**ПРАКТИЧЕСКОЕ ЗАДАНИЕ: РЕАЛИЗАЦИЯ ДИНАМИЧЕСКИХ СТРУКТУР СРЕДСТВАМИ ООП**

**Постановка задачи**

Реализовать программный код, демонстрирующий работу со связными списками: выполнить *Задания* *1 ÷3.*

Основные действия, производимые над элементами (узлами) связных списков:

* инициализация списка;
* добавление / удаление узла в список;
* удаление корня списка;
* вывод элементов списка;
* проверка на наличие элемента в списке;
* проверка на пустоту/наполненность списка;
* взаимообмен двух узлов списка.

Для реализации использовать соответствующие классы из коллекции ЯП C#: Stack<T>, Queue<T>, LinkedList<T>.

*Задание 1.* Создать стек, содержащий ***Вставка А*** слов. Вывести элементы стека. Реализовать основные операции над стеком, и процедуру/функцию, которая ***Вставка Б***, содержащее ***Вставка В***.

Для стека создается однонаправленный список, в котором элементы располагаются в обратном порядке.

|  |  |  |
| --- | --- | --- |
| ***Вставка А*** | ***Вставка Б*** | ***Вставка*** ***В*** |
| 1. 5 2. 7 3. 9 | 1. Вставляет слово 2. Удаляет слова из стека, пока не встретит слово, | * 1. Четное количество гласных   2. Подряд идущие 2 гласные   3. Начинается с гласной   4. Нечетное количество букв   5. Одинаковые первую и последнюю буквы. |

*Задание 2.* Создать очередь, содержащий ***Вставка А*** целых чисел из интервала ***Вставка Б.*** Вывести элементы очереди.

Реализовать основные операции над очередью и процедуру/функцию, которая добавляет элемент ***Вставка В****.*

Для очереди создается однонаправленный список, при этом запоминаются ссылки на первое и последнее узлы списка.

| ***Вставка А*** | ***Вставка Б*** | ***Вставка В*** |
| --- | --- | --- |
| 1. 10 2. 12 3. 13 4. 14 5. 15 6. 16 7. 11 | 1. (-10, 20) 2. (-14, 23) 3. (-5, 19) 4. (-7, 30) 5. (-1, 24) 6. (-15,34) 7. (-4, 25 ) 8. (-6, 27 ) 9. (-8, 35 ) 10. (-4, 23 ) | 1. Равный максимальному элементу очереди 2. Равный минимальному элементу очереди. 3. Элемент, равный следующему за первым отрицательным элементом очереди. 4. Элемент, равный следующему за первым нулем очереди. |

*Задание 3.* Создать двунаправленный связный список, содержащий ***Вставка А*** целых чисел из интервала ***Вставка*** *Б.* Вывести список.

Реализовать процедуру/функцию, которая ***Вставка В***элемент ***Вставка Г.*** Вывести новый список.

|  |  |  |  |
| --- | --- | --- | --- |
| ***Вставка А*** | ***Вставка Б*** | ***Вставка В*** | ***Вставка Г*** |
| 1. 10 2. 12 3. 13 4. 14 5. 15 6. 16 7. 11 | 1. (-10, 20) 2. (-14, 23) 3. (-5, 19) 4. (-7, 30) 5. (-1, 24) 6. (-15,34) 7. (-4, 25 ) 8. (-6, 27 ) 9. (-8, 35 ) 10. (-4, 23 ) | 1. Удаляет 2. Вставляет | 1. в середину списка . 2. после максимального элемента 3. после минимального элемента . 4. после первого простого числа. 5. после последнего четного элемента 6. после первого нуля. |

**Краткие теоретические сведения**

## **Класс Stack<T>**

Класс Stack<T> представляет коллекцию, которая реализует правило LIFO ("Last In-First Out"). По этому правилу каждый новый элемент добавляется поверх предыдущего на вершину стека, а извлекается тот элемент, который был добавлен последним и который находится на вершине стека.

Коллекция класса Stack<T> имеет динамический характер, расширяясь по мере необходимости, чтобы вместить все элементы, которые должны в ней храниться, и принадлежит пространству имен System.Collections.Generic. В классе Stack<T> реализуется ряд стандартных методов для работы со стеком; основные методы описаны в таблице 1.

