Федеральное агентство связи

Ордена трудового Красного Знамени

федеральное государственное

бюджетное образовательное учреждение

высшего образования

«Московский технический университет связи и информатики»

Кафедра информатики

Отчет по лабораторной работе №4 по СиАОД

«Реализация стека и дека»

Выполнил: студент группы БВТ1903

Клычёв Д.А

Проверил: Павликов А. Е.

Москва, 2021

# **1. Задание**

# **2. Ход работы**

Язык программирования, используемый для выполнения работы: C#, версия 9.0.

Для выполнения поставленных задач было создано решение в среде разработки MVS2019, включающее проект Lab1, исполняемый код которого представлен в классах Lab4.cs, MyStack.cs, MyDeque.cs, листинг представлен ниже:

public static class Lab4

{

internal static void Run4Lab()

{

Random rnd = new Random();

Stopwatch sw = new Stopwatch();

string filename = "Books.txt";

string dir = Directory.GetCurrentDirectory();

while (!File.Exists(dir + Path.DirectorySeparatorChar + filename))

{

dir = dir.Substring(0, dir.LastIndexOf('\\'));

}

#if stack

#region Stack

MyStack<int> stack = new MyStack<int>();

stack.Push(0);

stack.Push(3);

var peek = stack.Peek();

stack.Pop();

stack.Pop();

#endregion

#endif

#if deque

#region Deque

MyDeque<int> deque = new MyDeque<int>();

deque.Enqueue(5);

deque.Dequeue(4);

var head = deque.PopHead();

var tail = deque.PopTail();

var first = deque.RemoveFirst();

var last = deque.RemoveLast();

#endregion

#endif

#region Tasks

#region Task1

Console.WriteLine("Task 1");

Console.WriteLine();

MyDeque<string> deq1 = new MyDeque<string>();

using (StreamReader reader = new StreamReader(dir + Path.DirectorySeparatorChar + filename))

{

string line;

while ((line = reader.ReadLine()) != null)

{

deq1.Enqueue(line);

}

}

var t1res = Task1(deq1);

foreach (var item in t1res)

{

Console.WriteLine(item);

}

#endregion

#region Task2

Console.WriteLine("Task 2");

Console.WriteLine();

filename = "SourceMessage.txt";

string path = dir + Path.DirectorySeparatorChar + filename;

if (!File.Exists(path))

{

File.WriteAllText(path, "This message will be crypted!");

}

var message = File.ReadAllText(path);

var key = Task2\_keygen(message);

filename = "CryptedMessage.txt";

path = dir + Path.DirectorySeparatorChar + filename;

if (File.Exists(path))

{

File.Delete(path);

}

string t2crypt = Task2\_crypt(message, key, path);

string t2encrypt = Task2\_encrypt(t2crypt, key);

filename = "SourceMessage.txt";

path = dir + Path.DirectorySeparatorChar + filename;

Console.WriteLine("Сообщения для шифрования: " + File.ReadAllText(path));

Console.WriteLine("Зашифрованное сообщение: " + t2crypt);

Console.WriteLine("Расшифрованное сообщение: " + t2encrypt);

#endregion

#region Task3

Console.WriteLine("Task 3");

Console.WriteLine();

filename = "StackData.txt";

Plate plate6 = new Plate(9);

Plate plate5 = new Plate(7);

Plate plate4 = new Plate(4);

Plate plate3 = new Plate(4);

Plate plate2 = new Plate(2);

Plate plate1 = new Plate(1);

Plate[] plates = new Plate[] { plate6, plate5, plate4, plate3, plate2, plate1 };

BinaryFormatter bf = new BinaryFormatter();

using (FileStream fs = new FileStream(dir + Path.DirectorySeparatorChar + filename, FileMode.OpenOrCreate))

{

bf.Serialize(fs, plates);

}

using (FileStream fs = new FileStream(dir + Path.DirectorySeparatorChar + filename, FileMode.OpenOrCreate))

{

plates = bf.Deserialize(fs) as Plate[];

}

var t3res = Task3(plates);

foreach (var item in t3res)

{

for (int i = 0; i < t3res.PopTail().Size - item.Size; i++)

{

Console.Write(" ");

}

Console.WriteLine(item.Visual);

}

#endregion

#region Task4

Console.WriteLine("Task 4");

