**САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО**

**ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И КИБЕРБЕЗОПАСНОСТИ**

**ВЫСШАЯ ШКОЛА ПРОГРАММНОЙ ИНЖЕНЕРИИ**

**Лабораторная работа №1**

**«Динамические структуры данных. АТД - Словарь.**

**Односвязный упорядоченный список (Шаблон класса)»**

**по предмету «Алгоритмы и структуры данных»**

**Выполнил: студент гр. 5130904/30002 Севостьянова А.В.**

**Руководитель: Череповский Д.К.**

**Санкт-Петербург**

**2024 г.**

Оглавление

[1. Общая постановка задачи 3](#_Toc158560813)

[2. Детальные требования и тест план 4](#_Toc158560814)

[3. Программа 4](#_Toc158560815)

[main.cpp 4](#_Toc158560816)

[DictionaryList.h 4](#_Toc158560817)

[tests.h 7](#_Toc158560818)

[tests.cpp 7](#_Toc158560819)

[Приложение 10](#_Toc158560820)

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 слова из словаря (deleteWords). 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.

1. **Детальные требования и тест план**

Все данные заданы корректно.

1. **Программа**

## *main.cpp*

#include "List.h"

#include "tests.h"

int main()

{

std::setlocale(LC\_ALL, "ru");

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_testWithChar\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

testWithChar();

std::cout << "\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_testWithNumbers\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

testWithNumbers();

std::cout << "\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_testWithNumbers\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

testWithString();

return 0;

}

## *DictionaryList.h*

#pragma once

#include<iostream>

#include<string>

template <typename T>

class DictionaryList

{

private:

struct Node

{

T key;

Node\* next;

Node(T value) : key(value), next(nullptr) {};

};

Node\* head;

public:

DictionaryList() : head(nullptr) {};

~DictionaryList()

{

while (head)

{

Node\* ptr = head;

head = head->next;

delete ptr;

}

};

void insert(const T value)

{

if (!head) head = new Node(value);

else

if (!search(value))

{

Node\* new\_element = new Node(value);

if (head->key > value)

{

new\_element->next = head;

head = new\_element;

}

else

{

Node\* next = head->next;

Node\* previous = head;

while (next && (next->key < value))

{

next = next->next;

previous = previous->next;

}

previous->next = new\_element;

new\_element->next = next;

}

}

}

void remove(const T value)

{

if (!head) std::cout << "Словарь пуст. Элемент не может быть удален\n";

else

if (search(value))

{

if (head->key == value) head = head->next;

Node\* ptr = head;

while (ptr->next != search(value)) ptr = ptr->next;

ptr->next = search(value)->next;

delete search(value);

}

else std::cout << "Элемент не может быть удален - отсутствует в словаре\n";

}

Node\* search(const T value) const

{

Node\* ptr = head;

while (ptr != nullptr && ptr->key != value) ptr = ptr->next;

return ptr == nullptr ? nullptr : ptr;

}

void print() const

{

if (!head) std::cout << "Элементы еще не добавлены\n";

else

{

Node\* ptr = head;

while (ptr) {

std::cout << ptr->key << " ";

ptr = ptr->next;

}

std::cout << std::endl;

}

}

void merge(DictionaryList& list)

{

Node\* ptr2 = list.head;

while (ptr2)

{

if (head->key > ptr2->key && !search(ptr2->key))

{

Node\* temp = ptr2;

ptr2 = ptr2->next;

temp->next = head;

head = temp;

}

else

{

Node\* ptr1 = head->next;

Node\* previous = head;

while (ptr1 && ptr2)

{

if (search(ptr2->key)) ptr2 = ptr2->next;

else

{

if (ptr1->key > ptr2->key)

{

Node\* temp = ptr2;

ptr2 = ptr2->next;

temp->next = ptr1;

previous->next = temp;

}

previous = previous->next;

if (ptr2 && ptr1->key < ptr2->key) ptr1 = ptr1->next;

}

}

while (ptr2)

{

previous->next = ptr2;

previous = previous->next;

ptr2 = ptr2->next;

}

}

}

ptr2 = list.head;

list.head = list.head->next;

while (ptr2)

{

list.head = list.head->next;

ptr2 = nullptr;

ptr2 = list.head;

}

}

void deleteWords(const DictionaryList& list)

{

Node\* ptr = list.head;

while (ptr)

{

if (search(ptr->key)) remove(ptr->key);

ptr = ptr->next;

}

}

DictionaryList\* getIntersection(const DictionaryList& list)

{

DictionaryList<T>\* new\_list = new DictionaryList;

Node\* ptr = list.head;

while (ptr)

