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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

ТЕМА: ДЕРЕВЬЯ

Студент гр. 9382 _____ Иерусалимов.Н

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель

Изучение и реализация структуры бинарного дерева, а также алгоритмов его обработки.

Основные теоретические сведения

Бинарное дерево — конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом:

< БД > ::= < пусто > | < непустое БД >,
< пусто > ::= Λ,
< непустое БД > ::= (< корень > < БД > < БД >).

Например, скобочному представлению

(*a* (*b* (*d* Λ (*h* Λ Λ)) (*e* Λ Λ)) (*c* (*f* (*i* Λ Λ) (*j* Λ Λ)) (*g* Λ (*k* (*l* Λ Λ) Λ))))

соответствует рис.1.

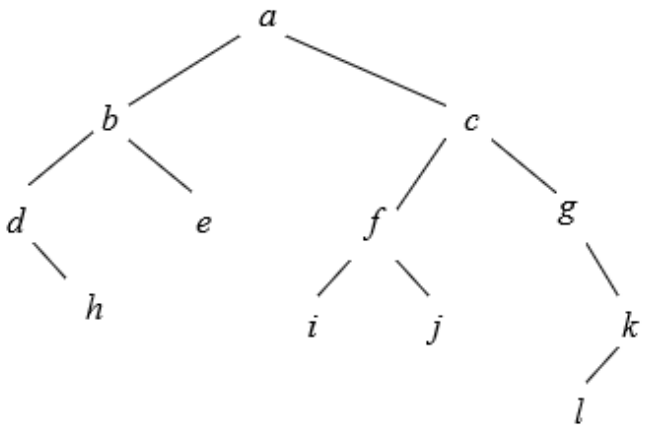


рис. 1

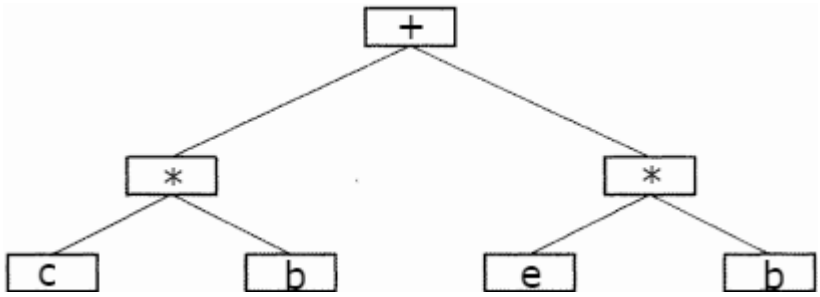
Задание

Вариант 15(д):

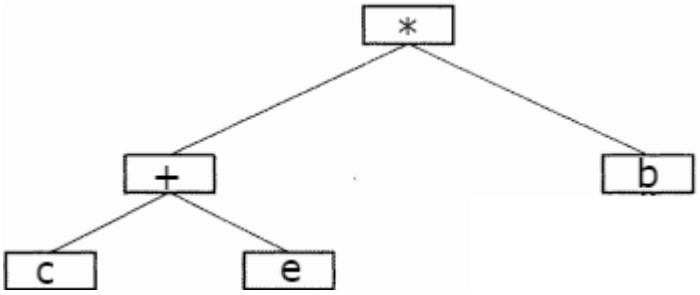
- преобразовать дерево-формулу t, заменяя в нем все поддеревья, соответствующие формулам $((f_1 * f_2) + (f_1 * f_3))$ и $((f_1 * f_3) + (f_2 * f_3))$, на поддеревья, соответствующие формулам $(f_1 * (f_2 + f_3))$ и $((f_1 + f_2) * f_3)$;

Алгоритм

1. Дерево заполняется при ЛКП-обходе, т. е. сначала при возможности обрабатываются левые поддеревья, в содержание заносится значение символа, а затем происходит обработка правых поддеревьев. Выражения, для которых выполняется бинарная операция, и являются этими поддеревьями, а знак операции — это содержание узла. Если встречен терминальный символ, содержанием становится он, а его поддеревья пусты, и происходит возврат из рекурсии.
2. Разложение выражений на множители осуществляется посредством ЛПК-обхода, что позволяет обрабатывать дерево с листьев. В зависимости от того, где была раньше обнаружена операция умножения, там проверяются равность одного из элементов к другому элементу из другого поддерева, если такие находятся, мы проверяем где именно стоят повторяющиеся элементы. Потом от этого обнуляем одно из поддерева предварительно записав все значение во второе. Выглядит это так.
3. 1)



