Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 2 з дисципліни

«Основи програмування

2. Модульне програмування»

«Файли даних. Бінарні файли»

Виконав студент ІП-14 Нікулін Павло Юрійович
(шифр, прізвище, ім'я, по батькові)

Перевірив Вітковська Ірина Іванівна
(прізвище, ім'я, по батькові)

Київ 2022

Лабораторна робота №2

Файли даних. Бінарні файли

Мета: вивчити особливості створення і обробки бінарних фалів.

Хід роботи

Задача:

Створити файл із списком справ на поточний день: умовна назва, час початку, передбачувана тривалість. Визначити, яка справа за списком наступна (найближча до поточного часу). Створити файл з інформацією про вільний час у другій половині дня (після 13:00): початок та закінчення тимчасового проміжку та його тривалість (розрахувати).

Розв'язання

1. **Постановка задачі**: результатом роботи мають бути два бінарних файли. У першому – список справ, у другому – проміжки вільного часу та його кількість. Складемо функції для створення списку справ, знаходження найближчої справи до поточного часу, обчислення проміжків і кількості вільного часу та функцію зчитування бінарних файлів.

2. Побудова **математичної моделі**. Складемо таблицю імен змінних для двох мов.

C++

Змінна	Тип	Ім'я	Призначення
Хедер до файлу з функціями	Файловий	FUNCS.H	Файл
Файл з функціями	Файловий	FUNCS.CPP	Файл
Тимчасовий файл	Файловий	TEMP.DAT	Файл
Список справ	Файловий	TODO_LIST.DAT	Результат
Інформація про вільний час	Файловий	SPARE_TIME.DAT	Результат
Створення списку справ	Універсальний	CREATE_SCHEDULE	Функція
Сортування справ за часом	Універсальний	SORT_SCHEDULE	Функція
Зчитування бінарних даних	Універсальний	READ_DATA	Функція
Найближча справа за часом	Універсальний	CLOSEST_ACTIVITY	Функція
Вільний час	Універсальний	REST_INFO	Функція
Потокове введення	ifstream	FIN	Початкове дане
Потокове виведення	Ifstream	FOUT	Початкове дане

Назва першого файлу	Рядковий	PATH	Початкове дане
Назва другого файлу	Рядковий	NEW_PATH	Початкове дане
Формальна назва 1-го файлу	Рядковий	FILE_PATH	Проміжне дане
Формальна назва 2-го файлу	Рядковий	NEW_FP	Проміжне дане
Об’єкт структури	Schedule	A	Проміжне дане
Флаг додавання справи	Булевий	ADD	Проміжне дане
Символ підтвердження	Символьний	ADD_STR	Проміжне дане
Масив для сортування справ	Цілий	MIN_ARR	Проміжне дане
Виведення структури	schedule	OUT_DATA	Результат
Виведення структури	spare_time	OUT_ST	Результат
Виведення структури	HM	OUT_REST	Результат
Кількість справ	Цілий	DATA_COUNT	Проміжне дане
Лічильник справ	Цілий	EL_COUNT	Проміжне дане
Поточний час	HM	CURRENT	Проміжне дане
Різниця хвилин	Цілий	DIFF_MIN	Проміжне дане
Наступна різниця хвилин	Цілий	NEXT_DIFF_MIN	Проміжне дане
Найближча справа	schedule	CLOSEST	Результат
Проміжки вільного часу	spare_time	INFO_ST	Проміжне дане
Кількість вільного часу	HM	REST_TIME	Проміжне дане
Год/хв вільного часу	Цілий	REST_HOURS/REST_MIN	Проміжне дане
Поточна хвилина	Цілий	CURR_MIN	Проміжне дане
Попередня хвилина	Цілий	PREV_MIN	Проміжне дане
Остання справа	Цілий	LAST_COUNT	Проміжне дане

