Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Основы алгоритмизации и программирования

Отчёт

по учебной практике(ознакомительной)

на тему

Программа управления расписанием поездов

Студент Е.О.Лукьянов

Руководитель А.М.Ковальчук

МИНСК 2023

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ..............................................................................................................3

1. ЗАДАЧИ И ЦЕЛИ ПРОГРАММЫ. КРАТКИЕ СВЕДЕНИЯ О РАЗРАБОТКЕ ПРОГРАММЫ...............................................................................4

2. ВХОДНЫЕ ДАННЫЕ.........................................................................................6

3. ВЫХОДНЫЕ ДАННЫЕ.....................................................................................8

4. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ................................................9

4.1. БЛОК-СХЕМА ГОЛОВНОГО МОДУЛЯ......................................................9

4.2. АЛГОРИТМ ПО ШАГАМ ФУНКЦИИ dijkstra()..........................................9

4.3. АЛГОРИТМ ПО ШАГАМ ФУНКЦИИ min\_distance()...............................11

4.4. АЛГОРИТМ ПО ШАГАМ ФУНКЦИИ napravl()........................................12

5. КОД ПРОГРАММЫ..........................................................................................13

5.1. main.cpp...........................................................................................................13

5.2. functions.cpp....................................................................................................13

6. РЕЗУЛЬТАТЫ...................................................................................................33

ЗАКЛЮЧЕНИЕ ....................................................................................................37

СПИСОК ЛИТЕРАТУРЫ ....................................................................................38

**ПРИЛОЖЕНИЕ А**...............................................................................................39

**ВВЕДЕНИЕ**

Язык C является одним из самых быстрых и популярных языков во всём мире. С позволяет осуществлять большинство преобразований типов. Контроль за выполнением этих преобразований, а также проверка некоторых ошибок (например, выход за границы массива) возлагается на программиста. Реализованная в С возможность напрямую манипулировать битами, байтами, словами и указателями необходима для программирования на системном уровне. Язык С считается структурированным языком. Отличительной чертой структурированного языка является разделение кода и данных. Одним из способов решения этой проблемы является использование подпрограмм (функций), широко использующих локальные переменные. Необходимо отметить, что излишнее использование глобальных переменных может приводить к фатальным ошибкам. Как и ряд других структурированных языков, С поддерживает ряд операторов цикла, условных операторов и операторов ветвления.

Язык С содержит стандартные библиотеки, предоставляющие функции, выполняющие наиболее типичные задачи. Эти библиотеки легко могут быть подключены, а также дополнены. Язык С позволяет разбивать программу на части и выполнять их раздельную компиляцию. Откомпилированные таким образом файлы объединяются для создания полного объектного кода. Преимущество раздельной компиляции в том, что при изменении одного файла не требуется перекомпиляции всей программы.

1. **ЗАДАЧИ И ЦЕЛИ ПРОГРАММЫ. КРАТКИЕ СВЕДЕНИЯ О РАЗРАБОТКЕ ПРОГРАММЫ**

Цель программы "Управление расписанием поездов" - создать эффективную систему управления расписанием поездов, которая позволит пользователям узнавать точную информацию об их поездках и добираться до пункта назначения по рациональным маршрутам.

Основными задачами данной программы являются выдача справки о маршруте, то есть время отправления и прибытия, расстояние между городами, поиск кратчайшего пути между ними, указание станций пересадок на маршруте (в случае их присутствия), поиск времени прибытия на станции пересадок и времени отбытия со станций пересадок (в случае их присутствия).

Для реализации данной программы был использован модифицированный алгоритм Дейкстры для учёта станций пересадок. Алгоритм Дейкстры - это алгоритм нахождения кратчайшего пути во взвешенном графе с неотрицательными весами ребер. Алгоритм начинает работу с одной вершины графа и находит кратчайшие пути до всех остальных вершин.

Алгоритм Дейкстры работает следующим образом:

1. Изначально все вершины помечаются как не посещенные, а расстояние до всех вершин, кроме стартовой, считается бесконечностью.

2. Устанавливаем расстояние от стартовой вершины до нее самой равным 0.

3. Выбираем вершину с наименьшим расстоянием из еще не посещенных вершин и помечаем ее как посещенную.

4. Обновляем расстояния до смежных вершин через выбранную вершину. Если расстояние до смежной вершины короче через текущую вершину, то обновляем его.

5. Повторяем шаги 3 и 4, пока все вершины не будут помечены как посещенные.

6. После завершения работы алгоритма расстояния до всех вершин графа будут определены и мы можем найти кратчайший путь до любой вершины.

Алгоритм Дейкстры с модификацией для учета станций пересадок работает следующим образом:

1. Создать дополнительную вершину-пересадку для каждой пары станций, между которыми нет прямого пути.

2. Соединить каждую из этих дополнительных вершин со всеми станциями, на которые можно пересесть с линии каждой из оригинальных станций.

3. Запустить алгоритм Дейкстры для поиска кратчайшего пути между стартовой и конечной станциями, учитывая при этом вершины-пересадки.

4. При построении кратчайшего пути учитывать, что на пути может быть несколько вершин-пересадок. В таком случае необходимо вывести информацию о пересадке и продолжить построение пути.

1. **ВХОДНЫЕ ДАННЫЕ**

f.txt - файл с информацией о всех городах.

#define N 20 // размер строковых переменных

#define t 10 // кол-во времён в массивах времён

#define V 32 // кол-во городов

struct s1 //объявление структуры, хранящей информацию о городе (населённого пункта)

{

char City[N]; //название города (населённого пункта)

char Time1[t][N]; //массив времён отбытия на t элементов

char Time2[t][N]; //массив времён прибытия на t элементов

int Line; //номер железно-дорожной линии, на которой расположен город (населённый пункт)

int Dist; //расстояние от данного города (населённого пункта) до Минска

};

struct s2 //объявление структуры, хранящей информацию о поездке пользователя

{

char CityFrom[N]; //название города (населённого пункта), из которого пользователь отправляется

char CityTo[N]; //название города (населённого пункта), в который пользователь отправляется

char Peresadki[3][N]; //массив станций пересадок, максимальное кол-во 3

char TimePeres1[3][t][N]; // массив времён отправления поездов со станций пересадки

char TimePeres2[3][t][N]; //массив времён отправления поездов со станций пересадки

char TimeFrom[t][N]; //массив времён отправления из данного города (населённого пункта)

char TimeTo[t][N]; //массив времён прибытия в данный город (населённый пункт)

int Distance; //расстояние между городом отправления и городои прибытия

};

struct Edge //структура для хранения информации о ребре между двумя городами

{

int u; // номер первой станции

int v; // номер второй станции

int weight; // вес ребра (расстояние между данными городами)

};

1. **ВЫХОДНЫЕ ДАННЫЕ**

Выходными данными является 1 текстовый файл:

result.txt – файл с информацией о поездке пользователя.