Таблица 1 - Методы класса Stack<T>

|  |  |  |  |
| --- | --- | --- | --- |
| **Методы** | | **Описание** | |
| Push(T) | | Добавляет элемент в вершину Stack<T> | |
| Pop() | | Удаляет и возвращает элемент из вершины стека. Если стек пуст, генерируется исключение типа InvalidOperationException (выдается при вызове метода, недопустимого для текущего состояния объекта) | |
| Peek() | | Возвращает элемент из вершины стека, при этом не удаляя его | |
| Contains(T) | | Проверяет наличие элемента в Stack<T> и возвращает true в случае его нахождения | |
| Clear() | | Удаляет все объекты из стека | |
| СоруТо(T[], Int 32) | | Копирует Stack<T> в существующий одномерный массив, начиная с указанного индекса массива | |
| Equals(Object) | | Определяет, равен ли заданный объект текущему объекту | |
| TrimExcess() | | Устанавливает емкость равной фактическому количеству элементов в стеке, если это количество составляет менее 90 процентов текущей емкости (или изменяет емкость стека, избавляясь от пустых элементов) | |

Кроме перечисленных выше методов в классе Stack<T> описано свойство Count, которое возвращает количество элементов в стеке. Пример реализации методов класса Stack<T>, приведён в Приложении А.

## **Класс Queue<T>**

Queue<T> — это коллекция, в которой элементы обрабатываются по правилу FIFO ("First In-First Out "). Согласно правилу, новый элемент добавляется в конец очереди, а при извлечении элемент, поставленный в очередь первым, и читается первым.

Этот класс также реализован в пространстве имен System.Collections.Generic. В таблице 2 описаны основные методы этого класса.

Таблица 2 - Методы класса Queue<T>

|  |  |
| --- | --- |
| **Методы** | **Описание** |
| Enqueue(T) | Добавляет элемент в конец Queue<T> |
| Dequeue() | Читает и удаляет элемент из головы очереди. Если на момент вызова метода Dequeue() элементов в очереди больше нет, генерируется исключение InvalidOperationException |
| Peek() | Читает элемент из головы очереди, но не удаляет его |
| MemberwiseClone() | Создает неполную копию текущего объекта |
| Clear() | Удаляет все объекты из очереди |
| Contains(T) | Определяет, входит ли элемент в коллекцию Queue<T> |
| **Методы** | **Описание** |
| TrimExcess() | Устанавливает емкость равной фактическому количеству элементов в очереди, если это количество составляет менее 90 процентов текущей емкости (проще говоря, изменяет емкость очереди, избавляясь от пустых элементов) |
| Equals(Object) | Определяет, равен ли заданный объект текущему объекту |

Также, как и в классе Stack<T>, в Queue<T> есть свойство Count, возвращающее число элементов в очереди.

Пример программы, реализующей методы класса Queue<T>, приведён в Приложении Б.

## **Класс LinkedList<T>**

Класс LinkedList<T> представляет двухсвязный список, в котором каждый элемент хранит ссылки как на следующий так и на предыдущий элемент. Преимущество двусвязного списка заключается в том, что операция вставки элемента в середину выполняется очень быстро. При этом только ссылки Next (следующий) предыдущего элемента и Previous (предыдущий) следующего элемента должны быть изменены так, чтобы указывать на вставляемый элемент.

К недостаткам двусвязного списка можно отнести то, что все элементы связных списков доступны лишь последовательно (друг за другом). Поэтому на поиск элемента из середины или конца списка требуется время.

Класс LinkedList<T> также, как и классы Queue<T>, Stack<T>, реализуется в пространстве имён System.Collections.Generic. Основные методы этого класса приведены в таблице 3.

В этом классе также прописаны три свойства: Count (возвращает количество узлов, содержащихся в LinkedList<T>), First (возвращает первый узел LinkedList<T>) и Last (возвращает последний узел LinkedList<T>).

