**Лабораторная работа № 1**

1. **Общая постановка задачи**

Словарь – динамическое множество, поддерживающее операции:

• добавить элемент (insert);

• удалить элемент (delete);

• проверить, есть ли элемент в словаре (search);

Словарь не может иметь записи с одинаковыми ключами (словами). АТД словарь (DictionaryList) необходимо реализовать на основе однонаправленного упорядоченного списка.

Словарные операции:

1) Создать «пустой» словарь

2) Добавить слово в словарь, обеспечивая лексикографическую упорядоченность и уникальность ключей (слов).

3) Найти слово в словаре.

4) Удалить слово из словаря.

Действия со словарями:

При выполнении действий необходимо учитывать лексикографическую упорядоченность и уникальность ключей (слов). Первый словарь – объект, к которому применяется метод, второй – параметр метода.

1) Объединить словари (merge). Метод класса для добавления в первый словарь слов, содержащихся во втором. Элементы второго словаря, уже присутствующие в первом не добавлять; после выполнения операции второй словарь должен быть пустым. При выполнении задания нельзя копировать списки, копировать узлы списков. Для вставки узлов нужно корректировать ссылки.

2) Удалить словарь (delete). Метод класса для удаления из первого словаря слов, встречающихся во втором. В процессе выполнения метода второй словарь не меняется.

3) Получить пересечение словарей (getIntersection). Дружественная функция, формирующая новый словарь, содержащий слова, присутствующие одновременно в двух словарях. Исходные словари остаются без изменения.

