МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Факультет автоматики и вычислительной техники**



**КУРСОВАЯ РАБОТА**

**по дисциплине: «Программирование»**

**на тему:«Динамические структуры данных в памяти. Вариант 1.6.3»**

Выполнил(а):Проверил:

Студент(ка) гр. «», «» *«»*

*« » « »*

«».«\_\_\_» \_\_\_\_\_\_ 20\_\_ г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (подпись)

Новосибирск

2020

Содержание

[1 Развернутое описание задания 2](#__RefHeading___Toc118_10860175)

[2 Структурное описание разработки 2](#__RefHeading___Toc120_10860175)

[2.1 Контейнер 2](#__RefHeading___Toc2497_806517012)

[2.2 Тип данных для движения поезда 4](#__RefHeading___Toc2499_806517012)

[2.3 Формат хранения данных 5](#__RefHeading___Toc124_10860175)

[2.4 Основные алгоритмы 5](#__RefHeading___Toc126_10860175)

[2.4.1 Удаление одного элемента 6](#__RefHeading___Toc1977_1737634799)

[2.4.2 Удаление нескольких элементов 6](#__RefHeading___Toc1979_1737634799)

[2.4.3 Сортировка (внешняя естественным слиянием) 7](#__RefHeading___Toc1981_1737634799)

[3 Функциональное описание 9](#__RefHeading___Toc146_10860175)

[3.1 Реализация программы для простого типа данных 11](#__RefHeading___Toc415_3319494419)

[3.2 Реализация базы данных рейсов поездов 11](#__RefHeading___Toc417_3319494419)

[4 Описание работы программы на контрольных примерах 13](#__RefHeading___Toc6288_149900317)

[4.1 Тестирование программы с простым типом 13](#__RefHeading___Toc6290_149900317)

[4.2 Тестирование базы данных междугородных переговоров 16](#__RefHeading___Toc6292_149900317)

[4.3 Исследование вычислительной сложности основных операций 18](#__RefHeading___Toc1307_3767299533)

[5 Выводы 20](#__RefHeading___Toc6311_149900317)

[Список использованных источников 21](#__RefHeading___Toc419_3319494419)

[ПРИЛОЖЕНИЕ - Исходный код 22](#__RefHeading___Toc1972_1737634799)

# 1 Развернутое описание задания

1. Двоичный файл содержит записи переменной длины. Формат записи предполагает ее переменную размерность. Реализовать набор операций над записями без загрузки одновременно всей структуры данных в память (поэлементная загрузка СД): добавление строки, извлечение, удаление, вставка по логическому номеру и редактирование (обновление) строки, вставка с сохранением порядка, сортировка. При изменении размерности записи она переписывается в конец файла.;
2. Протестировать структуру данных на простом типе данных (например, int, double) и сложном, из выбранного по заданию;
3. Движение поездов. Номер поезда, дни недели отправления, время отправления, время в пути (часов, минут). Станция отравления, назначения, промежуточные станции. Промежуточные станции запоминаются в динамическом массиве номеров станций. Поиск всех поездов, следующих до заданной станции.

# 2 Структурное описание разработки

## 2.1 Контейнер

Класс контейнера хранит имя файла и текущее число элементов в нем. Согласно заданию, контейнер должен корректно работать с объектами разных типов, в связи с этим целесообразно реализовать шаблонный класс.

На рисунке показан класс базы данных, при этом в класс контейнера добавлены методы, необходимые для выполнения задания. Для реализации "поиска и действие по различным критериям" в классе добавлена функция executeIf, принимающая два функциональных объекта.

База данных содержит методы:

1. Добавления элемента
2. Вставки элемента по логическому номеру
3. Удаление по логическому номеру
4. Обновление по логическому номеру
5. Удаление элемента
6. Сортировка
7. Вставка с сохранением порядка
8. Поиска и действие по различным критериям
9. Оператор []
10. Перегрузка оператора вывода



Рисунок 1 - класс Контейнера

Для поиска элемента по индексу в базе данных перегружен оператор [].

## 2.2 Тип данных для движения поезда

Класс данных о движении поездов хранит:

1. Номер поезда
2. Массив меток едет ли поезд в этот день недели или нет
3. Время отправления
4. Время в пути
5. Станция отправления
6. Станция назначения
7. Массив меток посещения промежуточных станций