Таблица 3 - Методы класса LinkedList<T>

|  |  |
| --- | --- |
| **Методы** | **Описание** |
| AddAfter (LinkedListNode<T>, T) | Добавляет в список узел со значением Т непосредственно после указанного узла LinkedListNode<T>; указываемый узел не должен быть пустым (null); метод возвращает ссылку на узел, содержащий значение. |
| AddBefore (LinkedListNode<T>, T) | Добавляет новый узел, содержащий указанное значение Т перед указанным узлом LinkedList- Node<T>; указываемый узел не должен быть пустым (null); метод возвращает ссылку на узел, содержащий значение |
| AddFirst(Т) | Добавляет новый узел, содержащий указанное значение Т в начале LinkedList<T>. |
| AddLast(Т) | Добавляет новый узел, содержащий указанное значение в конце LinkedList<T> |
| Clear() | Удаляет все узлы из LinkedList<T> |
| Contains(T) | Определяет, является ли значение в LinkedList<T> |
| *Окончание таблицы 3* | |
| **Методы** | **Описание** |
| Find(T) | Находит первый узел, содержащий указанное значение |
| FindLast(T) | Находит последний узел, содержащий указанное значение |
| Remove(T) | Удаляет первое вхождение указанного объекта из LinkedList<T> |
| RemoveFirst() | Удаляет узел в начале LinkedList<T> |
| RemoveLast() | Удаляет узел в конце LinkedList<T> |

Пример программы, реализующей методы класса LinkedList<T>, приведён в Приложении В.

# **Приложение А** Пример реализации операций над стеком методами класса Stack<T>

//Пример работы методов класса Stack<T>

//*Любезно представлен Беляевой О.Н.*

using System;

using System.Collections.Generic;

namespace Стек

{

class Program

{

static void Main()

{

Console.Title = "Пример работы стека";

Stack<int> chislo = new Stack<int>();//объявление объекта класса Stack<int>

Stack<string> stroka = new Stack<string>();>();//объявление объекта класса Stack<string>

int a, b, c;

string s;

do

{

Console.Clear();

Console.WriteLine("Выберите действие: ");

Console.WriteLine("1 - занести элемент в стек;");

Console.WriteLine("2 - извлечь элемент из вершины стека;");

Console.WriteLine("3 - проверка на пустоту/наполненность;");

Console.WriteLine("4 - вывести на экран последний занесённый элемент;");

Console.WriteLine("5 - проверить наличие элемента в стеке;");

Console.WriteLine("6 - удалить стек;");

Console.WriteLine("7 - выход.");

int.TryParse(Console.ReadLine(), out a);

switch (a)

{

case 1: //добавление элемента в стек

Console.Clear();

Console.WriteLine("Что заносится в стек?\n1 - число 2 - строка");

int.TryParse(Console.ReadLine(), out b);

if (b == 1)

{

Console.Write("Введите число:");

int.TryParse(Console.ReadLine(), out c);

chislo.Push(c);

}

else

{

Console.Write("Введите строку:");

s = Console.ReadLine();

stroka.Push(s);

stroka.e

}

Console.WriteLine("\nОперация выполнена!\nДля продолжения нажмите ENTER.\nДля выхода нажмите ESC.");

break;

case 2: //извлечение элемента из стека

Console.Clear();

Console.WriteLine("Что будет извлечено?\n1 - число 2 - строка");

int.TryParse(Console.ReadLine(), out b);

if (b == 1) Console.Write("Число, находящееся на вершине стека: {0}",chislo.Pop());

else Console.Write("Cтрока, находящаяся на вершине стека: {0}", stroka.Pop());

Console.WriteLine("\nОперация выполнена!\nДля продолжения нажмите ENTER.\nДля выхода нажмите ESC.");

break;

,,,,

case 7: //выход из системы

Console.Clear();

Console.WriteLine("Для выхода нажмите ESC.");

break;

default: //ошибка ввода данных

Console.WriteLine("Ошибка ввода!!! Попробуйте ещё раз.");

Console.WriteLine("Для продолжения нажмите ENTER.");

break;

}

}

while (Console.ReadKey(true).Key != ConsoleKey.Escape);

