

МИНОБРНАУКИ РОСИИ

*Федеральное государственное бюджетное образовательное учреждение высшего образования*

***«МИРЭА – Российский технологический университет»***

**РТУ МИРЭА**



Отчет по выполнению практического задания №5.2

**Тема:**

Алгоритмы поиска в таблице при работе с данными из файлаДисциплина: «Структуры и алгоритмы обработки данных»

|  |  |  |
| --- | --- | --- |
| Выполнил студент: Васильев Б.А.  Группа: ИКБО-20-23 |  |  |

Москва 2024

**СОДЕРЖАНИЕ**

[ЗАДАНИЕ 1 Создать двоичный файл из записей 3](#_Toc177418562)

[Формулировка задачи 3](#_Toc177418563)

[Описание подхода к решению 3](#_Toc177418564)

[Коды программы 4](#_Toc177418565)

[Результаты тестирования 9](#_Toc177418566)

[ЗАДАНИЕ 2 Поиск в файле с применением линейного поиска 10](#_Toc177418567)

[Формулировка задачи 10](#_Toc177418568)

[Алгоритм 10](#_Toc177418569)

[Код программы 11](#_Toc177418570)

[Результаты тестирования 12](#_Toc177418571)

[ЗАДАНИЕ 3 Поиск записи в файле с применением дополнительной структуры данных, сформированной в оперативной памяти. 14](#_Toc177418572)

[Формулировка задачи 14](#_Toc177418573)

[Алгоритм 15](#_Toc177418574)

[Код программы 16](#_Toc177418575)

[Результаты тестирования 19](#_Toc177418576)

[ВЫВОД 22](#_Toc177418577)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 22](#_Toc177418578)

**ЦЕЛЬ РАБОТЫ**

Получить практический опыт по применению алгоритмов поиска в таблице данных

# ЗАДАНИЕ 1 Создать двоичный файл из записей

### Формулировка задачи

Создать двоичный файл из записей, структура записи определена вариантом (рис. 1). Поле ключа записи в задании варианта подчеркнуто. Заполнить файл данными, используя для поля ключа датчик случайных чисел. Ключи записей в файле уникальны.

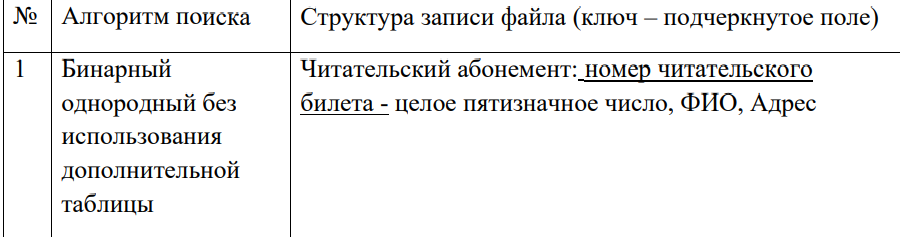


Рисунок 1 – Индивидуальный вариант задачи

### Описание подхода к решению

Создание файла из нужного количества записей можно разделить на два этапа: создание текстового файла, конвертация текстового файла в двоичный.

Структура записи файла определена вариантом (рис. 1). Размер двоичного представления записи в байтах состоит из суммы отведённого количества байтов на каждое из трёх полей. В данной работе на поле номера читательского билета было выделено 5 байт, на поля ФИО и адрес — по 55 байт. Итоговая длина записи равняется 115 байтам. Таким образом, для доступа к i-ой записи, достаточно перейти к байту с номером 115\*i.

### Коды программы

Реализуем алгоритм на языке программирования C++ (рис. 2-7)

