|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИИТ)**

**Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)**

**ОТЧЁТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №4**

**по дисциплине «Структуры и алгоритмы обработки данных»**

|  |  |  |  |
| --- | --- | --- | --- |
| Отчет представлен к  рассмотрению:  Студент группы ИКБО-36-22 | «02» октября  2023 г. | (подпись) | Утенков Ю.Ю. |
|  |  |  |  |
| Преподаватель | «02» октября  2023 г. | (подпись) | Красников С.А. |

Москва 2023 г.

Оглавление

[Цель работы. 3](#_Toc147144046)

[Ход работы. 3](#_Toc147144047)

[Формулировка задачи. 3](#_Toc147144048)

[Математическая модель решения (описание алгоритма). 3](#_Toc147144049)

[Код программы с комментариями. 7](#_Toc147144050)

[Результаты тестирования. 10](#_Toc147144051)

[Пример 1: 10](#_Toc147144052)

[Пример 2: 13](#_Toc147144053)

[Выводы. 15](#_Toc147144054)

[Список информационных источников. 15](#_Toc147144055)

# Цель работы.

освоить приёмы работы с нелинейными списками.

# Ход работы.

## Формулировка задачи.

Составить программу создания двоичного дерева поиска и реализовать процедуры для работы с деревом согласно 27 варианту. Процедуры оформить в виде самостоятельных режимов работы созданного дерева. Выбор режимов производить с помощью пользовательского (иерархического ниспадающего) меню. Провести полное тестирование программы на дереве размером n>=10 элементов, сформированном вводом с клавиатуры. Тест-примеры определить самостоятельно. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе. Сделать выводы о проделанной работе, основанные на полученных результатах. Оформить отчет с описанием принципов программной реализации алгоритмов работы с деревом, описанием текста исходного кода и проведенного тестирования программы.

Согласно варианту:

1. Тип дерева: Бинарное дерево поиска.
2. Тип значения узла: Символьный (char).
3. Возможность вставки элемента.
4. Возможность обратного обхода дерева.
5. Возможность симметричного обхода дерева.
6. Нахождение суммы значений листьев дерева.
7. Нахождение высоты данного дерева.

## Математическая модель решения (описание алгоритма).

**Листинг 3.**

Данный алгоритм вставляет узел с элементом типа char в бинарное дерево поиска. Сначала он проверяет допустимость символа и его значение. Если дерево пусто, создается новый узел с символом в качестве корня. В противном случае начинается поиск места для вставки символа, сравнивая его с символами в узлах и двигаясь вниз по дереву. Если символ уже существует в дереве, операция завершается неудачно. В противном случае создается новый узел и вставляется либо в левое, либо в правое поддерево родительского узла, в зависимости от результата сравнения символа с символом в родительском узле. В конечном итоге, функция возвращает успешное завершение, указывая на успешную вставку символа в дерево.

**Листинг 4.**

Данный алгоритм принимает на вход начальный узел (NodeTree\*) бинарного дерева и выполняет обратный обход данного дерева. Он выполняет следующие действия:

1. Проверяет, если текущий узел node равен nullptr, что указывает на пустое поддерево. В этом случае возвращается 0, так как у пустого узла нет данных для обработки.
2. Инициализирует переменную counter начальным значением 0.
3. Рекурсивно вызывает функцию reversePassageTree для левого поддерева node, добавляя результат к counter. То есть, алгоритм обходит все узлы в левом поддереве, собирая сумму символов.
4. Аналогично, рекурсивно вызывает функцию reversePassageTree для правого поддерева node, и суммирует её результат с текущим counter.
5. Инициализирует переменную counter значением символа в текущем узле ( node->symbol). Это начальное значение суммы с узла node.
6. По завершении обратного прохода по всем узлам дерева, возвращает общую сумму символов, собранную в переменной counter.

Приведённый ниже алгоритм представляет собой рекурсивный способ пройти через все узлы бинарного дерева и вычислить сумму символов (в кодировке ASCII), хранящихся в них.

