МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра систем автоматизированного проектирования

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Программирование»

Тема: Работа со строками. Использование struct и class

Студентка гр. 3353 _____ Карпенко А.Ю.

Преподаватель _____ Калмычков В.А.

Санкт-Петербург
2024

Оглавление

Исходная формулировка задания 2
Математическая постановка задания 2
Контрольный пример 2
Разработка интерфейса пользователя 3
Организация ввода-вывода 3
Организация хранения данных 3
Описание алгоритма 6
Текст программы 7
Результаты работы программы 10
Вывод 11

Исходная формулировка задания

Дан файл с символами, в первой строке содержатся маркеры. Каждая строка состоит из слов, разделенных пробелами, или является пустой. Переставить последовательно в строке подходящие слова следующим образом: по последнему символу слова найти к нему ближайшее справа, начинающееся с этого же символа, провести слияние слов (левое переносится перед правым, символ остается один)

Математическая постановка задания

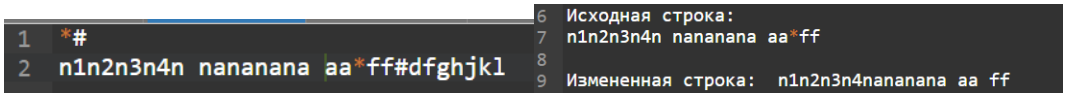
Дано: два файла, которых содержатся слова, в одном файле два маркера/ один маркер (для структуры/класса), в другом маркер и количество символов/ только количество символов для занесения в массив.

Необходимо: занести символы в массив, взять слово и найти к нему слово правее ток, чтобы оно начиналось с последней буквы изначального. Соединить их.

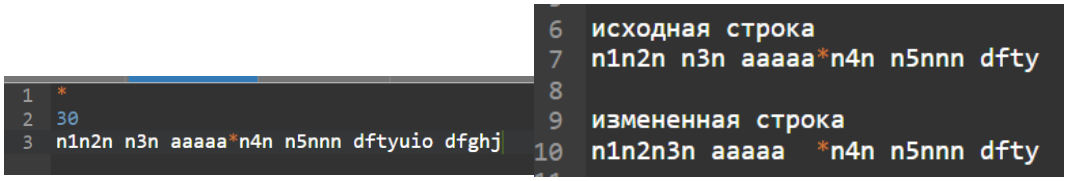
Способ решения: будем сравнивать последний символ слова с первыми символами последующих слов

Контрольный пример

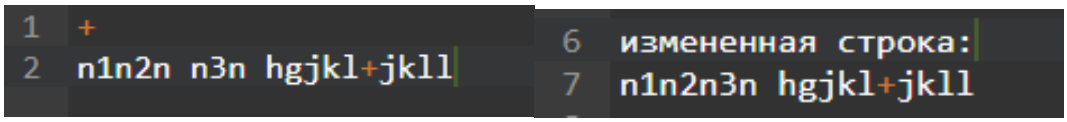
Использование структуры. 2 маркера



Использование структуры. 1 маркер и заданная максимальная длина



Использование класса. 1 маркер



Использование класса. Заданная максимальная длина

Разработка интерфейса пользователя

В отдельном файле пользователь заполняет массив, причем первая строка содержит обозначения двух маркеров для первого файла и маркер с числом для второго файла. А дальше в каждой строке записывается последовательность символов.

Для первого варианта входного файла

Структура

Marker marker2

Sssss… marker …sssss… marker2 …sssss

Класс

Marker

Sssss… marker …sssss

Для второго варианта входного файла

Структура

Marker

N

Sssss… marker …sssss

Класс

N

Sssss… …sssss

Первые строки в новом файле содержат информацию о студенте и о самой программе:

Карпенко Анастасия гр. 3353

59. Переставить последовательно в строке подходящие слова следующим образом:

в) по последнему символу слова найти к нему ближайшее справа, начинающееся с этого же символа, провести слияние слов

измененная строка:

Организация ввода-вывода

Для ввода из файла используем библиотеку fstream

(название потока).get(имя переменной);

Для вывода в файл используем библиотеку fstream

(название потока)<<«Текст»<<(имя переменной);

Так же стоит упомянуть, что для работы с файлами потребуется о открывать потоки вывода и открыть файлы, а так же:

fstream (название потока) ;

