

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. Метидології програмування»

«Бінарні файли»

Варіант 18

Виконав студент

ПІ-13 Король Валентин Олегович

(шифр, прізвище, ім'я, по батькові)

Перевірив

Всчерковська Анастасія Сергіївна

(прізвище, ім'я, по батькові)

Київ 2022

Лабораторна робота 2

Варіант 18

Створити файл із переліком технічних перерв у роботі каси: час початку та час кінця перерви. При введенні даних перевіряти, чи не накладається нова перерва на вже наявну. Визначити, чи встигне касир обслужити N клієнтів (N ввести з клавіатури), які стоять у черзі, якщо на одного клієнта в середньому витрачається 15 хв.

Код програми

C++

main.cpp

```
#include "header.h"
using namespace std;

int main()
{
    string path = "File.dat";
    ofstream FileOut(path, ios::binary);
    Period working_hours = Input_Working_Hours();
    int num;
    cout << "\nNumber of breaks: "; cin >> num;
    Output_Breaks_In_File(FileOut, Input_Breaks(num), working_hours, num);
    FileOut.close();

    ifstream FileIn(path, ios::binary);
    Output_File_In_Console(FileIn);
    FileIn.clear();
    FileIn.seekg(0, ios::beg);

    int N;
    int time_for_one_customer = 15;
    cout << "\nNumber of clients: "; cin >> N;
    if (Check_For_Serving_Customers(FileIn, working_hours, N, time_for_one_customer)) {
        cout << "\ncustomers will be served by cashier" << N << " customers for the
working day\n";
    }
    else {
```

```

        cout << "\ncashier will NOT have time to serve" << N << " customers for the
working day\n";
    }
    FileIn.close();
    system("pause");
    return 0;
}

```

header.h

```

#pragma once

#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include <Windows.h>
#include <vector>
using namespace std;

struct Time
{
    int hour;
    int min;
};

struct Period
{
    Time start;
    Time end;
};

void Output_File_In_Console(ifstream&);
Period Input_Working_Hours();
Period* Input_Breaks(int);
void Output_Breaks_In_File(ofstream&, Period*, Period, int);
bool Check_Break(Period*, Period&, Period, int);
bool Break_Is_In_Work_Time(Period&, Period);
bool Breaks_Is_Overlap(Period, Period&);
int Count_Duration(Period);
bool Check_For_Serving_Customers(ifstream&, Period, int, int);

```

header.cpp

```

#include "header.h"
using namespace std;

void Output_File_In_Console(ifstream& file)
{
    Period breaks;
    cout << "\nThe list of technical breaks in the work of cashier:\n\n";
    while (file.read((char*)&breaks, sizeof(Period))) {
        cout.fill('0');
        cout << setw(2) << breaks.start.hour << ":" << setw(2) << breaks.start.min << "
- "
        << setw(2) << breaks.end.hour << ":" << setw(2) << breaks.end.min <<
endl;
    }
}

```

```

Period Input_Working_Hours()
{
    Period work;
    char ch;
    cout << "Enter the start time of the working day (in the format hh:mm): ";
    cin >> work.start.hour >> ch >> work.start.min;
    cout << "Enter the end time of the business day (in the format hh:mm): ";
    cin >> work.end.hour >> ch >> work.end.min;
    return work;
}

Period* Input_Breaks(int n)
{
    char ch;
    Period* breaks = new Period[n];
    cout << "Please enter breaks in ascending order!\n" <<
        "\n--- Start entering the list of technical breaks--- \n";
    for (int i = 0; i < n; ++i) {
        cout << "\nEnter the start time of the break (in the format hh:mm): ";
        cin >> breaks[i].start.hour >> ch >> breaks[i].start.min;
        cout << "Enter the end time of the break (in the format hh:mm): ";
        cin >> breaks[i].end.hour >> ch >> breaks[i].end.min;
    }
    return breaks;
}

void Output_Breaks_In_File(ofstream& file, Period* breaks, Period work, int n)
{
    for (int i = 0; i < n; ++i) {
        if (Check_Break(breaks, breaks[i], work, i)) {
            file.write((char*)&breaks[i], sizeof(Period));
        }
    }
    delete[] breaks;
}

bool Check_Break(Period* breaks, Period& a_break, Period work, int k)
{
    if (!Break_Is_In_Work_Time(a_break, work))
        return false;
    for (int i = 0; i < k; ++i) {
        if (Breaks_Is_Overlap(breaks[i], a_break))
            return false;
    }
    return true;
}

bool Break_Is_In_Work_Time(Period& breaks, Period work)
{
    if (breaks.end.hour < work.start.hour ||
        breaks.end.hour == work.start.hour && breaks.end.min <= work.start.min)
        return false;
    if (breaks.start.hour > work.end.hour ||
        breaks.start.hour == work.end.hour && breaks.start.min >= work.end.min)
        return false;
    if (breaks.start.hour < work.start.hour ||
        breaks.start.hour == work.start.hour && breaks.start.min < work.start.min)
    {
        breaks.start.hour = work.start.hour;
        breaks.start.min = work.start.min;
    }
    if (breaks.end.hour > work.end.hour ||

