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 «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

Студент **Паламарчук А.Н.**

Группа **ИУ7-33Б**

Предмет **Типы и структуры данных**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Паламарчук А.Н.**

Преподаватель _____ **Никульшина Т. А.**

Преподаватель _____ **Барышникова М. Ю.**

2023 г.

Условие задачи
Построить дерево в соответствии со своим вариантом задания. Вывести его на экран в виде дерева. Реализовать префиксный, инфиксный и постфиксный обходы дерева. Сравнить эффективность алгоритмов вычисления выражения.

Техническое задание
Ввести значения переменных: от A 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.

Входные данные

- Команда (число) из меню программы (см. меню программы) – обозначение необходимой операции.

Выходные данные

- Таблицы эффективности
- Файл в DOT формате
- Файл .png с деревом
- Выражение, полученное префиксным обходом
- Выражение, полученное инфиксным обходом
- Выражение, полученное постфиксным обходом
- Результат вычисления выражения

Возможные аварийные ситуации

- Некорректный исходный файл
- Ошибка открытия файла
- Отсутствие исходного файла
- Некорректная команда

Описание внутренних структур данных

```
typedef struct tree_node
{
    const char *name;
    int val;
    // родитель
```

```
    struct tree_node *parent;
    // меньшие
    struct tree_node *left;
    // большие
    struct tree_node *right;
} tree_node_t;
```

Константы

```
#define COUNT_TESTS 100
#define FILE_DOT "graph.gv"
#define FILE_PNG "graph.png"
```

Используемые функции

```
int fill_tree(struct val_nodes *tree);
```

Заполнение дерева значениями пользователя

```
void export_to_dot(FILE *f, const char *tree_name, struct tree_node
*tree);
```

Перевод дерева в DOT формат

```
void apply_pre(tree_node_t *tree, void (*f)(tree_node_t*, void*), void
*arg);
```

Префиксный обход дерева

```
void apply_in(tree_node_t *tree, void (*f)(tree_node_t*, void*), void
*arg);
```

Инфиксный обход дерева

```
void apply_post(tree_node_t *tree, void (*f)(tree_node_t*, void*), void
*arg);
```

Постфиксный обход дерева

```
void print_name_node(tree_node_t *a, void *b);
```

Распечатывает название узла

```
int evaluate_postfix_expression(tree_node_t *node);
```

Вычисляет значение выражения

```
int cmp_tree_vs_stack(struct val_nodes tree);
```