(название потока).open(«(Имя файла)», ios::(in для ввода, out для ввывода и app для дополнения));

(название потока)..close();

Организация хранения данных

При использовании структуры

Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
isWordFirstCharIndex	Проверка, является ли символ началом слова по переданному индексу	MarkerString string, int index	-	bool	-

findNextSameCharIndex	Поиск индекса следующего символа с тем же значением	MarkerString string, int index	nextSameCharIndex	int	-
findWordLastCharIndex	Поиск индекса последнего символа слова	MarkerString string, int from = 0	lastWordCharIndex - 1	int	-
joinWords	Объединение слов перед маркером	MarkerString string, int prevWordLastCharIndex, int nextWordFirstCharIndex	prevWordLastCharIndex - 1	int	-
readMarkerStringWithTwoSymbols	Чтение строки из файла с двумя символами маркера	fstream &input	{chars, marker}	-	-
readMarkerStringWithLimit	Чтение строки из файла с ограничением на длину	fstream &input	{chars, marker}	-	-
joinAllWordsBeforeMarker	Объединение всех слов перед маркером	MarkerString string	-	void	-
readFromFile	Чтение строки из файла в зависимости от выбранной стратегии	fstream &input, int strategy = 0	-	-	-
print	Вывод строки в файл	MarkerString string, ofstream &output	-	void	Вывод в файл

Структура для представления строки

```
struct MarkerString {
    char *chars; // Массив символов строки
    char marker; // Маркер
    // Конструктор
    MarkerString(const char *chars, char marker) {
        this->chars = new char[MAX_LENGTH]; // Выделение памяти под массив символов
        int index = 0;
        // Копирование символов из переданного массива в новый
        while (chars[index] != NULL_CHAR) {
            this->chars[index] = chars[index];
            index++;
        }
        this->chars[index] = NULL_CHAR; // Установка завершающего символа
       this->marker = marker; // Установка маркера
    }
};
```

При работе с классом используются те же функции, но во входных параметрах не передается MarkerString string так как теперь все действия происходят в class MarkerString. При этом часть функций становиться закрытым типом, а часть открытым. Программа приобретает следующий вид.

private:

```
char *chars;  // указатель на массив символов
int length; //длина строки или индекс маркета
```

Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
isWordFirstCharIndex	Проверка, является ли символ началом слова по переданному индексу	int index	-	bool	-
findNextSameCharIndex	Поиск индекса следующего символа с тем же значением	int index	nextSameCharIndex	int	-
findWordLastCharIndex	Поиск индекса последнего символа слова	int from = 0	lastWordCharIndex - 1	int	-
joinWords	Объединение слов перед маркером	int prevWordLastCharIndex, int nextWordFirstCharIndex	prevWordLastCharIndex - 1	int	-
readMarkerStringWithTwoSymbols	Чтение строки из файла с двумя символами маркера	fstream &input	-	-	-
readMarkerStringWithLimit	Чтение строки из файла с ограничением на длину	fstream &input	-	-	-

public:

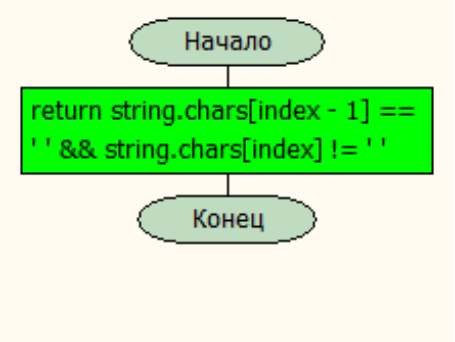
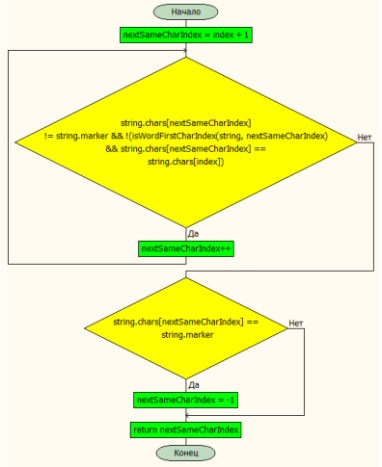
```
// Конструктор класса с параметрами, определяющий стратегию чтения строки с маркером
MarkerString(fstream &input, int strategy = 0) {
    if (strategy == 0) readMarkerStringWithSymbol(input);
    else readMarkerStringWithLimit(input);
}
```

