

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

Бочаров Г.С.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Представить математическое выражение в постфиксной форме в виде бинарного дерева. Построить выражение в постфиксной форме по заданному бинарному дереву. Освоить приемы работы с бинарными деревьями.

Задание.

Вариант 13 д):

- построить дерево-формулу t из строки, задающей формулу в постфиксной форме (перечисление узлов t в порядке ЛПК);
- упростить дерево-формулу t , выполнив в нем все операции вычитания, в которых уменьшаемое и вычитаемое – цифры. Результат вычитания – цифра или формула вида $(0 - \text{цифра})$

Основные теоретические положения.

Формулу вида:

$\langle \text{формула} \rangle ::= \langle \text{терминал} \rangle \mid (\langle \text{формула} \rangle \langle \text{знак} \rangle \langle \text{формула} \rangle)$

$\langle \text{знак} \rangle ::= + \mid - \mid *$

$\langle \text{терминал} \rangle ::= 0 \mid 1 \mid \dots \mid 9 \mid a \mid b \mid \dots \mid z$

можно представить в виде бинарного дерева («дерева-формулы») с элементами типа $\text{Elem}=\text{char}$ согласно следующим правилам:

- формула из одного терминала представляется деревом из одной вершины с этим терминалом;
- формула вида $(f1 \ s \ f2)$ представляется деревом, в котором корень - это знак s , а левое и правое поддеревья - соответствующие представления формул $f1$ и $f2$.

Функции и структуры данных.

Node - Структура для построения бинарного дерева. Используется для создания узлов и листьев дерева.

std::string Node :: value_ - строковое значение структуры

unique_ptr<Node> Node :: left_ - указатель на корень левого поддерева

unique_ptr<Node> Node :: right_ - указатель на корень правого поддерева

void Node :: createLeft(const std::string &value) — Функция принимает на вход строку и создает левого потомка данного узла.

void Node :: createRight(const std::string &value)— Функция принимает на вход строку и создает левого потомка данного узла.

void Node :: addChild(std::string &value)— Функция принимает на вход строку и создает левого потомка данного узла, если его еще нет, в противном случае создает правого потомка.

void Node :: addChild(std::unique_ptr<Node> &child) Функция принимает на вход указатель на элемент типа Node и делает его потомком текущего узла дерева.

template<typename StreamT>

std::vector<std::string> getToken(StreamT &stream) — Функция принимает на вход поток ввода. Функция преобразует считанную строку в массив токенов, для удобства работы с ней.

template<typename IterT>

void readOperator(IterT &first, const IterT &last, std::unique_ptr<Node> &parent)

— Функция принимает на вход указатели на начало и конец массива токенов, а также текущий узел дерева. Функция считывает оператор и заносит его в дерево.

```
template<typename IterT>
```

```
void readArgument(IterT &first, const IterT &last, std::unique_ptr<Node> &parent)
```

— Функция принимает на вход указатели на начало и конец массива токенов, а также текущий узел дерева. Функция считывает аргумент оператора и заносит его в дерево.

```
template<typename IterT>
```

```
void readSentence(IterT &first, const IterT &last, std::unique_ptr<Node> &parent)—
```

Функция принимает на вход указатели на начало и конец массива токенов, а также текущий узел дерева. Функция считывает корень дерева.

```
void TreeToSentence(const std::unique_ptr<Node> &head, std::vector<std::string> &Tokens) —
```

Функция принимает на вход корень дерева и массив строк для записи выражения. Функция преобразует дерево в формулу.

```
void printSentenceReverse(std::vector<std::string> &Tokens) —
```

Функция принимает на вход массив строк и выводит его в обратном порядке. Таким образом выводится формула в постфиксной форме.

```
void printTree(const std::unique_ptr<Node> &head)-
```

Функция принимает на вход корень дерева. Функция выводит дерево на экран.

```
bool subst(std::unique_ptr<Node> &head)
```

Функция принимает на вход указатель на корень дерева. Функция заменяет все операции вычитания, где уменьшаемое и вычитаемое — цифры, на результат вычитания.

`void launch(std::unique_ptr<Node> &head)` — Функция принимает на вход указатель на корень бинарного дерева. Функция запрашивает у пользователя формат ввода и формирует массив токенов

`void readPostfixFormula(std::vector<std::string> Tokens, std::unique_ptr<Node> &head)` — Функция принимает на вход указатель на корень бинарного дерева и массив строк. Функция строит бинарное дерево по заданному в постфиксной форме выражению.

