

МИНОБРНАУКИ РОССИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ТУЛЬСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ»**

Институт прикладной математики и компьютерных наук Кафедра
информационной безопасности

Разработка программ по варианту № 32

(Реализация стеков и двусвязных списков на языке c++)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по дисциплине

(полное наименование учебной дисциплины)

Студент гр.

(индекс группы)

(подпись и дата)

(инициалы и
фамилия)

Руководитель

доц. ИПМКН,

К.Т.Н.,

(должность и
ученая степень)

(подпись и дата)

Сафронова М.А.

(инициалы и
фамилия)

ТУЛА 2023

УТВЕРЖДАЮ

Дир. ИПМКН

_____ А.А.Сычугов
" ____ " _____ 20__ г.

ЗАДАНИЕ

на курсовую работу по программированию

студента гр. _____
(ФИО, группа)

ТЕМА: _____

(Название, номер варианта)

Исходные данные _____

Задание получил: _____
(ФИО, подпись)

Дата выдачи задания : _____

Задание выдал: _____
(ФИО, подпись)

Срок защиты курсовой работы: _____

Замечания консультанта: _____

К защите допущен. Консультант работы _____

" ____ " _____ 20__ г.

Оглавление

Введение.....	4
1. Стеки.....	6
1.1 Постановка задачи.....	6
1.2 Описание входной и выходной информации	7
1.3 Алгоритм решения задачи.....	9
1.4 Общие требования к программе.....	14
1.5 Описание структуры программы для решения задачи	16
1.6 Инструкции по эксплуатации программы	18
1.7 Описание контрольного примера.....	20
2. Двусвязный список	22
2.1 Постановка задачи.....	22
2.2 Описание входной и выходной информации	23
2.3 Алгоритм решения задачи	24
2.4 Общие требования к программе.....	27
2.5 Описание структуры программы	28
2.6 Инструкция по эксплуатации программы.....	30
2.7 Описание контрольного примера.....	32
Заключение	36
Библиографический список	38
ПРИЛОЖЕНИЕ А	39
ПРИЛОЖЕНИЕ Б.....	47

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.

1. 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 " $3+45$ " 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+45" 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 (пример записи представлен на рисунке 1).

Простое выражение	Прямая польская запись	Обратная польская запись
$X + 3 * Y$	$+ X * 3 Y$	$X 3 Y * +$
$(X + 3) * Y$	$* + X 3 Y$	$X 3 + Y *$
$1 + 2$	$+ 1 2$	$1 2 +$

Рисунок 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. Общее графическое представление стека изображено на рисунке 2.

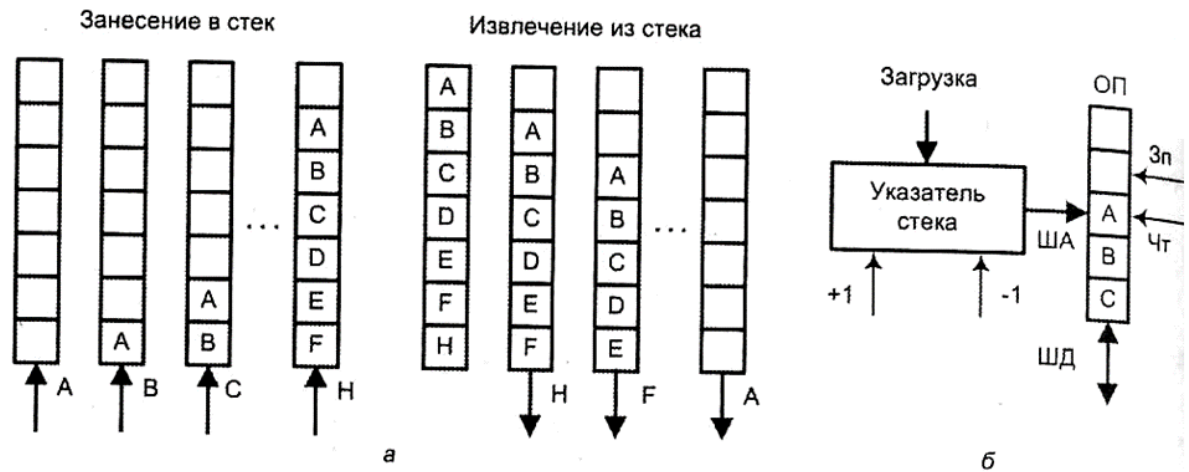


Рисунок 2 – Графическое представление стековой модели памяти

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 O_1 имеет меньший или равный приоритет, чем O_2 , то O_2 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 O_1 , или пока стек не станет пустым. После этого O_1 помещается в стек.

5. если оператор O_1 имеет больший приоритет, чем O_2 , то O_1 помещается в стек поверх O_2 .

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 конвертации из инфиксной формы записи выражения в постфиксную приведена на рисунке 3.

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.

1.4 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.

1.5 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. Схема взаимосвязи модулей между собой приведена на рисунке 4.

