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

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
ТЕМА: Бинарные деревья

Студентка гр. 7381 _____ Кушкоева А.О.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2018

Цель работы.

Ознакомиться и закрепить на практики способы реализации бинарного дерева и работы с ним.

Формулировка задачи.

Формулу вида

< формула > ::= < терминал > | (< формула > < знак > < формула >)

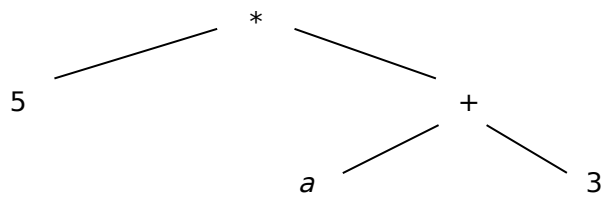
< знак > ::= + | - | *

< терминал > ::= 0 | 1 | ... | 9 | a | b | ... | z

можно представить в виде бинарного дерева («дерева-формулы») с элементами типа 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. Например, формула (5 * (a + 3)) представляется деревом-формулой, показанной на рисунке ниже.



Требуется:

а) для заданной формулы f построить дерево-формулу t;

б) для заданного дерева-формулы t напечатать соответствующую формулу f;

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

и) построить дерево-формулу t1 - производную дерева-формулы t по заданной переменной.

Описание алгоритма.

Необходимо считывать исходные данные в созданную структуру-формулу, содержащую в себе метку о виде содержимого формулы и само содержимое. Им может являться либо терминал, который, в свою очередь, может быть либо цифрой от 0 до 9, либо строчной буквой латинского алфавита (a,…, z); либо еще одну формулу. Если в формуле содержатся только терминалы-цифры, то программа выводит значения формулы. Так же программа имеет возможность строить дерево-производную заданной формулы по переменной, заданной пользователем.

Реализация задачи.

1. Функция, выводящая на экран и в файл сообщение об ошибке входных данных и производящая выход из программы.
   void ErrorMessege();
2. Функция считывания формулы с клавиатуры.
   void read_form(Formula* s, char x);
   Formula*& s – ссылка на указатель на голову текущего списка формул
   char x –переменная, хранящая в себе текущее значение символа входных данных
3. Функция печати формулы.
   void print(Formula* s);
4. Функция расчета результата формулы.
   void calculate(Formula* s, int* res, int* k);
   int* res – ссылка на переменную, хранящую результат вычислений
   bool* k – ссылка на переменную, определяющую, только ли цифры содержит формула
5. Функция, изображающая графически дерево-формулу.
   void build(Formula* s, int* depth, bool close[]);
   int* depth – переменная для определения текущей глубины дерева
   bool close[] - массив переменных для определения "закрытости" дерева с соответствующей глубиной, то есть программа работает со вторым его плечом

6. Функция для определения производной

   Formula* proizv(Formula* s, char x);

Тестирование.

Программа была собрана в компиляторе g++ в OS Linux Ubuntu 18.04 LTS.

Таблица 1 — примеры тестирования программы

Входные данные	Результат
((a+b)*3)	If you want to calculate the formula, you'll have to write tht formula without variables. ((a+b)*3) * | |--+ | | | |--a | | | |--b | |--3 ((a+b)*3) Calcutation can not be performed. ((a+b)*3) Please, enter the variable: ((a+b)*3)' = (3' * (a+b)) + (3 * (a+b)') 3' = 0 (a+b)' = a' + b' a' = 0 b' = 0 + | |--* | | | |--+ | | | | | |--a | | | | | |--b | | | |--0 |

	|--* | |--3 | |--+ | |--0 | |--0
(8*9))	If you want to calculate the formula, you'll have to write tht formula without variables. Error! Uncorrect input. The programm will be interructed.
((5+9)*3)	If you want to calculate the formula, you'll have to write tht formula without variables. ((5+9)*3) * | |--+ | | | |--5 | | | |--9 | |--3 The value of (5+9) is 14 The value of ((5+9)*3) is 42 ((5+9)*3) = 42 ((5+9)*3) Please, enter the variable: ((5+9)*3)' = (3' * (5+9)) + (3 * (5+9)') 3' = 0 (5+9)' = 5' + 9' 5' = 0 9' = 0 + | |--* | | | |--+ | | |

	| | |--5
	| | |
	| | |--9
	| |
	| |--0
	|
	|--*
	  |
	  |--3
	  |
	  |--+
	  |
	  |--0
	  |
	  |--0

Выводы.

