**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 1**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **Списки. Вариант: 15**

|  |  |  |
| --- | --- | --- |
| Студент гр. 0302 |  | Савенко Н.С |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2021

**Постановка задачи и описание решения**

Задачей является реализация односвязного списка с дополнительной функцией swap.

Класс реализован с использованием дженериков. В качестве хранимого типа можно использовать любой значимый тип.

Класс LinkedList<T>

Метод void push\_back(T value)

Проверяем последний элемент, если null, то устанавливаем в первый и последний элемент переданное значение, иначе ставим указатель на переданный элемент в \_next последнего.

Метод void push\_front (T value)

Устанавливаем \_next первого элемента как указатель на переданный элемент. Устанавливаем \_first как переданный элемент, если \_last == null, то устанавливаем \_last как переданный элемент.

Метод void pop\_back ()

Ищем элемент, у которого следующий элемент имеет указатель на следующий равный null. Устанавливаем найденный элемент как \_last У найденного элемента ставим \_next = null. Если он элемент был единственным, то ставим \_first,\_last как null

Метод void pop\_front ()

Устанавливаем первый элемент как следующий элемент первого элемента. Если он null, то дополнительно ставим \_last = null

Метод void insert (T value, int position)

Если position == 0, то делаем push\_front,если position равен длине списка, то делаем push\_back, иначе

Идем до элемента с идексом position – 1, устанавливаем Next переданного элемента как Next найденного элемента как

устанавливаем его \_next как переданный элемент, устанавливаем next найденного как переданный элемент

Метод T at(int position)

Идем до элемента с индексом position.Возвращаем его.

Метод void remove(int position)

Если position == длина – 1, то делаем pop\_back

Если position == 0, то делаем pop\_front()

Иначе идем до элемента индексом position – 1, устанавливаем его Next как Next его Next элемента

Метод int get\_size()

Идем по следующим элементам следующих элементов вперед по списку уведичивая счетчик.

Метод void clear()

Устанавливаем \_first = \_last = null;

Метод void set(int position, T value)

Идем до элемента с индексом position, устанавливаем его Value как value.

Метод bool isEmpty()

Возвращаем \_first == null

Метод void swap(int pos0, int pos1)

Записываем в буффер значение элемента с индексом pos0. Устанавливаем в элемент с индексом pos0 Значение элемента с индексом pos1. Устанавливаем в элемент с индексом pos1 значение буффера.

Метод string ToString()

*По общепринятым стандартам среди разработчиков на платформе .NET для вывода принято делать перегрузку оператора public virtual string? ToString() базового класса Object для всех типов CLR чтобы обеспечить им CLS совместимость.*

В реализации используется щадящий по отношению к пулу интернирования строк экземпляр класса StringBuilder, в него заносятся необходимые значения строк, затем возвращается результирующая строка

**Оценка временной сложности**

Метод void push\_back(T value)

O(1)

Метод void push\_front (T value)

O(1)

Метод void pop\_back ()

O(n)

Метод void pop\_front ()

O(1)

Метод void insert (T value, int position)

O(n)

Метод T at(int position)

O(n)

Метод void remove(int position)

O(n)

Метод int get\_size()

O(n)

Метод void clear()

O(1)

Метод void set(int position, T value)

O(n)

Метод bool isEmpty()

O(1)

Метод void swap(int pos0, int pos1)

O(n)

Метод string ToString()

O(n)

**Описание Unit тестов**

Метод void push\_back(T value)

Добавляем элементы через push\_back и сравниваем сериализованный список с эталонным

Метод void push\_front (T value)

Добавляем элементы через push\_front и сравниваем сериализованный список с эталонным

Метод void pop\_back ()

Добавляем элементы через push\_back и сравниваем сериализованный список с эталонным

Удаляем элементы через pop\_back и сравниваем сериализованный список с эталонным

Метод void pop\_front ()

Добавляем элементы через push\_front и сравниваем сериализованный список с эталонным

Удаляем элементы через pop\_front и сравниваем сериализованный список с эталонным

Метод void insert (T value, int position)

Добавляем элементы через insert и сравниваем сериализованный список с эталонным

Метод T at(int position)

Добавляем элементы через insert и сравниваем элемент полученный через at с эталонным.

Метод void remove(int position)

Добавляем элементы через insert и сравниваем сериализованный список с эталонным.

Удаляем через remove и сравниваем сериализованный список с эталонным.

Метод int get\_size()

Добавляем элементы через push\_front и сравниваем сериализованный список с эталонным. Получаем длину через get\_size и сравниваем с желанной.

Метод void clear()

Добавляем элементы через insert, чистим через clear и сравниваем с эталонным.

Метод void set(int position, T value)

Добавляем элементы через insert и сравниваем сериализованный список с эталонным. Устанавливаем через set и сравниваем с эталонным.

Метод bool isEmpty()

Добавляем элементы через insert и проверяем isEmpty на ложность. Чистим список, проверяем isEmpty на верность.

