|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | | | |  |
|  | Институт информационных технологий (ИТ) | |
|  | Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ) | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ОТЧЁТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ**  **«Работа с данными из файла»** | | | |  |
| **по дисциплине «Структуры и алгоритмы обработки данных (часть 2/2)»** | | | |  |
|  | | | |  |
| Выполнил студент группы ИКБО-41-23 | | Попов А.В. | |  |
|  | |  | |  |
| Принял  *Ассистент* | | Рысин М.Л. | |  |
| Практические работы выполнены | «\_\_\_»\_\_\_\_\_\_\_2024 г. | | (подпись студента) | |
| «Зачтено» | «\_\_\_»\_\_\_\_\_\_\_2024 г. | | (подпись преподавателя) | |
|  |  | |  | |

СОДЕРЖАНИЕ

[1 ПОСТАНОВКА ЗАДАЧИ 3](#__RefHeading___Toc815_1995867043)

[2 ЗАДАНИЕ 1 4](#__RefHeading___Toc817_1995867043)

[2.1 Формулировка задачи 4](#__RefHeading___Toc819_1995867043)

[2.2 Реализация задачи 4](#__RefHeading___Toc821_1995867043)

[3 ЗАДАНИЕ 2 7](#__RefHeading___Toc823_1995867043)

[3.1 Формулировка задачи 7](#__RefHeading___Toc825_1995867043)

[3.2 Реализация задачи 7](#__RefHeading___Toc827_1995867043)

[3.3 Тестирование программы 8](#__RefHeading___Toc829_1995867043)

[4 ЗАДАНИЕ 3 10](#__RefHeading___Toc831_1995867043)

[3.1 Формулировка задачи 10](#__RefHeading___Toc833_1995867043)

[3.2 Реализация задачи 11](#__RefHeading___Toc835_1995867043)

[3.1 Тестирование программы 14](#__RefHeading___Toc837_1995867043)

[5 ВЫВОД 15](#__RefHeading___Toc839_1995867043)

# 1 ПОСТАНОВКА ЗАДАЧИ

Разработать программу поиска записей с заданным ключом в двоичном файле с применением различных алгоритмов.

# 2 ЗАДАНИЕ 1

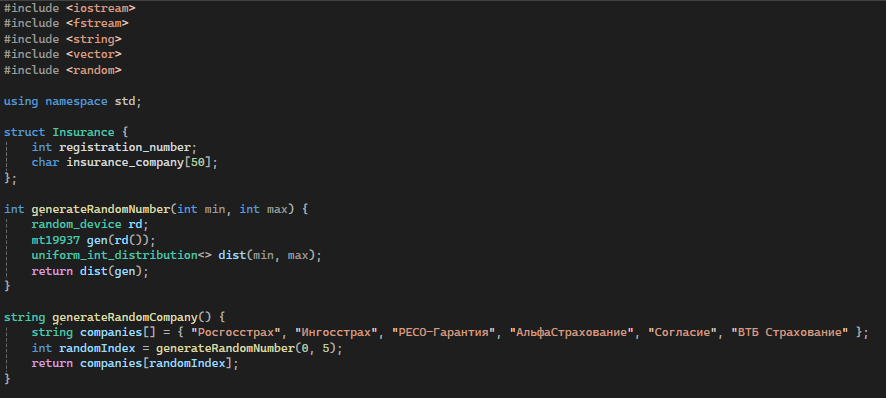
## 2.1 Формулировка задачи

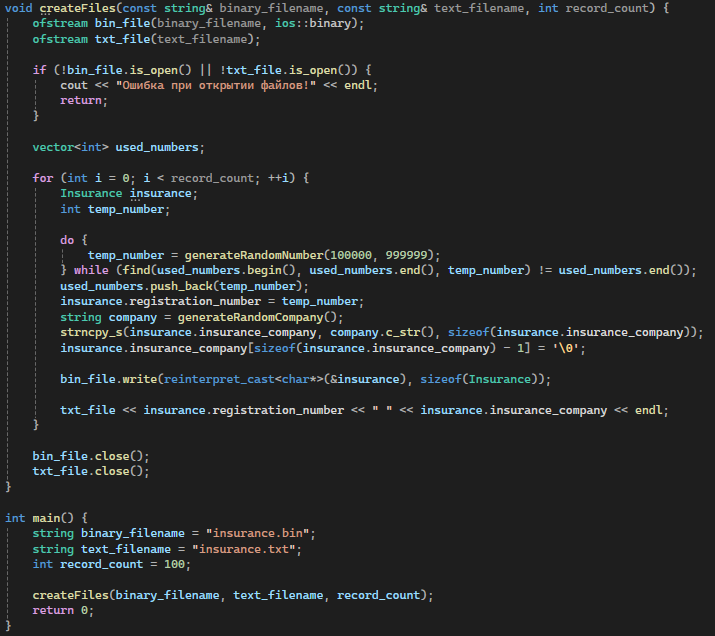
Создать двоичный файл из записей (структура записи определена вариантом). Поле ключа записи в задании варианта подчеркнуто. Заполнить файл данными, используя для поля ключа датчик случайных чисел. Ключи записей в файле уникальны.

