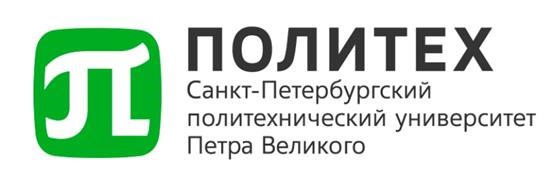
**Санкт-Петербургский политехнический университет Петра Великого**

## Институт компьютерных наук и технологий Высшая школа программной инженерии



# **К У Р С О В А Я Р А Б О Т А**

**Алгоритмы работы со словарями**

## по дисциплине «Алгоритмы и структуры данных»

Выполнил

студент гр.  Османов Тимур

3530904/20002

Руководитель Шемякин Илья Александрович

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2023

Санкт-Петербург

2023 г

Оглавление

[Институт компьютерных наук и технологий Высшая школа программной инженерии 1](#__RefHeading___Toc20126_1601283799)

[К У Р С О В А Я Р А Б О Т А 1](#__RefHeading___Toc20128_1601283799)

[по дисциплине «Алгоритмы и структуры данных» 1](#__RefHeading___Toc20130_1601283799)

[Введение. Общая постановка задачи: 2](#__RefHeading___Toc20132_1601283799)

[Основная часть работы 3](#__RefHeading___Toc20134_1601283799)

[1.Описания алгоритма решения и используемых структур 3](#__RefHeading___Toc33084_1601283799)

[2. Анализ Алгоритма 3](#__RefHeading___Toc20230_1601283799)

[3. Описание спецификации программы 4](#__RefHeading___Toc20232_1601283799)

[Заключение: 7](#__RefHeading___Toc20234_1601283799)

[Список использованных источников 8](#__RefHeading___Toc20236_1601283799)

[Приложение 1. Текст программы. 9](#__RefHeading___Toc20238_1601283799)

# **Введение. Общая постановка задачи:**

**Тема: Алгоритмы работы со словарями**

1. Для разрабатываемого словаря реализовать основные операции:

* + INSERT (ключ) – добавить запись с указанным ключом и значением 1. Если запись уже существовала, то увеличить значение на 1.
  + FIND (ключ)- найти запись с соответствующим ключом и вернуть ее значение.
  + ERASE (ключ)- удалить запись с указанным ключом

1. Предусмотреть обработку и инициализацию исключительных ситуаций.
2. Программа должна быть написана в соответствии со стилем программирования: C++ Programming Style Guidelines [(http://geosoft.no/development/cppstyle.html)](http://geosoft.no/development/cppstyle.html).
3. Тесты должны учитывать, как допустимые, так и не допустимые последовательности входных данных.

*1.3.4 Частотный словарь. Хеш-таблица.*

Разработать и реализовать алгоритм формирования частотного словаря:

* определить понятие слово
* прочитать текст и сформировать набор слов вместе с информацией о частоте их встречаемости
* определить три чаще всего встречающихся слова, для реализации задания использовать Хеш-таблицу***.*** Узел хеш-таблицы должен содержать:

- Пару, состоящую из КЛЮЧ + ЗНАЧЕНИЕ

- Хеш-код, полученный из ключа при помощи хеш-функции.

# **Основная часть работы**

## 1.Описания алгоритма решения и используемых структур

Для решения поставленной задачи мне потребовалось использовать две структуры данных — хеш-таблица и односвязный список, а так же адаптер частотный словарь.

* **Односвязный список** (forward\_list):

- это структура данных, состоящая из узлов каждый из которых хранит в себе некое значение и указатель на следующий узел. Первый узел в словаре (head\_) по сути является фиктивным, он не хранит никакого значения, а только ссылается на следующий узел.

Рисунок 1: Односвязный список

Односвязный список поддерживает несколько методов для получения итераторов: before\_begin(), begin() и end():

* before\_begin() - возвращает итератор на узел head\_.
* begin() - возвращает итератор на узел, следующий за head\_.
* end() - возвращает итератор на не существующий узел, находящийся в конце списка.

Аналогично устроенны константные итераторы cbefore\_begin(), cbegin() и cend().

Доступ к этим итераторам нужен нам для реализации метода вставки (insert\_after) и удаления (erase\_after) узла по итератору на предыдущий узел. Асимптотическая сложность данных методов — O(1).