```

```

        breaks.end.hour == work.end.hour && breaks.end.min > work.end.min)
    {
        breaks.end.hour = work.end.hour;
        breaks.end.min = work.end.min;
    }
    return true;
}

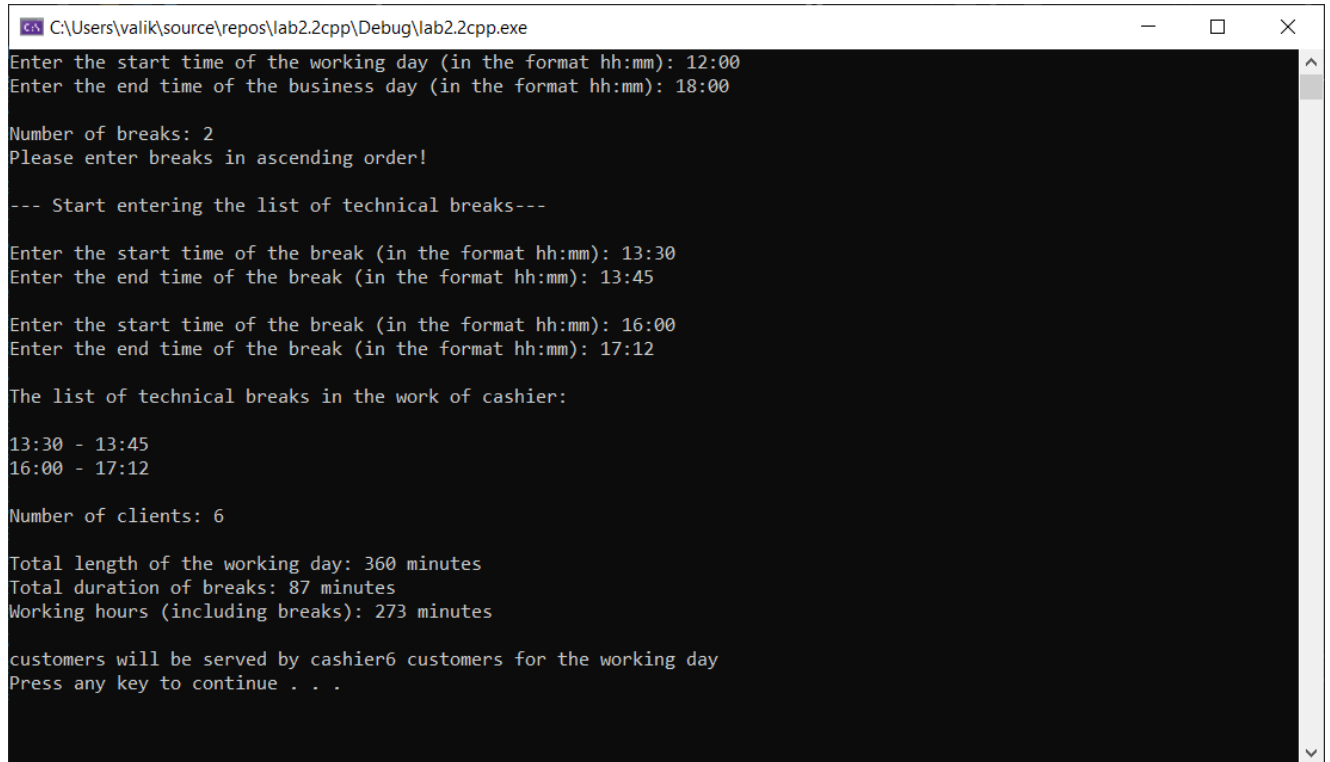
bool Breaks_Is_Overlap(Period A, Period& B)
{
    if ((B.start.hour > A.start.hour || B.start.hour == A.start.hour && B.start.min >=
A.start.min)
        && B.end.hour < A.end.hour || B.end.hour == A.end.hour && B.end.min <=
A.end.min)
        return true;
    if ((A.start.hour < B.start.hour || A.start.hour == B.start.hour && A.start.min <=
B.start.min)
        && (A.end.hour > B.start.hour || A.end.hour == B.start.hour && A.end.min >
B.start.min))
    {
        B.start.hour = A.end.hour;
        B.start.min = A.end.min;
    }
    else if ((A.end.hour > B.end.hour || A.end.hour == B.end.hour && A.end.min >=
B.end.min)
        && (A.start.hour > B.start.hour || A.start.hour == B.start.hour && A.start.min >
B.start.min))
    {
        B.end.hour = A.start.hour;
        B.end.min = A.start.min;
    }
    return false;
}

int Count_Duration(Period period)
{
    return (period.end.hour * 60 + period.end.min) - (period.start.hour * 60 +
period.start.min);
}

bool Check_For_Serving_Customers(ifstream& file, Period work, int num_of_cust, int
time_for_one_cust)
{
    int working_day_duration = Count_Duration(work);
    Period breaks;
    int break_time = 0;
    while (file.read((char*)&breaks, sizeof(Period))) {
        break_time += Count_Duration(breaks);
    }
    int work_time = working_day_duration - break_time;
    int time_for_customers = time_for_one_cust * num_of_cust;
    cout << "\nTotal length of the working day: " << working_day_duration << " minutes\n";
    cout << "Total duration of breaks: " << break_time << " minutes\n";
    cout << "Working hours (including breaks): " << work_time << " minutes\n";
    if (work_time >= time_for_customers) return true;
    else return false;
}