**Детальные требования, тест план**

1. **Таблица с детальными требованиями и тест планов**

|  |  |  |  |
| --- | --- | --- | --- |
| **Требование** | **Детальные требования** | **Данные** | **Ожидаемый результат** |
| 1. Тест Insert() | | | |
| Если элемент вставляется в словарь успешно, выводится результат «вставлено». Если такой элемент уже есть в словаре, выводится сообщение «уже существует». | std::cout << "\n-----Вставка нового элемента:\n";  if (dictInt.Insert(9))  std::cout << "вставлено\n";  if (dictInt.Insert(2))  std::cout << "вставлено\n";  if (dictInt.Insert(5))  std::cout << "вставлено\n";  if (dictInt.Insert(1))  std::cout << "вставлено\n";  if (dictInt.Insert(7))  std::cout << "вставлено\n";  std::cout << "\n-----Вставка существующего элемента:\n";  if (!dictInt.Insert(5))  std::cout << "уже существует\n";    std::cout << dictInt << '\n'; | 9 2 5 1 7 5 | -----Вставка нового элемента:  вставлено  вставлено  вставлено  вставлено  вставлено  -----Вставка существующего элемента:  уже существует  Cписок:  1 2 5 7 9 |
| 1. Тесты Search() | | | |
| Если элемент существует в словаре, выводится сообщение «найдено». Если элемента в словаре нет, выводится сообщение «не найдено» | std::cout << "\n-----Существующие элементы\n";  if (dictInt.Search(1))  {  std::cout << "найдено\n";  }  if (dictInt.Search(5))  {  std::cout << "найдено\n";  }  if (dictInt.Search(9))  {  std::cout << "найдено\n";  }  std::cout << "\n-----Несуществующие элементы\n";  if (!dictInt.Search(-8))  {  std::cout << "не найдено\n";  }  if (!dictInt.Search(4))  {  std::cout << "не найдено\n";  }  if (!dictInt.Search(15))  {  std::cout << "не найдено\n";  } | 1 5 9 -8 4 15 | -----Существующие элементы  найдено  найдено  найдено  -----Несуществующие элементы  не найдено  не найдено  не найдено |
| 1. Тесты Delete() | | | |
| Если элемент существует в списке и был успешно удален, выводится сообщение «удалено». Если такого элемента в списке нет, выводится сообщение «не найдено»  Если удаление происходит в пустом словаре, выводится сообщение «не элементов для удаления» | std::cout << "\n-----Существующие элементы\n";  if (dictInt.Delete(5))  {  std::cout << "удалено\n";  }  if (dictInt.Delete(1))  {  std::cout << "удалено\n";  }  if (dictInt.Delete(9))  {  std::cout << "удалено\n";  }  std::cout << dictInt << '\n';  std::cout << "\n-----Несуществующие элементы\n";  if (!dictInt.Delete(15))  {  std::cout << "не найдено\n";  }  std::cout << "\n-----Удаление оставшихся элементов\n";  if (dictInt.Delete(2))  {  std::cout << "удалено\n";  }  if (dictInt.Delete(7))  {  std::cout << "удалено\n";  }  std::cout << dictInt << '\n';  std::cout << "\n-----Удаление из пустого списка:\n";  if (!dictInt.Delete(5))  std::cout << "нет элементов для удаления\n";  std::cout << dictInt; | 5 1 9 15 2 7 5 | -----Существующие элементы  удалено  удалено  удалено  Cписок:  2 7  -----Несуществующие элементы  не найдено  -----Удаление оставшихся элементов  удалено  удалено  Cписок:  Пустой список!  -----Удаление из пустого списка:  нет элементов для удаления  Cписок:  Пустой список! |
| 1. Тесты Merge() | | | |
| Метод класса для добавления в первый словарь слов, содержащихся во втором. Элементы второго словаря, уже присутствующие в первом не добавлять; после выполнения операции второй словарь должен быть пустым. Если список пустой, выводится сообщение «Пустой список!» | DictionaryList<std::string> dictStr;  dictStr.Insert("789");  dictStr.Insert("345");  dictStr.Insert("567");  dictStr.Insert("321");  dictStr.Insert("82");  std::cout << "-----До MERGE():\n";  std::cout << dictStr << '\n';  DictionaryList<std::string> dictStr2;  dictStr2.Insert("81");  dictStr2.Insert("123");  dictStr2.Insert("456");  dictStr2.Insert("9");  dictStr2.Insert("987");  // повторяющийся элемент  dictStr2.Insert("567");  std::cout << dictStr2 << '\n';  dictStr.Merge(dictStr2);  std::cout << "-----После MERGE():\n";  std::cout << dictStr << '\n';  std::cout << dictStr2 << '\n';  std::cout << "\n-----Объединение с пустым списком:\n (непустой + пустой)";  dictStr.Merge(dictStr2);  std::cout << dictStr << '\n';  std::cout << dictStr2 << '\n';  std::cout << "\n(меняем местами)\n";  dictStr2.Merge(dictStr);  std::cout << dictStr << '\n';  std::cout << dictStr2 << '\n';  std::cout << "\n(еще раз меняем местами)\n";  dictStr.Merge(dictStr2);  std::cout << dictStr << '\n';  std::cout << dictStr2 << '\n';  std::cout << "\n(пустой + пустой)\n";  DictionaryList<std::string> dict1;  DictionaryList<std::string> dict2;  if (dict1.Merge(dict2))  std::cout << "\nОбъединены 2 пустых списка\n";  std::cout << dict1; | 789  345  567  321  82  81  123  456  9  987  567 | -----До MERGE():  Cписок:  321 345 567 789 82  Cписок:  123 456 567 81 9 987  -----После MERGE():  Cписок:  123 321 345 456 567 789 81 82 9 987  Cписок:  Пустой список!  -----Объединение с пустым списком:  (непустой + пустой)  Cписок:  123 321 345 456 567 789 81 82 9 987  Cписок:  Пустой список!  (меняем местами)  Cписок:  Пустой список!  Cписок:  123 321 345 456 567 789 81 82 9 987  (еще раз меняем местами)  Cписок:  123 321 345 456 567 789 81 82 9 987  Cписок:  Пустой список!  (пустой + пустой)  Объединены 2 пустых списка  Cписок:  Пустой список! |
| 1. Тесты DeleteDict() | | | |
| Метод класса для удаления из первого словаря слов, встречающихся во втором. В процессе выполнения метода второй словарь не меняется. Если происходит вычитание из пустого списка и метод работает без ошибок выводится сообщение «вычитание выполнено» | dictStr2.Insert("81");  dictStr2.Insert("123");  dictStr2.Insert("456");  dictStr2.Insert("9");  dictStr2.Insert("987");  dictStr2.Insert("888");  dictStr2.Insert("777");  dictStr2.Insert("444");  std::cout << "-----До DELETE\_DICT():\n";  std::cout << dictStr << '\n';  std::cout << dictStr2 << '\n';  dictStr.DeleteDict(dictStr2);  std::cout << "-----После DELETE\_DICT():\n";  std::cout << dictStr << '\n';  std::cout << dictStr2 << '\n';  std::cout << "\n-----Вычитание пустого списка\n";  dictStr.DeleteDict(DictionaryList<std::string>());  std::cout << dictStr << '\n';  std::cout << "\n-----Вычитание из пустого списка\n";  if (!DictionaryList<std::string>().DeleteDict(dictStr))  std::cout << "\nВычитание выполнено\n"; | 81  123  456  9  987  888  777  444 | -----До DELETE\_DICT():  Cписок:  123 321 345 456 567 789 81 82 9 987  Cписок:  123 444 456 777 81 888 9 987  -----После DELETE\_DICT():  Cписок:  321 345 567 789 82  Cписок:  123 444 456 777 81 888 9 987  -----Вычитание пустого списка  Cписок:  321 345 567 789 82  -----Вычитание из пустого списка  Вычитание выполнено |
| 1. Тесты getIntersection() | | | |
| Дружественная функция, формирующая новый словарь, содержащий слова, присутствующие одновременно в двух словарях. Исходные словари остаются без изменения. | DictionaryList<std::string> dictStr3;  dictStr3.Insert("789");  dictStr3.Insert("345");  dictStr3.Insert("567");  //неповторяющиеся элементы  dictStr3.Insert("589");  dictStr3.Insert("467");  std::cout << "\n-----До GET\_INTERSECTION():\n";  std::cout << dictStr << '\n';  std::cout << dictStr3 << '\n';  DictionaryList<std::string> dictStr4 = getIntersection(dictStr, dictStr3);  std::cout << "\n-----После GET\_INTERSECTION():\n";  std::cout << dictStr << '\n';  std::cout << dictStr3 << '\n';  std::cout << dictStr4 << '\n';  std::cout << "\n-----Пересечение пустых списков\n";  dictStr4 = getIntersection(DictionaryList<std::string>(), DictionaryList<std::string>());  std::cout << dictStr4 << '\n'; | 789  345  567  589  467 | -----До GET\_INTERSECTION():  Cписок:  321 345 567 789 82  Cписок:  345 467 567 589 789  -----После GET\_INTERSECTION():  Cписок:  321 345 567 789 82  Cписок:  345 467 567 589 789  Cписок:  345 567 789  -----Пересечение пустых списков  Cписок:  Пустой список! |