}

}

}

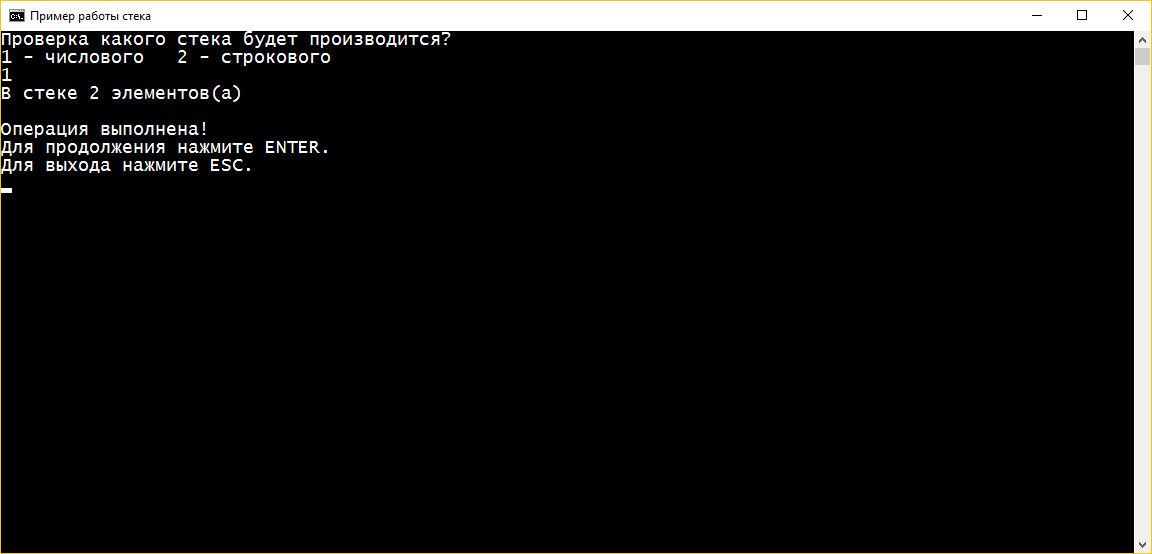


Рисунок 1 – Проверка на пустоту стека

# **Приложение Б Пример реализации работы с очередью методами класса Queue<T>**

//Беляева О.Н.

//Пример работы методов класса Queue<T>

using System;

using System.Collections.Generic;

namespace Очередь

{

class Program

{

static void Main()

{

Console.Title = "Пример работы очереди";

Queue<int> chislo = new Queue<int>();//объявление объекта класса Queue<int>

Queue<string> stroka = new Queue<string>();//объявление объекта класса Queue<string>

int a, b, c;

string s;

do

{

Console.Clear();

Console.WriteLine("Выберите действие: ");

Console.WriteLine("1 - занести элемент в конец очереди;");

Console.WriteLine("2 - извлечь элемент из начала очереди;");

Console.WriteLine("3 - проверка на пустоту/наполненность;");

Console.WriteLine("4 - вывести на экран элемент с головы очереди;");

Console.WriteLine("5 - проверить наличие элемента в очереди;");

Console.WriteLine("6 - удалить очередь;");

Console.WriteLine("7 - выход.");

int.TryParse(Console.ReadLine(), out a);

switch (a)

{

case 1: //добавление элемента очереди

Console.Clear();

Console.WriteLine("Что заносится в очередь?\n1 - число 2 - строка");

int.TryParse(Console.ReadLine(), out b);

if (b == 1)

{

Console.Write("Введите число:");

int.TryParse(Console.ReadLine(), out c);

chislo.Enqueue(c);

}

else

{

Console.Write("Введите строку:");

s = Console.ReadLine();

stroka.Enqueue(s);

}

Console.WriteLine("\nОперация выполнена!\nДля продолжения нажмите ENTER.\nДля выхода нажмите ESC.");

break;

case 2: //извлечение элемента очереди

Console.Clear();

Console.WriteLine("Что будет извлечено?\n1 - число 2 - строка");

int.TryParse(Console.ReadLine(), out b);

if (b == 1) Console.Write("Число, находящееся в начале очереди: {0}", chislo.Dequeue());

else Console.Write("Cтрока, находящаяся в начале очереди: {0}", stroka.Dequeue());

