязных списков, а также реализовать программы на языке C++ для демонстрации их работы в реальных условиях.

В данной работе будет представлено две задачи, каждая из которых освещает конкретные аспекты использования стеков и двусвязных списков. Каждая задача будет сопровождена подробным описанием алгоритма решения, структуры программы, а также инструкциями по использованию разработанных программ.

Помимо практического опыта, который будет получен в процессе выполнения этой работы, ожидается, что эта работа также расширит теоретические знания в области стеков и двусвязных списков и демонстрирует их важность в современном программировании.

Основная сложность работы со стеками и двусвязными списками заключается в их динамической природе. Это значит, что они не имеют фиксированного размера, как массивы, и могут расширяться или сжиматься в процессе выполнения программы. Благодаря этому они обладают гибкостью и масштабируемостью, которые критически важны для большинства современных приложений.

Однако, с другой стороны, эта динамическая природа требует от программиста особого внимания при управлении памятью и анализ возможных ошибок. Некорректное использование этих структур данных может привести к серьезным проблемам, таким как утечка памяти или непредсказуемое поведение программы.

Поэтому основной задачей данной курсовой работы является не только реализация стеков и двусвязных списков, но и глубокое понимание их внутренней структуры и принципов работы. Это поможет лучше понять, как эффективно и безопасно использовать эти структуры данных в реальных проектах.

В своей работе я буду использовать язык программирования C++ для реализации стеков и двусвязных списков. C++ является одним из самых популярных и мощных языков программирования, широко используемых в различных областях, начиная от разработки веб-сайтов и заканчивая созданием игр. Он обладает большим набором встроенных функций и удобен для работы со сложными структурами данных, что делает его идеальным выбором для этой работы.

Теперь, когда цели и задачи данной работы были определены, можно приступить к рассмотрению конкретных задач и их решениям.

# Стеки

## 1.1 Постановка задачи

Для того, чтобы показать преимущества и недостатки структур данных, как стеков, нашей задачей будет написание программы на C++, которая будет использовать стек для преобразования выражения из инфиксной записи в постфиксную (обратную польскую) запись.

Кроме того, программа должна обеспечивать проверку корректности выражения и корректность использования скобок. Входные данные для программы - строка, содержащая математическое выражение в инфиксной записи. Программа должна выводить строку, содержащую выражение в постфиксной записи и результат вычислений.

Для решения этой задачи необходимо реализовать стек и связанные с ним операции, такие как push (добавление элемента на вершину стека), pop (удаление элемента с вершины стека) и top (получение элемента с вершины стека). Также необходимо реализовать алгоритм преобразования инфиксной записи в постфиксную, который состоит из прохода по всем символам выражения и применения правил приоритета операций и скобок.

Важным аспектом решения задачи также является обработка ошибок, связанных с некорректным использованием скобок и операций, таких как деление на ноль и корень из отрицательного числа. Для этого необходимо реализовать соответствующие проверки и сообщения об ошибке.

Таким образом, решение данной задачи позволит более глубоко понять принципы работы стека и преобразование выражений, а также научиться использовать их на практике.

## 1.2 Описание входной и выходной информации

Входные данные для нашей программы представляют собой строку, которая символизирует арифметическое выражение в инфиксной нотации.

**Инфиксная нотация** — это метод записи математических и логических формул, в котором операторы располагаются между операндами, например: "2 + 2". Важно отметить, что мы ожидаем, что входная строка не содержит пробелов между символами.

Наша строка может включать следующие элементы:

* **Цифры (0-9)**: Эти символы представляют числовые значения, используемые в выражении.
* **Основные арифметические операторы (+, -, \*, /)**: Эти символы представляют операции, которые нужно выполнить с числовыми значениями.
* **Круглые скобки ( и )**: Эти символы используются для управления приоритетом операций в выражении.

Выходные данные для нашей программы — это строка, которая представляет то же арифметическое выражение, что и входная строка, но записанное в постфиксной (обратной польской) нотации.

**Постфиксная нотация** - это метод записи математических и логических формул, где операторы следуют после их операндов. Например, инфиксное выражение "2 + 2" будет записано в постфиксной нотации как "2 2 +".

Например, для входных данных "3+4*5" выходными данными будут "345*+".