{

if (search(ptr->key))

new\_list->insert(ptr->key);

ptr = ptr->next;

}

return new\_list;

}

};

## *tests.h*

#pragma once

#include"DictionaryList.h"

void testWithChar();

void testWithNumbers();

void testWithString();

## *tests.cpp*

#include "tests.h"

void testWithChar()

{

DictionaryList<char> list1;

DictionaryList<char> list2;

std::cout << "Создание словарей:\n\tlist1: ";

list1.print();

std::cout << "\tlist2: ";

list2.print();

list1.insert('c');

list1.insert('d');

list1.insert('c');

list1.insert('v');

std::cout << "\nЗаполнение словарей:\n\tlist1: ";

list1.print();

list2.insert('e');

list2.insert('d');;

list2.insert('a');

list2.insert('f');

std::cout << "\tlist2: ";

list2.print();

std::cout << "\nПоиск 'v' в list1:\t";

list1.search('v') ? std::cout << "Элемент в словаре\n" : std::cout << "Элемент не содержится в словаре\n";

list1.remove('v');

std::cout << "\nУдаление 'v' из list1:\t";

list1.print();

std::cout << "Удаление 't' из list1:\t";

list1.remove('t');

std::cout << "\n\tlist1: ";

list1.print();

std::cout << "\tlist2: ";

list2.print();

auto list3 = list1.getIntersection(list2);

std::cout << "\nПересечение list1 и list2:\t";

list3->print();

list1.deleteWords(list2);

std::cout << "\nУдаление из list1 элементов, совпадающих с list2\n\tlist1: ";

list1.print();

std::cout << "\tlist2: ";

list2.print();

list1.merge(list2);

std::cout << "\nПеремещение list2 -> list1\n\tlist1: ";

list1.print();

std::cout << "\tlist2: ";

list2.print();

delete list3;

}

void testWithNumbers()

{

DictionaryList<int> list1;

DictionaryList<int> list2;

std::cout << "Создание словарей:\n\tlist1: ";

list1.print();

std::cout << "\tlist2: ";

list2.print();

list1.insert(5);

list1.insert(2);

list1.insert(43);

list1.insert(4);

std::cout << "\nЗаполнение словарей:\n\tlist1: ";

list1.print();

list2.insert(15);

list2.insert(7);;

list2.insert(5);

list2.insert(4);

std::cout << "\tlist2: ";

list2.print();

std::cout << "\nПоиск '5' в list1:\t";

list1.search(5) ? std::cout << "Элемент в словаре\n" : std::cout << "Элемент не содержится в словаре\n";

list1.remove(5);

std::cout << "\nУдаление '5' из list1:\t";

list1.print();

std::cout << "Удаление '100' из list1:\t";

list1.remove(100);

std::cout << "\n\tlist1: ";

list1.print();

std::cout << "\tlist2: ";

list2.print();

auto list3 = list1.getIntersection(list2);

std::cout << "\nПересечение list1 и list2:\t";

list3->print();

list1.deleteWords(list2);

std::cout << "\nУдаление из list1 элементов, совпадающих с list2\n\tlist1: ";

list1.print();

std::cout << "\tlist2: ";

list2.print();

list1.merge(list2);

std::cout << "\nПеремещение list2 -> list1\n\tlist1: ";

list1.print();

std::cout << "\tlist2: ";

list2.print();

delete list3;

}

void testWithString()

{

DictionaryList<std::string> list1;

DictionaryList<std::string> list2;

std::cout << "Создание словарей:\n\tlist1: ";

list1.print();

std::cout << "\tlist2: ";

list2.print();

list1.insert("dog");

list1.insert("apple");

std::cout << "\nЗаполнение словарей:\n\tlist1: ";

list1.print();

list2.insert("dog");

list2.insert("cat");

std::cout << "\tlist2: ";

list2.print();

std::cout << "\nПоиск 'cat' в list1:\t";

list1.search("cat") ? std::cout << "Элемент в словаре\n" : std::cout << "Элемент не содержится в словаре\n";

list1.remove("dog");

std::cout << "\nУдаление 'dog' из list1:\t";

list1.print();

std::cout << "Удаление 'cat' из list1:\t";

list1.remove("cat");

std::cout << "\n\tlist1: ";

list1.print();

std::cout << "\tlist2: ";

list2.print();

auto list3 = list1.getIntersection(list2);

std::cout << "\nПересечение list1 и list2:\t";

list3->print();

list1.deleteWords(list2);

std::cout << "\nУдаление из list1 элементов, совпадающих с list2\n\tlist1: ";

list1.print();

std::cout << "\tlist2: ";

list2.print();

list1.merge(list2);

std::cout << "\nПеремещение list2 -> list1\n\tlist1: ";

list1.print();

