**Федеральное агентство связи**

**Государственное бюджетное образовательное учреждение высшего образования Ордена Трудового Красного Знамени**

**«Московский технический университет связи и информатики»**

**Лабораторная работа № 4**

«Реализация стека/дека»

Выполнила: студентка группы БВТ1904

Хорикова Софья Гарегиновна

Проверил: Павликов Артём Евгеньевич

Москва, 2020

1. Цель работы

Реализовать следующие структуры данных:

● Стек (stack):

операции для стека: инициализация, проверка на пустоту, добавление

нового элемента в начало, извлечение элемента из начала;

● Дек (двусторонняя очередь, deque):

операции для дека: инициализация, проверка на пустоту, добавление

нового элемента в начало, добавление нового элемента в конец,

извлечение элемента из начала, извлечение элемента из конца.

Разработать программу обработки данных, содержащихся в заранее

подготовленном txt-файле, в соответствии с заданиями, применив

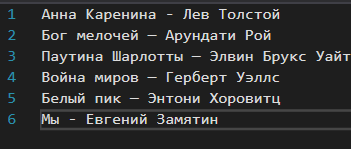
указанную в задании структуру данных. Результат работы программы

вывести на экран и сохранить в отдельном txt-файле.

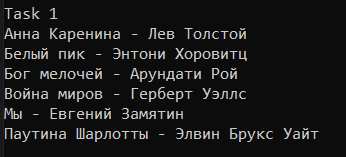
1. Заданные условия, выполнение, код

Задача 1

Текстовый файл №1.



Вывод в консоль:



Код:

/\*

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

использованием двух деков.

\*/

public static Deque<T> Task1<T>(Deque<T> input) where T : IComparable<T> //чтобы сравнивать

{

var first = new Deque<T>(); //для вывода

var second = new Deque<T>(); //буферный

first.AddFirst(input.TakeFirst()); //добавляем 1 строку

while (input.Count > 0) //пока на входе не останется ничего

{

var temp = input.TakeFirst(); //доб след строку

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

{

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

}

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

{

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

{

second.AddLast(first.RemoveLast());

}

first.AddLast(temp);

while (second.Count > 0)

{

first.AddLast(second.RemoveLast());

}

}

else

{

first.AddLast(temp);

}

}

return first;

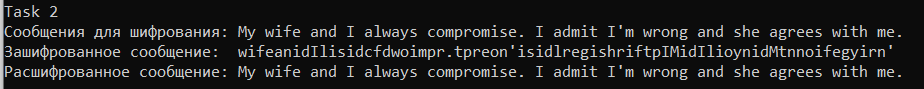
}

Задание 2

Текстовый файл 2



Вывод в консоль:



Текстовый файл 2.1



Текстовый файл 2.2



Код:

/\*

\* 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 Deque<char>();

var res = "";

foreach (var ch in key)

{

myDeq.AddLast(ch);

}

foreach (var ch in message)

{

while (myDeq.LookFirst() != ch)

{

myDeq.AddLast(myDeq.TakeFirst());

}

myDeq.AddLast(myDeq.TakeFirst());

myDeq.AddLast(myDeq.TakeFirst());

res += myDeq.LookFirst();

myDeq.AddFirst(myDeq.RemoveLast());

myDeq.AddFirst(myDeq.RemoveLast());

}

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

return res;

}

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

{

var myDeq = new Deque<char>();

var res = "";

foreach (var ch in key)

{

myDeq.AddLast(ch);

}

foreach (var ch in message)

{

while (myDeq.LookLast() != ch)

{

myDeq.AddFirst(myDeq.RemoveLast());

}

myDeq.AddFirst(myDeq.RemoveLast());

myDeq.AddFirst(myDeq.RemoveLast());

res += myDeq.LookLast();

myDeq.AddLast(myDeq.TakeFirst());

myDeq.AddLast(myDeq.TakeFirst());

}

return res;

}

public static string Task2\_keygen(string input)