Персональный вариант №10:

Структура записи файла: *регистрационный номер – шестизначное число*, название страховой компании.

## 2.2 Реализация задачи

Рисунок 1 — Структура записи в файл и функции для генерации случайных регистрационных номеров и названий компаний

Рисунок 2 — функция создания текстого и бинарного файлов и основная функция программы

Программа предназначена для генерации данных о страховых полисах и их записи в два файла: бинарный (insurance.bin) и текстовый (insurance.txt). Она генерирует записи, каждая из которых содержит уникальный шестизначный регистрационный номер и название страховой компании.

Названия страховых компаний выбираются случайным образом из заранее заданного списка. Для генерации случайных чисел и выбора компаний используются инструменты стандартной библиотеки C++, что гарантирует равномерное распределение псевдослучайных чисел.

В бинарном файле запись хранится в компактной форме с использованием побайтной записи структуры Insurance. Это позволяет в будущем быстро считывать данные программно. Размер одной записи вычисляется как сумма размеров всех полей структуры:

* int registration\_number: 4 байта.
* char insurance\_company[50]: 50 байт.

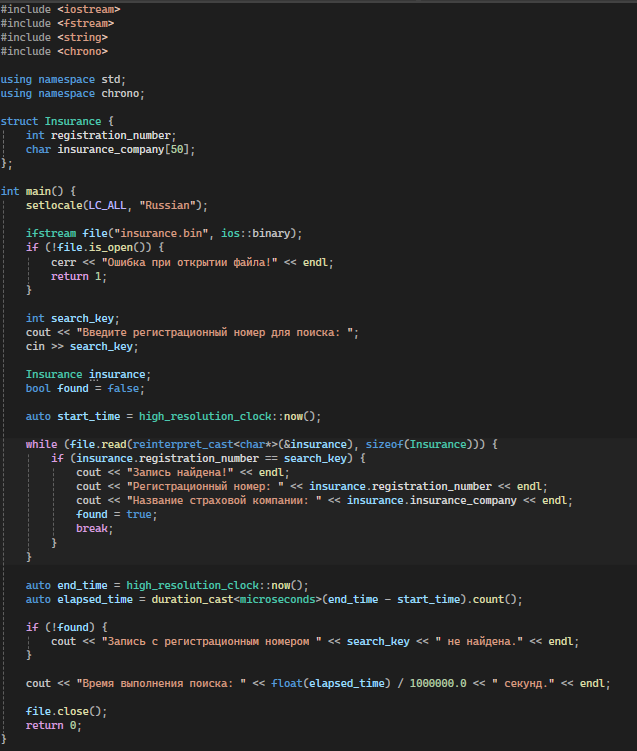
Общий размер записи составляет 54 байта. Это значение используется для организации прямого доступа к записям в бинарном файле.

# 3 ЗАДАНИЕ 2

## 3.1 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 и составить таблицу с указанием результатов замера времени.

## 3.2 Реализация задачи

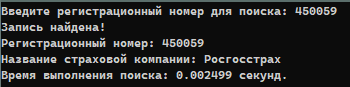
Рисунок 3 — реализация задания 2 на C++

После открытия файла в режиме бинарного чтения (ios::binary) программа проверяет, удалось ли открыть файл. Если файл недоступен, выводится сообщение об ошибке, и выполнение завершается. Пользователь вводит искомый регистрационный номер, который будет сравниваться с номерами в файле.

Записи читаются из файла последовательно, блоками размера структуры Insurance. Каждая запись сравнивается с введённым регистрационным номером. Если запись с указанным номером найдена, выводятся её данные.

Если запись найдена, поиск завершается досрочно, и программа переходит к выводу результатов. Для повышения информативности программа измеряет время выполнения поиска.

## 3.3 Тестирование программы

Рисунок 4 — Результат выполнения программы

Проведём практическую оценку времени выполнения поиска на файле объемом 100, 1000, 10 000 записей. Во всех случаях ключевое значение будет в конце файла, то есть будет воссоздан худший случай для данного алгоритма. Приведём результаты тестирования в таблице 1.

|  |  |
| --- | --- |
| **Кол-во записей** | Время выполнения (секунды) |
| 100 | 0.002873 |
| 1 000 | 0.003499 |
| 10 000 | 0.005686 |

Таблица 1 - Практическая оценка времени выполнения линейного поиска

В результате проделанных тестов можно сделать вывод о том, что сложность алгоритма линейного поиска в худшем случае, то есть когда ключевое значение находится на последнем месте, равна O(n). В лучшем случае, то есть когда ключевое значение находится на первом месте, сложность алгоритма будет равна O(1).

# 4 ЗАДАНИЕ 3

## 3.1 Формулировка задачи

Для оптимизации поиска в файле создать в оперативной памяти структур данных – таблицу, содержащую ключ и ссылку (смещение) на запись в файле. 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. Разработать функцию, которая принимает ссылку на запись в файле, считывает ее, применяя механизм прямого доступа к записям файла. Возвращает прочитанную запись как результат. Провести практическую оценку времени выполнения поиска на файле объемом 100, 1000, 10 000 записей.

