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»

«Бінарні файли»
Варіант 18

Виконав студент Лазьов Кирило Владиславович
(шифр, прізвище, ім'я, по батькові)

Перевірила Вєчерковська Анастасія Сергіївна
(прізвище, ім'я, по батькові)

Лабораторна робота №2

Бінарні файли

Варіант 18

Задача

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.

C++

main.cpp:

Header.h | Functions.cpp | Lab2.cpp

Lab2 | (Global Scope)

```cpp
#include"Header.h"

using namespace std;

int main()
{
    cout << "Work shift starts at 8:00 and end at 18:00" << endl;
    string text_write = capture_text();

    write_new_file("input.txt", text_write);
    string text = read_new_file("input.txt");

    printf("\nText on input:\n%s\n\n", text.c_str());

    int number;
    cout << "Enter number of customers: ";
    cin >> number;

    bool isEnough = count_time(text, number);

    if (isEnough) {
        cout << "There is enough time for " << number << " customers";
    }
    else {
        cout << "There is not enough time for " << number << " customers";
    }
}
```

Header.h:

Header.h | Functions.cpp | Lab2.cpp

Lab2 (Global Scope)

```cpp
#pragma once
#include <iostream>
#include<iomanip>
#include<fstream>
#include<vector>
#include<string>

using namespace std;

string capture_text();
bool check_Input(const string& line);
bool check_breaks(const string& line, const string& text);
vector<string> split_sentences(const string& text);
vector<string> split_breaks(const string& text);
void write_new_file(const string& file_name, const string& text);
string read_new_file(const string& file_name);
bool count_time(const string& text, int number);
```