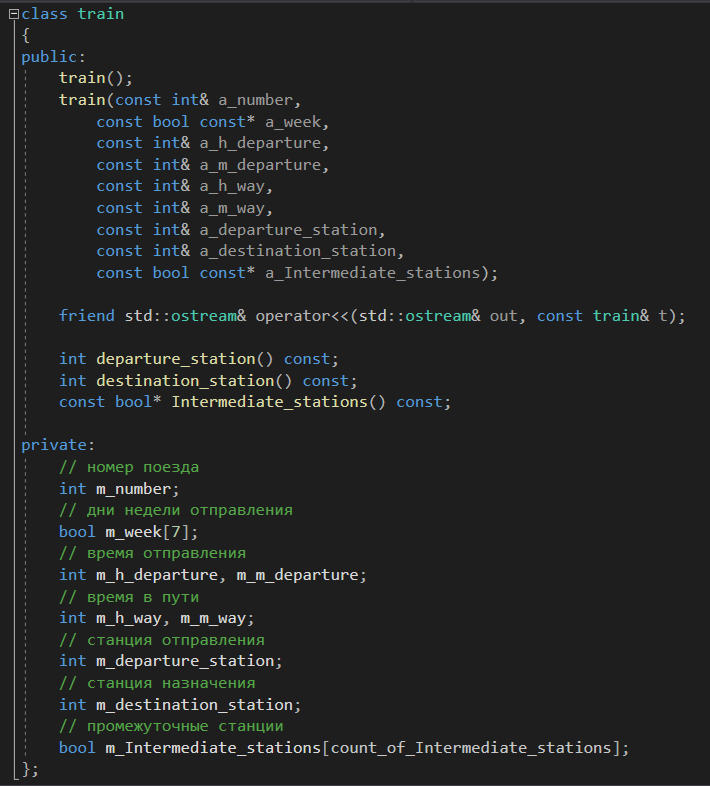


Рисунок 2 – Класс для хранения данных о движении поезда

Для выполнения задания добавлено 2 геттера на станции назначения, отправления и промежуточные.

## 2.3 Формат хранения данных

При сохранении данных в файл с помощью базы данных помещается:

* Бинарные данные элементов базы данных.

Для сохранения информации используется функция ostream::write, поэтому в файл сохраняется структура train, показанная на рисунке 2. В общем случае, размер (количество байт), необходимое для объекта структуры зависит от архитектуры компьютера и настроек выравнивания полей, используемых в проекте. Управлять выравниванием полей в С++ можно с помощью директивы #pragma pack. Выравнивание полей необходимо учитывать при чтении файла с помощью программы, написанной на другом языке программирования.

## 2.4 Основные алгоритмы

Наиболее интересные алгоритмы реализованы в классе массива: удаление одного элемента, оператор [] и сортировка. Рассмотрим каждый из них.

### 2.4.1 Удаление одного элемента

При удалении одного элемента массива выполняется копирование элементов из файла во временный файл, элемент с переданным индексом пропускается при записи. После чего, размер базы данных уменьшается на единицу. Основной файл удаляется, а временный получает имя основного. Код удаления показа на рисунке 3.



Рисунок 3 - Код алгоритма удаления элемента база данных по индексу

Вычислительная сложность такого алгоритма - O(n).

### 2.4.2 Оператор []

При получении данных по индексу сдвигаемся на величину байт равную индекс \* размер данных. Считываем данные с этого места с помощью fstream.read(). Возвращаем считанные данные. Сложность О(1).

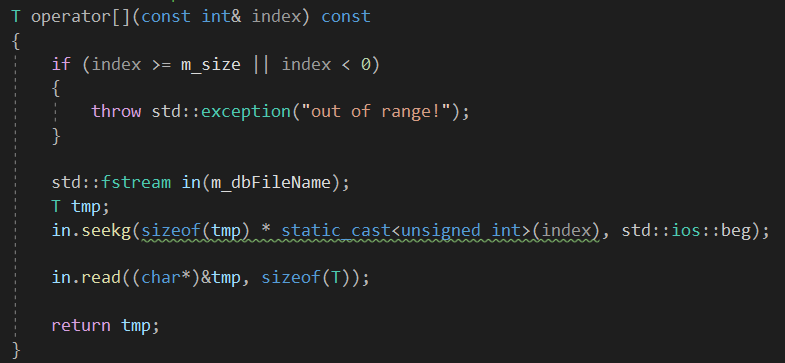


Рисунок 4 – код оператора []

### 2.4.3 Сортировка (внешняя естественным слиянием)

Внешняя сортировка – это сортировка данных, которые расположены на внешних устройствах и не вмещающихся в оперативную память.

Данные, хранящиеся на внешних устройствах, имеют большой объем, что не позволяет их целиком переместить в оперативную память, отсортировать с использованием одного из алгоритмов внутренней сортировки, а затем вернуть их на внешнее устройство. В этом случае осуществлялось бы минимальное количество проходов через файл, то есть было бы однократное чтение и однократная запись данных. Однако на практике приходится осуществлять чтение, обработку и запись данных в файл по блокам, размер которых зависит от операционной системы и имеющегося объема оперативной памяти, что приводит к увеличению числа проходов через файл и заметному снижению скорости сортировки.

