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. 9382 _____ Бочаров Г.С.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Научиться представлять логические выражения с помощью иерархических списков. Вычислить результат логического выражения с помощью иерархических списков и рекурсии.

Задание.

18) логическое, вычисление, добавить 4-ую операцию (которая может принимать 2 аргумента), префиксная форма.

Основные теоретические положения.

Пусть выражение (логическое, арифметическое, алгебраическое*) представлено иерархическим списком. В выражение входят константы и 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, упрощение (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 xi – переменная, а ci – её значение (константа).

В индивидуальном задании указывается: тип выражения (возможно дополнительно - состав операций), вариант действия и форма записи. Всего 9 заданий.

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

Функции и структуры данных.

Структура Node {std::string value, Node* next ,Node* child }
хранит строковое значение элемента списка, указатель на следующий элемент списка и указатель на начало подсписка.

Структура Var {std::string name; int val; } хранит имя и значение переменной.

template<typename IterT>

void readVariables(std::vector<Var> *res, IterT &first, const IterT &last)

Функция преобразует массив токенов в массив переменных. Функция принимает на вход массив для записи результата, указатель на первый последний элемент массива токенов. Также проверет корректность данного выражения

template<typename StreamT>

std::vector<std::string> getToken(StreamT &stream)

Функция преобразует строку в массив токенов, для удобства работы выражением. Функция принимает на вход поток ввода и возвращает массив токенов.

void getNext(IterT &first, const IterT &last, Node *&parent)

Функция считывает следующий элемент текущего списка.

void getChild(IterT &first, const IterT &last, Node *&parent)

Функция считывает начало подсписка.

template<typename IterT>

void getList(IterT &first, const IterT &last, Node *&parent)

Функция считывает начало иерархического списка.

bool getVarValue(Node *a, const std::vector<Var> &Vars)

Функция возвращает значение переменной, принимая на вход имя переменной и массив заданных переменных. Если значение переменной не было найдено или имя переменной некорректно выводит сообщение об ошибке

bool doAction(const std::string &opName, bool a, bool b = false)

Функция выполняет требуемую операцию, также проверяет корректность имен операторов

bool Calculate(Node *parent, std::vector<Var> &Variables)

Фуннкция рекурсивно вычисляет значение логического выражения, проходя по иерархическому списку в глубину и вычисляя значения узлов.

bool CalculateWithDetails(Node *parent, std::vector<Var> &Variables)

Фунция отличается от функции Calculate() только тем, что выводит промежуточные действия.

Описание алгоритма.

Добавлена операция ^ - исключающее ИЛИ.

На вход программе подается 2 выражение. Первое — задает имена и значения переменных, второе — задает само логическое выражение.

Для удобства работы оба выражения были разбиты на массивы токенов. Токены — скобки, операторы, константы, переменные.

Создается массив структур Var для хранения заданных переменных.

С помощью функций getList() getChild() getNext() создается иерархический список.

Структура списка следующая: Каждый оператор содержит указатель на список его аргументов (child). При этом сам оператор, может быть аргументом другого оператора. Переменные и константы не имеют аргументов.

Используя обход списка в глубину, вычисляются значения (результат выполнения той или иной логической операции) в узлах списка, и в корне.

Таким образом вычисляется значение всего выражения.

Также с помощью обхода в глубину данный список выводится на экран.

Графическое представление иерархического списка для выражения (- (+ a (^ 0 c))) изображено на рисунке 1 в Приложении Б.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	((a 1) (b 0)) (+ (- a) (- b))	(- 1) = 0 (- 0) = 1 (+ 0 1) = 1 --> res = 1	Вывод порядка действий и конечного результата
2.	((a 1) (b 0) (c 1)) (- (+ a (^ 0 c)))	(^ 0 1) = 1 (+ 1 1) = 1 (- 1) = 0 --> res = 0	
3.	((x1 0) (x2 0) (x3 1)) (* (- x1) (- (+ x2 x1)))	(- 0) = 1 (+ 0 0) = 0 (- 0) = 1 (* 1 1) = 1 --> res = 1	
4.	(+ 1 (- 0))	(- 0) = 1 (+ 1 1) = 1 --> res = 1 --- + 1	Вывод списка и результата вычислений

		- 0	
5.	((a 1) (b 0) (c 1)) (- (+ a (^ 0 G)))	Значение переменной <G> не указано	Значение переменной не указано
6.	((a 7) (23a 0) (7 b)) (- (+ a (^ 0 c)))	Недопустимое значение переменной <a>	Значение переменной не является нулем или единицей
7.		Неверный формат ввода	Пустое выражение
8.	((23a 1) (1 1)) (+ 1 1)	Недопустимое имя переменной <23a>	Некорректное имя переменной
9.	((a 1) (b 1) (c 0) (d 1)) (* (- (+ (^ 1 b) (* (- b) (+ a 0)))) 0)	--> res = 0 * - + ^ 1 b * - b + a 0 0	Вывод результата и списка.