* **Хеш-таблица** (hash\_table)

- это ассоциативный контейнер, поддерживающий методы вставки, удаления и поиска элемента за константное время. Хеш-таблица — одна из самых часто приминающихся структур данных (порядка 90% от всех структур), поскольку она позволяет программистам оптимизировать свой код. Хеш-таблица представляет из себя некий массив, хранящий в своих ячейках элементы с соответствующим им хеш-кодом. Однако есть весьма большая вероятность, что для разных данных хеш-код может совпасть, такие случаи называются коллизиями. Существует два вида их решений:

1. Прямая адресация

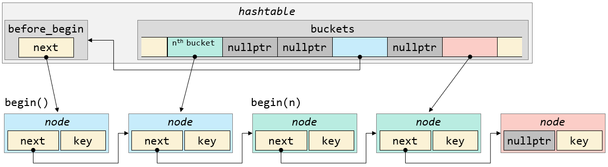
При прямой адресации элемент с совпадающим хеш-кодом вставляется в другую свободную ячейку. Для получения этого индекса существуют разные подходы. Самый простой из которых — линейное пробирование. При нем мы поочередно движемся по таблице, ища следующую свободную ячейку.

1. Метод цепочек

Используя метода цепочек, мы не храним элементы непосредственно в массиве. Каждая ячейка массива является указателем на список, и при совпадении хеш-кодов мы просто добавляем новый элемент в список, соответствующий полученному индексу ячейки.

Хеш-таблицы с открытой адресацией оказываются наиболее производительными, поскольку они отлично поддерживают кэширование. Однако в STL реализована хеш-таблица, использующая метод цепочек. Это так, поскольку разработчики поставили во главу удобство в использовании. Хеш-табица, использующая метод цепочек, лучше приспособлена к работе с итераторами, более того она поддерживает итератор по определенной ячейке.

Я решил использовать реализацию методом цепочек.

Рисунок 2: Хеш-таблица

Таким образом, моя хеш-таблица имеет следующие поля:

* Односвязный список (forward\_list) – elems, для хранения значений и их хеш-кодов
* Динамический массив — arr, хранящий в своих ячейках итераторы на определенные узлы односвязного списка, изначально все итераторы указывают на elems.end()
* Количество элементов в массиве — size\_
* Количество ячеек в массиве — bucket\_count\_
* Максимальный фактор загруженности таблицы — max\_load\_factor, в случае его преодоления таблица должна автоматически перестроится.

Стоит уточнить, что определенная ячейка arr хранит в себе итератор не на первый узел с хеш-кодом, соответствующим её индексу, а на узел, стоящий перед этим элементом.

Методы для получения итераторов в хеш-таблице, а именно begin()/cbegin() и end()/cend() работают аналогично таким же методам для односвязного списка, то есть begin() возвращает elems.begin() и так далее.

Метод вставки insert реализован следующим образом. Он принимает как аргумент пару ключ + значение. Для ключа вычисляется хеш-код, после чего находится его остаток от деление на количество ячеек в массиве bucket\_count\_. Полученное значение используется как индекс в массиве arr, и если итератор в этой ячейке показывает на this→end(), то элемент при помощи elems.insert\_after(elems.before\_begin(), value) вставляется в начало списка; соответствующей ячейке arr присваивается elems.before\_begin() и делается так, чтобы ячейка, которая ссылалась на elems.before\_begin() ранее, теперь указывала на итератор добавленного элемента.

Если же ячейка не показывала на this→end(), то при помощи elmes.insert\_after(), элемент просто вставлялся после итератора на который ссылалась эта ячейка.

После вставки элемента, в случае если отношение между size\_ и bucket\_count(load\_factor) превзошло max\_load\_factor, производится принудительный rehash.

В случае ошибки при выполнении вставки, исключение не генерируется, функция возвращает false, а объект сохраняет свое состояние.

Метод поиска find() принимает как аргумент определенный ключ. Для данного ключа вычисляется хеш-код и находится его остаток от деления на bucket\_count\_. Затем проверяем существует ли в arr ячейка с индексом, соответствующим этому значению. Если не существует, то функция возвращает this->end(), в другом же случает мы движемся по списку elems, пока на найдем нужное нам значение, возвращая итератор на него, или значение у которого отличается остаток от деления хеш-кода на bucket\_count\_, возвращая this→end().

В случае возникновения исключения он подбрасывается дальше, объект сохраняет свое состояние.

Метод удаления erase имеет две перегрузки одна из которых принимает итератор на удаляемый элемент, вторая ключ. Первая перегрузка удалят узел, на который ссылается данный итератор, затем перевязывает список и в случае необходимости, правильно изменяет итераторы в ячейках, требующих этого, и возвращает итератор на элемент ,предшествующий удаляемому. Вторая перегрузка при помощи метода find() находит итератор на удаляемый элемент и вызывает первую перегрузку. В случает удачного удаления она возвращает 1, в противном случае 0.

Исключение в методе erase может генерироваться только при сравнивании элементов и вызове хеш-функции. В данных случаях объект сохраняет свое состояние.

Метод rehash(N) увеличивает количество ячеек в хеш-таблице до N, если оно превышает их предыдущее количество. Он создает временную таблицу размера, в которую заново вставляет все элемент при помощи insert, предварительно пересчитав остатки от деления хеш-кодов элементов на новое количество ячеек.