**Листинг 5.**

Данный алгоритм выполняет симметричный обход бинарного дерева, начиная с заданного узла node. Он использует стек для хранения узлов и выполняет итеративный обход в порядке увеличения значений символов. Алгоритм начинает с проверки, существует ли узел node. Если узла нет (то есть, дерево пусто), то обход невозможен, и функция завершает выполнение. Затем создается стек, который будет использоваться для отслеживания уз current на начальный узел node. Алгоритм затем входит в цикл, который продолжается до тех пор, пока current не станет nullptr, и стек не пуст. Внутри цикла алгоритм:

* Перемещается влево по дереву (перемещается к левому потомку current) и добавляет каждый узел, который посещается на этом пути, в стек.
* Когда больше нет левых потомков (когда current становится nullptr), алгоритм извлекает последний узел из стека (с помощью stack.top()) и выводит его символ на экран. Это происходит в порядке увеличения значений символов.
* После вывода символа текущего узла, алгоритм перемещается вправо (к правому потомку current), чтобы продолжить обход в правой части дерева.

Алгоритм продолжает это поведение до тех пор, пока все узлы дерева не будут посещены и выведены на экран. Таким образом, данный алгоритм позволяет выполнить симметричный обход бинарного дерева, выводя значения символов на экран.

**Листинг 6.**

Данный алгоритм позволяет получить сумму значений только у листьев бинарного дерева. Он использует итеративный подход и стек для обхода дерева и выполнения следующих действий:

1. Инициализирует переменную leafSum нулем, которая будет хранить сумму значений листьев.
2. Создает стек stack, который будет использоваться для итеративного обхода дерева.
3. Помещает корневой узел (head) в стек, начиная с корня дерева.
4. Запускает цикл, который выполняется, пока стек не станет пустым. Внутри цикла:
   * Извлекает верхний узел из стека и присваивается переменной current.
   * Проверяет, является ли текущий узел листом, то есть не имеет левых и правых потомков (current->left == nullptr и current->right == nullptr).
   * Если current - лист, то его значение (current->symbol) добавляется к сумме leafSum.
   * В противном случае, если current не является листом, добавляет его правого и левого потомков (если они существуют) в стек для последующей обработки.
5. После завершения обхода всех узлов дерева, возвращается сумма значений листьев (leafSum).

Таким образом, этот алгоритм позволяет вычислить сумму значений только у листьев бинарного дерева, обходя его итеративным способом с использованием стека.

**Листинг 7.**

Этот алгоритм позволяет определить высоту бинарного дерева поиска (БДП), начиная с заданного узла node. Если дерево пусто (узел node равен nullptr), то высота дерева считается равной -1. В противном случае, алгоритм рекурсивно вычисляет высоту дерева как максимальную из высот левого и правого поддеревьев, увеличенную на 1. Это делается для каждого узла, начиная с корня и двигаясь вниз по дереву. Высота дерева представляет собой максимальное путь от узла до листа (глубину), на которых находятся узлы, и она используется для оценки структуры и размера дерева.

**Листинг 8.**

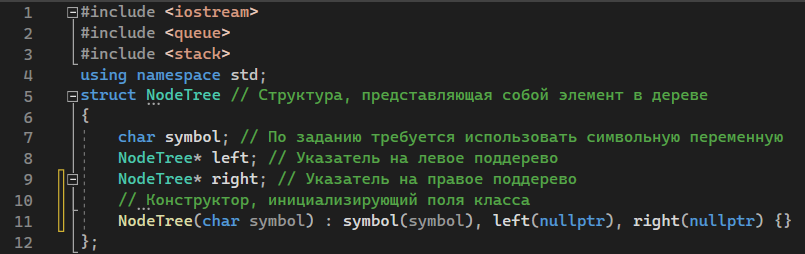
Данный фрагмент кода представляет собой деструктор для освобождения памяти, выделенной для бинарного дерева. Он использует симметричный обход дерева для удаления всех его узлов, при этом сначала удаляются листья, а затем родители, которые стали листьями после удаления своих потомков. Этот метод освобождения памяти является безопасным и эффективным для бинарных деревьев.

