Изображение выглядит как текст, небо, внешний

Автоматически созданное описание

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**   
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**   
**імені ІГОРЯ СІКОРСЬКОГО»**

**Інститут прикладного системного аналізу**   
**Кафедра системного проектування**  
   
   
 

**Звіт**

**про виконання лабораторної роботи №14**   
**«****Структури та бінарні файли»**

Виконав:   
студент I курсу, групи ДА-12   
Панасенко Олександр Андрійович

Прийняв:

к.т.н., доцент Безносик О. Ю.

**Мета роботи**

Вивчити особливості використання структур, об’єднань та файлів, навчитися застосовувати структури, об’єднання, файли в програмуванні.

**Завдання**

Створити масив структур. Кожна структура складається з таких елементів: напрям польоту та літак. Для напряму польоту задати льотні дні на тиждень і номер авіаційного рейсу. Для літака задати тип, кількість посадкових місць, вартість квитка. Створений масив записати до бінарного файлу. Передбачити можливість доповнити файл, замінити дані в файлі, видалити дані з файлу. Реалізувати запити: 1) визначити номери авіаційних рейсів заданого напрямку з мінімальною вартістю квитків; 2) визначити типи літаків і сумарну кількість посадкових місць у заданому напрямку по днях тижня; 3) вилучити з масиву відомості про рейси, якщо кількість польотів на тиждень менше двох.

**Аналіз умови задачі.**

Відповідно до умови задачі, я створю окремий хедер, у якому буду зберігати усі функції для роботи з бінарними файлами, таким чином створюючи невелике API. В основному файлі я створив функції для виконання завдань по пунктам, тобто функції, які отримують дані про літаки та порівнюють/оброблюють їх.

**Код програми**

**Файл main.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <kbin.h>

FILE\* restrict \_writebuffer;

FILE\* restrict \_readbuffer;

char \_path[32] = "../data/14/";

char \_filename[16] = "flight\_data.bin";

// returns 1 if flight\_dirs are equal, otherwise 0

int fdircmp (unsigned char days1[7], unsigned char days2[7]);

// converts enum PlaneType to string

const char\* get\_plane\_type (enum PlaneType plane\_type);

// "constructor" for flight direction (more convenient way)

struct flight\_dir get\_flight\_dir (enum DaysEnum days[], size\_t arr\_len, unsigned long flight\_id);

// returns the flight\_id of a plane with minimal ticket cost at desired flight direction

// returns FLT\_MAX if no appropriate flight\_id was find

unsigned long get\_min\_ticket\_price (enum DaysEnum days[], size\_t days\_len, const char\* filename);

// returns total capacity of all planes, that are flying at defined days

unsigned int get\_capacity (enum DaysEnum days[], size\_t days\_len, const char\* filename);

// removes planes from binary records that fly less than 2 times a week

void remove\_inactive (const char\* filename);

int main(void)

{

char\* filepath = strcat(\_path, \_filename);

struct flight\_data data[3] = {

{

{PASSANGER, 32, 6.99},

get\_flight\_dir((enum DaysEnum[]){TUESDAY}, 1, 3974596)

},

{

{CARGO, 220, 39.99},

get\_flight\_dir((enum DaysEnum[]){MONDAY, TUESDAY}, 2, 19930945)

},

{

{MILITARY, 160, 590.99},

get\_flight\_dir((enum DaysEnum[]){MONDAY, TUESDAY}, 2, 78433)

}

};

write\_data(\_writebuffer, filepath, &data[0], 3);

print\_flight\_data(\_readbuffer, filepath);

struct flight\_data \_data = {

{UNKNOWN, 233, 433.99},

get\_flight\_dir((enum DaysEnum[]){TUESDAY, MONDAY}, 2, 92376434)

};

replace\_data(\_writebuffer, filepath, &\_data, 2);

print\_flight\_data(\_readbuffer, filepath);

remove\_data(\_readbuffer, filepath, 1);

print\_flight\_data(\_readbuffer, filepath);

// PART 1

printf("-- Flight ID with minimal cost: %lu**\n**",

get\_min\_ticket\_price(

(enum DaysEnum[]){MONDAY, TUESDAY},

2, filepath)

);

// PART 2

printf("-- Overall capacity for target direction is %u**\n**",

get\_capacity(

(enum DaysEnum[]){MONDAY, TUESDAY},

2, filepath)

);

// PART 3

remove\_inactive(filepath);

print\_flight\_data(\_readbuffer, filepath);

return 0;

}

int fdircmp(unsigned char days1[7], unsigned char days2[7])

{

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

{

if (days1[i] != days2[i])

return 0;

}

// return dir1.flight\_id == dir2.flight\_id;

return 1; // <- we should ignore flight\_id according to task

}

const char\* get\_plane\_type(enum PlaneType plane\_type)

{

const char\* \_str\_type;

switch (plane\_type)

{

case PASSANGER:

\_str\_type = "passenger";

break;

case CARGO:

\_str\_type = "cargo";

break;

case MILITARY:

\_str\_type = "military";

break;

default:

\_str\_type = "unknown";

}

return \_str\_type;

