**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра ІПІ**

**Звіт**

з лабораторної роботи № 5 з дисципліни

«Алгоритми та структури даних 2. Структури даних»

„**Спискові структури даних**”

**Перевірив**

(прізвище, ім'я, по батькові)

*Соколовський Владислав Володимирович*

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-15 Плугатирьов Дмитро Валерійович ійович*

Київ 2022

*Мета роботи* – вивчити основні підходи формалізації евристичних алгоритмів і вирішення типових задач з їх допомогою.

**Завдання**

Розробити алгоритм розв’язання задачі відповідно до варіанту. Виконати програмну реалізацію задачі. Не використовувати вбудовані спискові структури даних (контейнери). Зробити висновок по лабораторній роботі.

**Варіант 25**

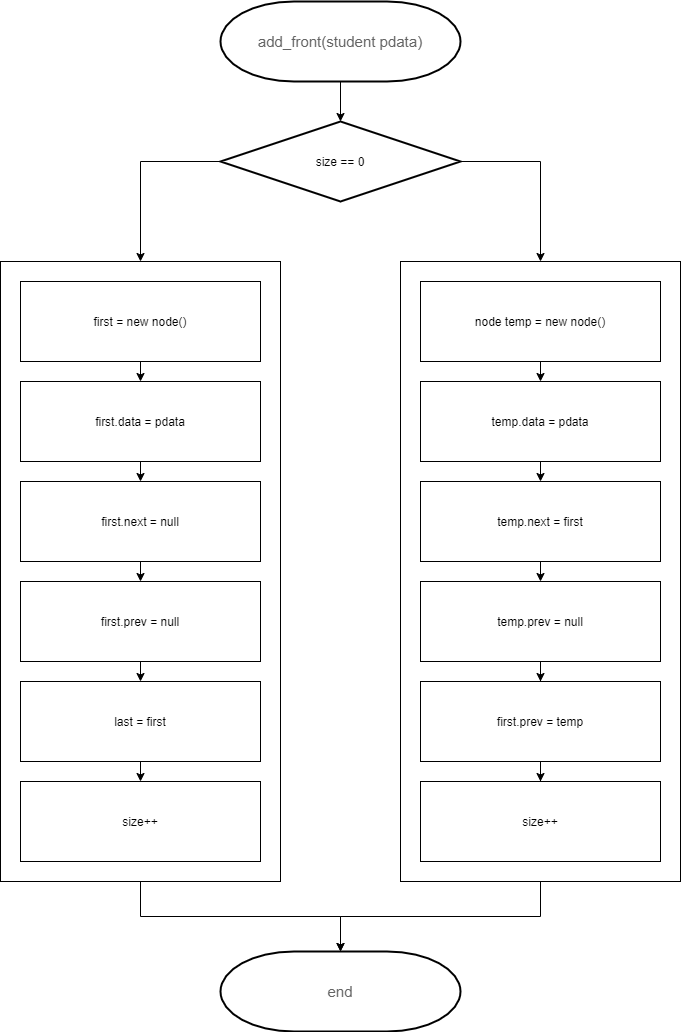
25. Створити двозв'язний список, елементами якого є слова тексту.

Вивести слова, які стоять на парних місцях при проході по списку в одному

напрямку, та слова, які стоять на непарних позиціях при проході по списку в

зворотньому напрямку.

**Блок-схема вставки елемента в початок списку**

****

**Вихідний код**

namespace Labwork\_5

{

    public class DoubleTierNode<T>

    {

        public T Data { get; set; }

        public DoubleTierNode<T>? Previous { get; set; }

        public DoubleTierNode<T>? Next { get; set; }

        public DoubleTierNode(T data)

        {

            Data = data;

        }

    }

}

using System.Collections;

namespace Labwork\_5

{

    public class DoublyTierList<T> : IEnumerable<T>

    {

        private DoubleTierNode<T>? \_head;

        private DoubleTierNode<T>? \_tail;

        private int \_count;

        public int Count { get => \_count; }

        public bool IsEmpty { get => \_count == 0; }

        public void Add(T data)

        {

            ip15\_pluhatyrov\_05.ValidateData(data);

            DoubleTierNode<T>? node = new DoubleTierNode<T>(data);

            if (\_head == null)

            {

                \_head = node;

            }

            else

            {

                \_tail.Next = node;

                node.Previous = \_tail;

            }

            \_tail = node;

            \_count++;

        }

        public void AddFirst(T data)

        {

            DoubleTierNode<T>? node = new DoubleTierNode<T>(data);

            DoubleTierNode<T>? tempNode = \_head;

            node.Next = tempNode;

            \_head = node;

            if (\_count == 0)

            {

                \_tail = \_head;

            }

            else

            {

                tempNode.Previous = node;

            }

            \_count++;

        }

        public bool Remove(T data)