Описание алгоритма:

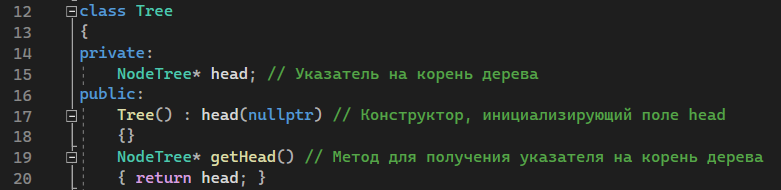
1. Создается стек stack, который будет использоваться для итеративного симметричного обхода дерева.
2. Устанавливается начальный указатель current на корень дерева (head).
3. Запускается цикл, который продолжается, пока current не станет равным nullptr и стек не станет пустым.
4. Внутри цикла алгоритм:
   * Добавляет узлы в стек, перемещаясь влево по дереву (к левому потомку current) до тех пор, пока не достигнет самого левого узла (листа) текущей ветви.
   * Когда больше нет левых потомков, алгоритм извлекает верхний узел из стека и временно сохраняет его в temp.
   * Затем перемещается вправо по дереву (к правому потомку current) и удаляет узел temp с помощью оператора delete. Это приводит к освобождению памяти, выделенной под этот узел.
5. После завершения цикла все узлы дерева будут удалены, и память будет освобождена.

Симметричный обход безопасен для освобождения памяти, потому что он гарантирует, что узлы удаляются в правильном порядке: сначала листья, а затем родители, которые стали листьями после удаления своих потомков. Это предотвращает утечки памяти и обеспечивает корректное освобождение всей динамической памяти, выделенной под узлы дерева.

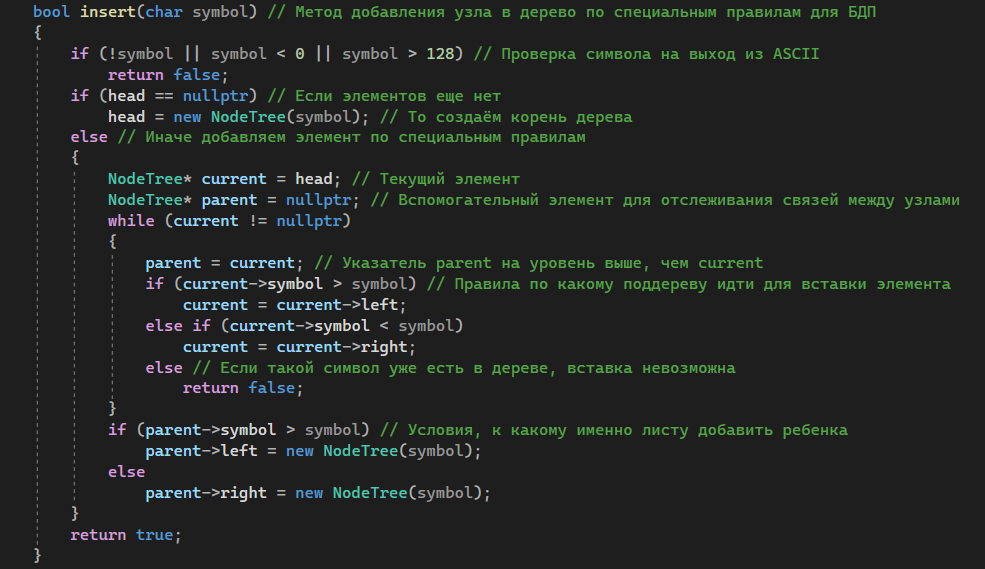
## Код программы с комментариями.