Структури С++

Назва	**ТИП**	**Опис**
HM	Структура	Години та хвилини
SCHEDULE	Структура	Список справ
SPARE_TIME	Структура	Інформація про вільний час
TIME	Об’єкт/елемент структури	Елемента структури *schedule*
HOURS	Елемент структури	Елемента структури *time*
MINUTES	Елемент структури	Елемента структури *time*
NAME	Елемент структури	Елемента структури *schedule*
DURATION	Елемент структури	Елемента структури *schedule*
START_TIME	Об’єкт/елемент	Елемента структури *spare_tiime*

	структури	
END_TIME	Об’єкт/елемент структури	Елемента структури *spare_tiime*

Python

Змінна	Тип	Ім’я	Призначення
Файл з функціями	Файловий	FUNCS.PY	Файл
Список справ	Файловий	TODO_LIST.PICKLE	Файл
Інформація про вільний час	Файловий	SPARE_TIME.PICKLE	Файл
Створення списку справ	Універсальний	CREATE_SCHEDULE	Функція
Зчитування бінарних даних	Універсальний	READ_DATA	Функція
Найближча справа за часом	Універсальний	CLOSEST_ACTIVITY	Функція
Інформація про вільний час	Універсальний	REST_INFO	Функція
Назва першого файлу	Рядковий	PATH	Початкове дане
Назва другого файлу	Рядковий	NEW_PATH	Початкове дане
Формальна назва файлу	Рядковий	FILE_PATH	Проміжне дане
Формальна назва файлу	Рядковий	NEW_FP	Проміжне дане
Відкриття/створення файлу	Потік	F	Проміжне дане
Список справ	Словник	SCHEDULE_DATA	Проміжне дане
Флаг додавання справи	Булевий	ADD	Проміжне дане
Символ підтвердження	Рядковий	ADD_STR	Проміжне дане
Список для сортування справ	Цілий	MIN_LIST	Проміжне дане
Дане для сортування	Цілий	TEMP	Проміжне дане
Відсортований список справ	Словник	FINAL_DATA	Проміжне дане
Виведення списку справ	Словник	OUT_DATA	Результат
Виведення інформації про вільний час	Словник	OUT_ST	Результат
Дублювання нуля для виведення	Рядковий	DN	Проміжне дане
Поточний час	Словник	CURRENT	Проміжне дане
Найближча справа	Словник	CLOSEST	Результат
Різниця хвилин	Цілий	DIFF_MIN	Проміжне дане
Наступна різниця хвилин	Цілий	NEXT_DIFF_MIN	Проміжне дане
Інформація про вільний час	Словник	INFO_ST	Проміжне дане
Кількість вільного часу	Цілий	REST_MIN	Проміжне дане
Попередня хвилина	Цілий	PREV_MIN	Проміжне дане

Поточна хвилина	Цілий	CURR_MIN	Проміжне дане

Словники Python

Назва	Тип	Опис
SCHEDULE	Словник	Список справ
NAME	Елемент словника	Назва справи
TIME	Елемент словника	Час початку справи
HOURS	Елемент словника	Година початку справи
MINUTES	Елемент словника	Хвилина початку справи
DURATION	Елемент словника	Тривалість справи
CLOCKS	Словник	Години та хвилини
HH	Елемент словника	Години
MM	Елемент словника	Хвилини
SPARE_TIME	Словник	Проміжки та кількість вільного часу
START_TIME	Елемент словника	Початок проміжку
END_TIME	Елемент словника	Кінець проміжку
REST_TIME	Елемент словника	Кількість вільного часу

Випробування коду

C++

Код

lab1.cpp

```cpp
#include <iostream>
#include "funcs.h"
using namespace std;

int main()
{
    string path = "todo_list.dat";
    string new_path = "spare_time
    create_schedule(path);
    read_data(path);

    closest_activity(path);

    rest_info(path, new_path);
    read_data(new_path);

    cout << "\n";
}
```

funcs.h

```cpp
#pragma once
#include <iostream>
using namespace std;

void create_schedule(string);
void read_data(string);

void closest_activity(string);

void rest_info(string, string);
```

