

## Практическая работа №6

**Тема:** «Структура данных дерево».

**Цель работы:** изучить структуру данных дерево, реализовать ее программно.

Ход работы.

Построить дерево арифметического выражения, заданного в ППЗ. Операнды - целочисленные константы.

Операции «+», «-», «\*», «/».

Вывести арифметическое выражение в ОПЗ.

Вычислить значение дерева и вывести результат работы в виде:

<операнд><операции><операнд>=<значение>

Необходимо реализовать следующее дерево, диаграмма деятельности для конструктора представлена на рисунке 1, структурный код дерева представлен на рисунке 2.

					АиСД.09.03.02.020000 ПР			
Изм	Лист	№ докум.	Подпись	Дата	Практическая работа №6 «Структура данных дерево»	Литера	Лист	Листов
Разраб.		Веретельников А. С.					1	
Провер.		Берёза А. Н.						
						ИСТ-Тб21		
Н. контр.								
Утверд								

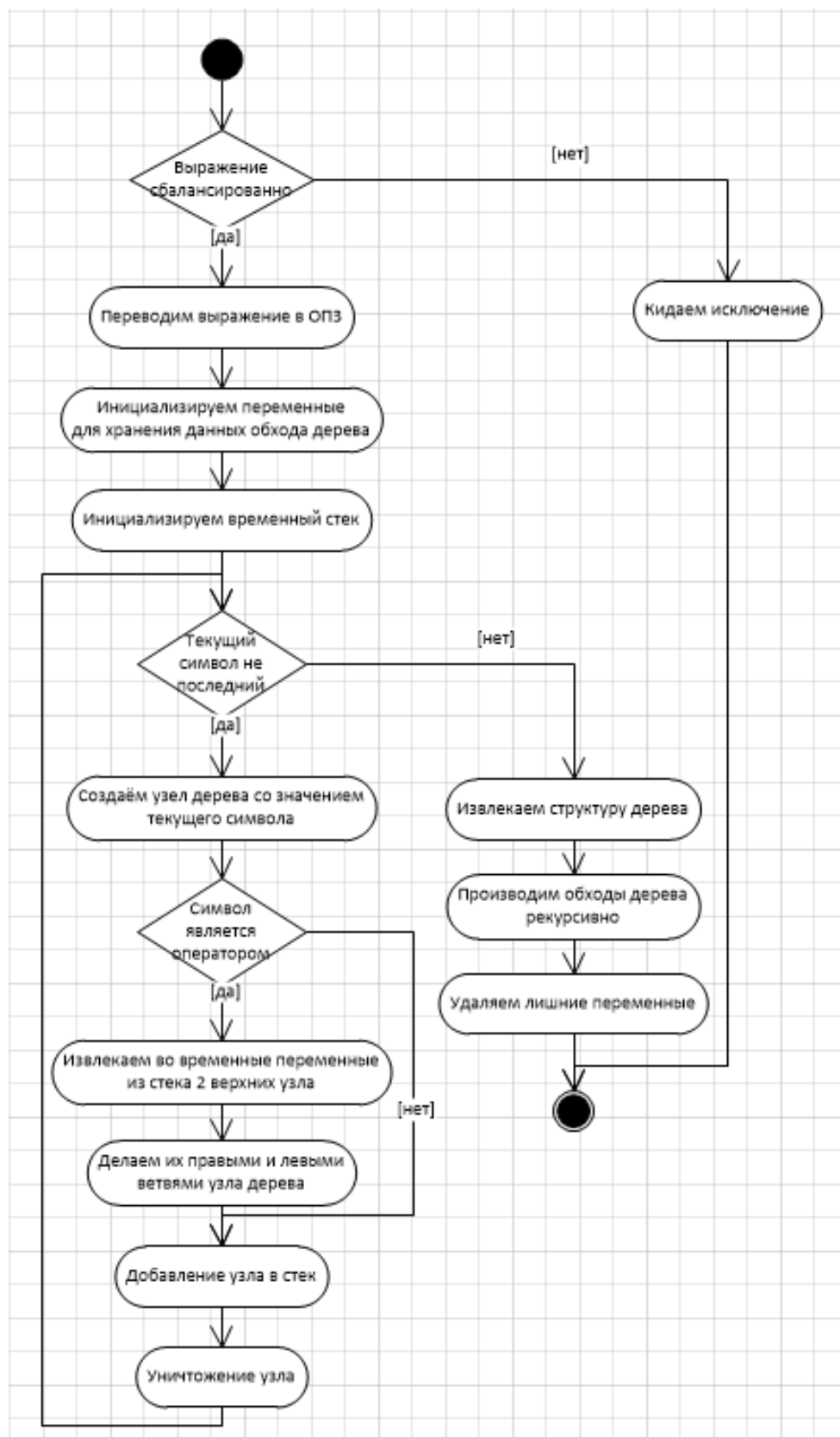


Рисунок 1 – Диаграмма деятельности конструктора класса.

```

from dataclasses import dataclass

@dataclass()
class NodeTree:
    value = None
    left = None
    right = None

class TreeExpression():
    def __init__(self, normal):
        if self.__check_exp(normal):
            postfix = self.__from_normal_exp_to_rpn_convert(normal)
            self.original_expression_numbers = self.__original_expression(normal)
        else:
            raise ("Expression error")
        self.tree_traversal_nlr = ''
        self.tree_traversal_lrn = ''
        self.tree_traversal_lnr_with_number = ''
        lis = []
        for char in postfix:
            t = NodeTree()
            t.value = char
            if self.__isOperator(char):
                t1 = lis.pop()
                t2 = lis.pop()
                t.right = t1
                t.left = t2
                lis.append(t)
            del t
        self.tree = lis.pop()
        self.tree_traversal_lnr = ''
        self.__lnr(self.tree)
        self.__nlr(self.tree)
        self.__lrn(self.tree)
        self.__lnr_with_number(self.tree)
        del lis, t1, t2

    def get_tree(self):
        return self.tree

    def __isOperator(self, c):
        if c in ['+', '-', '*', '/', ')', '(']:
            return True
        else:
            return False

    def __str__(self):
        out = "LRN:\n\t" + self.tree_traversal_lrn + "\nNRL:\n\t" + self.tree_traversal_nlr
        out += "\nLNR:\n\t" + self.tree_traversal_lnr
        out += '\nExpression in numbers LNR without brackets\n\t' + self.tree_traversal_lnr_with_number
        out += '\nExpression original\n\t' + self.original_expression_numbers + '\n'
        return out

    def __lnr(self, tree):
        if tree is not None:
            self.__lnr(tree.left)
            self.tree_traversal_lnr += str(tree.value)
            self.__lnr(tree.right)

```

					АиСД.09.03.02.020000 ПР	Лист
						3
Изм	Лист	№ докум.	Подпись	Дата		

```

def __lnr_with_number(self, tree):
    if tree is not None:
        self.__lnr_with_number(tree.left)
        self.tree_traversal_lnr_with_number += str(self.__convert_from_letter_to_digit(tree.value))
        self.__lnr_with_number(tree.right)

def __convert_from_letter_to_digit(self, letter):
    if not self.__isOperator(letter):
        return ord(str(letter))
    else:
        return str(letter)

def __nlr(self, tree):
    if tree is not None:
        self.tree_traversal_nlr += str(tree.value)