1. **РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

**4.1. БЛОК-СХЕМА ГОЛОВНОГО МОДУЛЯ**

Блок-схема головного модуля представлен в **ПРИЛОЖЕНИИ А.**

* 1. **. АЛГОРИТМ ПО ШАГАМ ФУНКЦИИdijksra()**

1. Начало алгоритма.
2. Объявление и инициализация переменных:

Вх:

int graph[V][V] – граф, хранящий расстояния между всеми вершинами.

int start – индекс города, из которого едет пользователь.

int end – индекс города, в который едет пользователь.

struct s1\* info – массив структур с информацией о всех городах.

struct s2\* infoWay – массив структур для хранения информации о поездке пользователя.

int\* num – указатель на переменную, хранящую информацию о количестве пересадок на маршруте.

Промеж:

int dist[V] – массив расстояний от стартовой вершины.

int visited[V] – массив посещенных вершин.

int prev[V] – массив предыдущих вершин на пути. int number = -1 – переменная для хранения кол-ва пересадок на маршруте.

int numb = 0 – переменная для записи нужных времён отбытия со станций пересадок и прибытия на станции пересадок.

int current = end – текущее положение начинается с конечного города поездки для прохождения по всем городам на маршруте.

int IndexTransfer – индекс города, где будет осуществлена ближайшая пересадка.

int nap – переменная для хранения направления поездки.

int ind = prev[current] – сохранение последовательности предыдущих вершин на пути.

int flag = 0 – город не является пересадкой.

Вых:

struct s2\* infoWay – массив структур с информацией о поездке пользователя.

1. Цикл от i = 0 до V инициализации переменных для каждого города.
2. Dist[i] = INT\_MAX – инициализация всех расстояний как “бесконечность”. Visited[i] = 0 – отметка всех вершин как не посещённых. Prev[i] = -1 – отсутствие предыдущих вершин на пути.
3. Конец цикла i.
4. Dist[start] = 0 – расстояние от начального города до самого себя равняется 0.
5. Цикл от i = 0 до V – 1 для поиска кратчайшего пути для всех вершин графа.
6. int u = min\_distance(dist, visited) – вызов функции для поиска вершины с наименьшей стоимостью среди вершин, ещё не включённых в кратчайший путь. Visited[u] = 1 – отметка выбранной вершины как посещённой.
7. Цикл от v = 0 до V для обновления расстояний до смежных вершин, учитывая станции пересадок.
8. Если вершина ещё не включена в кратчайший путь, есть ребро из выбранной вершины до смежной вершины и расстояние от источника до выбранной вершины плюс стоимость ребра меньше текущего расстояния до смежной вершины, то обновляем расстояние до смежной вершины: dist[v] = dist[u] + graph[u][v];  
   prev[v] = u;
9. Конец цикла v.
10. Конец цикла i.
11. Если расстояние до конечного города поездки равно “бесконечности”, то путь не найден.
12. Если начальный или конечный город является городом-пересадкой, то flag = 1.
13. Пока не пройдём всю последовательность предыдущих вершин на маршруте, выполняем…
14. Если линии данной станции и предыдущей на маршруте разные, то number = number + 1 и становимся на предыдущую вершину на маршруте (current = prev[current]).
15. Numb = number – сохраняем кол-во пересадок на маршруте для чтения времён прибытия на станцию пересадки и отбытия со станции пересадки.
16. Если number > 0, то…
17. Пока не пройдём всю последовательность предыдущих вершин на маршруте, выполняем…
18. Если линии данной станции и предыдущей на маршруте разные, то с помощью функции strcpy(infoWay[0].Peresadki[numb-1], info[prev[current]].City) копируем название города-пересадки с конца. Numb = numb – 1 – уменьшаем кол-во оставшихся пересадок.
19. Если numb == 0, то выходим из цикла, так как названия всех городов-пересадок уже скопированы.
20. Становимся на предыдущую вершину на маршруте.
21. С помощью функции IndexTransfer = find\_city\_index(info, infoWay[0].Peresadki[0]) получаем индекс ближайшей пересадки на маршруте.
22. С помощью функции nap = napravl(info, start, IndexTransfer, end, number) определяем направление поездки ( если nap = 1, то поездка от центра, если nap = 0, то к центру).
23. С помощью функции infoWay = GetTime(info, infoWay, start, end, nap, number) считываем нужные времена отбытия из начального города и прибытия в конечный город.
24. С помощью функции infoWay = GetTimeOfPeresadki(info, infoWay, nap, number) считываем нужные времена прибытия в город-пересадку и отбытия из него.
25. infoWay[0].Distance = dist[end] – сохраняем кратчайшее расстояние между начальным и конечным городами.
26. \*num = number – сохраняем кол-во пересадок для дальнейшего использования.
27. Возвращаем infoWay.
28. Конец алгоритма.
    1. **. АЛГОРИТМ ПО ШАГАМ ФУНКЦИИ min\_distance**():
29. Начало алгоритма.
30. Объявление и инициализация переменных:

Вх:

int dist[V] – массив расстояний от стартовой вершины.

int visited[V] – массив посещенных вершин.

int min = INT\_MAX – изначальное расстояние от источника принимается за бесконечно большое число.

Промеж:

int min – кратчайшее расстояние до вершины.

int min\_index – индекса вершины с кратчайшим расстоянием

Вых:

int min\_index – переменная направления поездки (если napravl = 1, то поездка от центра, если napravl = 0, то поездка к центру).

1. Если num == 0, то второе расстояние принимаем равным расстоянию до города, в который едет пользователь (dist2 = info[end].Dist).
2. Иначе второе расстояние принимаем равным расстоянию до города, в который едет пользователь (dist2 = info[end].Dist).
3. Если (dist1 – dist2) < 0, то napravl = 1, то есть поездка от центра, иначе napravl = 0, то есть поездка к центру.
4. Возвращаем napravl.
5. Конец алгоритма.
   1. **. АЛГОРИТМ ПО ШАГАМ ФУНКЦИИ napravl():**
6. Начало алгоритма.
7. Объявление и инициализация переменных:

Вх:

int index1 – индекс города, из которого едет пользователь.

int index2 – индекс ближайшего города-пересадки (если существует).

int end – индекс города, в который едет пользователь.

int num – количество пересадок на маршруте.

struct s1\* info – массив структур с информацией о всех городах.

Промеж:

int dist1 = info[index1].Dist – расстояние до начального города.

int dist2 – расстояние до конечного города или до ближайшего города-пересадки.

int napravl – переменная для хранения направления поездки (если napravl = 1, то поездка от центра, если napravl = 0, то поездка к центру).

Вых:

int napravl – переменная направления поездки (если napravl = 1, то поездка от центра, если napravl = 0, то поездка к центру).