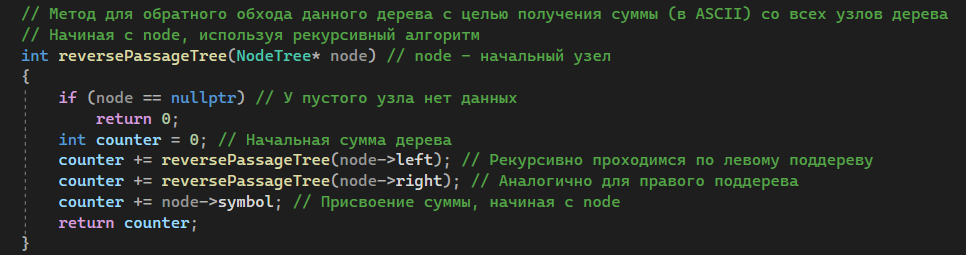
*Листинг 1. Структура узла в дереве.*

**

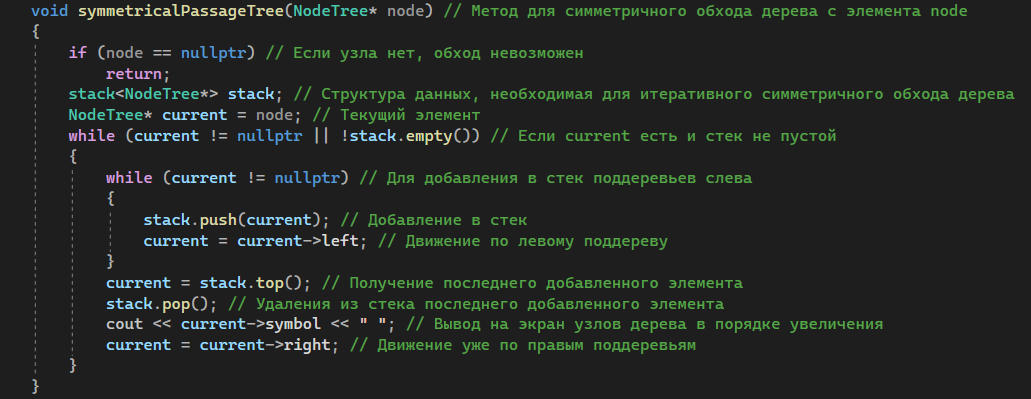
*Листинг 2. Поле метод и два конструктора дерева.*

**

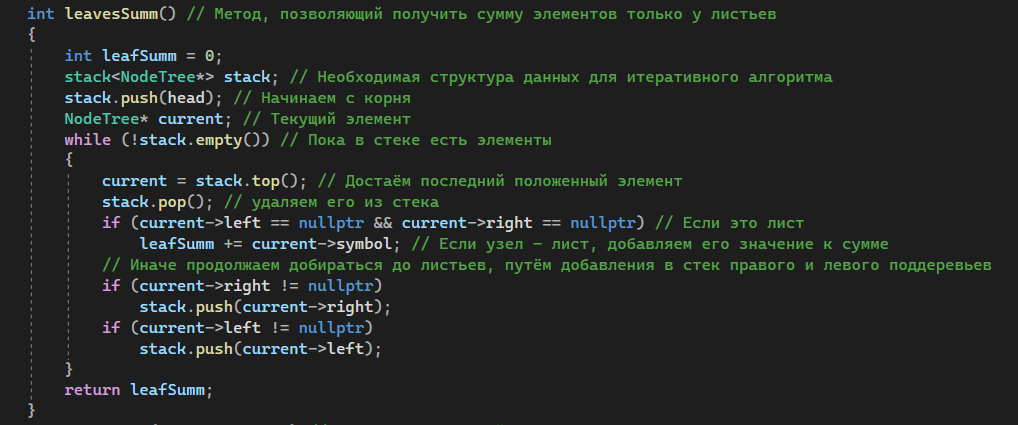
*Листинг 3. Метод, позволяющий добавлять узел БДП по специальным правилам.*



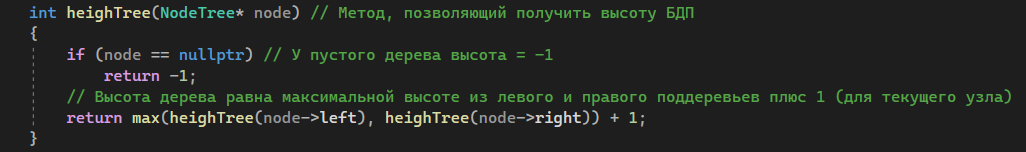
*Листинг 4. Метод, реализующий обратный обход данного бинарного дерева поиcка и вычисляющий сумму узлов в кодировке ASCII.*

**

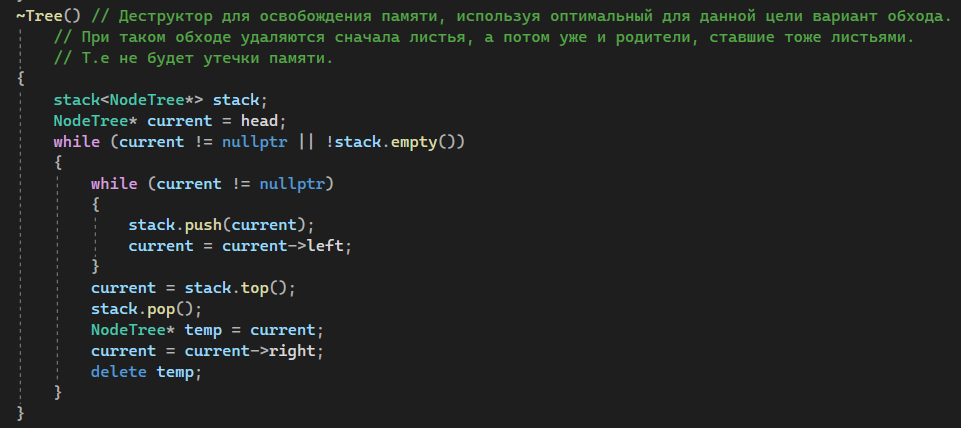
*Листинг 5. Метод, реализующий симметричный обход бинарного дерева с выводом узлов на экран в определённой последовательности.*