В случае исключения оно подбрасывается дальше. Исходный объект не меняет свое состояние.

Также реализован метод max\_load\_factor, который поваляет на увеличить max\_load\_factor\_, что приводит к увеличению коллизий.

Данный метод не генерирует исключения.

* **Частотный словарь** (dictionary\_map)

- это адаптер, который хранит в себе ключ + значение. Ключом частотного словаря является слово, как слово я выделил любой набор символов, разделенный между собой пробелами. Значением частотного словаря является количество повторений конкретного слова в тексте.

Как поле частотный словарь хранит хеш-таблицу table.

Метод вставки insert в частотный словарь принимает как аргумент ключ. Сначала в хеш-таблцие ищется узел с данным ключом, при помощи метода table.find(), если ключ найден, то значение в узле увеличивается на единицу. Если же ключ не найдет, то при помощи table.insert(), в таблицу происходит вставка узла с соответствующим колючем и значением 1.

Метод поиска find в частотном словаре вызывает table.find() и в случае если узел найден, метод возвращает его значение.

Метод удаления erase в частотном словаре вызывает table.erase() и ничего не возвращает;

Методы this→begin() и this→end(), возвращают table.begin() и table.end().

Метод поиска трех самых часто встречающихся слов принимает по указателю массив, состоящий из std::pair<std::string, size\_t>. Сначала он проверяет есть ли в таблице три элемент, если их нет то вставляет в массив все имеющиеся. В противном случае, вставляет в него три первых элемента и сортирует их по убыванию, затем идет дальше по элементам хеш-таблице и если видит подходящий, добавляет его в таблицу вместо наименьшего и выполняет сортировку.

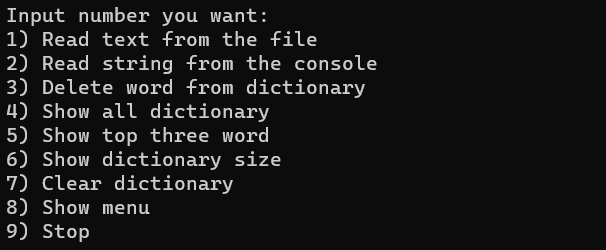
Исключения для каждого метода такие же как и исключения хеш-таблицы.

## 2. Анализ Алгоритма

* Метод поиска слова в словаре — в среднем O(1), так как нам достаточно посчитать хеш-код и пройти по одному бакету, для того чтобы найти элемент.
* Метод вставки - O(1), так для вставки мы должны выполнить поиск, что займет в среднем O(1), если ключ есть в словаре, то увеличить значение на единицу, в противном случае выполнить вставку в хеш-таблицу, что тоже займет O(1), поскольку вставка в односвязный список по итератору занимает O(1)
* Метод begin(), возвращающий итератор на начало дерева - O(1), так как итератор совпадает с итератором на элемент односвязного списка, стоящий после head\_.
* Метод end() – O(1), так как создает итератор на основе nullptr.
* Алгоритм вставки слова в словарь – O(1), если при вставке не происходит rehash хеш-таблицы, в данном случае вставка в среднем случае работает за O(N) в худшем за O(N2).
* Алгоритм поиска трех самый встречающихся слов – O (n), так как перебирает все элементы в хеш-таблице.
* Метод удаления – O(1), поиск элемента в хеш-таблице занимает в среднем O(1), как и удаление по итератору.
* Объем затрачиваемой памяти O(N+M), так как хранятся узлы списка + динамический массив. В случае rehash и вызове конструктора копирования вделается дополнительная память под временный массив.

## 3. Описание спецификации программы

Работа программы начинается с меню команд, доступных пользователю, и ожидания ввода номера команды. При неправильном номере команды высветится сообщение «Wrong command».

Рисунок 3: Пользовательский интерфейс

*Команда №1*: Read text from the file.

При вызове этой команды из прикрепленного файла считывается весь текст и добавляется в словарь.

|  |  |  |
| --- | --- | --- |
| Требование | Данные | Ожидаемый результат |
| 1. На вход подается набор различных символов из файла | (1)asd fasdf asdf asdf  asdf asd f--- asdf asldfj  9asd+ | (asdf : 4) (asd : 2)  (fasdf : 1) (f--- : 1)  (9asd+ : 1) (asldfj : 1) |

*Команда №2:* Read string from the console.

При вызове данной команды пользователю становится доступен ввод в консоль, и он должен ввести строку, которая будет разделена на слова и добавлена в словарь.

|  |  |  |
| --- | --- | --- |
| Требование | Данные | Ожидаемый результат |
| 1. На вход подается набор различных символов из консоли | (2)a a a a b b b b - - - c dda asd d | (d : 1) (asd : 1) (- : 3) (a : 4) (dda : 1) (b : 4) (c : 1) |

Команда №3: Delete word from dictionary.

