МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА» (НГТУ)

Учебно-научный институт радиоэлектроники и информационных технологий

Кафедра «Информационные радиосистемы»

ОТЧЕТ
по прохождению учебной практики

Направление подготовки/специальность: 11.03.01 Радиотехника

Образовательная программа: Радиоэлектронные системы

Выполнил:

Студент гр. 22-Рз _____ Наумов А.А.
(подпись практиканта)

Руководитель практики от кафедры
ассистент _____ Шабалина К.С.
(ученые звание и степень) *(подпись)*

Отчет защищен с оценкой: _____

Дата защиты «___» _____2024 г.

Нижний Новгород
2024

Содержание

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА УЧЕБНУЮ, ОЗНАКОМИТЕЛЬНУЮ

(вид, тип практики)

ПРАКТИКУ

Студента гр. 22-Рз Наумова Александра Андреевича

Направление подготовки/специальность: 11.03.01 Радиотехника
код и наименование направления подготовки
Образовательная программа Радиоэлектронные системы

Место прохождения практики НГТУ им. Р.Е. Алексеева, ИРИТ, каф. ИРС
(название предприятия или лаборатории, подразделения вуза)

Время прохождения практики
Дата начала практики « 29 » июня 2024 г.
Дата окончания практики « 12 » июля 2024 г.

Тема индивидуального задания:
Вывести на экран информацию о маршруте, номер которого введен с клавиатуры. Если таких маршрутов нет, вывести на экран соответствующее сообщение. Элементы структуры: название начального пункта маршрута, название конечного пункта маршрута, номер маршрута.

Содержание практики

Во время прохождения практики студент обязан:

Ознакомиться: компилятор OnlineGDB; основы программирования на языке С.
Изучить: структуры данных, функции, указатели, массивы в языке С; модульное программирование; создание блок-схем.
Выполнить следующие виды работ по приобретению практических навыков:
- сбор теоретического материала о структурах данных, функциях, указателях, массивах в языке С;
- разработка блок-схемы программы определения суммы всех значений координат *x* и *y* по отдельности с использованием указателя;
- написание программного кода с подробными комментариями каждой строки;
- компиляция программы;
- получения результата работы программы;
- описание всех используемых в программе функций.

Собрать материал по теме индивидуального задания для подготовки отчета по практике.

Должность на практике практикант.
(практикант, стажер, помощник, конкретная должность)

Планируемые результаты обучения при прохождении практики

Планируемые результаты освоения образовательной программы	Планируемые результаты обучения при прохождении практики		
Код компетенции	Знать	Уметь	Владеть
УК-3	- идеи других членов команды для достижения поставленной цели	- осуществлять обмен информацией, знаниями и опытом с членами команды, а также оценивать идеи других членов команды для достижения поставленной цели	- способностью самообучения; - способностью поиска необходимой информации; - умением нести ответственность за результат своей работы
ОПК - 3	- способы адаптации программных решений для представления, хранения и обработки информации	- адаптироваться к новым программам и ресурсам в соответствии с изменяющимися требованиями	- инструментальными средствами анализа и доработки программных решений

Результаты освоения обучающимся компетенций при прохождении практики оцениваются по итогам защиты отчета по прохождению практики, с учетом выполнения индивидуального задания и отзыва (характеристики) о прохождении практики на предприятии.

Руководитель практики от кафедры

ассистент _____ Шабалина К.С.

(ученые звание и степень) *(подпись)*

Задание на практику получил:

Студент _____ Наумов А.А.
(подпись) (ФИО)

«___» _____ 2024 г.

СОВМЕСТНЫЙ РАБОЧИЙ ГРАФИК (ПЛАН) ПРОВЕДЕНИЯ УЧЕБНОЙ, ОЗНАКОМИТЕЛЬНОЙ ПРАКТИКИ

(вид, тип практики)

Студента гр. 22-Рз Наумова Александра Андреевича

№ п/п	Разделы (этапы) практики		Сроки выполнения с 29 июня по 12 июля	Отметка о выполнении (подпись руководителя практики*)
1.	**Подготовительный (организационный) этап**			
1.1.	Проведение собрания студентов; получение индивидуального задания и путёвки на практику		**29 июня**	
2.	**Учебный этап**			
2.1.	Выполнение индивидуального задания:			
	1. Поиск информации о структурах и способах их определения в языке С		**30 июня**	
	2. Составление структуры программы		**1 июля**	
	3. Написание программы, отладка		**2-4 июля**	
3.	**Заключительный этап**			
3.1	Анализ и обобщение полученной информации		**5 июля**	
3.2	Написание отчёта по практике		**6 июля**	
	1. Создание и оформление блок-схем		**6 июля**	
	2. Написание пунктов «Введение», «Структурная реализация»		**7 июля**	
	3. Написание пунктов «Программная реализация», «Результат компиляции программного кода», «Результат выполнения программного кода», «Заключение»		**8 июля**	

Руководитель практики от кафедры

ассистент _____ Шабалина К.С.