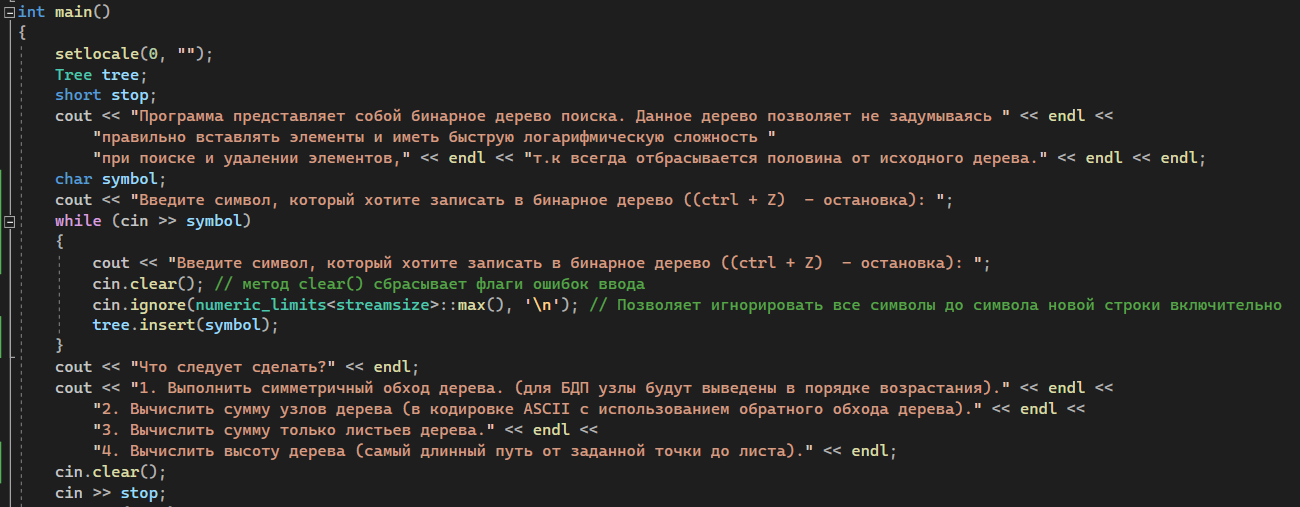
*Листинг 6. Метод, позволяющий вычислить сумму листьев дерева.*

**

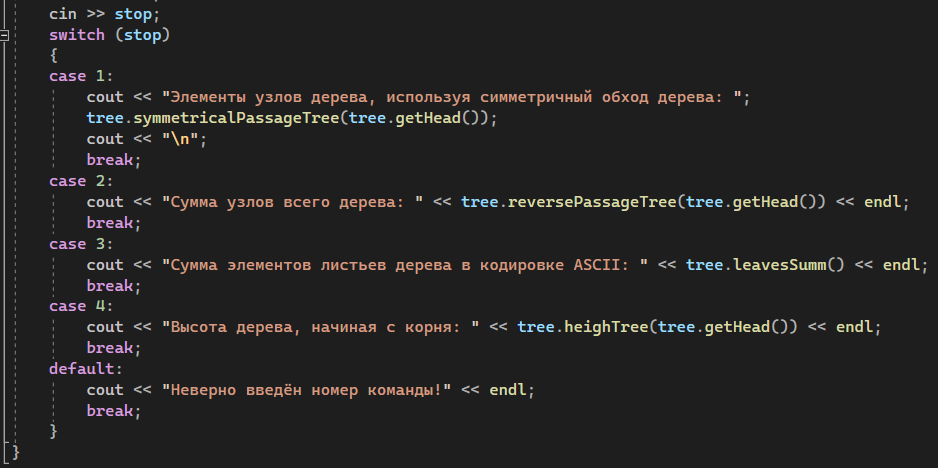
*Листинг 7. Метод, вычисляющий высоту сформированного дерева.*

