Отчет по работе Класс SingleList (vocabulary)

Выполнил: Смирнов Е. А., гр. 3530904/00006

Преподаватель: Киреев С.П.

Постановка:
Реализовать односвязный список, в котором элементы упорядочены в лексикографическом порядке, с использованием методов и дружественное функции, написать функцию тестирование этого класса.

Реализовать:

Словарные операции:

1) Создать «пустой» словарь
2) Добавить запись в словарь, обеспечивая лексикографическую упорядоченность.
3) Найти слово в словаре.
4) Удалить слово из словаря

Действия со словарями:

1) Добавить в словарь слова, встречающиеся во втором словаре. Слова, встречающиеся в двух словарях, включать в итоговый словарь один раз. Второй словарь после переноса слов сделать пустым. Объединение множеств.
2) Удалить из словаря все слова, встречающиеся в другом словаре. Вычитание множеств
3) Сформировать словарь, содержащий слова, присутствующие одновременно в двух других. Пересечение множеств.
4) Вывести все записи словаря

SinglyLinkedOrderedList.h

```h
#pragma once

#include <fstream>

template<typename T>
class SinglyLinkedOrderedList
{
public:
	SinglyLinkedOrderedList();   //Конструктор "по умолчанию"
	~SinglyLinkedOrderedList(); //Деструктор
	SinglyLinkedOrderedList(SinglyLinkedOrderedList<T>& src) noexcept;
	SinglyLinkedOrderedList(SinglyLinkedOrderedList<T>&& src) noexcept;
	SinglyLinkedOrderedList<T>& operator=(SinglyLinkedOrderedList<T>&& src);
	void addList(SinglyLinkedOrderedList& other);
	// обьединение словарей ( в вырвый ) без повторений
	// из 2 словарей  - второй становаится пустым
	void substractionList(SinglyLinkedOrderedList& other);
	// удаление во втором списке, если они есть в первом
	void insertItem(T data);   // добааление в список
	bool deleteItem(T data);  // Удаление узла с заданным значением
	bool searchItem(T data); // Поиск записи с заданным значением
	void print() const;     // вывод списка
	int getSize() { return count_; };

	template<typename T>
	friend SinglyLinkedOrderedList<T> mergeList(SinglyLinkedOrderedList<T>& list1,
SinglyLinkedOrderedList<T>& list2);
	// новый словарь только из общих слов двух словарей

private:
	template<typename T>
	struct Node
	{
		Node* next_;
		T item_;

		Node(T item = T(), Node* next = nullptr)
			//Конструктор для создания нового элемента
		{
			this->item_ = item;
			this->next_ = next;
		}
	};
	Node<T>* head_; //Указатель на голову
	int count_; //Количество элементов
	void deleteNode(Node<T>* data);
	// Удаление заданного узла
	Node<T>* searchNode(T data);
	// Поиск узла (адрес) с заданным значением
};
```