2)



Функции и СД

- **class bTree —** класс, описывающий структуру бинарного дерева
  - **private**
    - **bTree* newTree(const string& tree, int& i, int& depth) –** создание дерева; tree – строка-выражение, i – счётчик символов, depth – глубина рекурсии; возвращаемое значение — указатель на корень дерева
    - **bTree* newNode(const char c) –** создание узла; c – содержание узла; возвращаемое значение — указатель на данный узел
  - **public**
    - **bTree(const string& tree, int& i, int& depth) –** конструктор дерева (вызывается только для начального узла, который не является фактическим корнем дерева); tree – строка-выражение, i – счётчик символов, depth – глубина рекурсии
    - **~bTree()** - деструктор, рекурсивно удаляющий поддеревья
    - **void reWorkTree(int& depth) –** распределение множителей; depth – глубина рекурсии
    - **void printResult()** - вывод результирующего выражения на экран
    - **void printResultToFile(const string filename) –** печать результирующего выражения в файл
    - **void printTree(int& depth) –** вывод в виде дерева (повёрнутого на 90 градусов влево) на экран

- **inline bool isTerminal(const char c) —** проверка на терминальный символ; c – проверяемый символ; возвращаемое значение — соответствие терминальному символу
- **inline bool isSign(const char c) -** проверка на знак; c – проверяемый символ; возвращаемое значение — соответствие знаку
- **inline void avoid(const string& s, int& i) –** пропуск пробелов при анализе выражения; s – строка-выражение, i – счётчик символов
- **inline void indent(int n) –** Отрисовка деревьев
- **inline void printRec(int n) -** отрисовка глубины рекурсии
- **void writeToFile(const string filename, const string arg) -** запись строки-выражения в файл с указанным именем; filename – имя файла назначения, arg – записываемое выражение
- **int main()**:
  - пользовательский интерфейс (выбор способа ввода выражений)
  - вывод промежуточных и итоговых результатов на экран и в файл

Тестирование

№	Входные данные	Выходные данные
1	b	b
2	(b*(c+4))	(b*(c+4))
3	((b*6)+(b*8))	(b*(6+8))
4	(b*(6+(7+5)))	(b*(6+(7+5)))
5	(((2*b)+(6*b))+((8*b)+(7*b)))	(((2+6)+(8+7))*b)
6	((q+3)*2)	((q+3)*2) -> ((q*2)+(3*2))
7	(	Invalid entry!
8	B+c	Invalid entry!
9	(b+c)	(b+c)
10	((b*v)+(b*n)	Invalid entry!

