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 _____ Прашутинский К.И.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Обработать ввод бинарного дерева. Заменить выражения на требуемые.

Задание.

Вариант 14:

- преобразовать дерево-формулу t, заменяя в нем все поддеревья, соответствующие формулам (f1 * (f2 + f3)) и ((f1 + f2) * f3), на поддеревья, соответствующие формулам ((f1 * f2) + (f1 * f3)) и ((f1 * f3) + (f2 * f3)).

Выполнение работы.

Предполагается, что вводимые данные истины, то бишь бинарное дерево введено корректно, а также используется короткая запись. Дерево формируется так: сначала создается операция над константами, затем на левой ветке создается переменная или же другая операция и так до тех пор, пока слева не появится констатнта и когда она появится, то начнется создание правых веток и заполнение их константами. После успешного ввода над деревом производится замена выражений.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

Входные данные	Выходные данные
+	(+)
***+abc*d+efk	((((c * a)+(c*b))*((d*f)+(d * e)))* k)
**+abc*+def	(((c * a)+(c*b))*((f * d)+(f*e)))

Выводы.

В ходе выполнения лабораторный работы было создано и обработано бинарное дерево.

ПРИЛОЖЕНИЕ
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <Windows.h>
#include "BT.h"

std::ifstream infile;

void printBT(BT* head);

int main() {
	infile.open("text.txt");
	if (!infile.is_open()) return 4;

	BT head;
	std::string tmp;

	infile >> tmp;

	head.createBT(tmp);
	head.~BT();
	printBT(&head);
	head.replacementBT();
	std::cout << '\n';
	printBT(&head);
	std::cout << '\n';
	system("pause");
	infile.close();
	return 0;
}

void printBT(BT* head) {
	if (!head->IsNull()) {
		if (!head->left()->IsNull() && !head->right()->IsNull() && !(head->RootBT() >= 'a' &&
head->RootBT() <= 'z'))
			std::cout << '(';
		printBT(head->left());
		std::cout << head->RootBT();
		printBT(head->right());
		if (!head->left()->IsNull() && !head->right()->IsNull() && !(head->RootBT() >= 'a' &&
head->RootBT() <= 'z'))
			std::cout << ')';

	}
}
```

Название файла: BT.h

```
#include <iostream>
```

```
#ifndef BT_H
#define BT_H



class BT {
public:
	BT();
	BT(char data, BT* l, BT* r);
	BT* left();
	BT* right();
	char RootBT();

	void setRoot(char ch);
	void setLeft(BT* b);
	void setRight(BT* b);

	void createBT(std::string& tmp);
	void replacementBT();

	bool IsNull();

//private:
	BT* l;
	BT* r;
	char data;
};

#endif // !BT_H
```

Название файла: BT.cpp

```
#include "BT.h"


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


BT::BT(char data, BT *l, BT *r) {
	this->data = data;
	this->l = l;
	this->r = r;
}


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

```
}

BT* BT::right() {
	if (this == nullptr) { std::cerr << "Error: Right(null) \n"; std::exit(2); }
	return this->r;
}

char BT::RootBT(){
	if (this == nullptr) { std::cerr << "Error: RootBT(null) \n"; exit(3); }
	else return data;
}

void BT::setRoot(char ch) {
	this->data = ch;
}

void BT::setLeft(BT* b) {
	this->l = b->l;
}


void BT::setRight(BT* b) {
	this->r = b->r;
}

void BT::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 BT;
				this->r = new BT;
			}
```

```
                else
                {
                        this->data = tmp.front();
                        tmp.erase(0, 1);
                        this->l = new BT;
                        this->r = new BT;
                        this->l->createBT(tmp);
                        this->r->createBT(tmp);
                }
        }
        else
        {
                this->l->createBT(tmp);
                this->r->createBT(tmp);
        }
    }
}

void BT::replacementBT() {
    if (!this->IsNull()) {
        if (!this->left()->IsNull() && !this->right()->IsNull() && this->RootBT() == '*') {
            if (this->left()->RootBT() == '+' && !this->left()->left()->IsNull() && !this->left()->right()->IsNull() && (this->right()->RootBT() >= 'a' && this->right()->RootBT() <= 'z')) {
                this->r->l = new BT;
                this->r->r = new BT;
                this->r->l->data = this->r->RootBT();
                this->r->r->data = this->left()->right()->RootBT();
                this->l->r->data = this->l->l->data;
                this->l->l->data = this->right()->RootBT();
                this->r->data = '*';
                this->l->data = '*';
                this->data = '+';
            }
            else if (this->right()->RootBT() == '+' && !this->right()->left()->IsNull() && !this->right()->right()->IsNull() && (this->left()->RootBT() >= 'a' && this->left()->RootBT() <= 'z')) {
                this->l->l = new BT;
                this->l->r = new BT;
                this->l->l->data = this->l->RootBT();
```

```
                                this->l->r->data = this->right()->right()->RootBT();
                                this->r->r->data = this->r->l->data;
                                this->r->l->data = this->left()->RootBT();
                                this->l->data = '*';
                                this->r->data = '*';
                                this->data = '+';
                        }
                }
                this->left()->replacementBT();
                this->right()->replacementBT();

        }

}

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