**Черкаський національний університет імені Богдана Хмельницького**

(повне найменування вищого навчального закладу)

**Кафедра програмного забезпечення автоматизованих систем**

(повна назва кафедри)

# **КУРСОВА РОБОТА**

**з дисципліни “Програмування та алгоритмічні мови”**

**НА ТЕМУ «Автоматичне створення дерева математичного вигляду»**

Студента 1 курсу, групи КС-19

напряму підготовки **\_\_** «Програмна нженерія»

спеціальності «121 Інженерія програмного\_\_\_\_

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**забезпечення»**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**Падалка І.А. **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(прізвище та ініціали)

Керівник**\_\_\_\_\_\_\_**ст.в. Гребенович Ю.Є.**\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала: **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Кількість балів: **\_\_\_\_\_\_\_** Оцінка ECTS: **\_\_\_\_\_\_**

Члени комісії:**\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(підпис) (прізвище та ініціали)

**\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(підпис) (прізвище та ініціали)

**\_\_\_\_\_\_\_\_\_\_\_\_**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(підпис) (прізвище та ініціали)

**м. Черкаси – 2020 рік**

Зміст курсової роботи:

Вступ……………………………………………………………………………...…..3

Розділ 1. Огляд методів створення дерева математичного вигляду……….......…5

* 1. Бінарне дерево (binary tree)……………………………………...…...…..6
  2. Створення дерева на основі лямбда-виразів…………………....…….....7
  3. Створення дерев виразів за допомогою API-інтерфейсу……......……..7
  4. Синтаксичний аналіз дерева…………………………………...……..…..9

Розділ 2. Проектування програми створення дерева математичного виразу…..10

* 1. Додавання даних…………………………………………………..…….10
  2. Пошук вузла дерева за значенням……………………….……..………11
  3. Видалення вузла бінарного дерева……………………….……….……11
  4. Виведення дерева на екран………………………………….…………..14
  5. Програма для роботи з створенням дерева………………….…………14

Розділ 3. Тестування програми………………………………………………..…..15

3.1. Тестування програми………………………………………………..….15

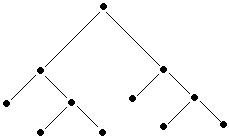
Висновок……………………………………………………………….……………16

Список використаних джерел……………………………………………..………17

Вступ

Поняття дерева широко застосовують у багатьох розділах математики й інформатики. Наприклад, дерева використовують як інструмент обчислень, зручний спосіб збереження даних, їх сортування чи пошуку.

Деревом називають зв’язний граф без простих циклів. Граф, який не містить простих циклів і складається з n компонент зв’язності, називають лісом з n дерев. З означення випливає, що дерева й ліси являють собою прості графи.



Мета моєї роботи полягає в написанні програми, яка автоматично створює дерево математичного вигляду.

Керуючись поставленою метою ставлю перед собою завдання:

1. Проаналізувати методи створення дерева математичного вигляду:
2. Визначити чіткі етапи виконання курсової роботи;
3. Дослідити та виявити можливі недоліки в програмі.

**Методи дослідження.** Для досягнення поставленої в роботі мети використані збір інформації з Intertet-сайтів, аналогічних програмних продуктів, книгах та фахових журналах.

Структура роботи відбиває поставлені перед дослідженням мету та завдання. Загальний обсяг її становить 42 сторінки.

Розділ 1. Огляд методів створення дерева математичного вигляду

Для початку потрібно проаналізувати та обрати необхідні варіанти для реалізації мого проекту. Адже, від цього буде залежити майбутній вигляд, функціонал, практичність, та швидкодія програми. Потрібно розглянути всі варіанти адже в багатьох з них можливо скривається похибка.

У цьому розділі ми розглянемо які є можливі варіанти програм, форми зчитування вхідної інформації та формат виведення.

Також вартим уваги є правильне обрання методів реалізації. У цьому розділі роглянемо методи побудови за допомогою засобів C#.

Перше, що розглянемо, – це програма написана за допомогою Console Application.

Доводи за:

1. простота та легкість написання програми;
2. короткий код;
3. швидкодія завдяки простоті програми.

Недоліки:

1. незручне для користувача введення даних;
2. відсутність будь-якого графічного дизайну;
3. застосування сторонніх програм для виведення;

Розглянемо другий варіант – написання програми за допомогою Windows Form

Доводи за:

1. зручне введення даних для користувача;
2. прості, але потужні механізми для управління графічним інтерфейсом;
3. гарна візуалізація за допомогою різноманітних засобів;
4. можливість вбудовування переглядача результату у самій програмі без відкриття сторонніх програм.

Недоліки:

1. складніше написання програми в порівняні з Console Application.

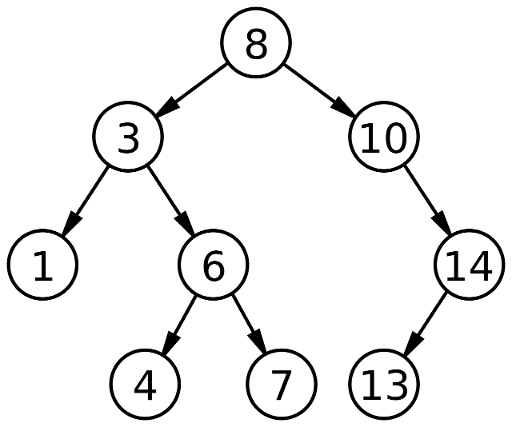
Вирішив взяти за основу використання бінарного дерева (binary tree).

Бінарне дерево (binary tree) - це структура даних, яка складається з вузлів, при цьому кожен вузол може мати не більше двох дочірніх. Перший вузол називається кореневим або батьківським, а дочірні - правим і лівим спадкоємцем (нащадком).

