



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ

ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА

ПРОГРАММНАЯ ИНЖЕНЕРИЯ (ИУ7)

О Т Ч Е Т

По лабораторной работе № 6

Название: Обработка деревьев

Дисциплина: Типы и структуры данных

Вариант: 6

Студент

ИУ7-31Б

(Группа)

Савинова М. Г.

(Фамилия И. О.)

Преподаватель

Барышникова М. Ю.

Оглавление

Цель работы.....	2
Условие задачи.....	3
Описание ТЗ.....	3
1. Описание входных данных.....	3
2. Описание выходных данных.....	4
3. Описание задачи, реализуемой в программе.....	4
4. Способ обращения к программе.....	5
5. Описание возможных аварийных ситуаций и ошибок пользователя.....	5
Описание внутренних структур данных.....	5
Описание алгоритма.....	6
Тестовые данные.....	6
1. Позитивные тесты.....	6
2. Негативные тесты.....	9
Оценка эффективности.....	10
Выводы.....	11
*Ответы на контрольные вопросы.....	11

Цель работы

Получить навыки применения двоичных деревьев, реализовать основные операции над деревьями: обход деревьев, включение, исключение и поиск узлов.

Условие задачи

Ввести значения переменных: от А до I. Построить и вывести на экран бинарное дерево следующего выражения: $A + (B * (C + (D * (E + F) - (G - H)) + I))$.

Написать процедуры постфиксного, инфиксного и префиксного обхода дерева и вывести соответствующие выражения на экран. Подсчитать результат. Используя «польскую» запись, ввести данное выражение в стек. Сравнить время вычисления выражения с использованием дерева и стека.

Описание ТЗ

1. Описание входных данных

Исходными данными, хранящимися в стеке, являются вещественные числа.

Ввод данных происходит с помощью выбора 0-17 пункта в меню.

Действия :

С деревом выражения :

1. Ввод коэффициентов
2. Запись выражения в дерево
3. Вычисление выражения с помощью дерева
4. Вычисление выражения с помощью стека
5. Сравнение вычисления выражения
- 6-8. Префиксный/инфиксный/постфиксный обход дерева
9. Вывод дерева в файл

С ддп :

10. Создание дерева
11. Поиск значения
12. Добавление узла
13. Удаление узла
- 14-16. Префиксный/инфиксный/постфиксный обход дерева
17. Вывод дерева в файл
0. Выход

Введите выбранный пункт:

Некорректный ввод вызывает завершение программы с ненулевым кодом возврата.

Ограничения:

- Элементы дерева выражения:
 - **Данные:** целые попарно различные числа **от -65536 до 65535**; знак операции;
- Элементы двоичного дерева поиска:
 - **Данные:** целые попарно различные числа **от -65536 до 65535**;
- Пункт меню: целое число **от 0 до 10**;

2. Описание выходных данных

Опции 1-9:

В результате выполнения программы будет сформировано дерево выражения, переведено выражение в постфиксную форму, произведено сравнение времени вычисления выражения с помощью стека и дерева.

Опция 10-17:

В результате выполнения программы будет сформировано двоичное дерево поиска.

3. Описание задачи, реализуемой в программе

Программа реализует:

- добавление элемента в ДДП
- поиск элемента в ДДП
- удаление элемента из ДДП
- префиксный/инфиксный/постфиксный обход дерева
- вычисление выражения разными способами
- измерение времени вычисления выражения с помощью дерева выражения и стека
- вывод дерева в файл

4. Способ обращения к программе

Обращение к программе происходит через консоль, путём запуска *.exe файла.

5. Описание возможных аварийных ситуаций и ошибок пользователя

Аварийная ситуация	Код завершения	Сообщение
Неверный размер стека	1	Wrong capacity :(
Дерево не инициализировано	2	Пустое дерево!
Неудачное выделение памяти	5	Ошибки выделения памяти!
Неверный выбор пункта меню	6	Неверный пункт меню!
Некорректные данные	7	Неверное значение!
Добавление в дерево уже имеющегося значения	8	Одинаковые значения в дереве!