Console.WriteLine();

filename = "Task4.txt";

string[] strings;

strings = File.ReadAllLines(dir + Path.DirectorySeparatorChar + filename);

var t4res = Task4(strings);

Console.WriteLine("Баланс круглых скобок выполнен - " + t4res);

#endregion

#region Task5

#endregion

Console.WriteLine("Task 5");

Console.WriteLine();

filename = "Task5.txt";

string[] strings2;

strings2 = File.ReadAllLines(dir + Path.DirectorySeparatorChar + filename);

var t5res = Task5(strings2);

Console.WriteLine("Баланс квадратных скобок выполнен - " + t5res);

#region Task6

#endregion

Console.WriteLine("Task 6");

Console.WriteLine();

filename = "Task6.txt";

string[] strings3;

strings3 = File.ReadAllLines(dir + Path.DirectorySeparatorChar + filename);

Task6(strings3);

Console.WriteLine();

#region Task7

#endregion

Console.WriteLine("Task 7");

Console.WriteLine();

filename = "Task7.txt";

string[] strings4;

strings4 = File.ReadAllLines(dir + Path.DirectorySeparatorChar + filename);

Task7(strings4);

Console.WriteLine();

#region Task8

Console.WriteLine("Task 8");

Console.WriteLine();

filename = "Task8\_source.txt";

string[] strings5;

strings5 = File.ReadAllLines(dir + Path.DirectorySeparatorChar + filename);

Task8(strings5, dir);

Console.WriteLine();

#endregion

#region Task9

Console.WriteLine("Task 9");

Console.WriteLine();

filename = "Task9.txt";

string string6;

string6 = File.ReadAllText(dir + Path.DirectorySeparatorChar + filename);

Console.WriteLine("Значение заданного логического выражения: " + Task9(string6));

Console.WriteLine();

#endregion

#region Task10

Console.WriteLine("Task 10");

Console.WriteLine();

filename = "Task10.txt";

string string7;

string7 = File.ReadAllText(dir + Path.DirectorySeparatorChar + filename);

Console.WriteLine("Значение заданного алгебраического выражения: " + Task10(string7));

Console.WriteLine();

#endregion

#region Task11

Console.WriteLine("Task 11");

Console.WriteLine();

filename = "Task11.txt";

string string8;

string8 = File.ReadAllText(dir + Path.DirectorySeparatorChar + filename);

Console.WriteLine("Введённая формула корректна: " + Task11(string8));

Console.WriteLine();

#endregion

#endregion

} /\*

Отсортировать строки файла, содержащие названия книг, в алфавитном порядке с

использованием двух деков.

\*/

public static MyDeque<T> Task1<T>(MyDeque<T> input) where T : IComparable<T>

{

var first = new MyDeque<T>();

var second = new MyDeque<T>();

first.Dequeue(input.RemoveFirst());

while (input.Count > 0)

{

var temp = input.RemoveFirst();

if (temp.CompareTo(first.PopHead()) <= 0)//если temp должен быть перед первым

{

first.Dequeue(temp);//ставим в первый дек в начало

}

else if (temp.CompareTo(first.PopTail()) < 0)//если temp должен быть перед последним

{

while (temp.CompareTo(first.PopTail()) <= 0)//пока temp не будет >= first.Tail

{

second.Enqueue(first.RemoveLast());

}

first.Enqueue(temp);

while (second.Count > 0)

{

first.Enqueue(second.RemoveLast());

}

}

else

{

first.Enqueue(temp);

}

}

return first;

}

/\*

\* Дек содержит последовательность символов для шифровки сообщений. Дан

\* текстовый файл, содержащий зашифрованное сообщение. Пользуясь деком,

\* расшифровать текст. Известно, что при шифровке каждый символ сообщения

\* заменялся следующим за ним в деке по часовой стрелке через один.