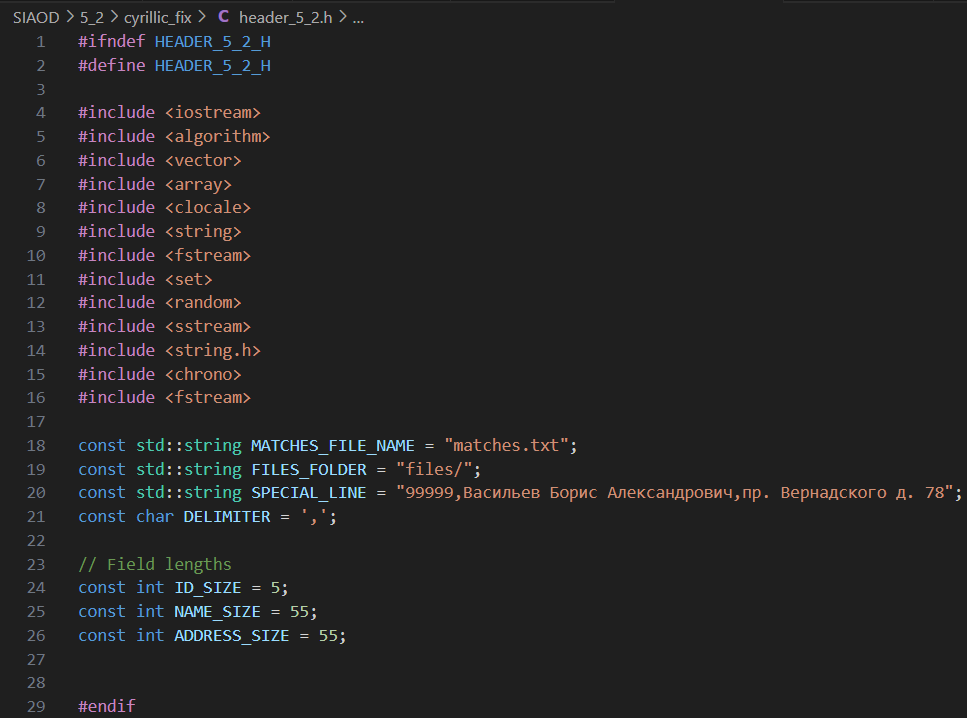


Рисунок 2 – Заголовочный файл общий для всех трёх заданий

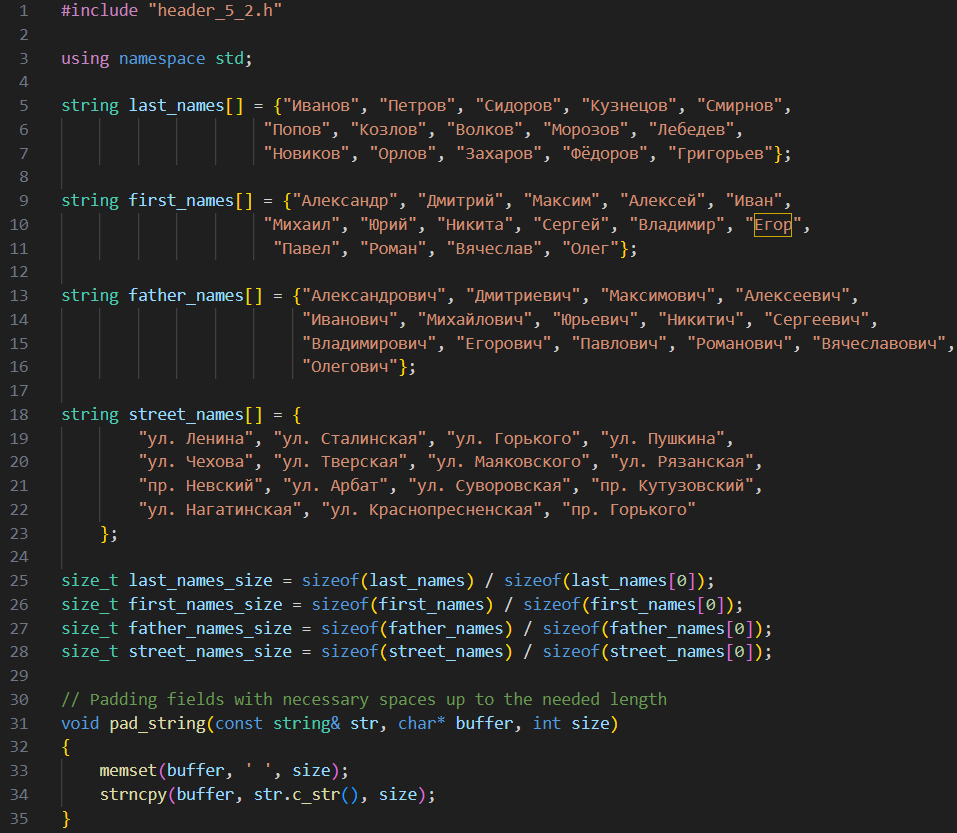


Рисунок 3 – Код к заданию 1 (часть 1)

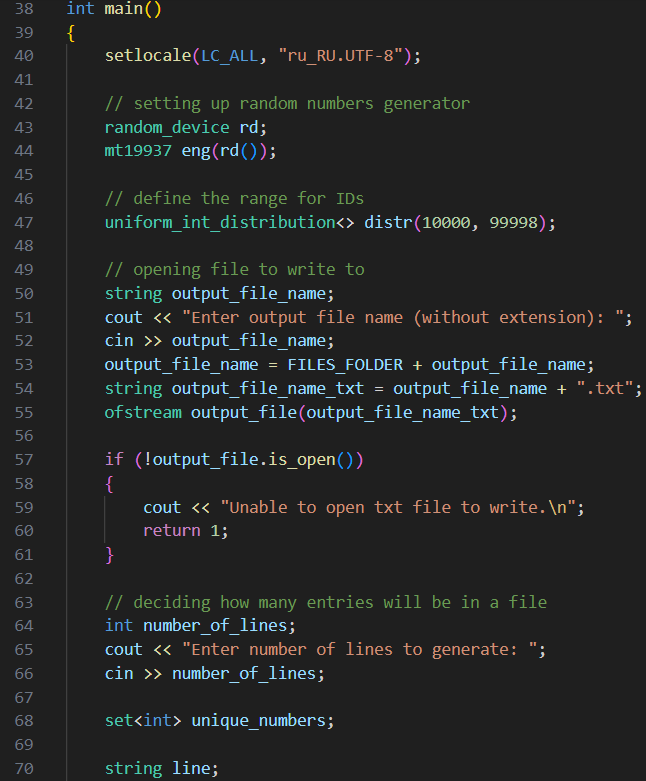


Рисунок 4 – Код к заданию 1 (часть 2)

