**СТРУКТУРА**

**звіту з лабораторної роботи**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**імені ІГОРЯ СІКОРСЬКОГО”**

**ЗВІТ**

**з лабораторної роботи №7**

**з навчальної дисципліни “Програмування складних алгоритмів”**

**Тема: Бінарні Дерева**

**Варіант №1**

**Виконав студент групи ТР–15**

Руденко Владислав Ігорович

з оцінкою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

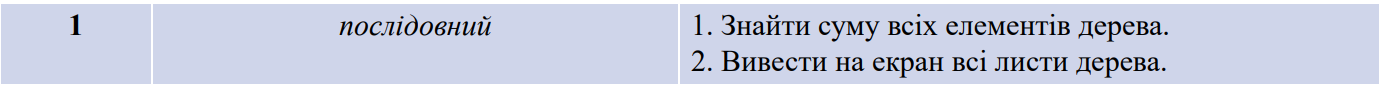
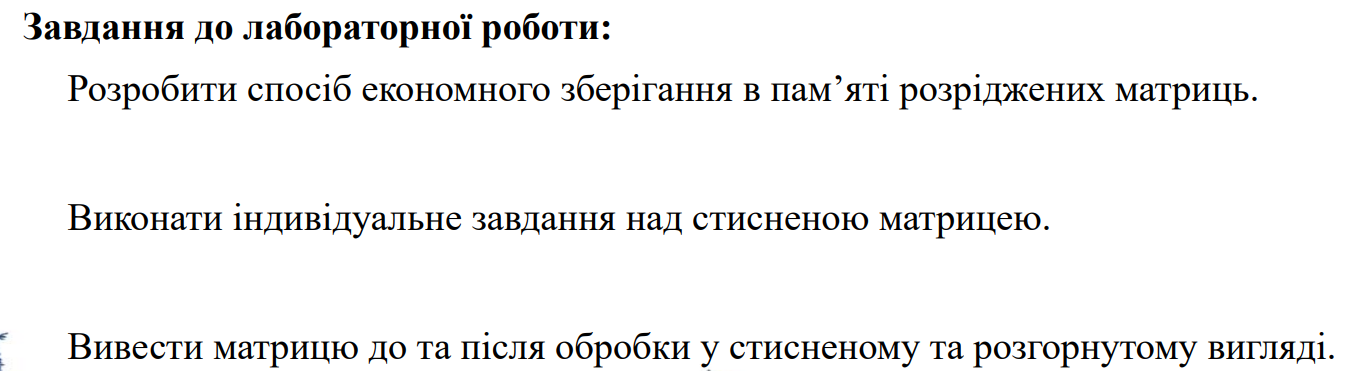
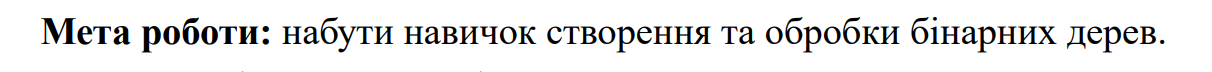
**Перевірив доцент кафедри**

Андрій ОНИСЬКО\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ року

**Київ 2022**

**І. Завдання + Мета:**

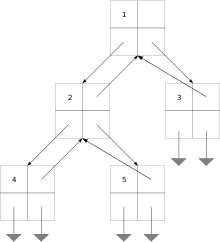


1. **Теоретична частина.**

У [програмуванні](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F" \o "Програмування) двійкове дерево — [структура даних](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85" \o "Структура даних) у вигляді [дерева](https://uk.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)" \o "Дерево (структура даних)), в якому кожна вершина має не більше двох дітей. Зазвичай такі діти називаються правим та лівим. На базі двійкових дерев будуються такі структури, як [двійкові дерева пошуку](https://uk.wikipedia.org/wiki/%D0%91%D1%96%D0%BD%D0%B0%D1%80%D0%BD%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D1%88%D1%83%D0%BA%D1%83" \o "Бінарне дерево пошуку) та [двійкові купи](https://uk.wikipedia.org/wiki/%D0%91%D1%96%D0%BD%D0%B0%D1%80%D0%BD%D0%B0_%D0%BA%D1%83%D0%BF%D0%B0" \o "Бінарна купа).

Залежно від задач, які вирішуються цими структурами та можливостей тої чи іншої [мови програмування](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F" \o "Мова програмування), існує декілька варіантів конструювання двійкових дерев.

Реалізація з використанням [вказівників](https://uk.wikipedia.org/wiki/%D0%92%D0%BA%D0%B0%D0%B7%D1%96%D0%B2%D0%BD%D0%B8%D0%BA" \o "Вказівник) передбачає зберігання в кожній вершині дерева x, разом із даними власне цієї вершини, також двох полів - правого та лівого (right[x] та left[x]), які містять вказівники на відповідних дітей цієї вершини.