}

struct flight\_dir get\_flight\_dir(enum DaysEnum days[], size\_t arr\_len, unsigned long flight\_id)

{

struct flight\_dir output = { // <- initialize empty-value struct

{0, 0, 0, 0, 0, 0, 0}, flight\_id

};

unsigned char \_days[7];

for (int i = 0; i < arr\_len; i++) // <- change desired days to 1

{

output.flight\_days[days[i]] = 1;

}

// return (struct flight\_dir){.flight\_days = days, .flight\_id = flight\_id};

return output;

}

unsigned long get\_min\_ticket\_price(enum DaysEnum days[], size\_t days\_len, const char\* filename)

{

struct flight\_data output;

read\_data(\_readbuffer, filename, 0, &output); // <- needed in order to assign the first ticket cost, so there are no UB (ignore unused\_result)

float min\_cost = \_\_FLT\_MAX\_\_; // <- in order to check correctly

unsigned long \_flight\_id;

unsigned char \_u\_days[7] = {0, 0, 0, 0, 0, 0, 0}; // <- temporary solution in order for program to work

for (int i = 0; i < days\_len; i++)

{

\_u\_days[days[i]] = 1;

}

// read\_data will return 0 when there is no more data left in binary file

for (int i = 0; read\_data(\_readbuffer, filename, i, &output); i++) // start from 1, as 0 is already in min\_cost

{

if (!fdircmp(\_u\_days, output.dir.flight\_days)) // returns 0 if dir1 and dir2 aren't equal

continue;

if (min\_cost > output.plane.ticket\_cost)

{

min\_cost = output.plane.ticket\_cost;

\_flight\_id = output.dir.flight\_id;

}

printf("[LOG] Iteration: %d; Minimal Cost: %f; Flight ID: %lu**\n**", i, min\_cost, \_flight\_id);

}

return \_flight\_id;

}

unsigned int get\_capacity(enum DaysEnum days[], size\_t days\_len, const char\* filename)

{

struct flight\_data output;

unsigned int \_ov\_capacity = 0; // overall capacity

unsigned char \_u\_days[7] = {0, 0, 0, 0, 0, 0, 0}; // <- temporary solution in order for program to work

for (int i = 0; i < days\_len; i++)

{

\_u\_days[days[i]] = 1;

}

for (int i = 0; read\_data(\_readbuffer, filename, i, &output); i++)

{

printf("[LOG] For flight id: %d**\n**[LOG] read\_data->days[]: ", output.dir.flight\_id);

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

{

printf("%d ", output.dir.flight\_days[i]);

}

printf("**\n**");

if (fdircmp(\_u\_days, output.dir.flight\_days))

{

const char\* plane\_type = get\_plane\_type(output.plane.plane\_type); // <- convert enum to string

printf("[LOG] Detected plane of type %s with capacity of %u**\n**", plane\_type, output.plane.capacity);

\_ov\_capacity += output.plane.capacity;

}

}

return \_ov\_capacity;

}

void remove\_inactive(const char\* filename)

{

struct flight\_data output;

for (int i = 0; read\_data(\_readbuffer, filename, i, &output); i++)

{

int \_fly\_count = 0;

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

{

if (output.dir.flight\_days[i] == 1)

\_fly\_count++;

}

if (\_fly\_count < 2)

{

printf("[LOG] Detected inactive flight data. Flight ID to be deleted: %u**\n**", output.dir.flight\_id);

remove\_data(\_writebuffer, filename, i); // unsafe operation (UB now)

}

}

}

**Файл kbin.h**

#ifndef KBIN

#define KBIN

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define TRUE 1

#define FALSE 0

// uint8\_t, uint16\_t, uint32\_t and uint64\_t.

// unsigned char, unsigned short, unsigned int and unsigned long long.

enum DaysEnum

{

MONDAY = 0,

TUESDAY = 1,

WEDNESDAY = 2,

THURSDAY = 3,

FRIDAY = 4,

SATURDAY = 5,

SUNDAY = 6

};

enum PlaneType

{

UNKNOWN = 0,

PASSANGER = 1,

CARGO = 2,

MILITARY = 3

};

struct flight\_dir

{

\_\_uint8\_t flight\_days[7]; // <- flight days per week (0 represent that this plane wont fly at that day, vice versa for 1)

\_\_uint64\_t flight\_id;

};

struct plane

{

enum PlaneType plane\_type;

\_\_uint32\_t capacity;

float ticket\_cost;

};

// flight data consists of plane structure and flight direction

// flight direction consists of uint array[7] which represents a week staring from MONDAY and of ulong - flight id

struct flight\_data

{

struct plane plane;

struct flight\_dir dir;

};

// write data with writebuffer to filename with a pointer at first element of array data, which size is len

void write\_data (FILE\* writebuffer, const char\* filename, struct flight\_data\* data, size\_t len);

// write data with writebuffer to filename with a pointer at the element you want to see at index i in binary file

void replace\_data (FILE\* writebuffer, const char\* filename, struct flight\_data\* data, unsigned int i);

// removes record from binary file at target index

void remove\_data (FILE\* readbuffer, const char\* filename, unsigned int i); //\_\_attribute\_malloc\_\_

