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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Бинарные деревья

Студент гр. 9304 _____ Силкин В.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Изучить бинарные деревья, способы их обхода и работы с данными, реализовать бинарное дерево на языке программирования C++ и выполнить поставленную задачу.

Задание.

Для всех вариантов (11-17):

- Для заданной формулы f построить дерево-формулу t;

- Для заданного дерева-формулы t напечатать соответствующую формулу f;

Вариант 11у:

- С помощью построения дерева-формулы t преобразовать заданную формулу f из инфиксной формы в префиксную (перечисление узлов t в порядке КЛП) и в постфиксную (перечисление узлов t в порядке КЛП);

- Если в дереве-формуле t терминалами являются только цифры, то вычислить (как целое число) значение дерева-формулы t.

Выполнение работы.

Сначала строка базово проверяется на количество открывающих и закрывающих скобок, после чего подаётся в конструктор дерева. Программа обрабатывает пробелы, но все ветки дерева, в том числе и терминалы, пишутся в скобках, а сами эти ветки записываются в порядке ЛКП. Дерево конструируется на основе поданной строки, которая потом разделяется на левую и правую строку, из которых конструируются ветки рекурсивно.

Дерево записано в программе, и потому она распечатывает его сначала в префиксной форме, затем в постфиксной, и наконец, в инфиксной, после чего проверяет, встречаются ли буквы в терминалах. Если не встречаются, то запускается рекурсивное вычисление операций в ветках. Если в ветке один из потомков пуст, он записывается как '_', и считается равным 0.

В реализации дерева используются шаблоны и умные указатели.

Разработанный программный код см. в приложении А.

Тестирование.

Компиляция выполняется командой make (или make lab3).Для тестирования написан bash-скрипт, лежащий в корневой папке лабораторной работы, он запускается через make testing, либо вручную через файл test_script. Если до этого компиляции не было, программа сама скомпилирует файл для тестов.

```
$ ./test_script
test 1 pass
test 2 pass
test 3 pass
test 4 pass
test 5 pass
```

Рисунок 1 - Вывод тестирующего bash-скрипта

```
PS C:\Users\minec\Desktop\lab3> .\lab3.exe
(((()*(7))-((8)+()))
(-(*_7)(+8_))
((_7*)(8_+)-)
Expression: ((_*7)-(8+_))
Expression = -8
PS C:\Users\minec\Desktop\lab3> .\lab3.exe
(((6)*(7))-((8)+(9)))
(-(*67)(+89))
((67*)(89+)-)
Expression: ((6*7)-(8+9))
Expression = 25
PS C:\Users\minec\Desktop\lab3> .\lab3.exe
(((a)+(b))*((c)-(d)))
(*(+ab)(-cd))
((ab+)(cd-)*)
Expression: ((a+b)*(c-d))
```

Рисунок 2 - Тестинг без скрипта

Результаты тестирования см. в приложении В.

Выводы.

Были изучены бинарные деревья, и способы работы с их данными, а также реализован программный код на языке программирования C++ для считывания деревьев и обработки данных в них.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
/*11у вариант: Преобразовать ЛКП перечисление в КЛП и ЛПК БД, и если все некорневые ноды - числа,
 * вычислить интовое значение (указатели).
*/

#include <iostream>
#include <memory>
#include <string>

template <typename T>
struct Node{
    std::shared_ptr<Node<T>> left, right;
    T data;
};

template <typename T>
class Tree {

private:
    std::shared_ptr<Node<T>> construct(std::string& str)
    {
        if(str.length() < 3) {
            return nullptr;
        } else if(str.length() == 3) {
            std::shared_ptr<Node<T>> node = std::make_shared<Node<T>>();
            node->left = nullptr;
                node->right = nullptr;
                node->data = str[1];
            return node;
        } else {
            int iter = 2;
            for(int count = 1; count; iter++) {
                if(str[iter] == '(') {
                    count++;
                } else if(str[iter] == ')') {
                    count--;
                }
            }
            std::shared_ptr<Node<T>> node = std::make_shared<Node<T>>();
            node->data = str[iter];
```

```
                std::string left = str.substr(1, iter-1);
                std::string right = str.substr(iter+1,(str.size()-2-iter));
                node->left = construct(left);
                node->right = construct(right);
                return node;
            }
        }

    public:
        std::shared_ptr<Node<T>> head;

        Tree<T>(std::string& str) {
            this->head = construct(str);
            this->print_expression_KLP(this->head);
            std::cout << '\n';
            this->print_expression_LPK(this->head);
            std::cout << '\n' << "Expression: ";
            if(!isalnum(this->head->data)) {
                print_expression(this->head);
            } else {
                std::cout << '(' << this->head->data << ')';
            }
            std::cout << '\n';
            if(!scan_char(this->head)) {
                std::cout << "Expression = " << count_tree(this->head) <<
'\n';
            }
        }

        void print_KLP(std::shared_ptr<Node<T>>& node_ptr) {
            std::cout << node_ptr->data << ' ';
            if(node_ptr->left != nullptr) {
                print_KLP(node_ptr->left);
            }
            if(node_ptr->right != nullptr) {
                print_KLP(node_ptr->right);
            }
        }

        void print_LPK(std::shared_ptr<Node<T>>& node_ptr) {
            if(node_ptr->left != nullptr) {
                print_LPK(node_ptr->left);
            }
            if(node_ptr->right != nullptr) {
                print_LPK(node_ptr->right);
            }
            std::cout << node_ptr->data << ' ';
        }
```