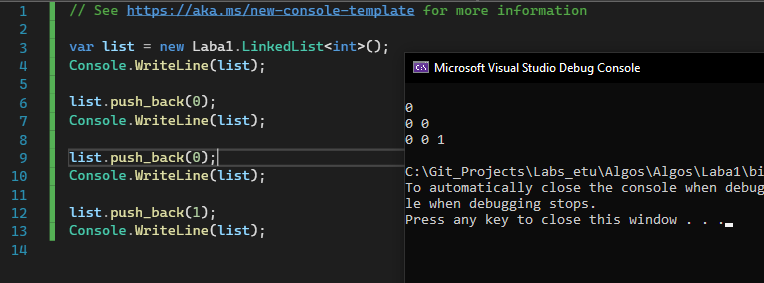
Метод void swap(int pos0, int pos1)

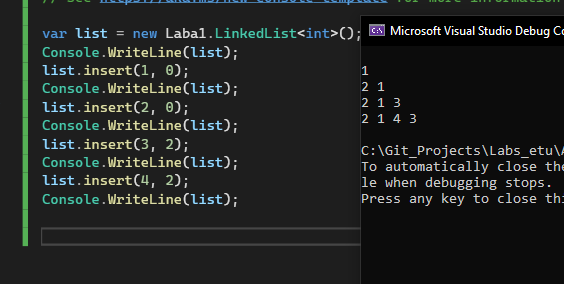
Добавляем элементы через insert и сравниваем сериализованный список с эталонным. Меняем через swap и сравниваем с эталонным.

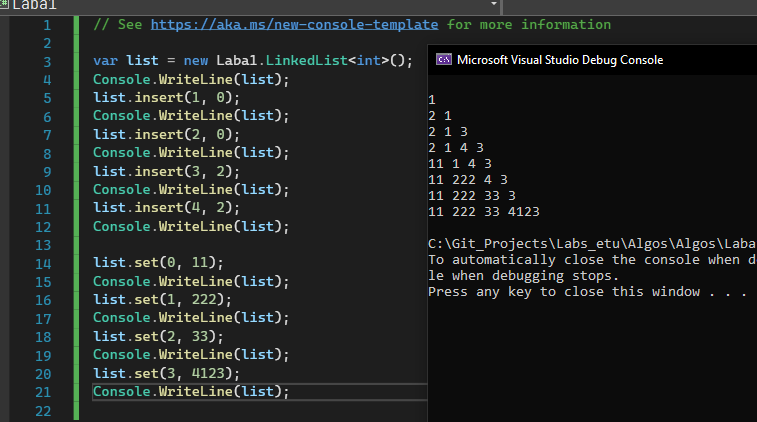
Метод string ToString()

Добавляем элементы через insert и сравниваем сериализованный список с эталонным.

**Примеры работы**







**Листинг**

public class LinkedNode<T>

{

public T Value { get; set; }

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

public LinkedNode(T value)

{

Value = value;

Next = null;

}

public override string ToString()

{

return Value.ToString();

}

}

public class LinkedList<T>

{

private LinkedNode<T>? \_first;

private LinkedNode<T>? \_last;

public int Count { get

{

int count = 0;

var current = \_first;

while (current != null)

{

current = current.Next;

count++;

}

return count;

}

}

public LinkedList() {

\_first = null;

\_last = \_first;

}

// Destructor triggers when GC is making 'magic'

// For details please read CLR via C# by Jeffrey Richter

~LinkedList()

{

while (\_first != null)

{

// it removes roots to elements

// thats gives CG opportunity to safely delete elements with no roots

pop\_front();

}

}

public void push\_back(T value)

{

var newNode = new LinkedNode<T>(value);

if (\_last != null)

{

\_last.Next = newNode;

\_last = newNode;

} else

{

\_first = newNode;

\_last = newNode;

}

}

public void push\_front(T value)

{

var newNode = new LinkedNode<T>(value);

newNode.Next = \_first;

\_first = newNode;

if (\_last == null)

{

\_last = newNode;

}

}

public void pop\_back()

{

var preLast = \_first;

if (preLast != null)

{

if (preLast.Next != null)

{

while (preLast.Next.Next != null)

{

preLast = preLast.Next;

}

preLast.Next = null;

\_last = preLast;

} else

{

\_first = null;

\_last = null;

}

}

else throw new InvalidOperationException();

}

public void pop\_front()

{

\_first = \_first.Next;

if (\_first == null)

{

\_last = null;

}

}

public void insert(T value, int position)

{

var newNode = new LinkedNode<T>(value);

if (position == 0)

{

push\_front(value);

}

else if (position == Count)

{

push\_back(value);

} else

{

var current = \_first;

for (var i = 0; i < position - 1; i++)

{

current = current.Next;

}

newNode.Next = current.Next;

current.Next = newNode;

}

}

public T at(int position)

{

var current = \_first;

for (var i = 0; i < position; i++)

{

current = current.Next;

}

return current.Value;

}

public LinkedNode<T> node\_at(int position)

{

var current = \_first;

for (var i = 0; i < position; i++)

{

current = current.Next;

}

return current;

}

public void remove(int position)

{

if (position == Count - 1)

{

pop\_back();

}

else if(position == 0)

{

pop\_front();

}

else

{

var current = \_first;

for (var i = 0; i < position - 1; i++)

{

current = current.Next;

}

current.Next = current.Next.Next;

}

}

public int get\_size()

{

return Count;

}

public void clear()

{

\_first = null;

\_last = null;

}

public void set(int position, T value)

{

var current = \_first;

for (var i = 0; i < position; i++)

{

current = current.Next;

}

current.Value = value;

}

public bool isEmpty()

{

return \_first == null;

}

public override string ToString()

{

var sb = new StringBuilder();

var current = \_first;

while (current != null)

{

if (current != \_first)

{

sb.Append(" ");

}

sb.Append(current.ToString());

current = current.Next;

}

return sb.ToString();

}

public void swap(int pos0, int pos1)

{

var buffer = node\_at(pos0).Value;

node\_at(pos0).Value = node\_at(pos1).Value;

node\_at(pos1).Value = buffer;

}

}