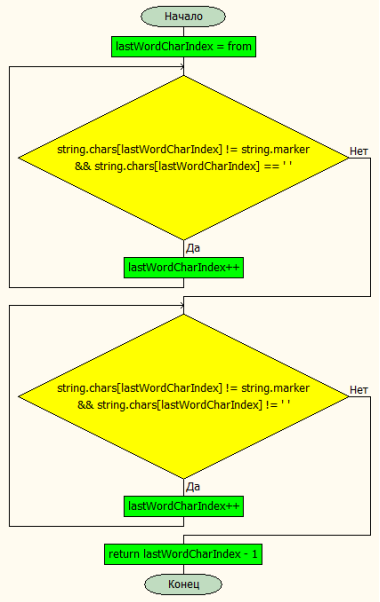
joinAllWordsBeforeMarker	Объединение всех слов перед маркером	-	-	void	-
print	Вывод строки в файл	ofstream &output	-	void	Вывод в файл

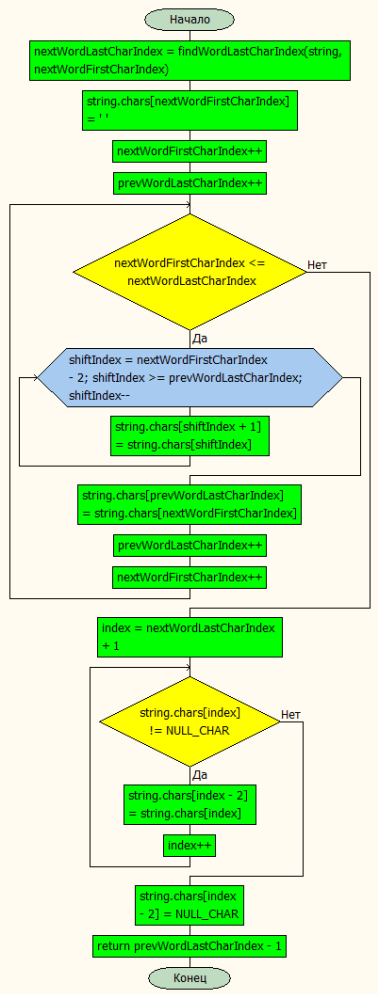
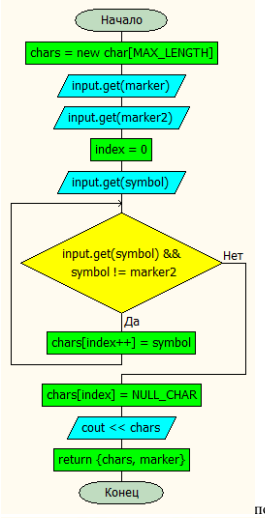
```
// Деструктор класса
~MarkerString() {
    delete[] chars;
}
```

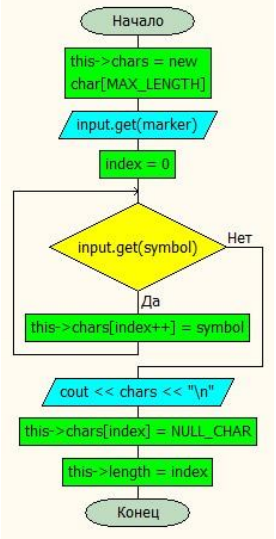
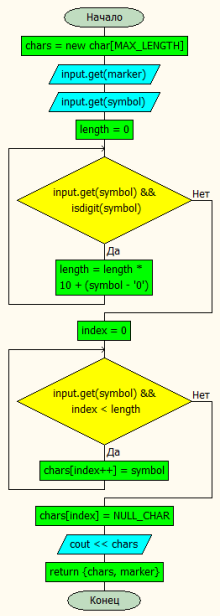
Описание алгоритма