Использование входных данных, включающих в себя элементы как круглые скобки, так и арифметические операции, вызывает необходимость учесть приоритет операций. В классической инфиксной нотации умножение и деление имеют больший приоритет, чем сложение и вычитание, поэтому "3+4*5"* должно быть вычислено как "3+(45)", а не "(3+4)5". Однако в постфиксной нотации приоритет операций определяется их положением относительно операндов, поэтому входная строка "3+45" корректно преобразуется в "345\*+".

Важно отметить, что программа должна корректно обрабатывать ситуации, когда во входной строке присутствуют скобки. Скобки в инфиксной нотации используются для изменения стандартного порядка выполнения операций, а в постфиксной нотации скобки не используются. Поэтому входная строка "3\*(4+5)" должна быть преобразована в "345+\*".

Помимо основной функции перевода инфиксного выражения в постфиксное, программа также должна быть способна обрабатывать некорректный ввод. В случае, если входная строка содержит символы, не представляющие числа, арифметические операторы или скобки, или если скобки в строке не сбалансированы, программа должна сообщить об ошибке. Так, входная строка "3+(4\*5" является некорректной из-за отсутствующей закрывающей скобки.

Следовательно, на основе описания входной и выходной информации можно сформулировать, что целью данной программы является обеспечение корректного и эффективного преобразования арифметических выражений из инфиксной формы записи в постфиксную форму.

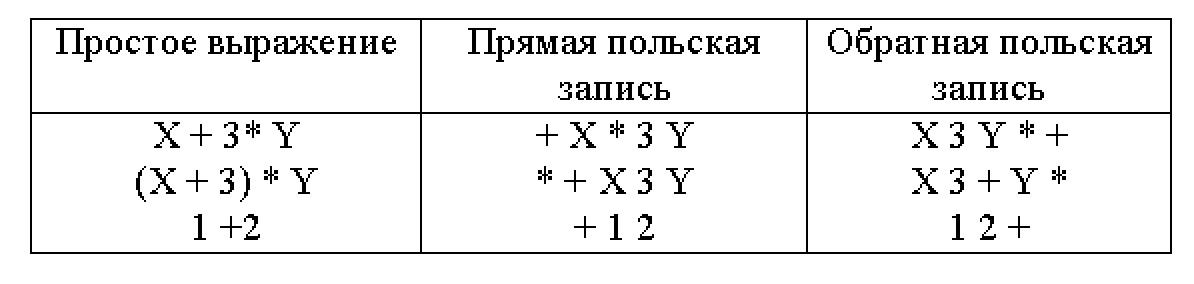


Рис. 1 – Пример прямой и обратной польской нотации

## 1.3 Алгоритм решения задачи

***1.3.1 Алгоритм работы стека***

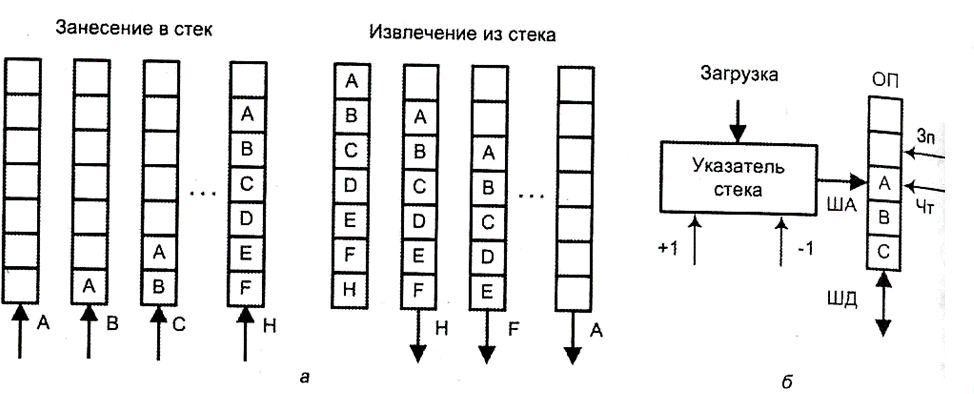
Стек — это тип данных, организованный по принципу "последний вошел, первый вышел" (LIFO - Last In, First Out). Основные операции, которые можно выполнять со стеком, это "push" (добавить элемент в стек) и "pop" (извлечь элемент из стека).