(ученые звание и степень) *(подпись)*

Введение

В предложенном отчёте будет рассмотрена следующая задача: «Вывести на экран информацию о маршруте, номер которого введён с клавиатуры. Если таких маршрутов нет, вывести на экран соответствующее сообщение. Элементы структуры: название начального пункта маршрута, название конечного пункта маршрута, номер маршрута.»

В качестве гораздо более эффективной замены предложенному онлайн компилятору «OnlineGDB», была использована одна из наиболее актуальных и применяемых, в промышленном масштабе, интегрированная среда разработки (IDE) для языка C/C++ от JetBrains, CLion 2024.1.3. В качестве плюсов выбранной IDE, можно выделить встроенный режим отладки (debugger), пошаговый режим отладки (step over, step into, step out), просмотр памяти, используемой программой, контекстный поиск и подсказки.

Во время написания программой реализации приведённой задачи были использованы следующие понятия из языка C: циклы, условия, функции, структуры, указатели, функции для работы с памятью (malloc, free). А также различные заголовочные файлы, такие как: stdio.h, stdbool.h, stdlib.h, string.h, time.h.

По рекомендациям к выполнению было предложено вводить данные структур с клавиатуры. Для придания программе более реалистичного стиля работы, было решено вводить с клавиатуры только количество маршрутов. Наборы маршрутов генерируются, используя данные из двух текстовых файлов, которые находятся рядом с исполняемым файлом программы - «array_start_locations.txt» и «array_end_locations.txt». Программа считывает все строки, которые предлагаются как названия начальных и конечных точек, и в случайном порядке собирает маршруты по количеству введённым пользователем. После генерации маршрутов, пользователь вводит номер маршрута, о котором он хочет получить информацию. Программа ищет запрошенный маршрут в массиве, и выдаёт информацию о нем на стандартное устройствовывода. После этого пользователю предлагается выбор: повторить поиск или закрыть программу.

1. Структурная реализация

В качестве переменных было принято решение о использовании двух структур, одна с основным массивом данных, из которого будет производиться поиск, и вторую, для других, необходимых в работе приложения, переменных. Рассмотрим их по очереди.

```
typedef struct array_template{
   char starting_point[NAME_MAX];
   char ending_point[NAME_MAX];
   int code;
}array_template;
```

В качестве количества символов в строке было выбрано значение 32, оно автоматически подставляется вместо макроса NAME_MAX, из-за использования #define в начале файла.

Как мы видим, структура array_template состоит из трёх полей, starting_point и ending_point, это массивы типа char, по количеству 32. И целочисленное code, которое является номером маршрута.

Вторая используемая структура имеет название - Application:

```
typedef struct Application{
   int length_of_an_array;
}Application;
```

Состоит из одного целочисленного поля — length_of_an_array, которое хранит количество маршрутов, запрошенное пользователем.

Для разграничения программы по модулям, функции запроса данных, поиска маршрута, и вывода информации были отделены от функций работы с файлами. Таким образом программа состоит из пяти основных файлов, и двух текстовых.

«main.cpp»

«application.h»

«application.cpp»

«handler.h»

«handler.cpp»

«array_start_locations.txt»

«array_end_locations.txt»

Рассмотрим прототипы функций их файла «application.h»:

int app_run(void* raw_app); - функция из «main.cpp», содержащая основной исполняемый код.

int app_get_length_of_an_array(void* raw_app); - функция запрашивает у пользователя количество маршрутов, сохраняет введённое значение в переменную length_of_an_array.

void app_do_output(void* raw_app, void* raw_array); - функция отвечает за поиск введённого пользователем маршрута в массиве. При нахождении выводит его в стандартное устройство вывода, при неудаче, сообщение об ошибке.

bool app_rerun(); - функция «спрашивает» у пользователя повторить поиск, для нового маршрута, или нет.