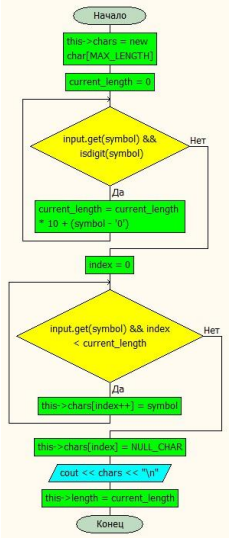
Отличие заключается в том, что при использовании структуры мы обращаемся к string.chars а при использовании класса просто к chars. string.marker становится length. обращение к массиву char* chars становится this->chars

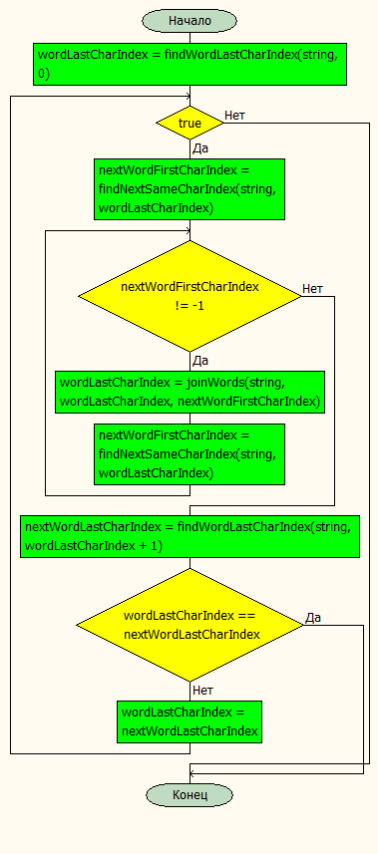
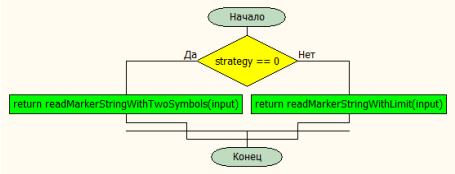
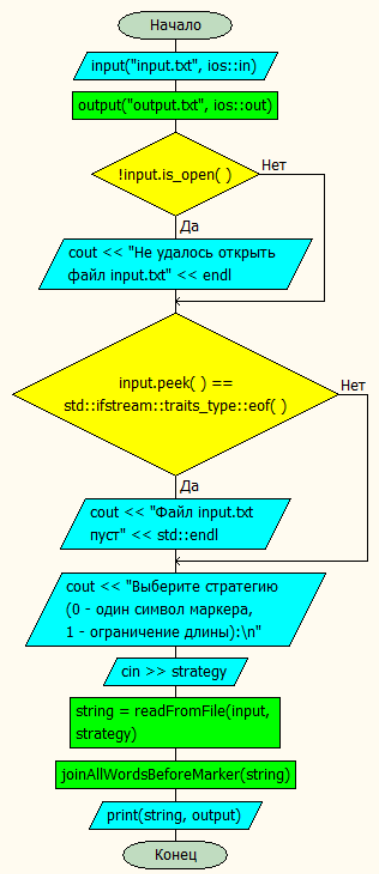
<table>
<tr><td>isWordFirstCharIndex</td><td>findNextSameCharIndex</td><td>findWordLastCharIndex</td></tr>
<tr><td> 
</td><td> 

При работе с классом string.chars[nextSameCharIndex] изменяется на nextSameCharIndex</td><td> 

string.chars[lastWordCharIndex] изменяется на lastWordCharIndex</td></tr>
<tr><td>joinWords</td><td>readMarkerStringWithTwoSymbols</td><td>readMarkerStringWithLimit</td></tr>
<tr><td> 
</td><td> 