\*/

public static string Task2\_crypt(string message, string key, string path\_to\_save)

{

var myDeq = new MyDeque<char>();

var res = "";

foreach (var ch in key)

{

myDeq.Enqueue(ch);

}

foreach (var ch in message)

{

while (myDeq.PopHead() != ch)

{

myDeq.Enqueue(myDeq.RemoveFirst());

}

myDeq.Enqueue(myDeq.RemoveFirst());

myDeq.Enqueue(myDeq.RemoveFirst());

res += myDeq.PopHead();

myDeq.Dequeue(myDeq.RemoveLast());

myDeq.Dequeue(myDeq.RemoveLast());

}

File.WriteAllText(path\_to\_save, res);

return res;

}

public static string Task2\_encrypt(string message, string key)

{

var myDeq = new MyDeque<char>();

var res = "";

foreach (var ch in key)

{

myDeq.Enqueue(ch);

}

foreach (var ch in message)

{

while (myDeq.PopTail() != ch)

{

myDeq.Dequeue(myDeq.RemoveLast());

}

myDeq.Dequeue(myDeq.RemoveLast());

myDeq.Dequeue(myDeq.RemoveLast());

res += myDeq.PopTail();

myDeq.Enqueue(myDeq.RemoveFirst());

myDeq.Enqueue(myDeq.RemoveFirst());

}

return res;

}

public static string Task2\_keygen(string input)

{

string str = "";

IEnumerable<char> res = input.Distinct();

foreach (var item in res)

{

str += item;

}

return str;

}

/\*

Даны три стержня и n дисков различного размера. Диски можно надевать на

cтержни, образуя из них башни. Перенести n дисков со стержня А на стержень С,

сохранив их первоначальный порядок. При переносе дисков необходимо соблюдать

следующие правила:- на каждом шаге со стержня на стержень переносить только один диск;

- диск нельзя помещать на диск меньшего размера;

- для промежуточного хранения можно использовать стержень В.

Реализовать алгоритм, используя три стека вместо стержней А, В, С. Информация о дисках хранится в исходном файле.\*/

public static MyDeque<Plate> Task3(Plate[] input)

{

var res = new MyDeque<Plate>();

var stacks = new MyStack<Plate>[3] { new MyStack<Plate>(), new MyStack<Plate>(), new MyStack<Plate>() };

foreach (var item in input)

{

stacks[0].Push(item);

}

PlacePlate(stacks[0], stacks[2], stacks[1], stacks[0].Count);

foreach (var item in stacks[2])

{

res.Enqueue(item);

}

return res;

void PlacePlate(MyStack<Plate> first, MyStack<Plate> third, MyStack<Plate> second, int count)

{

if (count != 0)

{

PlacePlate(first, second, third, count - 1);

third.Push(first.Pop());

PlacePlate(second, third, first, count - 1);

}

}

}

/\*

\* Дан текстовый файл с программой на алгоритмическом языке. За один просмотр

\* файла проверить баланс круглых скобок в тексте, используя стек.

\*/ public static bool Task4(string[] text)

{

var stack = new MyStack<char>();

foreach (var str in text)

{

foreach (var ch in str)

{

if (ch == '(')

{

stack.Push(ch);

}

if (ch == ')' && stack.Count != 0)

{

stack.Pop();

}

else if (ch == ')' && stack.Count == 0)

{

return false;

}

}

}

if (stack.IsEmpty())

{

return true;

}

return false;

}

/\*

\* Дан текстовый файл с программой на алгоритмическом языке. За один просмотр

\* файла проверить баланс квадратных скобок в тексте, используя дек.

\*/

public static bool Task5(string[] text)

{

var deq = new MyDeque<char>();

foreach (var str in text)

{

foreach (var ch in str)

{

if (ch == '[')

{

deq.Dequeue(ch);

}

if (ch == ']')

{

deq.Enqueue(ch);

}

}

}

while (deq.Count > 1)

{

if (deq.PopHead() == '[' && deq.PopTail() == ']')

{

deq.RemoveFirst();

deq.RemoveLast();

}

else

{

return false;

}

}

if (deq.IsEmpty())

{

return true;

}

return false;

}

/\*

\* Дан файл из символов. Используя стек, за один просмотр файла напечатать

\* сначала все цифры, затем все буквы, и, наконец, все остальные символы, сохраняя

\* исходный порядок в каждой группе символов.

\*/ public static void Task6(string[] text)

{

var stack = new MyStack<char>();

for (int i = text.Length - 1; i >= 0; i--)

{

for (int j = text[i].Length - 1; j >= 0; j--)

{

if (!char.IsDigit(text[i][j]) && !char.IsLetter(text[i][j]))

{

stack.Push(text[i][j]);

}

}

}

for (int i = text.Length - 1; i >= 0; i--)

{

for (int j = text[i].Length - 1; j >= 0; j--)

{

if (char.IsLetter(text[i][j]))

{

stack.Push(text[i][j]);

}

}

}

for (int i = text.Length - 1; i >= 0; i--)

{

for (int j = text[i].Length - 1; j >= 0; j--)

{

if (char.IsDigit(text[i][j]))

{

stack.Push(text[i][j]);

}

}

}

foreach (var item in stack)

{

Console.Write(item);

}

}

/\*

Дан файл из целых чисел. Используя дек, за один просмотр файла напечатать

сначала все отрицательные числа, затем все положительные числа, сохраняя

исходный порядок в каждой группе.