Перейдём к файлу «handler.h».

int handler_get_size_of_an_array(void* file_name); - функция для получения количества строк в файле, имя которого было передано функции в качестве аргумента.

char** handler_init_file (void* file_name); - функция для считывания строк из файла, имя которого было передано в качестве аргумента, возвращает указатель на массив считанных данных.

void* handler_init_array(int to_create); - функция для сборки массива маршрутов, возвращает указатель на этот массив.

1.2 Программная реализация

Как уже было упомянуто, программа разделена на два модуля, handler и application.

Рассмотрим их по очереди.

```
#include "handler.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
```

Модуль начинается добавлением необходимых заголовочных файлов.

#define HOLDER_MAX 32 — директива позволяет использовать имя HOLDER_MAX самого числа 32, что позволит изменить количество символов в стоке без лишних усилий.

int handler_get_size_of_an_array(void* file_name){ — начинается описание функции, в качестве аргумента принимается указатель типа void, на имя файла, с которым предстоит работать.

FILE *file = fopen(file_name, "r"); - открываем файл в режиме чтения, записываем указатель на него в памяти в переменную типа FILE – file.

if(file != NULL) { - если файл открыт, то начнётся работа функции.

int final_line_count = 0; - вводим переменную для подсчёта количества строк.

char* retrieved_line; - вводим указатель типа char.

char* tempStr = malloc(HOLDER_MAX * sizeof (char)); - вводим указатель типа char, который будет использоваться в качестве буффера для последующей функции. Выделяем ему память, равную 32-м элементам типа char.

do{ – начинаем цикл считывания строк.

retrieved_line = fgets(tempStr,HOLDER_MAX, file); - считываем строку функцией fgets, в качестве буфера указываем tempStr, кол-во элементов — 32, и имя файла из которого считывать информацию.

if(retrieved_line != NULL){

++final_line_count;

} - если считанная строка не равна NULL т. е. не пустая — добавим 1 к счётчику строк.

} while (retrieved_line != NULL); - цикл повторяется пока не считает все строки из файла т. е. очередная строка не окажется NULL.

free(tempStr); - освобождаем память, выделенную под указатель.

fclose(file); - закрываем файл с которым работали.

return final_line_count; - возвращаем число — количество строк в запрошенном файле.

else{

printf("ER:011 - Error while opening text file.\n");

return 0;

} - Если файл, по какой-то причине, не открыт, то выведется сообщение об ошибке, и вернётся 0.

char** handler_init_file (void* file_name){ - функция для считывания строк файла в массив, в качестве аргумента принимает имя файла, строки которого необходимо считать.

FILE *file = fopen(file_name, "r"); - открываем файл в режиме чтения, записываем указатель на него в памяти в переменную типа FILE – file.

if(file != NULL){ - если файл открыт, то начнётся работа функции.

int line_count = handler_get_size_of_an_array(file_name); - записываем количество строк в файле, посредством вызова функции handler_get_size_of_an_array, передавая имя необходимого файла.

char** resulting_array = malloc(sizeof (char*)*line_count); - объявляем переменную массива маршрутов, выделяем для него память.

for(int i = 0; i < line_count; ++i){

resulting_array[i] = malloc(HOLDER_MAX*sizeof (char));

fgets(resulting_array[i], HOLDER_MAX, file);

} - считываем все строки из файла, выделяем память для каждой, записываем её в выделенное место.

fclose(file); - закрываем файл.

for (int i = 0; i < line_count; ++i) {

size_t size = strlen(resulting_array[i]);

if (size > 0 && resulting_array[i][strlen(resulting_array[i])- 1] == '\n') {

resulting_array[i][strlen(resulting_array[i])- 1] = '\000';

}

} - идём по всем считанным строкам, если её последний символ — знак новой строки “\n”, то удаляем его.

return resulting_array; - возвращаем указатель на массив точек назначения.

else{

printf("ER:011 - Error while opening text file.\n");

return NULL;

} - Если файл не удалось открыть, то напишем сообщение об ошибке, вернём NULL.

void* handler_init_array(int to_create){ - функция формирования массива маршрутов, аргумент — кол-во маршрутов.

char *start_end_files[] = { NULL,

"array_start_locations.txt",

"array_end_locations.txt"}; -имена файлов, которые участвуют в формировании массива маршрутов.