1. Если num == 0, то второе расстояние принимаем равным расстоянию до города, в который едет пользователь (dist2 = info[end].Dist).
2. Иначе второе расстояние принимаем равным расстоянию до города, в который едет пользователь (dist2 = info[end].Dist).
3. Если (dist1 – dist2) < 0, то napravl = 1, то есть поездка от центра, иначе napravl = 0, то есть поездка к центру.
4. Возвращаем napravl.
5. Конец алгоритма.
6. **КОД ПРОГРАММЫ**

**5.1. main.cpp**

Далее представлен код функции main() программы управления расписанием поездов с комментариями на языке программирования С.

int main()

{

system("cls"); // очистка командной строки

int n; // переменная для хранения кол-ва считанных структур из текстового файла f.txt

int num; // переменная для хранения кол-ва станций пересадок

int graph[V][V] {…};

struct s1\* info; // объявление массива структур для хранения информации о всех городах

struct s2\* infoWay = (struct s2\*)calloc(1, sizeof(struct s2)); // объявление и выделение памяти массива структур для хранения информации о поездке пользователя

ReadInfoFromFile1(&info, &n);

infoWay = InitialMenu(graph, info, infoWay, n, &num); // пользовательское меню

free(info);

free(infoWay);

return 0;

}

**5.2. functions.cpp**

Далее представлен код программы управления расписанием поездов с комментариями на языке программирования С.

Функция записи массива структур с информацией о поездке в файл result.txt:

void InputInformationInFileS2(struct s2\* infoWay, int num) // запись массива структур в текстовый файл result.txt

{

FILE\* fp = fopen("result.txt", "w");

CheckFile(fp);

if (num == 0) // если на маршруте нет пересадок, то запись в файл будет иметь следующий вид

{

fprintf(fp, "%s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s -> %d", infoWay[0].CityFrom, infoWay[0].TimeFrom[0], infoWay[0].TimeFrom[1], infoWay[0].TimeFrom[2],

infoWay[0].TimeFrom[3], infoWay[0].TimeFrom[4], infoWay[0].TimeFrom[5], infoWay[0].TimeFrom[6], infoWay[0].TimeFrom[7], infoWay[0].TimeFrom[8], infoWay[0].TimeFrom[9], infoWay[0].CityTo, infoWay[0].TimeTo[0], infoWay[0].TimeTo[1],

infoWay[0].TimeTo[2], infoWay[0].TimeTo[3], infoWay[0].TimeTo[4], infoWay[0].TimeTo[5], infoWay[0].TimeTo[6], infoWay[0].TimeTo[7], infoWay[0].TimeTo[8], infoWay[0].TimeTo[9], infoWay[0].Distance);

}

else if (num == 1) // если на маршруте есть 1 (одна) пересадка, то запись в файл будет иметь следующий вид

{

fprintf(fp, "%s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s -> %d", infoWay[0].CityFrom, infoWay[0].TimeFrom[0], infoWay[0].TimeFrom[1], infoWay[0].TimeFrom[2],

infoWay[0].TimeFrom[3], infoWay[0].TimeFrom[4], infoWay[0].TimeFrom[5], infoWay[0].TimeFrom[6], infoWay[0].TimeFrom[7], infoWay[0].TimeFrom[8], infoWay[0].TimeFrom[9], infoWay[0].Peresadki[0],

infoWay[0].TimePeres1[0][0], infoWay[0].TimePeres1[0][1], infoWay[0].TimePeres1[0][2], infoWay[0].TimePeres1[0][3], infoWay[0].TimePeres1[0][4], infoWay[0].TimePeres1[0][5], infoWay[0].TimePeres1[0][6],

infoWay[0].TimePeres1[0][7], infoWay[0].TimePeres1[0][8], infoWay[0].TimePeres1[0][9], infoWay[0].Peresadki[0], infoWay[0].TimePeres2[0][0], infoWay[0].TimePeres2[0][1], infoWay[0].TimePeres2[0][2],

infoWay[0].TimePeres2[0][3], infoWay[0].TimePeres2[0][4], infoWay[0].TimePeres2[0][5], infoWay[0].TimePeres2[0][6], infoWay[0].TimePeres2[0][7], infoWay[0].TimePeres2[0][8], infoWay[0].TimePeres2[0][9],

infoWay[0].CityTo, infoWay[0].TimeTo[0], infoWay[0].TimeTo[1], infoWay[0].TimeTo[2], infoWay[0].TimeTo[3], infoWay[0].TimeTo[4], infoWay[0].TimeTo[5], infoWay[0].TimeTo[6], infoWay[0].TimeTo[7],

infoWay[0].TimeTo[8], infoWay[0].TimeTo[9], infoWay[0].Distance);

}

else // если на маршруте более 1 (одной) пересадки, то запись в файл будет иметь следующий вид

{

fprintf(fp, "%s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s -> %d", infoWay[0].CityFrom, infoWay[0].TimeFrom[0], infoWay[0].TimeFrom[1], infoWay[0].TimeFrom[2],

infoWay[0].TimeFrom[3], infoWay[0].TimeFrom[4], infoWay[0].TimeFrom[5], infoWay[0].TimeFrom[6], infoWay[0].TimeFrom[7], infoWay[0].TimeFrom[8], infoWay[0].TimeFrom[9], infoWay[0].Peresadki[0],

infoWay[0].TimePeres1[0][0], infoWay[0].TimePeres1[0][1], infoWay[0].TimePeres1[0][2], infoWay[0].TimePeres1[0][3], infoWay[0].TimePeres1[0][4], infoWay[0].TimePeres1[0][5], infoWay[0].TimePeres1[0][6],

infoWay[0].TimePeres1[0][7], infoWay[0].TimePeres1[0][8], infoWay[0].TimePeres1[0][9], infoWay[0].Peresadki[0], infoWay[0].TimePeres2[0][0], infoWay[0].TimePeres2[0][1], infoWay[0].TimePeres2[0][2],

infoWay[0].TimePeres2[0][3], infoWay[0].TimePeres2[0][4], infoWay[0].TimePeres2[0][5], infoWay[0].TimePeres2[0][6], infoWay[0].TimePeres2[0][7], infoWay[0].TimePeres2[0][8], infoWay[0].TimePeres2[0][9], infoWay[0].Peresadki[1],

infoWay[0].TimePeres1[1][0], infoWay[0].TimePeres1[1][1], infoWay[0].TimePeres1[1][2], infoWay[0].TimePeres1[1][3], infoWay[0].TimePeres1[1][4], infoWay[0].TimePeres1[1][5], infoWay[0].TimePeres1[1][6],

infoWay[0].TimePeres1[1][7], infoWay[0].TimePeres1[1][8], infoWay[0].TimePeres1[1][9], infoWay[0].Peresadki[1], infoWay[0].TimePeres2[1][0], infoWay[0].TimePeres2[1][1], infoWay[0].TimePeres2[1][2],

infoWay[0].TimePeres2[1][3], infoWay[0].TimePeres2[1][4], infoWay[0].TimePeres2[1][5], infoWay[0].TimePeres2[1][6], infoWay[0].TimePeres2[1][7], infoWay[0].TimePeres2[1][8], infoWay[0].TimePeres2[1][9],

infoWay[0].CityTo, infoWay[0].TimeTo[0], infoWay[0].TimeTo[1], infoWay[0].TimeTo[2], infoWay[0].TimeTo[3], infoWay[0].TimeTo[4], infoWay[0].TimeTo[5], infoWay[0].TimeTo[6], infoWay[0].TimeTo[7],

infoWay[0].TimeTo[8], infoWay[0].TimeTo[9], infoWay[0].Distance);