Выводы.

Был реализован алгоритм вычисления логического выражения в префиксной форме. Вычмслено значение логического выражения с помощью иерархического списка и рекурсии. Освоены приемы работы с иерархическими списками.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <iostream>
#include <vector>
#include <sstream>
#include <fstream>
#include <string>

int depth = 0;

struct Node {
    std::string value;
    Node *next; //следующий элемент в списке
    Node *child; // начало подсписка
};
struct Var {
    std::string name; //имя переменной
    int val;  //значение переменной
};

//Функция преобразует обрабатывает массив токенов и заносит переменные в
массив переменных
//также проверет корректность данного выражения
template<typename IterT>
void readVariables(std::vector<Var> *res, IterT &first, const IterT &last) {
    if (first >= last) {
        return;
    }
```

```
    if (*first == "(") {
        if (*(first + 3) != ")")
            throw std::runtime_error("Неправильный формат ввода");
        readVariables(res, ++first, last);
    } else if (*first == ")") {

        if (*(first - 3) != "("
            throw std::runtime_error("Неправильный формат ввода");
        readVariables(res, ++first, last);
    } else {
        if ((*(first + 1) != "1") && (*(first + 1) != "0"))
            throw std::runtime_error("Недопустимое значение переменной <" + *first +
">");
        if (!isalpha((*first)[0])) {
            throw std::runtime_error("Недопустимое имя переменной <" + *first +
">");
        }
        Var var = {*first, stoi(*(first + 1))};
        res->push_back(var);
        if (first + 2 >= last) {
            throw std::runtime_error("Неправильный формат ввода");
        } else {
            first += 2;
        }
        if (*first != ")")
            throw std::runtime_error("Неправильный формат ввода");
        readVariables(res, first, last);
        return;
    }
}
```

```
//Функция преобразует строку в массив токенов, для удобства работы с выражением
template<typename StreamT>
std::vector<std::string> getToken(StreamT &stream) {
    std::vector<std::string> result;
    std::string temp;
    char symbol = 0;
    std::string line;
    std::getline(stream, line);
    line.push_back('\n');
    std::istringstream str(line);
    str >> std::noskipws;
    while (symbol != '\n') {
        str >> symbol;
        if ((symbol == ' ') || ((symbol == '\n'))) {
            if (!temp.empty())
                result.push_back(temp);
            temp.clear();
        }
        if ((symbol == '(') || (symbol == ')')) {
            if (!temp.empty())
                result.push_back(temp);
            temp.clear();
            temp.push_back(symbol);
            result.push_back(temp);
            temp.clear();
        }
        if ((symbol != '(') && (symbol != ')') && (symbol != ' ') && (symbol != '\n'))
            temp.push_back(symbol);
    }
    return result;
```

```
}

//Ф-ция проверяет, является ли оператор унарным
bool isUnoOperator(const std::string &s) {
    if (s == "-")
        return true;
    return false;
}

//Ф-ция проверяет, является ли оператор бинарным
bool isBinOperator(const std::string &s) {
    if ((s == "+") || (s == "*") || (s == "^"))
        return true;
    return false;
}

template<typename IterT>
bool checkRComas(IterT &first, const IterT &last);

//Ф-ция проверяет проверяет может ли токен быть переменной
template<typename IterT>
bool checkVars(IterT &first, const IterT &last) {
    if (first >= last)
        return false;
    if (!isUnoOperator(*first) && !isBinOperator(*first) && (*first != "(") &&
(*first != ")"))
        return true;
    return false;
}

//Ф-ция проверяет проверяет может ли токен быть началом аргумента
```

оператора

```
template<typename IterT>
bool checkArg(IterT &first, const IterT &last) {
    if (first >= last)
        return false;
    if (checkVars(first, last))
        return true;
    if (checkRComas(first, last))
        return true;
    throw std::runtime_error("Неверные аргументы у оператора");
}

