

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ по**  
**лабораторной работе №2**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Рекурсивная обработка иерархических списков**

Студент гр. 8304

Самакаев Д.И.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2019

## **Вариант 21**

### **Цель работы.**

Изучить основные принципы работы иерархических списков и их обработки.

### **Постановка задачи.**

1. Изучив условия задачи разработать эффективный алгоритм обработки входных данных.

2. Сопоставить рекурсивное решение с итеративным решением задачи;

3. Сделать вывод о целесообразности и эффективности реализованного алгоритма решения задачи.

Пусть выражение (логическое, арифметическое, алгебраическое\*) представлено иерархическим списком. В выражение входят переменные, которые являются атомами списка. Операции представляются в префиксной форме ( (<операция> <аргументы> ) ), либо в постфиксной форме ( <аргументы> <операция> ). Аргументов может быть 1, 2 и более. Например (в префиксной форме): (+ a (\* b (- c))) или (OR a (AND b (NOT c))).

В задании даётся один из следующих вариантов требуемого действия с выражением: проверка синтаксической корректности, (преобразование), вычисление. Пример упрощения: (+ 0 (\* 1 (+ a b))) преобразуется в (+ a b). В задаче вычисления на входе дополнительно задаётся список значений переменных ( (x1 c1) (x2 c2) ... (xk ck) ), где  $x_i$  – переменная, а  $c_i$  – её значение (константа).

В индивидуальном задании указывается: тип выражения (возможно дополнительно – состав операций), вариант действия и форма записи.

\* - здесь примем такую терминологию: в арифметическое выражение входят операции +, -, \*, /, а в алгебраическое – +, -, \* и дополнительно некоторые функции.

21) арифметическое, вычисление, постфиксная форма

### **Описание алгоритма.**

Считываются строки выражения и значений переменных, после чего значения переменных заносятся в словарь по имени переменных. После этого,

при проходе по строке выражения, имена переменных заменяются на их значения. Следующим шагом создаётся и рекурсивно заполняется иерархический список. Последним действием программа производит вычисление значения выражения, рекурсивно проходя по ранее составленному иерархическому списку.

### **Спецификация программы.**

Программа предназначена для арифметического вычисления значения выражений, представленных в постфиксной форме.

Программа написана на языке C++. Входные данные подаются в виде строк текстового файла или консольным вводом.

### **Описание функций.**

1. `bool is_brackets_correct(std::string &expression)`

Определяет, правильно ли в строке expression расставлены скобки.

2. `void fill_map(std::string& values_str, std::map<std::string, int>& arguments_values_map)`

Находит в строке values\_str имена переменных и их значения, после чего заносит их в словарь arguments\_values\_map.

3. `bool create_node(std::string& expression, int &i, Node* &element)`
4. `bool rec_fill_branch(Node* &element, std::string& expression)`

Две взаимно рекурсивные функции, осуществляющие заполнение иерархического списка.

5. `void substitute(std::string& expression, std::map<std::string, int>& arguments_values_map)`

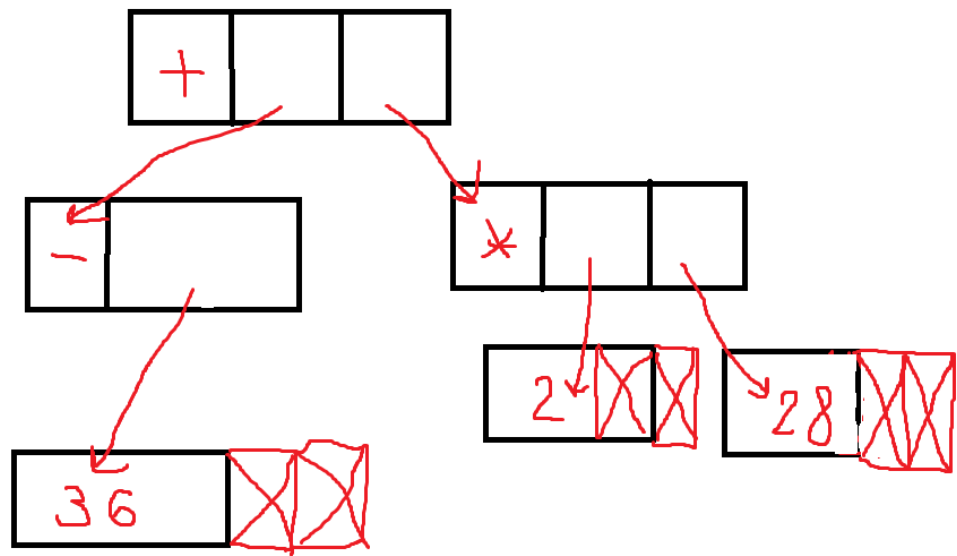
Меняет в строке expression аргументы на их значения.