При вызове данной команды нужно ввести в консоль слово, которое будет удалено из словаря.

|  |  |  |
| --- | --- | --- |
| Исходные данные | Ввод пользователя | Ожидаемый результат |
| (d : 1) (asd : 1) (- : 3) (a : 4) (dda : 1) (b : 4) (c : 1) | (3) - | (d : 1) (asd : 1) (a : 4) (dda : 1) (b : 4) (c : 1) |

|  |  |  |
| --- | --- | --- |
| Требование | Данные | Ожидаемый результат |
| 1. Слово должно существовать в словаре | (3) abasdasdbasdbd | Word is missing! |

Команда №4: Show top three word.

Выводит весь на консоль все элементы словаря. В случае если словарь пуст переносит строку.

|  |  |  |
| --- | --- | --- |
| Требование | Данные | Ожидаемый результат |
| 1. В словаре есть элементы | (4) | (d : 1) (asd : 1) (- : 3) (a : 4) |
| 2. Словарь пуст | (4) |  |

Команда №5: Show all dictionary.

Выводит весь на консоль 3 самых популярных слова в словаре. В случае если в словаре меньше 3 слов, выводит все элементы. Если словарь пуст переносит строку

|  |  |  |
| --- | --- | --- |
| Требование | Данные | Ожидаемый результат |
| 1. В словаре больше 3-х элементов | (5) | (my : 10) (I : 12) (and : 12) |
| 2. В словаре 2 элемента | (5) | (b : 1) (a : 1) |
| 3. Словарь пуст | (5) |  |

Команда №6: Show dictionary size.

Выводит на консоль размер словаря.

|  |  |  |
| --- | --- | --- |
| Исходные данные | Ввод пользователя | Ожидаемый результат |
| (d : 1) (asd : 1) (- : 3) (a : 4) (dda : 1) (b : 4) (c : 1) | (6) | 7 |

Команда №7: Clear dictionary.

Отчищает словарь. При этом сохраняется количество ячеек в словаре, а размер становится равен 0.

|  |  |  |
| --- | --- | --- |
| Исходные данные | Ввод пользователя | Ожидаемый результат |
| (d : 1) (asd : 1) (- : 3) (a : 4) (dda : 1) (b : 4) (c : 1) | (7) |  |

Команда №8: Show menu.

Выводит на консоль меню с командами.

Команда №9: Stop.

Завершает работу команды.

# **Заключение:**

В ходе этой работы удалось реализовать структуру данных – хеш-таблицу, оформление и способ реализации которой схож с STL. Возникла необходимость написать дополнительную структуру данных, а именно односвязный список и итераторы для него. Был выбран именно односвязный список, поскольку он не хранит дополнительного поля tail, в следствии чего на узел требуется меньше памяти. Итераторы же помогли организовать эффективный доступ к определенному элементу хеш-таблицы. Частотный словарь в своих методах эффективно использует методы хеш-таблицы. Критерии слова были выбраны не достаточно эффективные, поскольку они делают словарь менее универсальным, однако реализация программы стала гораздо проще.

Подводя итоги работы, стоит упомянуть, что хеш-таблица была и остается самый востребованной базой данной в сфере промышленной разработки, поэтому нужно знать ее структуру и уметь с ней работать. В ходе работы получилось углубить знания языка С++ и стандартной библиотеки STL.

# **Список использованных источников**

1. Исходный код std::unordered\_map
2. Исходный код std::forward\_list
3. https://github.com/gcc-mirror/gcc/blob/master/libstdc%2B%2B-v3/include/bits/unordered\_map.h
4. <https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2543.htm>
5. <https://www.youtube.com/watch?v=uWoj4SV_V-Q&t=2154s>
6. https://www.youtube.com/watch?v=2YFglchMJmA&t=2717s
7. <https://www.youtube.com/watch?v=XamV5-pTNc8&t=3161s>
8. https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D0%B0
9. https://stackoverflow.com/questions/29267493/before-begin-implementation-of-forward-list
10. https://en.cppreference.com/w/cpp/container/unordered\_map
11. https://cplusplus.com/reference/unordered\_map/unordered\_map/
12. https://en.cppreference.com/w/cpp/container/forward\_list
13. https://www.youtube.com/watch?v=ncHmEUmJZf4