Functions.cpp

Header.h | Functions.cpp | Lab2.cpp

Lab2 (Global Scope)

```cpp
#include"Header.h"

using namespace std;

string capture_text() {
    string text;
    string line;

    int acsii_code = 4;
    int counter = 0;

    bool isOverlap = false;
    bool isCorrect = true;

    cout << "Enter breaks(H:mm-H:mm). Ctrl + D - end writing"<<endl;
    while (int(line[0] != acsii_code)) {
        getline(cin, line);
        isCorrect = check_Input(line);
        if (isCorrect) {
            if (counter == 0) {
                text += line + '\n';
                counter++;
            }
            else if (counter > 0 and int(line[0]) != 4) {
                isOverlap = check_breaks(line, text);
                if (isOverlap) {
                    cout << "This break overlaps with another!" << endl;
                }
                else {
                    text += line + '\n';
                }
            }
        }
        else {
            cout << "Wrong Input!" << endl;
        }
    }
    text = text.substr(0, text.size() - 1);
    return text;
}
```

```cpp
bool check_Input(const string& line) {
    bool isCorrect = true;
    if (int(line[0]) != 4) {
        vector<string> brake = split_breaks(text: line);
        if (stoi(_Str: brake[0]) <= 8 && stoi(_Str: brake[1]) == 0 || stoi(_Str: brake[0]) >= 18 || stoi(_Str: brake[2]) >= 18) {
            isCorrect = false;
        }
        if (stoi(_Str: brake[0]) > stoi(_Str: brake[2])) {
            isCorrect = false;
        }
    }
    return isCorrect;
}

bool check_breaks(const string& line, const string& text) {
    vector<string> breaks = split_sentences(text);
    bool isOverlap = false;
    int i = 0;
    while (isOverlap == false and i < breaks.size() - 1) {
        vector<string> first_brake = split_breaks(text: breaks[i]);
        vector<string> second_brake = split_breaks(text: line);
        if (stoi(_Str: second_brake[0]) < stoi(_Str: first_brake[0]) and stoi(_Str: second_brake[2]) > stoi(_Str: first_brake[2]) || stoi(_Str: second_brake[0]) > stoi(_Str: first_brake[0]) and stoi(_Str: second_brake[0]) < stoi(_Str: first_brake[2])) {
            isOverlap = true;
        }
        if (stoi(_Str: second_brake[2]) == stoi(_Str: first_brake[0]) && stoi(_Str: second_brake[3]) >= stoi(_Str: first_brake[1])) {
            isOverlap = true;
        }
        if (stoi(_Str: second_brake[2]) == stoi(_Str: first_brake[2]) && stoi(_Str: second_brake[3]) <= stoi(_Str: first_brake[3])) {
            isOverlap = true;
        }
        if (stoi(_Str: second_brake[0]) == stoi(_Str: first_brake[2]) && stoi(_Str: second_brake[1]) <= stoi(_Str: first_brake[3])) {
            isOverlap = true;
        }
        i++;
    }
    return isOverlap;
}
```

```
vector<string> split_sentences(const string& text) {
    vector<string> sentences;
    int prev_pos = 0;
    int counter = 0;

    while (counter <= text.size()) {
        if (text[counter] == '\n' || counter == text.size()) {
            string sentence = text.substr(_Off:prev_pos, _Count:counter - prev_pos);
            sentences.push_back(_Val:sentence);
            prev_pos = counter + 1;
        }
        counter++;
    }

    return sentences;
}

vector<string> split_breaks(const string& text) {
    vector<string> breaks;
    int prev_pos = 0;
    int counter = 0;
    while (counter <= text.size()) {
        if (text[counter] == '\n' || text[counter] == ':' || text[counter] == '-' || counter == text.size()) {
            string time = text.substr(_Off:prev_pos, _Count:counter - prev_pos);
            breaks.push_back(_Val:time);
            prev_pos = counter + 1;
        }
        counter++;
    }
    return breaks;
}

void write_new_file(const string& file_name, const string& text) {
    ofstream write_file(file_name, ios::binary);
    write_file << text;
    write_file.close();
}

string read_new_file(const string& file_name) {
    ifstream open_file(file_name, ios::binary);
    string text;
    while (!open_file.eof()) {
        text += open_file.get();
    }
    open_file.close();
    text = text.substr(_Off:0, _Count:text.size() - 1);

    return text;
}


bool count_time(const string& text, int number) {
    int Overalltime = 600;
    vector<string> sentences = split_sentences(text);
    int minutes = 0;
    int hours = 0;
    int final_time;
    bool isEnough = false;

    for (int i = 0; i < sentences.size(); i++) {
        vector<string> brake = split_breaks(text:sentences[i]);
        if (stoi(_Str:brake[3]) >= stoi(_Str:brake[1])) {
            minutes += -(stoi(_Str:brake[1]) - stoi(_Str:brake[3]));
        }
        else if(stoi(_Str:brake[1]) > stoi(_Str:brake[3])) {
            hours--;
            minutes += 60 - (stoi(_Str:brake[1]) - stoi(_Str:brake[3]));;
        }
        hours += stoi(_Str:brake[2]) - stoi(_Str:brake[0]);

        if (minutes >= 60) {
            minutes -= 60;
            hours++;
        }
    }
    cout << "hours: " << hours << endl;
    cout << "minutes: " << minutes << endl;
```

```cpp
bool count_time(const string& text, int number) {
    int Overalltime = 600;
    vector<string> sentences = split_sentences(text);
    int minutes = 0;
    int hours = 0;
    int final_time;
    bool isEnough = false;

    for (int i = 0; i < sentences.size(); i++) {
        vector<string> brake = split_breaks(text: sentences[i]);
        if (stoi(_Str: brake[3]) >= stoi(_Str: brake[1])) {
            minutes += -(stoi(_Str: brake[1]) - stoi(_Str: brake[3]));
        }
        else if(stoi(_Str: brake[1]) > stoi(_Str: brake[3])) {
            hours--;
            minutes += 60 - (stoi(_Str: brake[1]) - stoi(_Str: brake[3]));;
        }
        hours += stoi(_Str: brake[2]) - stoi(_Str: brake[0]);

        if (minutes >= 60) {
            minutes -= 60;
            hours++;
        }
    }
    cout << "hours: " << hours << endl;
    cout << "minutes: " << minutes << endl;

    final_time = hours * 60 + minutes;
    cout << "Final time: " << final_time << endl;

    int Reqtime = number * 15;
    cout << "Required time: " << Reqtime << endl;

    if (Overalltime - final_time >= Reqtime) {
        isEnough = true;
    }
    return isEnough;
}
```

Pyhon:

main.py:

main.py func.py

```python
import func

print('Work shift starts at 8:00 and end at 18:00')
text_to_write = func.capture_text()

func.write_empty_file("input.txt", text_to_write)
text = func.read_file("input.txt")
print(f"Text on input:\n{text}\n")

number = int(input("Enter number of customers:"))
isEnough = func.count_time(text, number)
if(isEnough):
    print("There is enough time for", number, "customer")
else:
    print("There is not enough time for", number, "customer")
```

func.py:

main.py func.py

```python
def capture_text():
    print('Enter breaks(H:mm-H:mm). shift + 6 - end writing')
    isCorrect = True
    isOverlap = False
    line = " "
    text = ""
    counter = 0
    acsii_code = 94
    while(ord(line[0]) != 94):
        line = input()
        isCorrect = check_Input(line)
        if(isCorrect):
            if(counter == 0):
                text = text + line + '\n'
                counter = counter + 1
            elif(counter > 0 and ord(line[0]) != 94):
                isOverlap = check_breakes(line, text)
                if(isOverlap):
                    print("This break overlaps with another!")
                else:
                    text = text + line + '\n'
        else:
            print("Wrong input")
    text = text[:len(text) - 1]

    return text

def check_Input(line: str):
    isCorrect = True
    if(ord(line[0]) != 94):
        brakes = split_breaks(line)
        if(int(brakes[0][0]) <= 8 and int(brakes[0][1]) == 0 or int(brakes[0][0]) >=18 or int(brakes[1][0]) >=18):
            isCorrect = False
        if(int(brakes[0][0]) > int(brakes[1][0])):
            isCorrect = False
    return isCorrect
```

```python
def check_Input(line: str):
    isCorrect = True
    if(ord(line[0]) != 94):
        brakes = split_breaks(line)
        if(int(brakes[0][0]) <= 8 and int(brakes[0][1]) == 0 or int(brakes[0][0]) >=18 or int(brakes[1][0]) >=18):
            isCorrect = False
        if(int(brakes[0][0]) > int(brakes[1][0])):
            isCorrect = False
    return isCorrect

def check_breakes(line:str, text:str):
    breaks = text.split('\n')
    isOverlap = False
    i = 0
    while(isOverlap == False and i < len(breaks) - 1):
        first_break = split_breaks(breaks[i])
        second_break = split_breaks(line)
        if(int(second_break[0][0]) < int(first_break[0][0]) and int(second_break[1][0]) > int(first_break[0][0]) or int(second_break[0][0]) > int(first_break[0][0]) and int(second_break[0][0]) < int(first_break[1][0])):
            isOverlap = True
            print("Hours Overlap")
        if(int(second_break[1][0]) == int(first_break[0][0]) and int(second_break[1][1]) >= int(first_break[0][1])):
            isOverlap = True
            print("Minutes 1 overlap")
        if(int(second_break[1][0]) == int(first_break[1][0]) and int(second_break[1][1]) <= int(first_break[1][1])):
            isOverlap = True
            print("Minutes 2 overlap")
        if (int(second_break[0][0]) == int(first_break[1][0]) and int(second_break[0][1]) <= int(first_break[1][1])):
            isOverlap = True
            print("Minutes 3 overlap")
        i = i + 1
    return isOverlap
```

```python
    return isOverlap

def split_breaks(line: str):
    brakes = []
    brake = line.split('-')
    for i in range(len(brake)):
        brake_array = brake[i].split(':')
        brakes.append(brake_array)
    return brakes

def write_empty_file(file_name: str, text: str):
    write_file = open(file_name, 'wt')
    write_file.write(text)
    write_file.close()

def read_file(file_name: str):
    read_input_file = open(file_name, 'rt')
    text = read_input_file.read()
    read_input_file.close()

    return text
```

```python
def count_time(text: str, number: int):
    Overalltime = 600
    breaks = text.split('\n')
    minutes = 0
    hours = 0
    isEnough = False
    i = 0
    while( i < len(breaks)):
        brake = split_breaks(breaks[i])
        if(int(brake[1][1]) >= int(brake[0][1])):
            minutes += -(int(brake[0][1]) - int(brake[1][1]))
        elif(int(brake[0][1]) > int(brake[1][1])):
            hours = hours - 1
            minutes += 60 - (int(brake[0][1]) - int(brake[1][1]))
        hours += int(brake[1][0]) - int(brake[0][0])
        if(minutes >= 60):
            minutes -= 60
            hours += 1
        i += 1
    print("hours", hours)
    print("minutes", minutes)
    final_time = hours * 60 + minutes
    print("final time", final_time)
    Req_time = number * 15
    if(Overalltime - final_time >= Req_time):
        isEnough = True
    return isEnough
```

Результати виконання програми

C++:

```
Work shift starts at 8:00 and end at 18:00
Enter breaks(H:mm-H:mm). Ctrl + D - end writing
9:10-11:20
10:00-12:30
This break overlaps with another!
11:30-13:25
15:40-16:55
17:00-17:55
^D

Text on input:
9:10-11:20
11:30-13:25
15:40-16:55
17:00-17:55

Enter number of customers: 20
hours: 6
minutes: 15
Final time: 375
Required time: 300
There is not enough time for 20 customers
```

input.txt - Notepad

File Edit Format View Help

```
9:10-11:2011:30-13:2515:40-16:5517:00-17:55
```

Python:

```
Work shift starts at 8:00 and end at 18:00
Enter breaks(H:mm-H:mm). shift + 6 - end writing
9:20-11:40
12:00-14:15
9:10-12:20
This break overlaps with another!
15:30-16:20
^
Text on input:
9:20-11:40
12:00-14:15
15:30-16:20

Enter number of customers:15
hours 5
minutes 25
final time 325
There is enough time for 15 customer
```

input.txt - Notepad

File Edit Format View Help

```
9:20-11:40
12:00-14:15
15:30-16:20
```