К наиболее известным алгоритмам внешних сортировок относятся:

1. сортировки слиянием (простое слияние и естественное слияние);
2. улучшенные сортировки (многофазная сортировка и каскадная сортировка).

Из представленных внешних сортировок наиболее важным является метод сортировки с помощью слияния. Прежде чем описывать алгоритм сортировки слиянием введем несколько определений.

Основным понятием при использовании внешней сортировки является понятие серии. Серия (упорядоченный отрезок) – это последовательность элементов, которая упорядочена по ключу.

Количество элементов в серии называется длиной серии. Серия, состоящая из одного элемента, упорядочена всегда. Последняя серия может иметь длину меньшую, чем остальные серии файлов. Максимальное количество серий в файле N (все элементы не упорядочены). Минимальное количество серий одна (все элементы упорядочены).

В основе большинства методов внешних сортировок лежит процедура слияния и процедура распределения. Слияние – это процесс объединения двух (или более) упорядоченных серий в одну упорядоченную последовательность при помощи циклического выбора элементов, доступных в данный момент. Распределение – это процесс разделения упорядоченных серий на два и несколько вспомогательных файла.

Фаза – это действия по однократной обработке всей последовательности элементов. Двухфазная сортировка – это сортировка, в которой отдельно реализуется две фазы: распределение и слияние. Однофазная сортировка – это сортировка, в которой объединены фазы распределения и слияния в одну.

Двухпутевым слиянием называется сортировка, в которой данные распределяются на два вспомогательных файла. Многопутевым слиянием называется сортировка, в которой данные распределяются на N (N > 2) вспомогательных файлов.

Общий алгоритм сортировки слиянием

Сначала серии распределяются на два или более вспомогательных файлов. Данное распределение идет поочередно: первая серия записывается в первый вспомогательный файл, вторая – во второй и так далее до последнего вспомогательного файла. Затем опять запись серии начинается в первый вспомогательный файл. После распределения всех серий, они объединяются в более длинные упорядоченные отрезки, то есть из каждого вспомогательного файла берется по одной серии, которые сливаются. Если в каком-то файле серия заканчивается, то переход к следующей серии не осуществляется. В зависимости от вида сортировки сформированная более длинная упорядоченная серия записывается либо в исходный файл, либо в один из вспомогательных файлов. После того как все серии из всех вспомогательных файлов объединены в новые серии, потом опять начинается их распределение. И так до тех пор, пока все данные не будут отсортированы.

Выделим основные характеристики сортировки слиянием:

количество фаз в реализации сортировки;

количество вспомогательных файлов, на которые распределяются серии.

Рассмотрим основные и наиболее важные алгоритмы внешних сортировок более подробно.

Сортировка простым слиянием

Одна из сортировок на основе слияния называется простым слиянием.

Алгоритм сортировки простым слияния является простейшим алгоритмом внешней сортировки, основанный на процедуре слияния серией.

В данном алгоритме длина серий фиксируется на каждом шаге. В исходном файле все серии имеют длину 1, после первого шага она равна 2, после второго – 4, после третьего – 8, после k -го шага – 2k.

Алгоритм сортировки простым слиянием

Шаг 1. Исходный файл f разбивается на два вспомогательных файла f1 и f2.

Шаг 2. Вспомогательные файлы f1 и f2 сливаются в файл f, при этом одиночные элементы образуют упорядоченные пары.

Шаг 3. Полученный файл f вновь обрабатывается, как указано в шагах 1 и 2. При этом упорядоченные пары переходят в упорядоченные четверки.

Шаг 4. Повторяя шаги, сливаем четверки в восьмерки и т.д., каждый раз удваивая длину слитых последовательностей до тех пор, пока не будет упорядочен целиком весь файл.

После выполнения i проходов получаем два файла, состоящих из серий длины 2i. Окончание процесса происходит при выполнении условия 2i>=n. Следовательно, процесс сортировки простым слиянием требует порядка O(log n) проходов по данным.

Признаками конца сортировки простым слиянием являются следующие условия:

длина серии не меньше количества элементов в файле (определяется после фазы слияния);

количество серий равно 1 (определяется на фазе слияния).

при однофазной сортировке второй по счету вспомогательный файл после распределения серий остался пустым.