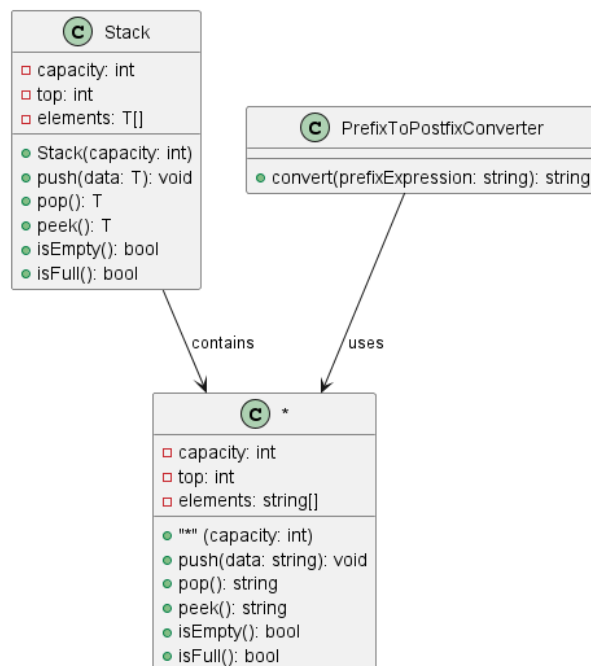


Рисунок 4 – Диаграмма классов проекта стека

1.6 Инструкции по эксплуатации программы

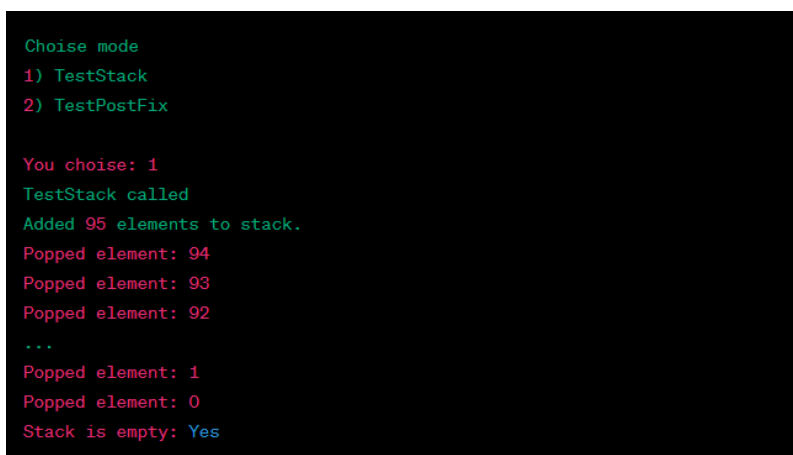
Компонент программы для тестирования стека и конвертации выражений представляет собой диалоговое консольное приложение, разработанное в рамках проекта C++ с использованием CMake для сборки под платформу Windows.

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. Диалоговое окно программы представлено на рисунке 5.



```
Choise mode
1) TestStack
2) TestPostFix

You choise: 1
TestStack called
Added 95 elements to stack.
Popped element: 94
Popped element: 93
Popped element: 92
...
Popped element: 1
Popped element: 0
Stack is empty: Yes
```

Рисунок 5 - 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.

Диалоговое окно программы отображено на рисунке 6.

```
Choise mode
1) TestStack
2) TestPostFix

You choise: 2
TestPostFix called
Enter an infix expression: 6*90+67-56*(48-800)
Postfix expression: 690*67+5648800-*-
```

Рисунок 6 - 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.

1.7 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+" (рисунок 7)

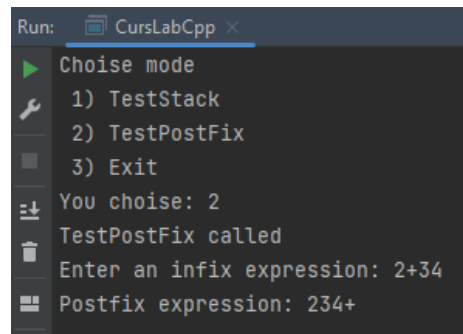


Рисунок 7 - Тест режима PostFix № 1

2. Входные данные: "7-5+2"

Ожидаемый результат: "75-2+" (рисунок 8)

```
Run: CursLabCpp x
You choise: 2
TestPostFix called
Enter an infix expression: 7-5+2
Postfix expression: 75-2+
Choise mode
1) TestStack
2) TestPostFix
3) Exit
```

Рисунок 8 - Тест режима PostFix № 2

3. Входные данные: $(8+6)3-9$

Ожидаемый результат: $86+39-$ (рисунок 9)

```
Choise mode
1) TestStack
2) TestPostFix
3) Exit
You choise: 2
TestPostFix called
Enter an infix expression: (8+6)3-9
Postfix expression: 86+39-
```

Рисунок 9 - Тест режима PostFix № 3

4. Входные данные: $5+((6-2)8)/4$

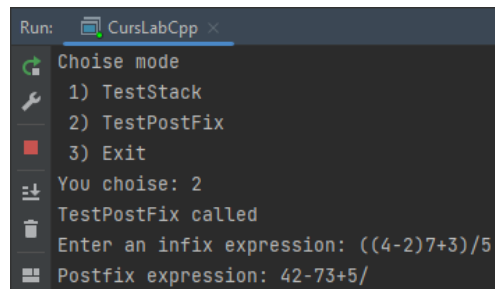
Ожидаемый результат: $562-84/+$ (рисунок 10)

```
Run: CursLabCpp x
Choise mode
1) TestStack
2) TestPostFix
3) Exit
You choise: 2
TestPostFix called
Enter an infix expression: 5+((6-2)8)/4
Postfix expression: 562-84/+
```