Вот общий алгоритм работы со стеком:

1. Создание пустого стека.
2. Добавление элемента в стек (операция push): новый элемент размещается на вершине стека. Все предыдущие элементы сдвигаются вниз по стеку.
3. Извлечение элемента из стека (операция pop): элемент, который находится на вершине стека, удаляется из него. Все оставшиеся элементы сдвигаются вверх по стеку.
4. Проверка стека на пустоту: если в стеке нет элементов, то он считается пустым.
5. Чтение вершины стека (операция top/peek): получение элемента с вершины стека без его удаления.

В зависимости от конкретной задачи могут быть добавлены и другие операции. Например, можно предусмотреть операцию очистки стека (удаления всех элементов), проверки размера стека и т.д.

Стек можно реализовать разными способами. Одним из наиболее популярных является реализация с использованием динамического массива или связного списка. Выбор конкретной реализации зависит от требований задачи и особенностей используемого языка программирования.

 Рис. 2 – Графическое представление стековой модели памяти

Преобразование инфиксного выражения в постфиксное проводится с помощью следующего алгоритма.

***1.3.2*** ***Алгоритм преобразования инфиксного выражения в постфиксное***

1. Создается пустой стек для хранения операторов. В начале работы алгоритма стек пуст.
2. Символы входной строки рассматриваются по одному. Если символ является числом, он сразу же добавляется в выходную строку.
3. Если символ является открывающей скобкой, он помещается в стек.
4. Если символ является закрывающей скобкой, то символы извлекаются из стека и добавляются в выходную строку до тех пор, пока не встретится открывающая скобка (она удаляется из стека, но в выходную строку не добавляется).
5. Если символ является оператором (O1), а на вершине стека находится другой оператор (O2), тогда происходит следующее:
   * + - если оператор O1 имеет меньший или равный приоритет, чем O2, то O2 извлекается из стека и добавляется в выходную строку. Этот процесс повторяется до тех пор, пока на вершине стека не окажется оператор с меньшим приоритетом, чем у O1, или пока стек не станет пустым. После этого O1 помещается в стек.
6. если оператор O1 имеет больший приоритет, чем O2, то O1 помещается в стек поверх O2.
7. После того как все символы входной строки были рассмотрены, операторы извлекаются из стека и добавляются в выходную строку до тех пор, пока стек не станет пустым.

Этот алгоритм обеспечивает корректное преобразование инфиксного выражения в постфиксное, учитывая приоритет операций и скобок.

Блок схема-алгоритма приводиться ниже:

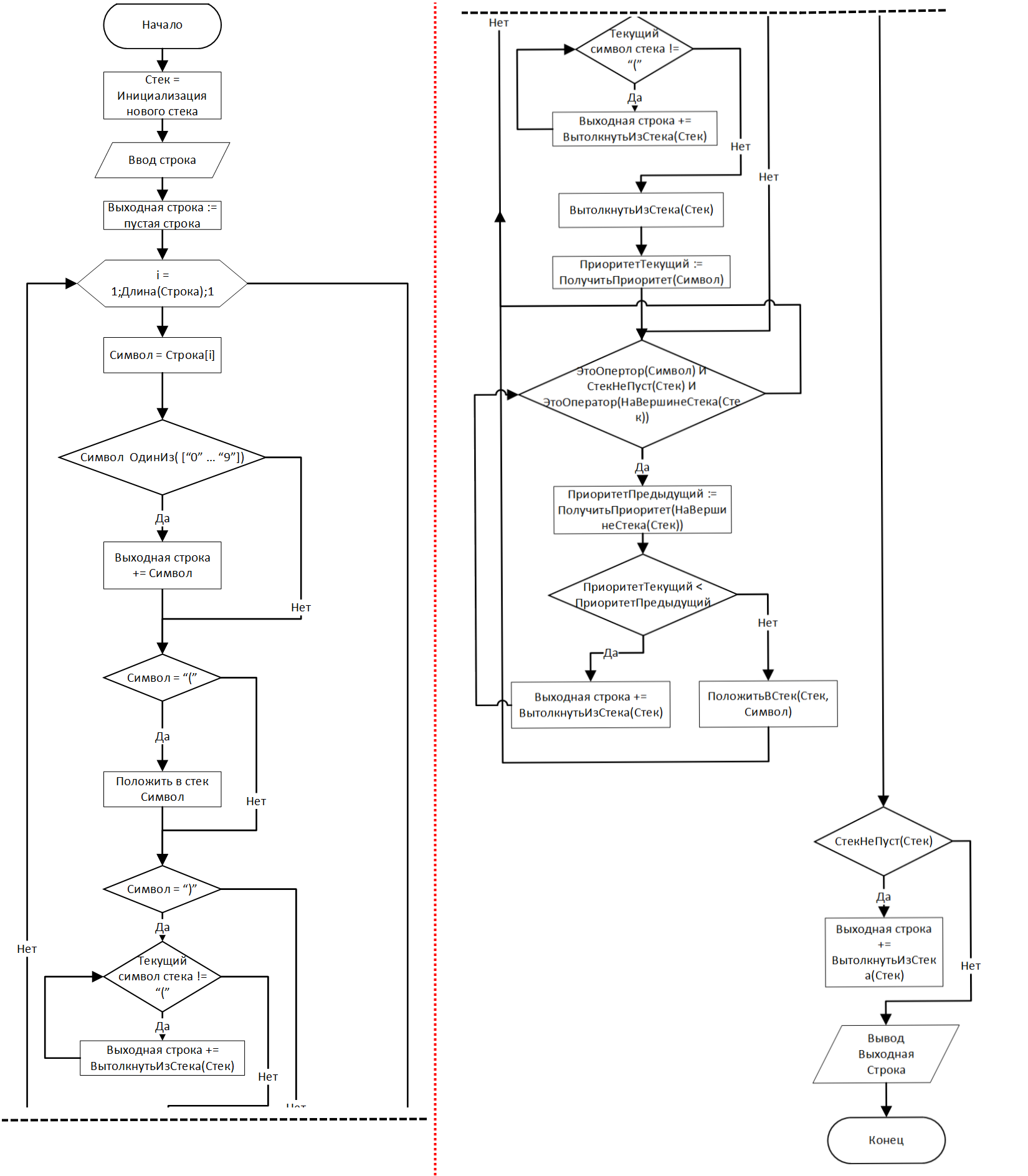


Рис.3 - Блок-схема алгоритма перевода в обратную польскую нотацию

* + 1. ***Иллюстрация работы алгоритма преобразования инфиксного выражения в постфиксное***

Чтобы лучше понять работу алгоритма, приведем пример:

Рассмотрим инфиксное выражение **(1 + 2) \* (3 + 4)**. Его постфиксная форма будет выглядеть следующим образом: **1 2 + 3 4 + \***.

Разберем подробно процесс преобразования:

1. Сканируем символы входного выражения слева направо. Сначала встречается символ "(". Он помещается в стек.
2. Далее идет число 1. Оно сразу записывается в выходную строку.
3. Следующий символ - "+". Он имеет больший приоритет, чем "(", поэтому помещается в стек.
4. Следующий символ - число 2. Оно сразу записывается в выходную строку.
5. Следующий символ - ")". В этом случае из стека извлекаются все символы до первой открывающей скобки (она удаляется, но в выходную строку не записывается). Таким образом, из стека извлекается и записывается в выходную строку "+".
6. Аналогичные шаги повторяются для второй части выражения "(3 + 4)".
7. В конце встречается символ "\*". Так как стек уже пуст, он просто помещается в стек.
8. После того как все символы входной строки были обработаны, оператор "\*" извлекается из стека и записывается в выходную строку.

Корректность данного алгоритма обусловлена свойствами операций, выполняемых над стеком. Действительно, стек позволяет обеспечить соблюдение порядка выполнения операций согласно их приоритету: оператор с большим приоритетом выполняется раньше оператора с меньшим приоритетом, даже если последний был прочитан раньше.

В целом, применение стека в данной задаче позволяет нам преобразовать инфиксное выражение в постфиксное с помощью довольно простого и понятного алгоритма.

## Общие требования к программе

Программа, разработанная в рамках этого проекта, должна быть реализована на языке C++. Это обусловлено несколькими причинами: во-первых, C++ предлагает мощные возможности для работы со структурами данных, включая стеки; во-вторых, большинство учебных программ в области информатики и программирования акцентируют внимание на этом языке.