`void printMenu()` - Функция выводит меню для выбора действия с выражением

`void doAct(std::unique_ptr<Node> &head)` Функция принимает на вход указатель на корень бинарного дерева. Функция производит выбранное пользователем действие

Описание алгоритма.

На вход программе подается выражение в постфиксной форме.

Для удобства работы оба выражения было разбито на массив строк - токенов.

Токены — скобки, операторы, константы, переменные.

С помощью функций `readSentence()`, `readOperator()`, `readArgument()` по заданному выражению создается дерево (дерево-формула).

В функции `subst()` поиском в глубину в дереве находятся все действия вычитания в которых аргументы являются цифрами. Если в результате работы функции была выполнена хотя бы одна такая замена, функция возвращает значение `true`, в противном случае функция возвращает `false`.

Графическое представление бинарного дерева для выражения

5 10 7 + / 8 * (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.	35 5 - с * 7 9 - e f g * + / -	<p>Вывод дерева</p> <p>-</p> <p>/</p> <p>+</p> <p>*</p> <p>g</p> <p>f</p> <p>e</p> <p>-</p> <p>9</p> <p>7</p> <p>*</p> <p>c</p> <p>-</p> <p>5</p> <p>35</p> <p>Вывод выражения</p> <p>35 5 - с * 7 9 - e f g * + / -</p> <p>Замена операций вычитания</p> <p>Совершено действие: 7 - 9 =-2</p> <p>Вывод дерева</p> <p>-</p> <p>/</p> <p>+</p>	<p>Вывод</p> <p>выражения и</p> <p>дерева до и</p> <p>после</p> <p>преобразован</p> <p>ия</p>

		<p>*</p> <p>g</p> <p>f</p> <p>e</p> <p>"0-2"</p> <p>*</p> <p>c</p> <p>-</p> <p>5</p> <p>35</p> <p>Вывод выражения</p> <p>35 5 - c * "0 -2" e f g * + / -</p>	
2.	0 9 -	<p>Вывод дерева</p> <p>-</p> <p>9</p> <p>0</p> <p>Вывод выражения</p> <p>0 9 -</p> <p>Замена операций вычитания</p> <p>Совершено действие: 0 - 9 =-9</p> <p>Вывод дерева</p> <p>"0-9"</p> <p>Вывод выражения</p> <p>"0 -9"</p>	
3.	7 4 - 8 5 - -	<p>Вывод дерева</p> <p>-</p> <p>-</p> <p>5</p> <p>8</p> <p>-</p>	

		<p>4</p> <p>7</p> <p>Вывод выражения</p> <p>7 4 - 8 5 - -</p> <p>Замена операций вычитания</p> <p>Совершено действие: $8 - 5 = 3$</p> <p>Совершено действие: $7 - 4 = 3$</p> <p>Совершено действие: $3 - 3 = 0$</p> <p>Вывод дерева</p> <p>0</p> <p>Вывод выражения</p> <p>0</p>	
4.	9	<p>Вывод дерева</p> <p>9</p> <p>Вывод выражения</p> <p>9</p>	константа
5.	9 7 - t a 7 - 7 + / 8 *	<p>Вывод дерева</p> <p>*</p> <p>8</p> <p>/</p> <p>+</p> <p>7</p> <p>-</p> <p>a7</p> <p>t</p> <p>-</p> <p>7</p> <p>9</p> <p>Вывод выражения</p>	

		$9\ 7 - t\ a7 - 7 + / 8 *$ Замена операций вычитания Совершено действие: $9 - 7 = 2$ Вывод дерева <pre> * / / + 7 - a7 t 2 </pre> Вывод выражения $2\ t\ a7 - 7 + / 8 *$	
6.	3 5 - +	Неверный формат входных данных	
7.	+ s a s -	Неверный формат входных данных	
8.	3 5 - c * d e f g * + /	Неверный формат входных данных	
9.	7 8 - 4 5 - - 4 *	Вывод дерева <pre> * / / - - 5 4 - </pre>	

		8 7 Вывод выражения 7 8 - 4 5 - - 4 * 3 Замена операций вычитания Совершено действие: 4 - 5 =-1 Совершено действие: 7 - 8 =-1 Вывод дерева * 4 - "0-1" "0-1" Вывод выражения "0 -1" "0 -1" - 4 *	
--	--	--	--

Выводы.