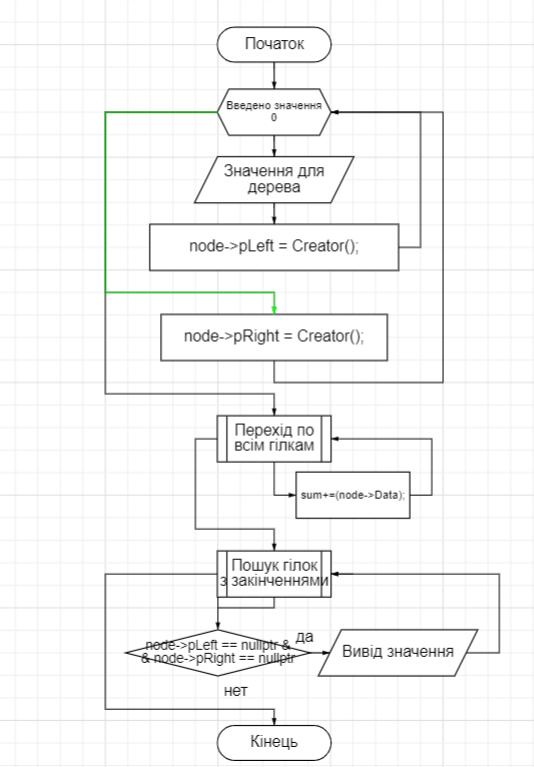
[](https://uk.wikipedia.org/wiki/Ð¤Ð°Ð¹Ð»:Binary_tree_parent_pointers.svg)

Змінена реалізація двійкового дерева. Кожна вершина містить також вказівник на батьківську вершину

Також іноді додається вказівник p[x] на батьківську вершину. Це спрощує деякі алгоритми та виявляється корисним, коли необхідно забезпечити швидкий доступ до батьківської вершини. Іноді достатньо тільки вказівника на батьківську вершину. Взагалі будь-яке орієнтоване дерево можна описати, знаючи тільки зв'язки від дітей до батьківської вершини.

**ІІ. Результати виконання лабораторної роботи.**

**Блок Схеми:**



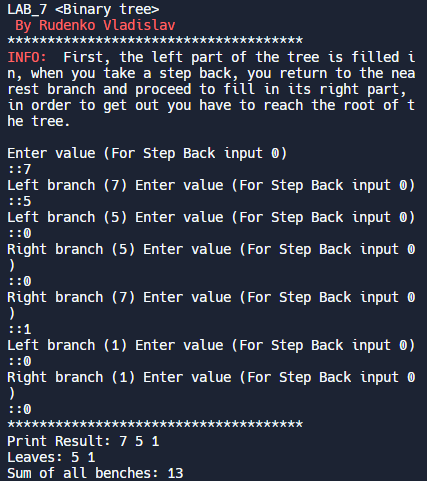
**Опис програми (написана на мові Сі++):**

При виконання роботи було створено декілька рекурсивних функцій, які дають змогу зкоротити код та покращити його ефективність. Так першою функцією є запис значення, поки користувач не натисне клавішу 0, значення будуть записуватись за певними гілками загального дерева, як тільки користувач натисне 0, відбудеться повернення до найближчої гілки і перехід на праву частину цієї гілки, за для завершення виконання, користувач мається крок за кроком перейти до корення дерева, та натиснути ще раз 0.

Після завершення запису програма автоматично починає орпацювання, а саме пошук по дереву та підрахунок сум його вітв та листів, оке=ремо також виводиться кількість і значення листів.

**Результати роботи:**

**Результати роботи у вигляді скріншотів:**



**Посилання на repl.it:**

**<https://replit.com/join/gcolrpfmxa-hetik>**

**Ш. Висновки.**

В ході виконання Лабораторної роботи №7 було розроблено бінарне дерево, яка надсилає значення до необхідних віток залежно від введених значень. Було ознайомлено з особливостями використання таких типів дерев та способи пошуку в ньому. В результаті програми відбувається підрахунок суми усіх віток дерева, та візуалізація усіх листів дерева. Було виявлено та справлено деяку кількість помилок.

**Програмний код:  
main.cpp:**

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

int sum;

#define RED "\x1b[31m"

#define RESET "\x1b[0m"

struct Node

{

int Data;

Node\* pLeft;

Node\* pRight;

};

struct Node\* Creator()

{

int choose;

struct Node\* node;

node = new Node;

cout<<"Enter value (For Step Back input 0)\n::";

cin>>choose;

if(choose == 0) return 0;

node->Data = choose;

cout<<"Left branch ("<< choose <<") ";

node->pLeft = Creator();

cout<<"Right branch ("<< choose <<") ";

node->pRight = Creator();

return node;

};

void AllbechsSum(Node\* tree);

void printLeaves(Node\* tree);

int main()

{

struct Node\* tree = nullptr;

cout << "LAB\_7 <Binary tree>\n" RED " By Rudenko Vladislav " RESET " \n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n" RED "INFO: " RESET " First, the left part of the tree is filled in, when you take a step back, you return to the nearest branch and proceed to fill in its right part, in order to get out you have to reach the root of the tree.\n\n";

tree = Creator();

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nPrint Result: ";

AllbechsSum(tree);

printLeaves(tree);

cout<< "\nSum of all benches: " << sum << "\n";

}

void printLeaves(Node\* tree)

{

if (tree == nullptr) return;

queue<Node\*> TreeQueue;

TreeQueue.push(tree);

cout<<"\nLeaves: ";

while (TreeQueue.size() != 0)

{

Node\* node = TreeQueue.front();

if(node->pLeft == nullptr && node->pRight == nullptr)

cout<<node->Data << " ";

TreeQueue.pop();

if (node->pLeft != nullptr) TreeQueue.push(node->pLeft);

if (node->pRight != nullptr) TreeQueue.push(node->pRight);

}

}

void AllbechsSum(Node\* tree)

{

if (tree == nullptr) return;

queue<Node\*> TreeQueue;

TreeQueue.push(tree);

while (TreeQueue.size() != 0)

{

Node\* node = TreeQueue.front();

cout<<node->Data<<" ";

sum+=(node->Data);

TreeQueue.pop();

if (node->pLeft != nullptr)

TreeQueue.push(node->pLeft);

if (node->pRight != nullptr)

TreeQueue.push(node->pRight);

}

}