

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»

«Дерева»

Варіант 28

Виконав студент ІП-13, Петров Ігор Ярославович
(шифр, прізвище, ім'я, по батькові)

Перевірів Вечерковська Анастасія Сергіївна
(прізвище, ім'я, по батькові)

Київ 2022

Лабораторна робота № 6 Варіант 28

Код C++

main.cpp

```
#include "tree.h"
#include "func.h"

int main() {
    string s;
    cout << "Enter expression separating operations with parentheses:\n";
    getline(cin, s);
    vector<string> expression = split(s);
    ExpressionTree Tree(expression);
    cout << "\nFixed expression:\n"; fixed_expression(expression);
    cout << "\nTree from infix expression:\n";
    Tree.print();
    Tree.modify();
    cout << "\nModified tree:\n\n";
    Tree.print();
    Tree.free_memory();
}
```

tree.cpp

```
#include "tree.h"
#include "func.h"

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

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

void Node::set_value(string key)
{
    this->value = key;
}

string Node::get_value()
{
    return value;
}

ExpressionTree::ExpressionTree(vector<string> symbs)
{
    int index = 0;
    Root = createTree(symbs, index);
}

Node* ExpressionTree::createTree(vector<string> symbs, int& index)
{
    Node* node = new Node();
    while (index < symbs.size()) {
        if (symbs[index] == "(") {
```

```

        return node;
    }
    if (syms[index] == "(") {
        node->left = createTree(syms, ++index);
        index++;
    }
    if (isdigit(syms[index][0])) {
        node->set_value(syms[index]);
        return node;
    }
    if (is_operator(syms[index])) {
        node->set_value(syms[index]);
        node->right = createTree(syms, ++index);
        index++;
    }
}
return node;
}

void ExpressionTree::print(Node* node, int space)
{
    if (node != NULL) {
        print(node->right, space + 1);
        for (int i = 0; i < space; i++) {
            cout << "\t";
        }
        cout << " " << node->get_value() << "\n";
        print(node->left, space + 1);
    }
}

void ExpressionTree::print()
{
    print(Root, 0);
}

void ExpressionTree::modify() {
    modify(Root);
}

void ExpressionTree::modify(Node* node) {
    if (node == NULL) {
        return;
    }
    create_final(node);
    modify(node->left);
    modify(node->right);
}

void ExpressionTree::create_final(Node* node) {
    if (check_formula(node) == 1) {
        string mult = node->right->get_value(), plus = node->left->right->get_value();
        node->set_value(node->left->get_value());
        node->left->set_value("*");
        node->right->set_value("*");
        node->left->right->set_value(mult);
        node->right->left = new Node;
        node->right->left->set_value(plus);
        node->right->right = new Node;
        node->right->right->set_value(mult);
    }
    else if (check_formula(node) == 2){
        string mult = node->left->get_value(), plus = node->right->right->get_value();
        node->set_value(node->right->get_value());
        node->right->set_value("*");
        node->left->set_value("*");
        node->right->right->set_value(mult);
        node->left->left = new Node;
    }
}

```

```
node->left->left->set_value(plus);
node->left->right = new Node;
node->left->right->set_value(mult);
}
}

int ExpressionTree::check_formula(Node* node) {
    if (node->get_value() == "*" && is_leaf(node->right) && (node->left->get_value() == "+" || node->
left->get_value() == "-") && is_leaf(node->left->left) && is_leaf(node->left->right)) {
        return 1;
    }
    else if (node->get_value() == "*" && is_leaf(node->left) && (node->right->get_value() == "+" ||
node->right->get_value() == "-") && is_leaf(node->right->right) && is_leaf(node->right->left)) {
        return 2;
    }
    return -1;
}

bool ExpressionTree::is_leaf(Node* node) {
    return(isdigit(node->get_value()[0]));
}

void ExpressionTree::free_memory(Node* node)
{
    if (node->left == NULL && node->right == NULL) {
        delete(node);
        return;
    }
    if (node->left != NULL) {
        free_memory(node->left);
    }
    if (node->right != NULL) {
        free_memory(node->right);
    }
}

void ExpressionTree::free_memory()
{
    free_memory(Root);
}
```

tree.h

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

using namespace std;

class Point {
private:
    double x, y;
public:
    Point() = default;
    Point(vector<int> vec);
    double get_x() { return x; };
    double get_y() { return y; };
};

class TQuadrangle {
protected:
    Point point1, point2, point3, point4;
public:
    TQuadrangle(vector <vector <int>>);
```

```
virtual double get_p() = 0;
virtual double get_s() = 0;
};

class Parallelogram : public TQuadrangle {
public:
    Parallelogram(vector<vector<int>> points) : TQuadrangle(points) {};
    double get_p() override;
    double get_s() override;
};

class Rectangle : public Parallelogram {
public:
    Rectangle(vector<vector<int>> points) : Parallelogram(points) {};
    double get_s() override;
};

class Square : public Rectangle {
public:
    Square(vector<vector<int>> points) : Rectangle(points) {};
    double get_p() override;
    double get_s() override;
};

double side_distance(Point, Point);
```

func.h

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

using namespace std;

vector<string> split(string);
void fixed_expression(vector<string>);
bool
is_operator(string);
```

func.cpp

```
#include "func.h"

vector<string> split(string line) {
    vector<string> res;
    string operators = "+-*/()";
    string slice = "";
    line = "(" + line + ")";
    for (char symb : line) {
        if (symb != ' ') {
            if (operators.find(symb) != string::npos) {
                if (slice.length() > 0) {
                    res.push_back(slice);
                    slice = "";
                }
                res.push_back(string(1, symb));
            }
            else slice += symb;
        }
    }
    return res;
}

void fixed_expression(vector<string> vec) {
    if (vec.size() > 2){
        for (int i = 1; i < vec.size() - 1; i ++)
```

```
{
    cout << vec[i] << ' ';
}
cout << "\n\n";
}

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

Результат роботи

```
Enter expression separating operations with parentheses:
((2+2)*3)-((1-4)*5)

Fixed expression:
( ( 2 + 2 ) * 3 ) - ( ( 1 - 4 ) * 5 )

Tree from infix expression:
      *
     / \
    /   \
   /     \
  /       \
 /         \
-           5
 / \
*   -
 / \ / \
3  4 1 4
 / \
*   +
 / \
2  2
 / \
2  2

Modified tree:
      *
     / \
    /   \
   /     \
  /       \
 /         \
-           5
 / \
*   -
 / \ / \
5  4 1 4
 / \
*   +
 / \
3  2
 / \
*   +
 / \
2  3
 / \
*   +
 / \
3  2
```