**1.1. Бінарне дерево (binary tree)**

Двійкове дерево пошуку (binary search tree) - це бінарне дерево, яке відповідає наступним умовам:

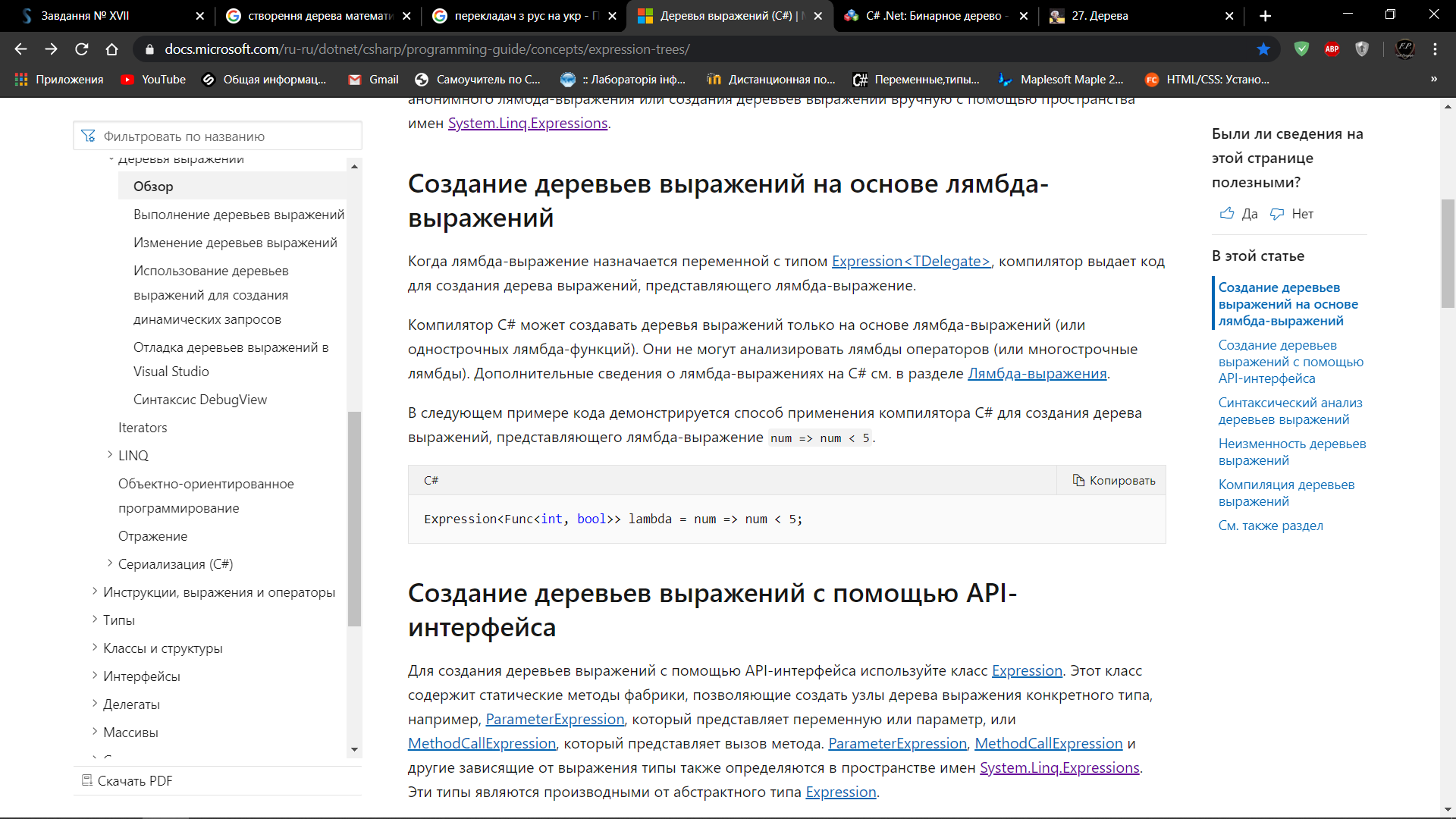
* Дочірні вузли є двійковими деревами пошуку;
* Для довільного вузла:
* Всі значення лівого піддерева повинні бути менші за значення батьківського вузла;
* Всі значення правого піддерева повинні бути більші за значення батьківського вузла.



1.2. Створення дерева на основі лямбда-виразів

Коли лямбда-вираз призначається змінної з типом Expression <TDelegate>, компілятор видає код для створення дерева виразів, що представляє лямбда-вираз.

Компілятор C # може створювати дерева виразів тільки на основі лямбда-виразів (або однорядкових лямбда-функцій). Вони не можуть аналізувати лямбда операторів (або багаторядкові лямбда). Додаткові відомості про лямбда-виразах на C # см. В розділі Лямбда-вирази.