В ходе работы был разработан алгоритм, создающий дерево-формулу для заданного выражения в постфиксной форме. Был разработан алгоритм представляющий бинарное дерево в виде выражения в постфиксной форме. Были изучены приемы работы с бинарными деревьями.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

int depth = 0;

struct Node {
    std::string value_;
    std::unique_ptr<Node> left_;
    std::unique_ptr<Node> right_;

    explicit Node(const std::string &value) {
        value_ = value;
    }

    Node(const std::string &value, std::unique_ptr<Node> &left,
std::unique_ptr<Node> &right) {
        value_ = value;
        left_ = std::move(left);
        right_ = std::move(right);
    }

    void createLeft(const std::string &value) {
        if (left_ != nullptr)
```

```

        throw std::runtime_error("Левое поддереву не пусто");
    left_ = std::make_unique<Node>(value);
}

void createRight(const std::string &value) {
    if (right_ != nullptr)
        throw std::runtime_error("Правое поддереву не пусто");
    right_ = std::make_unique<Node>(value);
}

void addChild(std::string &value) {
    if (left_ == nullptr)
        createLeft(value);
    else if (right_ == nullptr)
        createRight(value);
    else
        throw std::runtime_error("Некуда добавить потомка");
}

void addChild(std::unique_ptr<Node> &child) {
    if (left_ == nullptr)
        left_ = std::move(child);
    else if (right_ == nullptr)
        right_ = std::move(child);
    else
        throw std::runtime_error("Некуда добавить потомка");
}
};

//Функция проверяет является ли строка числом
bool isNumber(std::string &s) {

```

```

    for (auto &i: s)
        if (!isdigit(i))
            return false;
    return true;
}

bool isOperator(std::string &op) {
    if ((op == "+") || (op == "-") || (op == "*") || (op == "/"))
        return true;
    return false;
}

bool isVar(std::string &op) {
    if (isOperator(op) || op == "(" || op == ")")
        return false;
    if (!isNumber(op) && !isalpha(op.at(0)))
        return false;
    return true;
}

```

//Разбиение выражение на подстроки

```

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;
}

```

//Считывание оператора

```
template<typename IterT>
```

```
void readOperator(IterT &first, const IterT &last, std::unique_ptr<Node> &parent);
```

//Считывание аргументов оператора

```
template<typename IterT>
```

```
void readArgument(IterT &first, const IterT &last, std::unique_ptr<Node> &parent)
```

```
{
```

```
    if (first < last)
```

```
        throw std::runtime_error("Неверный формат входных данных");
```

```

if (isVar(*first))
    if (parent == nullptr)
        parent = std::make_unique<Node>(*first);
    else
        parent->addChild(*first);
else if (isOperator(*first))
    readOperator(first, last, parent);
else
    throw std::runtime_error("Неверный формат входных данных");
}

template<typename IterT>
void readOperator(IterT &first, const IterT &last, std::unique_ptr<Node> &parent) {
    if (first < last)
        throw std::runtime_error("Неверный формат входных данных");
    if (!isOperator(*first))
        throw std::runtime_error("Неверный формат входных данных");
    if (parent == nullptr) {
        parent = std::make_unique<Node>(*first);
        readArgument(--first, last, parent);
        readArgument(--first, last, parent);
    } else {
        std::unique_ptr<Node> child = std::make_unique<Node>(*first);
        readArgument(--first, last, child);
        readArgument(--first, last, child);
        parent->addChild(child);
    }
}

//Считывание корня
template<typename IterT>

```

```

void readSentence(IterT &first, const IterT &last, std::unique_ptr<Node> &parent) {
    if (first < last)
        throw std::runtime_error("Неверный формат входных данных");
    if (isOperator(*first))
        readOperator(first, last, parent);
    else
        readArgument(first, last, parent);
    if (first != last)
        throw std::runtime_error("Неверный формат входных данных");
}

```

//Функции считывания дерева в скобочном представлении

```

template<typename IterT>

```

```

void readTOperator(IterT &first, const IterT &last, std::unique_ptr<Node> &parent);

```

```

template<typename IterT>

```

```

void readTLComma(IterT &first, const IterT &last, std::unique_ptr<Node> &parent)

```

```

{
    if (first >= last)
        throw std::runtime_error("Неверный формат входных данных");
    if (*first != ",") {
        throw std::runtime_error("Неверный формат входных данных");
    }
}

```

```

template<typename IterT>

```

```

void readTArgument(IterT &first, const IterT &last, std::unique_ptr<Node> &parent)

```

```

{
    if (first >= last)
        throw std::runtime_error("Неверный формат входных данных");
}

```