        {

            DoubleTierNode<T>? current = \_head;

            while (current != null && !current.Data.Equals(data))

            {

                current = current.Next;

            }

            if(current != null)

            {

                if(current.Next != null)

                {

                    current.Next.Previous = current.Previous;

                }

                else

                {

                    \_tail = current.Previous;

                }

                if(current.Previous!=null)

                {

                    current.Previous.Next = current.Next;

                }

                else

                {

                    \_head = current.Next;

                }

                \_count--;

                return true;

            }

            return false;

        }

        public void Clear()

        {

            \_head = null;

            \_tail = null;

            \_count = 0;

        }

        public bool Contains(T data)

        {

            DoubleTierNode<T>? currentNode = \_head;

            while (currentNode != null)

            {

                if (currentNode.Data.Equals(data))

                {

                    return true;

                }

                currentNode = currentNode.Next;

            }

            return false;

        }

        IEnumerator IEnumerable.GetEnumerator()

        {

            return ((IEnumerable)this).GetEnumerator();

        }

        IEnumerator<T> IEnumerable<T>.GetEnumerator()

        {

            DoubleTierNode<T>? currentNode = \_head;

            while (currentNode != null)

            {

                yield return currentNode.Data;

                currentNode = currentNode.Next;

            }

        }

        public IEnumerable<T> BackEnumerator()

        {

            DoubleTierNode<T>? currentNode = \_tail;

            while (currentNode != null)

            {

                yield return currentNode.Data;

                currentNode = currentNode.Previous;

            }

        }

    }

}

namespace Labwork\_5

{

    class ip15\_pluhatyrov\_05

    {

        public static void Main(string[] args)

        {

            System.Console.WriteLine("You are entering values for the double-tier list:");

            DoublyTierList<string> doubleTierList = CaptureDoubleTierList();

            PrintHorizontalRule();

            System.Console.WriteLine("The content of double-tier list:");

            foreach (var data in doubleTierList)

            {

                System.Console.WriteLine(data);

            }

            PrintHorizontalRule();

            int i = 1;

            System.Console.WriteLine("Words on even places of the double-tier list in forward:");

            foreach (string data in doubleTierList)

            {

                if ((i & 1) == 0)

                {

                    Console.WriteLine(data);

                }

                i++;

            }

            PrintHorizontalRule();

            i = doubleTierList.Count;

            System.Console.WriteLine("Words on odd places of the double-tier list in reverse:");

            foreach (string? data in doubleTierList.BackEnumerator())

            {

                if ((i & 1) == 1)

                {

                    Console.WriteLine(data);

                }

                i--;

            }

            PrintHorizontalRule();

        }

        static DoublyTierList<string> CaptureDoubleTierList()

        {

            DoublyTierList<string> result = new DoublyTierList<string>();

            bool exceptionIsThrown;

            do

            {

                System.Console.Write("Enter the word: ");

                exceptionIsThrown = false;

                try

                {

                    result.Add(Console.ReadLine());

                }

                catch (ArgumentException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    exceptionIsThrown = true;

                }

                if (!exceptionIsThrown)

                {

                    System.Console.WriteLine("Enter <Backspace> to end typing or any key to continue");

                }

            } while (exceptionIsThrown || Console.ReadKey().Key != ConsoleKey.Backspace);

            return result;

        }

        public static void ValidateData(object? data)

        {

            if (data is string stringData && stringData == string.Empty)

            {

                throw new ArgumentException("The data isn't provided");

            }

        }

        static void PrintHorizontalRule()