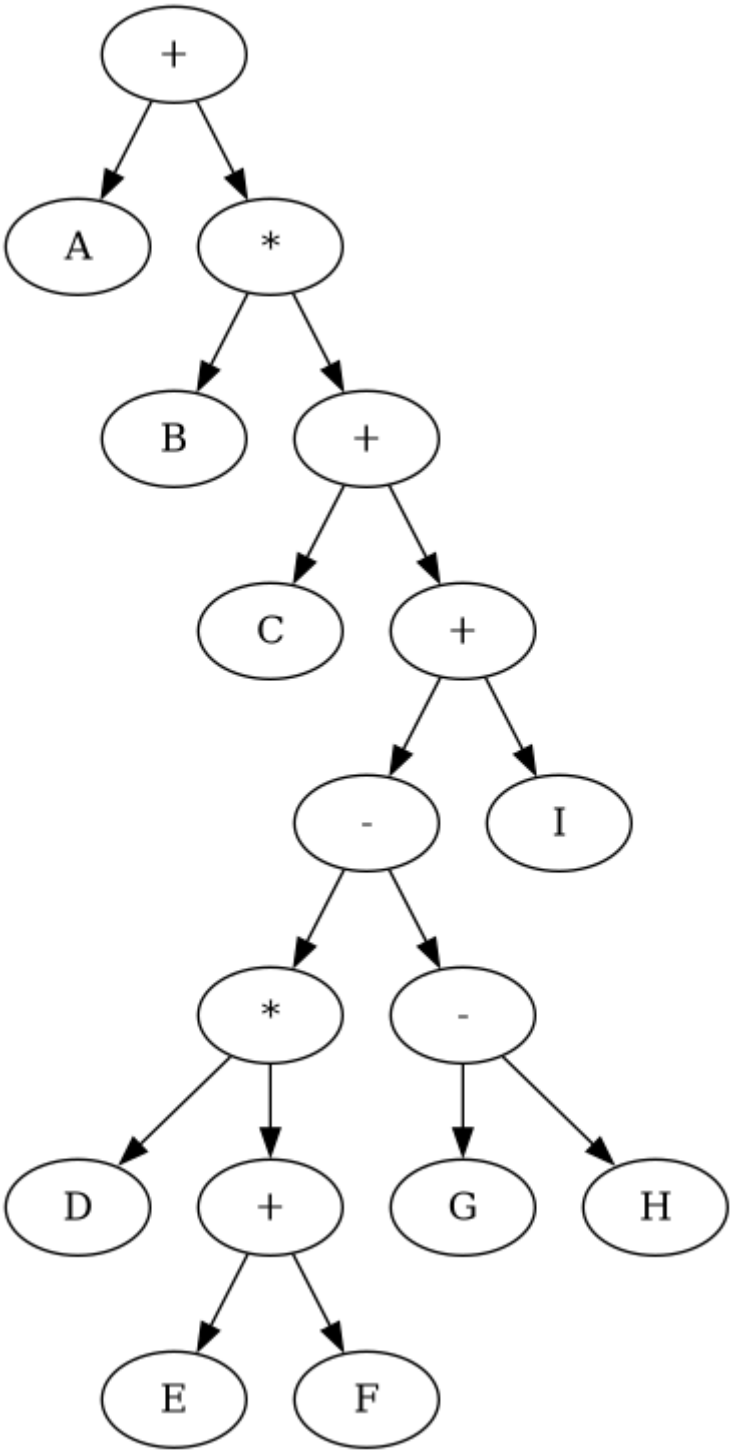
Выводит таблицы сравнения

Меню программы

Программа обрабатывает нужную команду:

```
Комманды:
1 - Ввести значения переменных: от A до I
2 - Вывести бинарное дерево выражения A + (B * (C + (D * (E + F) - (G - H)) + I))
3 - Вывести выражение (префиксный обход)
4 - Вывести выражение (инфиксный обход)
5 - Вывести выражение (постфиксный обход)
6 - Вычислить значение выражения
7 - Сравнить время вычисления выражения с использованием дерева и стека
0 - Завершить работу программы
```

Пример работы программы



```
Комманды:
1 - Ввести значения переменных: от A до I
2 - Вывести бинарное дерево выражения A + (B * (C + (D * (E + F) - (G - H)) + I))
3 - Вывести выражение (префиксный обход)
4 - Вывести выражение (инфиксный обход)
5 - Вывести выражение (постфиксный обход)
6 - Вычислить значение выражения
7 - Сравнить время вычисления выражения с использованием дерева и стека
0 - Завершить работу программы

3
+A*B+C+-*D+EF-GHI
```

```
Комманды:
1 - Ввести значения переменных: от A до I
2 - Вывести бинарное дерево выражения A + (B * (C + (D * (E + F) - (G - H)) + I))
3 - Вывести выражение (префиксный обход)
4 - Вывести выражение (инфиксный обход)
5 - Вывести выражение (постфиксный обход)
6 - Вычислить значение выражения
7 - Сравнить время вычисления выражения с использованием дерева и стека
0 - Завершить работу программы

4
A+B*C+D*E+F-G-H+I
```

```
Комманды:
1 - Ввести значения переменных: от A до I
2 - Вывести бинарное дерево выражения A + (B * (C + (D * (E + F) - (G - H)) + I))
3 - Вывести выражение (префиксный обход)
4 - Вывести выражение (инфиксный обход)
5 - Вывести выражение (постфиксный обход)
6 - Вычислить значение выражения
7 - Сравнить время вычисления выражения с использованием дерева и стека
0 - Завершить работу программы

5
ABCDEF+*GH--I++*+
```

```
Комманды:
1 - Ввести значения переменных: от A до I
2 - Вывести бинарное дерево выражения A + (B * (C + (D * (E + F) - (G - H)) + I))
3 - Вывести выражение (префиксный обход)
4 - Вывести выражение (инфиксный обход)
5 - Вывести выражение (постфиксный обход)
6 - Вычислить значение выражения
7 - Сравнить время вычисления выражения с использованием дерева и стека
0 - Завершить работу программы

1
Введите значение A:
1
Введите значение B:
2
Введите значение C:
3
Введите значение D:
4
Введите значение E:
5
Введите значение F:
6
Введите значение G:
7
Введите значение H:
8
Введите значение I:
9

Комманды:
1 - Ввести значения переменных: от A до I
2 - Вывести бинарное дерево выражения A + (B * (C + (D * (E + F) - (G - H)) + I))
3 - Вывести выражение (префиксный обход)
4 - Вывести выражение (инфиксный обход)
5 - Вывести выражение (постфиксный обход)
6 - Вычислить значение выражения
7 - Сравнить время вычисления выражения с использованием дерева и стека
0 - Завершить работу программы

6
Результат вычисления выражения: 115
```

Производительность

```
Тестирование проведено 100 раз
Время указано в наносекундах, затраты памяти в байтах

TIME
---------------------------------
|          Stack|           Tree|
---------------------------------
|            380|            220|
---------------------------------

MEMORY
---------------------------------
|          Stack|           Tree|
---------------------------------
|            148|            680|
---------------------------------
```

Выводы по проделанной работе

Деревья следует использовать в случаях, когда нужен быстрый поиск элементов (бинарное дерево), а также для хранения данных в иерархической форме. Кроме того, имеется существенный недостаток, который заключается в том, что при удалении требуется перестроение дерева, которое выполняется рекурсивно и требует определенных ресурсов.
В моей реализации вычисление значения выражения с помощью дерева занимает меньше времени, чем вычисление при помощи стека.

Контрольные вопросы

1. Что такое дерево? Как выделяется память под представление деревьев?

Дерево – это нелинейная структура данных, используемая для представления иерархических связей (один к нескольким).

Динамически под каждый узел связного списка.

2. Какие бывают типы деревьев?

- *Двоичное дерево* - иерархическая структура данных, в которой каждый узел имеет не более двух потомков.
- *Красно-чёрное дерево* - один из видов самобалансирующихся двоичных деревьев поиска, гарантирующих логарифмический рост высоты дерева от числа узлов и позволяющее быстро выполнять основные операции дерева поиска: добавление, удаление и поиск узла.

- *АВЛ-дерево* - сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

3. Какие стандартные операции возможны над деревьями?

Обход дерева, включение, исключение и поиск узлов

4. Что такое дерево двоичного поиска?

Двоичное дерево, для которого выполняются следующие дополнительные условия.

- оба поддерева - левое и правое - являются двоичными деревьями поиска;
- у всех узлов *левого* поддерева произвольного узла X значения ключей данных *меньше либо равны*, нежели значение ключа данных самого узла X;
- у всех узлов *правого* поддерева произвольного узла X значения ключей данных *больше*, нежели значение ключа данных самого узла X.