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

```
#include <string>
#include <ctype.h>
#include <fstream>
#include <iostream>


using namespace std;

inline bool isSign(const char c) {
    if (c == '+' || c == '-' || c == '*') {
        return true;
    }
    return false;
}

inline bool isDigitOrAlpha(const char c) {
    if (isdigit(c) || isalpha(c)) {
        return true;
    }
    return false;
}

inline void avoid(const string& s, int& i) {
    while (s[i] == ' ' || s[i] == '\t') {
        ++i;
    }
}

inline void indent(int n) {
    for (int i = 0; i < n; i++) {
        cout << '\t';
    }
}
inline void printRec(int n){
    for (int i = 0; i < n; i++) {
        cout << '.';
    }
}

void writeToFile(const string filename, const string arg) {
    ofstream output;
    output.open(filename, ios::app);
    output << arg;
    output.close();
}

class bTree{
private:
    char content;
    bTree* m;
    bTree* l;
    bTree* r;

    bTree(const char s){
```

```
        this->m = nullptr;
        this->l = nullptr;
        this-> r = nullptr;
        this->content = s;
    }

    bTree* newTree(const string& tree, int& i, int& depth){
        bTree* m = nullptr;
        if(tree[i]=='('){
           // ++depth;
            avoid(tree,i);

            printRec(depth);
            cout << "Examining expression of level " << depth << " : " <<
tree[i] << "\n";
            m = newNode(' ');
            m->l = newTree(tree, ++i, ++depth);
            avoid(tree, i);
            printRec(depth);
            cout << "Examining sign " << tree[i] << "\n";
            m->content = tree[i++];
            avoid(tree, i);
            m->r = newTree(tree, i, ++depth);
            printRec(depth);
            cout << "Finished examining expression of level " << depth << " : "
<< tree[i] << "\n";
            ++i;
        }else if(isDigitOrAlpha(tree[i])){
            printRec(depth);
            m = newNode(tree[i++]);
        }
        --depth;
        return m;
    }

    bTree* newNode(const char s){
        bTree* node = new bTree(s);
        return node;
    }

    bTree* copyTree(bTree const* const tree) {
        bTree* m = nullptr;
        if (tree != nullptr) {
            m = newNode(tree->content);
            m->l = copyTree(tree->l);
            m->r = copyTree(tree->r);
        }
        return m;
    }

public:
    bTree(const string& tree, int& i, int& depth){
        this->m = newTree(tree,i,depth);
        this->l = nullptr;
        this-> r = nullptr;
        this-> content = '\0';
```

```
    }

    ~bTree() {
        if (this->m != nullptr) {
            delete this->m;
        }
        else {
            if (this->l != nullptr) {
                delete this->l;
            }
            if (this->r != nullptr) {
                delete this->r;
            }
        }
    }

    void reWorkTree(int& depth) {
        if (this->m != nullptr) {//выходим из корня
            this->m->reWorkTree(depth);
        }
        else {
            //Перемещаемся в самый конец узлов
            bTree* l = this->l, * r = this->r;
            ++depth;
            printRec(depth);
            cout << "Examining symbol " << this->content << "\n";
            if (l != nullptr) {
                depth++;
                l->reWorkTree(depth);
                depth--;
            }
            if (r != nullptr) {
                depth++;
                r->reWorkTree(depth);
                depth--;
            }
            //Делаем проверку, узел с +?, если да значит дошли до конца
            if (this->content == '+') {
                //одинаковые символы слева
                if (l->content == '*' && isDigitOrAlpha(l->l->content) &&
isDigitOrAlpha(this->r->l->content) && l->l->content ==
this->r->l->content){ //Проверяем левый узел равен правому
                        this->content = '*'; //заменяем + на *
                        l->content = l->l->content; // перемещаем разные узлы в
нужное место
                        this->r->content = '+'; //ставим + вместо * после
перемещения узлов
                        this->r->l = l->r;
                    // обнуляем то от куда взяли узел
                        l->r= nullptr;
                        l->l= nullptr;

                        printRec(depth);
                        cout << "Repeat distribution\n";
                        this->reWorkTree(depth);
                }//одинаковые символы справа
```

```
                else if (r->content == '*'&& isDigitOrAlpha(l->r->content) &&
isDigitOrAlpha(this->r->r->content) && l->r->content == this->r->r->content)
{//Проверяем правый узел равен левому
                        this->content = '*';
                        r->content = l->r->content;
                        this->l->content = '+';
                        this->l->r = r->l;
                        r->l= nullptr;
                        r->r = nullptr;
                        printRec(depth);
                        cout << "Repeat distribution\n";
                        this->reWorkTree(depth);
                }
            }
            --depth;
        }
    }

    void printResultToFile(const string filename) {
        if (this->m != nullptr) {
            this->m->printResultToFile(filename);
        }
        else {
            string arg = "";
            arg += this->content;
            if (isSign(this->content)) {
                writeToFile(filename, "(");
            }
            if (this->l != nullptr) {
                this->l->printResultToFile(filename);
            }
            writeToFile(filename, arg);
            if (this->r != nullptr) {
                this->r->printResultToFile(filename);
            }
            if (isSign(this->content)) {