Описание внутренних структур данных

```
// структура для описания узла дерева
typedef struct node_tree node_tree_t;

struct node_tree
{
    char *data; // указатель на данные
    int index; // индекс узла
    node_tree_t *left; // указатель на правого потомка
    node_tree_t *right; // указатель на левого потомка
};

// структура для описания стека, хранящего адреса узлов дерева
typedef struct stack_tree stack_tree_t;

struct stack_tree
{
    node_tree_t *data; // указатель на данные
    stack_tree_t *next; // указатель на следующий элемент
};
```

```
// структура для описания элемента стека
typedef struct node_stack node_stack_t;
struct node_stack
{
    char *data;          // указатель на данные
    node_stack_t *next; // указатель на следующий элемент
};

// указатель функции
typedef void (*ptr_action_t)(node_tree_t *, void *);
```

Описание алгоритма

1. Выводится меню данной программы
2. Пользователь выбирает пункт меню
3. Выход из программы осуществляется в случае ошибки или пункта меню - «0»

Тестовые данные

1. Позитивные тесты

№	Наименование теста	Пользовательский ввод	Вывод
1.	Ввод к-тов выражения	1 0 1 2 3 4 5 6 7 8	1 2 3 4 5 6 + * 7 8 - - + 9 + * +
2.	Запись выражения в дерево	1 0 1 2 3 4	Выражение записано в дерево!

		5 6 7 8 2	
3.	Ввод к-тов и вычисление выражения с помощью стека	1 0 1 2 3 4 5 6 7 8 3	Вычисленное значение с помощью дерева: 115
4.	Ввод к-тов и вычисление выражения с помощью дерева	1 0 1 2 3 4 5 6 7 8 4	Вычисленное значение с помощью дерева: 115
5.	Инфиксный обход инициализированного дерева	1 0 1 2 3 4 5 6 7 8 7	Инфиксный обход: $(1 + (2 * ((3 + ((4 * (5 + 6)) - (7 - 8))) + 9)))$

6.	Префиксный обход инициализированного дерева	1 0 1 2 3 4 5 6 7 8 6	Префиксный обход: + 1 * 2 + + 3 - * 4 + 5 6 - 7 8 9
7.	Постфиксный обход инициализированного дерева	1 0 1 2 3 4 5 6 7 8 8	Постфиксный обход: 1 2 3 4 5 6 + * 7 8 - - + 9 + * +
8.	Создание ДДП	10 3 1 2 3	ДДП создано!
9.	Добавление элемента в ДДП	10 3 1 2 3 12 4	Введенное значение добавлено в дерево!
10.	Удаление элемента из ДДП	10 3 1 2	Введенное значение удалено!

		3 13 1	
11.	Поиск элемента в ДДП	10 3 1 2 3 11 1	Введенное значение найдено: 1
12.	Поиск элемента в ДДП	10 3 1 2 3 11 12	Введенное значение не найдено!

2. Негативные тесты

№	Наименование теста	Пользовательский ввод	Вывод
1.	Неверно выбранный пункт меню	18	Неверный пункт меню!
2.	Инициализация дерева без введенных к-тов	2	Неверное значение!
3.	Вычисление выражения без введенных к-тов	3	Пустое дерево!
4.	Префиксный обход не инициализированного дерева	6	Пустое дерево!
5.	Инфиксный обход не инициализированного дерева	7	Пустое дерево!

6.	Постфиксный обход не инициализированного дерева	8	Пустое дерево!
7.	Ввод одинаковых значений в дерево	10 5 1 1 1 1 1	Неверное значение!
8.	Удаление узла из пустого дерева	10 3 1 2 3 13 1 13 2 13 3 13	Пустое дерево!
9.	Вывод пустого дерева в файл	17	Пустое дерево!

Оценка эффективности

Были проведены замеры времени по вычислению значения следующих выражений:

1. $15+(2*(3+(4*(5+6)-(7-8))+9))$

2. $16*(2-3*4-(5*6*7+8*9)-(10+11-(12+13)))$

$$3. 17*(2*(3*(4-5)-6)-7)+8*9-10-11+12$$

Результаты замеров представлены в виде таблицы:

Для более наглядной разницы в полученных результатах, каждое вычисление выражения производилось 100 раз.