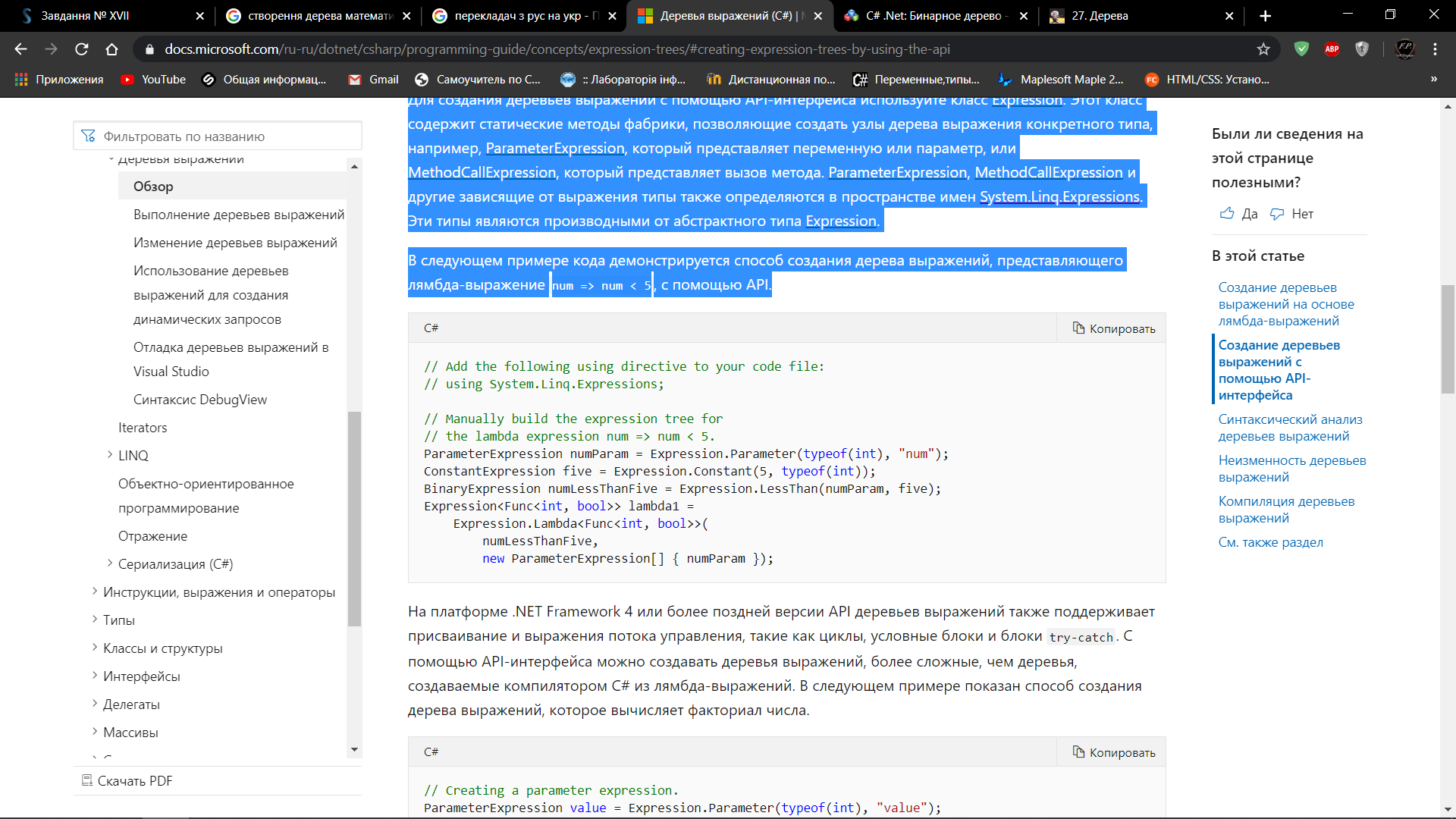
У наступному прикладі коду демонструється спосіб застосування компілятора C # для створення дерева виразів, що представляє лямбда-вираз num => num <5.

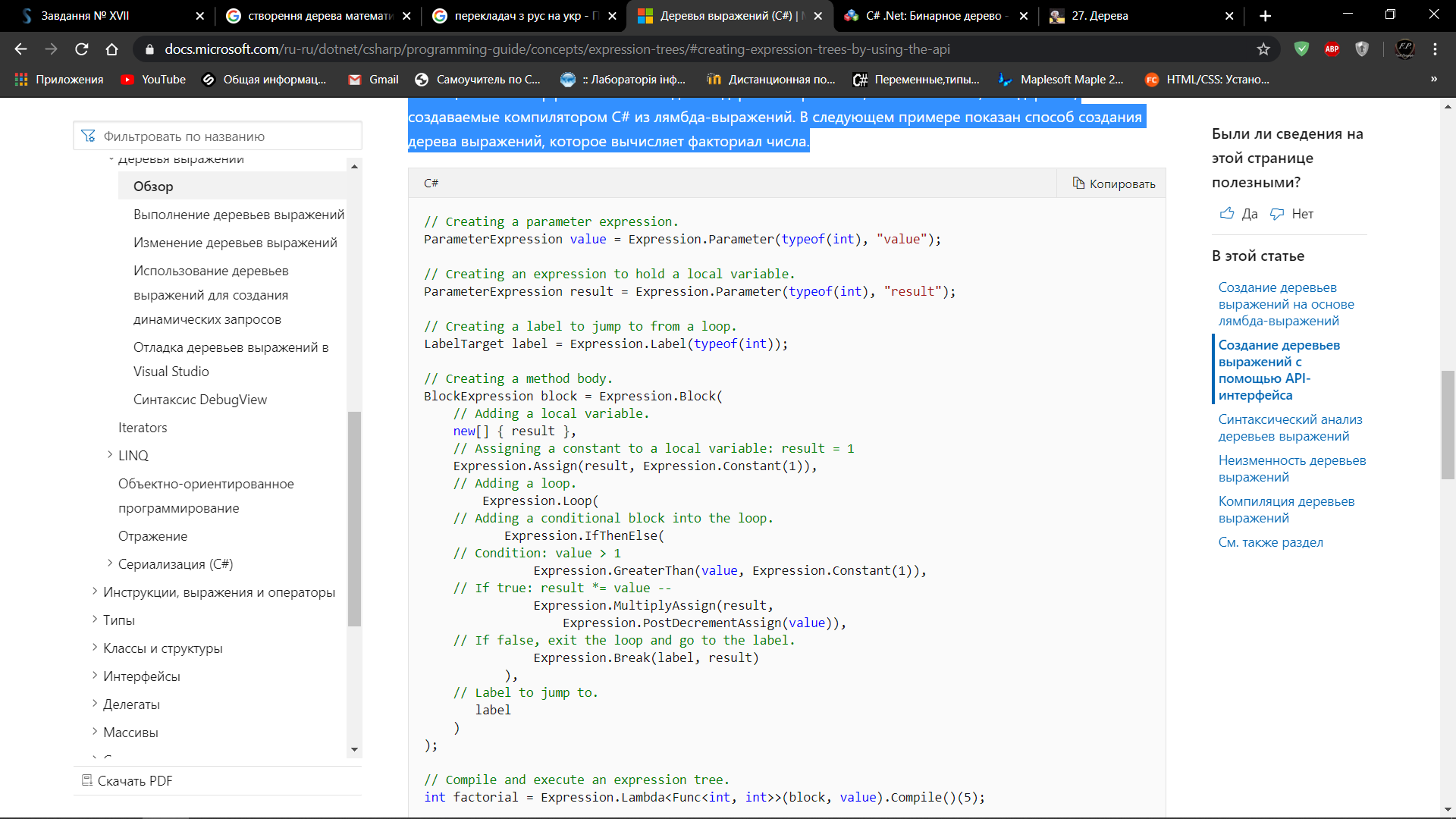
1.3. Створення дерев виразів за допомогою API-інтерфейсу

Для створення дерев виразів за допомогою API-інтерфейсу використовуйте клас Expression. Цей клас містить статичні методи фабрики, що дозволяють створити вузли дерева вираження конкретного типу, наприклад, ParameterExpression, який представляє змінну або параметр, або MethodCallExpression, який представляє виклик методу.