Программа должна быть написана в соответствии с принципами структурного программирования. Это подразумевает разбиение кода на малые, независимые части или функции. Каждая функция должна выполнять конкретную задачу и возвращать конкретный результат. Использование глобальных переменных следует свести к минимуму.

Программа должна быть кроссплатформенной и работать без изменений на различных операционных системах, включая Windows, Linux и MacOS. Это можно достичь с помощью стандартной библиотеки C++, которая обеспечивает универсальный доступ к базовым функциям операционной системы.

Также к программе предъявляются следующие требования:

* + Программа должна корректно обрабатывать все возможные входные данные, включая некорректные и граничные случаи.
  + Программа должна предоставлять информативные сообщения об ошибках. Если пользователь вводит некорректные данные, программа должна сообщить об этом и предложить повторить ввод.
  + Программа должна иметь простой и понятный интерфейс. Любой пользователь, даже не обладающий глубокими знаниями в области информатики, должен суметь понять, как работает программа, и какие действия нужно выполнять.
  + Программа должна быть эффективной с точки зрения использования ресурсов. Она не должна требовать большого объема оперативной памяти или процессорного времени для выполнения своих функций.
  + Код программы должен быть хорошо документирован. Комментарии должны быть написаны для каждой функции, объясняющие, что эта функция делает и какие аргументы она принимает.
  + Модульность: Программа должна быть разделена на отдельные модули или классы, каждый из которых выполняет свою уникальную функцию. Это облегчит тестирование, отладку и будущую модификацию программы.
  + Расширяемость: Дизайн программы должен быть гибким и позволять легко добавлять новые функции или изменять существующие, не нарушая при этом работу остальных частей программы.
  + Робастность: Программа должна быть устойчива к ошибкам. При возникновении ошибки, программа должна вести себя предсказуемо, сообщать об ошибке и не прерывать свою работу, если это возможно.
  + Использование стандартных библиотек: Для реализации базовых функций (например, ввода/вывода, работы с файлами, математических операций) следует использовать стандартные библиотеки языка C++. Это обеспечит совместимость кода с различными системами и упростит его понимание и поддержку.
  + Код программы должен быть написан в соответствии с общепринятыми стандартами кодирования и стиля кода для языка C++. Это поможет гарантировать, что код будет понятен и читаем для других разработчиков.

## Описание структуры программы для решения задачи

Приложение, разработанное для решения этой задачи, основывается на концепциях функционального программирования и структур данных в языке C++. Оно разбито на несколько модулей для обеспечения чистоты и модульности кода. Структура программы выглядит следующим образом:

***Структура программы для решения задачи с использованием стека включает следующие компоненты:***

1. Главная функция (main): это точка входа в программу. В главной функции происходит создание объекта CustomStack, который представляет собой реализацию стека. Затем происходит вызов функций для тестирования стека и конвертации выражений.
2. Модуль Stack: В этом модуле содержится реализация стека. Он состоит из структуры Stack, которая представляет собой структуру данных для хранения элементов стека, и набора функций для работы со стеком, таких как push (добавление элемента в стек), pop (извлечение элемента из стека), isEmpty (проверка, пуст ли стек) и peek (получение верхнего элемента стека без его удаления).
3. Модуль Postfix: В этом модуле содержится функция для конвертации инфиксного выражения в постфиксное. Она использует стек для хранения операторов и операндов при обработке выражения. Функция принимает инфиксное выражение в виде строки и возвращает его эквивалент в постфиксной нотации.
4. Модуль TestStack: В этом модуле содержится функция для тестирования стека. Она генерирует случайное количество элементов от 1 до 100 и добавляет их в стек. Затем последовательно извлекает элементы из стека и выводит их на экран. В конце проверяет, пуст ли стек, и выводит результат на экран.
5. Модуль TestPostFix: В этом модуле содержится функция для тестирования конвертации выражений. Она запрашивает у пользователя ввод инфиксного выражения, затем вызывает функцию из модуля Postfix для конвертации его в постфиксную нотацию. Полученное постфиксное выражение выводится на экран.

Структура программы разделена на модули, каждый из которых отвечает за определенную функциональность. Это позволяет разделить задачу на более мелкие части и обеспечить более гибкое управление кодом. Каждый модуль имеет свою сферу ответственности и предоставляет набор функций для взаимодействия с другими модулями.