\*/ public static void Task7(string[] text)

{

string str = null;

foreach (var st in text)

{

str += st + ' ';

}

var strings = str.Trim().Split(' ');

var nums = new List<int>();

foreach (var st in strings)

{

nums.Add(int.Parse(st));

}

var deq = new MyDeque<int>();

foreach (var num in nums)

{

if (num < 0)

{

deq.Enqueue(num);

}

}

foreach (var num in nums)

{

if (num >= 0)

{

deq.Enqueue(num);

}

}

foreach (var item in deq)

{

Console.Write(item + " ");

}

}

/\*

\* Дан текстовый файл. Используя стек, сформировать новый текстовый файл,

\* содержащий строки исходного файла, записанные в обратном порядке: первая

\* строка становится последней, вторая – предпоследней и т.д.

\*/ public static void Task8(string[] text, string dir)

{

var stack = new MyStack<string>();

var filename = "Task8\_output.txt";

foreach (var item in text)

{

stack.Push(item);

}

if (File.Exists(dir + Path.DirectorySeparatorChar + filename))

{

File.WriteAllText(dir + Path.DirectorySeparatorChar + filename, "");

}

while (stack.Count > 0)

{

File.AppendAllText(dir + Path.DirectorySeparatorChar + filename, stack.Pop() + "\n");

}

}

/\*

Дан текстовый файл. Используя стек, вычислить значение логического выражения,

записанного в текстовом файле в следующей форме:

< ЛВ > ::= T | F | (N<ЛВ>) | (<ЛВ>A<ЛВ>) | (<ЛВ>X<ЛВ>) | (<ЛВ>O<ЛВ>),

где буквами обозначены логические константы и операции:

T – True, F – False, N – Not, A – And, X – Xor, O – Or.

\*/ public static bool Task9(string input)

{

if (!Task4(new string[] { input }))

{

Console.WriteLine("Некорректный синтаксис выражения.");

return false;

}

input = input.ToUpper();

var stack = new MyStack<char>();

var sout = string.Empty;

foreach (var c in input)

{

if (IsConstant(c))

{

sout += c;

continue;

}

else

{

switch (c)

{

case '(':

stack.Push(c);

break;

case ')':

while (stack.Peek() != '(')

{

sout += stack.Pop();

}

stack.Pop();

break;

case 'A':

case 'O':

case 'X':

case 'N':

if (stack.IsEmpty())

{

stack.Push(c);

}

else

{

while (stack.Count > 0 && (GetPriority(c) <= stack.Peek()))

{

if ('(' == stack.Peek())

{

break;

}

else

{

sout += stack.Pop();

}

}

stack.Push(c);

}

break;

default:

{

Console.WriteLine("Некорректный синтаксис выражения.");

return false;

}

break;

}

}

}

while (!stack.IsEmpty())

{

sout += stack.Pop();

}

while (!IsConstant(sout[sout.Length - 1]))

{

if (IsConstant(sout[0]))

{

stack.Push(sout[0]);

sout = sout.Substring(1);

}

else

{

var oper = sout[0];

sout = sout.Substring(1);

switch (oper)

{

case 'A':

sout = AndOper() + sout;

break;

case 'O':

sout = OrOper() + sout;

break;

case 'X':

sout = XorOper() + sout;

break;

case 'N':

sout = DenyOper() + sout;

break;

default:

break;

}

}

}

if (stack.Count != 0)

{

Console.WriteLine("Некорректный синтаксис выражения.");

return false;

}

bool res = (true && false) || (!false && (!false && true) && (!(true && (true ^ false) || (true & true))));

if (sout[0] == 'T')

{

return true;

}

return false;

char AndOper()

{

var first = stack.Pop();

var second = stack.Pop();

if (first == 'F' || second == 'F')

{

return 'F';

}

return 'T';

}

char OrOper()

