Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 1 з дисципліни

“Мультипарадигменне програмування”

Виконав студент: ІП-01 Танасієнко Олександр

Перевірив: ас. Очеретяний О. К.

Київ 2022

1. **Завдання лабораторної роботи**

Завдання 1:

2. Обчислювальна задача тут тривіальна: для текстового файлу ми хочемо відобразити N (наприклад, 25) найчастіших слів і відповідну частоту їх повторення, упорядковано за зменшенням. Слід обов'язково нормалізувати використання великих літер і ігнорувати стоп-слова, як «the», «for» тощо. Щоб все було просто, ми не піклуємося про порядок слів з однаковою частотою повторень. Ця обчислювальна задача відома як **term frequency**.
3. Ось такий вигляд матимуть ввід і відповідно вивід результату програми:
4. **Input:**
5. `White tigers live mostly in India`
6. `Wild lions live mostly in Africa`
7.
8. **Output:**
9. `live - 2`
10. `mostly - 2`
11. `africa - 1`
12. `india - 1`
13. `lions - 1`
14. `tigers - 1`
15. `white - 1`
16. `wild - 1`

Завдання 2:

Тепер, нам потрібно виконати задачу, що називається словниковим індексуванням. Для текстового файлу виведіть усі слова в алфавітному порядку разом із номерами сторінок, на яких Ці слова знаходяться. Ігноруйте всі слова, які зустрічаються більше 100 разів. Припустимо, що сторінка являє собою послідовність із 45 рядків. Наприклад, якщо взяти книгу Pride and Prejudice, перші кілька записів індексу будуть:

```
abatement - 89
abhorrence - 101, 145, 152, 241, 274, 281
abhorrent - 253
abide - 158, 292
```

2. Опис алгоритму

Завдання 1:

1. Зчитування кількості слів для відображення, назви файлу.
2. Оголошення початкових змінних, масивів та виділення пам’яті під слова та їх повтори.
3. Оголошення та ініціалізація масиву з пунктуаційними знаками.
4. Оголошення та ініціалізація масиву стоп-слів.
5. Потокове зчитування частин тексту з файлу з відкиданням розділювачів(‘\n’, ‘ ‘), кожна частина записується у **symbolsSequence** в циклі.

5.1. Якщо ємність слів дорівнює поточній кількості слів, виділення удвічі більше пам’яті і копіювання старих даних.

5.2. Видалення зі зчитаної послідовності **symbolsSequence** яку потім можна буде вважати словом, знаків пунктуації, зберігаємо результат у змінну **word.**

5.3. Нормалізація великих літер шляхом перевірки, чи є ascii код кожного символу змінної **word** великою літерою, якщо так, то до його коду додається 32. Таким чином велика літера перетворюється в малу.

5.4. Перевірка, чи є word стоп-словом, шляхом порівняння **word** зі словами, що зберігаються в масиві стоп-слів.

5.5. Перевірка, чи містить **word** тільки англійські літери чи символи апострофа чи тире. Якщо ні, то відкидаю **word**(не вважаю словом). Також словами не є “”, “-“, “”.

5.6. Якщо слово не стоп-слово і є англійським, то:

5.6.1. Перевірка, чи було додане це слово раніше до списку слів з повторами, якщо так, то збільшую кількість повторів слова на 1; якщо ні, то додаю це слово в кінець списку слів, і кількість повторів встановлюю рівною 1.

6. З допомогою сортування бульбашкою сортую слова за повторами.
7. Виведення на екран слів та їх повторів відсортованих за кількістю повторів кількості, заданої користувачем, або менше, якщо у тексті слів менше, ніж ввів користувач.

Завдання 2:

1. Зчитування назви файлу.
2. Оголошення початкових змінних, масивів та виділення пам’яті під них для слів, їх повторів та номерів сторінок. Одна сторінка – 1800 символів без урахування роздіювачів (‘ ‘, ‘\n’).
3. Оголошення та ініціалізація масиву з пунктуаційними знаками.
4. Оголошення та ініціалізація масиву стоп-слів.

5. Потокове зчитування частин тексту з файлу з відкиданням розділювачів(‘\n’, ‘ ‘), кожна частина записується у **symbolsSequence** в циклі.

