Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №6 з дисципліни
«Алгоритми та структури даних 2. Структури даних»

«Деревовидні структури даних»
Варіант 3

Виконав студент ІП-15, Борисик Владислав Тарасович
(шифр, прізвище, ім'я, по батькові)

Перевірив Соколовський Владислав Володимирович
(прізвище, ім'я, по батькові)

Київ 2022

Мета лабораторної роботи

Мета роботи – вивчити основні підходи формалізації та імплементації алгоритмів побудови та обробки базових деревовидних структур даних.

Завдання

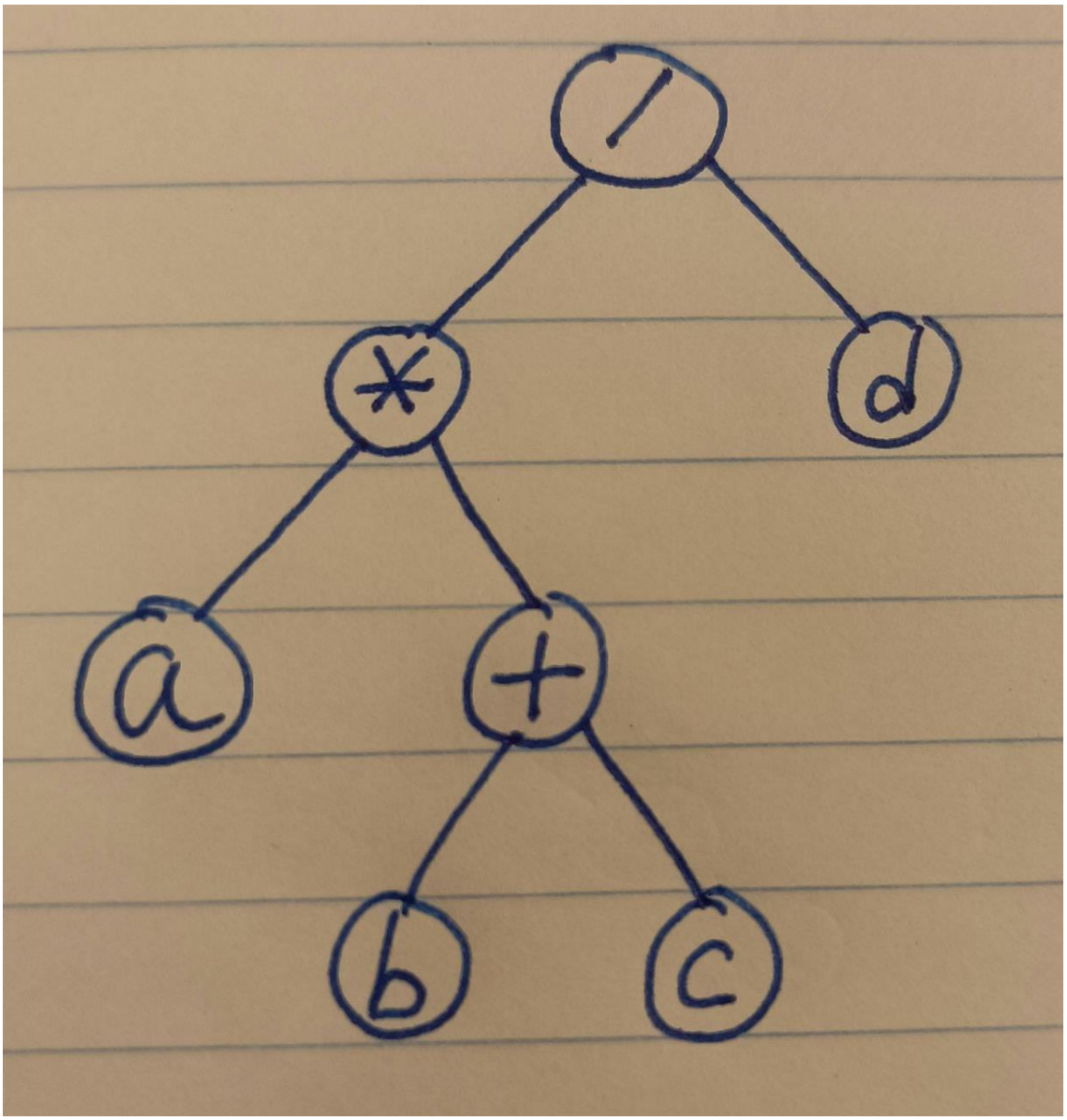
3. Побудувати дерево, що відображає формулу (a*(b+c))/d, де коренем дерева та його підкоренями є операції "*, +, -, /", а листками - змінні a, b, c, d. Надрукувати відповідне дерево/

Для виконання цієї лабораторної роботи я буду використовувати бінарне дерево виразів.

