Національний університет «Одеська політехніка»

Інститут комп'ютерних систем

Кафедра інформаційних систем

**КУРСОВА РОБОТА**

з дисципліни «Алгоритмізація та програмування»

Тема «Програмування динамічної структури даних – стек»

Студента (ки) \_\_1\_\_ курсу АІ-226 групи

Спеціальності 122 – «Комп’ютерні науки»

\_\_\_\_\_\_\_\_\_Ковальський В.Є.\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник доцент, к.т.н. Бабілунга О. Ю.

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

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

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

(підпис) (прізвище та ініціали)

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

(підпис) (прізвище та ініціали)

м. Одеса – 2023 рік

Національний університет «Одеська політехніка»

Інститут комп'ютерних систем

Кафедра інформаційних систем

**ЗАВДАННЯ**

НА КУРСОВУ РОБОТУ

студенту Ковальському Владиславу Євгеновичу група АІ-226

1. Тема роботи

«Програмування динамічної структури даних – стек»

2. Термін здачі студентом закінченої роботи 16.06.2023

3. Початкові дані до проекту (роботи)

Варіант 13

Предметна область – залізничний вокзал. Реалізувати динамічну структуру даних (стек), що містить інформацію про розклад руху поїздів по певній станції:

станція призначення; номер поїзда; час відправлення; час прибуття на станцію призначення; вартість проїзду. Програма повинна забезпечувати: – додавання елемента;

– видалення елемента; – можливість коригування даних; – виведення всіх даних; – видачу довідки про всі поїзди зазначеного напрямку;. – видачу довідки про час прибуття кожного з поїздів зазначеного напрямку на станцію призначення; – видачу довідки про час відправлення поїзда з заданим номером; – видалення з бази записів про потяги зазначеного напрямку; – сортування за часом відправлення.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які належить розробити)

Вступ. Теоретичні відомості про стек. Програмна реалізація стеку. Інструкція користувача. Висновки.

ВСТУП

Ця програма була розроблена та протестована досить успішно. При написані програми в мене не було ніяких проблем. Спочатку в мене була ідея в кожній функції звертатись до файлу, потім дані записувати до стеку і працювати з ними, але трохи пізніше я зрозумів, що так робити не треба, по це виглядає не професійно, та не оптимізовано. Потім я зрозумів, що можна попросту в головній функції відразу визивати функцію, яка буде записувати дані з файлу в стек, а потім працювати зі стеком за допомогою посилань на нього. А так, як в мене організований безкінечний цикл для роботи моєї програми, то в кожній ітеріції я просто зберігаю мій стек в файл, щоб людина вже могла наглядно дивитись на зміни в файлі, або змінювати ці дані. Також я подумав, щоб буде непогано реалізувати функцію clear\_terminal(), яка дає команду на очищення терміналу, в залежності від вашої системи. Наразі там є тільки 2 типи ядер: Windows, Linux.

ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО СТЕК

Це структура даних (стек) містить інформацію про потяги. Кожен елемент стеку (запис) містить такі поля як:

* destination (місце призначення) - рядок, що вказує місце, куди направлений потяг
* train\_number (номер потягу) - ціле число, що ідентифікує потяг
* departure\_time (час відправлення) - рядок, що вказує час відправлення потягу
* arrival\_time (час прибуття) - рядок, що вказує час прибуття потягу
* ticket\_price (ціна квитка) - дійсне число, що вказує ціну квитка на потяг
* next - вказівник на наступний елемент стеку

Структура даних train

struct train {

char destination[50]; // Место назначения

int train\_number; // Номер поезда

char departure\_time[10]; // Время отправления

char arrival\_time[10]; // Время прибытия

double ticket\_price; // Цена билета

struct train\* next;

};

ПРОГРАМНА РЕАЛІЗАЦІЯ СТЕКУ

**Вступ**

Програмна реалізація полягає в використанні структури train для збереження даних про поїзди. У цій структурі зберігаються наступні дані: місце призначення, номер поїзда, час відправлення, час прибуття та час квитка. Крім того, ще використовується вказівник на наступний елемент стеку.