5.1. Якщо ємність слів дорівнює поточній кількості слів, виділення удвічі більше пам’яті і копіювання старих даних з масивів.

5.2. Видалення зі зчитаної послідовності **symbolsSequence** яку потім можна буде вважати словом, знаків пунктуації, зберігаємо результат у змінну **word.**

5.3. Нормалізація великих літер шляхом перевірки, чи є ascii код кожного символу змінної **word** великою літерою, якщо так, то до його коду додається 32. Таким чином велика літера перетворюється в малу.

5.4. Перевірка, чи є word стоп-словом, шляхом порівняння **word** зі словами, що зберігаються в масиві стоп-слів.

5.5. Перевірка, чи містить **word** тільки англійські літери чи символи апострофа чи тире. Якщо ні, то відкидаю **word**(не вважаю словом). Також словами не є “”, “-“, “”.

5.6. Якщо слово не стоп-слово і є англійським, то:

5.6.1. Перевірка, чи було додане це слово раніше до списку слів, якщо так, то обраховую поточний номер сторінки, зберігаю його та збільшую кількість повторів слова на 1; якщо ні, то зберігаю слово у кінець списку слів, кількість повторів = 1, зберігаю розрахований номер сторінки.

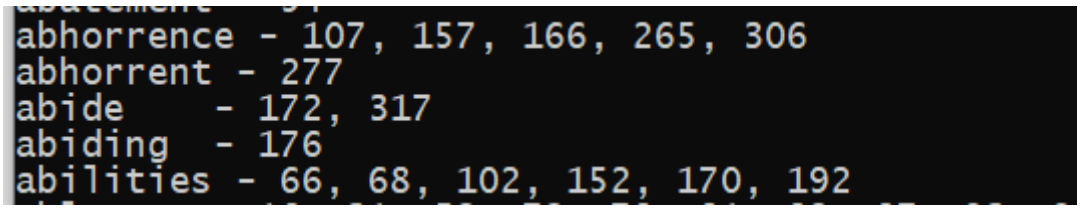
6. З допомогою сортування бульбашкою сортую слова, порівнюючи слова у алфавітному порядку таким чином: послідовно порівнюю ascii коди відповідних символів слів, якщо символ поточного слова має менший код, можна завершити порівняння, інакше якщо більший, потрібно робити обмін слів, і відповідно зберігаю у булеву змінну інформацію про це.
7. Виведення на екран слів та номерів сторінок, на яких вони зустрічаються.

3. Використовувані функції:

std::setw, std::left

```
cout << std::setw(8) << std::left << words[i] << " - ";
```

Використав для форматованого виводу результатів роботи програми:



Програмний код

Завдання 1:

```
#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

int main()
{
	cout << "Enter number of frequent words to display(N): ";
	int numberOfFrequentWordsToDisplay;
	cin >> numberOfFrequentWordsToDisplay;
	cout << "Enter name of file with text: ";
	string fileName;
	cin >> fileName;

	int wordsCapacity = 10;
	string* words = new string[wordsCapacity];
	int* wordOccurrences = new int[wordsCapacity];

	char punctuationMarksAndSymbols[] = { ',', '"', '\'', '.', '!', '?', ':', ';',
'(', ')', '{', '}' };
	int numberOfSymbolsToCheck = 12;

	string stopWords[] = { "the", "for", "in", "at", "on", "by", "of", "with", "a",
"but", "to", "am", "is", "are", "and"};
	int stopWordsNumber = 15;

	int currentWordIndex = 0;
	string symbolsSequence;

	int i = 0, j = 0;

	ifstream inFile(fileName);

loopstart:
	if (inFile >> symbolsSequence)
	{
		if (wordsCapacity == currentWordIndex)
		{
			wordsCapacity *= 2;
			string* newWords = new string[wordsCapacity];
			int* newWordOccurrences = new int[wordsCapacity];

			i = 0;
		copyWord:
			if (i < wordsCapacity / 2)
			{
				newWords[i] = words[i];
				newWordOccurrences[i] = wordOccurrences[i];
				i++;
				goto copyWord;
```

```
                }

                delete[] words;
                words = newWords;

                delete[] wordOccurrences;
                wordOccurrences = newWordOccurrences;
        }

        string word;
        i = 0;
symbolsRemoval:
        if (symbolsSequence[i] != '\0')
        {
                bool validSymbol = true;
                int k = 0;
        checkSymbol:
                if (k < numberOfSymbolsToCheck)
                {
                        if (symbolsSequence[i] == punctuationMarksAndSymbols[k])
                        {
                                validSymbol = false;
                                goto endCheckSymbol;
                        }
                        k++;
                        goto checkSymbol;
                }
        endCheckSymbol:
                if (validSymbol)
                {
                        word += symbolsSequence[i];
                }
                i++;
                goto symbolsRemoval;
        }

        i = 0;
capitalLettersNormalization:
        if (word[i] != '\0')
        {
                if (word[i] >= 65 && word[i] <= 90)
                {
                        word[i] += 32;
                }
                i++;
                goto capitalLettersNormalization;
        }

        bool isStopWord = false;
        i = 0;
checkStopWord:
        if (i < stopWordsNumber)
        {
                if (word == stopWords[i])
                {
                        isStopWord = true;
                        goto endCheckStopWord;
                }
                i++;
                goto checkStopWord;
        }
endCheckStopWord:

        bool isValidWord = true;
        i = 0;
```

```
	checkValidWord:
		if (word[i] != '\0')
		{
			if (!((word[i] >= 97 && word[i] <= 122) || (word[i] == '-') ||
(word[i] == '\'')))
			{
				isValidWord = false;
				goto endCheckValidWord;
			}
			i++;
			goto checkValidWord;
		}
	endCheckValidWord:

		if (word == "" || word == "'" || word == "-")
		{
			isValidWord = false;
		}

		if (!isStopWord && isValidWord)
		{
			i = 0;
			bool wordWasPreviouslyAdded = false;
		checkIfWordWasPreviously:
			if (i < currentWordIndex)
			{
				if (words[i] == word)
				{
					wordWasPreviouslyAdded = true;
					wordOccurrences[i]++;
					goto endCheckIfWordWasPreviously;
				}
				i++;
				goto checkIfWordWasPreviously;
			}
		endCheckIfWordWasPreviously:
			if (!wordWasPreviouslyAdded)
			{
				words[currentWordIndex] = word;
				wordOccurrences[currentWordIndex] = 1;
				currentWordIndex++;
			}
		}
		goto loopstart;
	}

	i = j = 0;
outerSortingLoop:
	if (i < currentWordIndex - 1)
	{
		j = 0;
	innerSortingLoop:
		if (j < currentWordIndex - i - 1)
		{
			if (wordOccurrences[j] < wordOccurrences[j + 1])
			{
				string temp = words[j];
				words[j] = words[j + 1];
				words[j + 1] = temp;
				int tempOccurences = wordOccurrences[j];
				wordOccurrences[j] = wordOccurrences[j + 1];
				wordOccurrences[j + 1] = tempOccurences;
			}
			j++;
			goto innerSortingLoop;
```

```
                }
                i++;
                goto outerSortingLoop;
        }

        int displayWordsNumber = 0;
        if (numberOfFrequentWordsToDisplay < currentWordIndex)
        {
                displayWordsNumber = numberOfFrequentWordsToDisplay;
        }
        else
        {
                displayWordsNumber = currentWordIndex;
        }

        if (displayWordsNumber == 0)
        {
                cout << "No words!";
        }
        else
        {
                i = 0;
        out:
                if (i < displayWordsNumber)
                {
                        cout << std::setw(8) << std::left << words[i] << " - " <<
wordOccurrences[i] << "\n";
                        i++;
                        goto out;
                }
        }

        delete[] words;
        delete[] wordOccurrences;
    }
```