{

string str = "";

IEnumerable<char> res = input.Distinct(); //убираем повторяющиеся символы

foreach (var item in res)

{

str += item;

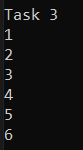
}

return str;

}

Задача 3

Вывод на консоль:



Код:

/\*

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 Deque<int> Task3(int[] input)

{

var res = new Deque<int>();

var stacks = new Stack<int>[3] { new Stack<int>(), new Stack<int>(), new Stack<int>() }; //три стека, объединенных в массив

foreach (var item in input)

{

stacks[0].Enter(item);

}

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

while (!stacks[2].IsEmpty())

{

res.AddLast(stacks[2].Take()); //записываем в результат

}

return res;

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

{

if (count != 0)

{

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

third.Enter(first.Take());

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

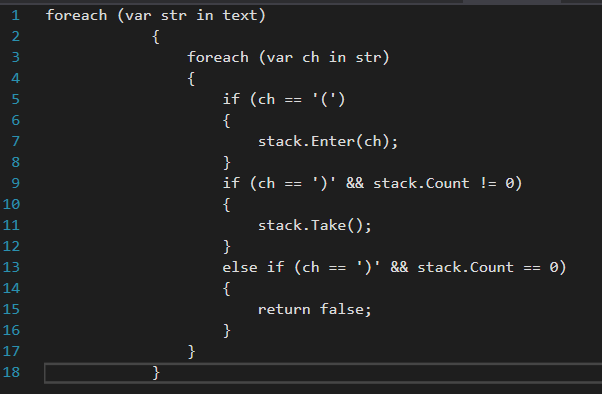
}

}

}

Задача 4

Текстовый файл №4



Вывод на консоль:



Код:

/\*

\* 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)

{

//храним открывающую, при встрече закрывающей мы убираем 1 элемент

var stack = new Stack<char>();

foreach (var str in text)

{

foreach (var ch in str)

{

if (ch == '(')

{

stack.Enter(ch);

}

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

{

stack.Take();

}

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

{

return false;

}

}

}

if (stack.IsEmpty())

{

return true;

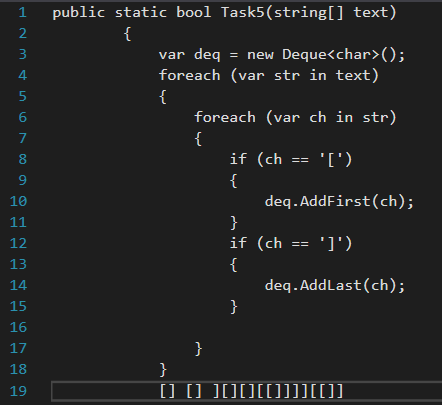
}

return false;

}

Задача 5

Текстовый файл №5



Вывод на консоль:



Код:

/\*

\* 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)

{

//открывающие с одного конца

//закрывающие с другого

//попарно убираем, если что-то осталось, то нет баланса

var deq = new Deque<char>();

foreach (var str in text)

{

foreach (var ch in str)

{

if (ch == '[')

{

deq.AddFirst(ch);

}

if (ch == ']')

{

deq.AddLast(ch);

}

}

}

while (deq.Count > 1)

{

if (deq.LookFirst() == '[' && deq.LookLast() == ']')

{

deq.TakeFirst();

deq.RemoveLast();

}

else

{

return false;

}

}

if (deq.IsEmpty())

{

return true;

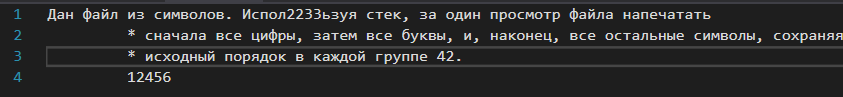
}

return false;

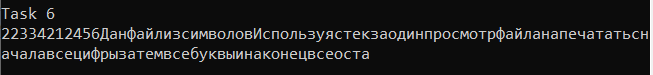
}

Задача 6

Текстовый файл №6



Вывод на консоль:



Код:

/\*

\* 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 Stack<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.Enter(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.Enter(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.Enter(text[i][j]);

}

}

}

for (int i = 0; i < stack.Count; i++)

{

Console.Write(stack.Take());

}

}

Задача 7

Текстовый файл №7



Вывод на консоль:



Код:

/\*

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(' '); //все что разделено пробелом, в массив строк

var nums = new List<int>();

foreach (var st in strings)

{

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

}

var deq = new Deque<int>();

foreach (var num in nums)

{

if (num < 0)

{

deq.AddLast(num);

}

}

foreach (var num in nums)

{

if (num >= 0)

{

deq.AddLast(num);

}

}

for (int i = 0; i < deq.Count; i++)

{

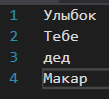
Console.Write(deq.TakeFirst() + " ");

}

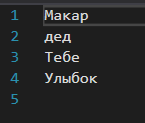
}

Задача 8

Текстовый файл №81



Текстовый файл №82



Код:

/\*

\* 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 Stack<string>();

var filename = "T82.txt";

foreach (var item in text)

{

stack.Enter(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.Take() + "\n");

}

}

Задача 9

Текстовый файл №9



Вывод на консоль:



Код:

/\*

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)

{

input = input.ToUpper();

var stack = new Stack<char>();

var sout = string.Empty;

foreach (var c in input)

{

if (IsConstant(c))

{

sout += c;

continue;

}

else

{

switch (c)

{

case '(':

stack.Enter(c);

break;

case ')':

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

{

sout += stack.Take();

}

stack.Take();

break;

case 'A':

case 'O':

case 'X':

case 'N':

if (stack.IsEmpty())

{

stack.Enter(c);

}

else

{

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

{

if ('(' == stack.Look())

{

break;

}

else

{

sout += stack.Take();

}

}

stack.Enter(c);

}

break;

default:

{

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

return false;

}

}

}

}

while (!stack.IsEmpty())

{

sout += stack.Take();

}

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

{

if (IsConstant(sout[0]))

{

stack.Enter(sout[0]);

sout = sout.Substring(1);

}

else

{

var oper = sout[0];

sout = sout.Substring(1);

switch (oper)

{

case 'A':

sout = And() + sout;

break;

case 'O':

sout = Or() + sout;

break;

case 'X':

sout = Xor() + sout;

break;

case 'N':

sout = Not() + sout;

break;

default:

break;

}

}

}

if (stack.Count != 0)

{

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

return false;

}

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

{

return true;

}

return false;

char And()

{

var first = stack.Take();

var second = stack.Take();

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

{

return 'F';

}

return 'T';

}

char Or()

{

var first = stack.Take();

var second = stack.Take();

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

{

return 'F';

}

return 'T';

}

char Xor()

{

var first = stack.Take();

var second = stack.Take();

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

{

return 'F';

}

return 'T';

}

char Not()

{

var first = stack.Take();

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;

}

}

Задача 10

Текстовый файл №10



Вывод на консоль:



Код:

/\*

\* 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)

{

input = input.ToUpper();

var stack = new Stack<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.Enter(c);

break;

case ')':

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

{

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

return -1;

}

stack.Take();

stack.Take();

sout += stack.Take();

break;

default:

{

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

return -1;

}

}

}

}

while (!stack.IsEmpty())

{

sout += stack.Take();

}

while (!char.IsDigit(sout[sout.Length - 1])) //пока не останется 1 цифра в строке

{

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

{

stack.Enter(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;

}

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

int Max()

{

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

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

return first >= second ? first : second;

}

int Min()

{

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

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

return first < second ? first : second;

}

}

Задача 11

Текстовый файл №11



Вывод на консоль:



Код:

/\*

\*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)

{

input = input.ToUpper();

var stack = new Stack<char>();

var sout = string.Empty;

foreach (var c in input)