подобная схема для класса

</td><td> 


</td></tr>
</table>

joinAllWordsBeforeMarker	readFromFile	main
		

Текст программы

Использование структуры

```
#include <iostream>
#include <fstream>
using namespace std;
const int MAX_LENGTH = 512;
const char NULL_CHAR = '\0';
// Структура для хранения строки с маркером
struct MarkerString {
    char *chars;  // массив символов
    char marker;  // маркер
    // Конструктор для инициализации строки с маркером
    MarkerString(const char *chars, char marker) {
        this->chars = new char[MAX_LENGTH];  // выделение памяти под массив символов
        int index = 0;
        // копирование символов из переданной строки
        while (chars[index] != NULL_CHAR) {
            this->chars[index] = chars[index];
            index++;
        }
        this->chars[index] = NULL_CHAR;  // установка завершающего нулевого символа
        this->marker = marker;  // инициализация маркера
    }
};
// Проверка, является ли символ первым в слове
bool isWordFirstCharIndex(MarkerString string, int index) {
    return string.chars[index - 1] == ' ' && string.chars[index] != ' ';
}
// Поиск индекса следующего встреченного символа, равного символу по заданному индексу
int findNextSameCharIndex(MarkerString string, int index) {
    int nextSameCharIndex = index + 1;

    // Поиск следующего символа такого же, как символ по заданному индексу
    while (string.chars[nextSameCharIndex] != string.marker &&
        !(isWordFirstCharIndex(string, nextSameCharIndex) && string.chars[nextSameCharIndex] == string.chars[index])) {
        nextSameCharIndex++;
    }
    if (string.chars[nextSameCharIndex] == string.marker) nextSameCharIndex = -1;
    return nextSameCharIndex;
}

// Поиск индекса последнего символа слова, начиная с заданного индекса
int findWordLastCharIndex(MarkerString string, int from = 0) {
    int lastWordCharIndex = from;
    // Пропуск пробелов
    while (string.chars[lastWordCharIndex] != string.marker && string.chars[lastWordCharIndex] == ' ') lastWordCharIndex++;
    // Поиск конца слова
    while (string.chars[lastWordCharIndex] != string.marker && string.chars[lastWordCharIndex] != ' ') lastWordCharIndex++;
    return lastWordCharIndex - 1;
}

// Объединение слов перед маркером
int joinWords(MarkerString string, int prevWordLastCharIndex, int nextWordFirstCharIndex) {
```

```
    int nextWordLastCharIndex = findWordLastCharIndex(string,nextWordFirstCharIndex);

    // Вставка пробела перед следующим словом
    string.chars[nextWordFirstCharIndex] = ' ';
    nextWordFirstCharIndex++;
    prevWordLastCharIndex++;

    // Перемещение символов следующего слова к текущему
    while (nextWordFirstCharIndex <= nextWordLastCharIndex) {
        for (int shiftIndex = nextWordFirstCharIndex - 2; shiftIndex >= prevWordLastCharIndex; shiftIndex--)
            string.chars[shiftIndex + 1] = string.chars[shiftIndex];
        string.chars[prevWordLastCharIndex] = string.chars[nextWordFirstCharIndex];

        prevWordLastCharIndex++;
        nextWordFirstCharIndex++;
    }

    // Удаление повторного вхождения следующего слова
    int index = nextWordLastCharIndex + 1;
    while (string.chars[index] != NULL_CHAR) {
        string.chars[index - 2] = string.chars[index];
        index++;
    }
    string.chars[index - 2] = NULL_CHAR;
    return prevWordLastCharIndex - 1;
}

// Чтение строки с маркером, использующей два символа для маркировки
MarkerString readMarkerStringWithTwoSymbols(fstream &input) {
    char* chars = new char[MAX_LENGTH];
    char marker;
    input.get(marker);
    char marker2;
    input.get(marker2);

    int index = 0;
    char symbol;
    input.get(symbol);
    while (input.get(symbol) && symbol != marker2) chars[index++] = symbol;
    chars[index] = NULL_CHAR;

    return {chars, marker};
}

// Чтение строки с маркером, использующей ограничение по длине
MarkerString readMarkerStringWithLimit(fstream &input) {
    char* chars = new char[MAX_LENGTH];
    char marker;
    input.get(marker);

    char symbol;
    input.get(symbol);
    size_t length = 0;
    while (input.get(symbol) && isdigit(symbol)) length = length * 10 + (symbol - '0');

    input.get(symbol);
    int index = 0;
    while (input.get(symbol) && index < length) chars[index++] = symbol;
    chars[index] = NULL_CHAR;
    return {chars, NULL_CHAR};
}

// Объединение всех слов перед маркером
void joinAllWordsBeforeMarker(MarkerString string) {
    int wordLastCharIndex = findWordLastCharIndex(string, 0);
    while (true) {
        int nextWordFirstCharIndex = findNextSameCharIndex(string, wordLastCharIndex);
        while (nextWordFirstCharIndex != -1) {
            wordLastCharIndex = joinWords(string, wordLastCharIndex, nextWordFirstCharIndex);
            nextWordFirstCharIndex = findNextSameCharIndex(string, wordLastCharIndex);
        }

        int nextWordLastCharIndex = findWordLastCharIndex(string, wordLastCharIndex + 1);
        if (wordLastCharIndex == nextWordLastCharIndex) break;
        else wordLastCharIndex = nextWordLastCharIndex;
    }
}

// Чтение строки с маркером из файла с учетом выбранной стратегии
MarkerString readFromFile(fstream &input, int strategy = 0) {
    if (strategy == 0) return readMarkerStringWithTwoSymbols(input);
    else return readMarkerStringWithLimit(input);
}

// Вывод строки на консоль
void print(MarkerString string, ofstream &output) {
    output << "Карпенко Анастасия гр. 3353" << "\n";
    output << "59. Переставить последовательно в строке подходящие слова следующим образом:" << "\n";
    output << "в) по последнему символу слова найти к нему ближайшее справа, начинающееся с этого же символа," << "\n";
    output << "провести слияние слов" << "\n";
    output << "\n";
    output << "измененная строка:" << endl;
    output << string.chars << endl;
}


int main() {
    fstream input("input.txt", ios::in);  // открытие файла для чтения
    ofstream output("output.txt", ios::out);  // открытие файла для записи

    if (!input.is_open()) {
        output << "Не удалось открыть файл input.txt" << endl;
        throw std::runtime_error("Ошибка открытия файла");
    }
    if (input.peek() == std::ifstream::traits_type::eof()) {
        output << "Файл input.txt пуст" << std::endl;
        throw std::runtime_error("Входной файл пуст");
    }
    cout << "Choose a strategy (0 - two marker characters, 1 - length limit):\n";
    bool strategy;
    cin >> strategy;
    MarkerString string = readFromFile(input, strategy);  // чтение строки из файла с учетом выбранной стратегии
    joinAllWordsBeforeMarker(string);  // объединение слов перед маркером
    print(string, output);  // вывод измененной строки в файл
```

```
}
```