char** starting_locations = handler_init_file(start_end_files[1]); - считываем все строки из первого файла, помещаем указатель на массив этих строк в переменную starting_locations.

char** ending_locations = handler_init_file(start_end_files[2]); - считываем все строки из второго файла, помещаем указатель на массив этих строк в переменную starting_locations.

array_template* main_array = malloc(sizeof (array_template)*to_create); - объявляем указатель на переменную массива маршрутов.

srand(time(0)); - задаём правило для генерации случайных чисел через команду rand().

int start_lines = handler_get_size_of_an_array(start_end_files[1]); - записываем кол-во строк в файле начальных точек в переменную.

int end_lines = handler_get_size_of_an_array(start_end_files[2]); - записываем кол-во строк в файле конечных точек в переменную.

```
for(int i = 0; i < to_create; ++i){
   strcpy(main_array[i].starting_point, starting_locations[rand()%
start_lines]);
   strcpy(main_array[i].ending_point, ending_locations[rand()%
end_lines]);
  main_array[i].code = rand()%100;
   if(i == 0){
    printf("Available routes: ");
   }
   printf("%d ", main_array[i].code);
```

} - в цикле, присваиваем каждому маршруту начальную точку, конечную точку, номер, случайным образом. Так же, выводим все доступные маршруты на стандартное устройство вывода.

free(starting_locations); - освобождаем память для массива начальных точек.

free(ending_locations); - освобождаем память для массива конечных точек.

return main_array; - возвращаем указатель на массив маршрутов

Теперь перейдём к модулю application.

#include "application.h" — добавляем заголовочный файл с прототипами функций и типом данных.

int app_run(void* raw_app){ - основная исполняющая функция, в качестве аргумента принимает указатель на структуру.

Application* app = (Application*) raw_app; - приводим указатель типа void к указателю типа Application.

app_get_length_of_an_array(app); - считываем кол-во элементов массива маршрутов, в аргументе передаём указатель на структуру.

int *p_array = handler_init_array(app→length_of_an_array); - собираем массив маршрутов, записываем указатель на него.

```
do {
   app_do_output(app, p_array);
}while(app_rerun()); - вызываем функцию поиска маршрута, передаём
```

два указателя, на структуру app, и массив маршрутов p_array. После одного поиска, вызовется функция app_rerun(), которая спросит у пользователя, провести поиск ещё раз, или нет.

free(p_array); - выгружаем из памяти массив маршрутов.

return 0; - возвращаем 0, программа выполнилась успешно.

void app_do_output(void* raw_app, void* raw_array){ - функция поиска маршрута. Если запрошенный маршрут найден, то информация о нем на стандартное устройство вывода.

Application* app = (Application*) raw_app;

array_template* array = (array_template *) raw_array; - приводим указатели типа void из аргументов, к указателям правильного типа данных.

int to_find; - вводим переменную для номера маршрута, который нужно найти.

```
printf("Which route you would like to know about?\n");
scanf_s("%d", &to_find);
```

fflush(stdin); - Выводим сообщение, считываем число в переменную, чистим буфер.

```
for(in i = 0; i < app->length_of_an_array; ++i){
  if(array[i].code == to_find){
    printf("\nRoute was found!\n# - %d\nStarting at - %s\nEnding at - %s\n\n", array[i].code, array[i].starting_point, array[i].ending_point);
      break;
   }
   if(i == (app->length_of_an_array)-1){
      printf("No appropriate route was found.\n");
   }
```

} - цикл, идём по каждому элементу массива маршрутов, если запрошенный пользователем маршрут найден, то информация о нем выводится на стандартное устройство вывода. Если нет, то выводится сообщение об ошибке.

```
int app_get_length_of_an_array(void* raw_app){
   Application* app = (Application*) raw_app;
   printf("Enter total number of routes:\n");
  scanf_s("%d", &(app->length_of_an_array));
   fflush(stdin);
   return 0;
```

} - функция используется для запроса количества маршрутов у пользователя, в качестве аргумента принимает указатель типа void на структуру app, приводим его к нужному типу, считываем, записываем. Как было описано ранее.

```
bool app_rerun(){
  int temp_read;
  printf("Repeat? (1 = yes, else = no)\n");
 scanf_s("%d", &temp_read);
```

```
fflush(stdin);
 if(temp_read == 1){
  return true;
}
  else{
    return false;
  }
```

} - функция для повторения процедуры поиска, если пользователь ответит да — 1, то поиск повторится ещё раз, если нет, то программа завершится.