Завдання 2:

```
#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

int main()
{
        cout << "Enter name of file with text: ";
        string fileName;
        cin >> fileName;

        int wordsCapacity = 10;

        int symbolsPerPage = 1800;
        int symbolsCounter = 0;
        int currentPage = 1;

        string* words = new string[wordsCapacity];
        int* wordOccurrences = new int[wordsCapacity];
        int** wordsPages = new int* [wordsCapacity];

        int i = 0, j = 0;

memoryAlloc:
        if (i < wordsCapacity)
        {
```

```
			wordsPages[i] = new int[101];
			i++;
			goto memoryAlloc;
		}

		char punctuationMarksAndSymbols[] = { ',', '"', '\'', '.', '!', '?', ':', ';',
'(', ')', '{', '}' };
		int numberOfSymbolsToCheck = 12;

		string stopWords[] = { "the", "for", "in", "at", "on", "by", "of", "with", "a",
"but", "to", "am", "is", "are", "and" };
		int stopWordsNumber = 15;

		int currentWordIndex = 0;
		string symbolsSequence;

		ifstream inFile(fileName);

	loopstart:
		if (inFile >> symbolsSequence)
		{
			if (wordsCapacity == currentWordIndex)
			{
				wordsCapacity *= 2;
				string* newWords = new string[wordsCapacity];
				int* newWordOccurrences = new int[wordsCapacity];
				int** newWordsPages = new int* [wordsCapacity];

				i = wordsCapacity / 2;
			newMemoryAlloc:
				if (i < wordsCapacity)
				{
					newWordsPages[i] = new int[101];
					i++;
					goto newMemoryAlloc;
				}

				i = 0;
			copyWord:
				if (i < wordsCapacity / 2)
				{
					newWords[i] = words[i];
					newWordOccurrences[i] = wordOccurrences[i];
					newWordsPages[i] = wordsPages[i];
					i++;
					goto copyWord;
				}

				delete[] words;
				words = newWords;

				delete[] wordOccurrences;
				wordOccurrences = newWordOccurrences;

				delete[] wordsPages;
				wordsPages = newWordsPages;
			}

			string word;
			i = 0;
		symbolsRemoval:
			if (symbolsSequence[i] != '\0')
			{
				symbolsCounter++;
				bool validSymbol = true;
```

```
                int k = 0;
        checkSymbol:
                if (k < numberOfSymbolsToCheck)
                {
                        if (symbolsSequence[i] == punctuationMarksAndSymbols[k])
                        {
                                validSymbol = false;
                                goto endCheckSymbol;
                        }
                        k++;
                        goto checkSymbol;
                }
        endCheckSymbol:
                if (validSymbol)
                {
                        word += symbolsSequence[i];
                }
                i++;
                goto symbolsRemoval;
        }

        i = 0;
capitalLettersNormalization:
        if (word[i] != '\0')
        {
                if (word[i] >= 65 && word[i] <= 90)
                {
                        word[i] += 32;
                }
                i++;
                goto capitalLettersNormalization;
        }

        bool isStopWord = false;
        i = 0;
checkStopWord:
        if (i < stopWordsNumber)
        {
                if (word == stopWords[i])
                {
                        isStopWord = true;
                        goto endCheckStopWord;
                }
                i++;
                goto checkStopWord;
        }
endCheckStopWord:

        bool isValidWord = true;
        i = 0;
checkValidWord:
        if (word[i] != '\0')
        {
                if (!((word[i] >= 97 && word[i] <= 122) || (word[i] == '-') ||
(word[i] == '\'')))
                {
                        isValidWord = false;
                        goto endCheckValidWord;
                }
                i++;
                goto checkValidWord;
        }
endCheckValidWord:

        if (word == "" || word == "'" || word == "-")
```

```
                {
                        isValidWord = false;
                }

                if (!isStopWord && isValidWord)
                {
                        i = 0;
                        bool wordWasPreviouslyAdded = false;
                checkIfWordWasPreviously:
                        if (i < currentWordIndex)
                        {
                                if (words[i] == word)
                                {
                                        wordWasPreviouslyAdded = true;
                                        if (wordOccurrences[i] < 100)
                                        {
                                                if (symbolsCounter >= symbolsPerPage)
                                                {
                                                        currentPage += symbolsCounter /
symbolsPerPage;
                                                        symbolsCounter = 0;
                                                }
                                                wordsPages[i][wordOccurrences[i]] = currentPage;
                                        }
                                        wordOccurrences[i]++;
                                        goto endCheckIfWordWasPreviously;
                                }
                                i++;
                                goto checkIfWordWasPreviously;
                        }
                endCheckIfWordWasPreviously:
                        if (!wordWasPreviouslyAdded)
                        {
                                words[currentWordIndex] = word;
                                wordOccurrences[currentWordIndex] = 1;

                                if (symbolsCounter >= symbolsPerPage)
                                {
                                        currentPage += symbolsCounter / symbolsPerPage;
                                        symbolsCounter = 0;
                                }
                                wordsPages[currentWordIndex][0] = currentPage;

                                currentWordIndex++;
                        }
                }
                goto loopstart;
        }

        i = j = 0;
outerSortingLoop:
        if (i < currentWordIndex - 1)
        {
                j = 0;
        innerSortingLoop:
                if (j < currentWordIndex - i - 1)
                {
                        bool makeWordsSwap = false;

                        int f = 0;
                wordsComparison:
                        if (words[j][f] < words[j + 1][f])
                        {
                                goto endWordsComparison;
                        }
```

```
                if (words[j][f] > words[j + 1][f])
                {
                        makeWordsSwap = true;
                        goto endWordsComparison;
                }
                if (words[j][f] != '\0' && words[j + 1][f] != '\0')
                {
                        f++;
                        goto wordsComparison;
                }
        endWordsComparison:
                if (makeWordsSwap)
                {
                        string temp = words[j];
                        words[j] = words[j + 1];
                        words[j + 1] = temp;

                        int tempOccurences = wordOccurrences[j];
                        wordOccurrences[j] = wordOccurrences[j + 1];
                        wordOccurrences[j + 1] = tempOccurences;

                        int* tempPages = wordsPages[j];
                        wordsPages[j] = wordsPages[j + 1];
                        wordsPages[j + 1] = tempPages;
                }
                j++;
                goto innerSortingLoop;
        }
        i++;
        goto outerSortingLoop;
}

if (currentWordIndex == 0)
{
        cout << "No words!";
}
else
{
        i = 0;
out:
        if (i < currentWordIndex)
        {
                if (wordOccurrences[i] <= 100)
                {
                        cout << std::setw(8) << std::left << words[i] << " - ";

                        j = 0;
                printPages:
                        if (j < wordOccurrences[i])
                        {
                                cout << wordsPages[i][j];
                                j++;
                                if (j < wordOccurrences[i])
                                {
                                        cout << ", ";
                                }
                                goto printPages;
                        }
                        cout << '\n';
                }
                i++;
                goto out;
        }
}
```