}

fprintf(fp, "\n");

fclose(fp);

}

Функция проверки существования введённого города в массиве структур с информацией о всех городах:

int CheckCity(struct s1\* info,char\* str) // проверка, есть ли введённый город в массиве структур

{

int flag = 0; // пусть изначально города нет в массиве структур

for (int i = 0; i < V; i++) // цикл для прохода по всем элементам (городам) массива структур

if (strcmp(info[i].City, str) == 0)

flag = 1; // если введённый город совпал с городом из массива структур, то flag = 1

if (flag == 0)

printf("\nThere is no this city. Try again please!");

return flag;

}

Функция ввода названия города (населённого пункта), в который едет пользователь:

char\* InputCityTo(struct s1\* info) // ввод названия города (населённого пункта), в который пользователь едет

{

int flag; // есть ли введённый город в массиве структур

char\* StrCityTo;

do

{

rewind(stdin);

flag = 0;

printf("\nEnter the city, where you are going: ");

StrCityTo = (char\*)calloc(N, sizeof(char)); // выделение динамической памяти для ввода названия города

fgets(StrCityTo, N, stdin);

DelEnter(StrCityTo);

flag = CheckCity(info, StrCityTo); // проверка, есть ли введённый город в массиве структур

} while (!flag); // если данного города нет, то пользователь вводит город снова

return StrCityTo;

}

Функция ввода названия города (населённого пункта), из которого едет пользователь:

char\* InputCityFrom(struct s1\* info) // ввод названия города (населённого пункта), из которого едет пользователь

{

int flag; // есть ли введённый город в массиве структур

char\* StrCityFrom;

do

{

rewind(stdin);

flag = 0;

printf("\nEnter the city, where you are coming from: ");

StrCityFrom = (char\*)calloc(N, sizeof(char)); // выделение динамической памяти для ввода названия города

fgets(StrCityFrom, N, stdin);

DelEnter(StrCityFrom);

flag = CheckCity(info, StrCityFrom); // проверка, есть ли введённый город в массиве структур

} while (!flag); // если данного города нет, то пользователь вводит город снова

return StrCityFrom;

}

Функция проверки открытия файла:

void CheckFile(FILE\* fp) // функция для проверки открытия файла

{

if (fp == NULL) // проверка открытия файла

{

printf("ERROR");

return;

}

}

Функция удаления ‘\n’ с конца строки:

void DelEnter(char str[]) // удаление \n при вводе информации

{

int l = strlen(str);

str[l - 1] = '\0';

}

Функция чтения информации о всех городах в массив струкутр info:

void ReadInfoFromFile1(struct s1\*\* info, int\* n) // чтение информации о всех городах в массив структур info

{

FILE\* fp = fopen("f.txt", "r");

CheckFile(fp);

int num = NumberOfStructs(fp); // узнаём кол-во структур в файле

\*info = (struct s1\*)calloc(num , sizeof(struct s1)); // выделение динамической памяти для массива структур

fseek(fp, 0, SEEK\_SET); // устанавливаем курсор на начало файал f.txt

for (int i = 0; i < num; i++) // цикл для чтения информации о num городах в массив структур из файла f.txt

fscanf(fp, "%s %d %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %d", &((\*info + i)->City), &((\*info + i)->Line), &((\*info + i)->Time1[0]), &((\*info + i)->Time1[1]), &((\*info + i)->Time1[2]),

&((\*info + i)->Time1[3]), &((\*info + i)->Time1[4]), &((\*info + i)->Time1[5]), &((\*info + i)->Time1[6]), &((\*info + i)->Time1[7]), &((\*info + i)->Time1[8]), &((\*info + i)->Time1[9]),

&((\*info + i)->Time2[0]), &((\*info + i)->Time2[1]), &((\*info + i)->Time2[2]), &((\*info + i)->Time2[3]), &((\*info + i)->Time2[4]), &((\*info + i)->Time2[5]), &((\*info + i)->Time2[6]),

&((\*info + i)->Time2[7]), &((\*info + i)->Time2[8]), &((\*info + i)->Time2[9]), &((\*info + i)->Dist)); // чтение полей структуры

fclose(fp);

\*n = num;

}

Функция вывода массива структур, хранящего информацию о всех городах:

void PrintStruct\_s1(struct s1\* info, int n) // вывод массива структур, хранящего информацию о всех городах

{

printf("\nThe information about all cities:\n");

printf("+----------------+-------+-------+----------+\n");

printf("| %-14s | %-5s | %-5s | %-5s | \n", "City", "Time1", "Time2", "Distance");

printf("+----------------+-------+-------+----------+\n");

for (int i = 0; i < n; i++) // цикл для вывода информации о всех городах

{

printf("| %-14s | %-5s | %-5s | %-8d | \n", info[i].City, info[i].Time1[0], info[i].Time2[0], info[i].Dist);

for (int j = 1; j < t; j++) // цикл для вывода массива времён

printf("| \t\t | %-5s | %-5s | \t | \n",info[i].Time1[j], info[i].Time2[j]);

printf("+----------------+-------+-------+----------+\n");

}

}

Функция вывода структуры, хранящей информацию о поездке пользователя:

void PrintStruct\_s2(struct s2\* infoWay, int num) // вывод структуры, хранящей информации о поездке пользователя

{

if (num == 0) // если пересадок нет, то форма вывода будет следующей

{

printf("+----------------+----------+-----------------+---------+-----------+\n");

printf("| %-14s | %-8s | %-15s | %-5s | %-9s | \n", "CityFrom", "TimeFrom", "CityTo", "TimeTo", "Distance");

printf("+----------------+----------+-----------------+---------+-----------+\n");

printf("| %-14s | %-8s | %-15s | %-7s | %-9d | \n", infoWay[0].CityFrom, infoWay[0].TimeFrom[0], infoWay[0].CityTo, infoWay[0].TimeTo[0], infoWay[0].Distance);

for (int j = 1; j < t; j++) // цикл для вывода массива времён

printf("| \t\t | %-8s | \t\t | %-7s | \t | \n", infoWay[0].TimeFrom[j], infoWay[0].TimeTo[j]);

printf("+----------------+----------+-----------------+---------+-----------+\n");