Описанную программу можно представить в виде блок схемы, изображённой на рисунке 1.

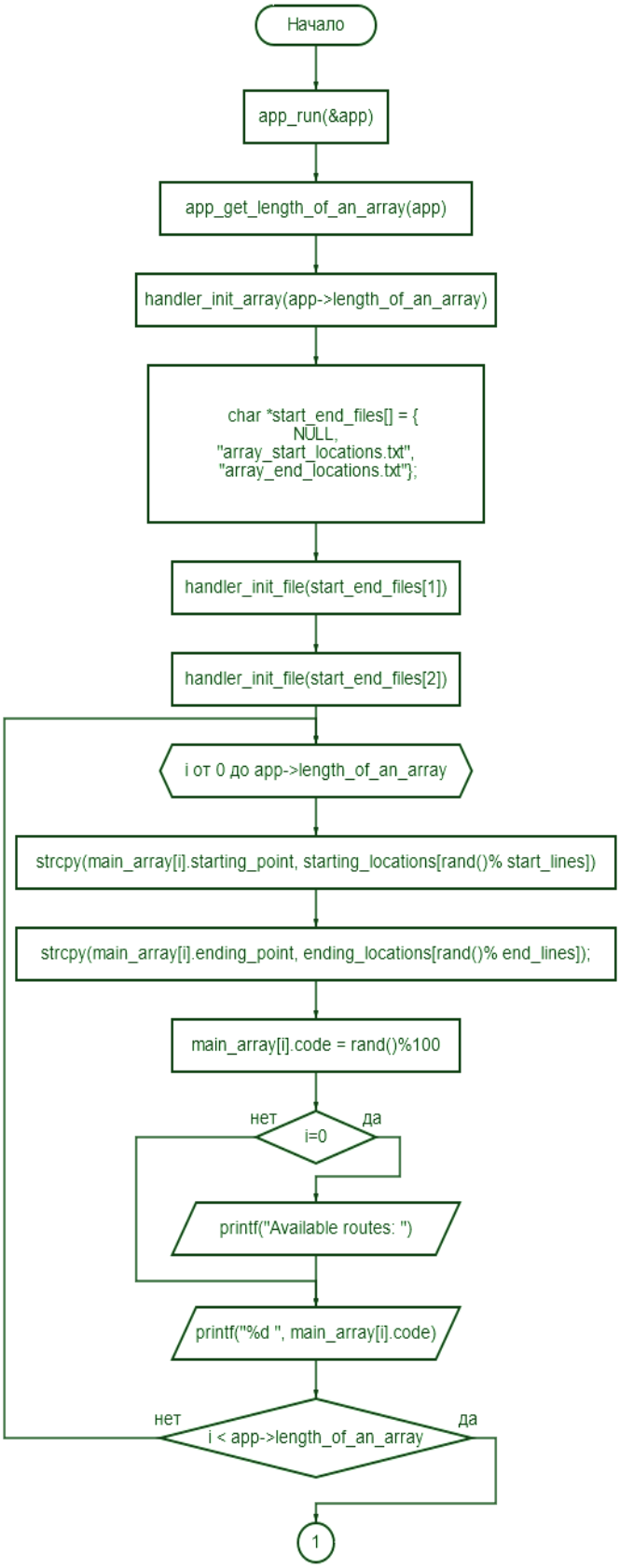
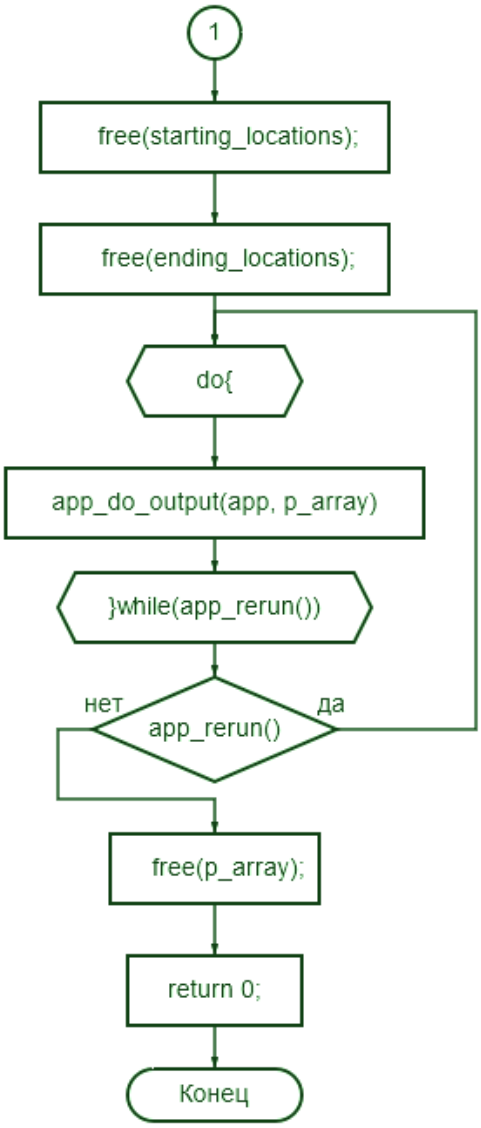