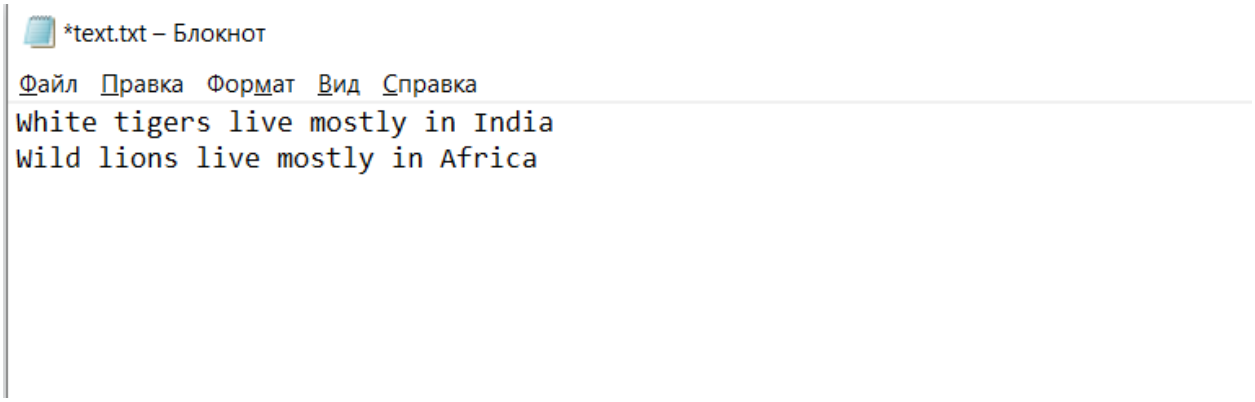
```
        delete[] words;
        delete[] wordOccurrences;

        i = 0;
clearMemory:
        if (i < wordsCapacity)
        {
                delete[] wordsPages[i];
                i++;
                goto clearMemory;
        }
        delete[] wordsPages;
    }
```