{

var first = stack.Pop();

var second = stack.Pop();

if (first == 'F' && second == 'F')

{

return 'F';

}

return 'T';

}

char XorOper()

{

var first = stack.Pop();

var second = stack.Pop();

if (first == 'F' && second == 'F' || second == 'T' && first == 'T')

{

return 'F';

}

return 'T';

}

char DenyOper()

{

var first = stack.Pop();

if (first == 'F')

{

return 'T';

}

return 'F';

}

int GetPriority(char c)

{

switch (c)

{

case '(':

return 3;

case 'N':

case 'A':

return 2;

case 'O':

case 'X':

return 1;

default:

break;

}

return 0;

}

bool IsConstant(char c)

{

if (c == 'T' || c == 'F')

{

return true;

}

return false;

}

}

/\*

\* Дан текстовый файл. В текстовом файле записана формула следующего вида:

\* <Формула> ::= <Цифра> | M(<Формула>,<Формула>) | N(Формула>,<Формула>)

\* < Цифра > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

\* где буквами обозначены функции:

\* M – определение максимума, N – определение минимума.

\* Используя стек, вычислить значение заданного выражения

\*/ public static int Task10(string input)

{

if (!Task4(new string[] { input }))

{

Console.WriteLine("Некорректный синтаксис выражения.");

return -1;

}

input = input.ToUpper();

var stack = new MyStack<char>();

var sout = string.Empty;

foreach (var c in input)

{

if (char.IsDigit(c))

{

sout += c;

continue;

}

else

{

switch (c)

{

case '(':

case ',':

case 'M':

case 'N':

stack.Push(c);

break;

case ')':

if (stack.Peek() != ',')

{

Console.WriteLine("Некорректный синтаксис выражения.");

return -1;

}

stack.Pop();

stack.Pop();

sout += stack.Pop();

break;

default:

{

Console.WriteLine("Некорректный синтаксис выражения.");

return -1;

}

break;

}

}

}

while (!stack.IsEmpty())

{

sout += stack.Pop();

}

while (!char.IsDigit(sout[sout.Length - 1]))

{

if (char.IsDigit(sout[0]))

{

stack.Push(sout[0]);

sout = sout.Substring(1);

}

else

{

var oper = sout[0];

sout = sout.Substring(1);

switch (oper)

{

case 'M':

sout = Max() + sout;

break;

case 'N':

sout = Min() + sout;

break;

default:

{

Console.WriteLine("Некорректный синтаксис выражения.");

return -1;

}

break;

}

}

}

if (stack.Count != 0)

{

Console.WriteLine("Некорректный синтаксис выражения.");

return -1;

}

int res = Math.Max(Math.Max(7, 9), Math.Min(3, 7));

return Convert.ToInt32(sout[0].ToString());

int Max()

{

int first = Convert.ToInt32(stack.Pop().ToString());

int second = Convert.ToInt32(stack.Pop().ToString());

return first >= second ? first : second;

}

int Min()

{

int first = Convert.ToInt32(stack.Pop().ToString());

int second = Convert.ToInt32(stack.Pop().ToString());

return first < second ? first : second;

}

}

/\*

\*Дан текстовый файл. Используя стек, проверить, является ли содержимое

\*текстового файла правильной записью формулы вида:

\*< Формула > ::= < Терм > | < Терм > + < Формула > | < Терм > - < Формула >

\*< Терм > ::= < Имя > | (< Формула >)

\*< Имя > ::= x | y | z

\*/ public static bool Task11(string input)

{

if (!Task4(new string[] { input }))

{

Console.WriteLine("Некорректный синтаксис выражения.");

return false;

}

input = input.ToUpper();

var stack = new MyStack<char>();

var sout = string.Empty;

foreach (var c in input)

{

if (IsName(c))

{

sout += c;

continue;

}

else

{

switch (c)

{

case '(':

case '+':

case '-':

stack.Push(c);

break;

case ')':

while (stack.Peek() != '(')

{

sout += stack.Pop();

}

stack.Pop();

break;

default:

{

Console.WriteLine("Некорректный синтаксис выражения.");

return false;

}

}

}

}

while (!stack.IsEmpty())

{

sout += stack.Pop();

}

while (!IsName(sout[sout.Length - 1]))

{

if (IsName(sout[0]))

{

stack.Push(sout[0]);

sout = sout.Substring(1);