# **Приложение 1. Текст программы.**

forward\_list.h

|  |
| --- |
| #ifndef LISH\_H  #define LIST\_H  #include <cassert>  #include <iostream>  struct ListNodeBase {  ListNodeBase(ListNodeBase\* \_Lptr = nullptr) :  next(\_Lptr)  {}  ListNodeBase\* next;  virtual ~ListNodeBase() = default;  };  template<class T>  struct ListNode : public ListNodeBase {  ListNode(const T& value, ListNodeBase\* \_Lptr = nullptr) :  ListNodeBase(\_Lptr),  data(value)  {}  T data;  };  template<class T>  class ConstIterator {  public:  using iterator\_category = std::forward\_iterator\_tag;  using \_Nodeptr = ListNodeBase\*;  using value\_type = T;  using difference\_type = ptrdiff\_t;  using pointer = const T\*;  using reference = const T&;  ConstIterator(\_Nodeptr \_Pnode = nullptr) : ptr\_(\_Pnode) {}  bool operator==(const ConstIterator& right) const {  return ptr\_ == right.ptr\_;  }  bool operator!=(const ConstIterator& right) const {  return !(\*this == right);  }  reference operator\*() const {  assert(ptr\_ != nullptr);  return static\_cast<ListNode<value\_type>\*>(ptr\_)->data;  }  pointer operator->() const {  assert(ptr\_ != nullptr);  return &(static\_cast<ListNode<value\_type>\*>(ptr\_)->data);  }  ConstIterator& operator++() {  assert(ptr\_ != nullptr);  ptr\_ = ptr\_->next;  return \*this;  }  ConstIterator operator++(int) {  ConstIterator temp = \*this;  ptr\_ = ptr\_->next;  return temp;  }  \_Nodeptr ptr\_;  };  template<class T>  class Iterator : public ConstIterator<T> {  public:  using \_Mybase = ConstIterator<T>;  using iterator\_category = std::forward\_iterator\_tag;  using \_Nodeptr = ListNodeBase\*;  using value\_type = T;  using difference\_type = ptrdiff\_t;  using pointer = T\*;  using reference = T&;  Iterator(\_Nodeptr \_Pnode = nullptr) : \_Mybase(\_Pnode) {}  reference operator\*() const {  return const\_cast<reference>(\_Mybase::operator\*());  }  pointer operator->() const {  return const\_cast<pointer>(\_Mybase::operator->());  }  Iterator& operator++() {  \_Mybase::operator++();  return \*this;  }  Iterator operator++(int) {  Iterator temp = \*this;  \_Mybase::operator++();  return temp;  }  };  template <class T>  class ForwardList {  public:  using value\_type = T;  using size\_type = size\_t;  using difference\_type = ptrdiff\_t;  using reference = value\_type&;  using const\_reference = const value\_type&;  using pointer = value\_type\*;  using const\_pointer = const value\_type\*;  using iterator = Iterator<T>;  using const\_iterator = ConstIterator<T>;  ForwardList();  ForwardList(const ForwardList& copy);  ForwardList(ForwardList&& move) noexcept;  ~ForwardList();  ForwardList& operator=(const ForwardList& copy);  ForwardList& operator=(ForwardList&& move) noexcept;  iterator before\_begin() { return iterator(&head\_); }  const\_iterator cbefore\_begin() { return const\_iterator(&head\_); }  iterator begin() { return iterator(head\_.next); }  iterator end() { return iterator(); }  const\_iterator cbegin() { return const\_iterator(head\_.next); }  const\_iterator cend() { return const\_iterator(); }  iterator insert\_after(iterator pos, const T& value);  iterator erase\_after(const\_iterator pos);  void swap(ForwardList& other) noexcept;  bool empty() const noexcept;  private:  ListNodeBase head\_;  };  template<class T>  inline ForwardList<T>::ForwardList() :  head\_(nullptr)  {}  template<class T>  inline ForwardList<T>::ForwardList(const ForwardList& copy) :  head\_(nullptr)  {  const ListNodeBase\* from = &copy.head\_;  ListNodeBase\* to = &this->head\_;  try {  while (from->next) {  const ListNode<T>\* nextf = static\_cast<ListNode<T>\*>(from->next);  to->next = new ListNode<T>(\*nextf);  from = from->next;  to = to->next;  }  }  catch (...) {  this->clear();  }  }  template<class T>  inline ForwardList<T>::ForwardList(ForwardList&& move) noexcept :  ForwardList<T>()  {  this->swap(move);  }  template<class T>  inline ForwardList<T>::~ForwardList() {  ListNodeBase\* p = head\_.next;  while (p) {  ListNode<T>\* temp = static\_cast<ListNode<T>\*>(p);  p = p->next;  delete temp;  }  head\_ = nullptr;  }  template<class T>  inline ForwardList<T>& ForwardList<T>::operator=(const ForwardList& copy) {  ForwardList<T>\* temp = new ForwardList<T>(copy);  this->swap(temp);  return \*this;  }  template<class T>  inline ForwardList<T>& ForwardList<T>::operator=(ForwardList&& move) noexcept{  this->swap(move);  return \*this;  }  template<class T>  inline Iterator<T> ForwardList<T>::insert\_after(iterator pos, const T& value) {  ListNode<T>\* p = new ListNode<T>(value);  try {  p->next = pos.ptr\_->next;  pos.ptr\_->next = p;  }  catch (...) {  delete p;  throw;  }  return iterator(p);  }  template<class T>  inline Iterator<T> ForwardList<T>::erase\_after(const\_iterator pos) {  ListNode<T>\* p = static\_cast<ListNode<T>\*>(pos.ptr\_->next);  pos.ptr\_->next = p->next;  delete p;  p = nullptr;  return iterator(pos.ptr\_->next);  }  template<class T>  inline void ForwardList<T>::swap(ForwardList& other) noexcept {  std::swap(head\_, other.head\_);  }  template<class T>  inline bool ForwardList<T>::empty() const noexcept {  return head\_.next == nullptr;  }  #endif |