// get data with readbuffer from filename at index i to \_dest

// index go from 0 to N (zero-based)

// RETURNS 0 if no data was found, 1 if data was successfully read

\_\_uint32\_t read\_data (FILE\* readbuffer, const char\* filename, unsigned int i, struct flight\_data\* \_dest) \_\_attribute\_warn\_unused\_result\_\_;

// print flight data with readbuffer at filename

void print\_flight\_data (FILE\* readbuffer, const char\* filename);

#endif // KBIN

**Файл kbin.c**

#include <stdio.h>

#include <kbin.h>

void remove\_data(FILE\* restrict readbuffer, const char\* filename, unsigned int i)

{

char\* tmp\_filename = malloc(strlen(filename) + 4);

strcpy(tmp\_filename, filename);

strcat(tmp\_filename, ".tmp");

int j = 0;

int found = FALSE;

readbuffer = fopen(filename, "rb");

if (!readbuffer)

{

printf("Unable to open file %s**\n**", filename);

return;

}

FILE\* tmpbuffer = fopen(tmp\_filename, "wb");

if (!tmpbuffer)

{

printf("Unable to open file temp file.**\n**");

return;

}

struct flight\_data output;

for (int j = 0; fread(&output, sizeof(struct flight\_data), 1, readbuffer) == 1; j++)

{

if (i == j)

{

printf("[LOG] Successfully deleted record at index %d**\n**", i);

found = TRUE;

}

else fwrite(&output, sizeof(struct flight\_data), 1, tmpbuffer);

}

if (!found)

printf("Couldn't find a record at index %d**\n**", i);

fclose(readbuffer);

fclose(tmpbuffer);

remove(filename);

rename(tmp\_filename, filename);

}

void write\_data(FILE\* writebuffer, const char\* filename, struct flight\_data\* data, size\_t len)

{

writebuffer = fopen(filename, "ab");

for (size\_t i = 0; i < len; i++)

{

fwrite(&data[i], sizeof(struct flight\_data), 1, writebuffer);

}

fclose(writebuffer);

}

void replace\_data(FILE\* restrict writebuffer, const char\* filename, struct flight\_data\* data, unsigned int i)

{

writebuffer = fopen(filename, "rb+");

fseek(writebuffer, sizeof(struct flight\_data) \* i, SEEK\_SET);

fwrite(data, sizeof(struct flight\_data), 1, writebuffer);

fclose(writebuffer);

}

unsigned int read\_data(FILE\* restrict readbuffer, const char\* filename, unsigned int i, struct flight\_data\* \_dest)

{

readbuffer = fopen(filename, "rb");

fseek(readbuffer, sizeof(struct flight\_data) \* i, SEEK\_SET);

unsigned int result = fread(\_dest, sizeof(struct flight\_data), 1, readbuffer);

fclose(readbuffer);

return result;

}

void print\_flight\_data(FILE\* restrict readbuffer, const char\* filename)

{

readbuffer = fopen(filename, "rb");

while (TRUE)

{

struct flight\_data output;

if (fread(&output, sizeof(struct flight\_data), 1, readbuffer) != 1)

{

if (feof(readbuffer))

printf("[LOG] EOF**\n**");

else printf("[LOG] Error while reading binary**\n**");

break;

}

printf("[LOG] Plane Type: %d; Capacity: %d; Cost: %f; Flight ID: %lu**\n**",

output.plane.plane\_type,

output.plane.capacity,

output.plane.ticket\_cost,

output.dir.flight\_id

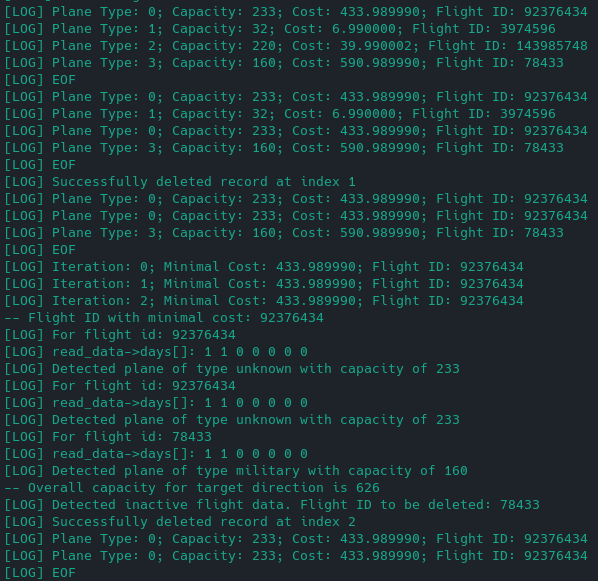
);

}

fclose(readbuffer);

}

Результат виконання програми зображено на рисунку 1.

рисунок 1

**Виновки**

У ході виконання лабораторної роботи №14 я розробив інтерфейс для роботи з бінарними файлами на мові програмування С. Інтерфейс засновано на типі данних FILE та функціях, для роботи з бінарними файлами, такі як fwrite(), fread(). Всі функції базуються на зчитуванні даних з файлу, за потреби редагування цих даних, та їх запису назад у файл.