}

else

{

var oper = sout[0];

sout = sout.Substring(1);

switch (oper)

{

case '+':

case '-':

sout = PMoper() + sout;

break;

default:

{

Console.WriteLine("Некорректный синтаксис выражения.");

return false;

}

}

}

}

if (stack.Count != 0)

{

Console.WriteLine("Некорректный синтаксис выражения.");

return false;

}

return true;

char PMoper()

{

var first = stack.Pop();

var second = stack.Pop();

if (first == 'X' || first == 'Y' || first == 'Z')

{

if (second == 'X' || second == 'Y' || second == 'Z')

{

return 'X';

}

}

return 'T';

}

bool IsName(char c)

{

if (c == 'X' || c == 'Y' || c == 'Z')

{

return true;

}

return false;

}

}

}

public class MyStack<T>:IEnumerable<T>

{

Node<T> Head { get; set; }

public int Count { get; protected set; }

public MyStack()

{

Count = 0;

Head = null;

}

public T Pop()

{

if (Count!=0)

{

var val = Peek();

Head = Head.Next;

Count--;

return val;

}

throw new InvalidOperationException("Stack empty.");

}

virtual public bool IsEmpty()

{

if (Count==0)

{

return true;

}

else

{

return false;

}

}

public void Push(T value)

{

Node<T> newbie = new Node<T>(value);

Count++;

newbie.Next = Head;

Head = newbie;

}

public T Peek()

{

return Head.Value;

}

public IEnumerator<T> GetEnumerator()

{

var temp = Head;

while (temp.HasNext())

{

yield return temp.Value;

temp = temp.Next;

}

yield return temp.Value;

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

internal class Node<T>

{

protected internal T Value { get; set; }

protected internal Node<T> Next { get; set; }

protected internal Node(T val)

{

Value = val;

Next = null;

}

protected internal bool HasNext()

{

if (Next!=null)

{

return true;

}

return false;

}

}

}

public class MyDeque<T>:IEnumerable<T>

{

public int Count { get; private set; }

DoubleNode<T> Head { get; set; }

DoubleNode<T> Tail { get; set; }

public MyDeque() : base()

{

Tail = null;

Head = null;

}

/// <summary>

/// Добавляет элемент в конец дека

/// </summary>

/// <returns></returns>

public void Enqueue(T val)

{

DoubleNode<T> newbie = new DoubleNode<T>(val);

if (Head == null)

{

Head = newbie;

Tail = newbie;

Count++;

return;

}

if (Tail != null)

{

Tail.Right = newbie;

newbie.Left = Tail;

Tail = newbie;

Count++;

}

}

/// <summary>

/// Извлекает головной элемент

/// </summary>

/// <returns></returns>

public T RemoveFirst()

{

if (Head != null)

{

var val = Head.Value;

var newhead = Head.Right;

Head.Right = null;

Head.Left = null;

if (newhead != null)

{

newhead.Left = null;

}

Head = newhead;

Count--;

if (Count<2)

{

Tail = Head;

}

return val;

}

else throw new InvalidOperationException("Dequeue empty.");

}

/// <summary>

/// Извлекает конечный элемент

/// </summary>

/// <returns></returns>

public T RemoveLast()

{

if (Tail != null)

{

var val = Tail.Value;

var newtail = Tail.Left;

Tail.Left = null;

if (newtail != null)

{

newtail.Right = null;

}

Tail = newtail;

Count--;

if (Count<2)

{

Head = Tail;

}

return val;

}

else throw new InvalidOperationException("Dequeue empty.");

}

/// <summary>

/// Добавляет элемент в начало дека

/// </summary>

/// <param name="val"></param>

public void Dequeue(T val)

{

DoubleNode<T> newbie = new DoubleNode<T>(val);

if (Head == null)

{

Head = newbie;

Tail = newbie;

Count++;

return;

}

if (Head != null)

{

Head.Left = newbie;

newbie.Right = Head;

Head = newbie;

Count++;

}

}

/// <summary>

/// Возвращает значение головного элемента без извлечения

/// </summary>

/// <returns></returns>

public T PopHead()

{

if (Head != null)

{

return Head.Value;

}

else throw new InvalidOperationException("Dequeue empty.");

}

/// <summary>

/// Возвращает значение последнего элемента без извлечения

/// </summary>

/// <returns></returns>

public T PopTail()

{

if (Tail != null)

{

return Tail.Value;