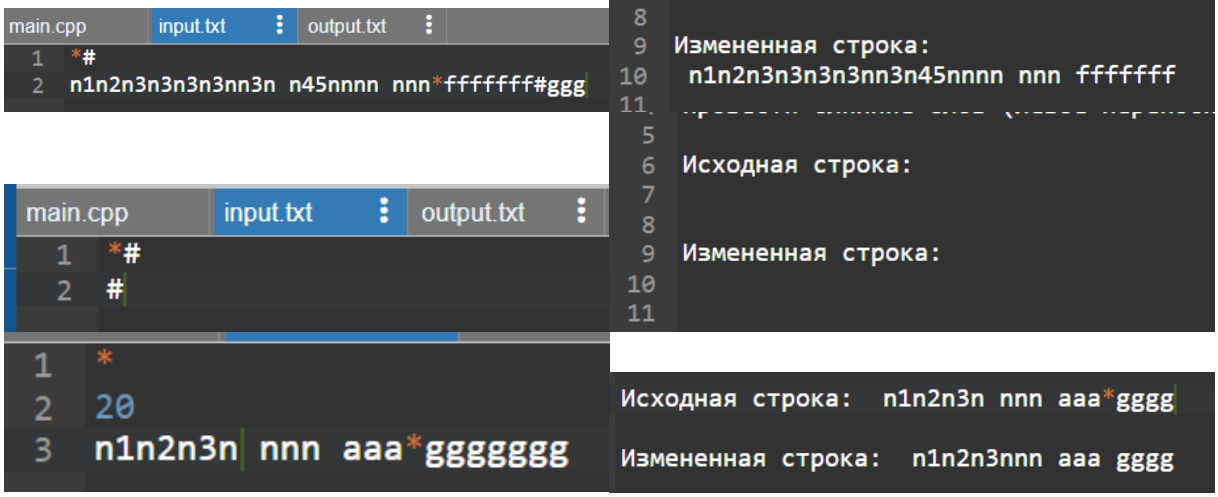
Использование класса

```
#include <iostream>
#include <fstream>
using namespace std;
const int MAX_LENGTH = 512;
const char NULL_CHAR = '\0';
// Класс для работы с строкой, содержащей маркер
class MarkerString {
private:
    char *chars;  // указатель на массив символов
    int length; //длина строки или индекс маркета
    // Приватный метод для определения, является ли символ первым в слове
    bool isWordFirstCharIndex(int index) {
        return chars[index - 1] == ' ' && chars[index] != ' ';}
    // Приватный метод для поиска индекса следующего встреченного символа, равного символу по заданному индексу
    int findNextSameCharIndex(int index) {
        int nextSameCharIndex = index + 1;

        while (nextSameCharIndex != length &&
            !(isWordFirstCharIndex(nextSameCharIndex) && chars[nextSameCharIndex] == chars[index])) {
            nextSameCharIndex++;
        }
        if (nextSameCharIndex == length) nextSameCharIndex = -1;
        return nextSameCharIndex;
    }
    // Приватный метод для поиска индекса последнего символа слова, начиная с заданного индекса
    int findWordLastCharIndex(int from = 0) {
        int lastWordCharIndex = from;
        // Пропустить пробелы
        while (lastWordCharIndex != length && chars[lastWordCharIndex] == ' ') lastWordCharIndex++;
        // Найти конец слова
        while (lastWordCharIndex != length && chars[lastWordCharIndex] != ' ') lastWordCharIndex++;
        return lastWordCharIndex - 1;
    }
    // Приватный метод для объединения слов
    int joinWords(int prevWordLastCharIndex, int nextWordFirstCharIndex) {
        int nextWordLastCharIndex = findWordLastCharIndex(nextWordFirstCharIndex);

        chars[nextWordFirstCharIndex] = ' ';
        nextWordFirstCharIndex++;
        prevWordLastCharIndex++;

        while (nextWordFirstCharIndex <= nextWordLastCharIndex) {
            for (int shiftIndex = nextWordFirstCharIndex - 2; shiftIndex >= prevWordLastCharIndex; shiftIndex--)
                chars[shiftIndex + 1] = chars[shiftIndex];
            chars[prevWordLastCharIndex] = chars[nextWordFirstCharIndex];

            prevWordLastCharIndex++;
            nextWordFirstCharIndex++;
        }

        int index = nextWordLastCharIndex + 1;
        while (chars[index] != NULL_CHAR) {
            chars[index - 2] = chars[index];
            index++;
        }
        chars[index - 2] = NULL_CHAR;
        return prevWordLastCharIndex - 1;
    }
    // Приватный метод для чтения строки с маркером, использующей один символ маркера
    void readMarkerStringWithSymbol(fstream &input) {
        this->chars = new char[MAX_LENGTH];
        char marker;
        input.get(marker);

        int index = 0;
        char symbol;
        while (input.get(symbol)) this->chars[index++] = symbol;
        cout<<chars<<"\n";
        this->chars[index] = NULL_CHAR;
        this->length = index;
    }
    // Приватный метод для чтения строки с маркером, использующей ограничение по длине
    void readMarkerStringWithLimit(fstream &input) {
        this->chars = new char[MAX_LENGTH];

        char symbol;
        int current_length = 0;
        while (input.get(symbol) && isdigit(symbol)) current_length = current_length * 10 + (symbol - '0');

        int index = 0;
        while (input.get(symbol) && index < current_length) this->chars[index++] = symbol;
        this->chars[index] = NULL_CHAR;
        cout<<chars<<"\n";
        this->length = current_length;
    }
public:
    // Конструктор класса с параметрами, определяющий стратегию чтения строки с маркером
    MarkerString(fstream &input, int strategy = 0) {
        if (strategy == 0) readMarkerStringWithSymbol(input);
        else readMarkerStringWithLimit(input);
    }
    // Деструктор класса
    ~MarkerString() {
        delete[] chars;
    }
    // Метод для получения символа по указанному индексу
    char chatAt(int index) {
        return chars[index];
    }
    // Метод для объединения всех слов перед маркером
    void joinAllWordsBeforeMarker() {
        int wordLastCharIndex = findWordLastCharIndex(0);
        while (true) {
            int nextWordFirstCharIndex = findNextSameCharIndex(wordLastCharIndex);
            while (nextWordFirstCharIndex != -1) {
                wordLastCharIndex = joinWords(wordLastCharIndex, nextWordFirstCharIndex);
```