Рисунок 10 - Тест режима PostFix № 4

5. Входные данные: $((4-2)7+3)/5$

Ожидаемый результат: "42-73+5/" (рисунок 11)

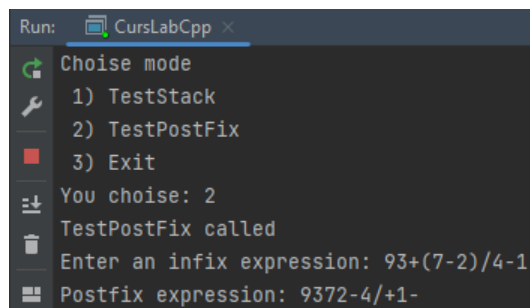


```
Run: CursLabCpp x
Choise mode
1) TestStack
2) TestPostFix
3) Exit
You choise: 2
TestPostFix called
Enter an infix expression: ((4-2)7+3)/5
Postfix expression: 42-73+5/
```

Рисунок 11 - Тест режима PostFix № 5

6. Входные данные: "93+(7-2)/4-1"

Ожидаемый результат: "9372-4/+1-" (рисунок 12)



```
Run: CursLabCpp x
Choise mode
1) TestStack
2) TestPostFix
3) Exit
You choise: 2
TestPostFix called
Enter an infix expression: 93+(7-2)/4-1
Postfix expression: 9372-4/+1-
```

Рисунок 12 - Тест режима PostFix № 6

2. 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

2.3 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 частных 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.

Графическое отображение структуры двусвязного списка можно видеть на рисунке 13.

Doubly Linked List

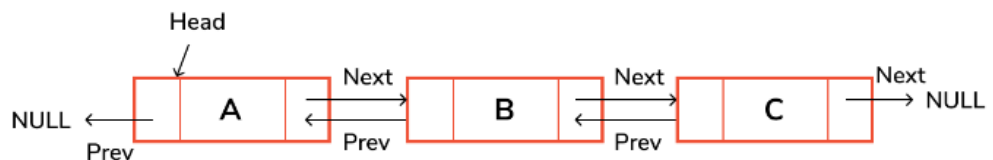


Рисунок 13 - Общее графическое представление двухсвязного списка

Отображение блок-схем алгоритмов основных операций с двусвязным списком, таких как:

- Добавление элемента в начало списка;
- Добавление элемента в конец списка;
- Удаление элемента из списка;
- Поиск элемента в списке;

представленно на рисунке 14.