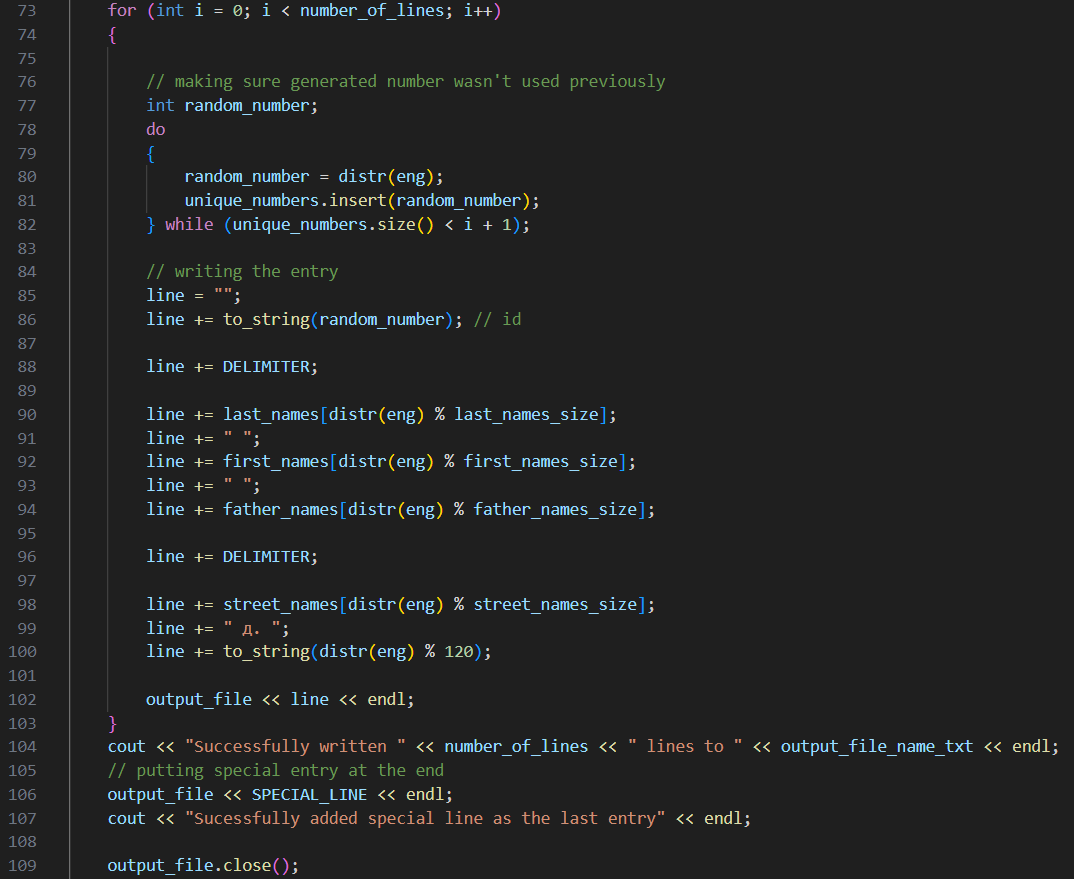


Рисунок 5 – Код к заданию 1 (часть 3)

