**Санкт-Петербургский политехнический университет Петра Великого**

**Институт компьютерных наук и технологий Высшая школа программной инженерии**

Изображение выглядит как текст

Автоматически созданное описание

**КУРСОВАЯ РАБОТА**

**по дисциплине «Технологии программирования»**

Выполнил   
студент гр.

Руководитель Шемякин И.А

«18» \_\_\_\_05\_\_\_\_\_\_ 2022

Санкт-Петербург

2022 г

**Содержание**

[Введение. Общая постановка задачи: 3](#_Toc103788231)

[Описание получившейся программы и совершаемых в ней действий. 4](#_Toc103788232)

[Описание тестирования 5](#_Toc103788233)

[Заключение 6](#_Toc103788234)

[Исходный код программы 7](#_Toc103788235)

# Введение. Общая постановка задачи:

**Вариант 1.2.1**

**Перекрестные ссылки. Бинарное дерево поиска (через контейнер)**

Разработать и реализовать алгоритм который способен:

* прочитать текст и вывести его с добавлением последовательных номеров строк;
* собрать все слова, встречающиеся в тексте;
* сформировать таблицу, в которой все слова будут расположены в алфавитном порядке и для каждого слова будет указан список строк его нахождения (по возрастанию номеров  
  строк)

# Описание получившейся программы и совершаемых в ней действий.

Для решения данного задания я использовал такие структуры данных как:

* вектор

Алгоритм решения был следующий: из указанного потока ввода (файла) считываются строчки при помощи **std::copy.**Оператор **operator >>** считывает строчку в входными данными при помощи **std::getline.** Затем мы считываем строку, и увеличиваем счетчик введенных строк **global\_number.** В самом цикле “длинны строки” заведем условие “пока не пробел или не последний индекс элемента” мы записываем в элемент **str0.** После чего помещаем каждое слово и номер строки, где находится само слово элемент типа **Element.** И при помощи std::copy таким образом заполняем вектор **std::vector< Element > vector**

Класс **Element** содержит свой вектор, где хранятся слова в строке **line** и поле **line,** где хранится номер строки, где встречаются данные слова.

При помощи функции **outInfo** вывел информацию о доступных командах в std::cout.

Использовав цикл пока поток ввода не закончился (**while (!std::cin.eof()))**. При каждом проходе будем отчищать поток ввода при помощи **std::cin.clear()** чтобы не возникало ошибок. Далее воспользовавшись функцией **coutResult** в зависимости от введенных команд будет выводиться результат.

Сама функция **coutResult** принимает в себя поток ввода, **vector< Element>** и поток вывода. В **coutResult** идет проверка на введенную команду. Если команда не является доступной, то выбрасывается исключение, которое затем выводится в **std::cout**. Далее в зависимости от введенной команды выполняются действия. Если введено **GET LINES** то при помощи **std::copy** и перегруженного оператора побитового сдвига выведем строки с порядковыми номерами. Если введено **GET TABLE** то создастся вектор **std::vector< Table\_element> table,**  который далее заполнится исходным вектором **std::vector< Table\_element> table**, причем элемент **Table\_element** содержит в себе строку и вектор, в котором хранятся номера строк, где встречается это слово. После заполнения вектор сортируется при помощи **std::sort** и компаратора, который только сравнивает строки, так как длинна вектора нам не важна и повторяющихся элементов в **vector< Table\_element>** нет. Затем полученный вектор выводится, образуя таблицу, в которой сначала стоит слово, а затем номера строк, где встретилось это слово. При введении команды **INFO** выведется информация о доступных командах.

# Описание тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| Требование | Детальные требования | Данные | Ожидаемый результат |
| При вводе команды GET LINES вывести в стандартный поток пронумерованные строки | * Если в файле с входными данными не будет текста, то в дерево ничего не запишется. * Если данные будут присутствовать в файле, то на экран выведется номер строки и сама строка. Каждое слово будет добавлено в дерево, а значение строки, в которой находится, слово добавится в список к слову. | qwertyuiopassdhnsdtv asdf;nask asdbg  asdfghjkl 156851 18531 zxcvbnm asdbg 18531  zxcvbnm asdbg  iospdurh  l  smd[oib] | 1. qwertyuiopassdhnsdtv asdf;nask asdbg  2. asdfghjkl 156851 18531 zxcvbnm asdbg 18531  3. zxcvbnm asdbg  4. iospdurh  5. l  6. smd[oib] |
| При вводе команды GET TABLE вывести в стандартный поток отсортированную таблицу с элементами и номерами строк | * В случае, если одно и то же слово встречается в одной строке, то в таблице его место отметится как повторение номера его строки. | qwertyuiopassdhnsdtv asdf;nask asdbg  asdfghjkl 156851 18531 zxcvbnm asdbg 18531  zxcvbnm asdbg  iospdurh  l  smd[oib] | ------------------------------------  | Words |On which line |  ------------------------------------  | 156851 | 2  | 18531 | 2 2  | asdbg | 1 2 3  | asdf;nask | 1  | asdfghjkl | 2  | iospdurh | 4  | l | 5  | qwertyuiopassdhnsdtv | 1  | smd[oib] | 6  | zxcvbnm | 2 3  ------------------------------------ |

# Заключение

В конце работы мне удалось разработать и реализовать алгоритм работы с контейнерами. Предусмотреть обработку исключительных ситуаций, связанных, с пустой строкой и повторением одного и того же слова в одной строке. Удалось изучить концепцию перегрузки оператора побитового сдвига, работу с std::copy.