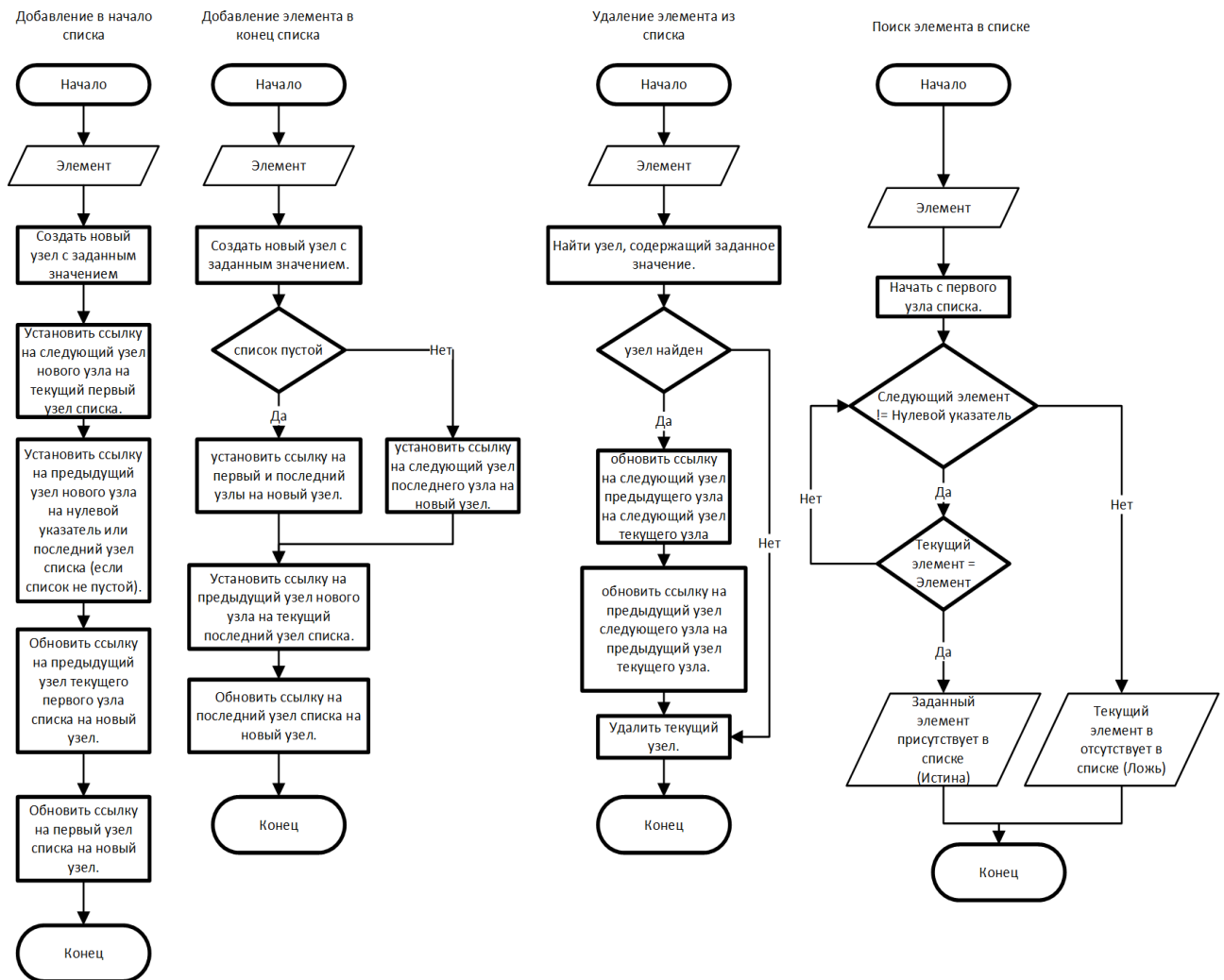


Рисунок 14 - Блок-схема методов объекта двусвязного списка.

2.4 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.

2.5 Описание структуры программы

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. Общая структура классов проекта представлена на рисунке 15.

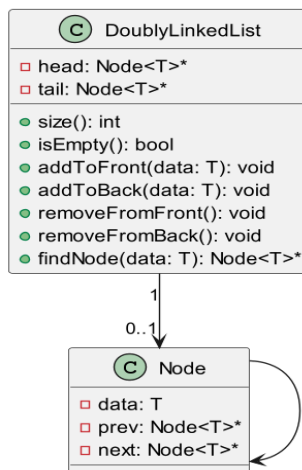


Рисунок 15 – Диаграмма классов проекта двусвязного списка

2.6 Инструкция по эксплуатации программы

1. Запуск программы:

- Убедитесь, что у вас установлен компилятор C++ и система сборки CMake.
- Скачайте исходный код программы и сохраните его в удобном для вас месте.
- Откройте командную строку или терминал и перейдите в каталог с исходным кодом программы.
- Создайте папку для сборки, например, "build", и перейдите в нее.
- Выполните команду сборки с использованием CMake: "cmake ." (указывает на папку с исходным кодом программы).
- После успешной генерации проекта, выполните команду сборки: "cmake --build .".

2. Запуск программы:

- После успешной сборки программы, найдите исполняемый файл в папке сборки (обычно с именем вашего проекта или "main").

- Запустите исполняемый файл, двойным кликом или через командную строку/терминал.

3. Ввод данных:

- При запуске программы, вам будет предложено ввести количество элементов в списке.

- Введите желаемое количество элементов и нажмите Enter.

4. Работа с двусвязным списком:

- Программа автоматически заполнит список случайными значениями и выполнит несколько операций с ним.

- Результаты операций будут выведены на экран.

5. Результаты выполнения программы:

- После завершения работы программы, вы увидите результаты операций с двусвязным списком.

- Результаты могут включать информацию о списке после вставки, наличии или отсутствии элемента, удалении элемента, состоянии списка после удаления и другие.

Пример вывода программы: List after insertion: [elements count = 20]
{ 27 34 24 36 34 33 28 35 37 38 28 28 30 32 34 33 35 21 39 21 } Is 33 in the
list? Yes Was 38 removed from the list? Yes List after removal: [elements count
= 19] { 27 34 24 36 34 33 28 35 37 28 28 30 32 34 33 35 21 39 21 } Element at
index 7: 35

6. Выход из программы:

- По завершении работы программы, закройте окно программы или нажмите клавишу "Ctrl+C" в командной строке/терминале, чтобы остановить выполнение программы.

Примечание: убедитесь, что в вашей системе установлены все необходимые компоненты для сборки и запуска программы. Если возникают проблемы, обратитесь к документации вашего компилятора C++ и системы сборки CMake для получения дополнительной информации.

2.7 Описание контрольного примера

В контрольном примере рассмотрим работу программы на примере создания и операций над двусвязным списком.

1. Ввод данных:

При запуске программы, пользователю будет предложено ввести количество элементов в списке.

Для контрольного примера, предположим, что пользователь ввел число 5.

2. Работа с двусвязным списком:

Программа автоматически создаст двусвязный список, состоящий из 5 элементов со случайными значениями.

Далее будут выполнены следующие операции над списком:

а) Вставка элемента:

Программа вставит элемент со значением 10 в начало списка.

Результат: список будет содержать элементы 10, значение, исходно сгенерированные случайным образом, например, 27, 34, 24, 36.