SinglyLinkedOrderedList.cpp

```cpp
#include <iostream>
#include <fstream>
#include "SinglyLinkedOrderedList.h"

// создать пустой словарь
template<typename T>
SinglyLinkedOrderedList<T>::SinglyLinkedOrderedList()
{
	head_ = nullptr;
	count_ = 0;
}

// Добавить запись в словарь, обеспечивая лексико-графическую упорядоченность
template<typename T>
bool SinglyLinkedOrderedList<T>::insertItem(T data)
{

	if (head_ == nullptr)
	{
		head_ = new Node<T>(data);
		count_++;
		return true;
	}
	else
	{
		Node<T>* current = head_;
		Node<T>* tmp = current;

		while (current and (data >= current->item_))
		{
			tmp = current;
			current = current->next_;
		}

		if (tmp->item_ != data)
		{
			Node<T>* newElement = new Node <T>(data);

			if (current == head_)
			{
				newElement->next_ = head_;
				head_ = newElement;
			}

			else
			{
				// разрыв связи между tmp и cur
				newElement->next_ = current;
				tmp->next_ = newElement;
			}
			count_++;
			return true;
		}


	}

	return false;

}

// Найти слово в словаре
template<typename T>
bool SinglyLinkedOrderedList<T>::searchItem(T data)
{
	return (searchNode(data) != nullptr);
}
```

```
template<typename T>
SinglyLinkedOrderedList<T>::Node<T>* SinglyLinkedOrderedList<T>::searchNode(T data)
{
	Node<T>* temp = head_;

	while ((temp != nullptr) && (temp->item_ != data))
	{
		temp = temp->next_;
	}

	return temp;
}

// Удалить слово из словаря.
template<typename T>
bool SinglyLinkedOrderedList<T>::deleteItem(T data)
{
	if(searchNode(data)){
	deleteNode(searchNode(data));
	return true;
	}
	return false;
}

template<typename T>
void SinglyLinkedOrderedList<T>::deleteNode(Node<T>* data)
{
	Node<T>* cur = head_;

	// если голова
	if (cur->item_ == data->item_) {
		head_ = head_->next_;
		delete cur;
		count_--;
	}

	else {

		if (data == nullptr)
		{
			throw ("SinglyLinkedList::deleteNode - неверно задан адрес
удаляемого узла");
		}

		else
		{
			while (cur->next_ != nullptr)
			{
				if ((cur->next_)->item_ == data->item_)
				{
					Node<T>* deleted = cur->next_;
					cur->next_ = (cur->next_)->next_;
					delete deleted;
					deleted = nullptr;
					break;
				}
				cur = cur->next_;
			}
			count_--;
		}
	}
}

// первое задание
template<typename T>
void SinglyLinkedOrderedList<T>::addList(SinglyLinkedOrderedList& other)
{
	Node<T>* current = other.head_;
	while (current != nullptr)
```

```
	{
		this->insertItem(current->item_);
		current = current->next_;
	}
	current = other.head_;
	Node<T>* temp = current;
	while (current != nullptr)
	{
		temp = current;
		current = current->next_;
		delete temp;
	}
	other.head_ = nullptr;
}

// второе задание
template<typename T>
void SinglyLinkedOrderedList<T>::substractionList(SinglyLinkedOrderedList& other)
{
	Node<T>* temp = nullptr;
	Node<T>* next = other.head_;
	while (next != nullptr)
	{
		temp = next;
		if (this->searchItem(temp->item_) == 1)
			this->deleteItem(temp->item_);
		next = next->next_;
	}
}

// третье задание
template<typename T>
SinglyLinkedOrderedList<T> mergeList(SinglyLinkedOrderedList<T>& list1,
SinglyLinkedOrderedList<T>& list2)
{
	SinglyLinkedOrderedList<T> list3;
	// SinglyLinkedOrderedList<T>::Node<T>* current = list2.head_;

	while (list2.head_ != nullptr)
	{
		if (list1.searchItem(list2.head_->item_))
		{
			list3.insertItem(list2.head_->item_);
		}
		list2.head_ = list2.head_->next_;
	}

	return list3;
}
template<typename T>
SinglyLinkedOrderedList<T>::SinglyLinkedOrderedList<T>(SinglyLinkedOrderedList<T>&&
src) noexcept
{
	head_ = nullptr;
	head_ = src.head_;
	src.head_ = nullptr;

	count_ = 0;
	count_ = src.count_;
	src.count_ = 0;
}

template<typename T>
SinglyLinkedOrderedList<T>&
SinglyLinkedOrderedList<T>::operator=(SinglyLinkedOrderedList<T>&& src)
{
	Node<T>* current = head_;
	Node<T>* temp = current;
	while (current != nullptr)
	{
```

```
            temp = current;
            current = current->next_;
            delete temp;
        }
        head_ = nullptr;

        head_ = src.head_;
        src.head_ = nullptr;

        return *this;
}

// вывести список
template<typename T>
void SinglyLinkedOrderedList<T>::print() const
{
        Node<T>* current = head_;

        while (current != nullptr)
        {
            std::cout << current->item_ << ' ';
            current = current->next_;  // Переход к следующему элементу
        }
        std::cout << std::endl;
}

template<typename T>
SinglyLinkedOrderedList<T>::~SinglyLinkedOrderedList()
{
        Node<T>* current = nullptr;
        Node<T>* next = head_;

        while (next != nullptr)
        {
            current = next;
            next = next->next_;
            delete current;
        }
}



//копирование
template<typename T>
SinglyLinkedOrderedList<T>::SinglyLinkedOrderedList(SinglyLinkedOrderedList<T>& src)
noexcept
{
        head_ = nullptr;
        count_ = 0;
        Node<T>* next = src.head_;
        while (next != nullptr) {
            insertItem(next->item_);
            next = next->next_;
        }
}
```