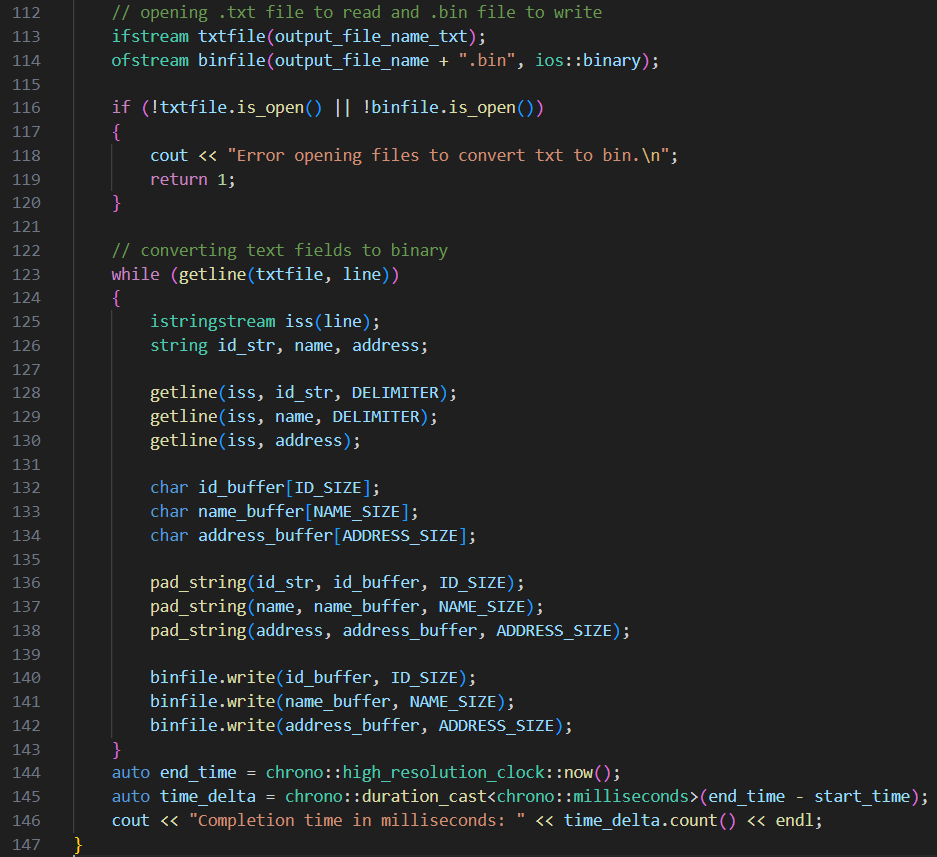


Рисунок 6 – Код к заданию 1 (часть 4)

### Результаты тестирования

Выполним тестирование программы для создания файла из 100 записей (рис. 7-9).

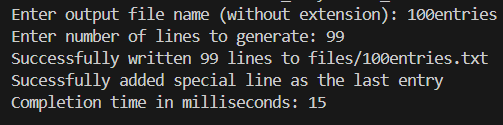
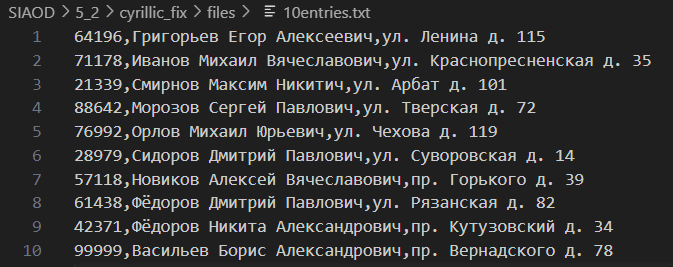
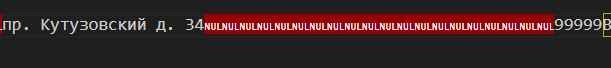


Рисунок 7 – Вывод программы к заданию 1 в консоль

  
Рисунок 8 – Созданный текстовый файл

  
Рисунок 9 – Пример последовательности символов из двоичного файла, просмотренных в кодировке UTF-8

Тестирование показало, что программа работает корректно.

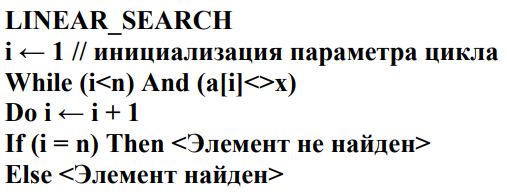
# ЗАДАНИЕ 2 Поиск в файле с применением линейного поиска

### Формулировка задачи

Разработать программу поиска записи по ключу в бинарном файле с применением алгоритма линейного поиска. Провести практическую оценку времени выполнения поиска на файле объемом 100, 1000, 10 000 записей. Составить таблицу с указанием результатов замера времени

### Алгоритм

Приведём алгоритм линейного поиска, написанный на псевдокоде (рис. 10).

  
Рисунок 10 – псевдокод алгоритма линейного поиска

### Код программы

Реализуем алгоритм на языке программирования C++ (рис. 11-12)

