ММИНОБРНАУКИ 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. 9384 _____ Николаев А.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Научиться создавать и обходить бинарные деревья.

Задание.

ВАРИАНТ 12.

12.

Построить дерево-формулу t из строки, задающей формулу в префиксной форме (перечисление узлов t в порядке КЛП);

Преобразовать дерево-формулу t, заменяя в нем все поддеревья , соответствующие формуле (f+f), на поддеревья, соответствующие формуле (2*f).

Выполнение работы.

В начале программа просит пользователя ввести строку, если в конце строки находится подстрока .txt, то программа понимает, что считать надо бинарное дерево из введенного файла.

Предполагается, что введенное бинарное дерево, введено правильно. Формирование происходит по следующему принципу. Функция при помощи рекурсии ищет последнюю ноду левой ветки и записывает в левую ветку значение, а затем в правую. Далее вызывается функция bt.change();, которая обходит дерево и если находит поддерево, соответствующее формуле (f+f), то в корень этого поддерева записывает *, а в левую ветку 2.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

Входные данные	Выходные данные
+ab +aa ++caa ++aac ++fg+aa	(a + b) ((2)* a) ((c + a)+ a) (((2)* a)+ c) ((f + g)+ ((2)* a))

Выводы.

В ходе выполнения данной лабораторной работы были получены навыки по созданию и обходу бинарных деревьев.

ПРИЛОЖЕНИЕ

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include "BinaryTree.h"

int main()
{
	BinaryTree bt;

	std::string tmp;
	std::cout << "Enter BT or FileName.txt: ";
	std::cin >> tmp;

	if (tmp.find(".txt", tmp.length() - 4) != std::string::npos)
	{
		std::ifstream in;
		in.open(tmp);
		std::getline(in, tmp);
		in.close();
	}

	bt.createBT(tmp);
	bt.printBT();

	std::cout << '\n';

	bt.change();
	bt.printBT();

	return 0;
}
```

Название файла: BinaryTree.h

```
#ifndef BINARYTREE_H
#define BINARYTREE_H

class BinaryTree {
public:
   BinaryTree();
   BinaryTree* left();
   BinaryTree* right();
   char RootBT();
   //BinaryTree* consBT(char tmp);
   void printBT();
   void change();
   void createBT(std::string& tmp);
   bool IsNull();
```

```
   ~BinaryTree();

private:
   BinaryTree* l;
   BinaryTree* r;
   char data;
};

#endif
```

Название файла: BinaryTree.cpp

```
#include <iostream>
#include "BinaryTree.h"

BinaryTree::BinaryTree()
{
	this->l = nullptr;
	this->r = nullptr;
}

BinaryTree* BinaryTree::left()
{
	if (this == nullptr) { std::cerr << "Error: Left(null) \n"; std::exit(1); }
	return this->l;
}

BinaryTree* BinaryTree::right()
{
	if (this == nullptr) { std::cerr << "Error: Right(null) \n"; std::exit(1); }
	return this->r;
}

char BinaryTree::RootBT()
{
	if (this == nullptr) { std::cerr << "Error: RootBT(null) \n"; exit(1); }
	else return data;
}

void BinaryTree::printBT() {
	if (!this->IsNull()) {
		if (!this->l->IsNull() && !this->r->IsNull() && !(this->RootBT() >= 'a' && this->RootBT() <= 'z'))
			std::cout << '(';
		this->l->printBT();
		std::cout << this->RootBT();
		this->r->printBT();
		if (!this->l->IsNull() && !this->r->IsNull() && !(this->RootBT() >= 'a' && this->RootBT() <= 'z'))
			std::cout << ')';
	}
```

```
}

void BinaryTree::change()
{
	if (!this->l->IsNull() && !this->r->IsNull())
	{
		if (this->data == '+' && this->l->data == this->r->data && this->l->data >= 'a' &&
this->l->data <= 'z' && this->r->data >= 'a' && this->r->data <= 'z')
		{
			this->data = '*';
			this->l->data = '2';
		}
		this->l->change();
		this->r->change();
	}
}

void BinaryTree::createBT(std::string& tmp)
{
	if (!tmp.empty())
	{
		if (this->l->IsNull() && this->r->IsNull())
		{
			if (tmp.front() >= 'a' && tmp.front() <= 'z')
			{
				this->data = tmp.front();
				tmp.erase(0, 1);
				this->l = new BinaryTree;
				this->r = new BinaryTree;
			}
			else
			{
				this->data = tmp.front();
				tmp.erase(0, 1);
				this->l = new BinaryTree;
				this->r = new BinaryTree;
				this->l->createBT(tmp);
				this->r->createBT(tmp);
			}
		}
		else
		{
			this->l->createBT(tmp);
			this->r->createBT(tmp);
		}
	}
}

bool BinaryTree::IsNull()
{
	return (this == nullptr);
}
```

```
BinaryTree::~BinaryTree()
{
}
```