ParameterExpression, MethodCallExpression і інші залежні від виразу типи також визначаються в просторі імен System.Linq.Expressions. Ці типи є похідними від абстрактного типу Expression.

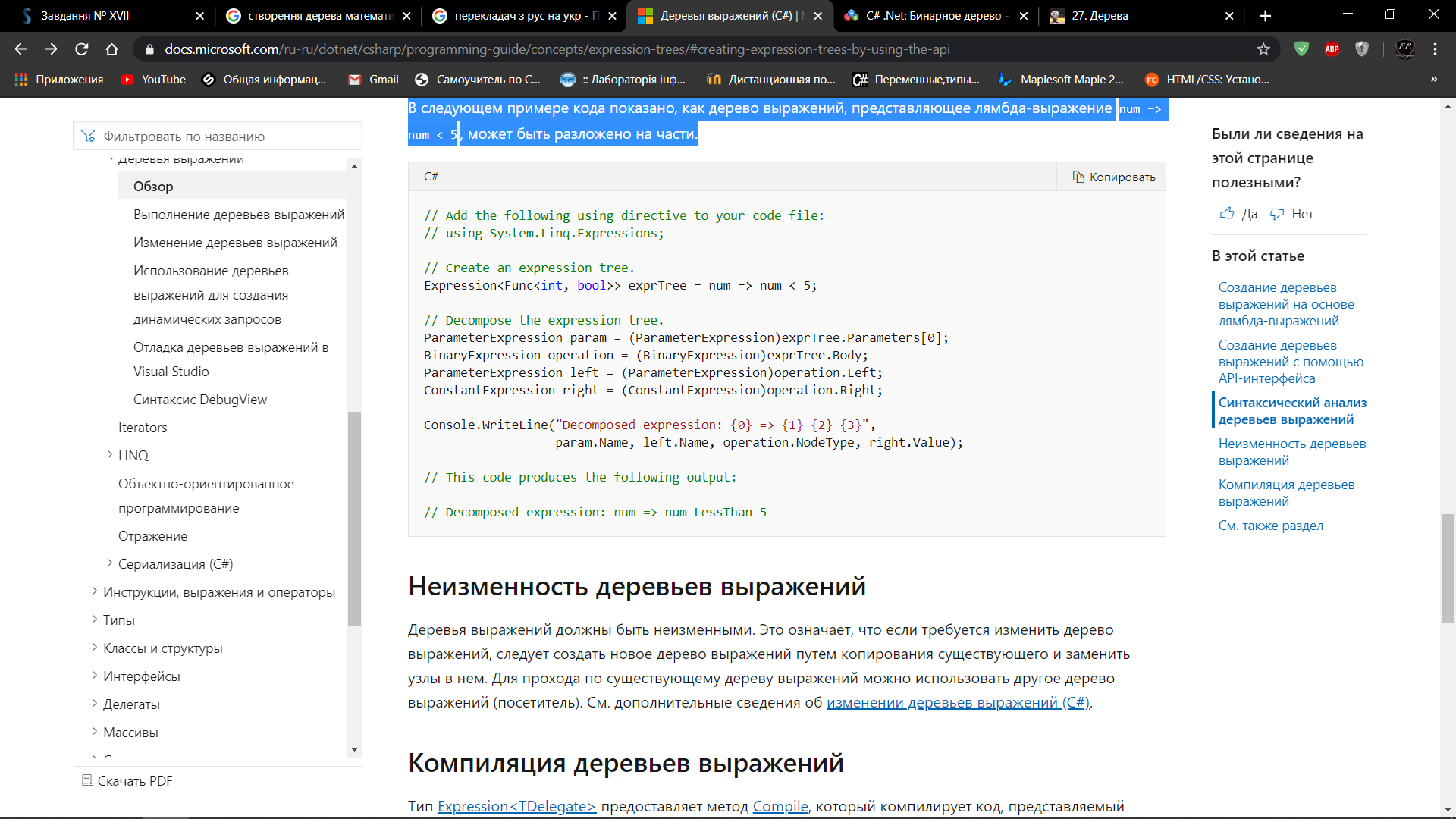
У наступному прикладі коду демонструється спосіб створення дерева виразів, що представляє лямбда-вираз num => num <5, за допомогою API.

На платформі .NET Framework 4 або пізнішої версії API дерев виразів також підтримує присвоювання і вирази потоку управління, такі як цикли, умовні блоки і блоки try-catch. За допомогою API-інтерфейсу можна створювати дерева виразів, більш складні, ніж дерева, створювані компілятором C # з лямбда-виразів. У наступному прикладі показаний спосіб створення дерева виразів, яке обчислює факторіал числа.



1.4. Синтаксичний аналіз дерева

У наступному прикладі коду показано, як дерево виразів, що представляє лямбда-вираз num => num <5, може бути розкладено на частини.