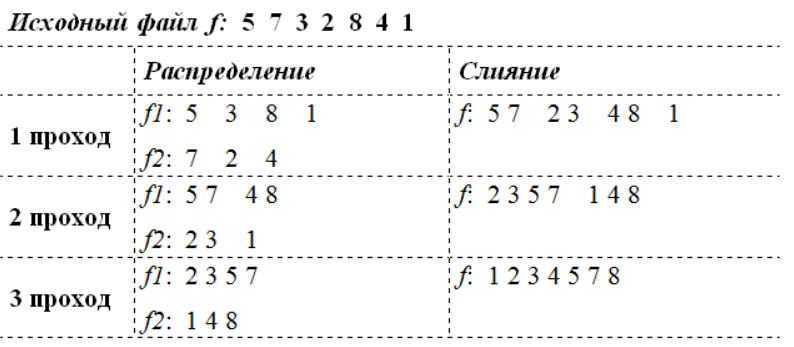


Рисунок 5 – Пример внешней сортировки слиянием

# 3 Функциональное описание

Для проверки функциональности программы реализован тестовый файл main.cpp, позволяющее протестировать все методы базы данных удаление элементов, выполнение сортировки и так далее. Для этого функция main выводит на экран все данные после проведения операций (рисунок 6).

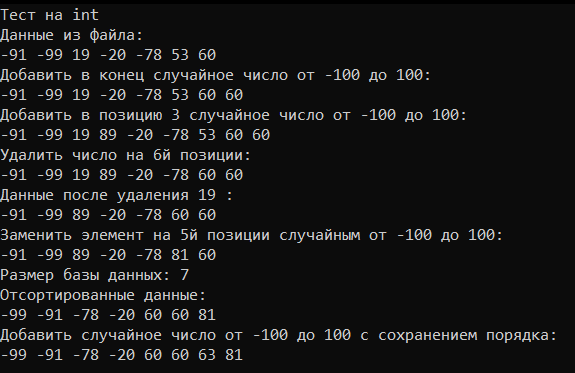


Рисунок 6 - Данные после проведения операций

## 3.1 Реализация программы для простого типа данных

Программа проверки корректности работы функций контейнера на простом типе данных состоит из двух файлов - binaryDB.h и main.cpp.

## 3.2 Реализация базы данных рейсов поездов

Для реализации базы рейсов поездов, по аналогии с предыдущей программой, в функцию main был добавлен вывод всей базы данных, ввод номера станции, создание лямбд для выбора нужных записей и нужного действия над ними(вывода на экран)

# 4 Описание работы программы на контрольных примерах

Обе программы - для работы с простым типом, и с рейсами поездов, имеют тестовый файл main.cpp. В консоли выводится результаты всех операции. Далее, программа выводит полученные результаты, или запрашивает необходимые для выполнения операции данных — например, для удаления элемента это его индекс. Тестирование программ проводилось вручную, некоторые варианты входных данных и результата описаны в разделах 4.1 и 4.2.

## 4.1 Тестирование программы с простым типом

Для тестирование программы было выполнено следующее:

1. запущена программа, файл пустой;
2. вставка элемента - попытка добавления элемента 23 в позицию с номером 1 привела к ошибке;
3. вставка элемента - вставка элемента 45 в позицию 0 завершилась успешно. Содержимое массива: 45.
4. вставка значения 7 в конец массива завершилась успешно, содержимое массива: 45, 7.
5. вставка значения 3 в конец массива завершилась успешно, содержимое массива: 45, 7, 3.
6. программа завершилась успешно;
7. запущена программа, массив пустой;
8. вставка значения 123 в конец массива завершилась успешно, содержимое массива: 123.
9. загрузка данных с файла завершилась успешно, содержимое массива 45, 7, 3 (старое содержимое сохранено в другом файле).
10. вставка значения 3.01 в конец массива завершилась успешно, содержимое массива: 45, 7, 3, 3.01.
11. вставка значения 3.0002 в конец массива завершилась успешно, содержимое массива: 45, 7, 3, 3.01, 3.0002.
12. вставка значения 4 в конец массива завершилась успешно, содержимое массива: 45, 7, 3, 3.01, 3.0002, 4
13. поиск значения по индексу 10 - элемент не найден;
14. поиск значения по индексу 4 - найден элемент 3.0002;
15. удаление элементов по значению 3.0100001 - не было удалено ни одного элемента, содержимое массива: 45, 7, 3, 3.01, 3.0002, 4;
16. поиск элементов по значению 3.0100001, результат - найден элемент на позиции 3;
17. сортировка, содержимое массива: 3.0002, 3.01, 3, 4, 7, 45.
18. вставка числа 5 с сохранением порядка, массива: 3.0002, 3.01, 3, 4, 5, 7, 45.

Снимок окна программы после выполнения тестирования приведен на рисунке 7.

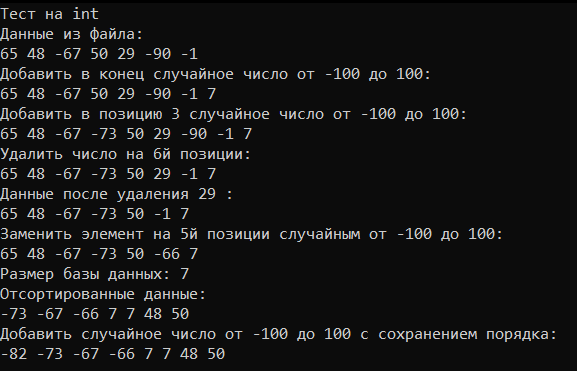


Рисунок 7 - Результаты тестирования программы с простым типом

В ходе выполнения тестирования, в программе был **найден и исправен ряд ошибок**.