```

if (*first != "(") {
    throw std::runtime_error("Неверный формат входных данных");
}
if (isVar(*(++first)))
    if (parent == nullptr)
        parent = std::make_unique<Node>(*first);
    else
        parent->addChild(*first);
else if (isOperator(*first))
    readTOperator(first, last, parent);
readTLComma(++first, last, parent);
}

template<typename IterT>
void readTOperator(IterT &first, const IterT &last, std::unique_ptr<Node> &parent)
{
    if (first >= last)
        throw std::runtime_error("Неверный формат входных данных");
    if (!isOperator(*first)) {
        throw std::runtime_error("Неверный формат входных данных");
    }
    if (parent == nullptr) {
        parent = std::make_unique<Node>(*first);
        readTArgument(++first, last, parent);
        readTArgument(++first, last, parent);
    } else {
        std::unique_ptr<Node> child = std::make_unique<Node>(*first);
        readTArgument(++first, last, child);
        readTArgument(++first, last, child);
        parent->addChild(child);
    }
}

```

```
}
```

```
//Преобразование дерева в формулу
```

```
void TreeToSentence(const std::unique_ptr<Node> &head, std::vector<std::string>
&Tokens) {
    if (head == nullptr)
        return;
    Tokens.push_back(head->value_);
    TreeToSentence(head->left_, Tokens);
    TreeToSentence(head->right_, Tokens);
}
```

```
//Вывод выражения на экран в обратном порядке
```

```
void printSentenceReverse(std::vector<std::string> &Tokens) {
    for (auto i = Tokens.end() - 1; i >= Tokens.begin(); i--)
        if (i->size() != 1 && i->at(0) == '-')
            std::cout << "\"0 " << *i << "\" ";
        else
            std::cout << *i << " ";
}
```

```
//Вывод дерева на экран
```

```
void printTree(const std::unique_ptr<Node> &head) {
    depth++;
    if (head == nullptr)
        return;
    std::string s(depth, ' ');
    if (head->value_.size() != 1 && head->value_.at(0) == '-')
        std::cout << s << "\"0" << head->value_ << "\"\" << std::endl;
    else
        std::cout << s << head->value_ << std::endl;
```

```

    printTree(head->left_);
    depth--;
    printTree(head->right_);
    depth--;
}

```

//Функция проверяет является ли строка цифрой

```

bool isDigit(const std::string &s) {
    if (s.size() == 1 && isdigit(s.at(0)))
        return true;
    return false;
}

```

```

bool subst(std::unique_ptr<Node> &head) {
    if (head == nullptr)
        return false;
    if (head->value_ == "-")
        if (isDigit(head->left_->value_) && isDigit(head->right_->value_)) {
            int d = std::stoi(head->right_->value_) - std::stoi(head->left_->value_);

            std::cout << "Совершено действие: " << head->right_->value_ << " " <<
head->value_ << " "
                << head->left_->value_ << " =" << d << std::endl;
            head->value_ = std::to_string(d);
            head->left_ = nullptr;
            head->right_ = nullptr;
            return true;
        }
    return subst(head->left_) || subst(head->right_);
}

```

//Функция выводит меню для выбора действия с выражением

```
void printMenu() {  
    std::cout << " 1 - вывести дерево, \n"  
        " 2 - вывести формулу,\n"  
        " 3 - заменить все разности с константами,\n"  
        " 0 - выход" << std::endl;  
}
```

//Функция производит выбранное пользователем действие

```
void doAct(std::unique_ptr<Node> &head) {  
    std::vector<std::string> result;  
    int action;  
    while ((std::cin >> action) && (action != 0)) {  
        switch (action) {  
            case 1: {  
                std::cout << "Вывод дерева" << std::endl;  
                printTree(head);  
                depth = 0;  
                break;  
            }  
            case 2: {  
                std::cout << "Вывод выражения" << std::endl;  
                result.clear();  
                TreeToSentence(head, result);  
                printSentenceReverse(result);  
                break;  
            }  
            case 3: {  
                std::cout << "Замена операций вычитания" << std::endl;  
                while (subst(head));  
                break;  
            }  
        }  
    }  
}
```

```

    }
    case 0: {
        break;
    }
    default:
        std::cout << "Выбрано неверное действие" << std::endl;
        break;
    }
}
}
}

```

//Функция строит дерево по формуле

```

void readPostfixFormula(std::vector<std::string> Tokens, std::unique_ptr<Node>
&head) {
    if (Tokens.empty())
        throw std::runtime_error("Пустое выражение!");
    auto end = Tokens.begin(), beg = Tokens.end() - 1;
    readSentence(beg, end, head);
    printMenu();
    doAct(head);
}

```

//Функция запрашивает у пользователя формат ввода и формирует массив токенов