В результате выполнения данной работы были освоены навыки использования бинарных деревьев. Закреплены навыки использования динамической памяти.

ПРИЛОЖЕНИЕ А
КОД ГОЛОВНОЙ ПРОГРАММЫ

```
#include <cstdlib>
#include "head.h"
#include "head.cpp"
using namespace std;

int main(){
	cout << "If you want to calculate the formula, you'll have to write tht formula without variables.\n";
	char x;

	Formula* head = new Formula;

	read_form(head); // вызов функции считывания формулы с клавиатуры

	cin.get(x);
	if (x != '\n') {
		ErrorMessege(); // если не все данные были считаны, то вывод об ошибке и прерывание программы
		return 0;
	}

	// печать дерева-формулы
	int depth = 0; // переменная для определения текущей глубины формулы
	bool close[50]; // массив переменных для определения "закрытости" дерева с соответствующей глубиной, то есть пограмма работает со вторым его плечом.

	for (int k = 1; k <= 49; k++)
		close[k] = 0;

	print(head); // вызов функции печати формулы
	cout << endl;
	build(head, &depth, close); // вызов функции печати дерева-формулы

	 // расчет результата формулы
	int k = 1; // переменная, определяющая наличие терминалов-букв
	int res; // переменная для записи в нее результата

	calculate(head, &res, &k); // вызов функции расчета результата
	print(head); // вызов функции печати формулы
	if (!k){ // печать о невозможности расчета результата из-за наличия в формуле не только цифр
```

```
		cout << "\nCalcutation can not be performed.\n";
	}
	else{ // печать результата
		cout << " = " << res << endl;
	}

	// печать дерева-формулы производной
	print(head); // вызов функции печати формулы
	cout << endl;

	cout << "Please, enter the variable: ";
	cin >> x; // запрос на введение переменной, по которой будет считаться производная

	Formula* p;
	p = new Formula; // объявление указателя на голову дерева-производной
	p = proizv(head, x); // вызов функции, создающей список дерев-поризводной
	int depth1 = 0;
	bool close1[50];
	for (int k = 1; k < 50; k++)
		close1[k] = 0;

	build(p, &depth1, close1);

	return 0;
}
```

ПРИЛОЖЕНИЕ Б.
ОПИСАНИЕ ФУНКЦИЙ, ИСПОЛЬЗУЮЩИХСЯ ДЛЯ РАБОТЫ С ФОРМУЛОЙ

```
#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

struct Formula;
struct Formula1{
Formula* f1;
Formula* f2;
char znak;
};
struct  Formula{
int tag;
union{
        union{
                int num;
                char let;
        }term;
        Formula1 form1;
}form;
};

void read_form(Formula* s);
void build(Formula* s, int* depth, bool close[]);
void print(Formula* s);
void calculate(Formula* s, int* res, int* k);
Formula* proizv(Formula* s, char x);
void ErrorMessege();
```