б) Удаление элемента:

Программа удалит элемент со значением 24 из списка.

Результат: элемент со значением 24 будет удален из списка, список будет содержать элементы 10, 27, 34, 36.

с) Поиск элемента:

Программа выполнит поиск элемента со значением 27 в списке.

Результат: программа сообщит, что элемент со значением 27 найден в списке.

d) Получение элемента по индексу:

Программа получит элемент с индексом 2 из списка.

Результат: программа сообщит, что элемент с индексом 2 имеет значение 34.

3. Вывод результатов:

После выполнения всех операций, программа выведет результаты на экран.

Пример вывода программы может быть следующим:

List after insertion: [elements count = 4] { 10 27 34 36 }

Is 27 in the list? Yes

Was 24 removed from the list? Yes

Element at index 2: 34

Примечание: Фактические значения элементов и их порядок может отличаться в каждом запуске программы из-за случайной генерации значений.

4. Завершение программы:

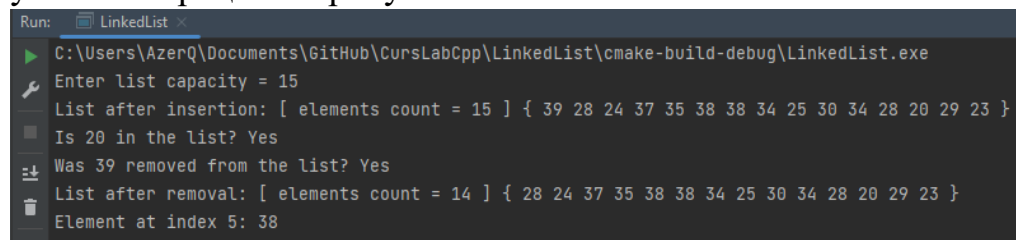
По завершении выполнения операций над списком, программа закончит свою работу.

Опишем несколько примеров использования входных данных для нашей программной реализации двухсвязного списка:

Входные данные: Емкость = 15

Результат:

Результат операций на рисунке 16.



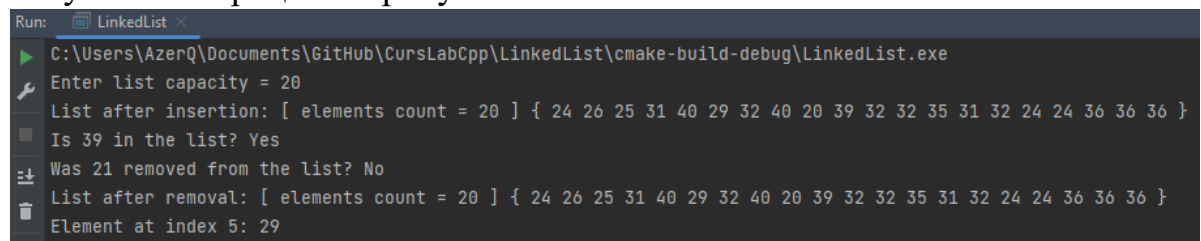
```
Run: LinkedList
C:\Users\AzerQ\Documents\GitHub\CursLabCpp\LinkedList\cmake-build-debug\LinkedList.exe
Enter list capacity = 15
List after insertion: [ elements count = 15 ] { 39 28 24 37 35 38 38 34 25 30 34 28 20 29 23 }
Is 20 in the list? Yes
Was 39 removed from the list? Yes
List after removal: [ elements count = 14 ] { 28 24 37 35 38 38 34 25 30 34 28 20 29 23 }
Element at index 5: 38
```

Рисунок 16 – Тест двухсвязного списка № 1

Входные данные Емкость = 20

Результат:

Результат операций на рисунке 17.



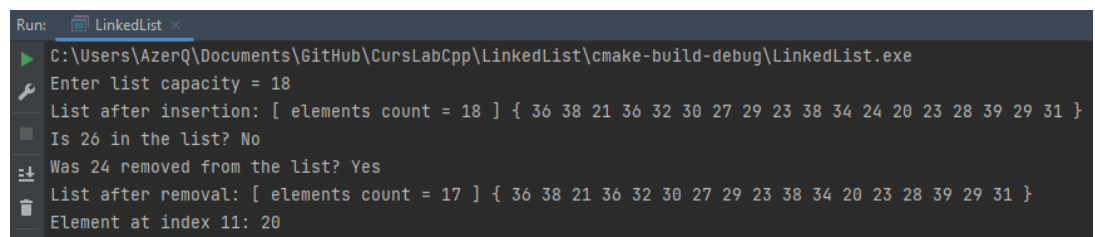
```
Run: LinkedList
C:\Users\AzerQ\Documents\GitHub\CursLabCpp\LinkedList\cmake-build-debug\LinkedList.exe
Enter list capacity = 20
List after insertion: [ elements count = 20 ] { 24 26 25 31 40 29 32 40 20 39 32 32 35 31 32 24 24 36 36 36 }
Is 39 in the list? Yes
Was 21 removed from the list? No
List after removal: [ elements count = 20 ] { 24 26 25 31 40 29 32 40 20 39 32 32 35 31 32 24 24 36 36 36 }
Element at index 5: 29
```

Рисунок 17 – Тест двухсвязного списка № 2

Входные данные Емкость = 18

Результат:

Результат операций на рисунке 18.