## Инструкции по эксплуатации программы

Компонент программы для тестирования стека и конвертации выражений представляет собой диалоговое консольное приложение, разработанное в рамках проекта C++ с использованием CMake для сборки под платформу Windows.

Пользователю предлагается выбрать один из двух режимов:

* TestStack - тестирование стека
* TestPostFix - тестирование конвертации между инфиксной и постфиксной нотацией

При выборе режима TestStack происходит следующий сценарий:

1. Генерируется случайное количество элементов от 1 до 100.
2. Сгенерированные числа из натурального ряда добавляются в стек.
3. Каждый элемент последовательно извлекается из стека и выводится на экран.
4. По завершении операций стек проверяется на пустоту, и результат проверки стека на кол-во элементов (пустой или не пустой) выводится на экран.

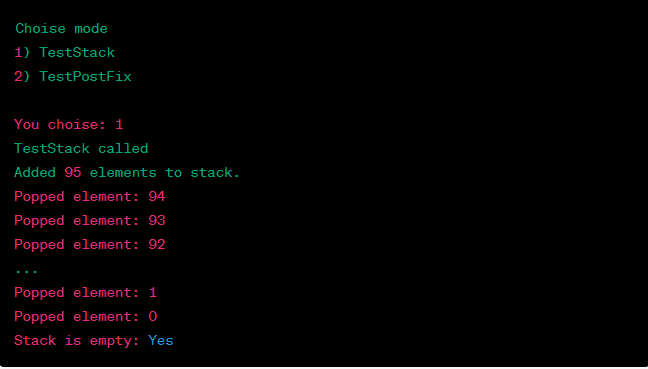


Рис. 4 Пример работы режима TestStack

При выборе режима TestPostFix происходит следующий сценарий:

1. Пользователю предлагается ввести инфиксное арифметическое выражение.
2. Введенное выражение конвертируется в постфиксную нотацию.
3. Постфиксное выражение выводится на экран.

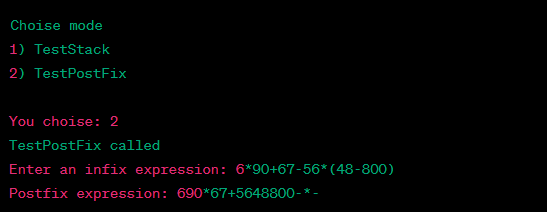


Рис. 5 Пример работы режима TestPostFix

***Сборка программы под Linux с использованием CMake***

Откройте командную строку или терминал и перейдите в каталог, содержащий ваш файл CMakeLists.txt.

Выполните следующие команды для сборки программы:

**mkdir build**

**cd build**

**cmake ..**

**make**

Команда cmake .. сканирует ваш файл CMakeLists.txt и создает файлы для сборки проекта, а команда make компилирует и собирает исходные файлы в исполняемый файл.

После успешной сборки программы вы можете найти исполняемый файл в каталоге build. Запустите его для выполнения вашей программы.

Используя файл CMakeLists.txt, вы можете управлять процессом сборки, добавлять новые файлы и модули, устанавливать опции компиляции и настраивать другие параметры сборки вашего проекта.

## Описание контрольного примера

Для режима TestPostFix нам требуется предоставить несколько наборов входных данных и ожидаемых результатов. Вот несколько примеров:

1. Входные данные: "2+34"

Ожидаемый результат: "234+"

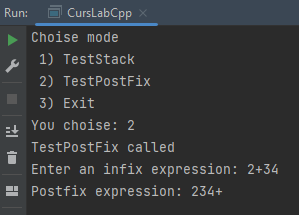


Рис. 6 - Тест режима PostFix № 1

1. Входные данные: "7-5+2"

Ожидаемый результат: "75-2+"

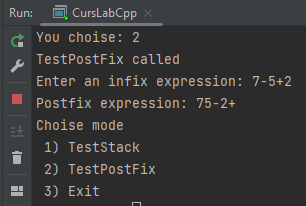


Рис. 7 - Тест режима PostFix № 2

1. Входные данные: "(8+6)3-9"

Ожидаемый результат: "86+39-"

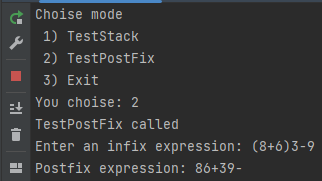


Рис. 8 - Тест режима PostFix № 3

1. Входные данные: "5+((6-2)8)/4"