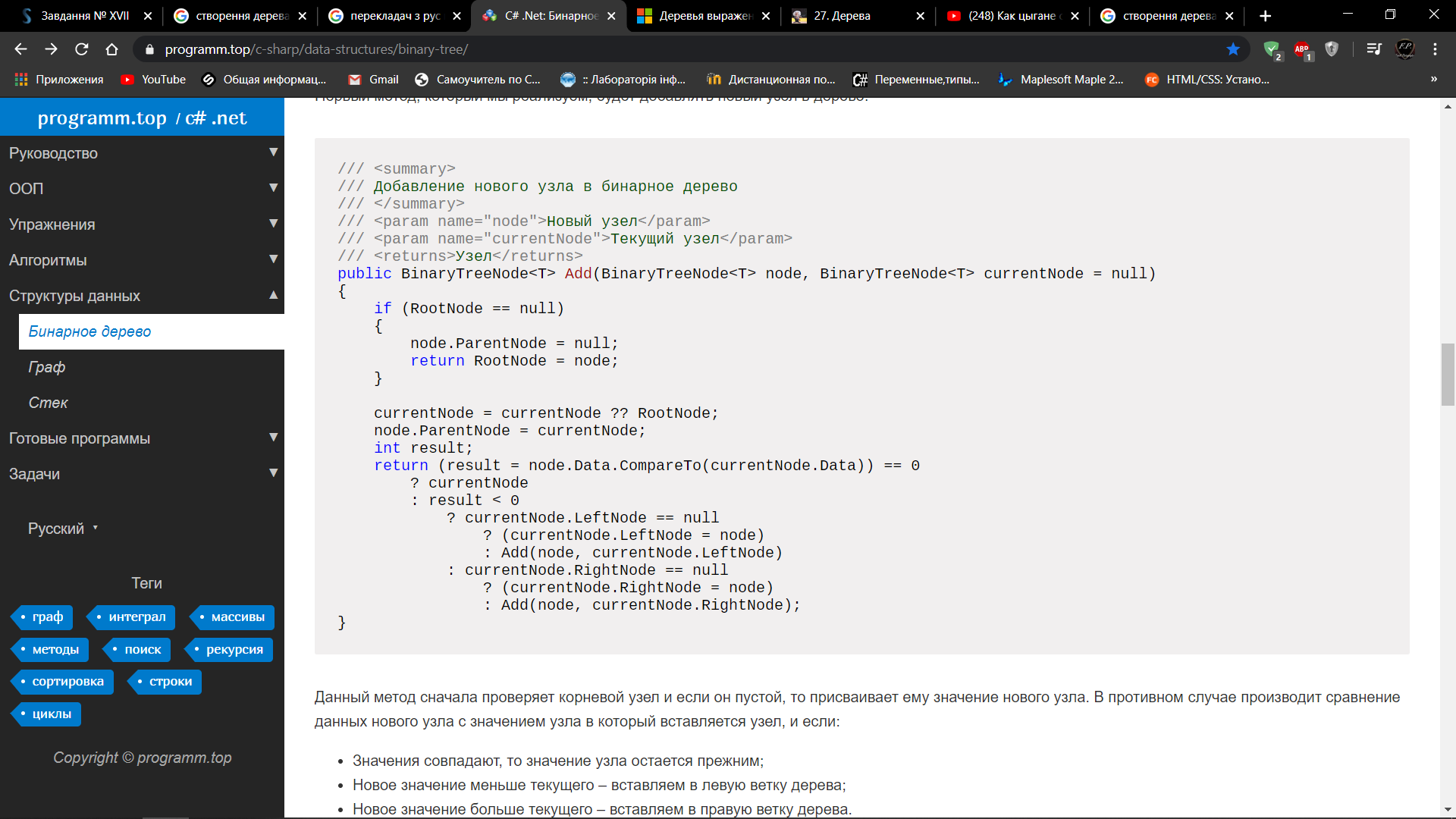


Розділ 2. Проектування програми створення дерева математичного виразу

Перед початком реалізації програмного продукту необхідно спроектувати його. Це є важливою частиною курсової роботи. Тому у цьому розділі розглянемо проект програми.

2.1 Додавання даних

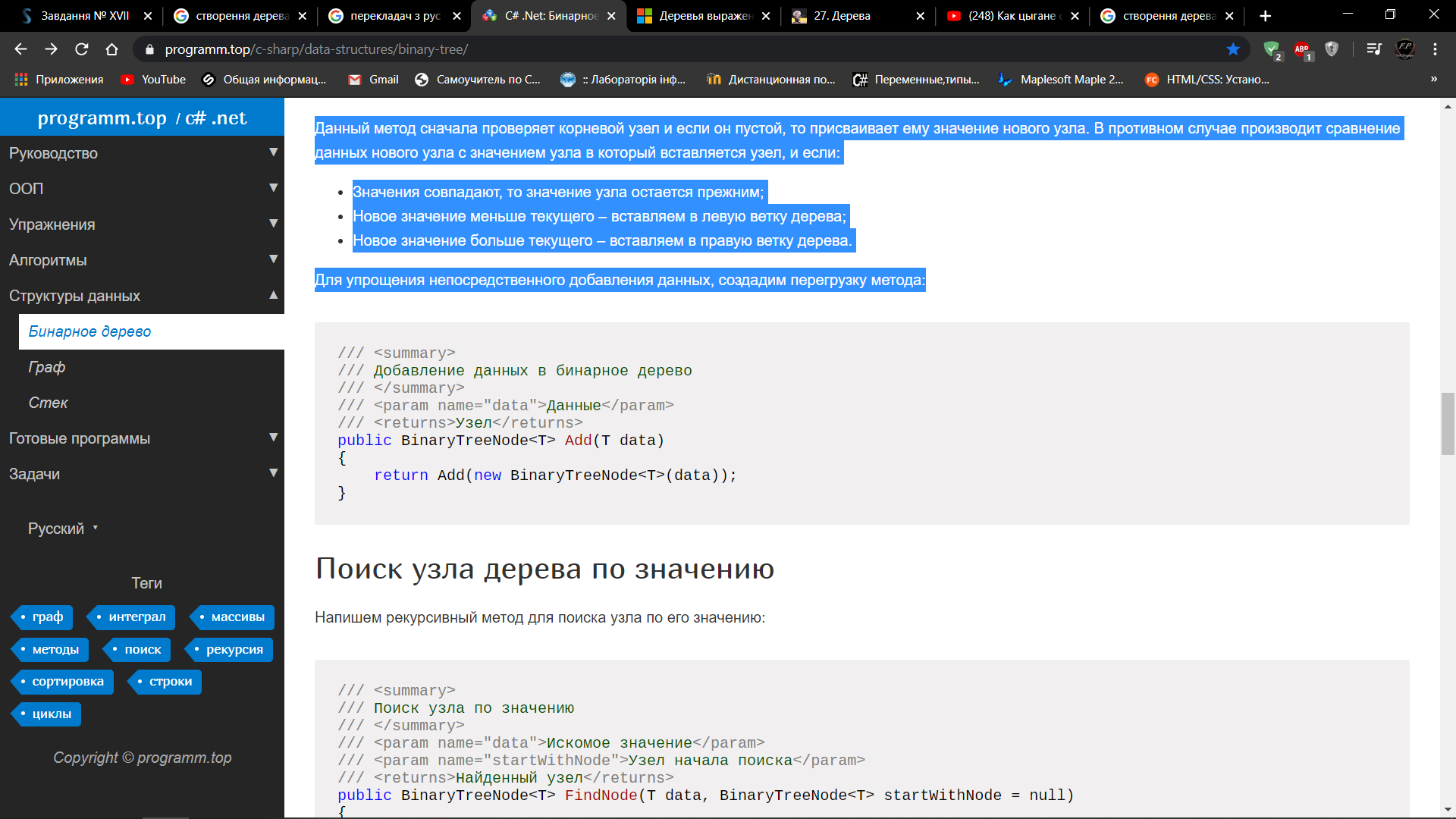
Перший метод, який ми реалізуємо, буде додавати новий вузол в дерево:



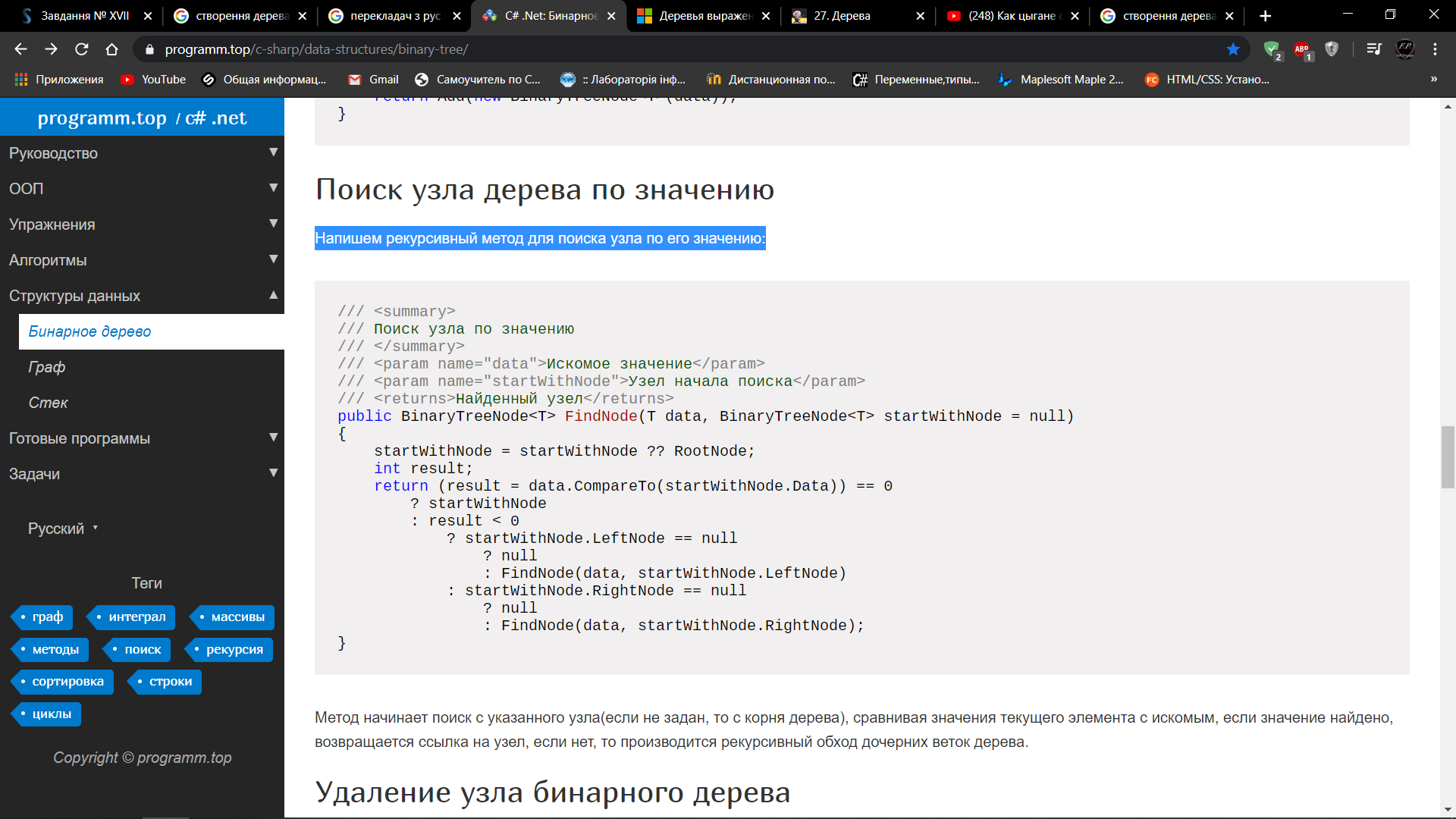
Даний метод спочатку перевіряє кореневої вузол і якщо він порожній, то привласнює йому значення нового вузла. В іншому випадку виробляє порівняння даних нового вузла з значенням вузла в який вставляється вузол, і якщо:

* Значення збігаються, то значення вузла залишається колишнім;
* Нове значення менше поточного - вставляємо в ліву гілку дерева;
* Нове значення більше поточного - вставляємо в праву гілку дерева.

Для спрощення безпосереднього додавання даних, створимо перевантаження методу:



2.2. Пошук вузла дерева за значенням

Напишемо рекурсивний метод для пошуку вузла по його значенню:

Метод починає пошук з зазначеного вузла (якщо не заданий, то з кореня дерева), порівнюючи значення поточного елемента з шуканим, якщо значення знайдено, повертається посилання на вузол, якщо немає, то проводиться рекурсивний обхід дочірніх гілок дерева.