hash\_table.h

|  |
| --- |
| #ifndef HASH\_TABLE  #define HASH\_TABLE  #include "forward\_list.h"  #include <iostream>  template <class T>  struct HashNode {  size\_t cache;  T data;  };  template <class Key,  class T,  class Hash = std::hash<Key>>  class HashTable {  public:  using value\_type = std::pair<const Key, T>;  using \_Nodeptr = HashNode<value\_type>;  using key\_type = Key;  using mapped\_type = T;  using hasher = Hash;  using size\_type = size\_t;  using difference\_type = ptrdiff\_t;  using reference = value\_type&;  using const\_reference = const value\_type&;  using pointer = value\_type\*;  using const\_pointer = const value\_type\*;  using iterator = Iterator<\_Nodeptr>;  using const\_iterator = ConstIterator<\_Nodeptr>;  explicit HashTable(size\_type count = 1);  HashTable(const HashTable& copy);  HashTable(HashTable&& move);  ~HashTable();  HashTable& operator=(const HashTable& copy);  HashTable& operator=(HashTable&& move) noexcept;  iterator begin() { return elems->begin(); }  iterator end() { return elems->end(); }  const\_iterator cbegin() const { return elems->cbegin(); }  const\_iterator cend() const { return elems->cend(); }  void rehash(size\_type n);  std::pair<iterator, bool> insert(const value\_type& value);  iterator erase(const\_iterator position);  size\_type erase(const key\_type& k);  iterator find(const key\_type& key);  const\_iterator find(const key\_type& key) const;  void swap(HashTable& ump) noexcept;  void clear();  size\_type size() const noexcept;  size\_type bucket\_count() const noexcept;  float load\_factor() const noexcept;  float max\_load\_factor() const noexcept;  void max\_load\_factor(float ml);  private:  ForwardList<\_Nodeptr>\* elems;  Iterator<\_Nodeptr>\* arr;  size\_type size\_;  size\_type bucket\_count\_;  float max\_load\_factor\_;  std::pair<iterator, bool> insert(\_Nodeptr node);  };  template<class Key, class T, class Hash>  inline HashTable<Key, T, Hash>::HashTable(size\_type count) :  elems(new ForwardList<\_Nodeptr>),  arr(new Iterator<\_Nodeptr>[count]),  size\_(0),  bucket\_count\_(count),  max\_load\_factor\_(1.0)  {}  template<class Key, class T, class Hash>  inline HashTable<Key, T, Hash>::HashTable(const HashTable& copy) :  HashTable(copy.bucket\_count\_)  {  try {  auto listNullIter = copy.elems->end();  for (auto i = copy.elems->begin(); i != listNullIter; ++i) {  size\_type hashCode = i->cache;  this->insert(\*i);  }  }  catch (...) {  delete this;  }  }  template<class Key, class T, class Hash>  inline HashTable<Key, T, Hash>::HashTable(HashTable&& move) :  HashTable()  {  this->swap(move);  }  template<class Key, class T, class Hash>  inline HashTable<Key, T, Hash>::~HashTable() {  delete elems;  delete[] arr;  arr = nullptr;  elems = nullptr;  }  template<class Key, class T, class Hash>  inline HashTable<Key, T, Hash>& HashTable<Key, T, Hash>::operator=(const HashTable& copy) {  HashTable temp(copy);  this->swap(temp);  return \*this;  }  template<class Key, class T, class Hash>  inline HashTable<Key, T, Hash>& HashTable<Key, T, Hash>::operator=(HashTable&& move) noexcept {  this->swap(move);  return \*this;  }  template<class Key, class T, class Hash>  inline void HashTable<Key, T, Hash>::rehash(size\_type n) {  if (n < bucket\_count\_) {  return;  }  HashTable tempHash(n);  auto listNullIter = elems->end();  for (auto i = elems->begin(); i != listNullIter; ++i) {  size\_type hashCode = i->cache;  tempHash.insert(\*i);  }  this->swap(tempHash);  }  template<class Key, class T, class Hash>  inline std::pair<Iterator<HashNode<std::pair<const Key, T>>>, bool> HashTable<Key, T, Hash>::insert(const value\_type& value) {  try {  size\_type hashCode = hasher{} (value.first);  if (!arr[hashCode % bucket\_count\_].ptr\_) {  iterator bktIter = elems->insert\_after(elems->before\_begin(), \_Nodeptr{ hashCode, value });  arr[hashCode % bucket\_count\_] = elems->before\_begin();  if (size\_ > 0) {  auto buff = bktIter++;  size\_t cacheMod = bktIter->cache % bucket\_count\_;  arr[cacheMod] = buff;  }  }  else {  elems->insert\_after(arr[hashCode % bucket\_count\_], \_Nodeptr{ hashCode, value });  }  try {  ++size\_;  if (load\_factor() > max\_load\_factor\_) {  this->rehash(bucket\_count\_ \* 2);  }  }  catch (...) {  --size\_;  erase(++arr[hashCode % bucket\_count\_]);  