Ожидаемый результат: "562-84/+"

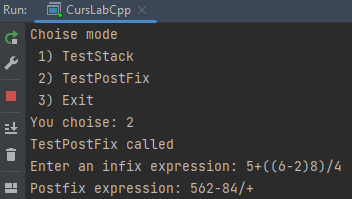


Рис. 9 - Тест режима PostFix № 4

1. Входные данные: "((4-2)7+3)/5"

Ожидаемый результат: "42-73+5/"

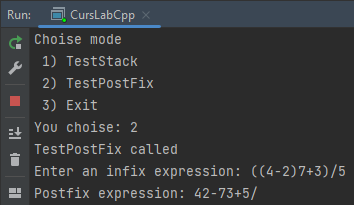


Рис. 10 - Тест режима PostFix № 5

1. Входные данные: "93+(7-2)/4-1"

Ожидаемый результат: "9372-4/+1-"

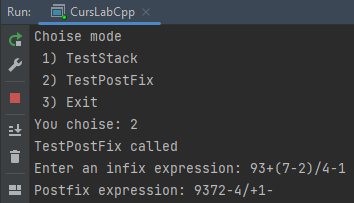


Рис. 11 - Тест режима PostFix № 6

# Двусвязный список

## 2.1 Постановка задачи

Для данной задачи требуется реализовать двусвязный список, который будет поддерживать операции вставки, удаления и поиска элементов.

Двусвязный список (Doubly Linked List) — это структура данных, состоящая из узлов, каждый из которых содержит ссылки на предыдущий и следующий узлы. Первый и последний узлы списка также имеют ссылки на nullptr, чтобы указать на начало и конец списка.

Задача заключается в реализации такого списка, который позволяет эффективно вставлять, удалять и находить элементы в нем.

Операции, которые требуется реализовать:

* Вставка элемента в список (в начало, в конец или после определенного элемента).
* Удаление элемента из списка.
* Поиск элемента в списке.

Двусвязный список предоставляет гибкость и эффективность в выполнении операций вставки и удаления, так как нет необходимости сдвигать остальные элементы списка при изменении его структуры.

В следующих пунктах мы более подробно рассмотрим описание входной и выходной информации, алгоритм решения задачи, общие требования к программе, описание структуры программы и другие важные аспекты задачи.

## 2.2 Описание входной и выходной информации

В данном пункте мы опишем, какие данные ожидаются на входе программы и какие результаты должны быть получены на выходе.

***Входные данные:***

Для работы с двусвязным списком могут быть следующие входные данные:

* Значение элемента, который нужно вставить в список.
* Значение элемента, который нужно удалить из списка.
* Значение элемента, который нужно найти в списке.

***Выходные данные:***

Результаты работы программы могут быть следующими:

* Список после выполнения операций вставки или удаления.
* Результат операции поиска элемента (true/false или позиция элемента в списке).

***Пример входных и выходных данных:***

Вставка элемента: значение элемента = 5

Входные данные: 5

Выходные данные: Список после вставки элемента 5: 2 -> 5 -> 7 -> 9

Удаление элемента: значение элемента = 7

Входные данные: 7

Выходные данные: Список после удаления элемента 7: 2 -> 5 -> 9

Поиск элемента: значение элемента = 9

Входные данные: 9

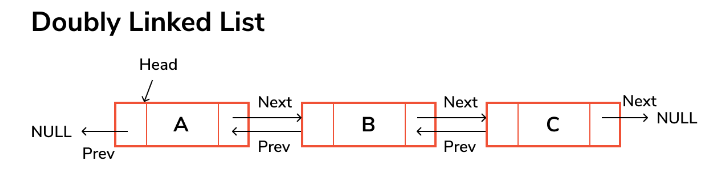
Выходные данные: Результат поиска элемента 9: true

## Алгоритм решения задачи

В данном пункте мы опишем алгоритм, который будет использоваться для решения задачи, связанной с двусвязным списком. Алгоритм будет включать основные операции, такие как вставка элемента, удаление элемента и поиск элемента.

