**Контрольная работа № 2. ООП. Линейные структуры данных. Работа с памятью.**

**Задача: частотный словарь на двусвязном списке, реализованном на динамическом массиве.**

*Указания:*

1. *Решение работы требуется сдать в 1)* ***edufpmi.bsu.by*** *2) систему в* ***iRunner https://acm.bsu.by****/ в виде* ***zip*** *файла с именем вида* ***kr2\_gr1\_ivanov\_ivan.zip*** *(\*красным цветом записаны персональные данные студента), содержащем файл решения (имена указаны в списке ниже), дополнительно желательно предоставить файл/ы \*.txt – файлы исходных данных, которые использовалить при тестировании, а также файлы тестов* *\*\_tests.cpp). Для файлов с тестами используйте постфикс “\_tests”.*
   1. bidirectional\_list\_on\_array.h
   2. file\_reader.h
   3. vocabulary.h
   4. word\_card.h
   5. \*\_tests.cpp
2. *Если Вы сдаете неполное решение, то всё равно необходимо создать функции и методы на все пункты (при этом они могут ничего не делать и возвращать просто любую константу). В противном случае, как и в случае использование неверных сигнатур, при компиляции на сервере произойдет ошибка.*
3. *При реализации алгоритмов не используйте стандартные контейнеры, если это не указано явно, а используйте собственную реализацию контейнеров. Оценка за контрольную будет учитывать качество кода и количество пройденных тестов в системе irunner, а также оценку кода перподавателем.*
4. *При реализации алгоритмов не используйте стандартные контейнеры, если это явно не указано, а используйте собственную реализацию контейнеров.*

**Задание:**

**Разработать объектно-ориентированное приложение приложения для решения задачи:**

Дан текстовый файл *input.txt*. Создать словарь, содержащий карточки слов этого текста. Словарь должен предоставлять функционал по модернизации (уплотнению) списка слов - создания частотного словаря. Словарь должен использовать для хранения списка карточек слов вашу собственную реализацию двусвязного списка, который хранит список указателей на карточку словаря (в карточке: слово и его частота).

**Важно:   
ПРИМЕРЫ КОДА - ЭТО ЗАГОТОВКА КОДА – вы должны уточнить сигнатуру и реализовать методы – КОЕ-ЧТО пропущено, чтобы вы сами это дописали**

1. **(2 балл) Класс WordCard**

using std::string;  
class WordCard {  
 public:  
 WordCard();  
 WordCard(string );  
  
 WordCard(WordCard&);  
 WordCard operator=(WordCard other);  
  
 WordCard(WordCard &&);  
 WordCard &operator=(WordCard &&other);  
  
 ~WordCard();  
  
 string GetWord();  
 size\_t GetCounter();  
  
 void IncCounter();  
  
 bool operator==(WordCard &other);  
 bool operator!=(WordCard &other);  
  
 bool operator<( WordCard &first, WordCard &second);  
  
 private:  
 string word\_;  
 size\_t counter\_;  
};

1. **(2 балл) Класс Vocabulary**

class Vocabulary {  
 public:  
 void read\_file\_to\_cards(std::string file\_path);  
 WordCard \*convert\_string\_to\_word\_card(string str);  
 void make\_frequency\_vocab();  
 void sort\_words(); // можно пузырьком 😊

map<string, size\_t> copy\_to\_map();  
  
 private:  
 BiDirectionalListOnArray<WordCard \*> list\_word\_cards;  
  
};

1. **(1 балл) Класс FileReader**

class FileReader {  
 public:  
 vector<string> ReadLines(const string &file\_path); // чтение строк из файла  
 vector<string> ReadWords(const string &file\_path); // вызов функции чтения строк из файла, и парсинг строк на слова   
};

1. **(5 баллов)** Создать шаблонный класс двусвязного списка BiDirectionalListOnArray <T> . Данный класс должен быть объявлен и определен в пространстве имен containers.

Его поля:  
private:  
 T \*data\_;  
 size\_t size\_ = 0;  
 size\_t capacity\_;  
  
 static const size\_t MIN\_CAPACITY = 10;  
 static const size\_t GROWTH\_FACTOR = 2;

Класс BiDirectionalList<T> должен иметь следующий публичный интерфейс:

* Конструктор по умолчанию.
* Конструктор от std::initializer\_list<T>.
* Конструктор копирования и копирующий оператор присваивания.
* Конструктор перемещения и перемещающий оператор присваивания.
* Методы int Size() и bool IsEmpty().
* Метод ToVector(), возвращающий представление в виде вектора (std::vector).
* Методы T\* Front() и T\* Back(), позволяющие получить первый / последний элемент в списке.
* Методы PushFront(constT& value) и PushBack(constT& value), осуществляющие вставку элемента в начало и конец списка.
* Их аналоги PushFront(T&& value) и PushBack(T&& value).
* Методы PopFront() и PopBack(), осуществляющие удаление первого/последнего элемента из списка.
* Методы InsertBefore(size\_t index, constT& value) и InsertAfter(size\_t index, constT& value), которые позволяют добавить элемент на произвольную позицию в список.
* Их аналоги InsertBefore(size\_t index, T&& value) и InsertAfter(size\_t index, T&& value).
* Метод Erase(size\_t index), который позволяет удалить произвольный элемент из списка.
* Метод int Find(constT& value), который возвращает позицию (индекс) первого вхождения элемента value в список. Если элемент отсутствует в списке, то вернуть -1.
* Метод std::vector<int> FindAll(constT& value), который возвращает индексы всех вхождений элемента value в список в порядке возрастания.
* Оператор обращения по индексу ([]), возвращающий T\*, который позволяет обратиться к элементу списка по его индексу.
* Операторы, обеспечивающие возможность проверки списков на равенство / различие.

Замечания:

* Необходимо создать константные аналоги для методов Back, Front и оператора [], которые будут возвращать const Node\*.
* Необходимо проверять корректность вызовов и передаваемых параметров. Если они ошибочны, то выбрасывать исключение std::excection или его наследников.