return std::pair<iterator, bool>(elems->end(), false);  }  return std::pair<iterator, bool>(arr[hashCode % bucket\_count\_], true);  }  catch (...) {  return std::pair<iterator, bool>(elems->end(), false);  }  }  template<class Key, class T, class Hash>  inline Iterator<HashNode<std::pair<const Key, T>>> HashTable<Key, T, Hash>::erase(const\_iterator position) {  if (!position.ptr\_) {  return elems->end();  }  auto deleter = position++;  size\_t hashCode = deleter->cache % bucket\_count\_;  auto checkPrev = arr[hashCode];  if (!position.ptr\_ && checkPrev.ptr\_->next == deleter.ptr\_) {  delete deleter.ptr\_;  checkPrev.ptr\_->next = nullptr;  arr[hashCode] = elems->end();  --size\_;  return checkPrev;  }  else if (position.ptr\_ && (position->cache % bucket\_count\_ != hashCode)) {  if (checkPrev.ptr\_->next == deleter.ptr\_) {  arr[hashCode] = elems->end();  checkPrev.ptr\_->next = deleter.ptr\_->next;  arr[position->cache % bucket\_count\_] = checkPrev;  }  else {  arr[hashCode] = checkPrev;  checkPrev.ptr\_->next = deleter.ptr\_->next;  }  }  else {  while (checkPrev.ptr\_->next != deleter.ptr\_) {  ++checkPrev;  }  checkPrev.ptr\_->next = deleter.ptr\_->next;  }  delete deleter.ptr\_;  deleter.ptr\_ = nullptr;  --size\_;  return checkPrev;  }  template<class Key, class T, class Hash>  inline size\_t HashTable<Key, T, Hash>::erase(const key\_type& k) {  auto buff = this->find(k);  if (!buff.ptr\_ || this->erase(buff) == elems->end()) {  return 0;  }  return 1;  }  template<class Key, class T, class Hash>  inline Iterator<HashNode<std::pair<const Key, T>>> HashTable<Key, T, Hash>::find(const key\_type& key) {  size\_type modHashCode = hasher{} (key) % bucket\_count\_;  iterator iter = arr[modHashCode];  if (iter.ptr\_) {  ++iter;  while (iter.ptr\_ && (iter->cache % bucket\_count\_ == modHashCode)) {  if (iter->data.first == key) {  return iter;  }  ++iter;  }  }  return elems->end();  }  template<class Key, class T, class Hash>  inline ConstIterator<HashNode<std::pair<const Key, T>>> HashTable<Key, T, Hash>::find(const key\_type& key) const {  size\_type modHashCode = hasher{} (key) % bucket\_count\_;  const\_iterator iter = arr[modHashCode];  if (iter.ptr\_) {  ++iter;  while (iter.ptr\_ && (iter->cache % bucket\_count\_ == modHashCode)) {  if (iter->data.first == key) {  return iter;  }  ++iter;  }  }  return elems->cend();  }  template<class Key, class T, class Hash>  inline void HashTable<Key, T, Hash>::clear() {  while (size\_ != 0) {  erase(begin());  }  }  template<class Key, class T, class Hash>  inline void HashTable<Key, T, Hash>::swap(HashTable& ump) noexcept {  std::swap(elems, ump.elems);  std::swap(arr, ump.arr);  std::swap(size\_, ump.size\_);  std::swap(bucket\_count\_, ump.bucket\_count\_);  std::swap(max\_load\_factor\_, ump.max\_load\_factor\_);  }  template<class Key, class T, class Hash>  inline size\_t HashTable<Key, T, Hash>::size() const noexcept {  return size\_;  }  template<class Key, class T, class Hash>  inline size\_t HashTable<Key, T, Hash>::bucket\_count() const noexcept {  return bucket\_count\_;  }  template<class Key, class T, class Hash>  inline float HashTable<Key, T, Hash>::load\_factor() const noexcept {  return static\_cast<float>(size\_) / static\_cast<float>(bucket\_count\_);  }  template<class Key, class T, class Hash>  inline float HashTable<Key, T, Hash>::max\_load\_factor() const noexcept {  return max\_load\_factor\_;  }  template<class Key, class T, class Hash>  inline void HashTable<Key, T, Hash>::max\_load\_factor(float ml) {  max\_load\_factor\_ = ml;  }  template<class Key, class T, class Hash>  inline std::pair<Iterator<HashNode<std::pair<const Key, T>>>, bool> HashTable<Key, T, Hash>::insert(\_Nodeptr node) {  size\_type hashCode = node.cache;  if (!arr[hashCode % bucket\_count\_].ptr\_) {  arr[hashCode % bucket\_count\_] = elems->before\_begin();  iterator bktIter = elems->insert\_after(elems->before\_begin(), \_Nodeptr{ hashCode, node.data });  if (size\_ > 0) {  auto value = bktIter++;  size\_t cacheMod = bktIter->cache % bucket\_count\_;  arr[cacheMod] = value;  }  }  else {  elems->insert\_after(arr[hashCode % bucket\_count\_], \_Nodeptr{ hashCode, node.data });  }  ++size\_;  return std::pair<iterator, bool>(arr[hashCode % bucket\_count\_], true);  }  #endif |