2.3. Видалення вузла бінарного дерева

/// <summary>

/// Удаление узла бинарного дерева

/// </summary>

/// <param name="node">Узел для удаления</param>

public void Remove(BinaryTreeNode<T> node)

{

if (node == null)

{

return;

}

var currentNodeSide = node.NodeSide;

//если у узла нет подузлов, можно его удалить

if (node.LeftNode == null && node.RightNode == null)

{

if (currentNodeSide == Side.Left)

{

node.ParentNode.LeftNode = null;

}

else

{

node.ParentNode.RightNode = null;

}

}

//если нет левого, то правый ставим на место удаляемого

else if (node.LeftNode == null)

{

if (currentNodeSide == Side.Left)

{

node.ParentNode.LeftNode = node.RightNode;

}

else

{

node.ParentNode.RightNode = node.RightNode;

}

node.RightNode.ParentNode = node.ParentNode;

}

//если нет правого, то левый ставим на место удаляемого

else if (node.RightNode == null)

{

if (currentNodeSide == Side.Left)

{

node.ParentNode.LeftNode = node.LeftNode;

}

else

{

node.ParentNode.RightNode = node.LeftNode;

}

node.LeftNode.ParentNode = node.ParentNode;

}

//если оба дочерних присутствуют,

//то правый становится на место удаляемого,

//а левый вставляется в правый

else

{

switch (currentNodeSide)

{

case Side.Left:

node.ParentNode.LeftNode = node.RightNode;

node.RightNode.ParentNode = node.ParentNode;

Add(node.LeftNode, node.RightNode);

break;

case Side.Right:

node.ParentNode.RightNode = node.RightNode;

node.RightNode.ParentNode = node.ParentNode;

Add(node.LeftNode, node.RightNode);

break;

default:

var bufLeft = node.LeftNode;

var bufRightLeft = node.RightNode.LeftNode;

var bufRightRight = node.RightNode.RightNode;

node.Data = node.RightNode.Data;

node.RightNode = bufRightRight;

node.LeftNode = bufRightLeft;

Add(bufLeft, node);

break;

}

}

}

Оскільки в метод для видалення, необхідно передавати екземпляр класу, то для зручності зробимо перевантаження для пошуку і видалення:

/// <summary>

/// Удаление узла дерева

/// </summary>

/// <param name="data">Данные для удаления</param>

public void Remove(T data)

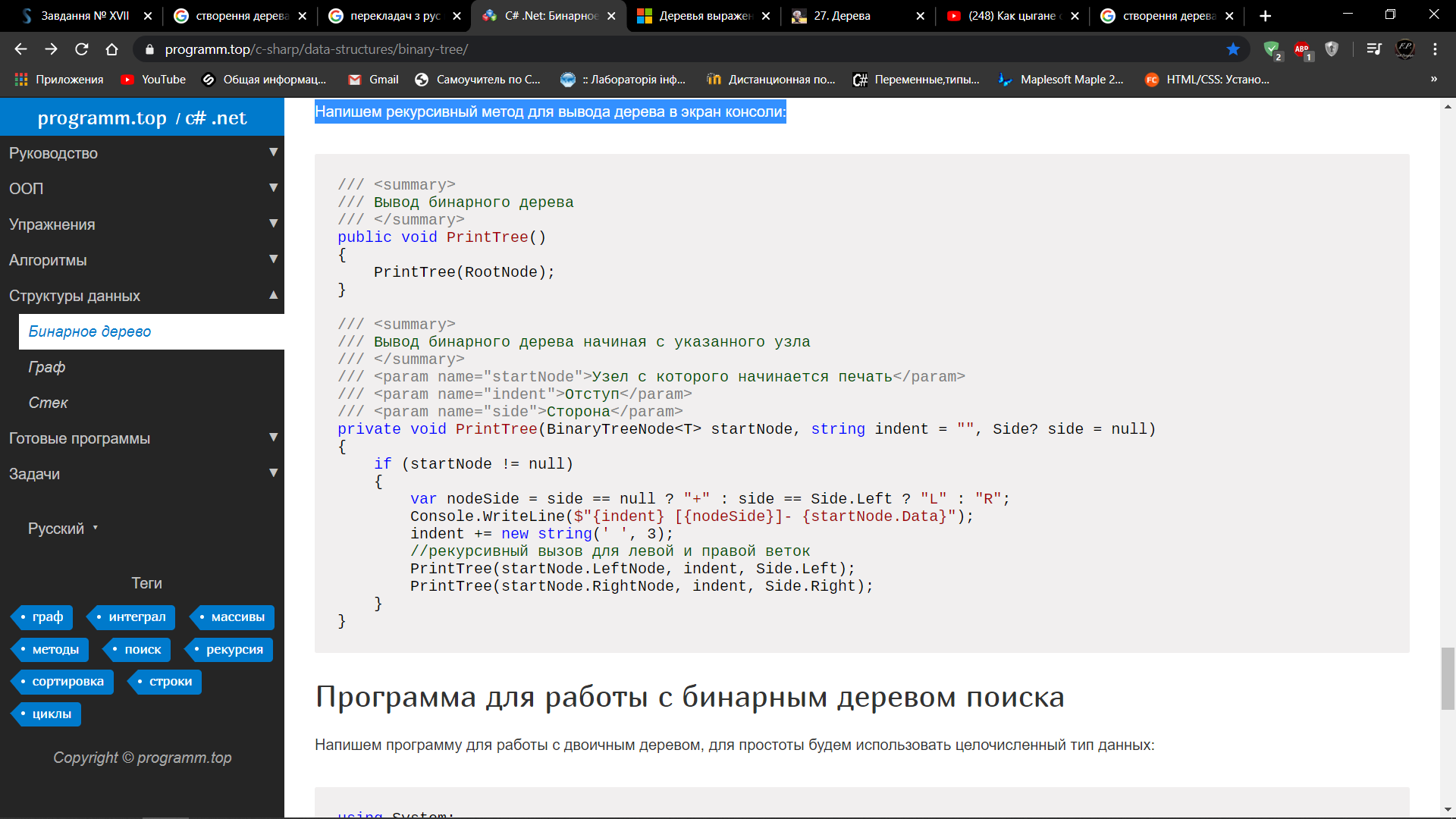
{

var foundNode = FindNode(data);

