

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. Модульне програмування»

«Дерева»
Варіант 3

Виконав студент ІП-13, Баран Софія Володимирівна
(шифр, прізвище, ім'я, по батькові)

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

Лабораторна робота №6

Дерева

Варіант 3

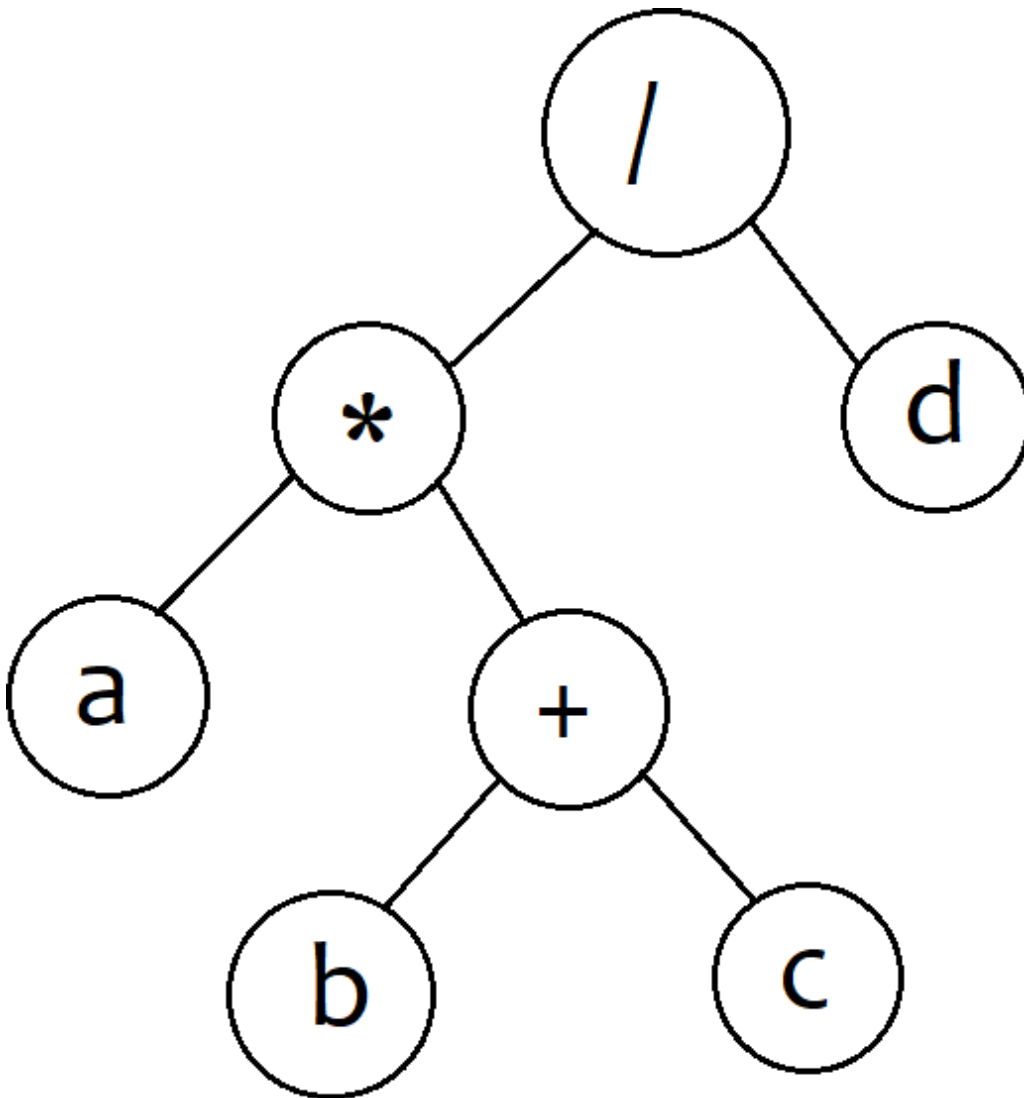
Задача

3. Побудувати дерево, що відображає формулу $(a*(b+c))/d$, де коренем дерева та його підкореннями є операції $"*, +, -, /"$, а листками - змінні a, b, c, d . Надрукувати дерево переліком своїх вершин на рівнях, які містять відповідні вершини.

Реалізація дерева

Для представлення формули $(a*(b+c))/d$ використаємо бінарне дерево виразів. В ньому операції $*$, $+$, $/$ будуть представлені коренем і підкореннями дерева, а листками будуть змінні a , b , c і d .

Ручна реалізація дерева:



Код

C++

main.cpp:

```
#include "functions.h"
```

```
int main() {  
    float a = capture_value('a');  
    float b = capture_value('b');  
    float c = capture_value('c');  
    float d = capture_d();  
  
    Tree tree = build_tree(a, b, c, d);  
  
    float tree_value = tree.calculate();  
    printf("Tree value: %.3f\n", tree_value);  
  
    tree.print();  
}
```

classes.h:

```
#ifndef INC_2LABWORK_6_CLASSES_H
#define INC_2LABWORK_6_CLASSES_H

#endif

#include <iostream>
using namespace std;

class Node{
private:
    float data;
    char action_operator;

    Node* left;
    Node* right;

public:
    Node(float data);
    Node(char action_operator);

    float get_data() const;
    char get_operator() const;
    Node* get_left();
    Node* get_right();

    void set_left(Node* new_left);
    void set_right(Node* new_right);
};

class Tree{
private:
    Node* root;

    float calculate(Node* node);
    float evaluate_operator(Node *node, float left_sum, float
right_sum);
    void print(Node* node, int depth);
public:
    Tree(Node* root);
    float calculate();
    void print();
};
```

classes.cpp:

```
#include "classes.h"

Node::Node(float data) {

    this->data = data;
    this->action_operator = '\000';
    left = nullptr;
    right = nullptr;
}

Node::Node(char action_operator) {
    this->data = 0;
    this->action_operator = action_operator;
    left = nullptr;
    right = nullptr;
}

float Node::get_data() const {
    return data;
}

char Node::get_operator() const {
    return action_operator;
}

Node *Node::get_left() {
    return left;
}

Node *Node::get_right() {
    return right;
}

void Node::set_left(Node *new_left) {
    left = new_left;
}

void Node::set_right(Node *new_right) {
    right = new_right;
}
```

```
}
```

```
Tree::Tree(Node *root) {  
    this->root = root;  
}
```

```
float Tree::evaluate_operator(Node *node, float left_sum, float  
right_sum) {  
    if(node->get_operator() == '+'){  
        float sum = left_sum + right_sum;  
        return sum;  
    }  
    else if(node->get_operator() == '-'){  
        float sub = left_sum - right_sum;  
        return sub;  
    }  
    else if(node->get_operator() == '*'){  
        float mult = left_sum * right_sum;  
        return mult;  
    }  
    else{  
        float div = left_sum / right_sum;  
        return div;  
    }  
}
```

```
float Tree::calculate() {  
    float left_sum = calculate(root->get_left());  
    float right_sum = calculate(root->get_right());  
  
    float sum = evaluate_operator(root, left_sum, right_sum);  
}
```

```

        return sum;
    }

float Tree::calculate(Node *node) {
    if(node->get_left() == nullptr && node->get_right() ==
nullptr){
        return node->get_data();
    }

    float left_sum = calculate(node->get_left());
    float right_sum = calculate(node->get_right());

    float sum = evaluate_operator(node, left_sum, right_sum);

    return sum;
}

void print_node(Node *node, int depth){
    for (int i = 0; i < depth; ++i) {
        cout << "    ";
    }

    if(node->get_operator() != '\000')
        cout << "-> " << node->get_operator() << endl;
    else
        cout << "-> " << node->get_data() << endl;
}

void Tree::print() {

```



```
int depth = 0;

this->print(root->get_left(), depth + 1);
print_node(root, depth);
this->print(root->get_right(), depth + 1);
}

void Tree::print(Node *node, int depth) {
    if(node != nullptr){
        this->print(node->get_left(), depth + 1);
        print_node(node, depth);
        this->print(node->get_right(), depth + 1);
    }
}
```

functions.h:

```
#ifndef INC_2LABWORK_6_FUNCTIONS_H
#define INC_2LABWORK_6_FUNCTIONS_H

#endif

#include "classes.h"

float capture_value(char value);
float capture_d();
Tree build_tree(float a, float b, float c, float d);
```

functions.cpp:

```
#include "functions.h"
```

```
float capture_value(char value){  
    printf("Enter %c value: ", value);
```

```
    float a;  
    cin >> a;
```

```
    return a;
```

```
}
```

```
float capture_d(){  
    cout << "Enter d value: ";
```

```
    float d;  
    cin >> d;
```

```
    while(d == 0){  
        cout << "d value can't be 0. Enter again: ";  
        cin >> d;  
    }
```

```
    return d;
```

```
}
```

```
Tree build_tree(float a, float b, float c, float d){  
    Node *root = new Node('/');
```

```
    Tree tree(root);
```

```
    // ініціалізуємо лівий вузол кореня  
    Node *multiply_node = new Node('*');  
    root->set_left(multiply_node);  
    // ініціалізуємо правий вузол кореня  
    Node *d_node = new Node(d);  
    root->set_right(d_node);
```

```
    // ініціалізуємо лівий вузол вузла множення  
    Node *a_node = new Node(a);  
    multiply_node->set_left(a_node);  
    // ініціалізуємо правий вузол вузла множення  
    Node *plus_node = new Node('+');  
    multiply_node->set_right(plus_node);
```

```
// ініціалізуємо лівий вузол вузла додавання
Node *b_node = new Node(b);
plus_node->set_left(b_node);
// ініціалізуємо правий вузол вузла додавання
Node *c_node = new Node(c);
plus_node->set_right(c_node);

return tree;
}
```

Результат виконання програми

```
Enter a value:1.1
Enter b value:2.2
Enter c value:3.3
Enter d value:4.4
Tree value: 1.375
```

```
-> 1.1
```

```
-> *
```

```
-> 2.2
```

```
-> +
```

```
-> 3.3
```

```
-> /
```

```
-> 4.4
```

```
Process finished with exit code 0
```

Калькулятор

Инженерный

$(1,1 \times (2,2 + 3,3)) \div 4,4 =$

1,375

RAD

F-E

MC

MR

M+

M-

MS

M*

Тригонометрия

Функция

2^{nd}

π

e

CE

$\leftarrow \boxtimes$

x^2

$\frac{1}{x}$

$|x|$

exp

mod