**

*Листинг 8. Деструктор, для безопасного освобождения памяти.*



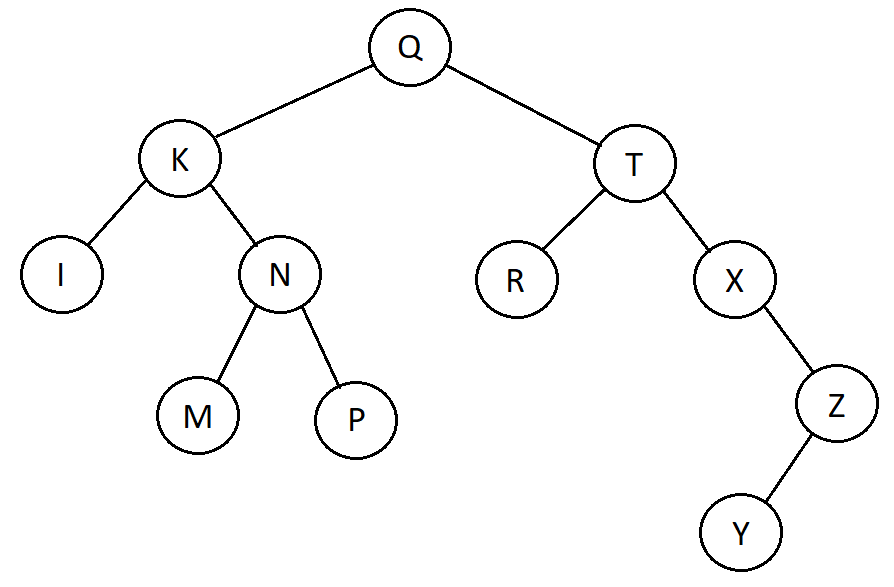
*Листинг 9. Основная функция main().*



*Листинг 10. Основная функция main()(Продолжение).*

## Результаты тестирования.

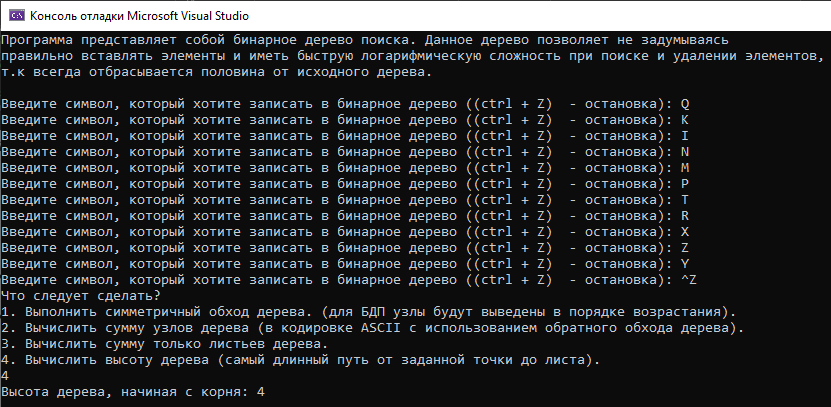
### **Пример 1:**



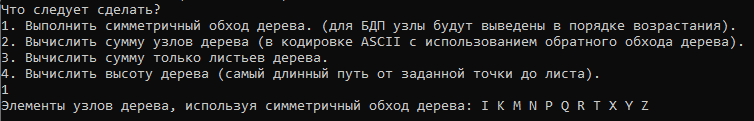
*Рис.1. Графическое изображение данного дерева.*

|  |  |
| --- | --- |
| Символ | Кодировка в ASCII |
| Q | 81 |
| K | 75 |
| I | 73 |
| N | 78 |
| M | 77 |
| P | 80 |
| T | 84 |
| R | 82 |
| X | 88 |
| Z | 90 |
| Y | 89 |