}

else if (num == 1) // если пересадка 1 (одна), то форма вывода будет следующей

{

printf("+----------------+----------+----------------+----------++----------+-----------------+---------+-----------+\n");

printf("| %-14s | %-8s | %-14s | %-8s || %-8s | %-15s | %-6s | %-9s | \n", "CityFrom", "TimeFrom", "Peresadka", "Time1", "Time2", "CityTo", "TimeTo", "Distance");

printf("+----------------+----------+----------------+----------++----------+-----------------+---------+-----------+\n");

printf("| %-14s | %-8s | %-14s | %-8s || %-8s | %-15s | %-7s | %-9d | \n", infoWay[0].CityFrom, infoWay[0].TimeFrom[0], infoWay[0].Peresadki[0], infoWay[0].TimePeres1[0][0], infoWay[0].TimePeres2[0][0], infoWay[0].CityTo, infoWay[0].TimeTo[0], infoWay[0].Distance);

for (int j = 1; j < t; j++) // цикл для вывода массива времён

printf("| \t\t | %-8s | \t\t | %-8s || %-8s | \t\t | %-7s | \t | \n", infoWay[0].TimeFrom[j], infoWay[0].TimePeres1[0][j], infoWay[0].TimePeres2[0][j], infoWay[0].TimeTo[j]);

printf("+----------------+----------+----------------+----------++----------+-----------------+---------+-----------+\n");

}

else if (num == 2) // если пересадок 2 (две), то форма вывода будет следующей

{

printf("+----------------+----------+----------------+----------++----------+-----------------+---------++---------+-----------------+---------+-----------+\n");

printf("| %-14s | %-8s | %-14s | %-8s || %-8s | %-15s | %-6s || %-7s | %-15s | %-5s | %-9s | \n", "CityFrom", "TimeFrom", "Peresadka1", "Time1", "Time2", "Peresadka2", "Time1", "Time2", "CityTo", "TimeTo", "Distance");

printf("+----------------+----------+----------------+----------++----------+-----------------+---------++---------+-----------------+---------+-----------+\n");

printf("| %-14s | %-8s | %-14s | %-8s || %-8s | %-15s | %-7s || %-7s | %-15s | %-7s | %-9d | \n", infoWay[0].CityFrom, infoWay[0].TimeFrom[0], infoWay[0].Peresadki[0], infoWay[0].TimePeres1[0][0], infoWay[0].TimePeres2[0][0], infoWay[0].Peresadki[1], infoWay[0].TimePeres1[1][0], infoWay[0].TimePeres2[1][0], infoWay[0].CityTo, infoWay[0].TimeTo[0], infoWay[0].Distance);

for (int j = 1; j < t; j++) // цикл для вывода массива времён

printf("| \t\t | %-8s | \t\t | %-8s || %-8s | \t\t | %-7s || %-7s | \t\t | %-7s | \t | \n", infoWay[0].TimeFrom[j], infoWay[0].TimePeres1[0][j], infoWay[0].TimePeres2[0][j], infoWay[0].TimePeres1[1][j], infoWay[0].TimePeres2[1][j], infoWay[0].TimeTo[j]);

printf("+----------------+----------+----------------+----------++----------+-----------------+---------++---------+-----------------+---------+-----------+\n");

}

else return;

}

Функция поиска количества структур в текстовом файле:

int NumberOfStructs(FILE\* fp) // поиск кол-ва структур в файле

{

int i = 0;

char str[256];

while (fgets(str, 256, fp) != NULL) // считываем строки до тез пор, пока не дойдём до конца

i++;

return i;

}

Функция возврата пользовательского меню:

int returnFunc() // функция для возвращения пользовательского меню

{

int choice;

printf("\nEnter the any button to return the menu: ");

scanf\_s("%d", &choice);

rewind(stdin);

return 1;

}

Функция пользовательского меню:

struct s2\* InitialMenu(int graph[V][V], struct s1\* info, struct s2\* infoWay, int n, int\* num) // пользовательское меню

{

system("cls");

printf("1.Enter information about the trip.\n");

printf("2.Display information about all cities of the railway.\n");

printf("3.Display information about your last trip.\n");

printf("4.Save your trip way information in TXT file.\n");

printf("5.Exit.\n");

char\* StrCityFrom = (char\*)calloc(N, sizeof(char)); // объявление и выделение памяти для хранения названия города, из которого едет пользователь

char\* StrCityTo = (char\*)calloc(N, sizeof(char)); // объявление и выделение памяти для хранения названия города, в который пользователь едет

int IndexFrom; // индекс города, из которого едет пользователь, в массиве структур

int IndexTo; // индекс города, в который едет пользователь, в массиве структур

int choice = getchar(); // выбор опции

rewind(stdin);

switch (choice)

{

case '1': // ввод информации о поездке

system("cls"); // очитска консоли

StrCityFrom = InputCityFrom(info); // ввод названия города, из которого едет пользователь

StrCityTo = InputCityTo(info); // ввод названия города, в который едет пользователь

strcpy(infoWay[0].CityFrom, StrCityFrom); // сохранение названий городов в структуру, хранящую информацию о поездке

strcpy(infoWay[0].CityTo, StrCityTo);

IndexFrom = find\_city\_index(info, StrCityFrom); // получение индекса города, из которого едет пользователь, в массиве структур

IndexTo = find\_city\_index(info, StrCityTo); // получение индекса города, в который едет пользователь, в массиве структур

infoWay = dijkstra(graph, IndexFrom, IndexTo, info, infoWay, num); // алгорит Дейкстры для получения информации о станциях пересадок, если такие есть, нахождение кратчайшего пути между станциями и сохранение данной информации в структуру

returnFunc(); // возвращение в пользовательское меню

infoWay = InitialMenu(graph, info, infoWay, n, num); // вывод пользовательского меню и сохранение информации в структуру

break;

case '2': // вывод информации о всех городах

system("cls");

printf("The information about all cities:\n");

PrintStruct\_s1(info, n); // вывод массива структур, хранящего информацию о всех городах

returnFunc();

infoWay = InitialMenu(graph, info, infoWay, n, num);

break;

case '3': // вывод информации о поездке

system("cls");

printf("The information about your trip:\n");

if (infoWay[0].CityFrom == NULL) // если в стрктуре отсутствует инофрмация о городе, из которого едет пользователь, то информции о поездке нет

{

printf("There is no information about your trip!\n");

returnFunc();

infoWay = InitialMenu(graph, info, infoWay, n, num);

break;