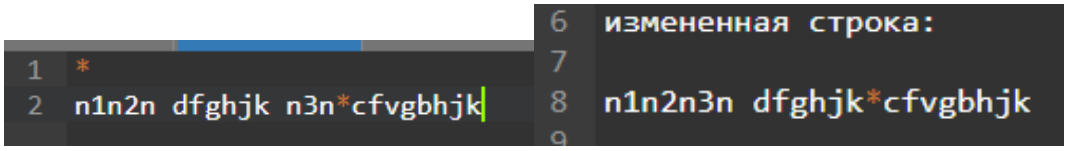
```
            nextWordFirstCharIndex = findNextSameCharIndex(wordLastCharIndex);
        }

        int nextWordLastCharIndex = findWordLastCharIndex(wordLastCharIndex + 1);
        if (wordLastCharIndex == nextWordLastCharIndex) break;
        else wordLastCharIndex = nextWordLastCharIndex;
      }
    }
    // Метод для вывода строки на консоль
    void print(ofstream &output) const {
      output << "Карпенко Анастасия гр. 3353" << "\n";
      output << "59. Переставить последовательно в строке подходящие слова следующим образом:" << "\n";
      output << "в) по последнему символу слова найти к нему ближайшее справа, начинающееся с этого же символа," << "\n";
      output << "провести слияние слов" << "\n";
      output << "\n";
      output << "измененная строка:" << endl;
      output << chars << endl;
    }
};
// Точка входа в программу
int main() {
    fstream input("input.txt", ios::in);  // открытие файла для чтения
    ofstream output("output.txt", ios::out);  // открытие файла для записи

    // Проверка успешности открытия файла для чтения
    if (!input.is_open()) {
        output << "Не удалось открыть файл input.txt" << endl;
    }

    // Проверка, является ли файл пустым
    if (input.peek() == std::ifstream::traits_type::eof()) {
        output << "Файл input.txt пуст" << std::endl;
    }

    cout << "Choose a strategy (0 - one marker character, 1 - length limit):\n";
    bool strategy;
    cin >> strategy;

    // Создание объекта класса MarkerString с передачей стратегии чтения
    MarkerString string(input, strategy);
    string.joinAllWordsBeforeMarker();  // Объединение слов перед маркером
    string.print(output);  // Вывод измененной строки в файл
}
```