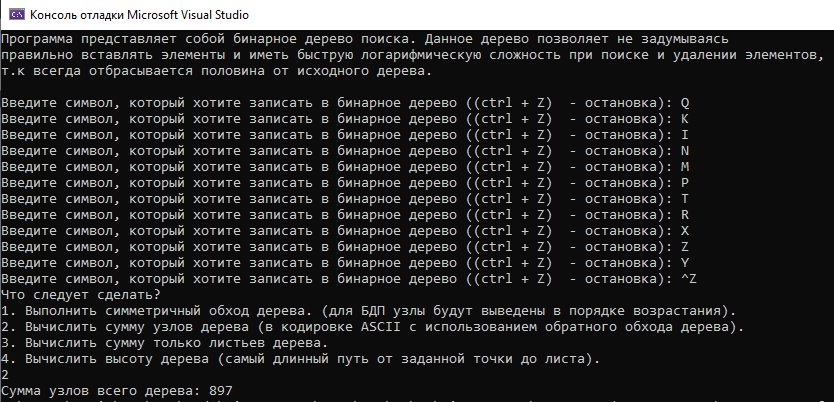
*Табл.1. Таблица перекодировки данных символов.*

**

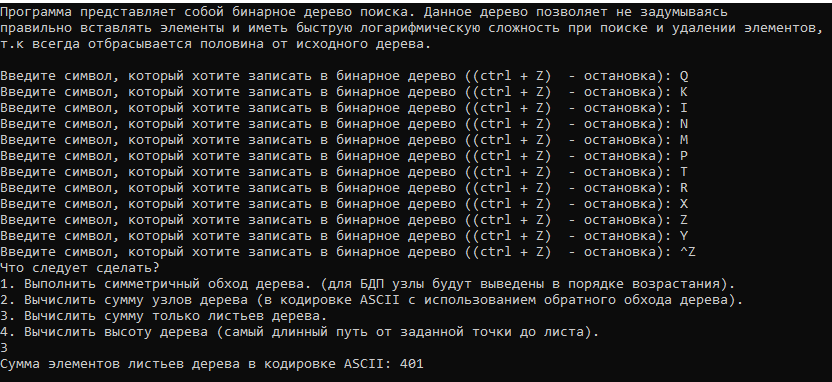
*Рис.2. Результат добавления узлов и вычисления высоты для данного дерева.*

**

*Рис.3. Пример симметричного обхода данного дерева.*

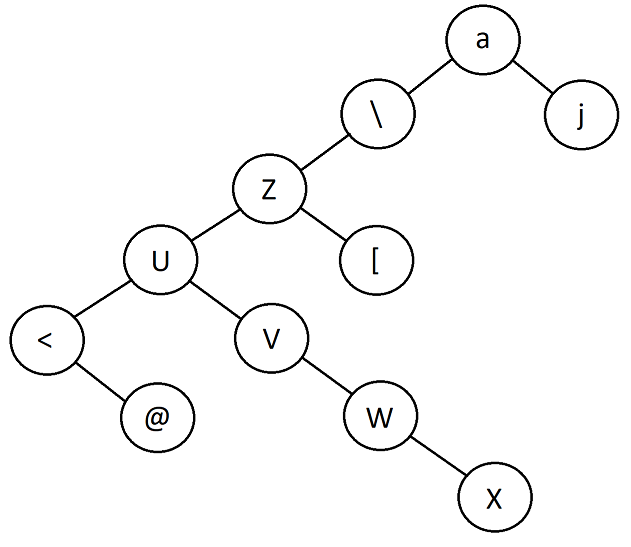
**

*Рис.4. Пример обратного обхода дерева и вычисления суммы всех узлов.*

**

*Рис.5. Пример вычисления суммы узлов листьев в дереве.*

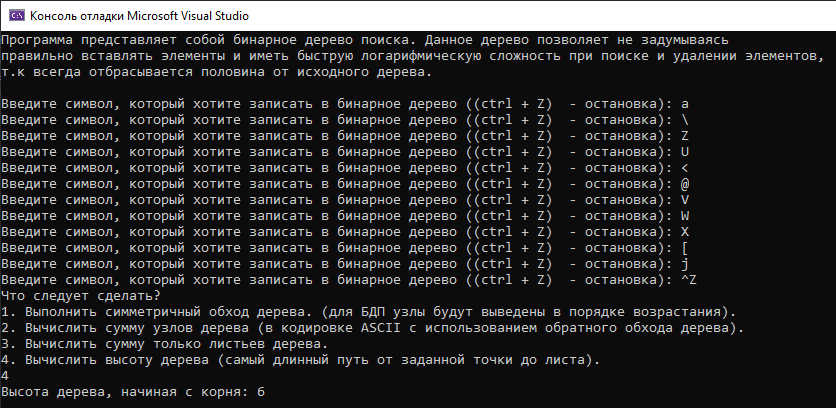
### **Пример 2:**

**

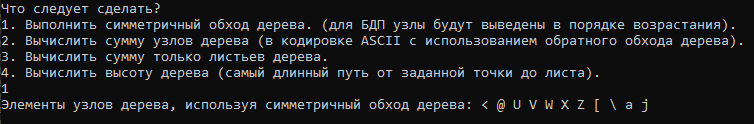
*Рис.6. Графическое изображение данного дерева.*