}

else

PrintStruct\_s2(infoWay, \*num); // иначе вывод структуры, хранящей информацию о поездке пользователя

returnFunc();

infoWay = InitialMenu(graph, info, infoWay, n, num);

break;

case '4': // сохранение информации о поездке пользователя в текстовый файл

system("cls");

InputInformationInFileS2(infoWay, \*num); // запись структуры в текстовый файл

printf("TXT file is successfuly created. Name of this file is result.txt.\n");

returnFunc();

infoWay = InitialMenu(graph, info, infoWay, n, num);

break;

case '5': // закончить выполнение программы

system("cls");

printf("Thank you fo using our service. Goodbye!");

break;

default: // при вводе другого варианта, выводится сообщение

printf("There is no this option. Try again please!");

}

return infoWay;

}

Функция поиска минимального расстояния:

int min\_distance(int dist[], int visited[]) // функция для поиска индекса вершины с наименьшим расстоянием от стартовой вершины

{

int min = INT\_MAX, min\_index; // изначальное расстояние принимается за бесконечно большое число - min, min index -

for (int i = 0; i < V; i++) // цикл для прохождения по всем городам

{

if (visited[i] == 0 && dist[i] <= min)

{

min = dist[i];

min\_index = i; // запоминаем индекс элемента с минимальным расстоянием

}

}

return min\_index;

}

Функция реализующая алгоритм Дейкстры с модификацией для учёта станций пересадок:

struct s2\* dijkstra(int graph[V][V], int start, int end, struct s1\* info, struct s2\* infoWay, int\* num)

{

int dist[V]; // массив расстояний от стартовой вершины

int visited[V]; // массив посещенных вершин

int prev[V]; // массив предыдущих вершин на пути

int number = -1;

int numb = 0;

for (int i = 0; i < V; i++) // цикл для инициализации переменных для каждого города

{

dist[i] = INT\_MAX; // инициализация всех расстояний как "бесконечность"

visited[i] = 0; // отметка всех вершин как не посещённых

prev[i] = -1;

}

dist[start] = 0;

for (int i = 0; i < V - 1; i++) // поиск кратчайшего пути для всех вершин графа

{

int u = min\_distance(dist, visited); // поиск вершины с наименьшей стоимостью среди вершин, ещё не включённых в кратчайший путь

visited[u] = 1; // отметка выбранной вершины как посещённой

for (int v = 0; v < V; v++) // обновление расстояний до смежных вершин, учитывая станции пересадок

{

if (visited[v] == 0 && graph[u][v] && dist[u] != INT\_MAX && dist[u] + graph[u][v] < dist[v]) // Если вершина ещё не включена в кратчайший путь, есть ребро из выбранной вершины до смежной вершины, и расстояние от источника до выбранной вершины плюс стоимость ребра меньше текущего расстояния до смежной вершины,

{

dist[v] = dist[u] + graph[u][v]; // то обновляем расстояние до смежной вершины

prev[v] = u;

}

}

}

if (dist[end] == INT\_MAX) // если расстояние до конечного города поездки равно "бесконечности", то путь не найден

{

printf("No path found.\n");

exit(-1);

}

int current = end; // текущее положение начинается с конечного города поездки

int index = 0;

int IndexTransfer; // индекс города, где будет осуществленна ближайшая пересадка

int nap; // направление

int ind = prev[current]; // сохранение последовательности предыдущих вершин на пути

int flag = 0; // город не является пересадкой

if (info[start].Line == 8 || info[start].Line == 7 || info[end].Line == 8 || info[end].Line == 7) // если начальный или конечный город является городом-пересадкой,то

flag = 1; // обозначаем это

while (prev[current] != -1) // пока не пройдём всю последовательность предыдущих вершин на маршруте

{

if (info[current].Line != info[prev[current]].Line) // если линии разные, то

number = number + 1; // +1 к кол-ву пересадок на маршруте

current = prev[current]; // становимся на предыдущиую вершину на маршруте

}

numb = number; // сохраняем кол-во пересадок на маршруте

prev[current] = ind; // возвращаем последовательнось предыдущих вершин на пути

if (number > 0) // если пересадки есть, то

{

while (prev[current] != -1) // снова проходим по всей последовательности предыдущих вершин на маршруте

{

if (info[current].Line != info[prev[current]].Line) // если линии разные, то

{

strcpy(infoWay[0].Peresadki[numb - 1], info[prev[current]].City); // сохраняем название города, как город-пересадку

numb = numb - 1; // уменьшаем кол-во пресадок

if (numb == 0) // если кол-во пересадок == 0, то заканчиваем цикл

break;

}

current = prev[current]; // становимся на предыдущую вершину на маршруте

}

}

printf("\n");

IndexTransfer = find\_city\_index(info, infoWay[0].Peresadki[0]); // получаем индекс ближайшей пересадки

nap = napravl(info, start, IndexTransfer, end, number); // определяем напрваление (если nap == 1, то поездке от центра, если == 0, то к центру)

infoWay = GetTime(info, infoWay, start, end, nap, number); // считываем нужные времена отбытия и прибытия

infoWay = GetTimeOfPeresadki(info, infoWay, nap, number); // считываем нужные времена прибытия и отбытия со станций пересадок

infoWay[0].Distance = dist[end]; // сохраняем расстояние между начальным и конечным городами

\*num = number; // сохраняем кол-во пересадок

return infoWay;

}

Функция получения индекса города:

int find\_city\_index(struct s1\* info, char\* name) // функция для получения индекса города

{

for (int i = 0; i < V; i++) // цикл для прохождения по всем городам массива структур

{

if (strcmp(info[i].City, name) == 0) // если есть совпадение, то возвращаем i, то есть индекс данного города в массиве структур

return i;

}

return -1; // иначе возвращаем -1

}

Функция определения направления поездки:

int napravl(struct s1\* info, int index1, int index2, int end, int num) // определение напрваления поездки ( от центра или к центру) для чтения необходимых масивов времён

{

if (num == 0) // если станций пересадок нет, то...

{

int dist1 = info[index1].Dist; // первое расстояние принимаем равным расстоянию города, из которого едет пользователь

int dist2 = info[end].Dist; // второе расстояни принимаем равным расстоянию города, в который едет пользователь

int napravl; // переменная напрваления

if ((dist1 - dist2) < 0) // если разность данных расстояний отрицательная, то пользователь двитжется от центра

napravl = 1; // пользователь движется от центра

else

napravl = 0; // пользователь движется к центру

return napravl;

}

else // если станции пересадок есть, то...