ПРОТОТИП В

РЕАЛИЗАЦИЯ ФУНКЦИЙ, ИСПОЛЬЗУЮЩИХСЯ ДЛЯ РАБОТЫ С ФОРМУЛОЙ

```
void ErrorMessege(){
	cout << "Error! Uncorrect input. The programm will be interructed.\n";

}

void read_form(Formula* s){ // рекурсивная функция считывания формулы с клавиатуры
	char x;
	cin.get(x);
	if ((x >= '0') && (x <= '9')){
		s->tag = 1;
		s->form.term.num = int(x) - 48;
	}
	else{
		if ((x >= 'a') && (x <= 'z')){
			s->tag = 2;
			s->form.term.let = x;
		}
		else{
			if (x != '(') ErrorMessege();
			else{
				s->tag = 0;
				s->form.form1.f1 = new Formula;
				read_form(s->form.form1.f1);
				cin.get(x);
				if ((x != '*') && (x != '+') && (x != '-')) ErrorMessege();
				else{
					s->form.form1.znak = x;
				}
				s->form.form1.f2 = new Formula;
				read_form(s->form.form1.f2);
				cin.get(x);
				if (x != ')') ErrorMessege();
			}
		}
	}
}

void print(Formula* s){
	if (s->tag == 1){
```

```
		cout << s->form.term.num;
	}

	if (s->tag == 2){
		cout << s->form.term.let;
	}

	if (!s->tag){
		cout << "(";
		print(s->form.form1.f1);
		cout << s->form.form1.znak;
		print(s->form.form1.f2);
		cout << ")";
	}
}

void calculate(Formula* s, int* res, int* k){
	if (*k){
		if (s->tag == 2)
			*k = 0;
		else{
			if (s->tag == 1){
				*res = s->form.term.num;
			}
			else{
				int res1, res2;
				calculate(s->form.form1.f1, &res1, k);
				calculate(s->form.form1.f2, &res2, k);
				if (s->form.form1.znak == '+') (*res) = res1 + res2;
				if (s->form.form1.znak == '-') (*res) = res1 - res2;
				if (s->form.form1.znak == '*') (*res) = res1 * res2;
				if (*k){
					cout << "The value of  ";
					print(s);
					cout << " is " << (*res) << endl;
				}
			}
		}
	}
}


void build(Formula* s, int* depth, bool close[]){
	(*depth)++;
	if (s->tag == 1)
```

```
        cout << s->form.term.num << endl;

if (s->tag == 2)
        cout << s->form.term.let << endl;

if (!s->tag){
        cout << s->form.form1.znak << endl;

        for (int j = 1; j <= (*depth); j++){
                if (!close[j])
                        cout << "|  ";

                else
                        cout << "   ";

        }

        cout << endl;

        for (int j = 1; j <= (*depth-1); j++){
                if (!close[j])
                        cout << "|  ";

                else
                        cout << "   ";
        }

        cout << "|--";

        build(s->form.form1.f1, depth, close);
        (*depth)--;
        for (int j = 1; j <= *depth; j++){
                if (!close[j])
                        cout << "|  ";

                else
                        cout << "   ";

        }
        cout << endl;

        for (int j = 1; j <= (*depth - 1); j++){
                if (!close[j])
                        cout << "|  ";
```

```
                else
                        cout << "   ";


        }
        cout << "|--";
        close[*depth] = 1;
        build(s->form.form1.f2, depth, close);
        (*depth)--;
        close[*depth] = 0;
    }
}


Formula* proizv(Formula* s, char x){
    Formula* p = new Formula;
    if (s->tag == 1){
        p->tag = 1;
        p->form.term.num = 0;
        cout << s->form.term.num << "' = 0\n";
    }
    if (s->tag == 2){
        p->tag = 2;
        if (s->form.term.let == x){
            p->form.term.let = '1';
            cout << s->form.term.let << "' = 1\n";
        }
        else{
            p->form.term.let = '0';
            cout << s->form.term.let << "' = 0\n";
        }
    }
    if (!s->tag){
        p->tag = 0;
        p->form.form1.f1 = new Formula;
        p->form.form1.f2 = new Formula;
        if ((s->form.form1.znak == '+') || (s->form.form1.znak == '-')) {
            print(s);
            cout << "' = ";
            print(s->form.form1.f1);
            cout << "' " << s->form.form1.znak << " ";
            print(s->form.form1.f2);
            cout << "'\n";
            p->form.form1.znak = s->form.form1.znak;
            p->form.form1.f1 = proizv(s->form.form1.f1, x);
```

```
                p->form.form1.f2 = proizv(s->form.form1.f2, x);
            }
            else{
                print (s);
                cout << "' = (";
                print(s->form.form1.f2);
                cout << "' * ";
                print(s->form.form1.f1);
                cout << ") + (";
                print(s->form.form1.f2);
                cout << " * ";
                print(s->form.form1.f1);
                cout << "')\n";
                p->form.form1.f1->form.form1.f1 = new Formula;
                p->form.form1.f1->form.form1.f2 = new Formula;
                p->form.form1.f2->form.form1.f1 = new Formula;
                p->form.form1.f2->form.form1.f2 = new Formula;
                p->form.form1.znak = '+';
                p->form.form1.f1->tag = 0;
                p->form.form1.f2->tag = 0;
                p->form.form1.f1->form.form1.znak = '*';
                p->form.form1.f2->form.form1.znak = '*';
                p->form.form1.f1->form.form1.f1 = s->form.form1.f1;
                p->form.form1.f1->form.form1.f2 = proizv(s->form.form1.f2, x);
                p->form.form1.f2->form.form1.f1 = s->form.form1.f2;
                p->form.form1.f2->form.form1.f2 = proizv(s->form.form1.f1, x);
            }
        }
        return p;
}
```