# Исходный код программы

Код main.cpp

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <iomanip>  #include "Element.hpp"  #include "Functions.hpp"  int main(int argc, char\* argv[])  {  std::ifstream input;  if (argc == 2)  {  input.open(argv[1]);  }  else  {  std::cerr << "<INVALID FILENAME>" << std::endl;  return 1;  }  std::istream\_iterator< Element > inputStart(input);  std::istream\_iterator< Element > inputEnd;  std::vector< Element > vector;  while (!input.eof())  {  input.clear();  std::copy(inputStart, inputEnd, std::back\_inserter(vector));  }  input.close();  outInfo(std::cout);  while (!std::cin.eof())  {    std::cin.clear();  try  {  coutResult(std::cin, vector, std::cout);  }  catch (std::exception& e)  {  std::cerr << e.what();  }  }  return 0;  } |

Код Element.hpp

|  |
| --- |
| #ifndef ELEMENT\_H  #define ELEMENT\_H  #include <vector>  #include <iostream>  struct Element  {  public:  size\_t line;  std::vector<std::string> words;  };  struct Table\_element  {  public:  std::string word;  std::vector<size\_t> lines;  };  std::istream& operator >> (std::istream& in, Element& data);  std::ostream& operator << (std::ostream& out, const Element& data);  #endif |

Код Element.сpp

|  |
| --- |
| #include "Element.hpp"  #include <iostream>  #include <string>  static size\_t global\_number = 0;  std::istream& operator >> (std::istream& in, Element& data)  {  data.words.clear();  std::string str = "";  std::string str0 = "";  std::getline(in, str);  global\_number++;  if (!str.size())  {  std::getline(in, str);  }  for (size\_t i = 0; i < str.size(); i++)  {  if (str[i] != ' ' and i != str.size() - 1)  str0 += str[i];  else  {  if (i == str.size() - 1)  str0 += str[i];  if (str0.size() != 0)  {  data.line = global\_number;  data.words.push\_back(str0);  }  str0 = "";  }  }  return in;  }  std::ostream& operator << (std::ostream& out, const Element& data)  {  out << data.line << ". ";  for (auto it = data.words.begin(); it != data.words.end(); ++it)  {  out << it->c\_str() << " ";  }  return out;  } |

Код Functions.hpp

|  |
| --- |
| #ifndef FUNCTIONS\_H  #define FUNCTIONS\_H  #include <iostream>  #include <vector>  #include "Element.hpp"  void coutResult(std::istream& in, const std::vector<Element>& vector, std::ostream& out);  void outInfo(std::ostream& out);  void getTable(std::ostream& out, const std::vector<Table\_element>& vector);  bool comparator(const Table\_element& lhs, const Table\_element& rhs);  #endif |

Код Functions.cpp

|  |
| --- |
| #include "Functions.hpp"  #include "Element.hpp"  #include <iostream>  #include <string>  #include <algorithm>  #include <vector>  #include <iomanip>  static size\_t maxWordSize = 0;  bool find(std::string str, std::vector< Table\_element>& table)  {  for (auto it = table.begin(); it != table.end(); ++it)  {  if (it->word == str)  return 1;  }  return 0;  }  bool comparator(const Table\_element& lhs, const Table\_element& rhs)  {  if (lhs.word != rhs.word)  {  return lhs.word < rhs.word;  }  }  void coutResult(std::istream& in, const std::vector<Element>& vector, std::ostream& out)  {  std::string str;  std::getline(in, str);  if (str == "GET LINES")  {  std::copy(vector.begin(), vector.end(), std::ostream\_iterator <Element>(out, "\n"));  }  else  {  if (str == "GET TABLE")  {  std::vector< Table\_element> table;    for (auto it = vector.begin(); it != vector.end(); ++it)  {  for (auto elem = it->words.begin(); elem != it->words.end(); ++elem)  {  if (!find(elem->c\_str(), table))  {  /\*table\_element.lines.clear();\*/  Table\_element table\_element;  table\_element.lines.push\_back(it->line);  table\_element.word = elem->c\_str();  maxWordSize = std::max(maxWordSize, table\_element.word.size());  table.push\_back(table\_element);    }  else  {  for (auto its = table.begin(); its != table.end(); ++its)  {  if (its->word == elem->c\_str())  {  its->lines.push\_back(it->line);  break;  }  }  }  }  }  std::sort(table.begin(), table.end(), comparator);  getTable(out, table);  }  else  {  if(str == "INFO")  outInfo(out);  else  throw std::logic\_error("<INVALID COMMAND>\n");  }  }  }  void outInfo(std::ostream& out)  {  out << "Write: 'GET LINES' - to get rows with their numbers\n"  << "Write: 'GET TABLE' - to get a sorted table with row numbers where words occur\n"  << "Write: 'INFO' - to get info about available commands\n";  }  void getTable(std::ostream& out, const std::vector<Table\_element>& vector)  {  for (size\_t i = 0; i < maxWordSize \* 2; i++)  out << "-";  out << "\n";  out << std::setw(maxWordSize + 1) << std::left << "| Words" << " |" << "On which line " << "\n";  for (size\_t i = 0; i < maxWordSize \* 2; i++)  out << "-";  out << "\n";  for (auto its = vector.begin(); its != vector.end(); ++its)  {  out << "| " << std::setw(maxWordSize + 1) << std::left << its->word << "| ";  std::copy(its->lines.begin(), its->lines.end(), std::ostream\_iterator <size\_t>(out, " "));  out << "\n";  }  for (size\_t i = 0; i < maxWordSize \* 2; i++)  out << "-";  out << "\n";  } |