Стек реалізований у вигляді однозв’язного списку, де кожен елемент стеку представлений структурою train. Голова стеку вказує на перший елемент. Для додавання, видалення елементів є відповідні функції: add\_train(), add\_new\_train(), delete\_train(). Також є функції для відображення всіх елементів стеку, функції редагування стеку, функції додавання елементів у стек, функції видалення елементів, функції сортування.

Програмна реалізація цього стеку може бути використана для зберігання даних про розклад руху поїздів та запису цих даних у файл trains.txt. Цей стек може бути корисним для розробки програм, що дозволяють користувачам шукати та переглядати розклади поїздів за різними критеріями, такими як номер поїзда, напрямок, час відправлення тощо.

**Функції та їх призначення**

1. void clear\_terminal() – функція для очищення терміналу. Функція використовується перед виконанням кожної команди для очищення консолі.
2. add\_train(struct train\*\* head, struct train new\_train) – функція для додавання інформації про потяг у стек. Окремо цю функцію користувачеві визвати неможливо, бо вона була створена для гарної роботи функції add\_new\_train()
3. print\_train(struct train one\_train, int number) – функція для написання даних про один рейс у консоль (термінал). Цю функцію користувачеві визвати неможливо, бо вона була створена як допоміжна функції print\_all\_trains()
4. save\_to\_file(struct train\* head, const char\* filename) – функція для запису даних із стеку в файл. Функція запускається після кожної команди для запису даних в файл trains.txt
5. load\_from\_file(const char\* filename) – функція для запису даних з файлу в стек. Ця функція запускається тільки один раз при запуску програми, для того, щоб користувач мав змогу працювати з даними з файлу, які він записав сам, або записав за допомогою програми.
6. add\_new\_train(struct train\*\* head) – функція для додавання інформації про рейс. Користувач викликає саме цю функцію, якщо йому необхідно додати інформацію про новий рейс у стек (далі – в файл).
7. delete\_train(struct train\*\* head) – функція для видалення рейсу зі стеку. Цю функцію викликає користувач, потім вводить номер з розкладу для того, щоб видалити якийсь рейс.
8. edit\_element\_ by\_number(struct train\* head) – функція для редагування рейсу за номером. Користувач може змінити дані в розкладі на нові за допомогою цієх функції.
9. print\_all\_trains(struct train\* head) – функція для написання всіх рейсів. Ця функція використовує допоміжну для неї функцію print\_train(). Користувач викликає цю функцію, якщо йому необхідно вивести розклад (інфо. про всі рейси).
10. show\_train\_by\_destination(struct train\* head) – функція для написання всіх рейсів по вказаному напрямку. При запуску цієї функції користувач вводить напрямок, після чого функція повертає всі рейси за вказаним напрямком.
11. show\_train\_by\_number\_and\_destination(struct train\* head) – функція для написання всіх рейсів по номеру потяга та місцем призначення. Користувач викликає цю функцію, якщо йому необхідно побачити всі рейси, ввівши тільки номер потягу та місце призначення.
12. show\_train\_info(struct train\* head) – функція, яка показує інформацію про рейс за введеним номером потягу. Цю функцію викликає користувач, якщо йому необхідно дізнатись всі рейси тільки по номеру потягу.
13. delete\_trains\_by\_destination(struct train\*\* head) – функція для видалення рейсів за напрямком. Користувач викликає цю функцію, якщо йому необхідно видалити всі рейси за введеним напрямком.
14. swap\_nodes(struct train\* node1, struct train\* node2) – функція для свайпу елементів черги. Користувач не може самостійно викликати цю функцію. Цю функцію викликає функція sort\_trains\_by\_time().
15. sort\_trains\_by\_time(struct train\* head) – функція для сортування рейсів за часом відправлення. Цю функцію може викликати користувач, якщо йому необхідно відсортувати всі рейси за часом відправлення.
16. main() – Головна функція, яка викликається при запуску програми. В ній розташовані всі необхідні налаштування, ініти та основний функціонал програми.

