**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №0**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: Двунаправленный список с внешним итератором

|  |  |  |
| --- | --- | --- |
| Студент гр. 6382 |  | Мартыненко П.П. |
| Преподаватель |  | Филатов А.Ю. |

Санкт-Петербург

2018

**Цель работы.**

Изучить концепции объектно-ориентированного программирования и реализовать двунаправленный список с внешним итератором.

**Основные теоретические положения.**

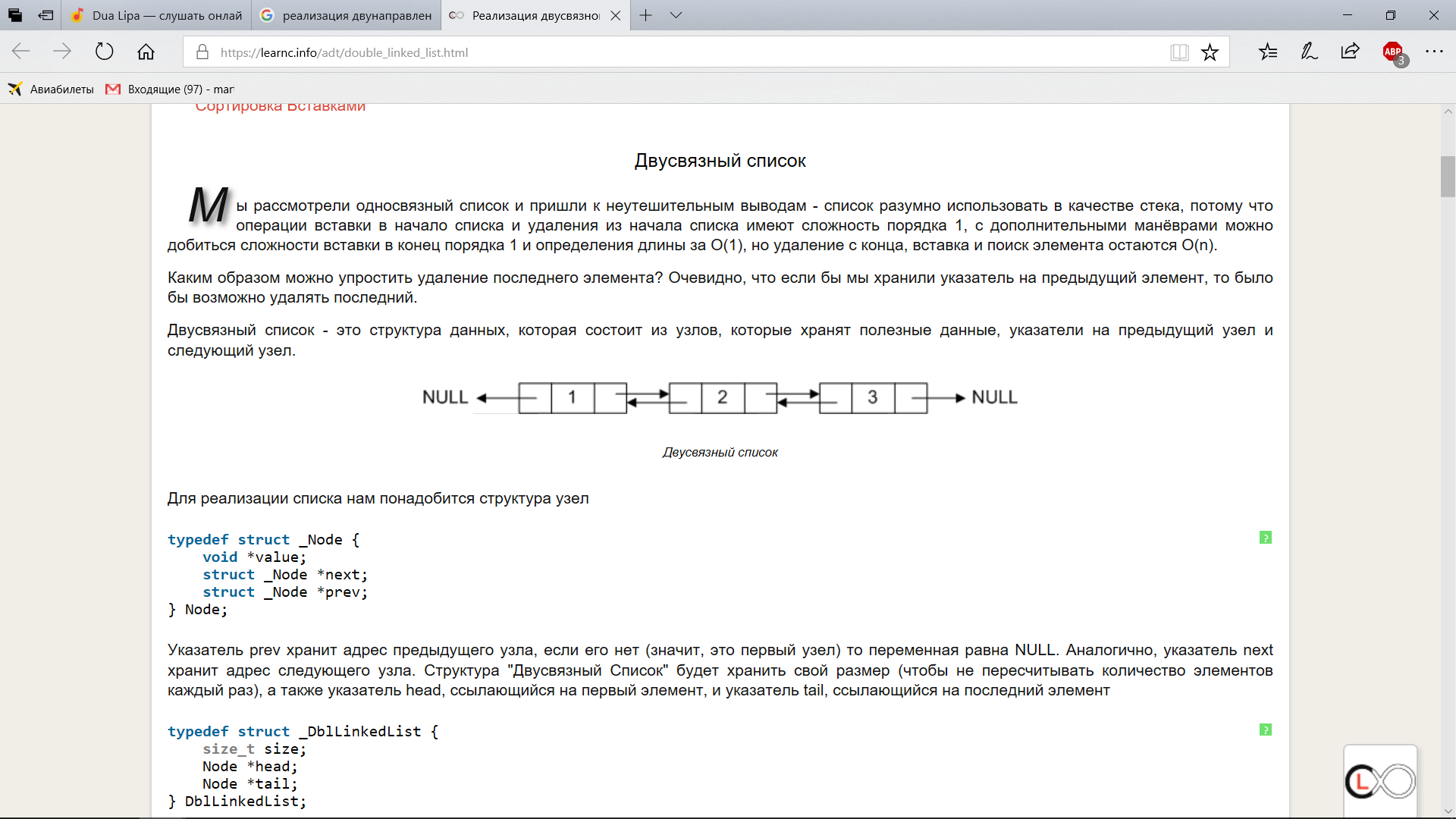
Двусвязный список - это структура данных, которая состоит из узлов, которые хранят полезные данные, указатели на предыдущий узел и следующий узел.