Алгоритм решения задачи для двусвязного списка:

1. Создание структуры узла (Node):
   * Определение структуры Node, содержащей два указателя: prev (на предыдущий узел) и next (на следующий узел), а также поле value (значение узла).
2. Создание класса для работы с двусвязным списком (DoublyLinkedList):
   * Определение класса DoublyLinkedList.
   * В классе определение приватных переменных: указатель на начальный узел списка (head) и указатель на конечный узел списка (tail).
   * В классе определение публичных методов для выполнения операций с двусвязным списком, таких как вставка, удаление и поиск элементов.
3. Метод вставки элемента (insert):
   * Создание нового узла с переданным значением.
   * Если список пустой (head = nullptr), то новый узел становится начальным и конечным узлом списка.
   * Если список не пустой:
     + Присоединение нового узла к концу списка (текущий конечный узел next указывает на новый узел, а новый узел prev указывает на текущий конечный узел).
     + Обновление указателя на конечный узел списка (tail).
4. Метод удаления элемента (remove):
   * Поиск узла с переданным значением в списке.
   * Если узел найден:
     + Обновление указателей соседних узлов, чтобы обойти узел, который нужно удалить.
     + Освобождение памяти, занимаемой удаленным узлом.
   * Если узел не найден, вывод сообщения об ошибке.
5. Метод поиска элемента (search):
   * Начиная с начального узла, последовательно проверяем каждый узел списка на соответствие искомому значению.
   * Если узел с искомым значением найден, возвращаем true.
   * Если пройдены все узлы и искомое значение не найдено, возвращаем false.
6. Реализация дополнительных методов:
   * Метод для печати списка (print), который выводит значения всех узлов списка в порядке следования.
7. Создание объекта класса DoublyLinkedList и вызов методов для выполнения операций с двусвязным списком.



## Общие требования к программе

Общие требования к программе:

* + Код должен быть написан на языке C++.
  + Программа должна быть организована в виде модулей, каждый из которых отвечает за определенную функциональность.
  + Все модули и компоненты программы должны быть хорошо структурированы и читаемы.
  + Код должен быть написан в соответствии с принципами модульности и разделения ответственности.
  + Все функции и переменные должны быть названы ясно и описательно, согласно стандартам именования в C++.
  + Все входные и выходные данные должны быть корректно обработаны и проверены на ошибки.
  + Программа должна быть эффективной и оптимизированной, с минимальным использованием ресурсов.
  + Весь код должен быть документирован с помощью комментариев, объясняющих его назначение, логику и входные/выходные данные.
  + Программа должна быть кроссплатформенной и должна работать как на операционных системах Windows, так и на Linux или MacOS.
  + Программа должна быть написана с использованием современных практик программирования и соблюдать стандарты языка C++.
  + Код должен быть проверен на отсутствие ошибок и протестирован для обеспечения корректной работы.
  + Общие требования к программе помогут обеспечить ее качество, поддерживаемость, эффективность и надежность, а также улучшат понимание и использование кода другими разработчиками.

## Описание структуры программы для решения задачи

Структура программы для решения задачи с двусвязным списком может быть организована следующим образом:

1. Главный модуль (main.cpp):
   * Отвечает за взаимодействие с пользователем, ввод данных и вывод результатов.
   * Создает и инициализирует объекты двусвязного списка и другие необходимые компоненты.
   * Вызывает функции и методы для выполнения операций с двусвязным списком.
2. Модуль двусвязного списка (LinkedList.cpp, LinkedList.h):
   * Определяет класс LinkedList, представляющий структуру данных двусвязный список.
   * Содержит методы для добавления, удаления и обработки элементов списка, а также для доступа к элементам списка и его размеру.
   * Включает конструкторы, деструкторы и другие необходимые методы для управления списком.
3. Модуль узла списка (Node.cpp, Node.h):
   * Определяет класс Node, представляющий узел двусвязного списка.
   * Содержит переменные для хранения значения узла и указателей на предыдущий и следующий узлы.
   * Включает конструкторы, деструкторы и методы для работы с узлами списка.
4. Дополнительные модули (если необходимо):
   * Модули для тестирования функциональности двусвязного списка.
   * Модули для вспомогательных функций и операций над списком.

Структура программы позволяет разделить логику работы с двусвязным списком на отдельные модули, обеспечивая модульность, удобство сопровождения и повторное использование кода. Каждый модуль отвечает за свою часть функциональности и взаимодействует с другими модулями посредством вызовов функций и методов.