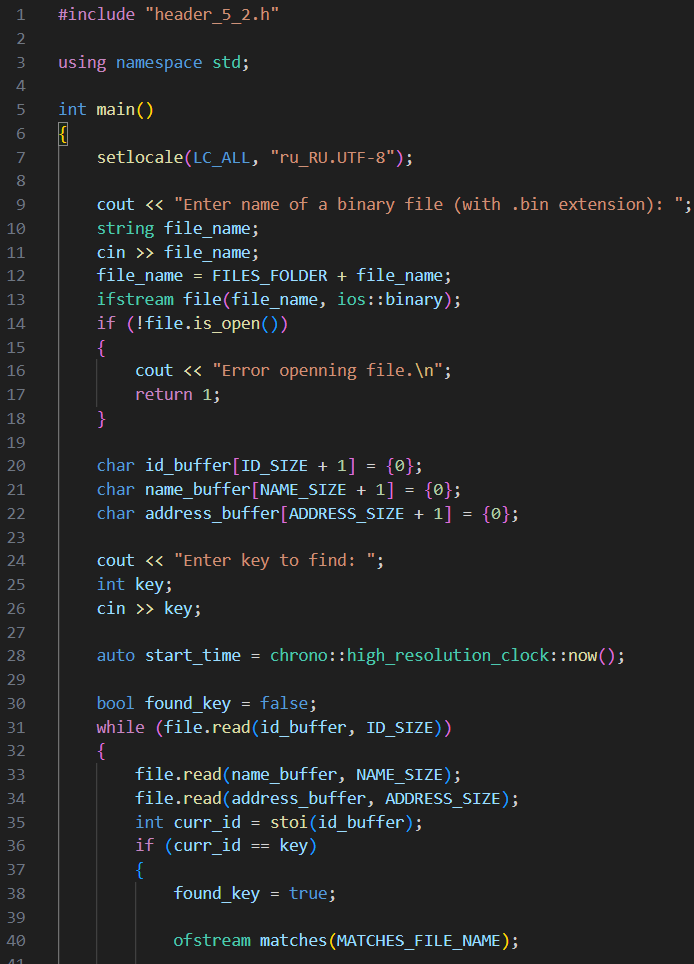


Рисунок 11 – Код к заданию 2 (часть 1)

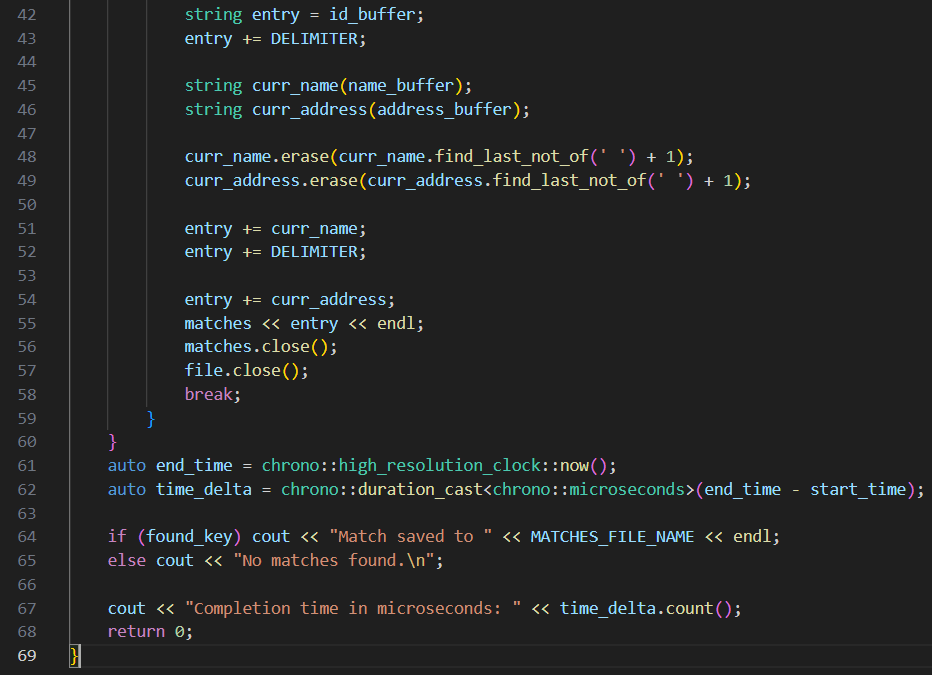


Рисунок 12 – Код к заданию 2 (часть 2)

### Результаты тестирования

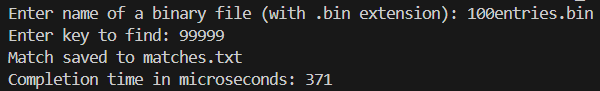
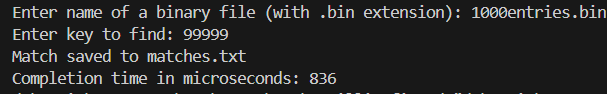


Рисунок 13 – Тестирование программы на файле из 100 записей

  
Рисунок 14 – Тестирование программы на файле из 1000 записей

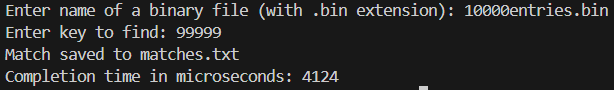
  
Рисунок 15 – Тестирование программы на файле из 10000 записей



Рисунок 16 – Файл с найденной по ключу записью

Тестирование показало, что программа работает корректно (рис.13-16). Запишем полученные в результате тестирования данные о времени выполнения поиска в таблицу 1.

Таблица 1 — замер времени выполнения линейного поиска

|  |  |
| --- | --- |
| Количество записей в файле | Время поиска в микросекундах |
| 100 | 371 |
| 1000 | 836 |
| 10000 | 4124 |

# ЗАДАНИЕ 3 Поиск записи в файле с применением дополнительной структуры данных, сформированной в оперативной памяти.

### Формулировка задачи

1. Для оптимизации поиска в файле создать в оперативной памяти

структур данных – таблицу, содержащую ключ и ссылку (смещение) на

запись в файле.

2. Разработать функцию, которая принимает на вход ключ и ищет в

таблице элемент, содержащий ключ поиска, а возвращает ссылку на

запись в файле. Алгоритм поиска определен в варианте.

3. Разработать функцию, которая принимает ссылку на запись в файле,

считывает ее, применяя механизм прямого доступа к записям файла.