Реалізація дерева

Для представлення формули (a*(b+c))/d використаємо бінарне дерево виразів. В ньому операції *, +, / будуть представлені коренем і підкоренями дерева, а листками будуть змінні a, b, c і d.

Ручна реалізація дерева:



Псевдокод алгоритму

```
def __is_empty(self):
    Якщо self.root == None:
        повернути True
    Інакше:
        повернути False
```

```
def __evaluate_operator(self, leaf, left_sum, right_sum):
    # якщо підкорінь має значення '+', то додаємо ліву і праву суму значень
    Якщо leaf.value == '+':
        sum = left_sum + right_sum
        повернути sum
    # якщо підкорінь має значення '-', то віднімаємо ліву і праву суму значень
    Інакше якщо leaf.value == '-':
        sub = left_sum - right_sum
        повернути sub
    # якщо підкорінь має значення '*', то множимо ліву і праву суму значень
    Інакше якщо leaf.value == '*':
        mult = left_sum * right_sum
        повернути mult
    # якщо підкорінь має значення '/', то ділимо ліву і праву суму значень
    Інакше:
        div = left_sum / right_sum
        повернути div
```

```
def calculate_tree(self, leaf: Leaf or None):
    # якщо користувач не задав вузол, то вузлом буде корінь дерева
    Якщо leaf == None:
        leaf = self.root

    # якщо дерево пусте, то повертаємо 0
    Якщо self.__is_empty():
            повернути 0
    # якщо ж дерево не пусте
    Інакше:
        # якщо лівий листок вузла порожній і правий листок вузла порожній
        # (тобто вузол є листком (має нульовий степінь))
        Якщо leaf.left_leaf == None і leaf.right_leaf == None:
            # повертаємо значення вузла
            повернути leaf.value

        # рекурсивно рахуємо суму лівих листків вузла
        left_sum = self.calculate_tree(leaf.left_leaf)

        # рекурсивно рахуємо суму правих листків вузла
        right_sum = self.calculate_tree(leaf.right_leaf)

        # рахуємо загальну суму
        sum = self.__evaluate_operator(leaf, left_sum, right_sum)

        return sum
```

Програмна реалізація алгоритму на мові Python

```
def __is_empty(self):
    if self.root is None:
        return True
    else:
        return False

def __evaluate_operator(self, leaf, left_sum, right_sum):
    # якщо підкорінь має значення '+', то додаємо ліву і праву суму значень
    if leaf.value == '+':
        sum = left_sum + right_sum
        return sum
    # якщо підкорінь має значення '-', то віднімаємо ліву і праву суму значень
    elif leaf.value == '-':
        sub = left_sum - right_sum
        return sub
    # якщо підкорінь має значення '*', то множимо ліву і праву суму значень
    elif leaf.value == '*':
        mult = left_sum * right_sum
        return mult
    # якщо підкорінь має значення '/', то ділимо ліву і праву суму значень
    else:
        div = left_sum / right_sum
        return div

def calculate_tree(self, leaf: Leaf or None):
    # якщо користувач не задав вузол, то вузлом буде корінь дерева
    if leaf is None:
        leaf = self.root

    # якщо дерево пусте, то повертаємо 0
    if self.__is_empty():
        return 0
    # якщо ж дерево не пусте
    else:
        # якщо лівий листок вузла порожній і правий листок вузла порожній
        # (тобто вузол є листком (має нульовий степінь))
        if leaf.left_leaf is None and leaf.right_leaf is None:
            # повертаємо значення вузла
            return leaf.value

        # рекурсивно рахуємо суму лівих листків вузла
        left_sum = self.calculate_tree(leaf.left_leaf)

        # рекурсивно рахуємо суму правих листків вузла
        right_sum = self.calculate_tree(leaf.right_leaf)

        # рахуємо загальну суму
        sum = self.__evaluate_operator(leaf, left_sum, right_sum)

        return sum

def print_tree(self, leaf: Leaf, depth = 0):
    # якщо дерево пусте, то повертаємо 0
```

```
if self.__is_empty():
    return 0
# якщо ж дерево не пусте
else:
    # якщо вузол не пустий
    if leaf is not None:
        # рекурсивно йдемо до лівого листка вузла
        self.print_tree(leaf.left_leaf, depth + 1)
        # виводимо значення в консоль
        print(f"{' ' * 4 * depth} -> {leaf.value}")
        # рекурсивно йдемо до правого листка вузла
        self.print_tree(leaf.right_leaf, depth + 1)
```