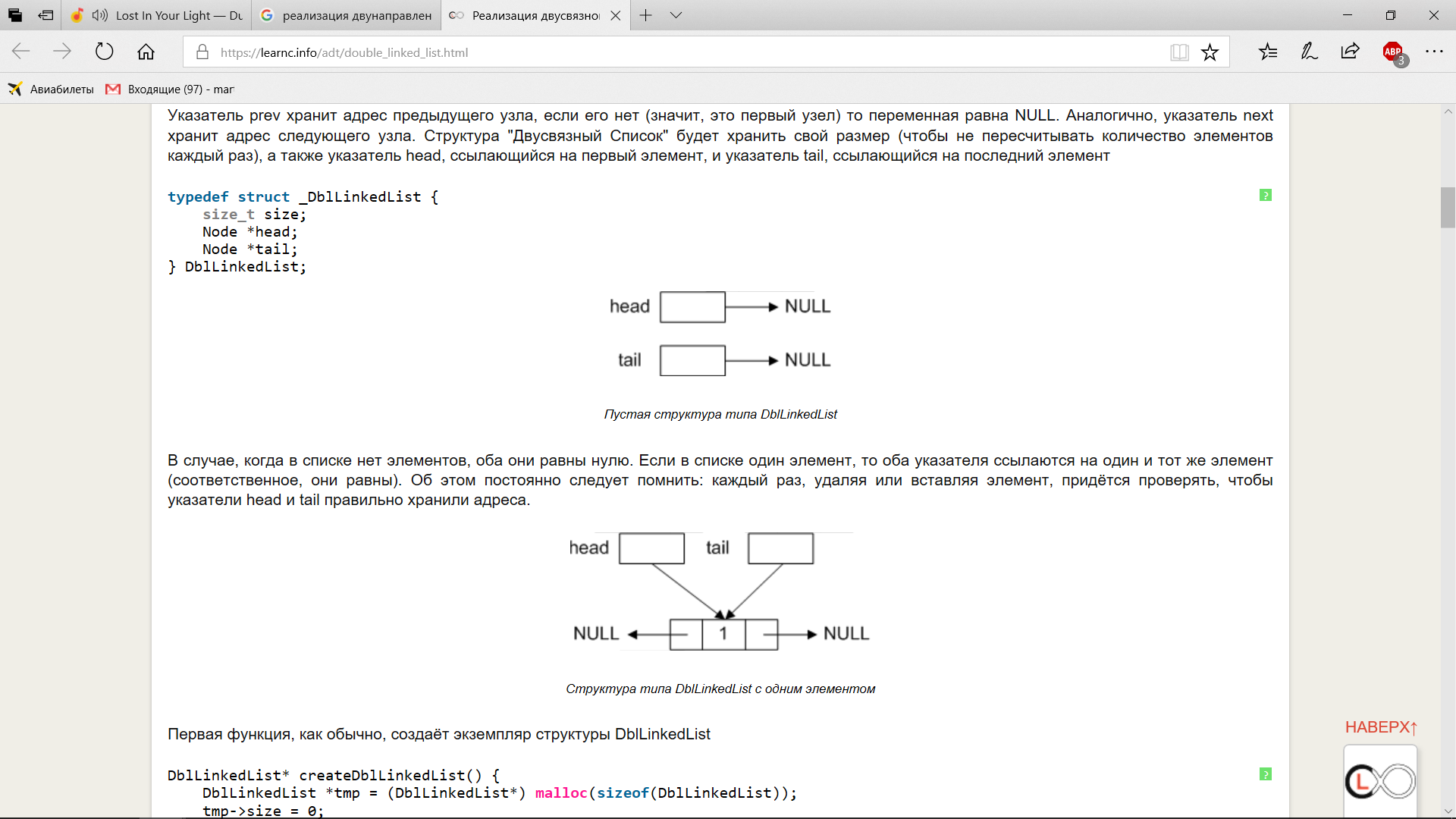
Рис.1.Двусвязный список.

