Додаток 1

Міністерство освіти і науки України Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 6

з дисципліни «Основи програмування – 2. Метидології програмування»

«Дерева»

Варіант 18

Виконав студент ІП-13 Король Валентин Олегович
(шифр, прізвище, ім'я, по батькові)

Перевірив Вєчерковська Анастасія Сергіївна
(прізвище, ім'я, по батькові)

Київ 2022

Лабораторна робота 6

Варіант 18

Відповідно до виразу, що читається з текстового файлу, побудувати дерево-формулу та обчислити значення цієї формули.

Код програми

C++

main.cpp

```
#include "Functions.h"
#include "Node.h"
using namespace std;

int main()
{

	ifstream inFile("text.txt");
	if (!inFile.is_open()) {
		cout << "Cant open file!\n\n";
		return -1;
	}
	vector<string> mass = readFile(inFile);
	inFile.close();
	ofstream outFile("result.txt");

	if (mass[0] == "") {
		cout << "Incorrect input of expression!\n\n";
		outFile << "Incorrect input of expression!";
		outFile.close();
		return -1;
	}

	outFile << "Entered expression:\n";
	outputVector(mass, outFile);

	BinaryTree Tree(mass);
	outFile << "\n\nBuilt tree-expression:\n\n";
	Tree.printTree(outFile);

	double result = Tree.countValueOfExpression();
	outFile << "\n\nResult of calculating the expression: " << result;
	cout << "The constructed tree and the obtained result were successfully output to a file.\n\n";

	Tree.clearMemory();
	outFile.close();

	system("pause");
	return 0;
```

}

Func

tions.

h

```
#pragma once
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Branch {
	int number;
Branch* left;
	Branch* right;
};
void AddElement(Branch*&, int); void
print_tree(Branch* root); void pr_obh(Branch*
branch, vector <int> &base);
bool is_element_in_vector(int elem, vector <int> base);
```

binarytree.h

```
#pragma once
#include "Node.h"
#include <vector>
#include <fstream>
using namespace std;

class BinaryTree
{
	static Node* createTree(const vector<string>& symbs, int& index);
	static void printTree(Node* node, int level, ofstream& out);
	static void clearMemory(Node* node);
	static double count(Node* node);

public:
	Node* Root;

	BinaryTree(const vector<string>& symbs);

	double countValueOfExpression();
	void printTree(ofstream& out);
	void clearMemory();
};
```

Node.h

```
#pragma once
#include <string>
using namespace std;

class Node
{
	string key;

public:
	Node* left,
		* right;
```

```
        Node();
        Node(string key);

        void setKey(string key);
        string getKey();
};
```

Functions.cpp

```
#include "Functions.h"

using namespace std;

bool isOperation(string symb)
{
        if (symb == "+" || symb == "-" || symb == "*" || symb == "/")
                return true;
        return false;
}

vector<string> readFile(ifstream& file)
{
        vector<string> mass;
        string expr;
        getline(file, expr);
        int k = 0, m;
        while (k < expr.length()) {
                string st = expr.substr(k, 1);
                if (st == " ") {
                        k++;
                }
                else if (st == "(" || st == ")" || isOperation(st)) {
                        mass.push_back(st);
                        k++;
                }
                else if (isdigit(expr[k])) {
                        m = k + 1;
                        while (m < expr.length()) {
                                if (isdigit(expr[m]) || expr[m] == '.') {
                                        m++;
                                }
                                else break;
                        }
                        string num = expr.substr(k, m - k);
                        mass.push_back(num);
                        k = m;
                }
                else return { "" };
        }
        return mass;
}

void outputVector(const vector<string>& vect, ofstream& out)
{
        for (int i = 0; i < vect.size() - 1; ++i) {
                out << vect[i] << " ";
        }
        out << vect[vect.size() - 1];
}
```

Binarytree.cpp

```
#include "BinaryTree.h"
#include "Functions.h"
using namespace std;
```

```
BinaryTree::BinaryTree(const vector<string>& symbs)
{
	int index = 0;
	Root = createTree(symbs, index);
}
Node* BinaryTree::createTree(const vector<string>& symbs, int& index)
{
	Node* node = new Node();
	while (index < symbs.size()) {
		if (symbs[index] == ")") {
			return node;
		}
		if (symbs[index] == "(") {
			node->left = createTree(symbs, ++index);
			index++;
		}
		if (isdigit(symbs[index][0])) {
			node->setKey(symbs[index]);
			return node;
		}
		if (isOperation(symbs[index])) {
			node->setKey(symbs[index]);
			node->right = createTree(symbs, ++index);
			index++;
		}
	}
	return node;
}

void BinaryTree::printTree(Node* node, int level, ofstream& out)
{
	if (node != NULL) {
		printTree(node->right, level + 1, out);
		for (int i = 0; i < level; ++i) {
			out << "\t";
		}
		out << "   " << node->getKey() << "\n";
		printTree(node->left, level + 1, out);
	}
}
void BinaryTree::printTree(ofstream& out)
{
	printTree(Root, 0, out);
}

void BinaryTree::clearMemory(Node* node)
{
	if (node->left == NULL && node->right == NULL) {
		free(node);
		return;
	}
	if (node->left != NULL) {
		clearMemory(node->left);
	}
	if (node->right != NULL) {
		clearMemory(node->right);
	}
}
void BinaryTree::clearMemory()
{
	clearMemory(Root);
}

double BinaryTree::count(Node* node)
```

```
{
	if (isdigit(node->getKey()[0])) {
		return stof(node->getKey());
	}
	switch (node->getKey()[0]) {
	case '+':
		return count(node->left) + count(node->right);
	case '-':
		return count(node->left) - count(node->right);
	case '*':
		return count(node->left) * count(node->right);
	case '/':
		return count(node->left) / count(node->right);
	}
}
double BinaryTree::countValueOfExpression()
{
	return count(Root);
}
```