        {

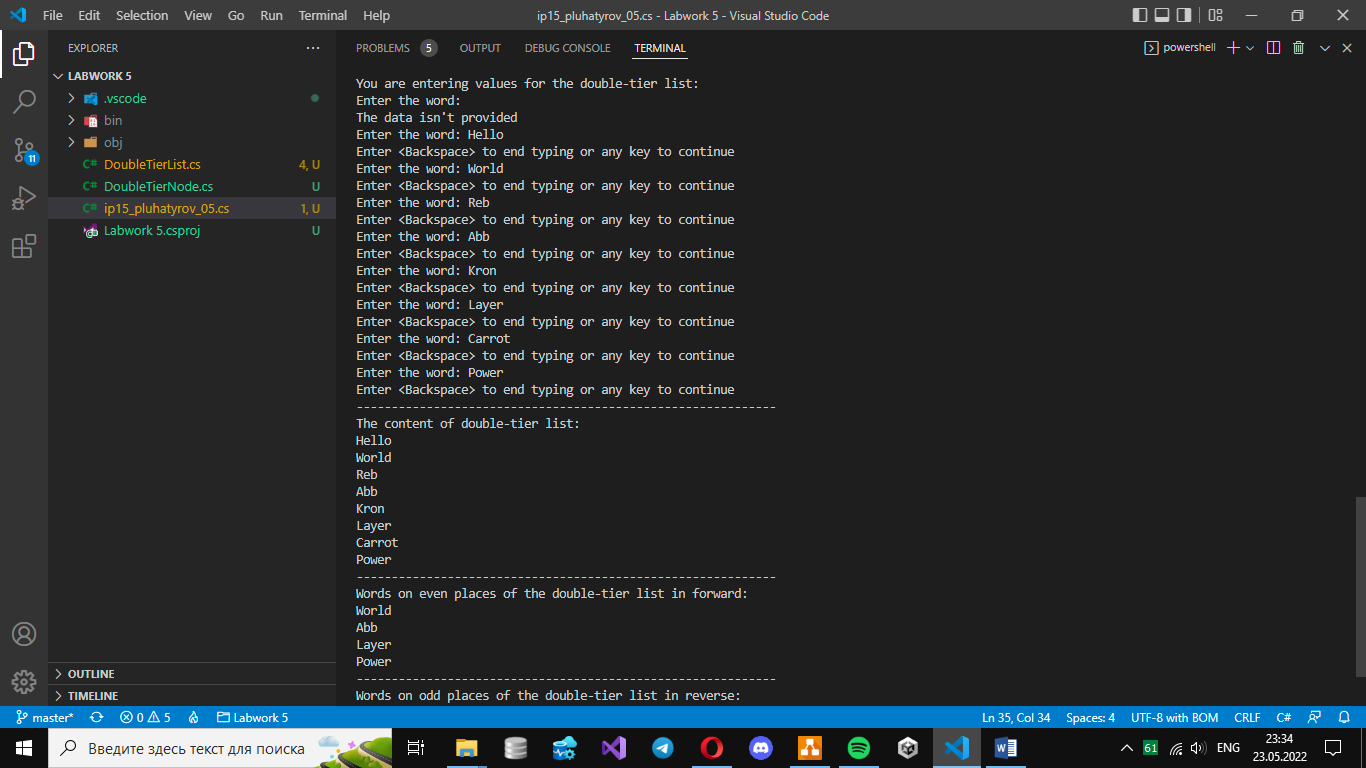
            System.Console.WriteLine(new string('-', 60));