//Ф-ция проверяет проверяет может ли токен быть оператором
template<typename IterT>
bool checkOperators(IterT &first, const IterT &last) {
    if (first >= last)
        return false;
    if (!isUnoOperator(*first) && !isBinOperator(*first))
        return false;
    else if (isUnoOperator(*first))
        return checkArg(++first, last);
    else if (isBinOperator(*first)) {
        return checkArg(++first, last) && checkArg(++first, last);
    }
    return false;
}

//Ф-ция проверяет проверяет является ли токен закрывающей скобкой
template<typename IterT>
bool checkLComas(IterT &first, const IterT &last) {
    if (first >= last)
```

```
        return false;
    if (*first == ")")
        return true;
    return false;
}

//Ф-ция проверяет проверяет является ли токен открывающей скобкой
template<typename IterT>
bool checkRComas(IterT &first, const IterT &last) {
    if (first >= last)
        return false;
    if (*first == "(") {
        if (!checkOperators(++first, last))
            throw std::runtime_error("Ошибка оператора");
        if (!checkLComas(++first, last))
            throw std::runtime_error("Отсутствует закрывающая скобка");
        return true;
    }
    return false;
}

template<typename IterT>
bool checkSentence(IterT &first, const IterT &last) {
    return checkRComas(first, last);
}

template<typename IterT>
void getChild(IterT &first, const IterT &last, Node *&parent);


//Ф-ция считывает следующий элемент списка
```

```
template<typename IterT>
void getNext(IterT &first, const IterT &last, Node *&parent) {
    if (*first == "(")
        first++;
    if (*first == ")")
        return;
    parent->next = new Node{*first, nullptr, nullptr};
    if (isBinOperator(*first) || isUnoOperator(*first)) {
        getChild(++first, last, parent->next);
    }
    getNext(++first, last, parent->next);

}


//Ф-ция считывает начало подсписка
template<typename IterT>
void getChild(IterT &first, const IterT &last, Node *&parent) {
    if (*first == "(")
        first++;
    if (*first == ")")
        return;
    parent->child = new Node{*first, nullptr, nullptr};
    if (isBinOperator(*first) || isUnoOperator(*first)) {
        getChild(++first, last, parent->child);
    }
    getNext(++first, last, parent->child);

}

//Ф-ция считывает начало подсписка
```

```
template<typename IterT>
void getList(IterT &first, const IterT &last, Node *&parent) {

    if (first >= last) {
        return;
    }
    if (*first == "(") {
        getList(++first, last, parent);
    } else if (*first == ")") {
        getList(++first, last, parent);
    } else if (isBinOperator(*first) || isUnoOperator(*first)) {

        if (parent == nullptr) {
            parent = new Node{*first, nullptr, nullptr};
        }
        getChild(++first, last, parent);
    } else {
        if (parent == nullptr) {
            parent = new Node{*first, nullptr, nullptr};
        }
    }
}

//Ф-ция выводит построенное дерево на экран
void listPrint(Node *parent) {
    if (parent == nullptr)
        return;
    std::string str(depth, ' ');
    std::cout << str << parent->value << std::endl;
    depth++;
    listPrint(parent->child);
```

```
    depth--;
    listPrint(parent->next);

}

//Ф-ция возвращает значение переменной, принимая на вход имя переменной и массив заданных переменных. Если значение переменной не было найдено выводит сообщение об ошибке
bool getVarValue(Node *a, const std::vector<Var> &Vars) {

    if (!isalpha(a->value.at(0)))
        throw std::runtime_error("Имя переменной <" + a->value + "> неверно");
        for (auto &i:Vars)
            if (i.name == a->value)
                return i.val;
    throw std::runtime_error("Значение переменной <" + a->value + "> не указано");
}

//Ф-ция выполняет  требуемую операцию, также проверяет корректность имен операторов
bool doAction(const std::string &opName, bool a, bool b = false) {
    if (opName == "+")
        return a || b;
    else if (opName == "*")
        return a && b;
    else if (opName == "^")
        return a ^ b;
    else if (opName == "-")
        return !a;
    else
```

```
        throw std::runtime_error("Неопознанный оператор <" + opName + ">");
}