Main.cpp и тестовая функция

```
#include <iostream>
#include "SinglyLinkedOrderedList.h"
#include "SinglyLinkedOrderedList.cpp"

void Test();
int main()
{
	setlocale(LC_ALL, "RUS");
	Test();
	return 0;
}

void Test()
{
	SinglyLinkedOrderedList <int> test1;
	std::cout << "Вывод пустого списка" << std::endl << std::endl;
	test1.print();

	for (int i = 0; i <= 20; i++)
	{
		// заполнение
		test1.insertItem(i);
	}

	test1.insertItem(10); // повторное слово не добовляется

	SinglyLinkedOrderedList <int> test2;
	for (int i = 10; i <= 30; i++)
	{
		test2.insertItem(i);
	}

	// поиск по значению
	int search = 5;
	if (test1.searchItem(search)) {
		std::cout << "Find" << std::endl;
	}
	else {
		std::cout << "not Find" << std::endl;
	}
	if (test2.searchItem(search)) {
		std::cout << "Find" << std::endl;
	}
	else {
		std::cout << "not Find" << std::endl;
	}

	// вывод спискска
	std::cout << "Вывод первого списка" << std::endl;
	test1.print();
	std::cout << "Вывод второго списка" << std::endl;
	test2.print();

	// add list из 2 списка в первой неповторяющиеся и 2 список будет пустой
	test1.addList(test2);
	std::cout << "Вывод списков 1 и 2 после метода addlist списка" << std::endl <<
std::endl;
	test1.print();
	test2.print();

	for (int i = 10; i <= 31; i++)
	{
		test2.insertItem(i);
	}
	// удаляет все слова из 1 списка,которые есть во втором
```

```
	std::cout << "Вывод списков 1 и 2 после метода substractionList списка" <<
std::endl << std::endl;
	test1.substractionList(test2);

	test1.print();
	test2.print();

	SinglyLinkedOrderedList <std::string> test3;
	SinglyLinkedOrderedList <std::string> test6; // пустой словарь

	test3.insertItem("Aaa");
	test3.insertItem("AAa");
	test3.insertItem("Bbb");
	test3.insertItem("Ccc");
	test3.insertItem("daa");
	test3.insertItem("Saq");
	test3.insertItem("Saa");
	test3.insertItem("aaaa");

	std::cout << "проврека на правельность выволнения при пустом списке(тест 6):" <<
std::endl;

	test6.substractionList(test3);
	std::cout << "Вывод третьего списка" << std::endl;
	test3.print();
	std::cout << "Вывод шестого списка" << std::endl;
	test6.print();


	// копирование
	SinglyLinkedOrderedList <std::string> test4 = test3;

	std::cout << "удаляем из тест 3 элемент" << std::endl;
	std::cout << "Вывод третьего списка" << std::endl;
	test3.print();
	std::cout << std::endl;
	std::cout << "Вывод четветрого списка" << std::endl;
	test4.print();
	std::cout << std::endl;

	SinglyLinkedOrderedList <std::string> test5;
	test5 = mergeList(test3, test4); // формуирует новый список из значений, которые
есть и в первом и во втором.
	std::cout << "Новый список после метода mergeList" << std::endl;
	std::cout << std::endl;
	test5.print();
}
```

Выходные значения