Кроме того, **найден недочет** интерфейса (набора открытых функций) класса массива - функция удаления элемента по значению недостаточно гибкая и может не сработать для элементов дробного типа, которые надо сравнивать с погрешностью. Решить проблему можно с помощью добавления функции типа remove\_if, работающей по подобию find\_if.

## 4.2 Тестирование базы данных рейсов поездов

Тестирование БД рейсов проводилось подобно тестированию программы для встроенного типа данных. На рисунке 8 показана исходная база данных рейсов поездов, на рисунке 9 - вывод рейсов идущих в введённую станцию. Также, были проверены функции сохранения базы в файл, загрузки, сортировки, удаления.

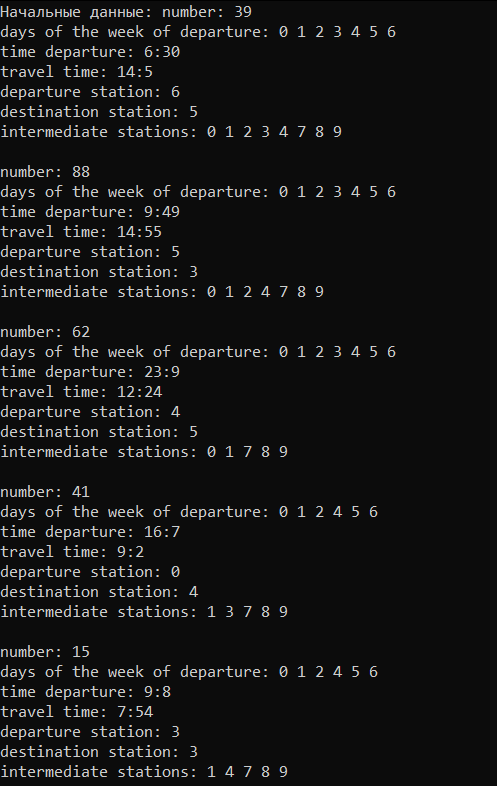


Рисунок 8 - Тестирование вывода базы данных

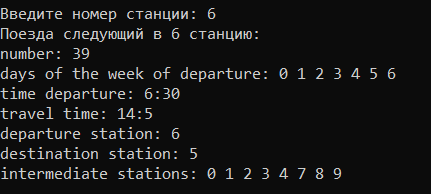


Рисунок 9 - Вывод неоплаченных переговоров

## 4.3 Исследование вычислительной сложности основных операций

Исследуем на практике вычислительную сложность функций сортировки и удаления элементов с заданным значением. Для этого напишем одну небольшую программу, в качестве типа данных выберем int, чтобы было проще генерировать данные для массива и не возникало проблем со сравнением элементов (точностью).

Чтобы замерять время работы программы будем использовать библиотеку chrono, так как она позволяет изменить время работы в миллисекундах (в отличии от time.h).

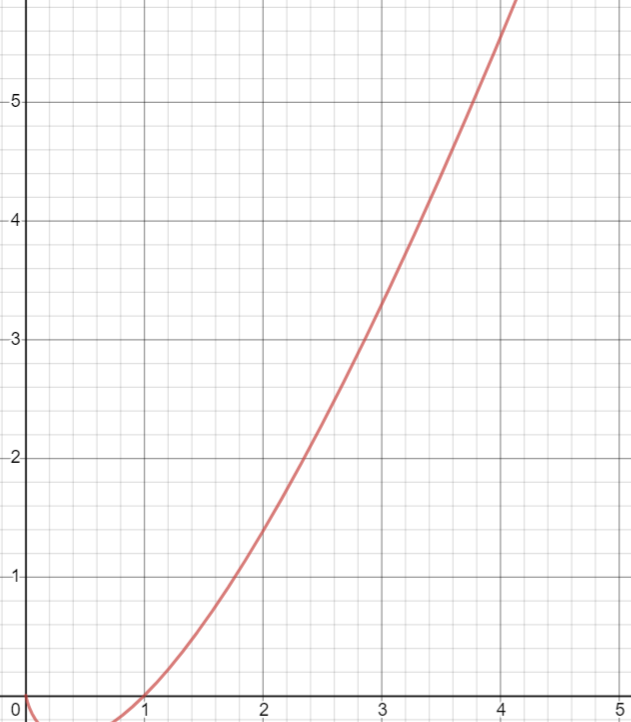


Рисунок 10 - Графики зависимости времени сортировки от размера массива

# 5 Выводы

В ходе выполнения работы были разработаны:

* класс бинарной базы данных;
* программа, выполняющая проверку работоспособности класса базы данных на простом типе данных;
* программа, реализующая базу данных рейсов поездов, использующая класс базы данных с пользовательским типом данных;
* программа, выполняющая замер времени, необходимого для выполнения ключевых операций с базой данных.