Рисунок 1 — Блок схема программы

Продолжение рисунка 1.



1.3 Результат компиляции программного кода

Результат компиляции программного кода представлен на рисунке 2.

```
====================[ Build | 4_SEM_PRAC_1_MY | Debug ]========================
C:\Users\Anatejl\AppData\Local\JetBrains\Toolbox\apps\CLion\ch-0\241.17890.19\bin\cmake\win\x64\bin\cmake.exe --build
[ 25%] Building C object CMakeFiles/4_SEM_PRAC_1_MY.dir/4_Practice_1/My/main.c.obj
[ 50%] Building C object CMakeFiles/4_SEM_PRAC_1_MY.dir/4_Practice_1/My/application.c.obj
[ 75%] Building C object CMakeFiles/4_SEM_PRAC_1_MY.dir/4_Practice_1/My/handler.c.obj
[100%] Linking C executable 4_SEM_PRAC_1_MY.exe
[100%] Built target 4_SEM_PRAC_1_MY

Build finished
```

Рисунок 2 — Результат компиляции программного кода

Сборка программы проводилась с применением общепризнанной системы сборки CMake.

Состав файла cmakelists.txt:

```
cmake_minimum_required(VERSION 3.23)
project(NNTU)
set(CMAKE_C_STANDARD 99)
add_executable(4_SEM_PRAC_1_MY
4_Practice_1/My/main.c
4_Practice_1/My/application.c
4_Practice_1/My/application.h
4_Practice_1/My/handler.c
4_Practice_1/My/handler.h)
```

1.4 Результат выполнения программного кода

Результат выполнения программного кода представлен на рисунке 3.

```
E:\Git\NNTU_INF\cmake-build-debug\4_SEM_PRAC_1_MY.exe
Enter total number of routes:
20
Available routes: 39 83 98 49 89 36 11 57 23 61 43 37 20 6 20 93 50 26 49 72

Which route you would like to know about?
39

Route was found!
# - 39
Starting at - Moonshadow
Ending at - Jadeport

Repeat? (1 = yes, else = no)
```

Рисунок 3 — Результат выполнения программного кода

Как было описано в пункте 1.2, после выдачи результата по маршруту, пользователю задаётся вопрос, повторить ли поиск. Продолжение представлено на рисунке 4.

```
Repeat? (1 = yes, else = no)
1
Which route you would like to know about?
72

Route was found!
# - 72
Starting at - Moonglade
Ending at - Ultharian

Repeat? (1 = yes, else = no)
4

Process finished with exit code 0
```

Рисунок 4 — Продолжение выполнения программного кода

Заключение

1. В ходе выполнения задания по учебной практике, я получил навыки составления программ, для которых используются внешние файлы. Научился обрабатывать их, считывать строки, считать считанные строки. Обновил знания по работе с памятью. По передаче структур по указателю, обращение к полям структуры, представленной указателем (→).

В качестве результата выполнения практического задания получилась программная реализация, работающая с внешними текстовыми файлами, что позволяет, при должной подготовке, использовать программу в качестве модуля, в другой системе, ведь для работы, необходимо соблюсти всего несколько условий, а именно:

1.2. Наличие файлов с определенным именем “array_start_locations.txt” и "array_end_locations.txt".