funcs.cpp

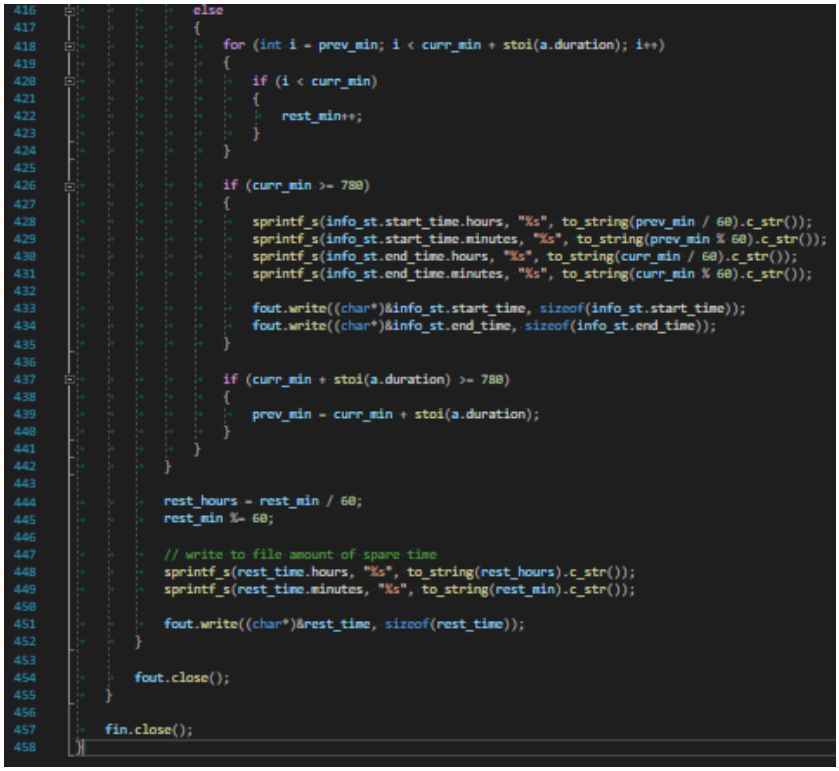
```cpp
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
using namespace std;

struct HM
{
    char hours[255],
        minutes[255];
};

struct schedule
{
    char name[255];
    HM time;
    char duration[255];
};

struct spare_time
{
    HM start_time,
        end_time;
};

void read_data(string file_path)
{
    ifstream fin(file_path, ios::binary);

    schedule out_data;
    spare_time out_st;
    HM out_rest;

    int data_count = 0;
    int el_count = 0;

    if (!fin.is_open())
    {
        printf("Can't open '%s' file :(", file_path.c_str());
    }
    else
    {
        if (file_path.compare("todo_list.dat") == 0) // output for to-do list
        {
            cout << "-----SCHEDULE FOR TODAY-----\n";
            while (fin.read((char*)&out_data, sizeof(schedule)))
            {
                cout << left << setw(7) << out_data.name;
                cout << left << setw(7) << "" << out_data.time.hours << ":" << out_data.time.minutes;
                cout << left << setw(7) << "" << out_data.duration << " minutes\n";
            }
        }
        else // output for info about spare time
        {
            cout << "\n-----SPARE TIME-----";

            while (fin.read((char*)&out_st, sizeof(spare_time)))
            {
                data_count++;
            }

            fin.clear();
            fin.seekg(0);

            while (fin.read((char*)&out_st, sizeof(spare_time)))
            {
                el_count++;

                if (strcmp(out_st.start_time.hours, "0") == 0)
                {
                    sprintf_s(out_st.start_time.hours, "%s", "00");
                }
                else if (strcmp(out_st.start_time.minutes, "0") == 0)
                {
                    sprintf_s(out_st.start_time.minutes, "%s", "00");
                }
                else if (strcmp(out_st.end_time.hours, "0") == 0)
                {
                    sprintf_s(out_st.end_time.hours, "%s", "00");
                }
                else if (strcmp(out_st.end_time.minutes, "0") == 0)
                {
                    sprintf_s(out_st.end_time.minutes, "%s", "00");
                }

                printf("\n%s:%s - %s:%s", out_st.start_time.hours, out_st.start_time.minutes, out_st.end_time.hours, out_st.end_time.minutes);
```

```
        }

        fin.clear();
        fin.seekg(0);

        while (fin.read((char*)&out_rest, sizeof(HM)))
        {
            if (stoi(out_rest.hours) <= 11)
            {
                printf("\nTotal: %s hours %s min", out_rest.hours, out_rest.minutes);
            }
        }
    }

    fin.close();
}

void sort_schedule(string fp)
{
    schedule a;
    int el_count = 0;

    ifstream fin(fp, ios::in | ios::binary);

    if (!fin.is_open())
    {
        printf("Can't open '%s' file :(", fp.c_str());
    }
    else
    {
        ofstream fout("temp.dat", ios::out | ios::binary);

        if (!fout.is_open())
        {
            printf("Can't open 'temp.dat' file :(");
        }
        else
        {
            while (fin.read((char*)&a, sizeof(schedule)))
            {
                fout.write((char*)&a, sizeof(schedule));
            }
        }

        fout.close();
    }

    fin.close();

    fin.open("temp.dat", ios::in | ios::binary);

    if (!fin.is_open())
    {
        printf("Can't open 'temp.dat' file :(");
    }
    else
    {
        while (fin.read((char*)&a, sizeof(schedule)))
        {
            el_count++;
        }

        fin.clear();
        fin.seekg(0);

        int* min_arr = new int[el_count];
        el_count = 0;
        while (fin.read((char*)&a, sizeof(schedule)))
        {
            min_arr[el_count] = stoi(a.time.hours) * 60 + stoi(a.time.minutes);
            el_count++;
        }

        for (int k = 0; k < el_count - 1; ++k)
        {
            for (int i = 0; i < el_count - k - 1; ++i)
            {
                if (min_arr[i] > min_arr[i + 1])
                {
                    int temp = min_arr[i];
                    min_arr[i] = min_arr[i + 1];
                    min_arr[i + 1] = temp;
                }
            }
        }