ІНСТРУКЦІЯ КОРИСТУВАЧА

**Загальне**

Інструкцію з використання програми можна побачити на платформі YouTube.

[Тут](https://github.com/vladislavkovalskyi/course_work) показаний весь функціонал програми.

Джерело: <https://youtu.be/wPgkth1VPp4>

Вихідний код програми можна [побачити](https://github.com/vladislavkovalskyi/course_work) на платформі GitHub. В репозиторії якого є вже скомпільований файл програми та відео-інструкція в форматі .mp4 (це ж відео знаходиться на платформі YouTube)

Джерело: <https://github.com/vladislavkovalskyi/course_work>

В випадку, якщо ваша система на Windows, ви можете обійтись без компілювання коду. Вам буде достатньо лише клонувати репозиторій з GitHub за допомогою команди:

git clone <https://github.com/vladislavkovalskyi/course_work.git>

Після чого запустити програму (app.exe)

**Інструкція**

1. Спочатку потрібно скомпілювати нашу програму. Я візьму для прикладу компілятор MinGW. Компілювання відбувається цією командою: gcc Назва\_файлу.c -o Назва\_файлу.exe -Wall -Wextra -g3
2. Ви можете вручну створити файл trains.txt на одному рівні з вашою програмою та записати туди дані по такому шаблону:

Місто Номер\_потягу Час\_відправлення Час\_прибуття Ціна

Наприклад:

Kyiv 24 00:05 07:14 150.00

1. Після компілювання потрібно запустити програму. Я буду показувати на прикладі системи на ОС Windows. Вам потрібно в вашому терміналі (CMD, Powershell), в каталозі вашого файлу ввести команду ./Назва\_файлу.exe
2. Після запуску програми вас зустрічає user-friendly-інтерфейс, за допомогою якого відбувається керування програмою. Вам буде показано меню програми, після чого буде запитувати в вас номери команд, які потрібно виконати. Приклад використання можна побачити ось тут: <https://youtu.be/wPgkth1VPp4>
3. Ваші дані з файлу записуються в стек ТІЛЬКИ при запуску програми, тому щоб ви могли працювати з вашими записаними даними, вам потрібно перезапустити програму, написавши 10 в меню програми, або натиснувши комбінацію клавіш CTRL + C. Після чого знову запустити програму.
4. Дані при роботі програми відразу зберігаються в файлі після кожної команди.

ВИСНОВОК

Після написання цієї програми я досконально вивчив принцип роботи зі стеками та роботу з файлами. Про режими роботи з файлами я писати не буду, бо я їх знав і до цього. Після написання наступних програм, де будуть використані структури даних я зможу робити це більш впевнено.

У даній роботі було представлено реалізацію стеку з використанням мови програмування C (Clang). Стек був реалізований у вигляді однозв’язного списку, де кожен елемент стеку представлений структурою train. Були реалізовані функції додавання, видалення та редагування елементів стеку, а також функції для відображення всіх елементів або відображення елементів за критеріями.

Програма була успішно протестована, і її функціональність була розширена за допомогою функції clear\_terminal(), яка дає змогу очищувати термінал автоматично.

Використання стеку в програмуванні може бути дуже корисним. У цілому дана програма є простим і зручним інструментом для збереження, організації інформації про рух поїздів.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Блок-схема алгоритму – 1 аркуш формату А1.

КОД ПРОГРАМИ

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#define TRAINS\_PATH "trains.txt"

// Структура данных train

struct train {

char destination[50]; // Место назначения

int train\_number; // Номер поезда

char departure\_time[10]; // Время отправления

char arrival\_time[10]; // Время прибытия

double ticket\_price; // Цена билета

struct train\* next;

};

// Функция для очистки терминала

void clear\_terminal() {

#ifdef \_WIN32

system("cls");

#elif \_\_linux\_\_

system("clear");

#else

printf(ERROR\_TEXT

"Ви скомпілювали код на невідомій ОС. Очищення терміналу неможливе\n");

#endif

}

// Функция для добавления поезда

void add\_train(struct train\*\* head, struct train new\_train) {

struct train\* new\_node = (struct train\*)malloc(sizeof(struct train));