{

int dist1 = info[index1].Dist; // первое расстояние принимаем равным расстоянию города, из которого едет пользователь

int dist2 = info[index2].Dist; // второе расстояни принимаем равным расстоянию города, в котором осуществляется ближайшая пересадка с начала поездки

int napravl;

if ((dist1 - dist2) < 0)

napravl = 1;

else

napravl = 0;

return napravl;

}

}

Функция сохранения нужных времён отправления из начального города и прибытия в конечный город:

struct s2\* GetTime(struct s1\* info, struct s2\* infoWay, int indexFrom, int indexTo, int napravl, int num) // функция сохранения нужных времён прибытия и отбытия

{

if (napravl == 0) // ЕСЛИ поездке к центру, то

{

if (num == 0) // если кол-во пересадок == 0, то

{

for (int i = 0; i < t; i++) // цикл для чтения массива времён

{

strcpy(infoWay[0].TimeFrom[i], info[indexFrom].Time2[i]); // считываем время отбытия из начального города

strcpy(infoWay[0].TimeTo[i], info[indexTo].Time2[i]); // считываем время прибытия в конечный город

}

}

else // иначе

{

for (int i = 0; i < t; i++) // цикл для чтения массива времён

{

strcpy(infoWay[0].TimeFrom[i], info[indexFrom].Time2[i]); // считываем время отбытия из начального города

strcpy(infoWay[0].TimeTo[i], info[indexTo].Time1[i]); // считываем время прибытия в конечный город учитывая пересдку

}

}

}

else // ИНАЧЕ (поездка от центра), то

{

if (num == 0) // если кол-во пересадок == 0, то

{

for (int i = 0; i < t; i++) // цикл для чтения массива времён

{

strcpy(infoWay[0].TimeFrom[i], info[indexFrom].Time1[i]); // считываем время отбытия из начального города

strcpy(infoWay[0].TimeTo[i], info[indexTo].Time1[i]); // считываем время прибытия в конечный город

}

}

else // иначе

{

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

{

strcpy(infoWay[0].TimeFrom[i], info[indexFrom].Time1[i]); // считываем время отбытия из начального города учитывая пересадку

strcpy(infoWay[0].TimeTo[i], info[indexTo].Time2[i]); // считываем время прибытия в конечный город

}

}

}

return infoWay;

}

Функция сохранения нужных времён прибытия в город-пересадку и отправления из города-пересадки:

struct s2\* GetTimeOfPeresadki(struct s1\* info, struct s2\* infoWay, int napravl, int number) // функция сохранения нужных времён прибытия и отбытия станций пересадок

{

if (napravl == 0) // ЕСЛИ поездке к центру, то

{

for (int j = 0; j < number; j++) // цикл для прохождения по всем городам-пересадкам

for (int i = 0; i < t; i++) // цикл для чтения массива времён

{

if ((j == 1) && strcmp(info[0].City, infoWay[0].Peresadki[j]) == 0) // если второй пересадкой является Минск, то

{

strcpy(infoWay[0].TimePeres1[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]); // считываем время прибытия в город-пересадку

strcpy(infoWay[0].TimePeres2[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time1[i]); // считываем время отбытия из города-пересадки

}

else if (j == 1)

{

strcpy(infoWay[0].TimePeres1[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time1[i]); // считываем нужное время прибытия в город-пересадку

strcpy(infoWay[0].TimePeres2[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time1[i]); // считываем нужное время отбытия из города-пересадки

}

else if ((j == 0) && strcmp(info[8].City, infoWay[0].Peresadki[j]) == 0) // если первой пересадкой является город Орша, то

{

strcpy(infoWay[0].TimePeres1[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]); // считываем время прибытия в город-пересадку

strcpy(infoWay[0].TimePeres2[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]); // считываем время отбытия из города-пересадки

}

else

{

strcpy(infoWay[0].TimePeres1[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]); // считываем время прибытия в город-пересадку

strcpy(infoWay[0].TimePeres2[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time1[i]); // считываем время отбытия из города-пересадки

}

}

}

else // ИНАЧЕ (поездка от центра), то

{

for (int j = 0; j < number; j++)

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

{

if ((j == 1) && strcmp(info[0].City, infoWay[0].Peresadki[j]) == 0)

{

strcpy(infoWay[0].TimePeres1[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time1[i]);

strcpy(infoWay[0].TimePeres2[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time1[i]);

}

else if (j == 1)

{

strcpy(infoWay[0].TimePeres1[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]);

strcpy(infoWay[0].TimePeres2[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]);

}

else if ((j == 0) && strcmp(info[8].City, infoWay[0].Peresadki[j]) == 0)

{

strcpy(infoWay[0].TimePeres1[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]);

strcpy(infoWay[0].TimePeres2[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]);

}

else

{

strcpy(infoWay[0].TimePeres1[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time1[i]);

strcpy(infoWay[0].TimePeres2[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]);

}

}

}

return infoWay;

}

1. **РЕЗУЛЬТАТЫ**

Допустим пользователь вёл, что собирается поехать из Бреста в Смолевичи.

На рисунках 6.1-6.8 представлены пример работы программы по исходным данным.