```

```
    ofstream fout(fp, ios::out | ios::binary);

    if (!fout.is_open())
    {
        printf("Can't open '%s' file :(", fp.c_str());
    }
    else
    {
        fin.clear();
        fin.seekg(0);

        for (int i = 0; i < el_count; i++)
        {
            fin.clear();
            fin.seekg(0);

            while (fin.read((char*)&a, sizeof(schedule)))
            {
                if (stoi(a.time.hours) * 60 + stoi(a.time.minutes) == min_arr[i])
                {
                    fout.write((char*)&a, sizeof(schedule));
                }
            }
        }
    }

    fout.close();
}

fin.close();

remove("temp.dat");
}

void create_schedule(string file_path)
{
    schedule a;
    bool add = 1;
    char add_str = ' ';
    int el_count = 0;

    ifstream fin(file_path, ios::in | ios::binary);

    if (!fin.is_open())
    {
        cout << "-----CREATE YOUR SCHEDULE------";
    }
    else
    {
        read_data(file_path);
        cout << "\n-----ADD NEW ACTIVITIES------";
    }

    fin.close();

    ofstream fout(file_path, ios::out | ios::binary | ios::app);

    if (!fout.is_open())
    {
        printf("Can't open '%s' file :(", file_path.c_str());
    }
    else
    {
        while (add)
        {
            cout << "\nAdd an activity? [y] ";
            cin >> add_str;

            if (add_str == 'y' || add_str == 'Y')
            {
                add = 1;
            }
            else
            {
                add = 0;
            }

            if (add) // adding new activity
            {
                cout << "Enter name: ";
                cin >> a.name;
                cout << "Enter start hour (hh): ";
                cin >> a.time.hours;
                cout << "minute (mm): ";
                cin >> a.time.minutes;
                cout << "Enter duration (minutes): ";
                cin >> a.duration;
```

```
void rest_info(string file_path, string new_fp)
{
    schedule a;
    spare_time info_st;
    HM rest_time;

    int rest_hours = 0, rest_min = 0; // general rest time
    int curr_min = 0, prev_min = 780;
    int last_count = 0, el_count = 0;

    ifstream fin(file_path, ios::binary);

    if (!fin.is_open())
    {
        printf("Can't open '%s' file :(", file_path.c_str());
    }
    else
    {
        ofstream fout(new_fp, ios::binary);

        if (!fout.is_open())
        {
            printf("Can't open '%s' file :(", new_fp.c_str());
        }
        else
        {
            while (fin.read((char*)&a, sizeof(schedule)))
            {
                last_count++;
            }

            fin.clear();
            fin.seekg(0);

            while (fin.read((char*)&a, sizeof(schedule)))
            {
                el_count++;

                curr_min = (stoi(a.time.hours) * 60) + stoi(a.time.minutes);
                if (el_count == last_count)
                {
                    for (int i = prev_min; i < 1440; i++)
                    {
                        if (i < curr_min || i >= curr_min + stoi(a.duration))
                        {
                            rest_min++;
                        }
                    }

                    // write to file spare time interval (same below)
                    sprintf_s(info_st.start_time.hours, "%s", to_string(prev_min / 60).c_str());
                    sprintf_s(info_st.start_time.minutes, "%s", to_string(prev_min % 60).c_str());
                    sprintf_s(info_st.end_time.hours, "%s", to_string(curr_min / 60).c_str());
                    sprintf_s(info_st.end_time.minutes, "%s", to_string(curr_min % 60).c_str());

                    fout.write((char*)&info_st.start_time, sizeof(info_st.start_time));
                    fout.write((char*)&info_st.end_time, sizeof(info_st.end_time));

                    prev_min = curr_min + stoi(a.duration);

                    if (prev_min < 1440)
                    {
                        sprintf_s(info_st.start_time.hours, "%s", to_string(prev_min / 60).c_str());
                        sprintf_s(info_st.start_time.minutes, "%s", to_string(prev_min % 60).c_str());