Возвращает прочитанную запись как результат.

4. Провести практическую оценку времени выполнения поиска на файле

объемом 100, 1000, 10 000 записей.

5. Составить таблицу с указанием результатов замера времени.

### Алгоритм

**Однородный бинарный поиск** — это алгоритм, который, подобно классическому бинарному поиску, делит массив на части, но делает это равномерными шагами с использованием степеней двойки. Начальная позиция устанавливается в определённой точке (например, в начале), а затем вычисляется шаг смещения, который уменьшается с каждым циклом, пока не станет достаточно малым для точного нахождения элемента. В зависимости от того, больше или меньше текущий элемент целевого значения, алгоритм перемещает позицию вперёд или назад на рассчитанное смещение.

Ниже приведён алгоритм бинарного однородного поиска в таблице на псевдокоде.

i ← 0 // начальная позиция

pow ← 1 // инициализация степени двойки

n ← размер(vec)

Do

pow ← pow \* 2 // увеличиваем степень двойки

offset ← (n + (pow / 2)) / pow

If (vec[i][0] = x) Then

<Элемент найден в позиции i>

Exit

Else If (vec[i][0] < x) Then

i ← i + offset

Else

i ← i - offset

While (offset > 0)

<Элемент не найден>

### Код программы

Реализуем алгоритм на языке программирования C++ (рис. 17-20)

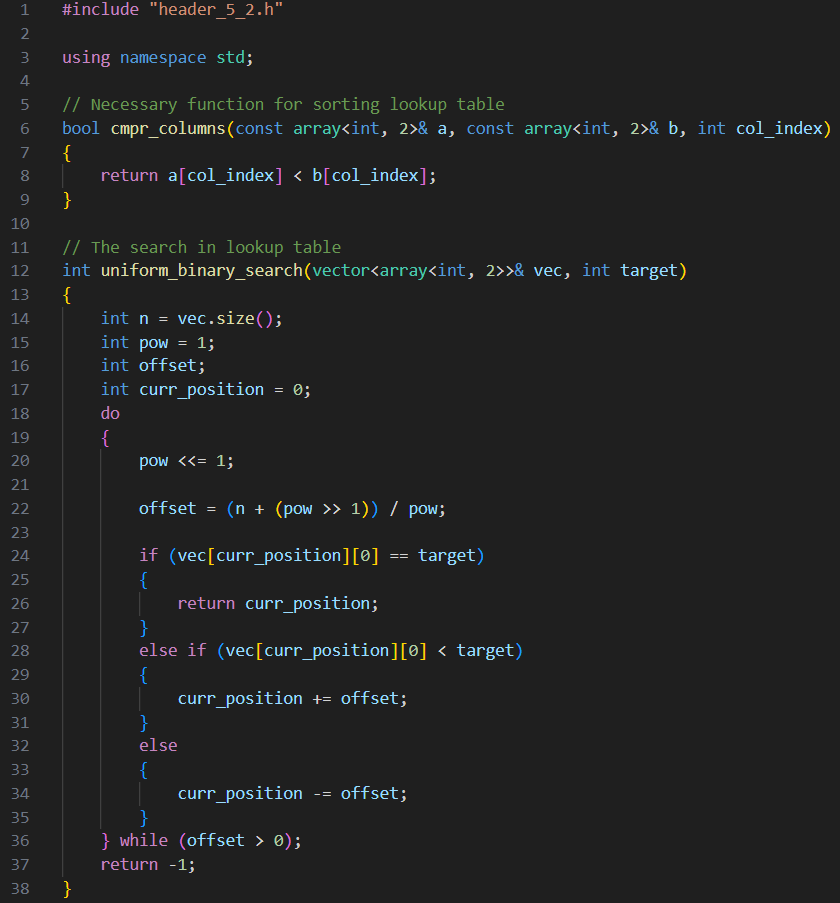


Рисунок 17 – Код к заданию 3 (часть 1)

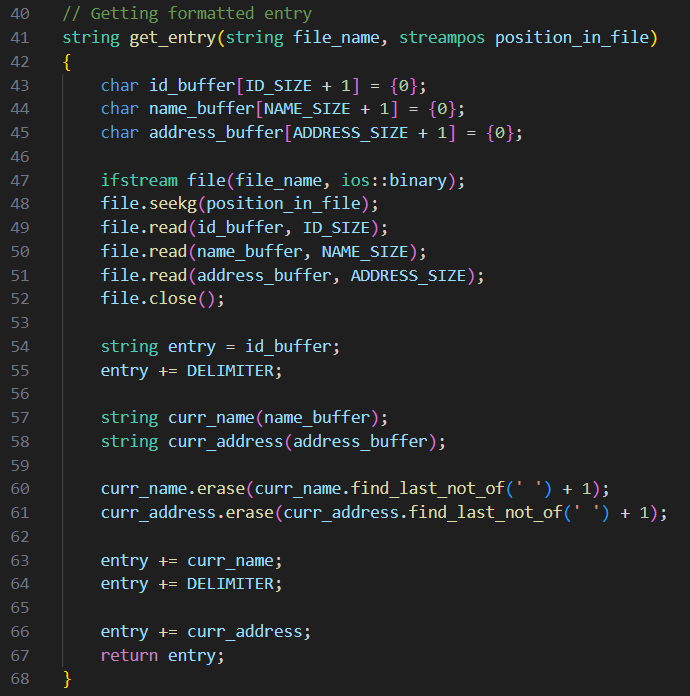


Рисунок 18 – Код к заданию 3 (часть 2)