}

else throw new InvalidOperationException("Dequeue empty.");

}

/// <summary>

/// Проверяет дек на пустоту

/// </summary>

/// <returns></returns>

public bool IsEmpty()

{

if (Count != 0)

{

return false;

}

else

{

return true;

}

}

public IEnumerator<T> GetEnumerator()

{

var temp = Head;

while (temp.HasNext())

{

yield return temp.Value;

temp = temp.Right;

}

yield return temp.Value;

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

private protected class DoubleNode<T>

{

internal T Value { get; set; }

protected internal DoubleNode<T> Right { get; set; }

protected internal DoubleNode<T> Left { get; set; }

protected internal DoubleNode(T val)

{

Value = val;

Right = null;

Left = null;

}

protected internal bool HasNext()

{

if (Right!=null)

{

return true;

}

return false;

}

}

}

Результаты выполнения программы представлены на рисунках ниже:

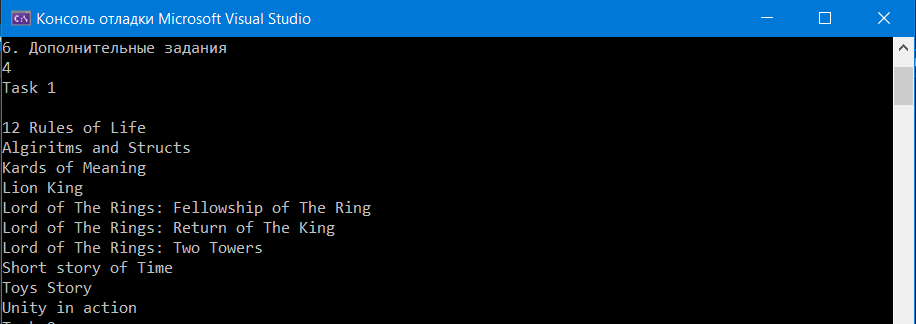


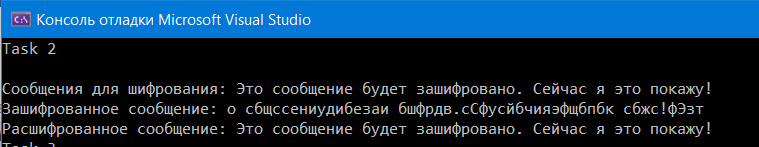
Рис. 1 – Результат сортировки содержимого файла с помощью двух стеков  
  


Рис. 2 – Результат шифрования сообщения с помочью дека

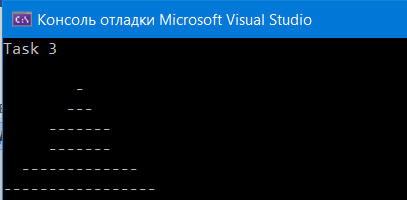
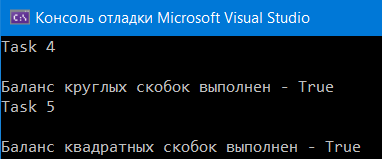


Рис. 3 – Результат решения задачи о Ханойских башнях с помощью трёх стеков

  
Рис. 4 – Результат проверки баланса круглых и квадратных скобок в строках файла

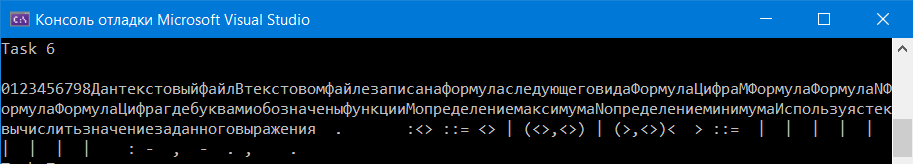


Рис. 5 – Результат записи содержимого файла в порядке «цифры, буквы, остальные символы» с помощью стека