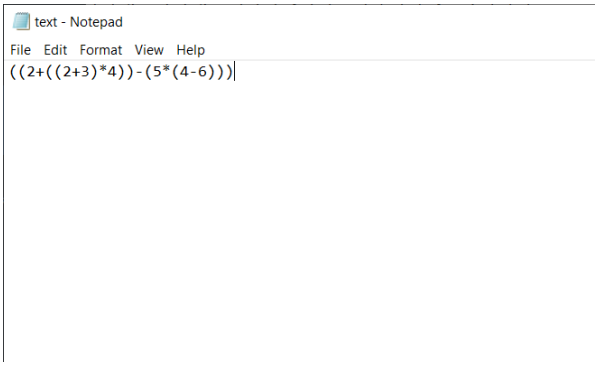
Node.cpp

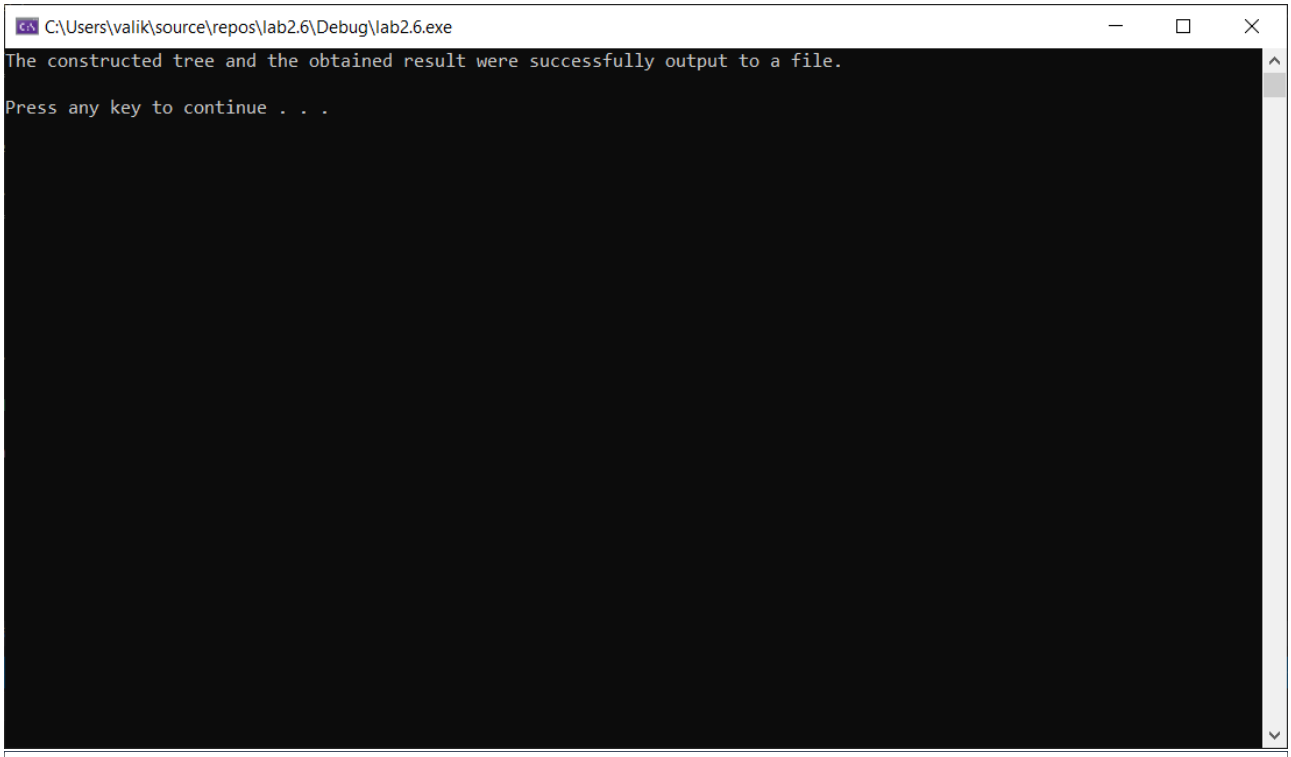
```
#include "Node.h"

Node::Node()
{
	this->key = "";
	left = NULL;
	right = NULL;
}
Node::Node(string key)
{
	this->key = key;
	left = NULL;
	right = NULL;
}

void Node::setKey(string key)
{
	this->key = key;
}
string Node::getKey()
{
	return key;
}
```

Тестування:





result - Notepad

File Edit Format View Help

```
Entered expression:
( ( 2 + ( ( 2 + 3 ) * 4 ) ) - ( 5 * ( 4 - 6 ) ) )

Built tree-expression:

                        6
                -
                        4
        *
                5
  -
                        4
                *
                                3
                        +
                                2
        +
                2


Result of calculating the expression: 32
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

Висновки:

Я вивчив особливості організації та оброки дерев.