Предложены следующие улучшения разработанных программ

* добавить поддержку итераторов (как в std::vector);
* добавить проверку корректности полей (в класс рейсов).

# Список использованных источников

1. Буч Градди Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд. / Буч Градди, Максимчук Роберт А., Энгл Майкл У., Янг Бобби Дж., Коналлен Джим, Хьюстон Келли А.: Пер с англ. — М.: ООО «И.Д. Вильямс», 2010. — 720 с.
2. Расставим точки над структурами C/C++. URL: https://habr.com/ru/post/142662/ (дата обращения - 18.10.2020)
3. Скиена С. Алгоритмы. Руководство по разработке.-2-е изд.: пер. с англ.-СПб.:БХВ-Петербург, 2011.-720с.: ил.
4. Рекурсия в программировании. Анализ алгоритмов. URL: https://pro-prof.com/archives/813 (дата обращения - 18.10.2020)
5. Google C++ Style Guide. URL: https://google.github.io/styleguide/cppguide.html

# ПРИЛОЖЕНИЕ - Исходный код

Файл binaryDB.h (общий для всех проектов):

#pragma once

#include <string>

#include <fstream>

#include <exception>

#include <cstdio>

#include <functional>

template<class T>

class binaryDB

{

public:

binaryDB(const std::string& a\_dbFileName) : m\_dbFileName(a\_dbFileName)

{

std::ifstream in(m\_dbFileName, std::ifstream::ate | std::ifstream::binary);

if (!in.is\_open())

{

throw std::exception("File not exist");

}

m\_size = in.tellg() / sizeof(T);

}

// добавление строки

void add(const T& f)

{

std::ofstream out(m\_dbFileName, std::ios::app);

out.write((char\*)&f, sizeof(T));

++m\_size;

}

// вставка по логическому номеру

void addByIndex(const T& f, const int& index)

{

if (index >= m\_size || index < 0)

{

throw std::exception("out of range!");

}

int i = -1;

T tmp;

std::fstream in(m\_dbFileName);

const std::string tmpFileName = "tmpFile";

std::ofstream tmpFile(tmpFileName);

while (!in.eof())

{

in.read((char\*)&tmp, sizeof(T));

++i;

if (i == index)

{

tmpFile.write((char\*)&f, sizeof(f));

}

tmpFile.write((char\*)&tmp, sizeof(tmp));

}

m\_size++;

in.close();

tmpFile.close();

std::remove(m\_dbFileName.c\_str());

std::rename(tmpFileName.c\_str(), m\_dbFileName.c\_str());

}

void eraseByIndex(const int& index)

{

if (index >= m\_size || index < 0)

{

throw std::exception("out of range!");

}

int i = -1;

T tmp;

std::fstream in(m\_dbFileName);

const std::string tmpFileName = "tmpFile";

std::ofstream tmpFile(tmpFileName);

while (!in.eof())

{

in.read((char\*)&tmp, sizeof(T));

++i;

if (i == index)

{

continue;

}

tmpFile.write((char\*)&tmp, sizeof(tmp));

}

in.close();

tmpFile.close();

--m\_size;

std::remove(m\_dbFileName.c\_str());

std::rename(tmpFileName.c\_str(), m\_dbFileName.c\_str());

}

// удаление

void erase(const T& f)

{

T tmp;

bool E = false;

std::fstream in(m\_dbFileName);

const std::string tmpFileName = "tmpFile";

std::ofstream tmpFile(tmpFileName);

while (!in.eof())

{

in.read((char\*)&tmp, sizeof(T));

if (tmp == f && !E)

{

--m\_size;

E = true;

continue;

}

tmpFile.write((char\*)&tmp, sizeof(tmp));

}

in.close();

tmpFile.close();

std::remove(m\_dbFileName.c\_str());

std::rename(tmpFileName.c\_str(), m\_dbFileName.c\_str());

}

// обновление строки

void replace(const T& f, const int& index)

{

if (index >= m\_size || index < 0)

{

throw std::exception("out of range!");

}

std::fstream in(m\_dbFileName);

in.seekg(sizeof(T) \* index, std::ios::beg);

in.write((char\*)&f, sizeof(T));

}

int size() const

{

return m\_size;

}

// сортировка

void sort()

{

int k, i, j, tmp;

T t1, t2;

std::ifstream fi, fi1, fi2;

std::ofstream fo, fo1, fo2;

k = 1;

while (k < m\_size)

{

fi.open(m\_dbFileName);

fo1.open("smsort\_tmp\_1");

fo2.open("smsort\_tmp\_2");

if (!fi.eof())