```

Тестування:



```
C:\Users\valik\source\repos\lab2.2cpp\Debug\lab2.2cpp.exe
Enter the start time of the working day (in the format hh:mm): 12:00
Enter the end time of the business day (in the format hh:mm): 18:00

Number of breaks: 2
Please enter breaks in ascending order!

--- Start entering the list of technical breaks---

Enter the start time of the break (in the format hh:mm): 13:30
Enter the end time of the break (in the format hh:mm): 13:45

Enter the start time of the break (in the format hh:mm): 16:00
Enter the end time of the break (in the format hh:mm): 17:12

The list of technical breaks in the work of cashier:

13:30 - 13:45
16:00 - 17:12

Number of clients: 6

Total length of the working day: 360 minutes
Total duration of breaks: 87 minutes
Working hours (including breaks): 273 minutes

customers will be served by cashier6 customers for the working day
Press any key to continue . . .
```

Python

functions.py

```
from Classes import Period
import pickle

def Output_File_In_Console(file, n):
    print('\n--- Перелік технічних перерв у роботі каси ---\n')
    a_break = Period()
    file.seek(0,0)
    for i in range(n):
        a_break = pickle.load(file)
        print(str(a_break.start_hour).zfill(2),':',str(a_break.start_min).zfill(2),' - ',
              str(a_break.end_hour).zfill(2),':',str(a_break.end_min).zfill(2),sep='')
    return

def Input_Working_Hours():
    work = Period()
    work.start_hour,work.start_min = map(int, input('Введіть час початку робочого дня (у
форматі гг:хх): ').split(':'))
    work.end_hour,work.end_min = map(int, input('Введіть час кінця робочого дня (у форматі
гг:хх): ').split(':'))
    return work

def Input_Breaks(n):
    print('Будь ласка, вводьте перерви у порядку зростання їх початку!\n',
          '\n--- Початок введення переліку технічних перерв ---')
    breaks = []
    for i in range(n):
        a_break = Period()
        a_break.start_hour,a_break.start_min = map(int, input('\nВведіть час початку перерви
(у форматі гг:хх): ').split(':'))
        a_break.end_hour,a_break.end_min = map(int, input('Введіть час кінця перерви (у
форматі гг:хх): ').split(':'))
        breaks.append(a_break)
    return breaks

def Output_Breaks_In_File(file, breaks, work, n):
    m = 0
    for i in range(n):
        flag, breaks[i] = Check_Break(breaks[i], breaks, work, i)
        if flag:
            pickle.dump(breaks[i], file)
            m += 1
    return m

def Check_Break(a_break:Period, breaks, work, k):
    flag1,a_break = Break_Is_In_Work_Time(a_break,work)
    if not flag1:
        return False, a_break
    for i in range(k):
        flag2,a_break = Breaks_Is_Overlap(breaks[i],a_break)
        if flag2:
            return False, a_break
    return True, a_break

def Break_Is_In_Work_Time(a_break:Period, work:Period):
    if (a_break.end_hour < work.start_hour or a_break.end_hour == work.start_hour and
a_break.end_min <= work.start_min
```

```

        or a_break.start_hour > work.end_hour or a_break.start_hour == work.end_hour and
a_break.start_min >= work.end_min):
            return False, a_break
        if a_break.start_hour < work.start_hour or a_break.start_hour == work.start_hour and
a_break.start_min < work.start_min:
            a_break.start_hour = work.start_hour
            a_break.start_min = work.start_min
        if a_break.end_hour > work.end_hour or a_break.end_hour == work.end_hour and
a_break.end_min > work.end_min:
            a_break.end_hour = work.end_hour
            a_break.end_min = work.end_min
        return True, a_break

def Breaks_Is_Overlap(a_break:Period, check_break:Period):
    if ((a_break.start_hour < check_break.start_hour or a_break.start_hour ==