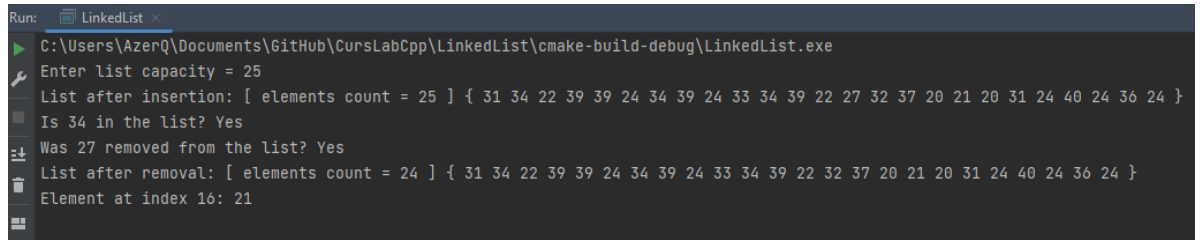
```
Run: LinkedList
C:\Users\AzerQ\Documents\GitHub\CursLabCpp\LinkedList\cmake-build-debug\LinkedList.exe
Enter list capacity = 18
List after insertion: [ elements count = 18 ] { 36 38 21 36 32 30 27 29 23 38 34 24 20 23 28 39 29 31 }
Is 26 in the list? No
Was 24 removed from the list? Yes
List after removal: [ elements count = 17 ] { 36 38 21 36 32 30 27 29 23 38 34 20 23 28 39 29 31 }
Element at index 11: 20
```

Рисунок 18 – Тест двухсвязного списка № 3

Входные данные: Емкость = 25

Результат:

Результат операций на рисунке 19.



```
Run: LinkedList
C:\Users\AzerQ\Documents\GitHub\CursLabCpp\LinkedList\cmake-build-debug\LinkedList.exe
Enter list capacity = 25
List after insertion: [ elements count = 25 ] { 31 34 22 39 39 24 34 39 24 33 34 39 22 27 32 37 20 21 20 31 24 40 24 36 24 }
Is 34 in the list? Yes
Was 27 removed from the list? Yes
List after removal: [ elements count = 24 ] { 31 34 22 39 39 24 34 39 24 33 34 39 22 32 37 20 21 20 31 24 40 24 36 24 }
Element at index 16: 21
```

Рисунок 19 – Тест двусвязного списка № 4

Заключение

В ходе выполнения данной курсовой работы был проведен анализ, проектирование и реализация двух структур данных: стека и двусвязного списка. Была разработана программа на языке C++, которая демонстрирует работу этих структур данных.

Стек реализован с использованием функционального программирования и шаблонного типа данных. Он обеспечивает основные операции, такие как добавление элемента (push), удаление элемента (pop) и получение верхнего элемента (peek). Были проведены тесты стека, которые подтвердили его правильную работу.

Двусвязный список реализован с использованием структуры и указателей на предыдущий и следующий элементы. Он предоставляет функциональности, такие как вставка элемента, удаление элемента, поиск элемента по значению и получение элемента по индексу. Были проведены тесты двусвязного списка, которые подтвердили его корректность.

Программа включает в себя консольное приложение, которое позволяет пользователю выбрать режим работы: тестирование стека или двусвязного списка. В режиме тестирования стека производится заполнение стека случайными числами из натурального ряда и последующее извлечение всех элементов. В режиме тестирования двусвязного списка демонстрируется вставка элементов, удаление элементов, поиск элементов и получение элементов по индексу.

В результате выполнения данной курсовой работы была получена работающая программа, которая успешно реализует стек и двусвязный список. Она демонстрирует основные операции над этими структурами данных и позволяет пользователю убедиться в их правильной работе.

Дальнейшее развитие программы может включать расширение функциональности стека и двусвязного списка, а также оптимизацию алгоритмов работы со структурами данных для повышения производительности.

В целом, выполнение данной курсовой работы позволило более глубоко изучить структуры данных, алгоритмы и язык программирования C++. Полученные знания и навыки могут быть применены в дальнейшей работе в области разработки программного обеспечения.

Библиографический список

1. Гасфилов, В. М. (2007). Структуры данных и алгоритмы в C++. БХВ-Петербург.
2. Weiss, M. A. (2013). Data Structures and Algorithm Analysis in C++. Pearson.
3. Sedgewick, R., & Wayne, K. (2011). Algorithms (4th Edition). Addison-Wesley Professional.
4. Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). Data Structures and Algorithms in C++. Wiley.
5. Алексеев, И. В. (2015). Структуры данных и алгоритмы: учебник для вузов. Москва: Бином.

