**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ФКТИ**

отчет

**по лабораторной работе №1**

**по дисциплине «Программирование»**

Тема: Представление строки символов с маркером, её обработка

|  |  |  |
| --- | --- | --- |
| Студент гр. 4335 |  | Иванов Г.Д. |
| Преподаватель |  | Калмычков В.А. |

Санкт-Петербург

2025

**Исходная формулировка задания**

Дана строка, состоящая из нескольких слов, разделенных пробелами. Удалить все символы, не являющиеся буквами, в слове под номером N

**Структура представления данных.**

Для хранения строковых типов используется класс marker\_string:

class marker\_string{

char \* \_data; -- указатель на начало хранимой строки

char \_marker; -- символ, обозначающий конец строки

};

**Формат входного и выходного файлов.**

1. Входной файл: lab1.in:

Первая строка – исходная строка для обработки

Вторая строка – номер слова, из которого удаляются лишние символы

1. Выходной файл: lab1.out:

Единственная строка – результат обработки

Пример входного файла:

Hello world 123123hahaha !!!

3

Пример выходного файла:

Hello world hahaha !!!

**Реализация типов данный и функций:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя модуля (файла)** | **Имя структуры, класса или функции/метода** | **Назначение** | **Входные параметры** | **Возвращаемое значение** |
| main.cpp | marker\_string() | Конструктор по умолчанию. Инициализирует указатель \_data как nullptr и маркер \_marker символом '$' | - | - |
| main.cpp | marker\_string(const char\* cstr) | Конструктор из C-строки. Выделяет память, копирует строку, добавляет маркер в конец. | const char \* - указатель на  с-style строку | - |
| main.cpp | marker\_string(const std::string& str) | Конструктор из std::string. Аналогичен предыдущему, но принимает объект std::string. | const std::stirng & - строка по ссылке | - |
| main.cpp | marker\_string(const marker\_string& other) | Конструктор копирования. Создаёт глубокую копию другого объекта marker\_string. | Const marker\_string & - объект для копирования | - |
| main.cpp | ~marker\_string() | Деструктор. Освобождает динамически выделенную память. | - | - |
| main.cpp | marker\_string::size() const | Возвращает длину строки (до маркера). | - | int |
| main.cpp | marker\_string::clear() | Освобождает память, выделенную под \_data. | - | void |
| main.cpp | marker\_string::data() | Возвращает указатель на данные. | - | char\* |
| main.cpp | marker\_string::marker() | Возвращает символ-маркер окончания строки. | - | char |
| main.cpp | operator << | Заносит строку в структуре marker\_string в поток вывода | std::basic\_ostream<char>&,  const marker\_string & | std::basic\_ostream<char> & |
| main.cpp | remove\_special\_symbols\_from\_word\_n | Удаляет все небуквенныесимволы из n-го слова строки in. Остальная часть строки остается неизменной. | const marker\_string &, int n | marker\_string |
| main.cpp | main | Точка входа в программу | - | int |

**Алгоритм работы обработки строки**

1. Инициализация
   1. Копирует строку in в out (результат работы).
   2. i = 0 – индекс позиции в строке out
   3. current\_is\_word = <Первый символ строки in не пробел>
   4. word\_count = <1 если current\_is\_word иначе 0>
2. Цикл по символам

Для каждого символа \*ch:

* 1. Определяем границы слов:
     1. Если \*ch — не пробел, а предыдущий был пробел → word\_count++.
     2. Если \*ch — пробел → current\_is\_word = false.
  2. Обработка символов:
     1. Пробелы копируются в out без изменений.
     2. Для n-ного слова:
        1. Пропускаем все символы, кроме латинских букв (a-z, A-Z).
     3. Остальные символы копируются без изменений

1. Добавление маркера в out на позицию i.
2. Возвращение значения.

Изображение выглядит как зарисовка, диаграмма, рисунок, Штриховая графика

Содержимое, созданное искусственным интеллектом, может быть неверным.

Диаграмма 1. Алгоритм работы функции remove\_special\_symbols\_from\_word\_n

Изображение выглядит как зарисовка, диаграмма, рисунок, дизайн

Автоматически созданное описание

Диаграмма 2. Алгоритм работы функции main

**Примеры работы программы.**

1. Обычный случай:  
   Входной файл:   
   Hello world 123123hahaha !!! :D  
   3  
   Выходной файл:  
   Hello world hahaha !!! :D
2. Удаляется единственное слово целиком:

Входной файл:   
123123  
1  
Выходной файл:

1. Нет слова с номером N:

Входной файл:   
Hello there!  
15  
Выходной файл:

Hello there!