Меню пользователя

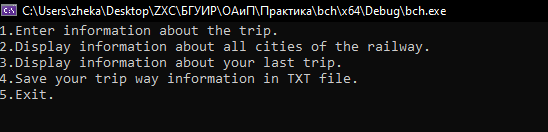


Рис. 6.1 – Результат работы функции InitialMenu() в консоли

Вывод информации о всех городах

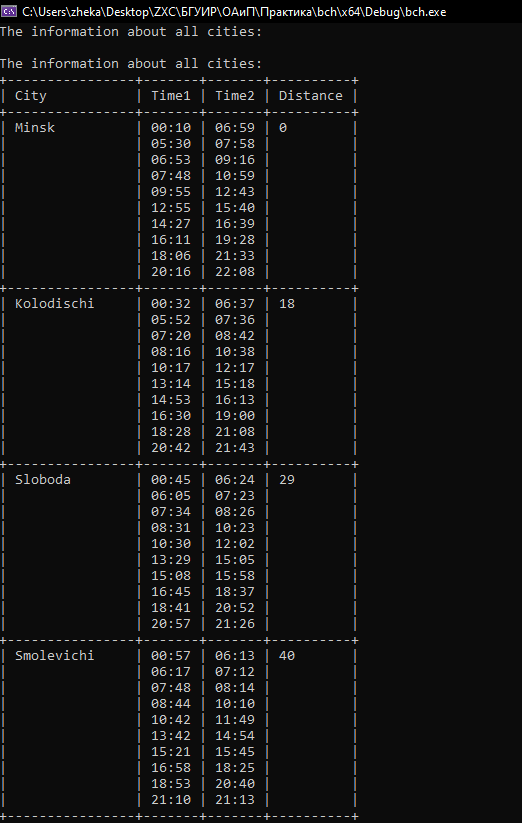


Рис. 6.2 – Результат работы функции PrintStruct\_s1() в консоли

Пример хранения информации о всех городах в текстовом файле

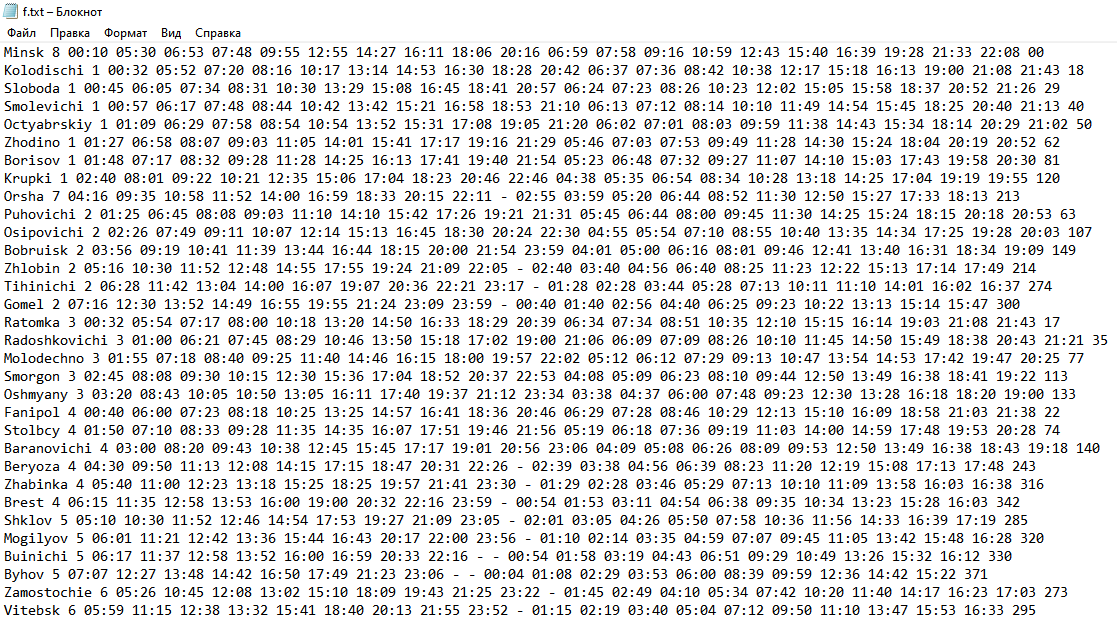


Рис. 6.3 – Файл с информацией о всех городах (файл f.txt)

Ввод названий городов

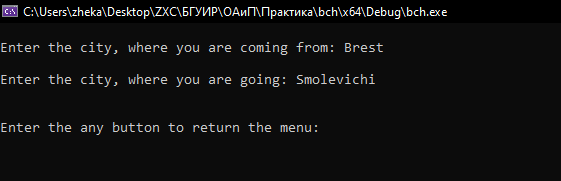


Рис. 6.4 – Результат работы функций InputCityTo() и InputCityFrom() в консоли

Вывод информации о поездке пользователя

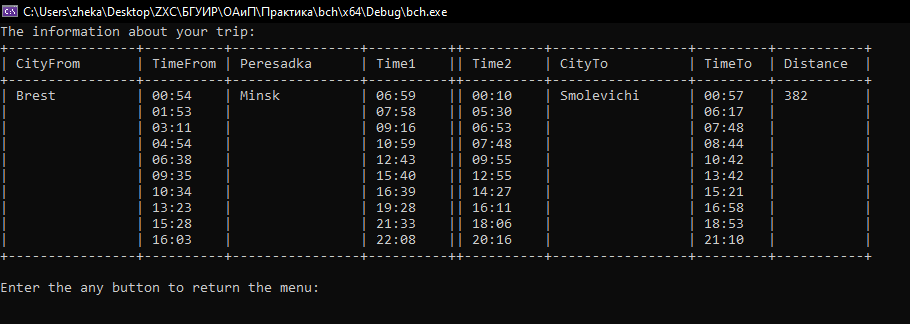


Рис. 6.5 – Результат работы функции PrintStruct\_s2() в консоли

Сохранение информации о поездке пользователя в текстовый файл

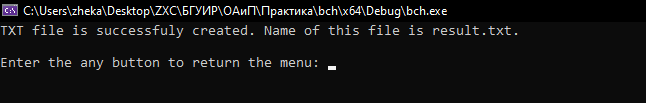


Рис. 6.6 – Результат работы функции InputInformationInFileS2() в консоли

Пример хранения информации о поездке пользователя в текстовом файле

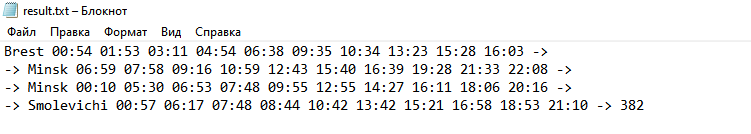


Рис. 6.7 – Результат работы функции InputInformationInFileS2() в текстовом файле result.txt

Конец выполнения программы

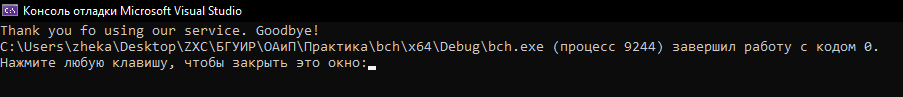


Рис. 6.8 – Результат работы функции InitialMenu() в консоли

**ЗАКЛЮЧЕНИЕ**

В результате выполнения учебной практики были выполнены первоначально заданные цели, закреплены теоретические и практические знания в использовании структур данных, работы с файлами и т.д.

На сегодняшний день существует огромное количество баз данных, которые хранят и анализируют введённую информацию, а также позволяют пользователям взаимодействовать с ней. Данная программа предназначена для таких же целей. Пример приложения с такими же задачами: БЧ (беларуская чыгунка).

Дальнейшее развитие программы может включать в себя:

1. Расширение функционала программы, включая возможность расчёта и вывода стоимости поездки, бронирования билетов, выбор определённого вагона и места и т.д.
2. Добавление большего количества железнодорожных линий и разных видов поездов (линии эконом-класс, линии бизнес-класс, региональные линии, городские линии и т.д.).
3. Графический интерфейс.

Для успешного запуска данного программного средства и работы с ним требуются следующие системные требования:

* OC Windows 10
* Среда разработки Microsoft Visual Studio 2022
* 64 – разрядная архитектура (x64)

**СПИСОК ЛИТЕРАТУРЫ**

Луцик Ю.А., Ковальчук А.М., Сасин Е.А. – Учебное пособие по курсу «Основы алгоритмизации и программирования: язык С». – Минск: БГУИР, 2015 г.

Луцик Ю.А., Ковальчук А.М., Лукьянова И.В., Бушкевич А.В. – Лабораторный практикум по курсу «Основы алгоритмизации и программирования» (в 2-х частях). – Минск: БГУИР, 2008 г.

**ПРИЛОЖЕНИЕ А**

(обязательное)