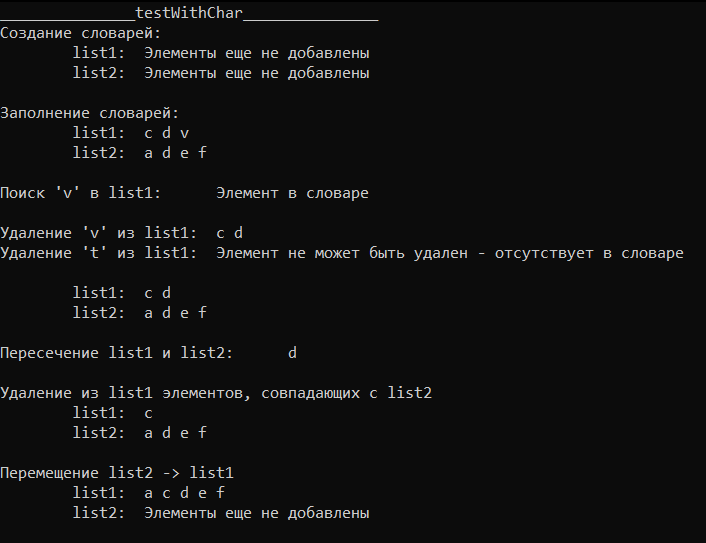
std::cout << "\tlist2: ";

list2.print();

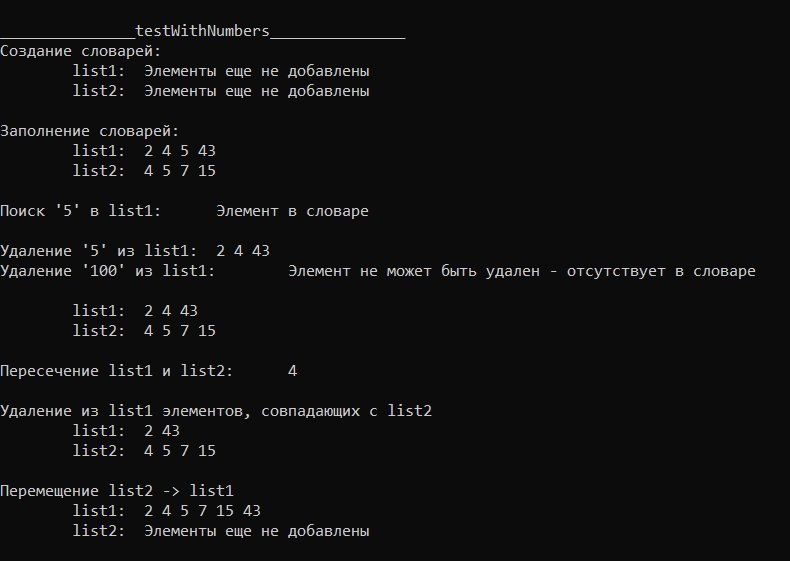
delete list3;

}

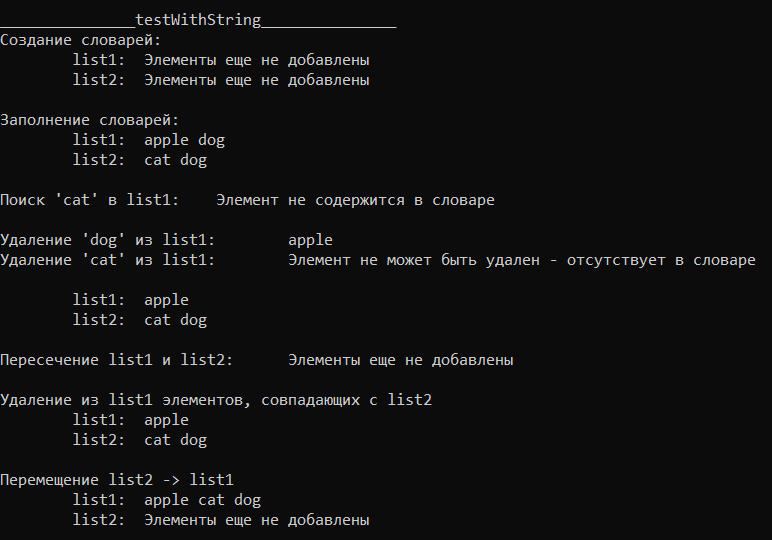
# Приложение

****

*Рис. 1 тестирование функцией testWithChar*

****

*Рис. 2 тестирование функцией testWithNumbers*

****

*Рис. 3 тестирование функцией testWithString*