Console.WriteLine("\nОперация выполнена!\nДля продолжения нажмите ENTER.\nДля выхода нажмите ESC.");

break;

...

case 6: //удаление очереди

Console.Clear();

Console.WriteLine("Какая очередь будет удалёна?\n1 - числовая 2 - строковая");

int.TryParse(Console.ReadLine(), out b);

if (b == 1) chislo.Clear();

else stroka.Clear();

Console.WriteLine("\nОперация выполнена!\nДля продолжения нажмите ENTER.\nДля выхода нажмите ESC.");

break;

case 7: //выход из системы

Console.Clear();

Console.WriteLine("Для выхода нажмите ESC.");

break;

default: //ошибка ввода данных

Console.WriteLine("Ошибка ввода!!! Попробуйте ещё раз.");

Console.WriteLine("Для продолжения нажмите ENTER.");

break;

}

}

while (Console.ReadKey(true).Key != ConsoleKey.Escape);

}

}

}

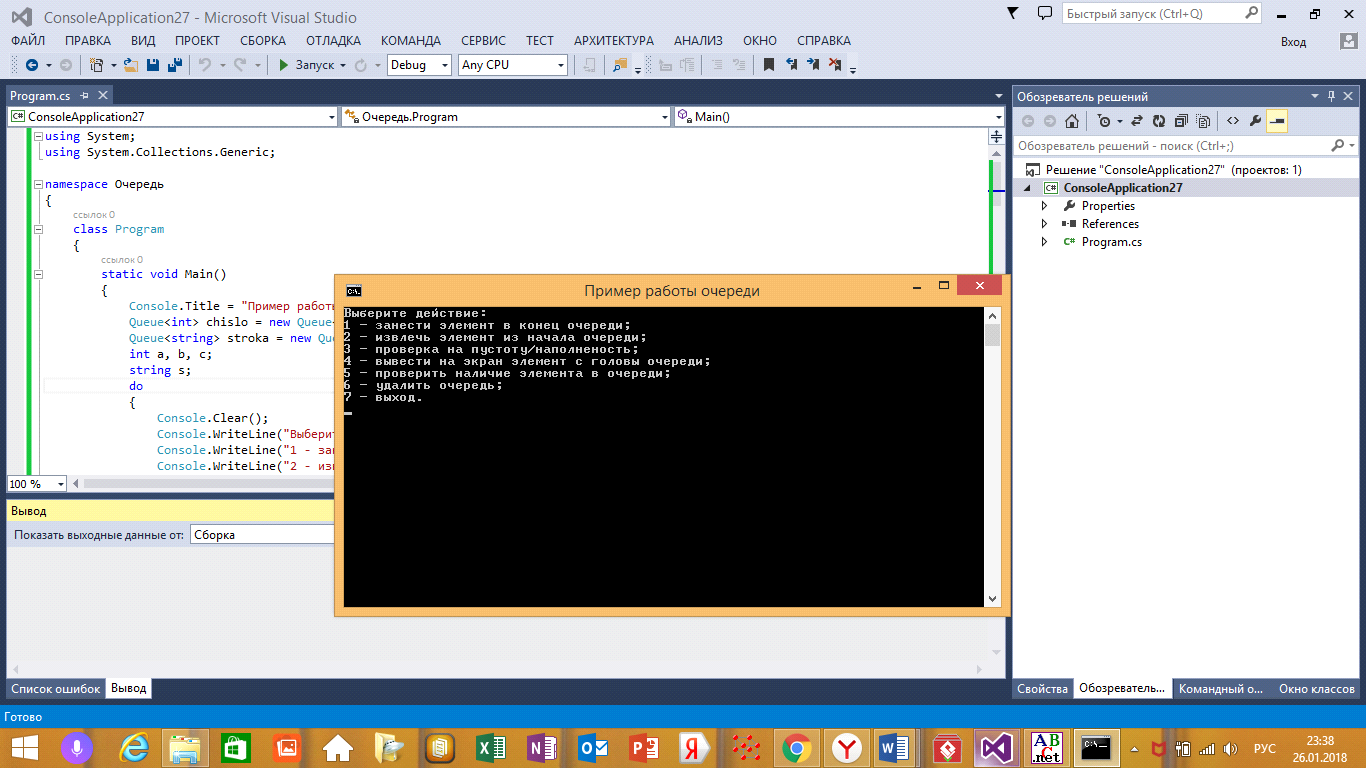


Рисунок 2 – Меню работы с очередью

# **Приложение В Пример реализации работы со списком методами класса LinkedList<T>**

//Пример работы методов класса LinkedList<Т>

using System;