|  |  |
| --- | --- |
| Символ | Кодировка в ASCII |
| a | 97 |
| \ | 92 |
| Z | 90 |
| U | 85 |
| < | 60 |
| @ | 64 |
| V | 86 |
| W | 87 |
| X | 88 |
| [ | 91 |
| j | 106 |

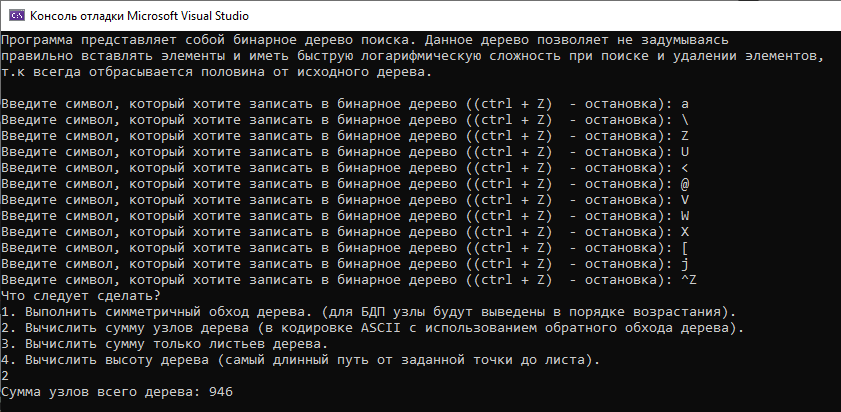
*Табл.2. Таблица перекодировки данных символов.*

**

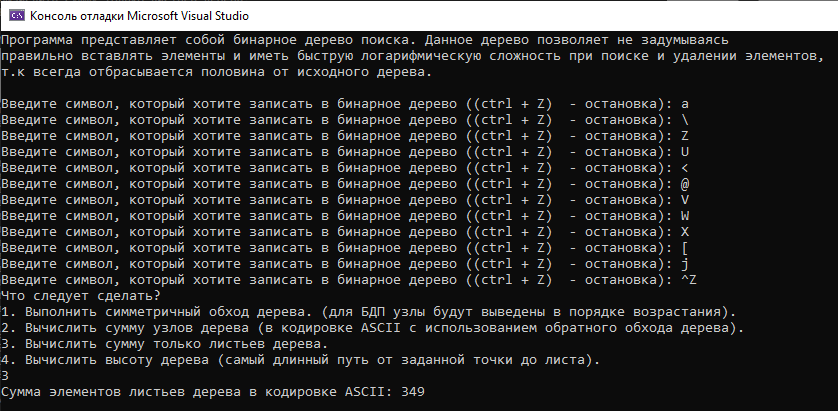
*Рис.7. Результат добавления узлов и вычисления высоты для данного дерева.*

**

*Рис.8. Пример симметричного обхода данного дерева.*

**

*Рис.9. Пример обратного обхода дерева и вычисления суммы всех узлов.*

**

*Рис.10. Пример вычисления суммы узлов листьев в дереве.*

# Выводы.

В ходе выполнение данной работы я получил навыки работы с БДП и изучил их особенности. Бинарные деревья поиска представляют собой важную структуру данных, обеспечивающую эффективное управление упорядоченными данными. Основные операции, такие как вставка, поиск и удаление, выполняются быстро, что делает БДП полезными в различных приложениях.

# Список информационных источников.

1. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих, 2017. – С. 100-126
2. Кораблин Ю.П. Структуры и алгоритмы обработки данных. Часть 1 [Электронный ресурс]: учебно методическое пособие / Ю.П.Кораблин, В.П.Сыромятников, Л.А. Скворцова – М.: РТУ МИРЭА, 2020. – 1 электрон. опт. диск (CD-ROM)
3. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL:https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 20.09.2023).