6. `bool is_operand(char c)`

Проверяет, является ли символ знаком операции.

7. `int Node::calculate()`

Рекурсивно выполняет вычисление значения выражения.



### Вывод.

Была реализована программа, позволяющая вычислять значения арифметических выражений в постфиксной форме.

## Приложение.

### 1)Тестирование.

```

1 : ((a b + ) ci + )
((a wow)(b 222)(ci 313))
Incorrect arguement value!
You didn't set all values
2 : ((a b - ) (c d + ) *)
((a 228)(b 150)(c 114)(d 114))
17784
3 : (((a - ) b - ) c - ) d - )
((a 123)(b 222)(c 313)(d 424))
-1082
4 : (a - )
((a 123))
-123
5 : ((a b *) c *)
((a 123)(b 222)(c 313))
8546778
6 : ((a b + ) c /)
((a 123)(b 23)(c 3))
48
7 : ((a b /) c /)
((a 256)(b 16)(c 4))
4

```

((a b + ) ci + ) ((a wow)(b 222)(ci 313))	Incorrect arguement value! You didn't set all values
((a b - ) (c d + ) *) ((a 228)(b 150)(c 114)(d 114))	17784
(((a - ) b - ) c - ) d - ) ((a 123)(b 222)(c 313)(d 424))	-1082
(a - ) ((a 123))	-123
((a b *) c *) ((a 123)(b 222)(c 313))	8546778
((a b + ) c /) ((a 123)(b 23)(c 3))	48
((a b /) c /) ((a 256)(b 16)(c 4))	4

## 2) Исходный код.

```
#include <iostream>
#include <regex>
#include <string>
#include <fstream>
#include <variant>
#include <map>

struct Node {

    Node() = default;

    std::variant<char, int> value;
    std::variant<std::pair<Node*, Node*>, Node*> arguments;

    int calculate();
};

bool rec_fill_branch(Node& element, std::string& expression);

int Node::calculate()
{
    if (std::holds_alternative<char>(value))
    {
        if (std::get<char>(value) == '+')
            return std::get<std::pair<Node*, Node*>>(arguments).first->calculate() +
                    std::get<std::pair<Node*, Node*>>(arguments).second->calculate();
        if (std::get<char>(value) == '*')
            return std::get<std::pair<Node*, Node*>>(arguments).first->calculate() *
                    std::get<std::pair<Node*, Node*>>(arguments).second->calculate();
        if (std::get<char>(value) == '-')
        {
            if (std::holds_alternative<std::pair<Node*, Node*>>(arguments))
                return std::get<std::pair<Node*, Node*>>(arguments).first->calculate() -
                        std::get<std::pair<Node*, Node*>>(arguments).second->calculate();
            else
                return (-1) * std::get<Node*>(arguments)->calculate();
        }
        if (std::get<char>(value) == '/')
            return std::get<std::pair<Node*, Node*>>(arguments).first->calculate() /
                    std::get<std::pair<Node*, Node*>>(arguments).second->calculate();
        else
            return std::get<int>(value);
    }
}
```

```

        return 0;
    }

    bool is_operand(char c) {
        if (c == '-' || c == '+' || c == '*' || c == '/')
            return true;
        else return false;
    }

    bool create_node(std::string& expression, int &i, Node* &element) {

        int bracket_cnt = 0;
        std::string buff;

        if (expression[i] == '(') {
            while (true) {
                buff += expression[i];
                if (expression[i] == '(') {
                    bracket_cnt++;
                }
                else if (expression[i] == ')') {
                    bracket_cnt--;
                }
                i++;
                if (bracket_cnt == 0) {
                    break;
                }
            }

            if (!rec_fill_branch(element, buff))
                return false;
        }
        else {
            while (expression[i] != ' ' && expression[i] != '(' &&
expression[i] != ')') {
                if (isdigit(expression[i])) {
                    buff += expression[i];
                    i++;
                }
                else
                    return false;
            }
            element->value = stoi(buff);
        }

        return true;
    }

    bool rec_fill_branch(Node* &element, std::string& expression) {

        int i = 1;
        std::string buff;

        Node* left = new Node;
        bool f = create_node(expression, i, left);

        if (!f) {
            return false;
        }
    }