using System.Collections.Generic;

namespace Двунаправленный\_список

{

class Program

{

static void Main()

{

Console.Title = "Пример работы двунаправленного списка";

LinkedList<int> chislo = new LinkedList<int>();//объявление объекта класса LinkedList<int>

LinkedList<string> stroka = new LinkedList<string>();//объявление объекта класса LinkedList<string>

int a, b, c, d;

string s, st;

do

{

Console.Clear();

Console.WriteLine("Выберите действие: ");

Console.WriteLine("1 - добавить узел со значением в начало списка;");

Console.WriteLine("2 - добавить узел со значением в конец списка;");

Console.WriteLine("3 - добавить узел со значением после определённого элемента;");

Console.WriteLine("4 - добавить узел со значением перед определённым элементом;");

Console.WriteLine("5 - вывести количество узлов;");

Console.WriteLine("6 - найти узел с заданным значением;");

Console.WriteLine("7 - удалить узел с указанным значением;");

Console.WriteLine("8 - удалить первый узел;");

Console.WriteLine("9 - удалить последний узел;");

Console.WriteLine("10 - удалить все узлы;");

Console.WriteLine("11 - выход.");

int.TryParse(Console.ReadLine(), out a);

switch (a)

{

case 1://добавление узла в начало списка

…

break;

case 4://добавление узла перед указанным элементом

Console.Clear();

Console.WriteLine("Что добавляется в список?\n1 - число 2 - строка");

int.TryParse(Console.ReadLine(), out b);

if (b == 1)

{

Console.Write("Введите элемент, перед которым будет вставлено число:");

int.TryParse(Console.ReadLine(), out c);

LinkedListNode<int> list = chislo.Find(c);

if (list != null)

{

Console.Write("Введите число:");

int.TryParse(Console.ReadLine(), out d);

chislo.AddAfter(list, d);

}

else Console.WriteLine("Элемент не найден!");

}

else

{

Console.Write("Введите элемент, перед которым будет вставлена строка:");

s = Console.ReadLine();

LinkedListNode<string> list = stroka.Find(s);

if (list != null)

{

Console.Write("Введите строку:");

st = Console.ReadLine();

stroka.AddAfter(list, st);

}

else Console.WriteLine("Элемент ненайден!");

}

Console.WriteLine("\nОперация выполнена!\nДля продолжения нажмите ENTER.\nДля выход анажмите ESC.");

break;

case 5://вывод на экран количества узлов списка

Console.Clear();

Console.WriteLine("Информация о каком списке будет выведена?\n1 - о числовом 2 - о строковом");

int.TryParse(Console.ReadLine(), out b);

if (b == 1)

{

if (chislo.Count == 0) Console.WriteLine("Очередьпуста!");

else

{

Console.WriteLine("В очереди {0} элементов(а)", chislo.Count);

Console.WriteLine("Значение первого узла = {0}", chislo.First);

Console.WriteLine("Значение последнего узла = {0}", chislo.Last);

}

}

else

{

if (stroka.Count == 0) Console.WriteLine("Очередь пуста!");

else

{

Console.WriteLine("В очереди {0} элементов(а)", stroka.Count);

}

}

Console.WriteLine("\nОперация выполнена!\nДля продолжения нажмите ENTER.\nДля выхода нажмите ESC.");

break;

case 6://поиск элмента в списке

Console.Clear();

Console.WriteLine("В каком списке будет проводится поиск?\n1 - в числовом 2 - в строковом");

int.TryParse(Console.ReadLine(), out b);

if (b == 1)