Вихідний код

main.py:

```
from functions import *

a, b, c, d = capture_values()

tree = build_tree(a,b,c,d)

tree_value = tree.calculate_tree(None)
print(f"Tree value: {tree_value}\n")

print("Tree representation:\n")
tree_root = tree.get_root()
tree.print_tree(tree_root)
```

classes.py:

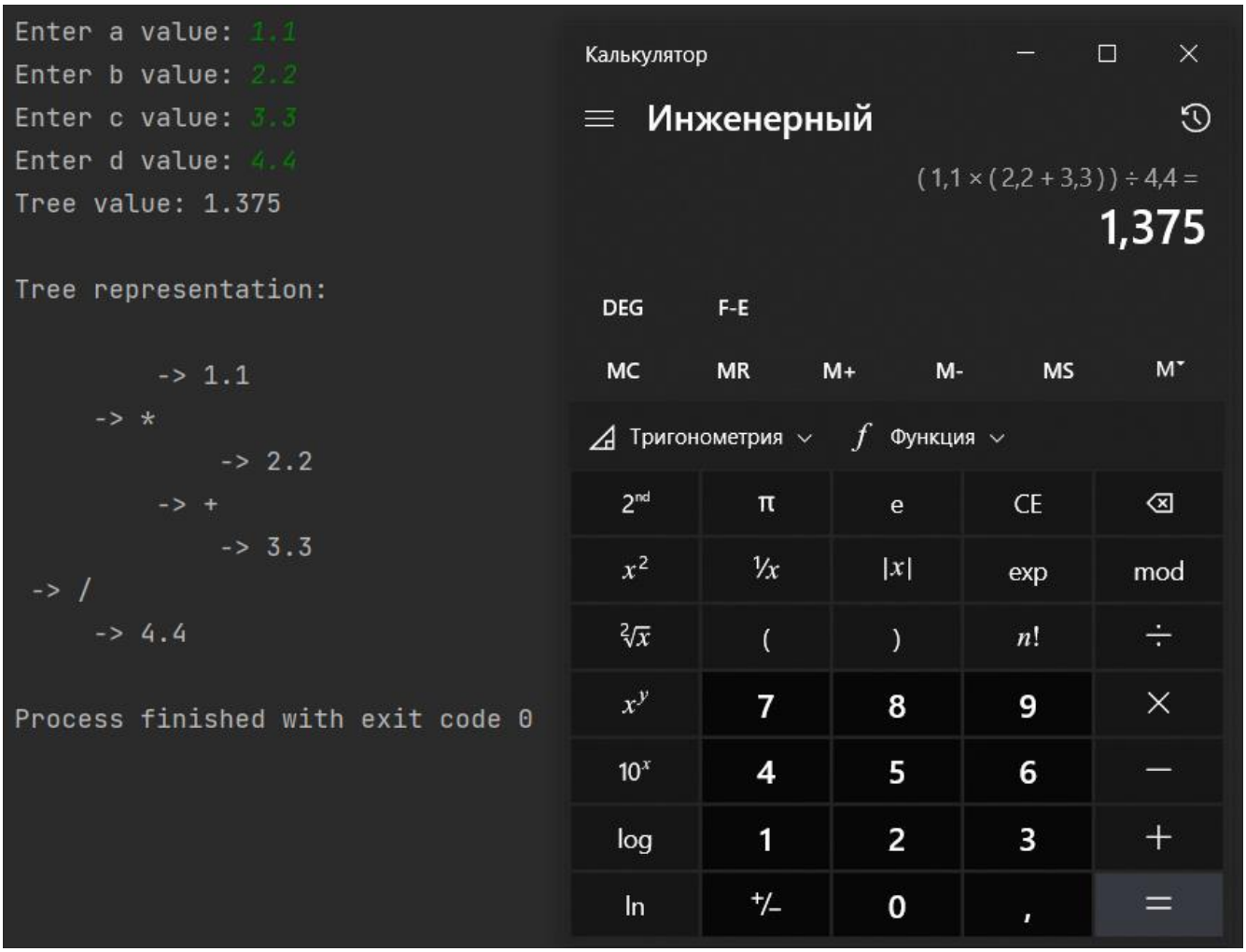
```
class Leaf:
    def __init__(self, value):
        self.left_leaf = None
        self.right_leaf = None

        self.value = value


class Tree:
    def __init__(self, root: Leaf):
        self.root = root

    def get_root(self):
        return self.root

    def __is_empty(self):
        if self.root is None:
            return True
        else:
            return False

    def __evaluate_operator(self, leaf, left_sum, right_sum):
        # якщо підкорінь має значення '+', то додаємо ліву і праву суму значень
        if leaf.value == '+':
            sum = left_sum + right_sum
            return sum
        # якщо підкорінь має значення '-', то віднімаємо ліву і праву суму значень
        elif leaf.value == '-':
            sub = left_sum - right_sum
            return sub
        # якщо підкорінь має значення '*', то множимо ліву і праву суму значень
        elif leaf.value == '*':
            mult = left_sum * right_sum
            return mult
        # якщо підкорінь має значення '/', то ділимо ліву і праву суму значень
        else:
            div = left_sum / right_sum
            return div

    def calculate_tree(self, leaf: Leaf or None):
        # якщо користувач не задав вузол, то вузлом буде корінь дерева
        if leaf is None:
            leaf = self.root

        # якщо дерево пусте, то повертаємо 0
        if self.__is_empty():
            return 0
        # якщо ж дерево не пусте
        else:
            # якщо лівий листок вузла порожній і правий листок вузла порожній
            # (тобто вузол є листком (має нульовий степінь))
            if leaf.left_leaf is None and leaf.right_leaf is None:
                # повертаємо значення вузла
```