```
        bool scan_char(std::shared_ptr<Node<T>>& node_ptr) {
            if(node_ptr == nullptr) {
                return false;
            }
            return (isalpha(node_ptr->data) || scan_char(node_ptr->left) ||
scan_char(node_ptr->right));
        }

        int count_tree(std::shared_ptr<Node<T>>& node_ptr) {
            int a,b;
            if(node_ptr->left != nullptr) {
                if(isdigit(node_ptr->left->data)) {
                    a = node_ptr->left->data - '0';
                } else {
                    a = count_tree(node_ptr->left);
                }
            } else {
                a = 0;
            }
            if(node_ptr->right != nullptr) {
                if(isdigit(node_ptr->right->data)) {
                    b = node_ptr->right->data - '0';
                } else {
                    b = count_tree(node_ptr->right);
                }
            } else {
                b = 0;
            }
            if(node_ptr->data == '*') {
                return a*b;
            } else if(node_ptr->data == '-') {
                return a-b;
            } else if(node_ptr->data == '+') {
                return a+b;
            } else {
                std::cerr << "Err: unexpected symbol";
                exit(1);
            }
        }

        void print_expression(std::shared_ptr<Node<T>>& node_ptr) {
            std::string math_sym = "*-+";
            std::cout << '(';
            if(node_ptr->left != nullptr) {
                if(math_sym.find_first_of(node_ptr->left->data) ==
std::string::npos) {
                    std::cout << node_ptr->left->data;
```

```
                } else {
                    print_expression(node_ptr->left);
                }
            } else {
                std::cout << '_';
            }
            std::cout << node_ptr->data;
            if(node_ptr->right != nullptr) {
                if(math_sym.find_first_of(node_ptr->right->data) ==
std::string::npos) {
                    std::cout << node_ptr->right->data;
                } else {
                    print_expression(node_ptr->right);
                }
            } else {
                std::cout << '_';
            }
            std::cout << ')';
        }

        void print_expression_KLP(std::shared_ptr<Node<T>>& node_ptr) {
            std::string math_sym = "*-+";
            std::cout << '(';
            std::cout << node_ptr->data;
            if(node_ptr->left != nullptr) {
                if(math_sym.find_first_of(node_ptr->left->data) ==
std::string::npos) {
                    std::cout << node_ptr->left->data;
                } else {
                    print_expression_KLP(node_ptr->left);
                }
            } else {
                std::cout << '_';
            }
            if(node_ptr->right != nullptr) {
                if(math_sym.find_first_of(node_ptr->right->data) ==
std::string::npos) {
                    std::cout << node_ptr->right->data;
                } else {
                    print_expression_KLP(node_ptr->right);
                }
            } else {
                std::cout << '_';
            }
            std::cout << ')';
        }

        void print_expression_LPK(std::shared_ptr<Node<T>>& node_ptr) {
```

```
            std::string math_sym = "*-+";
            std::cout << '(';
            if(node_ptr->left != nullptr) {
                if(math_sym.find_first_of(node_ptr->left->data) ==
std::string::npos) {
                    std::cout << node_ptr->left->data;
                } else {
                    print_expression_LPK(node_ptr->left);
                }
            } else {
                std::cout << '_';
            }
            if(node_ptr->right != nullptr) {
                if(math_sym.find_first_of(node_ptr->right->data) ==
std::string::npos) {
                    std::cout << node_ptr->right->data;
                } else {
                    print_expression_LPK(node_ptr->right);
                }
            } else {
                std::cout << '_';
            }
            std::cout << node_ptr->data;
            std::cout << ')';
        }
    /*
    1 * 2
    ()
    (a)
    ((a)*(b))
    (((a)+(b))*((c)-(d)))
    (((6)*(7))-((8)+(9)))
    Инфикс: a+b*c-d
    Префикс: *+ab-cd
    Постфикс: ab+cd-*
    */
    };

    bool checkstring(std::string& str) {
        std::string good_char = "*-+()";
        std::string new_str;
        int count = 0;
        for(auto iter = str.begin();iter != str.end(); iter++) {
            if(isspace(*iter)) {
                continue;
            } else if(*iter == '(') {
                count++;
            } else if(*iter == ')') {
```

```
                count--;
                if(count<0){
                    return false;
                }
            } else if(!isalnum(*iter) && good_char.find_first_of(*iter) ==
std::string::npos) {
                return false;
            }
            new_str.push_back(*iter);
        }
        if(count) {
            return false;
        }
        str = new_str;
        return true;
    }

    int main() {
        std::string str;
        std::cin.ignore (std::string::npos,'\n');
        std::getline(std::cin,str);
        if(!checkstring(str)) {
            std::cout << "Incorrect string!\n";
            return 0;
        }
        Tree<char> tree(str);
        return 0;
    }
```

ПРИЛОЖЕНИЕ В

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

№	Входные данные	Выходные данные	Комментарии
1	(((6)*(7))-((8)+(9)))	(-(*67)(+89)) ((67*)(89+)-) Expression: ((6*7)-(8+9)) Expression = 25	Префиксная, постфиксная и инфиксная форма. 42-17 = 25.
2	((a)-(b))	(-ab) (ab-) Expression: (a-b)	Простое буквенное дерево.
3	(((a)+(b))*((c)-(d)))	(*(+ab)(-cd)) ((ab+)(cd-)*) Expression: ((a+b)*(c-d))	В больших буквенных деревьях также не считается формула.
4	(((6)*())-((8)+()))	(-(*6_)(+8_)) ((6_*)(8_+)-) Expression: ((6*_)-(8+_)) Expression = -8	Пустые терминалы заменяются на '_' при записи и на 0 при подсчёте. 0-8=-8.
5	((())	Incorrect string!	Неравное число скобок.