{

fi.read((char\*)&t1, sizeof(T));

}

while (!fi.eof())

{

for (i = 0; i < k && !fi.eof(); i++)

{

fo1.write((char\*)&t1, sizeof(T));

fi.read((char\*)&t1, sizeof(T));

}

for (j = 0; j < k && !fi.eof(); j++)

{

fo2.write((char\*)&t1, sizeof(T));

fi.read((char\*)&t1, sizeof(T));

}

}

fi.close();

fo1.close();

fo2.close();

fo.open(m\_dbFileName);

fi1.open("smsort\_tmp\_1");

fi2.open("smsort\_tmp\_2");

if (!fi1.eof())

{

fi1.read((char\*)&t1, sizeof(T));

}

if (!fi2.eof())

{

fi2.read((char\*)&t2, sizeof(T));

}

while (!fi1.eof() && !fi2.eof())

{

i = 0;

j = 0;

while (i < k && j < k && !fi1.eof() && !fi2.eof())

{

if (t1 < t2)

{

fo.write((char\*)&t1, sizeof(T));

fi1.read((char\*)&t1, sizeof(T));

i++;

}

else

{

fo.write((char\*)&t2, sizeof(T));

fi2.read((char\*)&t2, sizeof(T));

j++;

}

}

while (i < k && !fi1.eof())

{

fo.write((char\*)&t1, sizeof(T));

fi1.read((char\*)&t1, sizeof(T));

i++;

}

while (j < k && !fi2.eof())

{

fo.write((char\*)&t2, sizeof(T));

fi2.read((char\*)&t2, sizeof(T));

j++;

}

}

while (!fi1.eof())

{

fo.write((char\*)&t1, sizeof(T));

fi1.read((char\*)&t1, sizeof(T));

}

while (!fi2.eof())

{

fo.write((char\*)&t2, sizeof(T));

fi2.read((char\*)&t2, sizeof(T));

}

fi1.close();

fi2.close();

fo.close();

k \*= 2;

}

fi1.close();

fi2.close();

fo.close();

remove("smsort\_tmp\_1");

remove("smsort\_tmp\_2");

}

// вставка с сохранением порядка

void addWithKeepingOrder(const T& f)

{

std::fstream file(m\_dbFileName);

int i = 0;

T tmp;

file.read((char\*)&tmp, sizeof(T));

while (tmp < f && i < m\_size)

{

file.read((char\*)&tmp, sizeof(T));

++i;

}

file.close();

this->addByIndex(f, i);

}

// извлечение строки

T operator[](const int& index) const

{

if (index >= m\_size || index < 0)

{

throw std::exception("out of range!");

}

std::fstream in(m\_dbFileName);

T tmp;

in.seekg(sizeof(tmp) \* static\_cast<unsigned int>(index), std::ios::beg);

in.read((char\*)&tmp, sizeof(T));

return tmp;

}

void executeIf(std::function<bool(const T&)> condition, std::function<void(T&)> action)

{

const std::string tmpFileName = "tmpFile";

std::ofstream tmpFile(tmpFileName);

T tmp;

for (int i = 0; i < m\_size; ++i)

{

tmp = this->operator[](i);

if (condition(tmp))

{

action(tmp);

}

tmpFile.write((char\*)&tmp, sizeof(tmp));

}

tmpFile.close();

std::remove(m\_dbFileName.c\_str());

std::rename(tmpFileName.c\_str(), m\_dbFileName.c\_str());

}

friend std::ostream& operator<<(std::ostream& out, const binaryDB<T> db)

{

for (int i = 0; i < db.size(); ++i)

{

out << db[i] << ' ';

}

return out;

}

private:

std::string m\_dbFileName;

int m\_size;

};

Файл main.cpp (с простым типом):

#include "binaryDB.h"

#include <iostream>

#include <cmath>

#include <ctime>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

srand(time(0));

int size = 7, maxNumber = 100;

// запись случайных данных в файл

ofstream out("db.txt");

int t;

for (int i = 0; i < size; ++i)

{

t = pow(-1, rand() % 2) \* (rand() % maxNumber);

out.write((char\*)&t, sizeof(int));