{

if (IsName(c))

{

sout += c;

continue;

}

else

{

switch (c)

{

case '(':

case '+':

case '-':

stack.Enter(c);

break;

case ')':

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

{

sout += stack.Take();

}

stack.Take();

break;

default:

{

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

return false;

}

}

}

}

while (!stack.IsEmpty())

{

sout += stack.Take();

}

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

{

if (IsName(sout[0]))

{

stack.Enter(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.Take();

var second = stack.Take();

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;

}

}

Реализация стека:

public class Stack<T>

{

Node<T> First { get; set; }

public int Count { get; protected set; }

public Stack()

{

Count = 0;

First = null;

}

/// <summary>

/// берёт верхний элемент в стэке

/// </summary>

/// <returns></returns>

public T Take()

{

if (Count != 0) //если в стеке что-то есть

{

var val = Look();

First = First.Next; //делаем следующий эл-т верхним

Count--;

return val;

}

throw new InvalidOperationException("Stack is empty.");

}

virtual public bool IsEmpty()

{

if (Count == 0)

{

return true;

}

else

{

return false;

}

}

/// <summary>

/// добавляет элемент сверху

/// </summary>

/// <param name="value"></param>

public void Enter(T value)

{

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

Count++;

newNode.Next = First;

First = newNode;

}

public T Look()

{

return First.Value; //не извлекая 1 эл-т

}

internal class Node<T> //один элемент

{

protected internal T Value { get; set; }

protected internal Node<T> Next { get; set;} //хранение ссылки на следующую ноду

protected internal Node(T val)

{

Value = val;

Next = null;

}

}

}

Реализация дека:

public class Deque<T>

{

public int Count { get; private set; }

DoubleNode<T> First { get; set; }

DoubleNode<T> Last { get; set; }

public Deque()

{

Last = null;

First = null;

}

/// <summary>

/// Добавляет элемент в конец дека

/// </summary>

/// <returns></returns>

public void AddLast(T val)

{

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

if (First == null)

{

First = newbie;

Last = newbie;

Count++;

return;

}

if (Last != null)

{

Last.Right = newbie;

newbie.Left = Last;

Last = newbie;

Count++;

}

}

/// <summary>

/// Извлекает головной элемент

/// </summary>

/// <returns></returns>

public T TakeFirst()

{

if (First != null)

{

var val = First.Value;

var newhead = First.Right;

First.Right = null;

First.Left = null;

if (newhead != null)

{

newhead.Left = null;

}

First = newhead;

Count--;

if (Count < 2)

{

Last = First;

}

return val;

}

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

}

/// <summary>

/// Извлекает конечный элемент

/// </summary>

/// <returns></returns>

public T RemoveLast()

{

if (Last != null)

{

var val = Last.Value;

var newtail = Last.Left;

Last.Left = null;

if (newtail != null)

{

newtail.Right = null;

}

Last = newtail;

Count--;

if (Count < 2)

{

First = Last;

}

return val;

}

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

}

/// <summary>

/// Добавляет элемент в начало дека

/// </summary>

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

public void AddFirst(T val)

{

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

if (First == null)

{

First = newbie;

Last = newbie;

Count++;

return;

}

if (First != null)

{

First.Left = newbie;

newbie.Right = First;

First = newbie;

Count++;

}

}

/// <summary>

/// Возвращает значение головного элемента без извлечения

/// </summary>

/// <returns></returns>

public T LookFirst()

{

if (First != null)

{

return First.Value;

}

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

}

/// <summary>

/// Возвращает значение последнего элемента без извлечения

/// </summary>

/// <returns></returns>

public T LookLast()

{

if (Last != null)

{

return Last.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;

}

}

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;

}

}

1. Вывод:

В ходе данной работы были реализованы коллекции для работы с данными Stack и Deque, а также выполнены прилагающиеся к ним задания. Полученные навыки как при реализации структур, так и решении задач, являются необходимыми в работе программиста.