ПРИЛОЖЕНИЕ А

Исходный код модуля postfix.h

```
#ifndef POSTFIX_H
#define POSTFIX_H
#include <string>
#include "stack.h"
std::string infix_to_postfix( std::string infix);
#endif // POSTFIX_H
```

Исходный код модуля stack.h

```
#pragma once
#include <optional>
namespace CustomStack
{
    template <typename T>
    struct Node
    {
        T data;
        Node<T>* next;
    };

    template <typename T>
    struct Stack
    {
        Node<T>* top;
    };

    template <typename T>
    Stack<T> CreateStack()
    {
        return Stack<T> { nullptr };
    }
}
```

```

}

template <typename T>
void push(Stack<T>& stack, T value)
{
    Node<T>* node = new Node<T> { value, stack.top };
    stack.top = node;
}

template <typename T>
std::optional<T> pop(Stack<T>& stack)
{
    if (stack.top == nullptr)
    {
        return {};
    }
    else
    {
        Node<T>* node = stack.top;
        T value = node->data;
        stack.top = node->next;
        delete node;
        return value;
    }
}

template <typename T>
bool isEmpty(const Stack<T>& stack)
{
    return stack.top == nullptr;
}

template <typename T>
std::optional<T> peek(const Stack<T>& stack)
{

```



```

        if (stack.top == nullptr)
        {
            return {};
        }
        else
        {
            return stack.top->data;
        }
    }
}

```

Исходный код модуля postfix.cpp

```

#include "include/postfix.h"
#include <iostream>
int get_precedence(char op)
{
    switch (op)
    {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        default:
            return 0;
    }
}

std::string infix_to_postfix(std::string infix)
{

```

```

//std::cout << infix;
CustomStack::Stack<char> stack;
std::string postfix;
for (char c : infix)
{
    if (std::isdigit(c))
    {
        postfix += c;
    }
    else if (c == '(')
    {
        CustomStack::push(stack, c);
    }
    else if (c == ')')
    {
        while (!CustomStack::isEmpty(stack) &&
CustomStack::peek(stack).has_value() &&
CustomStack::peek(stack).value() != '(')
        {
            auto popped = CustomStack::pop(stack);
            if (popped.has_value())
            {
                postfix += popped.value();
            }
        }
        if (!CustomStack::isEmpty(stack))
        {
            CustomStack::pop(stack); // Pop the '('
        }
    }
    else
    { // If the character is an operator

```

```

        while (!CustomStack::isEmpty(stack) && get_precedence(c)
<= get_precedence(CustomStack::peek(stack).value_or(0)))
        {
            std::optional<char> popped;
            try
            {
                popped = CustomStack::pop(stack);
            }
            catch (const std::exception &e)
            {
            }
            if (popped.has_value())
            {
                postfix += popped.value();
            }
        }
        CustomStack::push(stack, c);
    }
}

// Pop any remaining operators from the stack and add to postfix
while (!CustomStack::isEmpty(stack))
{
    auto popped = CustomStack::pop(stack);
    if (popped.has_value())
    {
        postfix += popped.value();
    }
}
return postfix;
}

```

```

#include <iostream>
#include "include/postfix.h"
#include "include/stack.h"
#include <random>
using namespace std;

void TestStack()
{
    cout << "TestStack called" << endl;
    // Создаем пустой стек
    CustomStack::Stack<int> stack = CustomStack::CreateStack<int>();

    // Генератор случайных чисел
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> dis(0, 100);

    // Случайное количество элементов
    int n = dis(gen);

    // Добавляем случайное количество элементов в стек
    for (int i = 0; i < n; ++i) {
        CustomStack::push(stack, i);
    }

    std::cout << "Added " << n << " elements to stack." << std::endl;

    // Удаляем элементы из стека и печатаем их
    while(!CustomStack::isEmpty(stack)) {
        if(auto val = CustomStack::pop(stack)) {
            std::cout << "Popped element: " << *val << std::endl;
        }
    }
}

```

```

    }
}

// Проверяем, пуст ли стек
std::cout << "Stack is empty: " << (CustomStack::isEmpty(stack) ?
"Yes" : "No") << std::endl;
}

void TestPostFix()
{
    try
    {
        cout << "TestPostFix called" << endl;
        std::string infix;
        std::cout << "Enter an infix expression: ";
        std::cin >> infix;
        //std::cout << infix << '\n';
        std::string postfix = infix_to_postfix(infix);
        std::cout << "Postfix expression: " << postfix << '\n';
    }
    catch (const std::exception &e)
    {
        std::cout << "Ошибка: " << e.what() << std::endl;
    }
}

int main() {
    int userChoise = 0;
    while (userChoise != 3) {
        cout << "Choise mode \n 1) TestStack \n 2) TestPostFix \n 3)
Exit \nYou choise: ";
        cin >> userChoise;
        switch (userChoise) {
            case 1:

```

```
        TestStack();
        break;
    case 2:
        TestPostFix();
        break;
    default:
        break;
    }
}

cin.get();
return 0;

}
```

Исходный код модуля Node.h

```
#ifndef NODE_H
#define NODE_H
template <typename T>
struct Node {
    T value;
    Node* prev;
    Node* next;
};
#endif // NODE_H
```