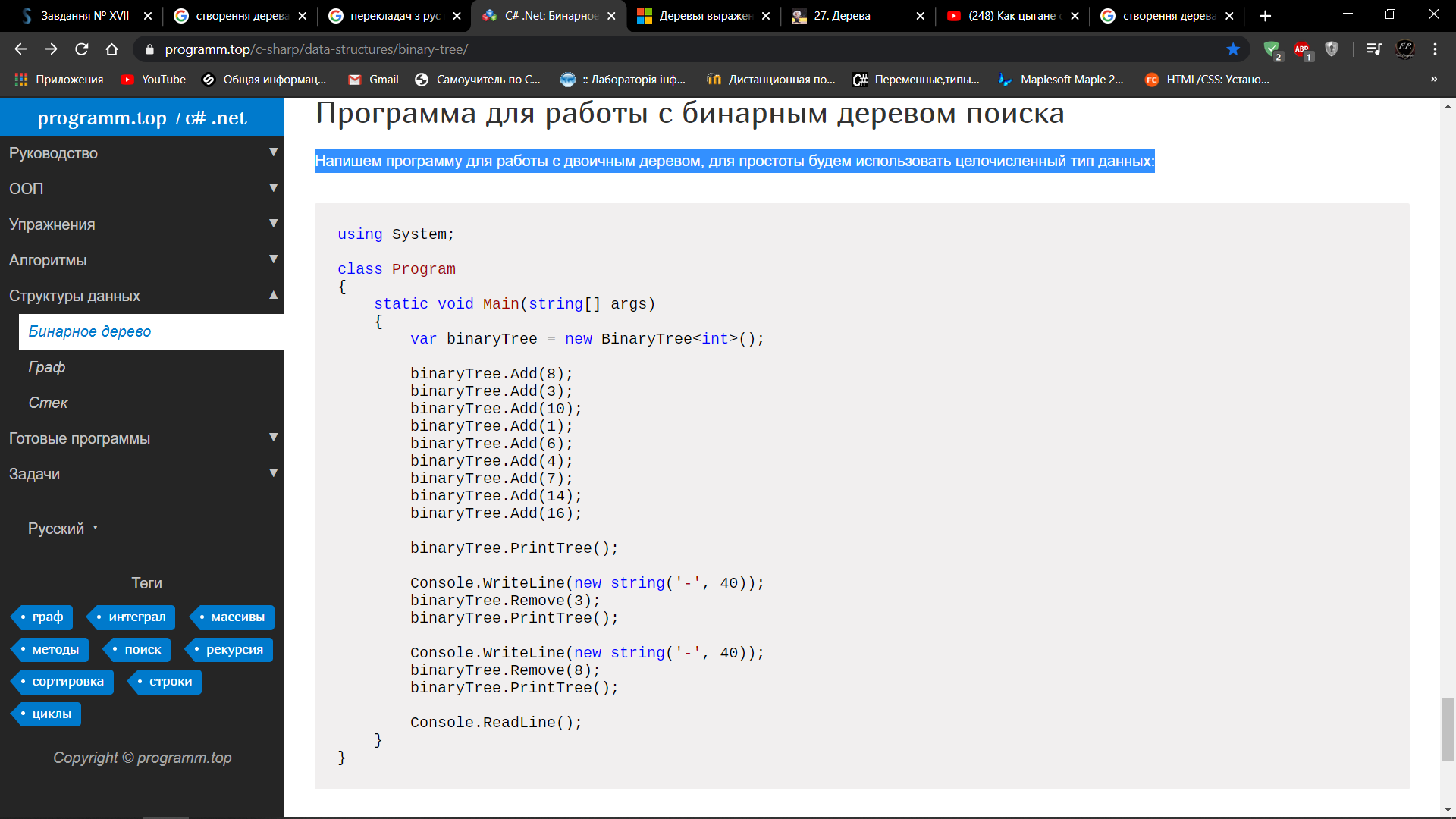
Remove(foundNode);

}

2.4. Виведення дерева на екран

Напишемо рекурсивний метод для виведення дерева в екран консолі:

2.5. Програма для роботи з створенням дерева

Для роботи з деревом, будемо використовувати цілочисельний тип даних:

Розділ 3. Тестування програми

Цей розділ є фінальним та присвячений реалізації програми та її тестуванню. Ми коротко опишемо реалізацію основних функцій та засоби, завдяки яким це можливе. Наша програма написана мовою C# на Windows Form, тож розглянемо зміст цього розділу.

* 1. Тестування програми

Тестування виконувалося математичним шляхом без використання сторонніх програм. Тестування програмного забезпечення - це процес, що використовується для виміру якості розроблюваного програмного забезпечення. Зазвичай, поняття якості обмежується такими поняттями, як коректність, повнота, безпечність, але може містити більше технічних вимог, які описані в стандарті ISO 9126.

Основна мета стандарту ISO / IEC 9126 полягає в тому, щоб усунути деякі з добре відомих людських упереджень, які можуть негативно вплинути на доставку і сприйняття проекту розробки програмного забезпечення. Ці упередження включають зміна пріоритетів після початку проекту або відсутність чітких визначень «успіху».

Тестування - це процес технічного дослідження, який виконується на вимогу замовників, і призначений для вияву інформації про якість продукту відносно контексту, в якому він має використовуватись. До цього процесу входить виконання програми з метою знайдення помилок .

Тестування - це такий ітеративний процес. Як тільки він виявить і виправить помилку, він обіцяє показати тест, який може гарантовано працювати правильно. Крім того, тестові візити можуть бути достатніми, щоб включити цю проблему. Професор Еджер Дейкстра з 1972 року: «Ви можете опублікувати екзаменаційні підходи, щоб написати помилку, але їх не можна замінити і довести, що вони недоступні!»

Висновок

Підіб’ємо підсумок цієї роботи. Сфера використання програми є специфічною, адже має спеціалізоване направлення. Програмний продукт можна використовувати з навчальною метою для вивчення математичних дисциплін, наприклад «Дискретна математика». Також ці основи будуть дуже корисними для опанування знань у галузі цифрової комбінаційної схеми. Програмний продукт може використовуватись і для інших потреб за необхідністю.

Отже, дана програма має потенціал у розвитку та поширенню у навчальних цілях.

Список використаних джерел

1. <https://en.wikipedia.org/wiki/ISO/IEC_9126>
2. <http://lib.mdpu.org.ua/e-book/vstup/L11.htm>
3. <https://evgavrilenko.ucoz.ru/DS/LEKCIYA_4.pdf>
4. <https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/expression-trees/#creating-expression-trees-by-using-the-api>
5. <https://programm.top/c-sharp/data-structures/binary-tree/>
6. <https://rmdiscala.developpez.com/cours/LesChapitres.html/Cours4/Chap4.7.htm>
7. <https://studfile.net/preview/5992429/page:8/>