check_break.start_hour and a_break.start_min <= check_break.start_min)
        and (a_break.end_hour > check_break.end_hour or a_break.end_hour == check_break.end_hour
and a_break.end_min >= check_break.end_min)):
        return True, check_break
    if ((a_break.start_hour < check_break.start_hour or a_break.start_hour ==
check_break.start_hour and a_break.start_min <= check_break.start_min)
        and (a_break.end_hour > check_break.start_hour or a_break.end_hour ==
check_break.start_hour and a_break.end_min > check_break.start_min)):
        check_break.start_hour = a_break.end_hour
        check_break.start_min = a_break.end_min
    elif ((a_break.end_hour > check_break.end_hour or a_break.end_hour == check_break.end_hour
and a_break.end_min >= check_break.end_min)
        and (a_break.start_hour > check_break.start_hour or a_break.start_hour ==
check_break.start_hour and a_break.start_min > check_break.start_min)):
        check_break.end_hour = a_break.start_hour
        check_break.end_min = a_break.start_min
    return False, check_break

def Count_Duration(period:Period):
    return (period.end_hour * 60 + period.end_min) - (period.start_hour * 60 +
period.start_min)

def Check_For_Serving_Customers(file, work:Period, num_of_custs, time_for_one_cust, n):
    working_day_duration = Count_Duration(work)
    a_break = Period()
    break_time = 0
    file.seek(0,0)
    for i in range(n):
        a_break = pickle.load(file)
        break_time += Count_Duration(a_break)
    work_time = working_day_duration - break_time
    time_for_customers = time_for_one_cust * num_of_custs
    print('\nЗагальна тривалість робочого дня:',working_day_duration,'хв')
    print('Сумарна тривалість перерв:',break_time,'хв')
    print('Робочий час (з урахуванням перерв):',work_time,'хв')
    if work_time >= time_for_customers:
        return True
    return Falsepass

```

Classes

```

class Period:
    start_hour : int
    start_min : int
    end_hour : int
    end_min : int

```

main.py


```

from functions import *

path = "File.dat"
working_hours = Input_Working_Hours()
num_of_breaks = int(input('\nВведіть кількість технічних перерв у роботі каси: '))
FileOut = open(path, 'wb')
num_of_checked_breaks =
Output_Breaks_In_File(FileOut, Input_Breaks(num_of_breaks), working_hours, num_of_breaks)
FileOut.close()

FileIn = open(path, 'rb')
print('\nГодини роботи:
', str(working_hours.start_hour).zfill(2), ':', str(working_hours.start_min).zfill(2), ' - ',
      str(working_hours.end_hour).zfill(2), ':', str(working_hours.end_min).zfill(2), sep='')
Output_File_In_Console(FileIn, num_of_checked_breaks)

time_for_one_customer = 15
N = int(input('\nВведіть кількість клієнтів, що стоять у черзі: '))
if Check_For_Serving_Customers(FileIn, working_hours, N,
time_for_one_customer, num_of_checked_breaks):
    print('\nКасир встигне обслужити', N, 'клієнтів за робочий день')
else:
    print('\nКасир НЕ встигне обслужити', N, 'клієнтів за робочий день')
FileIn.close()

```

Тестування:

```

C:\WINDOWS\system32\cmd.exe
Введіть час кінця перерви (у форматі гг:хх): 13:30
Введіть час початку перерви (у форматі гг:хх): 19:45
Введіть час кінця перерви (у форматі гг:хх): 21:00

Години роботи: 08:00 - 22:30

--- Перелік технічних перерв у роботі каси ---

09:30 - 10:05
12:40 - 13:30
19:45 - 21:00

Введіть кількість клієнтів, що стоять у черзі: 31

Загальна тривалість робочого дня: 870 хв
Сумарна тривалість перерв: 160 хв
Робочий час (з урахуванням перерв): 710 хв

Касир встигне обслужити 31 клієнтів за робочий день
Press any key to continue . . .

```

Висновки:

Я вивчив особливості створення і обробки бінарних файлів даних.
Застосував ці навички на практиці.