dictionary\_map.h

|  |
| --- |
| #ifndef DICTIONARY\_MAP  #define DICTIONARY\_MAP  #include "hash\_table.h"  template<class Key>  class DictionaryMap {  public:  using key\_type = Key;  using mapped\_type = size\_t;  using value\_type = std::pair<Key, mapped\_type>;  using \_Nodeptr = HashNode<value\_type>;  using iterator = Iterator<\_Nodeptr>;  using const\_iterator = ConstIterator<\_Nodeptr>;  DictionaryMap(size\_t count = 1);  DictionaryMap(const DictionaryMap<Key>& copy) = default;  DictionaryMap(DictionaryMap<Key>&& move) = default;  DictionaryMap<Key>& operator=(const DictionaryMap<Key>& copy) = default;  DictionaryMap<Key>& operator=(DictionaryMap<Key>&& move) noexcept = default;  ~DictionaryMap() = default;  iterator begin() { return table.begin(); }  iterator end() { return table.end(); }  void insert(const key\_type& key);  bool erase(const key\_type& key);  std::size\_t find(const key\_type& key);  size\_t size() noexcept;  bool empty() noexcept;  void clear();  void popularWords(value\_type\* arr);  void print(std::ostream& out);  private:  HashTable<Key, size\_t> table;  void sortArr(value\_type\* arr);  };  template<class Key>  inline DictionaryMap<Key>::DictionaryMap(size\_t count) :  table{}  {  table.rehash(count);  }  template<class Key>  inline void DictionaryMap<Key>::insert(const key\_type& key) {  auto elem = table.find(key);  if (elem.ptr\_) {  ++elem->data.second;  return;  }  table.insert(std::pair<Key, size\_t>(key, 1));  }  template<class Key>  inline bool DictionaryMap<Key>::erase(const key\_type& key) {  if (table.erase(key)) {  return true;  }  return false;  }  template<class Key>  inline std::size\_t DictionaryMap<Key>::find(const key\_type& key) {  auto node = table.find(key);  if (!node.ptr\_) {  return 0;  }  return node->data.second;  }  template<class Key>  inline size\_t DictionaryMap<Key>::size() noexcept {  return table.size();  }  template<class Key>  inline bool DictionaryMap<Key>::empty() noexcept {  return table.size() == 0;  }  template<class Key>  inline void DictionaryMap<Key>::clear() {  table.clear();  }  template<class Key>  inline void DictionaryMap<Key>::popularWords(value\_type\* arr) {  if (table.size() == 0) {  return;  }  if (table.size() < 4) {  auto iter = table.begin();  for (size\_t i = 0; i < table.size(); ++i) {  arr[i] = iter->data;  ++iter;  }  return;  }  auto iter = table.begin();  for (size\_t i = 0; i < 3; ++i) {  arr[i] = iter->data;  ++iter;  }  sortArr(arr);  while (iter.ptr\_) {  if (iter->data.second > arr[0].second) {  arr[0] = iter->data;  sortArr(arr);  }  ++iter;  }  }  template<class Key>  inline void DictionaryMap<Key>::print(std::ostream& out) {  auto iter = table.begin();  while (iter.ptr\_) {  out << '(' << iter->data.first << " : " << iter->data.second << ") ";  ++iter;  }  std::cout << '\n';  }  template<class Key>  inline void DictionaryMap<Key>::sortArr(value\_type\* arr) {  for (size\_t i = 0; i < 2; i++) {  bool isSwap = false;  for (int j = 0; j < 2 - i; j++) {  if (arr[j].second > arr[j + 1].second) {  std::swap(arr[j], arr[j + 1]);  isSwap = true;  }  }  if (!isSwap) {  break;  }  }  }  #endif |