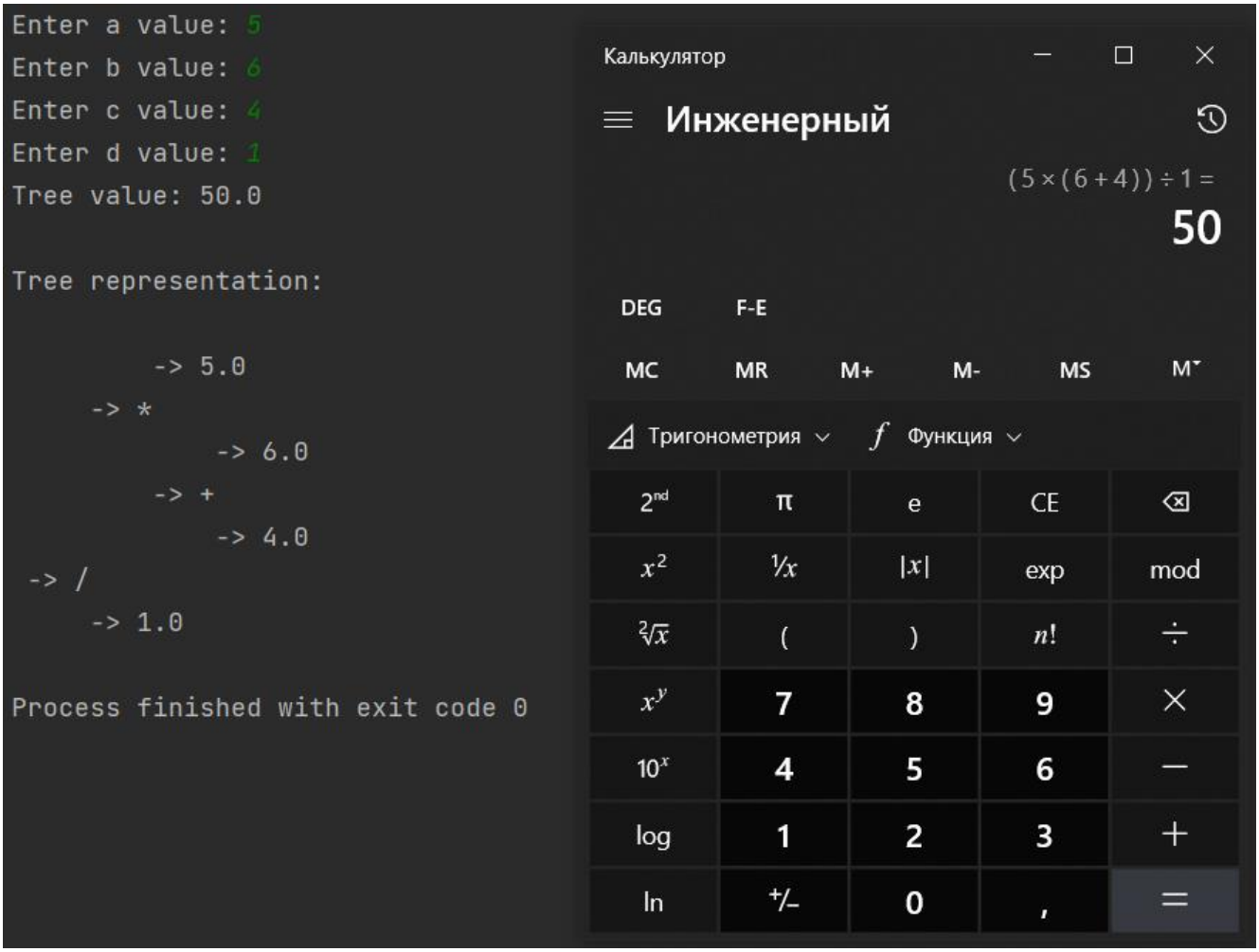
```
            return leaf.value

        # рекурсивно рахуємо суму лівих листків вузла
        left_sum = self.calculate_tree(leaf.left_leaf)

        # рекурсивно рахуємо суму правих листків вузла
        right_sum = self.calculate_tree(leaf.right_leaf)

        # рахуємо загальну суму
        sum = self.__evaluate_operator(leaf, left_sum, right_sum)

        return sum

def print_tree(self, leaf: Leaf, depth = 0):
    # якщо дерево пусте, то повертаємо 0
    if self.__is_empty():
        return 0
    # якщо ж дерево не пусте
    else:
        # якщо вузол не пустий
        if leaf is not None:
            # рекурсивно йдемо до лівого листка вузла
            self.print_tree(leaf.left_leaf, depth + 1)
            # виводимо значення в консоль
            print(f"{' ' * 4 * depth} -> {leaf.value}")
            # рекурсивно йдемо до правого листка вузла
            self.print_tree(leaf.right_leaf, depth + 1)
```