1.3. ввести число для генерации массива. Если потребует заказчик, то можно перестроить программный код, чтобы получать это число через аргумент вызова самой программы. Полный программный код, с комментариями, представлен в приложении.

2. В результате прохождения ознакомительной, учебной практики были

(наименование практики)

приобретены следующие практические навыки и умения:

Работа с текстовыми файлами, работа с void указателями, передача структур данных через void указатели, приведение типа переменной, модульный подход к организации структуры программы.

Список использованных источников

1. Brian Kernighan, The C Programming Language
2. Stephen Kochan, Programming in C (4th Edition)
3. Jens Gustedt, Modern C
4. Прата Стивен, Язык программирования C

Приложение 1.

main.c

```
#include "application.h"

int main(int argc, char *argv[]){

   //declare a variable of type Application and name in to app
   Application app;

   //call main run function and pass address of that app variable to it
   int ret = app_run(&app);

   return ret;
}
```

application.h

```
#ifndef NNTU_APPLICATION_H
#define NNTU_APPLICATION_H

#include "handler.h"
#include <stdlib.h>
#include <stdbool.h>
#include <stdio.h>

typedef struct Application{

   int length_of_an_array;

}Application;

int app_run(void* raw_app);
int app_get_length_of_an_array(void* raw_app);
void app_do_output(void* raw_app, void* raw_array);
bool app_rerun();

#endif //NNTU_APPLICATION_H
```

```c
#include "application.h"

int app_run(void* raw_app){

   //cast an address received from main function, to a pointer to Application type, name it app
   Application* app = (Application*) raw_app;

   //prompt user to enter a number of routes to generate
   app_get_length_of_an_array(app);

   //get a pointer to main array of routes with starting and ending locations
   //as an argument, give it number entered by user in previous line
   int *p_array = handler_init_array(app->length_of_an_array);

   //do output cycle once, then ask user whether he wants to repeat
   do {

      //call output function that will ask a user of a route he wants to gen info on,
      //find it in array then sequentially display its contents. Starting and ending locations
      //and route number.
      //if happens that route wasn't found, display error message stating that.
      app_do_output(app, p_array);

   }while(app_rerun()); //ask user whether he wants to repeat search

   //deallocate main array
   free(p_array);

   return 0;
}

void app_do_output(void* raw_app, void* raw_array){

   //cast a void pointer of an app address to an Application type pointer named app
   Application* app = (Application*) raw_app;
```

```
    //same to main array pointer
    array_template* array = (array_template *) raw_array;

    //declare number to find in an array
    int to_find;

    //ask user to enter that number
    printf("Which route you would like to know about?\n");

    //retrieve that number from input
    scanf_s("%d", &to_find);

    //clear input buffer
    fflush(stdin);

    //go through each entry in the main array
    for(int i = 0; i < app->length_of_an_array; ++i){
        // if route number is equal to one in the array's entry, -> display it and
break from cycle
        if(array[i].code == to_find){
            printf("\nRoute was found!\n# - %d\nStarting at - %s\nEnding at -
%s\n\n", array[i].code, array[i].starting_point, array[i].ending_point);
            break;
        }
        //if no route was found, display clarification on that
        if(i == (app->length_of_an_array)-1){
            printf("No appropriate route was found.\n");
        }
    }
}

int app_get_length_of_an_array(void* raw_app){

    //cast a void pointer of an app address to an Application type pointer
named app
    Application* app = (Application*) raw_app;

    //ask user of total number of routes to build array upon
    printf("Enter total number of routes:\n");
    scanf_s("%d", &(app->length_of_an_array));
```

```
    fflush(stdin);

    return 0;
}

bool app_rerun(){

    //used to ask user whether he wants to repeat search cycle, if answer is 1 (i.e. - yes)
    // then search again, if else, effectively end the program
    int temp_read;
    printf("Repeat? (1 = yes, else = no)\n");
    scanf_s("%d", &temp_read);
    fflush(stdin);
    if(temp_read == 1){
        return true;
    }
    else{
        return false;
    }
}
```

