**Пример контрольной работы № 2. ООП. Линейные структуры данных. Работа с памятью.**

*Указания:*

1. *Решение каждой из задач следует оформить в виде отдельного файла с именем вида* ***kr2\_gr1b\_v1\_ivanov\_ivan\_z1.cpp. (т.о. Вы будете сдавать несколько файлов \*.h и \*.cpp, \*.txt).***
2. *Оценка за контрольную равняется min(10, round(0,5 + x)), где 0.5 - бонус, x - сумма набранных по задачам баллов (10.5 баллов можно набрать выполнив все пункты контрольной работы).*
3. *Для получения полного балла, решение всех подзадач в задании необходимо оформить в виде класса(классов) с членами-функций. Таким образом, в функции main не должно быть никакой бизнес-логики, только переменные с данными и вызовы методов для решения подзадач.*
4. *При реализации алгоритмов не используйте стандартные контейнеры, а используйте собственную реализацию контейнеров.*

**Задание:**

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

Дан текстовый файл *input.txt*, содержащий текст, состоящий из букв и знаков \*.

*Пример***:**

|  |  |
| --- | --- |
| input.txt | output.txt |
| PR\*\*\*\*IO\*R\*I\*\*T\*Y\*\*QU\*E\*\*\*U\*E\*\* | RPORIITYUQEUE |

**Правило**: “Буква в тексте означает **операцию “вставить”** символ (эту букву) в коллекцию “приоритетная очередь”, а звездочка — **операцию “возвратить”** элемент с наибольшим приоритетом из коллекции, удалив из коллекции.”

Определить последовательность значений, возвращаемых операциями “возвратить” элемент с наибольшим приоритетом, получаемой в результате преобразования исходной последовательности по указанному выше правилу.

**Определение:**  Предположим, что для каждого элемента определен некоторый

приоритет. В простейшем случае значение приоритета может совпадать со значением элемента. В общем случае соотношение элемента и приоритета может быть произвольным. **Приоритетной очередью** (англ. priority queue) называется такая *абстрактная структура данных*, интерфейс которой включает в себя следующие операции:

1) Pull() поиск и удаление (~возвращение) элемента с самым высоким приоритетом;

2) Insert(x; prior(x)) добавление элемента x с указанным приоритетом.

Интерфейс структуры данных “приоритетная очередь” может быть реализован на основе различных структур данных

1. **(1 балл)** Реализовать класс Queue (и NodeQueue), реализующий интерфейс очереди на последовательности связанных компонент. Контруктор(ы), деструктор.
2. **(0.5 балла)** Элемент коллекции (NodeQueue) реализован через внутренний класс.
3. **(2.5 балла)** Реализовать класс PriorityQueue (и NodePriorityQueue), реализующий интерфейс “приоритетной очереди” на последовательности связанных компонент - на основе списка, в котором элементы упорядочены по убыванию приоритета элементов. Конструктор(ы), деструктор.
4. **(0.5 балла)** Элемент коллекции (NodePriorityQueue) реализован через внутренний класс.
5. **(0.5 балла)** Классы Queue, PriorityQueue и NodeQueue, NodePriorityQueue реализованы через шаблоны.
6. **(1 балла)** Функция ReadInput, которая принимает в качестве аргумента имя входного файла, ссылку на коллекцию-очередь (Queue) элементов, значение которых имеет тип char (или Element, см.п.7). Символы читаются из файла, [помещаются в объект класса Element (см.п.7)], и добавляются в очередь.
7. **(0.5 балла)** Если для символа будет создан класс-обертка Element, который в своем поле хранит символ, в NodeQueue поле value имеет тип Element символ будет помещен в объект класса.
8. **(2.5 балл)** Функция Conversion, которая единственный аргумент: ссылку на коллекцию, созданную на основе исходных данных, и возвращает коллекцию - очередь символов-элементов. При реализации использовать коллекцию - “приоритетную очередь”, построенной по правилу, указанному выше.
9. **(0.5 балла)** Функция WriteCollection, которая выводит полученную коллекцию (очередь) символов-элементов (имя файла - параметр).
10. **(0.5 балла)** Проверить корректность вызовов и передаваемых в методы параметров с помощью assertов.