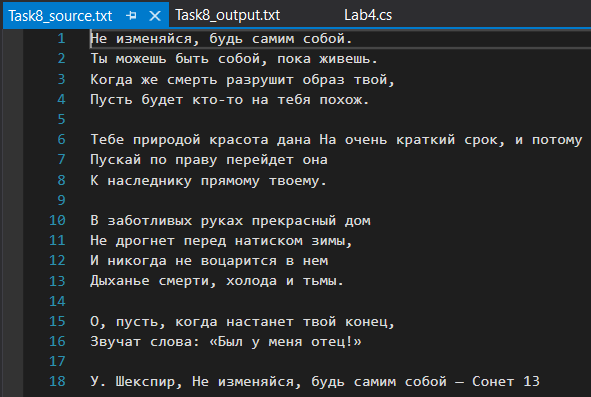
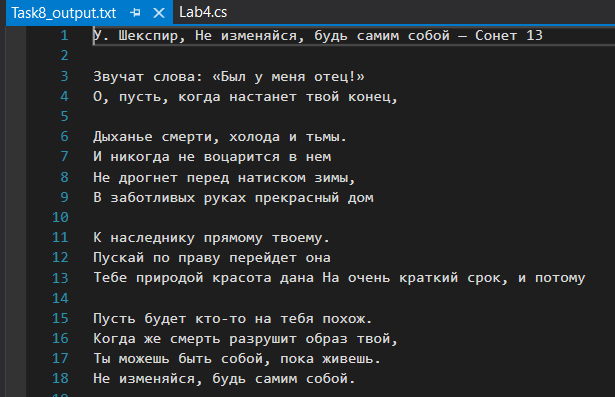
 

Рис. 6 – Результат записи строк файла в обратном порядке с помощью стека

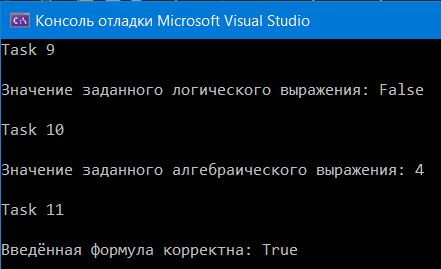


Рис. 6 – Результат проверки корректности ввода логического и алгебраического выражений с помощью стека

**3. Вывод**

В ходе данной работы были реализованы коллекции для работы с данными Stack и Deque, а также выполнены прилагающиеся к ним задания. Каждое из 11 заданий является широкоизвестной прикладной задачей, которая эффективно решается именно с помощью этих структур данных. Полученные навыки как при реализации структур, так и решении задач, являются необходимыми в работе программиста.

Дополнительные задания:

Дан массив отрезков intervals, в котором intervals[i] = [starti , endi ], некоторые отрезки могут пересекаться. Напишите функцию, которая объединяет все пересекающиеся отрезки в один и возвращает новый массив непересекающихся отрезков

static int[][] IntervalsProblem(int[][] input)

{

if (input == null || input.Length == 0) return new int[0][];

Array.Sort(input, 0, input.Length, new IntervalComparer());

var stack = new Stack<int[]>();

var curr = input[0];

for (int i = 1; i < input.Length; i++)

{

var next = input[i];

if (curr[1] >= next[0])

curr = new int[] { curr[0], Math.Max(curr[1], next[1]) };

else

{

stack.Push(curr);

curr = next;

}

}

stack.Push(curr);

var res = new int[stack.Count][];

for (int i = res.Length - 1; i >= 0; i--)

res[i] = stack.Pop();

return res;

}

class IntervalComparer : IComparer<int[]>

{

public int Compare(int[] x, int[] y)

{

return x[0].CompareTo(y[0]);

}

}

Задан правильный *N*-угольник. Требуется выбрать наименьшее количество его вершин, которые также образуют правильный многоугольник.

static int RightAngle(int input)

{

if (input % 3 == 0)

{

return 3;

}

else if (input % 2 == 1)

{

int sqrt = (int)Math.Sqrt(input);

if (sqrt<3)

{

return input;

}

int res = sqrt;

for (int j = sqrt; j > 2; j--)

{

if (input % j == 0 && IsSimple(j))

{

res = j;

}

}

return res;

}

else if (input % 2 == 0 && input % 4 != 0)

{

int sqrt = (int)Math.Sqrt(input);

if (sqrt < 3)

{

return input;

}

int res = sqrt;

for (int j = sqrt; j > 2; j--)

{

if (input % j == 1 && IsSimple(j))

{

res = j;

}

}

if (res == sqrt)

{

return input / 2;

}

return res;

}

else

{

return 4;

}

bool IsSimple(int num)

{

int counter = 0;

for (int i = 1; i <= num; i++)

{

if (num % i == 0)

{

counter++;

}

if (counter > 2)

{

return false;

}

}

if (counter != 2)

{

return false;

}

return true;

}

}