functions.py

```python
from classes import *


def capture_values():
    a = float(input("Enter a value: "))
    b = float(input("Enter b value: "))
    c = float(input("Enter c value: "))
    d = float(input("Enter d value: "))

    return a, b, c, d


def build_tree(a: float, b: float, c: float, d: float):
    # перший вузол (корінь): ділення
    root = Leaf("/")

    # ініціалізуємо дерево з цим коренем
    tree = Tree(root)

    # праве значення кореня: d
    root.right_leaf = Leaf(d)
    # ліве значення кореня: множення
    root.left_leaf = Leaf("*")

    # ліве значення попередньої вершини: a
    root.left_leaf.left_leaf = Leaf(a)
    # праве значення попередньої вершини: +
    root.left_leaf.right_leaf = Leaf("+")

    # ліве значення попередньої вершини: b
    root.left_leaf.right_leaf.left_leaf = Leaf(b)
    # праве значення попередньої вершини: c
    root.left_leaf.right_leaf.right_leaf = Leaf(c)

    return tree
```

Приклад роботи програми





Бачимо, що у всіх випадках програма коректно порахувала значення дерева і надрукувала його, отже, вона працює правильно.

Висновок

Під час виконання цієї лабораторної роботи я вивчив основні підходи формалізації та імплементації алгоритмів побудови та обробки базових деревовидних структур даних. В результаті виконання лабораторної роботи я отримав програму, яка будує дерево, що відображає формулу (a*(b+c))/d і друкує відповідне дерево.