Исходный код модуля DoublyLinkedList.h

```
#ifndef DOUBLYLINKEDLIST_H
#define DOUBLYLINKEDLIST_H
#include "Node.h"
#include <iostream>
using namespace std;
template<typename T>
class DoublyLinkedList {
public:
    DoublyLinkedList();
    ~DoublyLinkedList();
    void insert(const T &value);
    bool remove(const T &value);
    bool search(const T &value) const;
    void print() const;
    Node<T> getHead();
};
```

```

        Node<T> getTail();
        int getCount();
        Node<T> *getElementByIndex(int index);
private:
        Node<T> *head;
        Node<T> *tail;
        int count = 0;

};

template<typename T>
int DoublyLinkedList<T>::getCount() {
    return count;
}

template<typename T>
DoublyLinkedList<T>::DoublyLinkedList() : head(nullptr), tail(nullptr)
{}

template<typename T>
DoublyLinkedList<T>::~~DoublyLinkedList() {
    Node<T> *current = head;
    while (current != nullptr) {
        Node<T> *next = current->next;
        delete current;
        current = next;
    }
}

template<typename T>
void DoublyLinkedList<T>::insert(const T &value) {
    Node<T> *newNode = new Node<T>;
    newNode->value = value;

```



```

        newNode->prev = nullptr;
        newNode->next = nullptr;
        if (head == nullptr) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
        count++;
    }
template<typename T>
Node<T> DoublyLinkedList<T>::getHead() {
    return *head;
}
template<typename T>
Node<T> DoublyLinkedList<T>::getTail() {
    return *tail;
}

template<typename T>
bool DoublyLinkedList<T>::remove(const T &value) {
    Node<T> *current = head;
    while (current != nullptr) {
        if (current->value == value) {
            if (current == head) {
                head = current->next;
                if (head != nullptr) {
                    head->prev = nullptr;
                } else {
                    tail = nullptr;
                }
            }
        }
        current = current->next;
    }
}

```

```

        }
    } else if (current == tail) {
        tail = current->prev;
        if (tail != nullptr) {
            tail->next = nullptr;
        } else {
            head = nullptr;
        }
    } else {
        current->prev->next = current->next;
        current->next->prev = current->prev;
    }
    delete current;
    count--;
    return true;
}
current = current->next;
}
return false;
}

template<typename T>
bool DoublyLinkedList<T>::search(const T &value) const {
    Node<T> *current = head;
    while (current != nullptr) {
        if (current->value == value) {
            return true;
        }
        current = current->next;
    }
    return false;
}

template<typename T>

```

```

void DoublyLinkedList<T>::print() const {
    cout << "[ elements count = " << count << " ] " << "{ ";
    Node<T> *current = head;
    while (current != nullptr) {
        cout << current->value << " ";
        current = current->next;
    }
    cout << "}" << endl;
}

template<typename T>
Node<T> *DoublyLinkedList<T>::getElementByIndex(int index) {
    if (index < 0 || index >= count) {
        return nullptr;
    }
    Node<T> *current = nullptr;
    // Если индекс в первой половине списка, идем от начала
    if (index < count / 2) {
        current = head;
        for (int i = 0; i < index; i++) {
            current = current->next;
        }
    }
    // Если индекс во второй половине списка, идем от конца
    else {
        current = tail;
        for (int i = count - 1; i > index; i--) {
            current = current->prev;
        }
    }
    return current;
}

#endif // DOUBLYLINKEDLIST_H

```

Исходный код модуля main.cpp для второй задачи

```
#include <iostream>
#include "include/DoublyLinkedList.h"
#include <cstdlib>
#include <ctime>

void testDoublyLinkedList(int N) {
    const int MAX_VALUE = 40;
    const int MIN_VALUE = 20;

    auto getRand { [](){return (rand() % (MAX_VALUE - MIN_VALUE + 1) +
MIN_VALUE);} };

    DoublyLinkedList<int> list;
    srand(time(0)); // Инициализация генератора случайных чисел
    // Заполнение списка случайными числами
    for (int i = 0; i < N; i++) {
        int randomNumber = getRand(); // Случайное число от 20 до 200
        list.insert(randomNumber);
        //cout << "Inserted " << randomNumber << " into the list." <<
endl;
    }
    // Вывод списка
    cout << "List after insertion: ";
    list.print();
    // Поиск элемента в списке
    int searchNumber = getRand(); // Случайное число от 20 до 200
    bool found = list.search(searchNumber);
    cout << "Is " << searchNumber << " in the list? " << (found ?
"Yes" : "No") << endl;
    // Удаление элемента из списка
    int removeNumber = getRand(); // Случайное число от 20 до 200
    bool removed = list.remove(removeNumber);
    cout << "Was " << removeNumber << " removed from the list? " <<
(removed ? "Yes" : "No") << endl;
    // Вывод списка после удаления
```

```

    cout << "List after removal: ";
    list.print();
    // Поиск элемента по индексу
    int index = rand() % N; // Случайный индекс
    Node<int>* node = list.getElementByIndex(index);
    if (node != nullptr) {
        cout << "Element at index " << index << ": " << node->value <<
endl;
    } else {
        cout << "No element at index " << index << endl;
    }
}

int main() {
    int listCapacity;
    cout << "Enter list capacity = ";
    cin >> listCapacity;
    testDoublyLinkedList(listCapacity);
    getchar();
    return 0;
}

```