                        sprintf_s(info_st.end_time.hours, "%s", "23");
                        sprintf_s(info_st.end_time.minutes, "%s", "59");

                        fout.write((char*)&info_st.start_time, sizeof(info_st.start_time));
                        fout.write((char*)&info_st.end_time, sizeof(info_st.end_time));
                    }
                }
```

```
                cin >> a.duration;

                fout.write((char*)&a, sizeof(schedule));
            }
            else
            {
                cout << "\n";
            }
        }
    }

    fout.close();

    sort_schedule(file_path);
}
void closest_activity(string file_path)
{
    HM current;

    int diff_min = 1440, next_diff_min; // difference in time between 2 activities
    int data_count = 0;
    int el_count = 0;

    cout << "\n-----CLOSEST ACTIVITY-----\nEnter current time:\n";
    cout << "hours (hh): ";
    cin >> current.hours;
    cout << "minutes (mm): ";
    cin >> current.minutes;

    schedule closest;

    ifstream fin(file_path, ios::binary);

    if (!fin.is_open())
    {
        printf("Can't open '%s' file :(", file_path.c_str());
    }
    else
    {
        while (fin.read((char*)&closest.name, sizeof(closest.name)))
        {
            data_count++;

            if (data_count % 2 == 0 && data_count % 4 != 0)
            {
                el_count++;
            }
        }

        for (int i = 0; i < el_count; i++)
        {
            fin.clear();
            fin.seekg(0);

            while (fin.read((char*)&closest, sizeof(schedule)))
            {
                if (stoi(closest.time.hours) > stoi(current.hours) || (stoi(closest.time.hours) == stoi(current.hours) && stoi(closest.time.minutes) > stoi(current.minutes)))
                {
                    next_diff_min = (stoi(closest.time.hours) * 60) - (stoi(current.hours) * 60) + stoi(closest.time.minutes) - stoi(current.minutes);
                    if (next_diff_min >= 0 && next_diff_min <= diff_min)
                    {
                        diff_min = next_diff_min;
                    }
                }
            }
        }

        fin.clear();
        fin.seekg(0);

        while (fin.read((char*)&closest, sizeof(schedule)))
        {
            next_diff_min = (stoi(closest.time.hours) * 60) - (stoi(current.hours) * 60) + stoi(closest.time.minutes) - stoi(current.minutes);

            if (next_diff_min == diff_min) // output the closest activity
            {
                printf("\nClosest activity is '%s' at %s:%s\n", closest.name, closest.time.hours, closest.time.minutes);
            }
        }
    }

    fin.close();
}
```



```
            else
            {
                for (int i = prev_min; i < curr_min + stoi(a.duration); i++)
                {
                    if (i < curr_min)
                    {
                        rest_min++;
                    }
                }

                if (curr_min >= 780)
                {
                    sprintf_s(info_st.start_time.hours, "%s", to_string(prev_min / 60).c_str());
                    sprintf_s(info_st.start_time.minutes, "%s", to_string(prev_min % 60).c_str());
                    sprintf_s(info_st.end_time.hours, "%s", to_string(curr_min / 60).c_str());
                    sprintf_s(info_st.end_time.minutes, "%s", to_string(curr_min % 60).c_str());

                    fout.write((char*)&info_st.start_time, sizeof(info_st.start_time));
                    fout.write((char*)&info_st.end_time, sizeof(info_st.end_time));
                }

                if (curr_min + stoi(a.duration) >= 780)
                {
                    prev_min = curr_min + stoi(a.duration);
                }
            }
        }

        rest_hours = rest_min / 60;
        rest_min %= 60;

        // write to file amount of spare time
        sprintf_s(rest_time.hours, "%s", to_string(rest_hours).c_str());
        sprintf_s(rest_time.minutes, "%s", to_string(rest_min).c_str());

        fout.write((char*)&rest_time, sizeof(rest_time));
    }

    fout.close();
}

fin.close();
}
```