}

out.close();

binaryDB<int> db("db.txt");

cout << "Тест на int\nДанные из файла:\n";

std::cout << db << '\n';

db.add(pow(-1, rand() % 2) \* (rand() % maxNumber));

cout << "Добавить в конец случайное число от -100 до 100:\n" << db << '\n';

db.addByIndex(pow(-1, rand() % 2) \* (rand() % maxNumber), 3);

cout << "Добавить в позицию 3 случайное число от -100 до 100:\n" << db << '\n';

db.eraseByIndex(6);

cout << "Удалить число на 6й позиции:\n" << db << '\n';

int ind = rand() % db.size();

int del = db[ind];

db.erase(db[ind]);

cout << "Данные после удаления " << del <<" :\n" << db << '\n';

db.replace(pow(-1, rand() % 2) \* (rand() % maxNumber), 5);

cout << "Заменить элемент на 5й позиции случайным от -100 до 100:\n" << db << '\n';

cout << "Размер базы данных: " << db.size() << '\n';

db.sort();

cout << "Отсортированные данные:\n" << db << '\n';

db.addWithKeepingOrder(pow(-1, rand() % 2) \* (rand() % maxNumber));

cout << "Добавить случайное число от -100 до 100 с сохранением порядка:\n" << db << '\n';

}

Файл train.h (БД поездов):

#pragma once

#include <string>

#include <iostream>

constexpr int count\_of\_Intermediate\_stations = 10;

class train

{

public:

train();

train(const int& a\_number,

const bool const\* a\_week,

const int& a\_h\_departure,

const int& a\_m\_departure,

const int& a\_h\_way,

const int& a\_m\_way,

const int& a\_departure\_station,

const int& a\_destination\_station,

const bool const\* a\_Intermediate\_stations);

friend std::ostream& operator<<(std::ostream& out, const train& t);

int departure\_station() const;

int destination\_station() const;

const bool\* Intermediate\_stations() const;

private:

// номер поезда

int m\_number;

// дни недели отправления

bool m\_week[7];

// время отправления

int m\_h\_departure, m\_m\_departure;

// время в пути

int m\_h\_way, m\_m\_way;

// станция отправления

int m\_departure\_station;

// станция назначения

int m\_destination\_station;

// промежуточные станции

bool m\_Intermediate\_stations[count\_of\_Intermediate\_stations];

};

Файл train.cpp (БД поездов):

#include "train.h"

train::train() : m\_number(-1), m\_h\_departure(-1), m\_m\_departure(-1),

m\_h\_way(-1), m\_m\_way(-1), m\_departure\_station(-1), m\_destination\_station(-1)

{

for (int i = 0; i < 7; i++)

{

m\_week[i] = false;

}

}

train::train(const int& a\_number,

const bool const\* a\_week,

const int& a\_h\_departure,

const int& a\_m\_departure,

const int& a\_h\_way,

const int& a\_m\_way,

const int& a\_departure\_station,

const int& a\_destination\_station,

const bool const\* a\_Intermediate\_stations)

: m\_number(a\_number),

m\_h\_departure(a\_h\_departure),

m\_m\_departure(a\_m\_departure),

m\_h\_way(a\_h\_way),

m\_m\_way(a\_m\_way),

m\_departure\_station(a\_departure\_station),

m\_destination\_station(a\_destination\_station)

{

for (int i = 0; i < 7; i++)

{

m\_week[i] = a\_week[i];

}

for (int i = 0; i < count\_of\_Intermediate\_stations; i++)

{

m\_Intermediate\_stations[i] = a\_Intermediate\_stations[i];

}

}

int train::departure\_station() const

{

return m\_departure\_station;

}

int train::destination\_station() const

{

return m\_destination\_station;

}

const bool\* train::Intermediate\_stations() const

{

return m\_Intermediate\_stations;

}

std::ostream& operator<<(std::ostream& out, const train& t)

{

out << "number: " << t.m\_number << '\n';

out << "days of the week of departure: ";

for (int i = 0; i < 7; ++i)

{

if (t.m\_week[i])

{

out << i << ' ';

}

}

out << '\n';

out << "time departure: " << t.m\_h\_departure << ':' << t.m\_m\_departure << '\n';

out << "travel time: " << t.m\_h\_way << ':' << t.m\_m\_way << '\n';

out << "departure station: " << t.m\_departure\_station << '\n';

out << "destination station: " << t.m\_destination\_station << '\n';

out << "intermediate stations: ";

for (int i = 0; i < count\_of\_Intermediate\_stations; ++i)

{

if (t.m\_Intermediate\_stations[i])

{

out << i << ' ';

}

}

out << '\n';

return out;

}

Файл main.cpp (БД поездов):

#include "binaryDB.h"

#include "train.h"

#include <iostream>

#include <cmath>

#include <ctime>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

srand(time(0));

binaryDB<train> db("db.txt");

cout << "Начальные данные: ";

cout << db << '\n';

int number\_station;

cout << "Введите номер станции: ";

cin >> number\_station;

cout << "Поезда следующий в " << number\_station <<" станцию:\n";

std::function<bool(const train&)> condition = [&](const train& x) -> bool

{

return x.departure\_station() == number\_station

|| x.destination\_station() == number\_station

|| x.Intermediate\_stations()[number\_station];

};

std::function<void(train&)> action = [](train& x)

{

cout << x << '\n';

};

db.executeIf(condition, action);

}