1. Пустая исходная строка:

Входной файл:   
  
1  
Выходной файл:

**Визуализация внутреннего представления**

Структура класса marker\_string в памяти:



Исходный текст программы.

#include <string>  
#include <iostream>  
#include <fstream>  
  
class marker\_string {  
 char \* \_data;  
 char \_marker;  
  
 void m\_init() {  
 \_data = nullptr;  
 \_marker = '$';  
 }  
  
public:  
 marker\_string() : \_data{ nullptr }, \_marker{ '$' } {}  
 marker\_string(const char \* cstr) : marker\_string() {  
 auto len = strlen(cstr);  
 \_data = new char [len + 1];  
 memcpy(\_data, cstr, len);  
 \_data[len] = \_marker;  
 }  
 marker\_string(const std::string & str) : marker\_string() {  
 auto len = str.size();  
 \_data = new char [len + 1];  
 memcpy(\_data, str.c\_str(), len);  
 \_data[len] = \_marker;  
 }  
 marker\_string(const marker\_string & other) :  
 \_marker{other.\_marker}  
 {  
 int len = other.size();  
 \_data = new char [len + 1];  
 \_data[len] = \_marker;  
 for (int i = 0; i < len; i++)  
 \_data[i] = other.\_data[i];  
 }  
 ~marker\_string() {  
 clear();  
 }  
  
 int size () const {  
 int out = 0;  
 auto ch = data();  
 while (\*ch++ != \_marker) out += 1;  
 return out;  
 }  
 void clear () { if (\_data) delete [] \_data; }  
 char \* data () const { return \_data; }  
 char marker () const { return \_marker; }  
};  
  
inline std::basic\_ostream<char> & operator << (std::basic\_ostream<char> & stream, const marker\_string & str) {  
 auto ch = str.data();  
 while (\*ch != str.marker())  
 stream << \*ch++;  
 return stream;  
}  
  
marker\_string remove\_special\_symbols\_from\_word\_n(const marker\_string & in, int n) {  
 marker\_string out {in};  
 int i = 0;  
 bool current\_is\_word = (\*in.data() != ' ');  
 int word\_count = current\_is\_word ? 1 : 0;  
 for (char \* ch = in.data(); \*ch != in.marker(); ch++) {  
 if (\*ch != ' ' && (current\_is\_word == false)) {  
 word\_count++;  
 current\_is\_word = true;  
 }  
 else if (\*ch == ' ') current\_is\_word = false;  
 else current\_is\_word = true;  
  
 if (\*ch == ' ') {  
 out.data()[i++] = \*ch;  
 continue;  
 }  
  
 if (word\_count == n) {  
 if ((\*ch >= 'a' && \*ch < 'z') || (\*ch >= 'A' && \*ch <= 'Z')) {  
 out.data()[i++] = \*ch;  
 }  
 }  
 else {  
 out.data()[i++] = \*ch;  
 }  
  
 }  
 out.data()[i] = out.marker();  
 return out;  
}  
  
int main (int argc, char \* argv []) {  
 std::ifstream fin ("lab1.in");  
 if (!fin) {  
 std::cerr << "Cannot open input file 'lab1.in'!";  
 return 1;  
 }  
 int n = 0;  
 std::string line;  
 std::getline(fin, line);  
 fin >> n;  
  
 fin.close();  
  
 marker\_string str (line);  
 marker\_string res = remove\_special\_symbols\_from\_word\_n(str, n);  
  
 std::cout << "Result: " << res << std::endl;  
  
 std::ofstream fout ("lab1.out");  
 if (!fout) {  
 std::cerr << "Cannot open output file 'lab1.out'!";  
 return 2;  
 }  
 fout << res << std::endl;  
 fout.close();  
 return 0;  
}

Вывод.

В ходе лабораторной работы была реализована программа для удаления небуквенных символов из заданного слова строки с использованием пользовательской структуры marker\_string. Реализация включала ручное управление памятью и работу с символами на низком уровне. Программа успешно обрабатывает различные входные данные и корректно формирует результат. Работа позволила закрепить навыки работы со строками, динамической памятью и пользовательскими типами данных.