	Стек		Дерево	
	Время(мкс.)	Значение	Время(мкс.)	Значение
1. $15+(2*(3+(4*(5+6)-(7-8))+9))$	965	129	807	129
2. $16*(2-3*4-(5*6*7+8*9)-(10+11-(12+13)))$	1228	-4608	1219	-4608
3. $17*(2*(3*(4-5)-6)-7)+8*9-10-11+12$	1254	-362	1085	-362

Выводы

В ходе работы были изучены способы и алгоритмы работы с деревьями, а также проведены замеры времени по вычислению выражений разными способами. Из них можно сделать вывод о том, что время выполнения программы при разных реализациях практически совпадает, то есть рекурсивный алгоритм (для бинарного дерева), имеет такую же временную сложность, что и последовательный алгоритм для стека.

*Ответы на контрольные вопросы

1. Что такое дерево?

Дерево – это нелинейная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим». Дерево с базовым типом T определяется рекурсивно либо как пустая структура (пустое дерево), либо как узел типа T с конечным числом древовидных структур этого типа, называемых поддеревьями.

2. Как выделяется память под представление деревьев?

Способ выделения памяти под деревья определяется способом их представления в программе. С помощью матрицы или списка может быть реализована таблица связей с предками или связный список сыновей.

Целесообразно использовать списки для упрощенной работы с данными, когда элементы требуется добавлять и удалять, т.е. выделять память под каждый элемент отдельно по мере необходимости. При реализации матрицей память выделяется статически.

3. Какие бывают типы деревьев?

N-арное дерево, двоичное дерево поиска, АВЛ-дерево, красно-черное дерево, и т.д.

4 Какие стандартные операции возможны над деревьями?

Обход дерева, поиск по дереву, включение в дерево, исключение из дерева.

5. Что такое дерево двоичного поиска?

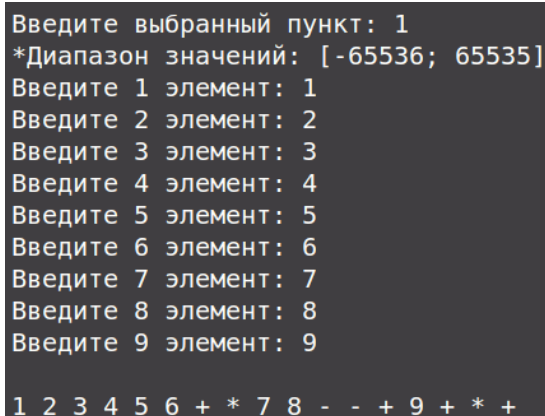
Дерево двоичного поиска – это такое дерево, в котором все левые потомки(значения) моложе(меньше) предка, а все правые – старше(больше). Это свойство называется характеристическим свойством дерева двоичного поиска и выполняется для любого узла, включая корень.

С учетом этого свойства поиск узла в двоичном дереве поиска можно осуществить, двигаясь от корня в левое или правое поддерево в зависимости от значения ключа поддерева.

***Приложение**

Ниже расположены скриншоты для проверки корректности работы программы.

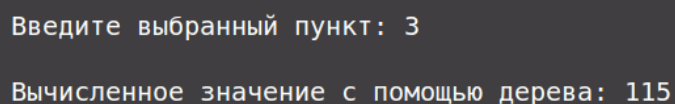
1. Ввод к-тов выражения



```
Введите выбранный пункт: 1
*Диапазон значений: [-65536; 65535]
Введите 1 элемент: 1
Введите 2 элемент: 2
Введите 3 элемент: 3
Введите 4 элемент: 4
Введите 5 элемент: 5
Введите 6 элемент: 6
Введите 7 элемент: 7
Введите 8 элемент: 8
Введите 9 элемент: 9

1 2 3 4 5 6 + * 7 8 - - + 9 + * +
```

2. Вычисление выражения деревом



```
Введите выбранный пункт: 3

Вычисленное значение с помощью дерева: 115
```

3. Вычисление выражения стеком

Введите выбранный пункт: 4

Вычисленное значение с помощью стека: 115

4. Префиксный обход

Введите выбранный пункт: 6

Префиксный обход: + 1 * 2 + + 3 - * 4 + 5 6 - 7 8 9

5. Инфиксный обход

Введите выбранный пункт: 7

Инфиксный обход: (1 + (2 * ((3 + ((4 * (5 + 6)) - (7 - 8)) + 9)))

6. Постфиксный обход

Введите выбранный пункт: 8

Постфиксный обход: 1 2 3 4 5 6 + * 7 8 - - + 9 + * +

7. Запись дерева в файл