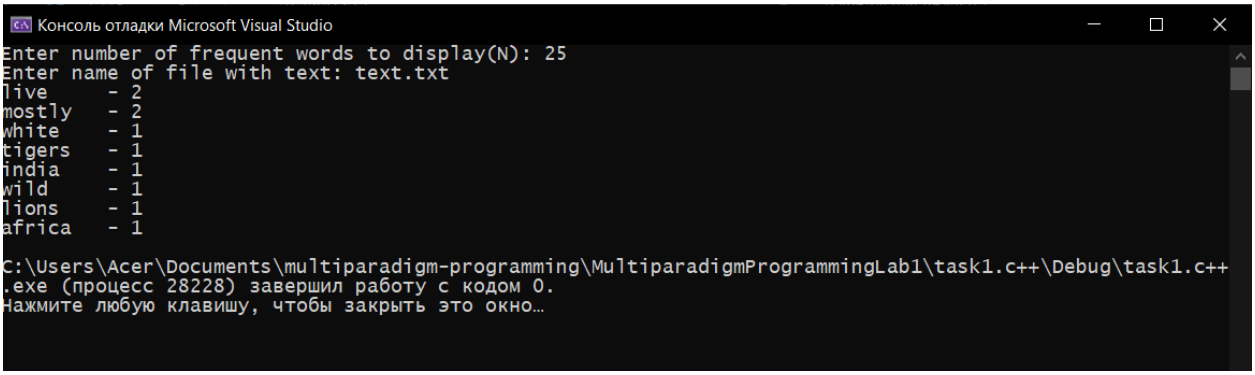
4. Приклади виконання програм

Завдання 1:

Вхідний файл:

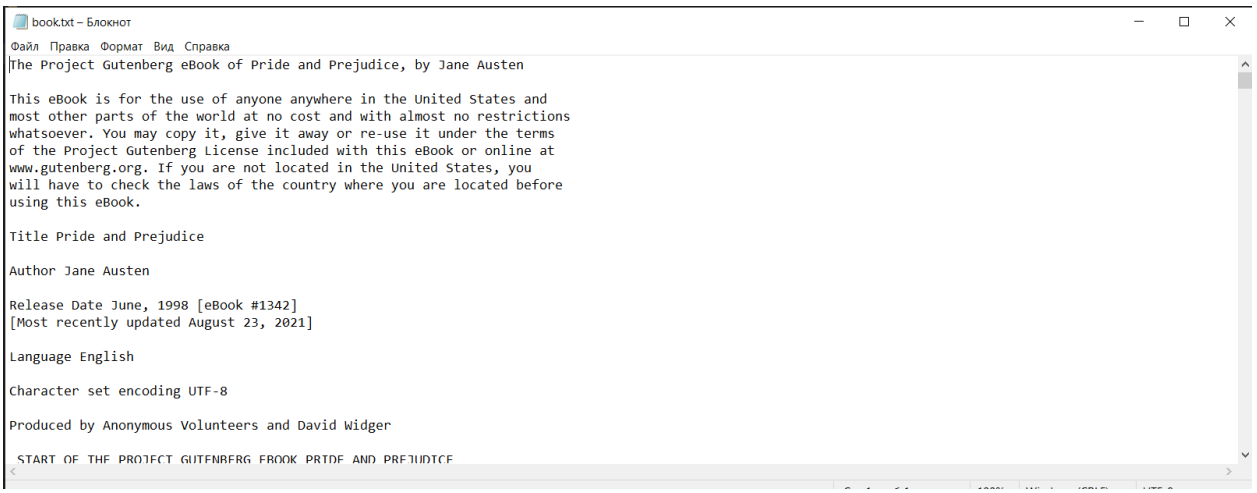


Результати роботи програми:



Завдання 2:

Вхідний файл:



Результати роботи програми: