# 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.

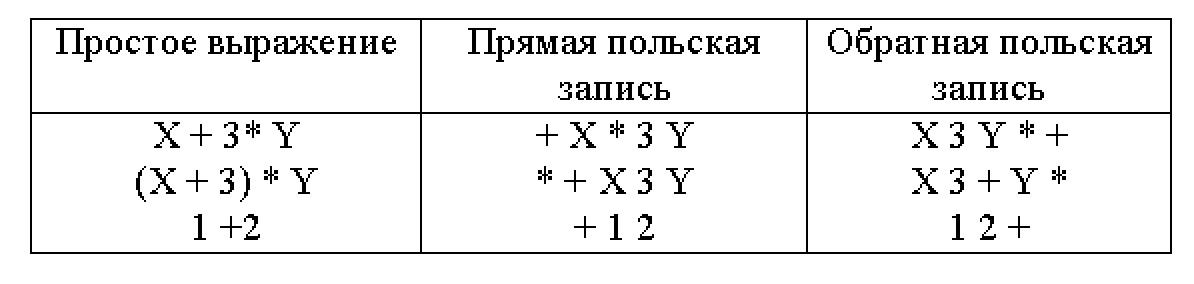


Рис. 1 – Пример прямой и обратной польской нотации

## 1.3 Алгоритм решения задачи

***1.3.1 Алгоритм работы стека***

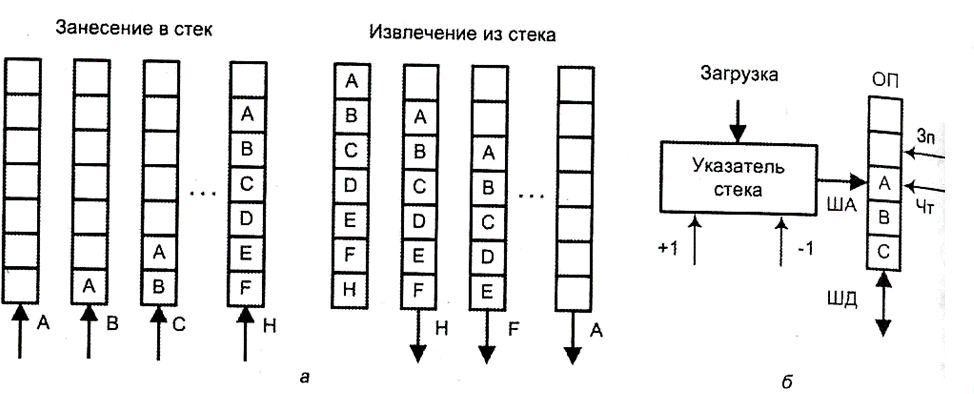
Стек — это тип данных, организованный по принципу "последний вошел, первый вышел" (LIFO - Last In, First Out). Основные операции, которые можно выполнять со стеком, это "push" (добавить элемент в стек) и "pop" (извлечь элемент из стека).

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).
   * 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.
5. если оператор O1 имеет больший приоритет, чем O2, то O1 помещается в стек поверх O2.
6. 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.

* + 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.

(Вставить блок-схемы)

## 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.

## Структура программы

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:

***1. Модуль стека (stack.h и stack.cpp)***:

* Этот модуль содержит определение и реализацию стека, который является основой нашего приложения. Вместо того, чтобы использовать встроенный стек STL, мы создали свой собственный тип стека для большего контроля над его поведением. Стек реализован как структура Stack с двумя полями: head и size. head - это указатель на вершину стека, а size - текущий размер стека.
* В модуле стека определены функции push, pop, top и is\_empty, которые соответствуют стандартным операциям стека.

***2. Модуль обработки входных данных (input\_handler.h и input\_handler.cpp)***:

* В этом модуле реализованы функции для чтения входных данных из файла и преобразования их в стек чисел. Этот модуль представляет собой служебный инструмент для подготовки данных для основного алгоритма.

***3. Модуль обработки выходных данных (output\_handler.h и output\_handler.cpp)***:

* После выполнения основного алгоритма результаты должны быть представлены пользователю или сохранены в файл. Этот модуль отвечает за представление результатов работы программы.

***4. Главный модуль (main.cpp):***

* Главный модуль содержит основной цикл программы. Он использует функции из других модулей для чтения входных данных, выполнения операций над стеком и вывода результатов.
* Общий стиль программирования, принятый в этой программе, соответствует стандартам C++ и принципам функционального программирования. Это означает, что весь код написан с целью минимизации изменяемого состояния и побочных эффектов. Все функции являются чистыми функциями, которые не изменяют своего окружения и всегда возвращают одинаковый результат для одинаковых входных данных. Кроме того, большинство данных являются неизменяемыми после их инициализации, что помогает предотвратить ошибки, связанные с изменением состояния.