{

Console.Write("Введите искомый элемент: ");

int.TryParse(Console.ReadLine(), out c);

if (chislo.Find(c) == null) Console.WriteLine("Такого числа в списке нет.");

else Console.WriteLine("Такое число присутствует в списке.");

}

else

{

Console.Write("Введите искомый элемент: ");

s = Console.ReadLine();

if (stroka.Find(s) == null) Console.WriteLine("Такойстроки в спискенет.");

else Console.WriteLine("строка найдена в списке.");

}

Console.WriteLine("\nОперация выполнена!\nДля продолжения нажмите ENTER.\nДля выхода нажмите ESC.");

break;

case 7://удаление элемента

Console.Clear();

Console.WriteLine("Из какого списка будет удалён элемент?\n1 - из числового 2 - из строкового");

int.TryParse(Console.ReadLine(), out b);

if (b == 1)

{

Console.Write("Введите элемент, который будет удалён: ");

int.TryParse(Console.ReadLine(), out c);

LinkedListNode<int> list = chislo.Find(c);

if (list != null) chislo.Remove(list);

else Console.WriteLine("Данный элемент отсутствует ");

}

else

{

Console.Write("Введите элемент, который будет удалён: ");

s = Console.ReadLine();

LinkedListNode<string> list = stroka.Find(s);

if (list != null) stroka.Remove(list);

else Console.WriteLine("Данный элемент отсутствует");

}

Console.WriteLine("\nОперация выполнена!\nДля продолжения нажмите ENTER.\nДля выхода нажмите ESC.");

break;

case 11://выход из системы

Console.Clear();

Console.WriteLine("Для выхода нажмите ESC.");

break;

default://ошибка ввода данных

Console.WriteLine("Ошибка ввода!!! Попробуйте ещё раз.");

Console.WriteLine("Для продолжения нажмите ENTER.");

break;

} }

while (Console.ReadKey(true).Key != ConsoleKey.Escape);

} }

}

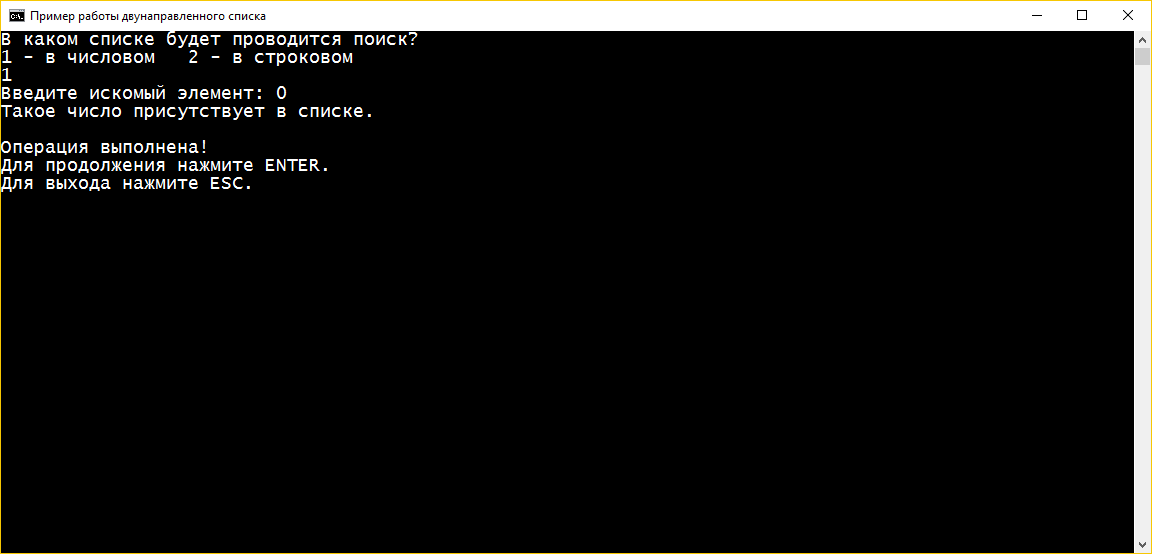


Рисунок 3 – Поиск узла в списке