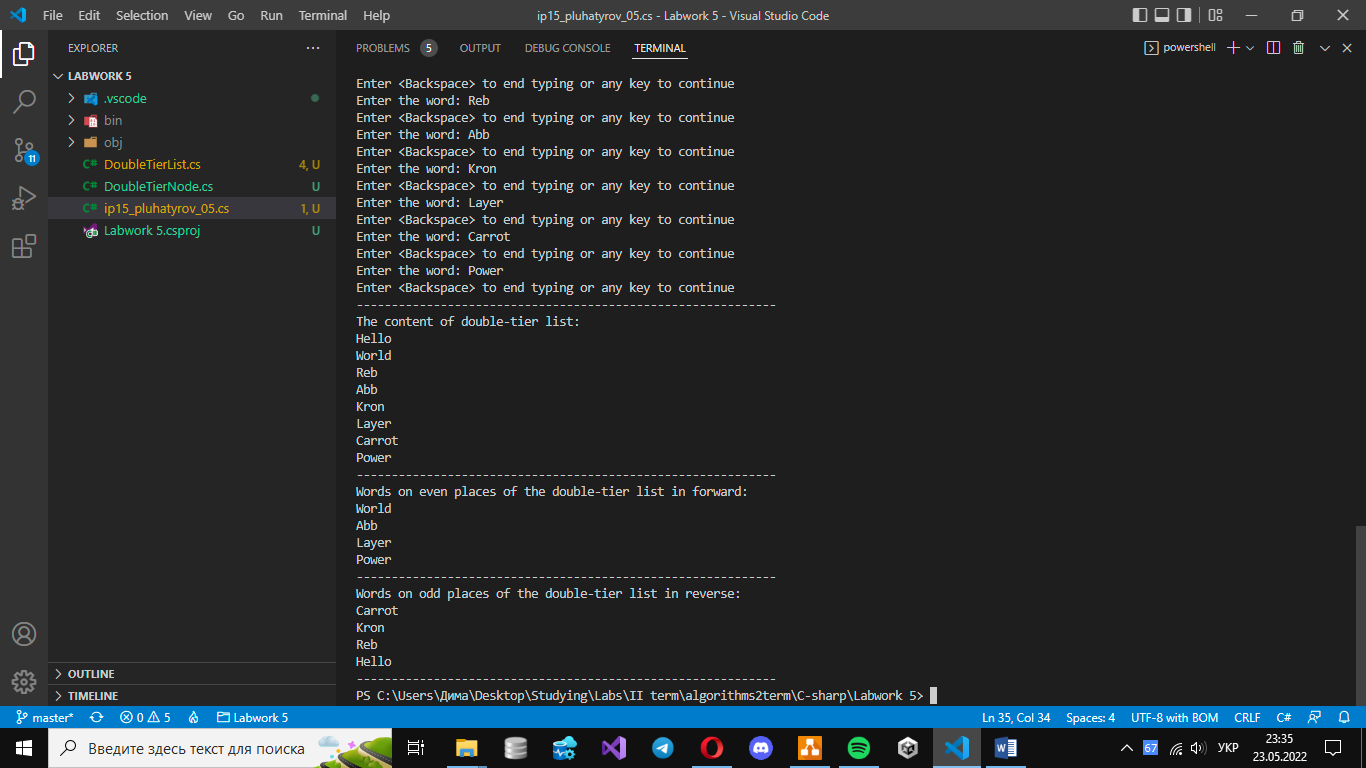
        }

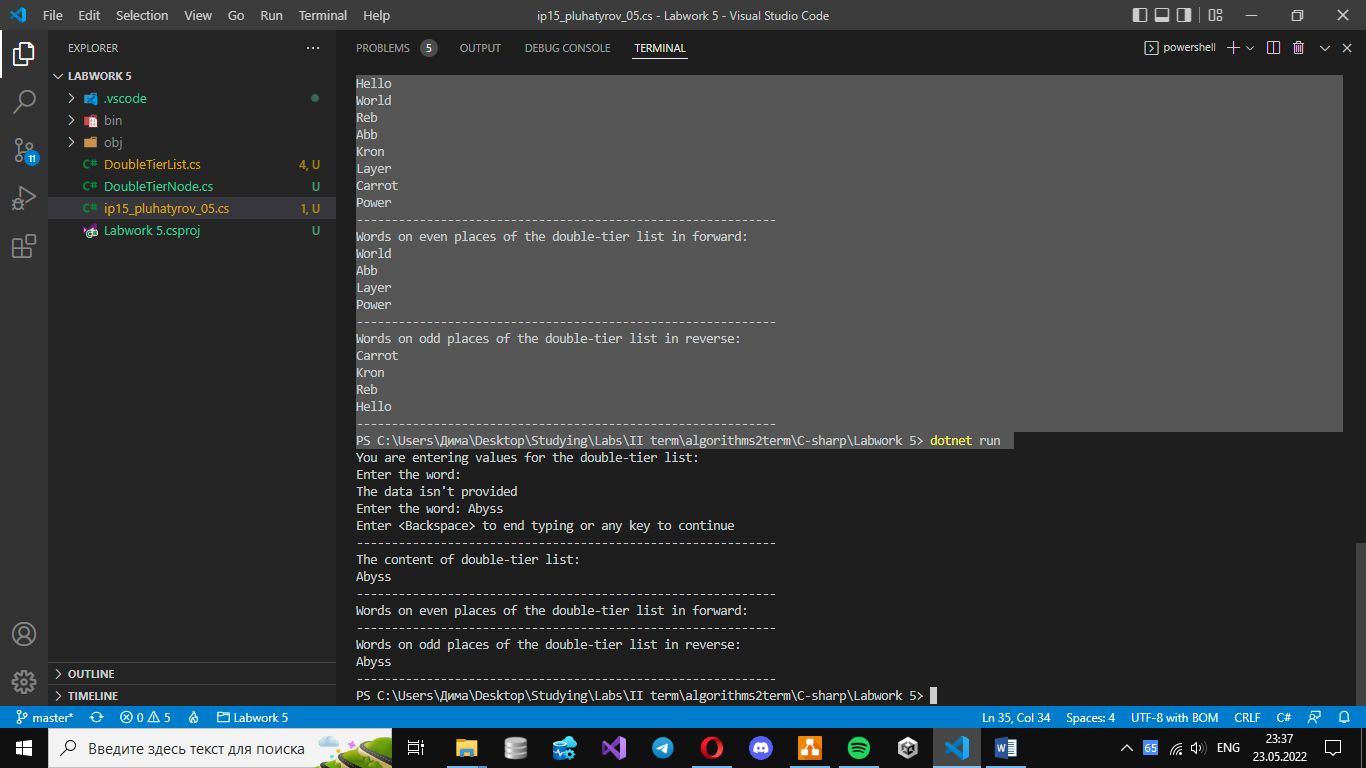
    }

}

**Приклади роботи програми**







**Висновок**

На цій лабораторній роботі я створив двозв’язний список. Він складається з вузлів, які мають посилання на своїх сусідів за їх наявності. На відміну від однозв’язного списку, двозв’язний спрощує додавання нових вузлів в певну позицію, надає можливість «проходитися» по списку з обох сторін та полегшує видалення елемента з певних індексом із послідовності. Хоча, під створення нової змінної-покажчика на попередній елемент вимагає додаткової пам’яті.