```

```

    }

    while (expression[i] == ' ') {
        i++;
    }

    if (is_operand(expression[i])) {
        if (expression[i] != '-')
            return false;
        else {
            element->value = expression[i];
            element->arguments = left;
            return true;
        }
    }

    else {
        Node* right = new Node;
        f = create_node(expression, i, right);

        if (!f) {
            return false;
        }

        while (expression[i] == ' ') {
            i++;
        }

        if (is_operand(expression[i]))
            element->value = expression[i];
        else
            return false;

        element->arguments = std::make_pair(left, right);
        return true;
    }

    return true;
}

bool is_brackets_correct(std::string &expression) {

    int brackets_cnt = 0;

    for (size_t i = 0; i < expression.length(); i++) {
        if (brackets_cnt < 0)
            return false;
        else {
            if (expression[i] == '(')
                brackets_cnt++;
            else if (expression[i] == ')')
                brackets_cnt--;
            else continue;
        }
    }

    if (brackets_cnt == 0)
        return true;
    else return false;
}

```



```

}

void fill_map(std::string& values_str, std::map<std::string, int>&
arguements_values_map) {

    std::regex pattern("\\((\\w+) (-?\\d+)\\)");
    std::smatch match;

    int is_like_pattern = 0;
    while (is_like_pattern = std::regex_search(values_str, match,
pattern)) {
        arguements_values_map.insert(std::pair<std::string,
int>(match[2].str(), stoi(match[3])));
        values_str.erase(match.position(2), match[2].length());
    }

    //std::cout << "Arguements values" << std::endl;

    for (size_t i = 0; i < values_str.length(); i++) {
        if (isalpha(values_str[i])) {
            std::cout << "Incorrect argument value!" << std::endl;
            break;
        }
    }

    /*for (auto it = arguements_values_map.begin(); it !=
arguements_values_map.end(); ++it) {
        std::cout << it->first << " = " << it->second << std::endl;
    }*/
}

//replace var names with their values
bool substitute(std::string& expression, std::map<std::string, int>&
arguements_values_map) {

    int pos = 0;
    std::string buff;
    for (auto it = arguements_values_map.begin(); it !=
arguements_values_map.end(); ++it) {
        buff = std::to_string(it->second);
        pos = expression.find(it->first);
        while (pos != -1) {
            if ((pos == 0 || !isalpha(expression[pos - 1])) &&
!isalpha(expression[pos + it->first.length()])) {
                expression.replace(pos, it->first.length(), buff);
                pos = expression.find(it->first, pos);
            }
            else pos = expression.find(it->first, pos + 1);
        }
    }

    arguements_values_map.clear();

    //std::cout << "replaced variables with their values:" << std::endl <<
expression << std::endl;

    for (size_t i = 0; i < expression.length(); i++) {
        if (isalpha(expression[i])) {

```

```

        std::cout << "You didn't set all values" << std::endl;
        return false;
    }
}

return true;
}

void console_input(std::map <std::string, int>& arguments_values_map,
Node* head) {

    std::cout << "Please enter an expression" << std::endl;
    std::string expression;

    getline(std::cin, expression);

    std::cout << "Please enter the values" << std::endl;
    std::string values;

    getline(std::cin, values);

    fill_map(values, arguments_values_map);

    if (is_brackets_correct(expression)) {
        //std::cout << expression << std::endl;
        if (substitute(expression, arguments_values_map)) {
            if (rec_fill_branch(head, expression)) {
                std::cout << head->calculate() << std::endl;;
            }
            else
                std::cout << "Incorrect expression detected" <<
std::endl;
        }
    }
}

void file_input(char* argv, std::map <std::string, int>&
arguments_values_map, Node* head) {

    std::ifstream file;
    std::string file_name = argv;

    file.open(file_name);
    if (!file.is_open())
        std::cout << "Error! File isn't open" << std::endl;

    std::string expression;
    std::string values;
    int i = 0;
    while (!file.eof()) {
        i++;
        getline(file, expression);
        getline(file, values);

        std::cout << i << " : " << expression << std::endl << values <<
std::endl;
    }
}

```

```

fill_map(values, arguments_values_map);

if (is_brackets_correct(expression)) {
    //std::cout << expression << std::endl;
    if (substitute(expression, arguments_values_map)) {
        if (rec_fill_branch(head, expression)) {
            std::cout << head->calculate() << std::endl;;
        }
        else
            std::cout << "Incorrect expression detected" <<
std::endl;
    }
}

}

}

int main(int argc, char** argv)
{
    Node* head = new Node;

    std::map <std::string, int> arguments_values_map;

    if (argc == 1)
        console_input(arguments_values_map, head);
    else if (argc == 2)
        file_input(argv[1], arguments_values_map, head);
    else std::cout << "Please check arguments are correct";
}

```