//Ф-ция рекурсивно вычисляет значение логического выражения, проходя по дереву в глубину и вычисляя значения узлов.
bool Calculate(Node *parent, std::vector<Var> &Variables) {
    if (parent == nullptr) {
        throw std::runtime_error("Неверный формат ввода");
    }
    if (parent->child == nullptr) {
        if (parent->value == "1")
            return true;
        else if (parent->value == "0")
            return false;
        else return getVarValue(parent, Variables);
    } else if (isUnoOperator(parent->value)) {
        return doAction(parent->value, Calculate(parent->child, Variables));
    } else
        return doAction(parent->value, Calculate(parent->child, Variables),
                        Calculate(parent->child->next, Variables));
}

//Ф-ция рекурсивно вычисляет значение логического выражения, проходя по дереву в глубину и вычисляя значения узлов.
//Выводит порядок действий
bool CalculateWithDetails(Node *parent, std::vector<Var> &Variables) {
    if (parent == nullptr) {
        throw std::runtime_error("Неверный формат ввода");
    }
    if (parent->child == nullptr) {
        if (parent->value == "1")
```

```
            return true;
        else if (parent->value == "0")
            return false;
        else return getVarValue(parent, Variables);
    } else if (isUnoOperator(parent->value)) {
        depth++;
        bool a = CalculateWithDetails(parent->child, Variables);

        std::string sp(depth, '\t');
        bool res = doAction(parent->value, a);
        std::cout << sp << "( " << parent->value << " " << a << " ) = " << res <<
std::endl;

        depth--;
        return res;
    } else {
        depth++;
        bool a = CalculateWithDetails(parent->child, Variables);
        bool b = CalculateWithDetails(parent->child->next, Variables);

        bool res = doAction(parent->value, a, b);
        std::string sp(depth, '\t');
        std::cout << sp << "( " << parent->value << " " << a << " " << b << " ) = " <<
res << std::endl;
        depth--;
        return res;
    }
}

//Функция выводит массив токенов
void printTokens(const std::vector<std::string> &Tokens) {
```

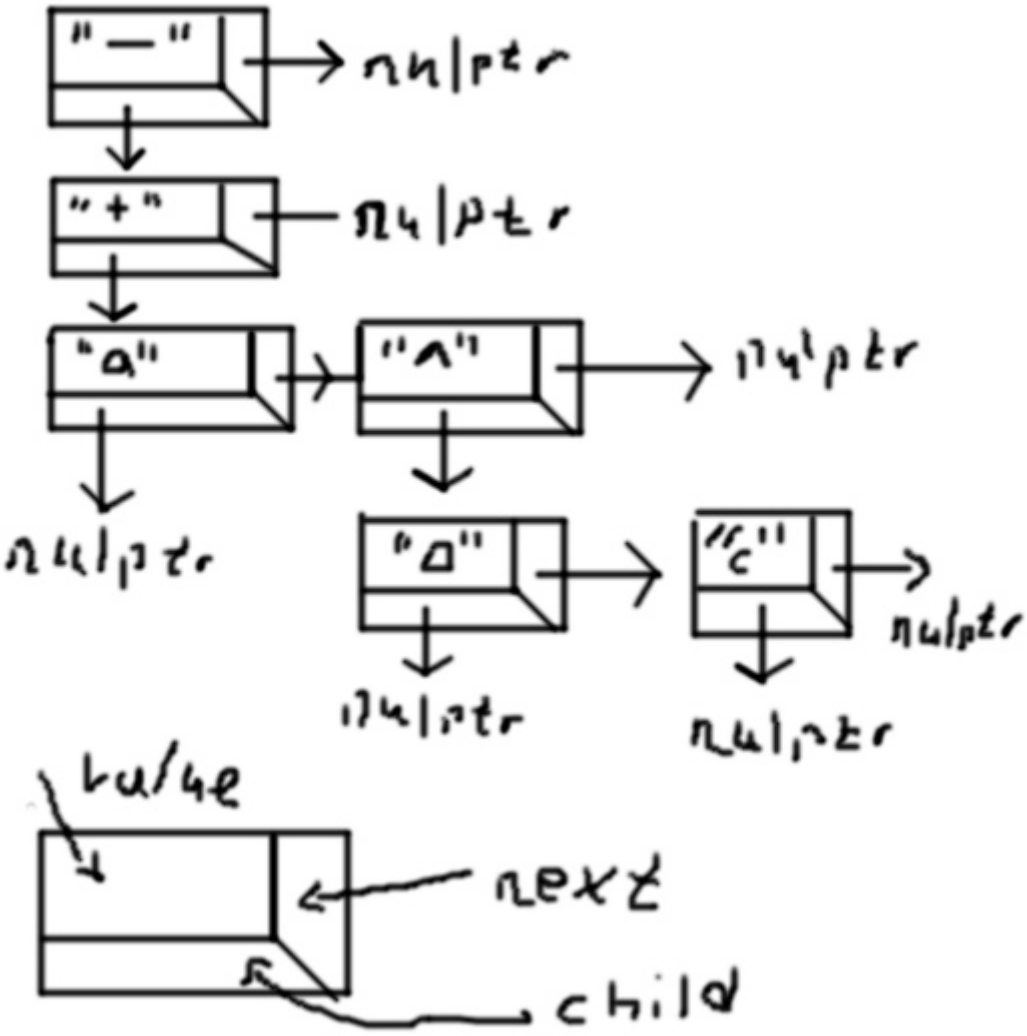
```
    for (auto &i:Tokens)
        std::cout << i << " ";
    std::cout << std::endl;
}