\*new\_node = new\_train;

new\_node->next = \*head;

\*head = new\_node;

}

// Функция для написания поезда

void print\_train(struct train one\_train, int number) {

printf(

"• (#%d) Інформація про розклад\n"

"Місце признач.:\t%s\n"

"Номер потягу:\t%d\n"

"Час відправки:\t%s\n"

"Час прибуття:\t%s\n"

"Ціна білету:\t%.2f\n\n",

number, one\_train.destination, one\_train.train\_number,

one\_train.departure\_time, one\_train.arrival\_time, one\_train.ticket\_price);

}

// Функция для записи данных из стека в файл

void save\_to\_file(struct train\* head, const char\* filename) {

FILE\* fp = fopen(filename, "w");

if (fp == NULL) {

printf("Не удалось открыть файл для записи\n");

return;

}

while (head != NULL) {

fprintf(fp, "%s %d %s %s %.2f\n", head->destination, head->train\_number,

head->departure\_time, head->arrival\_time, head->ticket\_price);

head = head->next;

}

fclose(fp);

}

// Функция для считывания данных из файла и записи их в стек

struct train\* load\_from\_file(const char\* filename) {

FILE\* fp = fopen(filename, "r");

if (fp == NULL) {

printf("Не удалось открыть файл для чтения\n");

return NULL;

}

struct train\* head = NULL;

struct train\* tail = NULL;

char dest[50];

int number;

char dep\_time[10];

char arr\_time[10];

double price;

while (fscanf(fp, "%s %d %s %s %lf", dest, &number, dep\_time, arr\_time,

&price) == 5) {

struct train\* new\_train = (struct train\*)malloc(sizeof(struct train));

strcpy(new\_train->destination, dest);

new\_train->train\_number = number;

strcpy(new\_train->departure\_time, dep\_time);

strcpy(new\_train->arrival\_time, arr\_time);

new\_train->ticket\_price = price;

new\_train->next = NULL;

if (head == NULL) {

head = new\_train;

tail = new\_train;

} else {

tail->next = new\_train;

tail = new\_train;

}

}

fclose(fp);

printf("Данные загружены из файла %s\n", filename);

return head;

}

// Функция добавления нового поезда

void add\_new\_train(struct train\*\* head) {

struct train new\_train;

// Для того, чтобы в буфере не оставались знаки переноса строки

while (getchar() != '\n')

;

printf("Введіть інформацію про новий поїзд:\n");

printf("• Місце призначення: ");

scanf("%s", new\_train.destination);

printf("• Номер поїзду: ");

scanf("%d", &new\_train.train\_number);

printf("• Час відправлення: ");

scanf("%s", new\_train.departure\_time);

printf("• Час прибуття: ");

scanf("%s", new\_train.arrival\_time);

printf("• Ціна білету: ");

scanf("%lf", &new\_train.ticket\_price);

add\_train(head, new\_train);

clear\_terminal();

printf("• Новий рейс був успішно доданий та збережений в файл %s\n",

TRAINS\_PATH);

}

// Удаление рейса

void delete\_train(struct train\*\* head) {

struct train\* current = \*head;

struct train\* previous = NULL;

int i = 1;

int number = 0;

printf("• Вкажіть номер розкладу: ");

scanf("%d", &number);

// Если удаляем первый элемент

if (number == 1) {

\*head = current->next;

free(current);

return;

}

// Поиск элемента по номеру

while (i < number && current != NULL) {

previous = current;

current = current->next;

i++;

}

if (current == NULL) {

printf("• Розкладу з номером #%d не знайдено.\n", number);

return;

}

previous->next = current->next;

free(current);

}

// Редактирование рейса

void edit\_element\_by\_number(struct train\* head) {

struct train\* current = head;

int i = 1;

int number = 0;

printf("• Вкажіть номер розкладу: ");

scanf("%d", &number);

while (i < number && current != NULL) { // Поиск элемента по номеру

current = current->next;

i++;

}

if (current == NULL) { // Если элемент не найден

printf("• Розкладу з номером #%d не знайдено.\n", number);

return;

}

// Редактирование элемента

printf("• Місце призначення: ");

scanf("%s", current->destination);

printf("• Номер поїзду: ");

scanf("%d", &current->train\_number);

printf("• Час відправлення: ");

scanf("%s", current->departure\_time);

printf("• Час прибуття: ");

scanf("%s", current->arrival\_time);

printf("• Ціна білету: ");

scanf("%lf", &current->ticket\_price);

}

// Функция написания всех поездов

void print\_all\_trains(struct train\* head) {

struct train\* current = head;

int number = 0;

while (current != NULL) {

number++;

print\_train(\*current, number);

current = current->next;

}

}

// Функция написания поездов по указанному направлению

void show\_train\_by\_destination(struct train\* head) {

struct train\* current = head;

char destination[50];

int found = 0;

printf("• Вкажіть напрямок: ");

scanf("%49s", destination);

while (current != NULL) {

if (strcmp(current->destination, destination) == 0) {

printf("Місце признач.:\t%s\n", current->destination);

printf("Номер потягу:\t%d\n", current->train\_number);

printf("Час відправки:\t%s\n", current->departure\_time);

printf("Час прибуття:\t%s\n", current->arrival\_time);

printf("Ціна білету:\t%.2lf\n\n", current->ticket\_price);

found = 1;

}

current = current->next;

}

if (!found) {

printf("Напрямку \"%s\" не знайдено.\n", destination);

}

}

// Функция, которая печатает поезда по номеру поезда и месту

void show\_train\_by\_number\_and\_destination(struct train\* head) {

struct train\* current = head;

int train\_number;

char destination[50];

printf("• Введіть номер потягу: ");

scanf("%d", &train\_number);

printf("• Введіть станцію призначення: ");

scanf("%49s", destination);

int train\_found = 0;

while (current != NULL) {

if (current->train\_number == train\_number &&

strcmp(current->destination, destination) == 0) {

printf("• Час прибуття:\t%s\n", current->arrival\_time);

train\_found = 1;

}

current = current->next;

}

if (!train\_found) {

printf("Потяг з номером %d та напрямком \"%s\" не знайдено.\n",

train\_number, destination);

}

}

// Показывает инфо о поезде по его номеру

void show\_train\_info(struct train\* head) {

int train\_number;

printf("• Вкажіть номер потягу: ");

scanf("%d", &train\_number);

int found = 0;

struct train\* current = head;

while (current != NULL) {

if (current->train\_number == train\_number) {

printf("• Час відправ.:\t%s\n", current->departure\_time);

printf("• Ціна білету:\t%.2lf грн\n\n", current->ticket\_price);

found = 1;

}

current = current->next;

}

if (!found) {

printf("Потяг з номером %d не знайдено.\n", train\_number);

}

}

// Удаляет поезда по направлению

void delete\_trains\_by\_destination(struct train\*\* head) {

char destination[50];

printf("• Вкажіть напрямок: ");

scanf("%49s", destination);

struct train\* current = \*head;

struct train\* prev = NULL;

while (current != NULL) {

// Если направление совпадает

if (strcmp(current->destination, destination) == 0) {

// Если удаляемый элемент первый в списке

if (prev == NULL) {

\*head = current->next;

} else {

prev->next = current->next;

}

struct train\* temp = current;

current = current->next;

free(temp);

} else {

prev = current;

current = current->next;

}

}

printf("• Всі рейси за напрямком \"%s\" видалені.\n", destination);

}

// Функция для свайпа стеков

void swap\_nodes(struct train\* node1, struct train\* node2) {

char temp\_destination[50];

int temp\_train\_number;

char temp\_departure\_time[10];

char temp\_arrival\_time[10];

double temp\_ticket\_price;

// Сохраняем значения полей первой ноды

strcpy(temp\_destination, node1->destination);

temp\_train\_number = node1->train\_number;

strcpy(temp\_departure\_time, node1->departure\_time);

strcpy(temp\_arrival\_time, node1->arrival\_time);

temp\_ticket\_price = node1->ticket\_price;

// Копируем значения полей в первую ноду

strcpy(node1->destination, node2->destination);

node1->train\_number = node2->train\_number;

strcpy(node1->departure\_time, node2->departure\_time);

strcpy(node1->arrival\_time, node2->arrival\_time);

node1->ticket\_price = node2->ticket\_price;

// Копируем сохраненные значения полей первой ноды во вторую

strcpy(node2->destination, temp\_destination);

node2->train\_number = temp\_train\_number;

strcpy(node2->departure\_time, temp\_departure\_time);

strcpy(node2->arrival\_time, temp\_arrival\_time);

node2->ticket\_price = temp\_ticket\_price;

}

// Функция, которая сортирует поезда по времени

void sort\_trains\_by\_time(struct train\* head) {

struct train\* current;

struct train\* next\_node;

int swapped;

// Если head пустой или только один элемент

if (head == NULL || head->next == NULL) {

return;

}

do {

swapped = 0;

current = head;

while (current->next != NULL) {

next\_node = current->next;

// Если время следующего узла меньше времени текущего узла, меняем их

// местами

if (strcmp(current->departure\_time, next\_node->departure\_time) > 0) {

swap\_nodes(current, next\_node);

swapped = 1;

}

current = next\_node;

}

} while (swapped);

printf("• Рейси були відсортовані.\n");

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

// Загрузка данных из файла в стек при запуске

struct train\* trains = load\_from\_file(TRAINS\_PATH);

int choice = 0;

while (1) {

// Очистка буфера stdin (если вдруг остались данные после scanf)

fflush(stdin);

printf(

"• Меню залізничного вокзалу:\n"

"1. Додати новий рейс\n"

"2. Видалити рейс\n"

"3. Редагувати рейс\n"

"4. Вивести всі рейси\n"

"5. Вивести рейси за напрямком\n"

"6. Вивести час прибуття за потягом та місцем призначення\n"

"7. Вивести інфо. за номером потягу\n"

"8. Видалити рейси за напрямком\n"

"9. Відсортувати рейси за часом\n"

"10. Вихід\n"

"• Введіть номер дії: ");

scanf("%d", &choice);

switch (choice) {

case 1:

clear\_terminal();

add\_new\_train(&trains);

break;

case 2:

clear\_terminal();

delete\_train(&trains);

break;

case 3:

clear\_terminal();

edit\_element\_by\_number(trains);

break;

case 4:

clear\_terminal();

print\_all\_trains(trains);

break;

case 5:

clear\_terminal();

show\_train\_by\_destination(trains);

break;

case 6:

clear\_terminal();

show\_train\_by\_number\_and\_destination(trains);

break;

case 7:

clear\_terminal();

show\_train\_info(trains);

break;

case 8:

clear\_terminal();

delete\_trains\_by\_destination(&trains);

break;

case 9:

clear\_terminal();

sort\_trains\_by\_time(trains);

print\_all\_trains(trains);

break;

case 10:

printf("• До побачення!\n");

exit(0);

break;

default:

printf("• Ви ввели невідому дію.\n");

break;

}

// Сохранение данных в файлах после каждой команды

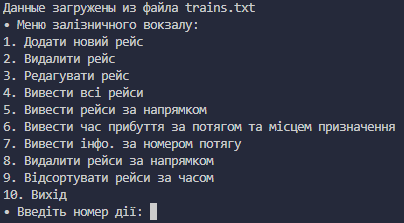
save\_to\_file(trains, TRAINS\_PATH);

}

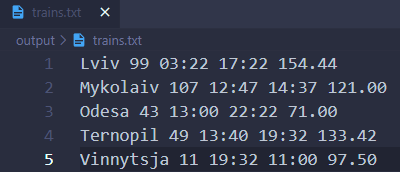
return 0;

}

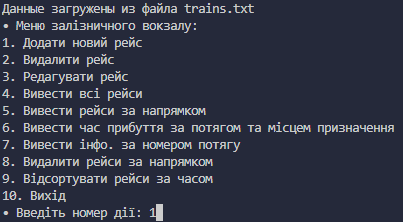
СКРИНШОТИ РЕЗУЛЬТАТІВ РОБОТИ

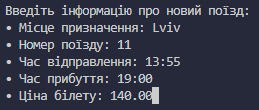


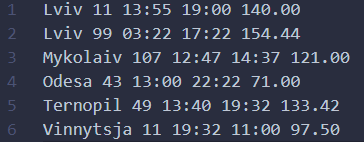
При запуску програми



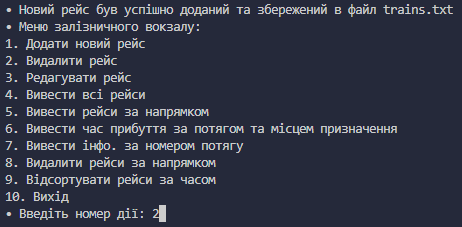
Наприклад, я вже додав в файл дані

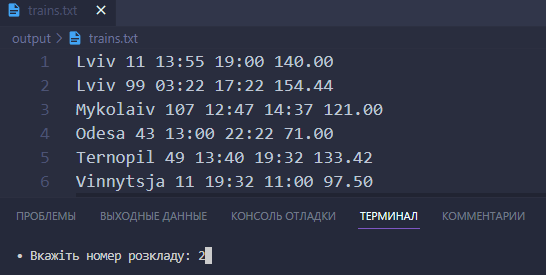


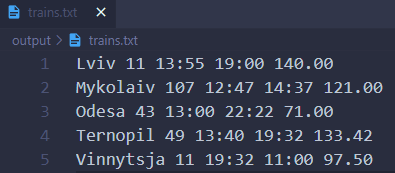




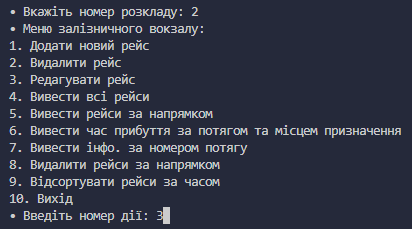
Додавання потягу та результат в файлі

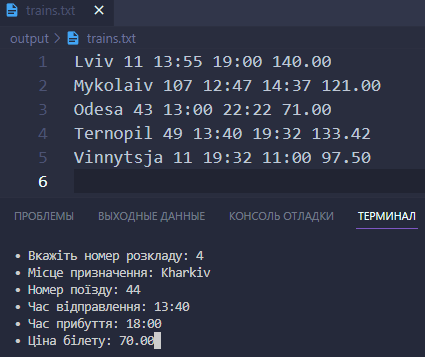


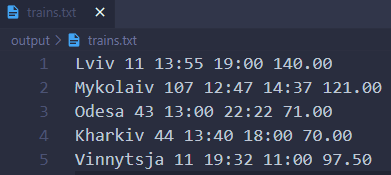




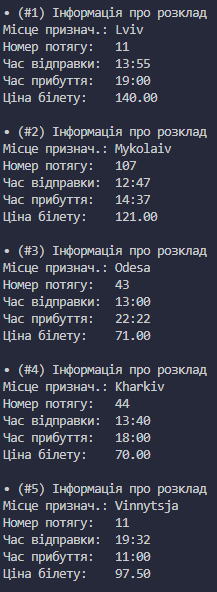
Видалення елементу за номером в розкладі



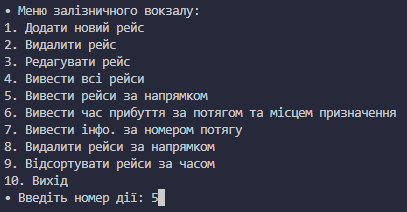


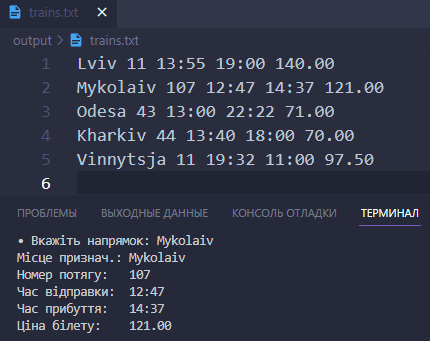


Редагування рейсу за номером в розкладі

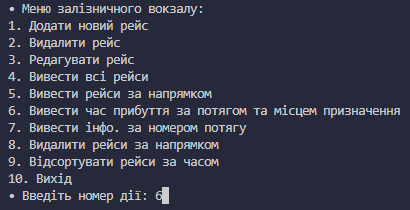


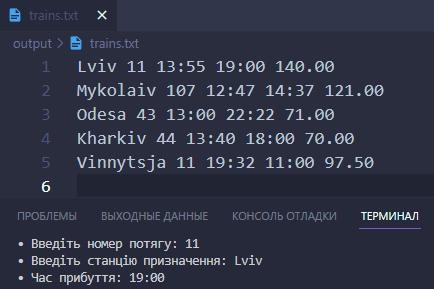
Виведення всіх даних з файлу



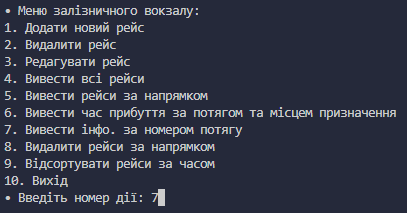


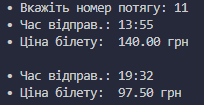
Виведення даних за напрямком



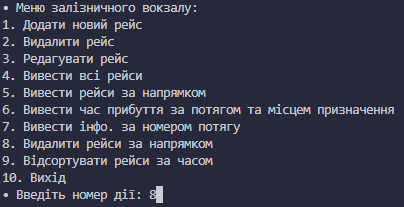


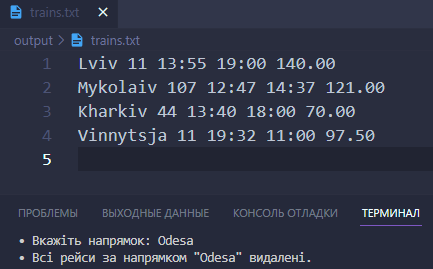
Виведення даних за номером потягу та місцем призначення



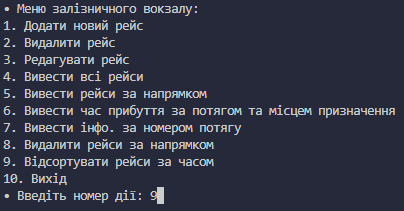


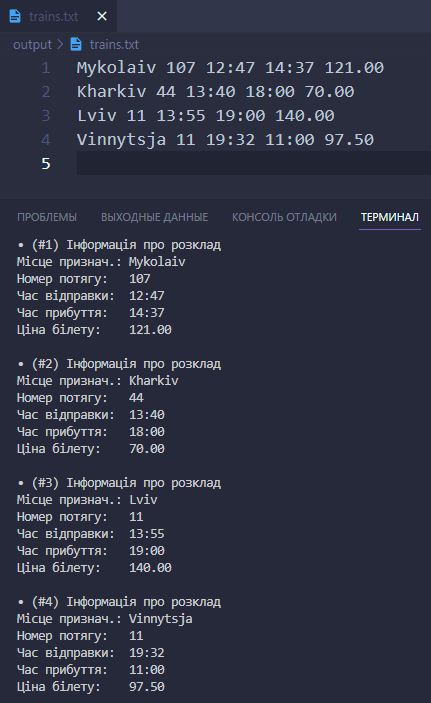
Виведення даних за номером потягу



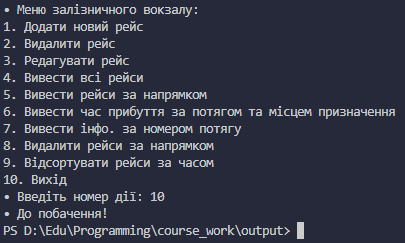


Видалення всіх даних за вказаним напрямком





Сортування рейсів за часом відправлення



Вихід з програми

Завдання видано 20.03.23 \_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Завдання прийнято до виконання 20.03.23 \_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис студента)