```

void launch(std::unique_ptr<Node> &head) {
    std::vector<std::string> Tokens;
    int readFormat;
    while (1) {
        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) {
            Tokens = getToken(in);
        } else
            throw std::runtime_error("Файл не найден!");
        in.close();
        break;
    }
    case 1: {
        std::cout << "Введите выражение в постфиксной форме. Например : a b
+ c * d e f g * + / -"
        << std::endl;
        Tokens = getToken(std::cin);
        break;
    }
    default: {
        throw std::runtime_error("Неверное действие");
    }
}
readPostfixFormula(Tokens, head);
int temp;
std::cout << "Для повторного ввода нажмите 1, для выхода - любую другую
клавишу" << std::endl;
std::cin >> temp;

```

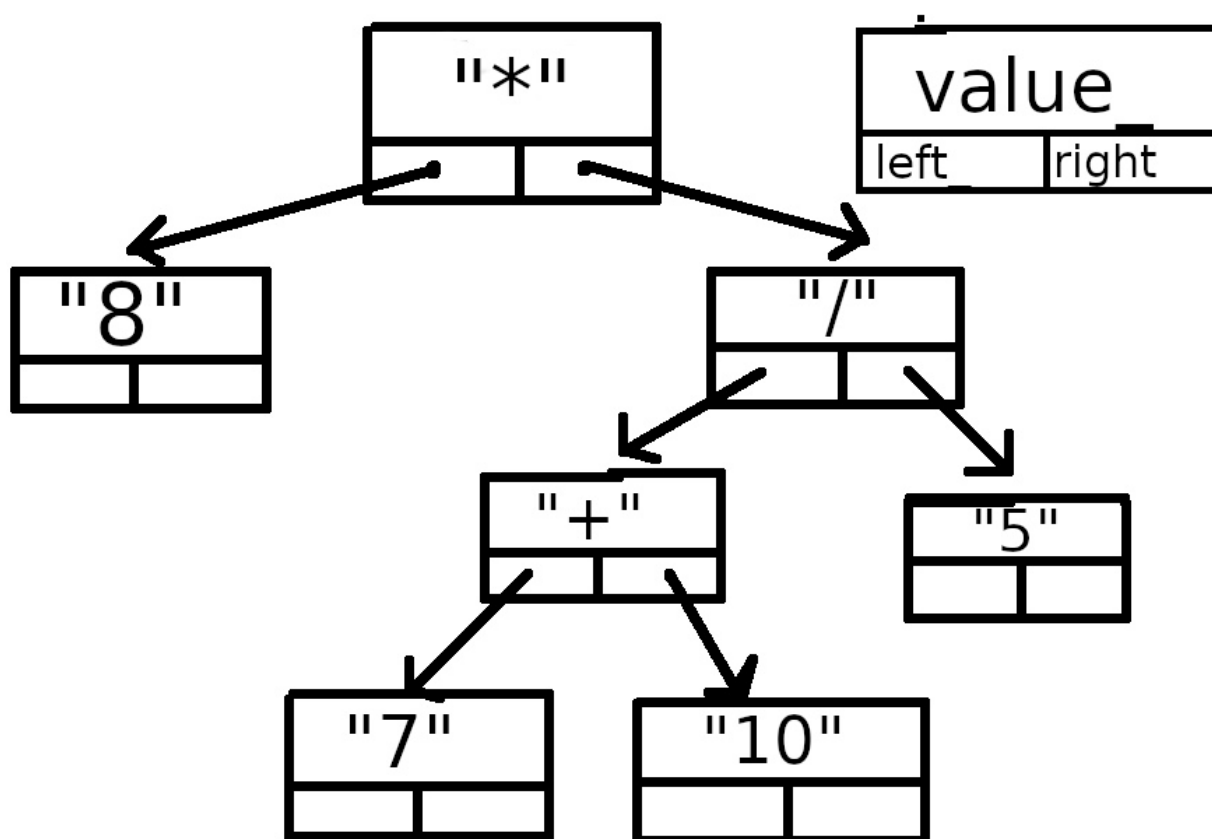
```

        std::cin.ignore();
        if (temp != 1)
            break;
        head.reset();
    }
}

int main() {
    std::unique_ptr<Node> head;
    try {
        launch(head);
    }
    catch (std::exception &e) {
        std::cerr << e.what() << std::endl;
    }
    return 0;
}

```

ПРИЛОЖЕНИЕ Б



(Рисунок 1)