int main() {
    try {
        while (1) {
            Node *head = nullptr;
            std::vector<std::string> TokensVar; // Масив токенов для выражения,
задающего имена и значения переменных
            std::vector<std::string> TokensSentence; // Масив токенов для логического
выражения
            std::vector<Var> Variables;
            int readFormat;
            std::cout << "0 - считать из файла, 1 - считать с консоли" << std::endl;
            std::cin >> readFormat;
            std::cin.ignore();
            switch (readFormat) {
                case 0: {
                    std::cout << "Введите имя файла : ";
                    std::ifstream in;
                    std::string fileName;
                    std::cin >> fileName;
                    in.open(fileName);
                    if (in) {
                        TokensVar = getToken(in);
                        TokensSentence = getToken(in);
                    } else
                        throw std::runtime_error("Файл не найден!");
                    in.close();
```

```
            break;
        }
        case 1: {
            std::cout << "Введите имена переменных и их значения. Например :
((a 0) (x 1))" << std::endl;
            TokensVar = getToken(std::cin);

            std::cout << "Введите выражение в префиксной форме. Например :
(+ a (* 0 (- x)))" << std::endl;
            TokensSentence = getToken(std::cin);
            break;
        }
        default:
            throw std::runtime_error("Неверное действие");
    }
    if (!TokensVar.empty()) {
        if ((*TokensVar.begin() != "(") || (*(TokensVar.end() - 1) != ")"))
            throw std::runtime_error("Неправильный формат ввода");
        auto beg = TokensVar.begin() + 1, last = TokensVar.end() - 1;
        readVariables(&Variables, beg, last);
    }

    auto beg = TokensSentence.begin(), last = TokensSentence.end();
    if (!checkSentence(beg, last))
        throw std::runtime_error("Неверный формат ввода");
    beg = TokensSentence.begin(), last = TokensSentence.end();
    getList(beg, last, head);
    int action;
    std::cout << " 1 - вывести значения переменных, \n"
                 " 2 - вывести выражение,\n"
                 " 3 - вывести результат вычислений,\n"
```

```
                " 4 - вывести результат вычислений с выводом порядка
действий,\n"
                " 5 - вывести дерево,\n"
                " 0 - выход" << std::endl;
        while ((std::cin >> action) && (action != 0)) {
            switch (action) {
                case 1: {
                    printTokens(TokensVar);
                    break;
                }
                case 2: {
                    printTokens(TokensSentence);
                    break;
                }
                case 3: {
                    bool res = Calculate(head, Variables);
                    std::cout << "--> res = " << res << std::endl;
                    break;
                }
                case 4: {
                    bool res = CalculateWithDetails(head, Variables);
                    std::cout << "--> res = " << res << std::endl;
                    break;
                }
                case 5: {
                    listPrint(head);
                    break;
                }
                case 0: {
                    break;
                }
```

```
                default:
                    std::cout << "Выбрано неверное действие" << std::endl;
                    break;
                }
            }
            int temp;
            std::cout << "Для повторного ввода нажмите 1, для выхода - любую другую клавишу" << std::endl;
            std::cin >> temp;
            std::cin.ignore();
            if (temp != 1)
                break;
        }
    } catch (std::exception &e) {
        std::cerr << e.what() << std::endl;
    }
    return 0;
}
```

ПРИЛОЖЕНИЕ Б



(Рисунок 1)