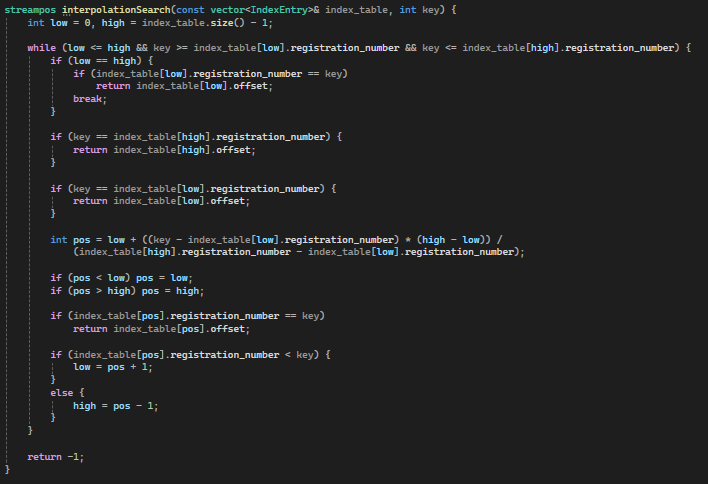
Персональный вариант №10:

Алгоритм поиска: Интерполяционный поиск.

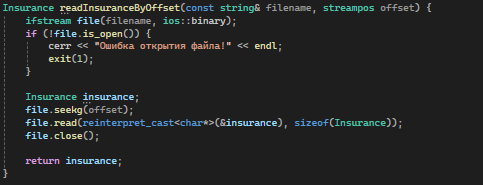
## 3.2 Реализация задачи

Рисунок 5 - Структуры данных, функции сравнения и создания в оперативной памяти таблицы

Функция createIndexTable предназначена для создания индексационной таблицы, которая позволяет ускорить поиск записей в бинарном файле. Основная идея заключается в том, чтобы собрать информацию о каждой записи в файле, включая её регистрационный номер и смещение (позицию) в файле, и сохранить эти данные в виде вектора структур IndexEntry. Затем таблица сортируется по регистрационным номерам, чтобы обеспечить возможность реализовать интерполяционный алгоритм поиска.

Рисунок 6 — Функция интерполяционного поиска

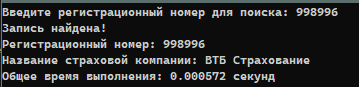
Функция interpolationSearch реализует интерполяционный поиск в отсортированном массиве индексов. Этот алгоритм оптимизирует процесс поиска за счёт использования математической интерполяции, которая определяет позицию искомого элемента, основываясь на значении ключа. В отличие от бинарного поиска, интерполяционный более эффективен для данных с равномерным распределением.

Рисунок 7 — Функция прямого доступа к данным

Функция readInsuranceByOffset предназначена для чтения конкретной записи из бинарного файла на основе её смещения (позиции в файле). Она используется для прямого доступа к данным, что особенно полезно в сочетании с индексной таблицей, где хранится информация о смещениях всех записей.

Рисунок 8 — основная функция программы

## 3.1 Тестирование программы

Рисунок 9 - Результат выполнения программы

Проведём практическую оценку времени выполнения поиска на файле объемом 100, 1000, 10 000 записей. Во всех случаях ключевое значение будет в с наибольшим ключевым значением, то есть будет воссоздан худший случай для данного алгоритма.

Приведём результаты тестирования в таблице 2.

|  |  |
| --- | --- |
| **Кол-во записей** | Время выполнения (секунды) |
| 100 | 0.000572 |
| 1 000 | 0.002138 |
| 10 000 | 0.018679 |

Таблица 2 - Практическая оценка времени выполнения интерполяционного поиска

В результате проделанных тестов можно сделать вывод о том, что сложность алгоритма линейного поиска в худшем случае, то есть когда ключевое значение находится на последнем месте, равна O(n). В лучшем случае, то есть когда ключевое значение находится на первом месте, сложность алгоритма будет равна O(1). В среднем случае его сложность будет равна O(log n \* log n), что лучше, чем O(log n) для бинарного поиска.

# 5 ВЫВОД

В ходе выполнения данной работы были изучены алгоритмы поиска в таблице при работе с данными из файла. Было разработано и реализовано два эффективный алгоритма внешней сортировки: линейный и интерполяционный. Линейный алгоритм был лёгок в реализации, его можно использовать и с отсортированными данными, и с не отсортированными. Однако, имеет в среднем случае сложность O(n), что, в свою очередь, замедляет его выполнение для больших наборов данных. Интерполяционный алгоритм поиска является более сложным в реализации, его можно использовать только с отсортированными данными. Но он имеет в среднем случае сложность O(log n \* log n), что делает его очень достаточно для поиска в отсортированных данных и на большом их наборе.