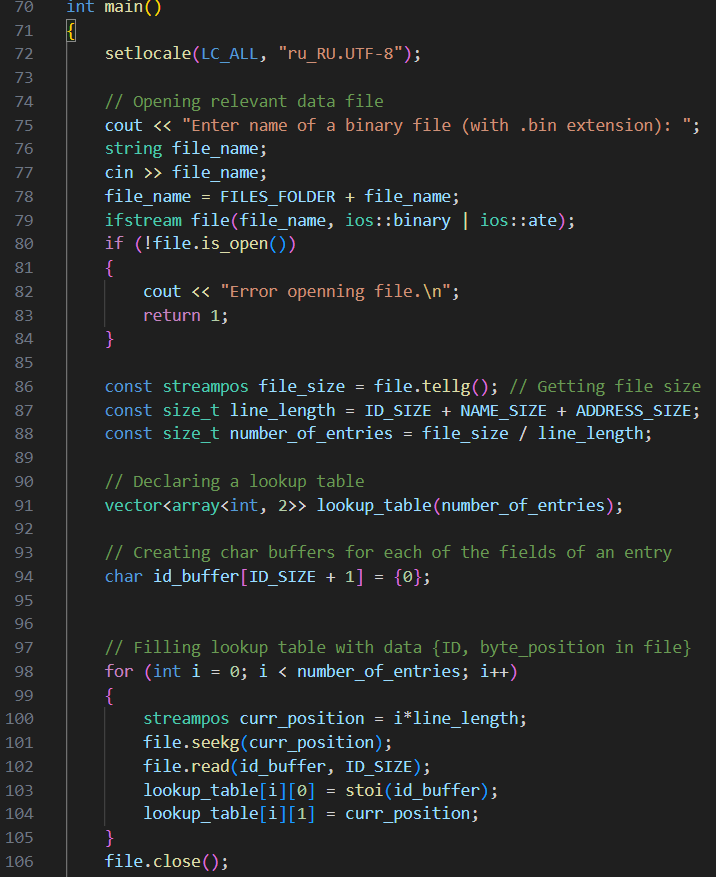


Рисунок 19 – Код к заданию 3 (часть 3)