Для реализации списка нам понадобится шаблонная структура-узел

|  |
| --- |
| **template**<**typename** T>  **struct** Node {  **T** \*value;  Node \*next;  Node \*prev;  } |

Указатель prev хранит адрес предыдущего узла, если его нет (значит, это первый узел) то переменная равна NULL. Аналогично, указатель next хранит адрес следующего узла. Структура "Двусвязный Список" будет хранить указатель head, ссылающийся на первый элемент, и указатель tail, ссылающийся на последний элемент.

|  |
| --- |
| **template**<**typename** T>  **struct** List {  **size\_t** size;  Node<T> \*head;  Node<T> \*tail;  } |

Рис.2.Пустая структура типа List.

В случае, когда в списке нет элементов, оба они равны нулю. Если в списке один элемент, то оба указателя ссылаются на один и тот же элемент (соответственное, они равны). Об этом постоянно следует помнить: каждый раз, удаляя или вставляя элемент, придётся проверять, чтобы указатели head и tail правильно хранили адреса.

**Спецификация программы.**

Контейнер – объект, который может содержать набор некоторых объектов. Понятие контейнера обобщает понятие массива в C++. В C++ массиве элементы хранятся в памяти линейно, один за другим. Для контейнера способ хранения не важен (т. е. может быть любым), важен сам факт хранения объектов.

Итератор - объект, обеспечивающий доступ к элементам контейнера. Понятие итератор обобщает понятие указатель. Обобщение заключается в том, что операции, применяемые к указателям для доступа к элементам массива, распространяются на более "широкий" класс объектов - итераторы. Например, применение операции ++ к итератору предполагает его смещение к следующему элементу контейнера, а применение разыменовывания - получение ссылки на элемент. Очевидно, что из-за различия во внутренней организации контейнеров, операции "доступа" не могут быть определены (в отличие от указателей) одним способом для всех контейнеров. В C++ это реализуется за счет определения для каждого класса-контейнера, собственного класса-итератора. Класс-итератор "знает" о способе хранения элементов в "его" классе-контейнере, и требуемые операции перегружаются соответствующим образом.

Отметим еще несколько особенностей программирования контейнеров/итераторов.

Обычно класс-итератор определяется как открытый вложенный класс для класса-контейнера, а не как отдельный класс. Класс-контейнер содержит функции, возвращающие итератор, указывающий на первый и "конечный" элементы. Под "конечным" подразумевается не последний, а некоторый мнимый элемент, "расположенный" сразу после последнего. Итератор на "конечный" элемент нельзя разыменовывать, т. к. этот элемент не существует. Этот итератор имеет логический смысл и используется только в условиях. Например, осуществляется перебор элементов контейнера с использованием операции ++ для смещения итератора к следующему элементу. Если после приращения итератор стал равен итератору на "конечный" элемент, то перебор нужно заканчивать.

Итератор списка (класс Iterator) определен как вложенный класс для List, но является открытым. Итератор хранит адрес узла списка, в котором хранится элемент, на который этот итератор указывает. Для класса Iterator определены следующие операции:

* Обращение к первому элементу списка. Возвращает класс Iterator, указывающий на первый элемент – метод begin().
* Обращение к «последнему» элементу списка. Возвращает класс Iterator, указывающий на нулевой элемент – метод end(). Используется для обозначения конца списка.
* Обращение к последнему элементу списка. Возвращает класс Iterator, указывающий на последний элемент – метод last().
* Разыменовывание. Возвращает ссылку на элемент, хранящийся в узле – **operator \***,
* Инкремент. Переход к следующему элементу (берет из узла и сохраняет адрес следующего узла) – **operator ++**,
* Декремент. Переход к предыдущему элементу (берет из узла и сохраняет адрес предыдущего узла) – **operator --**,
* Равенство и неравенство (простое сравнение указателей) – **operator ==** & **operator !=** ,
* Переход к узлу, на который указывает Iterator– **operator ->.**

Для класса List определены следующие операции:

* Вставка в начало и конец списка, а также вставка по номеру – методы pushFront(…), pushBack(…) и Insert(…) соответственно.
* Удаление из начала и конца списка, а также удаление по номеру – методы popFront(…), popBack(…) и Delete(…) соответственно.
* Показ списка – метод show().

**Выводы.**

В данной лабораторной работе были рассмотрены и изучены основные концепции объектно-ориентированного программирования, такие как дружественные классы, перегрузка операторов, шаблоны классов и т.д. Также был реализован двунаправленный список в рамках ООП.