Результат

```
-----SCHEDULE FOR TODAY-----
kpi             11:00       180 minutes
lunch           13:30       20 minutes
dinner          18:10       20 minutes
film            21:45       125 minutes

-----ADD NEW ACTIVITIES------
Add an activity? [y] y
Enter name: appoinment
Enter start hour (hh): 15
minute (mm): 45
Enter duration (minutes): 60

Add an activity? [y] n

-----SCHEDULE FOR TODAY-----
kpi             11:00       180 minutes
lunch           13:30       20 minutes
appoinment         15:45       60 minutes
dinner          18:10       20 minutes
film            21:45       125 minutes

-----CLOSEST ACTIVITY-----
Enter current time:
hours (hh): 07
minutes (mm): 59

Closest activity is 'kpi' at 11:00

-----SPARE TIME-----
14:00 - 13:30
13:50 - 15:45
16:45 - 18:10
18:30 - 21:45
23:50 - 23:59
Total: 6 hours 45 min
```

Python

Код

main.py

```python
from funcs import *

path = "todo_list.pickle"
new_path = "spare_time.pickle"

create_schedule(path)
read_data(path)

closest_activity(path)

rest_info(path, new_path)
read_data(new_path)

```

funcs.py

```python
import pickle
import os

schedule = {
    'name': [],
    'time': {'hours': [], 'minutes': []},
    'duration': []}

clocks = {'hh': ' ',
          'mm': ' '}

spare_time = {'start_time': {'hh': [], 'mm': []},
              'end_time': {'hh': [], 'mm': []},
              'time': {'hh': " ", 'mm': " "}}

def create_schedule(file_path):
    if not os.path.exists('todo_list.pickle'):
        print("-----CREATE YOUR SCHEDULE-----")
        schedule_data = schedule
    else:
        read_data(file_path)
        print("\n-----ADD NEW ACTIVITIES-----")

        with open(file_path, 'rb') as f:
            schedule_data = pickle.load(f)

        os.remove(file_path)

    add = 1
    while add:
        add_str = input("\nAdd an activity? [y] ")
        if add_str == 'y' or add_str == 'Y':
            add = 1
        else:
            add = 0

        if add:  # adding new activity
            schedule_data['name'].append(input("Enter name: "))
            schedule_data['time']['hours'].append(input("Enter start hour (hh): "))
            schedule_data['time']['minutes'].append(input("minute (mm): "))
            schedule_data['duration'].append(input("Enter duration: "))

    with open(file_path, 'wb') as f:
        pickle.dump(schedule_data, f)  # pickling activity to file

    sort_schedule(file_path)

def sort_schedule(fp):
    with open(fp, 'rb') as f:
        schedule_data = pickle.load(f)

    os.remove(fp)

    with open("temp.pickle", 'wb') as temp:
        pickle.dump(schedule_data, temp)

        min_list = []

        for i in range(len(schedule_data['name'])):
```

```
            min_list.append(int(schedule_data['time']['hours'][i]) * 60 + int(schedule_data['time']['minutes'][i]))

        for i in range(len(min_list)):
            for j in range(0, len(min_list) - i - 1):
                if min_list[j] > min_list[j + 1]:
                    temp = min_list[j]
                    min_list[j] = min_list[j + 1]
                    min_list[j + 1] = temp

        with open("temp.pickle", 'rb'):
            final_data = {'name': [],
                          'time': {'hours': [], 'minutes': []},
                          'duration': []}
            for i in range(len(min_list)):
                for j in range(len(schedule_data['name'])):
                    if int(schedule_data['time']['hours'][j]) * 60 + int(schedule_data['time']['minutes'][j]) == min_list[i]:
                        final_data['name'].append(schedule_data['name'][j])
                        final_data['time']['hours'].append(schedule_data['time']['hours'][j])
                        final_data['time']['minutes'].append(schedule_data['time']['minutes'][j])
                        final_data['duration'].append(schedule_data['duration'][j])

        with open(fp, 'wb') as f:
            pickle.dump(final_data, f)

    os.remove("temp.pickle")

def read_data(file_path):
    if os.path.exists("todo_list.pickle") or os.path.exists("spare_time.pickle"):
        with open(file_path, 'rb') as f:
            if file_path == "todo_list.pickle":  # output for to-do list
                print("-----SCHEDULE FOR TODAY-----")

                out_data = pickle.load(f)

                for i in range(len(out_data['name'])):
                    print("%s %s:%s %s min" % (
                        out_data['name'][i],
                        out_data['time']['hours'][i], out_data['time']['minutes'][i],
                        out_data['duration'][i]))