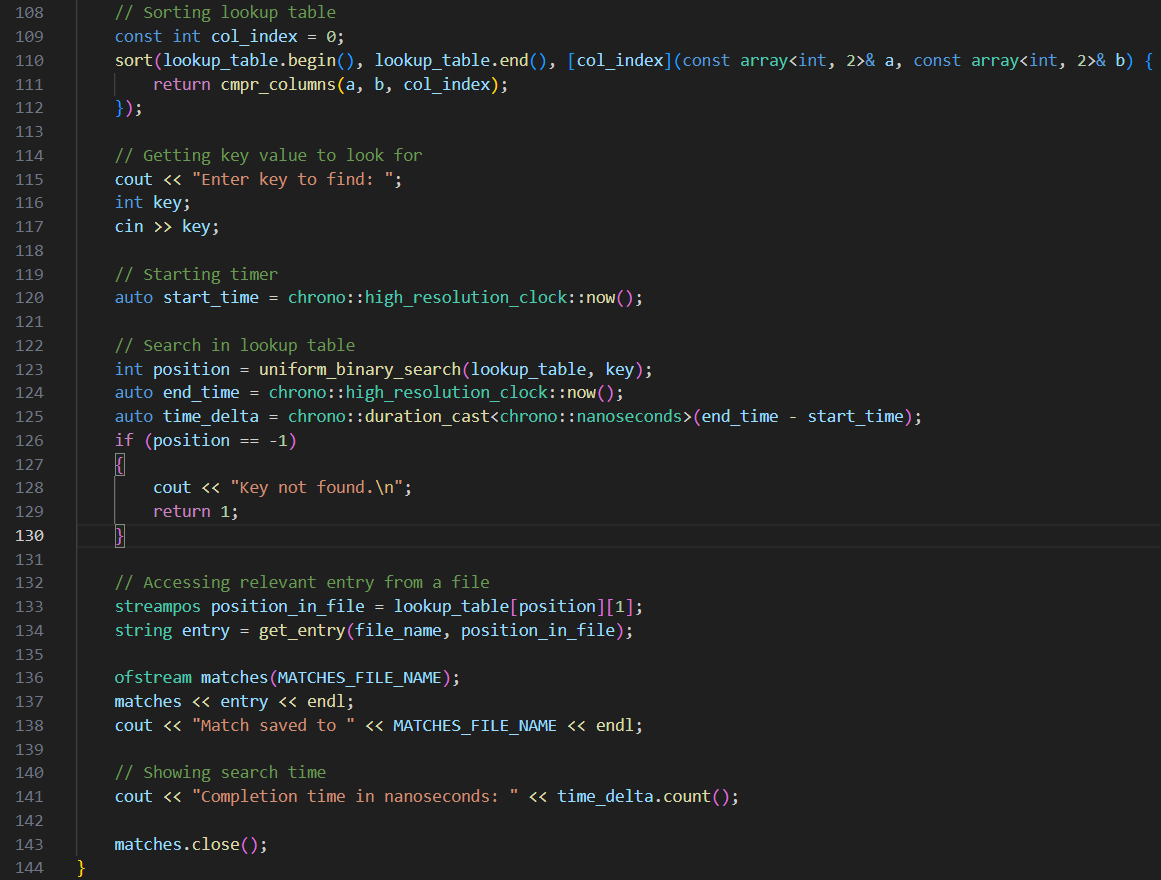


Рисунок 20 – Код к задаче 3 (часть 4)

### Результаты тестирования

Проведём тестирование программы бинарного однородного поиска на файлах из 100, 1000 и 10000 записей (рис. 21-29)

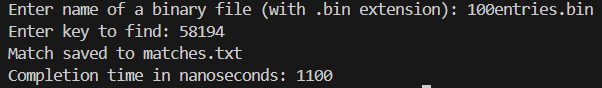
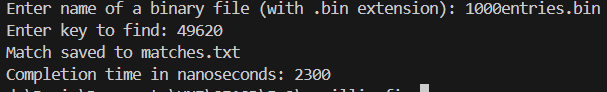
  
Рисунок 21 – Тестирование программы на файле из 100 записей



Рисунок 22 – Файл с найденной по ключу записью

  
Рисунок 23 – Тестирование программы на файле из 100 записей

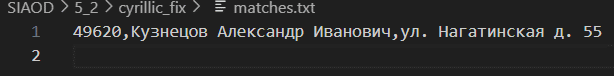


Рисунок 24 – Файл с найденной по ключу записью

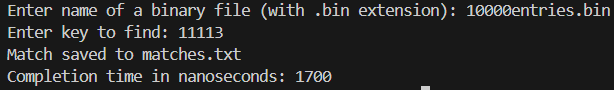
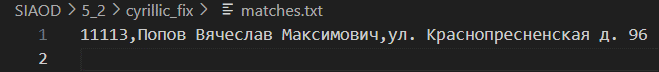
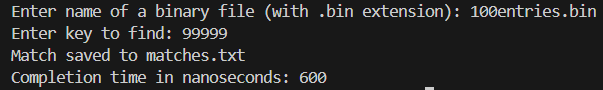
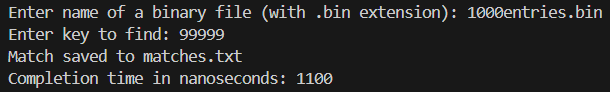
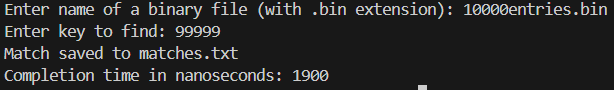
  
Рисунок 25 – Тестирование программы на файле из 100 записей

Рисунок 26 – Файл с найденной по ключу записью

  
Рисунок 27 – Тестирование программы на файле из 100 записей c поиском последней записи

  
Рисунок 28 – Тестирование программы на файле из 1000 записей c поиском последней записи

  
Рисунок 29 – Тестирование программы на файле из 10000 записей c поиском последней записи

Тестирование показало, что программа работает корректно. Запишем полученные в результате тестирования данные о времени выполнения поиска в таблицу 2.

Таблица 2 — замер времени выполнения бинарного однородного поиска

|  |  |
| --- | --- |
| Количество записей в файле | Время поиска в наносекундах |
| 100 | 600 |
| 1000 | 1100 |
| 10000 | 1900 |

Как видно из сравнения данных второго столбца таблиц 1 и 2, алгоритм бинарного однородного поиска является значительно более эффективным по сравнению с алгоритмом линейного поиска, особенно при большом количестве данных.

# ВЫВОД

В результате выполнения работы был получен практический опыт по применению алгоритмов поиска в таблицах данных. Было сделано сравнение алгоритмов линейного и бинарного однородного поиска, второй оказался более чем в десятки раз эффективнее даже на относительно небольшом наборе данных.

# СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Рысин, М. Л. Введение в структуры и алгоритмы обработки данных : учебное пособие / М. Л. Рысин, М. В. Сартаков, М. Б. Туманова. — Москва : РТУ МИРЭА, 2022 — Часть 2 : Поиск в тексте. Нелинейные структуры данных. Кодирование информации. Алгоритмические стратегии — 2022. — 111 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/310826 (дата обращения: 16.09.2024).
2. Документация по языку С++ [Электронный ресурс]. URL: https://docs.microsoft.com/ruru/cpp/cpp/ (дата обращения 16.09.2024).