        self.__nlr(tree.left)
        self.__nlr(tree.right)

def __lrn(self, tree):
    if tree is not None:
        self.__lrn(tree.left)
        self.__lrn(tree.right)
        self.tree_traversal_lrn += str(tree.value)

def __from_normal_exp_to_rpn_convert(self, fpn_exp):
    output = ''
    lis_sign = []
    for element in fpn_exp:
        if self.__isOperator(element):
            if len(lis_sign) == 0 or element == '(':
                lis_sign.append(element)

            elif element == ')':
                while lis_sign[len(lis_sign) - 1] != '(':
                    output += lis_sign.pop(len(lis_sign) - 1)
                lis_sign.pop(len(lis_sign) - 1)

            elif self.__priority(element) > self.__priority(lis_sign[len(lis_sign) - 1]):
                lis_sign.append(element)

            elif self.__priority(element) <= self.__priority(lis_sign[len(lis_sign) - 1]):
                if len(lis_sign) > 1:
                    while self.__priority(element) <= self.__priority(lis_sign[len(lis_sign) - 1]):
                        output += lis_sign.pop(len(lis_sign) - 1)
                else:
                    output += lis_sign.pop(len(lis_sign) - 1)
                lis_sign.append(element)

            else:
                output += element
        while len(lis_sign) != 0:
            output += lis_sign.pop(len(lis_sign) - 1)
    return output

def __check_exp(self, exp):
    count = 0
    for element in exp:

```

АиСД.09.03.02.020000 ПР					Лист
					4
Изм	Лист	№ докум.	Подпись	Дата	

```

        if element == '(':
            count += 1
        elif element == ')':
            count -= 1
            if count == -1:
                return False
    if count != 0:
        return False
    else:
        return True

def __priority(self, symbol):
    if symbol in ['*', '/']:
        return 3
    elif symbol in ['+', '-']:
        return 2
    elif symbol == '(':
        return 1

def __original_expression(self, expr):
    out = ''
    for c in expr:
        out += str(self.__convert_from_letter_to_digit(c))
    return out

def calc_exp_with_output(self):
    lis = []
    for element in self.tree_traversal_lrn:
        if not self.__isOperator(element):
            lis.insert(0, ord(element))
        else:
            l = lis.pop(0)
            r = lis.pop(0)
            if element == '+':
                lis.insert(0, (r + l))
                print(r, '+', l, '=', lis[0], end='\n', sep='')
            elif element == '-':
                lis.insert(0, (r - l))
                print(r, '-', l, '=', lis[0], end='\n', sep='')
            elif element == '*':
                lis.insert(0, (r * l))
                print(r, '*', l, '=', lis[0], end='\n', sep='')
            elif element == '/':
                lis.insert(0, (r / l))
                print(r, '/', l, '=', lis[0], end='\n', sep='')
    print('Total: ', lis[0], sep='')

norm = input('Enter expression')
print('Original expression', normal, sep='\n\t')
r = TreeExpression(normal)
print(r)
r.calc_exp_with_output()

```

Рисунок 2 – Дерево арифметического выражения.

**Вывод:** в ходе выполнения работы была изучена структура дерева и написана программная реализация дерева арифметического выражения.

					АиСД.09.03.02.020000 ПР	Лист
						5
Изм	Лист	№ докум.	Подпись	Дата		