                writeToFile(filename, ")");
            }
        }
    }

    void printResult() {
        if (this->m != nullptr) {
            this->m->printResult();
        }
        else{
            if (isSign(this->content)) {
                cout << "(";
            }
            if (this->l != nullptr) {
                this->l->printResult();
            }
            cout << this->content;
            if (this->r != nullptr) {
                this->r->printResult();
            }
```

```
            if (isSign(this->content)) {
                cout << ")";
            }
        }
    }

    void printTree(int& depth) {
        if (this->m != nullptr) {
            this->m->printTree(depth);
        }
        else{
            if (this->r != nullptr) {
                depth++;
                this->r->printTree(depth);
                depth--;
            }
            indent(depth);
            cout << this->content << "\n";
            if (this->l != nullptr) {
                depth++;
                this->l->printTree(depth);
                depth--;
            }
        }
    }
};


bool isEntryValid(const string& tree, int& i) {
    if (tree.length() == 1 && isDigitOrAlpha(tree[0])) {
        return true;
    }
    avoid(tree, i);
    if (tree[i] == '(') {
        if (isEntryValid(tree, ++i)) {
            avoid(tree, i);
            if (isSign(tree[i])) {
                if (isEntryValid(tree, ++i)) {
                    avoid(tree, i);
                    if (tree[i] == ')') {
                        ++i;
                        return true;
                    }

                }
            }
        }
    }
    else if (i != 0 && isDigitOrAlpha(tree[i++])) {
        return true;
    }
    return false;
}

int main() {
    int command = 0;
```

```
    cout << "Enter : 1 - Console input , 2 - File Input:\n\n";
    cin >> command;  //Выбор, ввод с консоли или файла
 if (command == 2) {
     //-----Ввод с файла-----//
        string input_filename;
        const string output_filename = "output.txt";
        ifstream in;
        ofstream out;

        out.open(output_filename);
        out << "";
        out.close();

        cout << "Enter the input file name: \n\n";
        cin >> input_filename;
        in.open(input_filename);

        if (in.is_open()) {
            string s = "";

            getline(in, s);
            do {
                cout << "\n\n" << s << "\n";
                int i = 0;
                int depth = 0;
                if (isEntryValid(s, i)) { //проверяем валидны ли входные данные
                    i = 0;
                    bTree* tree = new bTree(s, i, depth); // создаем дерево,
передавая каждый встреченный символ и обрабатываем его там
                    depth = 0;
                    tree->printResultToFile(output_filename);  //Выводим на
экран то что считали
                    tree->printTree(depth); //строим дерево на экране
                    cout << "\n  --reWork--  \n\n"; //
                    tree->reWorkTree(depth); //Обрабатываем дерево, спускаемся
в самые низы, и от туда изменяем дерево постепенно переходя выше. ((f1
*f2 )+(f1 *f3 )) = (f1*(f2+f3))
                    writeToFile(output_filename, "- after rework -");
                    tree->printResultToFile(output_filename);//вводим в файл
результат
                    writeToFile(output_filename, "\n");
                    cout << "\nNew tree: \n\n";
                    tree->printTree(depth); //выводим новое дерево
                    cout << "\n\nResult expression: \n\n";
                    tree->printResult(); // выводим результат
                    cout << "\n\n";
                    delete tree;
                }
                else {
                    cout << "Invalid entry, the tree wasn't created\n";
                    writeToFile(output_filename, "INVALID ENTRY !\n");
                }
                cout << "_____________________________\n";
                s = "";
                getline(in, s);
            } while (!in.eof());
```

```
            cout << "\nCheck out the results in \"output.txt\"\n";
        }
        else {
            cout << input_filename << " doesn't exist!\n";
        }
    }


    if (command == 1) {
        string s = "";

        cout << "\nEnter the expression (or \"!\" to quit): \n";
        getline(cin, s);
        while (s != "!") {
            int i = 0;
            int depth = 0;
            if (isEntryValid(s, i)) {//проверяем валидны ли входные данные
                i = 0;\

cout<<"_____Create_a_sample_tree_____
_____\n";
                bTree* tree = new bTree(s, i, depth);// создаем дерево,
передавая строку и обрабатывая ее там
                depth = 0;

cout<<"\n_____Tree_____
_____\n";
                tree->printTree(depth);//строим дерево на экране

cout<<"\n_____Rework_____
_____\n";
                tree->reWorkTree(depth);//Обрабатываем дерево, спускаемся в
самые низы, и от туда изменяем дерево постепенно переходя выше. ((f1 *f2 )+(f1
*f3 )) = (f1*(f2+f3))

cout<<"\n_____New_Tree_____
_____\n";
                tree->printTree(depth);//строим новое дерево на экране

cout<<"\n_____Result_____
_____\n";
                tree->printResult();// выводим результат
                cout << "\n\n";
                cout <<
"
_____
____\n\n";
                delete tree;
            }
            else if (s != "") {
                cout << "Invalid entry, the tree wasn't created\n";
            }
            s = "";
            getline(cin, s);
        }
    }
    else {
```

```
        cout << "\nUnknown command, program finished\n";
    }
    return 0;
}
//((b*6)+(b*8))
//((b*6)+(((b*7)+(b*5))))
//((((2*b)+(6*b))+((8*b)+(7*b)))
//((((2*b)+(6*b))+((8*b)+(7*b)))
```