            else:  # output for info about spare time
                out_st = pickle.load(f)
                print("\n-----SPARE TIME-----")
                for i in range(len(out_st['start_time']['hh'])):
                    out_st['start_time']['hh'][i] = double_null(out_st['start_time']['hh'][i])
                    out_st['start_time']['mm'][i] = double_null(out_st['start_time']['mm'][i])
                    out_st['end_time']['hh'][i] = double_null(out_st['end_time']['hh'][i])
                    out_st['end_time']['mm'][i] = double_null(out_st['end_time']['mm'][i])

                    print("%s:%s - %s:%s" % (out_st['start_time']['hh'][i], out_st['start_time']['mm'][i],
                                             out_st['end_time']['hh'][i], out_st['end_time']['mm'][i]))
                print("Total: %s hours %s min" % (out_st['time']['hh'], out_st['time']['mm']))
    else:
        print("Can't open '%s' file :(" % file_path)

def double_null(dn): # doubling null func
    if dn == "0":
        dn = ''.join("00")

    return dn
```

```
def closest_activity(file_path):
    current = clocks

    diff_min = 1440

    current['hh'] = input("\n-----CLOSEST ACTIVITY-----\nEnter current hour (hh): ")
    current['mm'] = input("Enter current minute (mm): ")

    with open(file_path, 'rb') as f:
        closest = pickle.load(f) # getting data from file

        for i in range(len(closest['name'])): # calculations for each activity
            if int(closest['time']['hours'][i]) > int(current['hh']) or (
                    int(closest['time']['hours'][i]) == int(current['hh']) and int(closest['time']['minutes'][i]) > int(current['mm'])):
                next_diff_min = (int(closest['time']['hours'][i]) * 60) - (int(current['hh']) * 60) + int(
                    closest['time']['minutes'][i]) - int(current['mm'])
                if 0 <= next_diff_min <= diff_min:
                    diff_min = next_diff_min

        for i in range(len(closest['name'])): # output the closest activity
            next_diff_min = (int(closest['time']['hours'][i]) * 60) - (int(clocks['hh']) * 60) + int(
                closest['time']['minutes'][i]) - int(clocks['mm'])
            if next_diff_min == diff_min:
                print("\nClosest activity is '%s' at %s:%s" % (
                    closest['name'][i], closest['time']['hours'][i], closest['time']['minutes'][i]))


def rest_info(file_path, new_fp):
    info_st = spare_time

    rest_min = 0
    prev_min = 780
    with open(file_path, 'rb') as f:
        schedule_data = pickle.load(f)
        for i in range(len(schedule_data['name'])):
            curr_min = (int(schedule_data['time']['hours'][i]) * 60) + int(schedule_data['time']['minutes'][i])

            if i == len(schedule_data['name']) - 1:
                for j in range(prev_min, 1440):
                    if j < curr_min or j >= curr_min + int(schedule_data['duration'][i]):
                        rest_min += 1

                # write data to dict (same below)
                info_st['start_time']['hh'].append(str(prev_min // 60))
                info_st['start_time']['mm'].append(str(prev_min % 60))
                info_st['end_time']['hh'].append(str(curr_min // 60))
                info_st['end_time']['mm'].append(str(curr_min % 60))

                prev_min = curr_min + int(schedule_data['duration'][i])

                if prev_min < 1440:
                    info_st['start_time']['hh'].append(str(prev_min // 60))
                    info_st['start_time']['mm'].append(str(prev_min % 60))
                    info_st['end_time']['hh'].append("23")
                    info_st['end_time']['mm'].append("59")

            else:
                for j in range(prev_min, curr_min + int(schedule_data['duration'][i])):
```

```
                    if j < curr_min:
                        rest_min += 1
                if curr_min >= 780:
                    info_st['start_time']['hh'].append(str(prev_min // 60))
                    info_st['start_time']['mm'].append(str(prev_min % 60))
                    info_st['end_time']['hh'].append(str(curr_min // 60))
                    info_st['end_time']['mm'].append(str(curr_min % 60))

                if curr_min + int(schedule_data['duration'][i]) >= 780:
                    prev_min = curr_min + int(schedule_data['duration'][i])

    info_st['time']['hh'] = (str(rest_min // 60))
    info_st['time']['mm'] = (str(rest_min % 60))

    with open(new_fp, 'wb') as f:
        pickle.dump(info_st, f)