handler.h

```
#ifndef NNTU_HANDLER_H
#define NNTU_HANDLER_H

#define NAME_MAX 25

typedef struct array_template{
    char starting_point[NAME_MAX];
    char ending_point[NAME_MAX];
    int code;
}array_template;

int handler_get_size_of_an_array(void* file_name);
char** handler_init_file (void* file_name);
void* handler_init_array(int to_create);


#endif //NNTU_HANDLER_H
```

```c
#include "handler.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

//Assume a line in an array won't be larger than 32 symbols
#define HOLDER_MAX 32

int handler_get_size_of_an_array(void* file_name){

   //open a file of given with an argument, hence "file_name"
   FILE *file = fopen(file_name, "r");

   if(file != NULL) {
      //Counter for lines in file
      int final_line_count = 0;

      //fully retrieved line
      char *retrieved_line;

      //temporary variable to act as a buffer for fgets()
      char *tempStr = malloc(HOLDER_MAX * sizeof(char));

      //cycle until all lines in provided file are read
      do {
         //get a line
         retrieved_line = fgets(tempStr, HOLDER_MAX, file);

         //if a line isn't NULL -> add 1 to line counter
         if (retrieved_line != NULL) {
            ++final_line_count;
         }
      } while (retrieved_line != NULL);

      //deallocate our "buffer" variable
      free(tempStr);
```

```
        //close the file which was read through
        fclose(file);

        //return number of lines in a file
        return final_line_count;
    }
    else{
        // if any error occurs with file, print an error and return NULL
        printf("ER:011 - Error while opening text file.\n");
        return 0;
    }
}

char** handler_init_file (void* file_name){

    //open a file of given with an argument, hence "file_name"
    FILE *file = fopen(file_name, "r");

    //if a file of given name exists, then do code, else -> print an error
    if(file != NULL){
        //total lines in file, given for function to process
        int line_count = handler_get_size_of_an_array(file_name);

        //Read all lines into array
        //malloc for resulting array
        char** resulting_array = malloc(sizeof (char*)*line_count);

        //sequentially read all lines
        for(int i = 0; i < line_count; ++i){
            //malloc for each line in the array, assume line is no longer then 32
            resulting_array[i] = malloc(HOLDER_MAX*sizeof (char));

            //read line into the array's line
            fgets(resulting_array[i], HOLDER_MAX, file);
        }

        //close the file
        fclose(file);

        //for each line
```

```
    for (int i = 0; i < line_count; ++i) {

      //knowing the length of a string
      size_t size = strlen(resulting_array[i]);

      //if last character of a string contains newline escape
      if (size > 0 && resulting_array[i][strlen(resulting_array[i])- 1]  == '\
n') {

        //then delete it
        resulting_array[i][strlen(resulting_array[i])- 1] = '\000';
      }
    }

    //return resulting array
    return resulting_array;
  }
  else{

    // if any error occurs with file, print an error and return NULL
    printf("ER:011 - Error while opening text file.\n");
    return NULL;
  }
}

void* handler_init_array(int to_create){

  //provide names of two files being used
  char *start_end_files[] = { NULL,
                    "array_start_locations.txt",
                    "array_end_locations.txt"};

  //get contents of both files into arrays of starting and ending cities
  char** starting_locations = handler_init_file(start_end_files[1]);
  char** ending_locations = handler_init_file(start_end_files[2]);

  //malloc for main array
  array_template* main_array = malloc(sizeof (array_template)*to_create);
  //give a rand() seed to generate upon
  srand(time(0));
```

```
    //get lines from each file
    int start_lines = handler_get_size_of_an_array(start_end_files[1]);
    int end_lines = handler_get_size_of_an_array(start_end_files[2]);

    //assemble main array
    //for each of number entered as a threshold of locations
    for(int i = 0; i < to_create; ++i){

        //randomly select starting and ending point of the route
        strcpy(main_array[i].starting_point, starting_locations[rand()%
start_lines]);
        strcpy(main_array[i].ending_point, ending_locations[rand()%
end_lines]);

        //arbitrarily limit routes to 100, then random from 1 to 100
        // it's common sense that routes of public transport is usually in 1-100
        main_array[i].code = rand()%100;

        //give a user a list of available routes
        if(i == 0){
            printf("Available routes: ");
        }
        //print a route number for each route being generated
        printf("%d ", main_array[i].code);

        //Debug entry to display routes being generated
        //printf("\n%d is:\n",i);
        //printf("%s\n%s\n%d\n\n",main_array[i].starting_point,
main_array[i].ending_point, main_array[i].code);
    }
    //print two newline escapes into terminal
    printf("\n\n");

    //deallocate both distinct arrays of starting and ending locations
    free(starting_locations);
    free(ending_locations);

    return main_array;
}
```