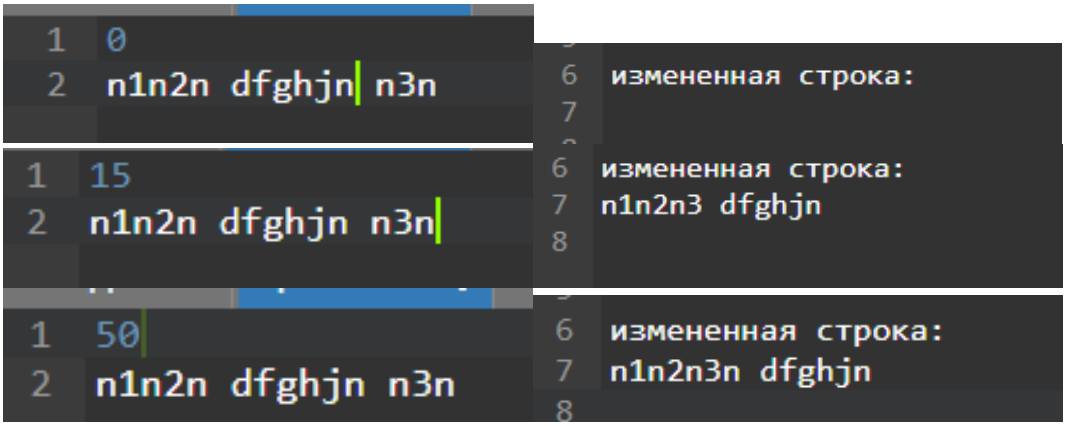
Результаты работы программы

Использование структуры



Использование класса





Вывод

В ходе работы была изучена организация информации с помощью структуры и класса. Была проведена работа с функциями, были использованы маркеры для контроля считывания и обработки информации из файла.