```

Результат

```
-----SCHEDULE FOR TODAY-----
kpi 11:00 180 min
lunch 13:30 20 min
dinner 18:10 20 min
film 21:45 125 min

-----ADD NEW ACTIVITIES-----

Add an activity? [y] y
Enter name: appoinment
Enter start hour (hh): 15
minute (mm): 45
Enter duration: 60

Add an activity? [y] n
-----SCHEDULE FOR TODAY-----
kpi 11:00 180 min
lunch 13:30 20 min
appoinment 15:45 60 min
dinner 18:10 20 min
film 21:45 125 min

-----CLOSEST ACTIVITY-----
Enter current hour (hh): 08
Enter current minute (mm): 00

Closest activity is 'kpi' at 11:00

-----SPARE TIME-----
14:00 - 13:30
13:50 - 15:45
16:45 - 18:10
18:30 - 21:45
23:50 - 23:59
Total: 6 hours 45 min
```

Висновок

Під час виконання лабораторної роботи було досліджено особливості створення і обробки бінарних фалів. Для збереження даних і подальшого їх запису у бінарний файл було використано структури у *C++* та словники у *Python*. Існуючий список справ записується у тимчасовий файл; при додаванні нової справ, увесь список сортується за часом початку, записується у призначений для нього файл, а тимчасовий файл програмно видаляється. Час початку справи, проміжки вільного часу зберігаються в якості символів, для коректного виводу, і перекладається у цілий тип даних для розрахунків. Знайдено усі проміжки вільного часу, а також його кількість загалом. Виведено усі початкові, проміжні і кінцеві дані. Роботу виконано на двох мовах програмування, програма працює коректно.