Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Лабораторная работа № 10

Тема: «Сохранение данных в файле с использованием потоков»

Выполнил: студент группы РИС-22-2б

Мизёв В.А. ф

Проверил: доцент кафедры ИТАС

Полякова О.А.

г. Пермь – 2024

*Постановка задачи:*

1. Создать пользовательский класс с минимальной функциональностью.
2. Написать функцию для создания объектов пользовательского класса (ввод исходной информации с клавиатуры) и сохранения их в потоке (файле).
3. Написать функцию для чтения и просмотра объектов из потока.
4. Написать функцию для удаления объектов их потока в соответствии с заданием варианта. Для выполнения задания выполнить перегрузку необходимых операций.
5. Написать функцию для добавления объектов в поток в соответствии с заданием варианта. Для выполнения задания выполнить перегрузку необходимых операций.
6. Написать функцию для изменения объектов в потоке в соответствии с заданием варианта. Для выполнения задания выполнить перегрузку необходимых операций.
7. Для вызова функций в основной программе предусмотреть меню.

*Задание вариант 1:*

Создать класс Time работы с временными интервалами. Интервал должен быть представлен в виде двух полей: минуты типа int и секунды типа int. При выводе минуты отделяются от секунд двоеточием. Реализовать:

- сложение временных интервалов (учесть, что в минуте не может быть более 60 секунд)

- сравнение временных интервалов (==)

**Задание**

- удалить все записи равные заданному значению.

- увеличить все записи с заданным значением на 1 минуту 30 секунд

- добавить K записей после элемента с заданным номером.

*Анализ задачи:*

Основываясь на задании необходимо реализовать один класс “Time”, с полями m\_minutes, m\_seconds, а также методы для получения и установки значений для этих полей. Три конструктора: конструктор по умолчанию Time(), копирования Time(const Time& time) и с параметрами Time(int minutes, int seconds). Реализовать перегрузку операторов для сравнения, сложения и присваивания.

А также функции для работы с файлами: write() – для записи, read() – для чтения, remove() – для удаления, increaser\_specified\_time() – для добавления времени, add() – для добавления записей в файл, начиная с n элемента.

*Код программы:*

Прикреплён в приложении 1.

*UML-диаграмма класса:*

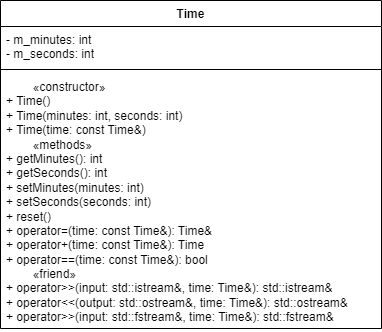


Рисунок 1 - UML-диаграмма классов

*Скриншот работы программы:*

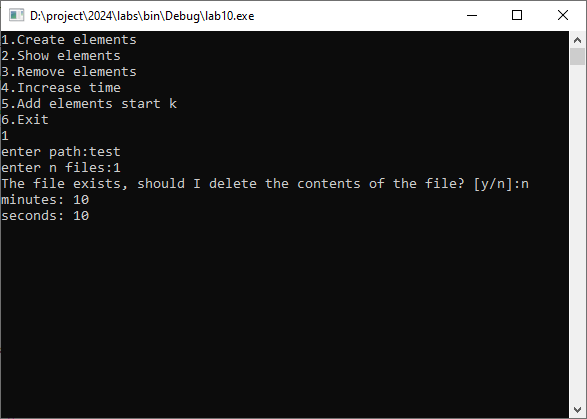


Рисунок 2 – Создание нового файла, запись 1 элемента в файл

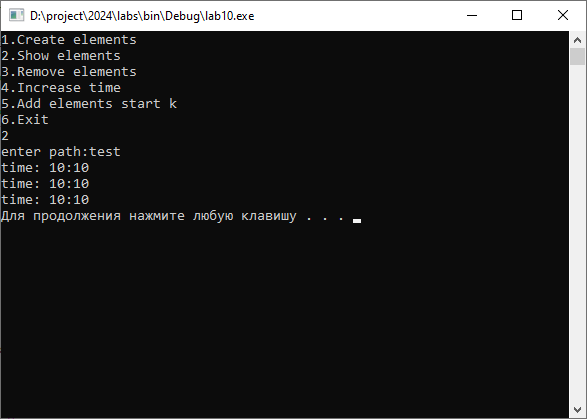


Рисунок 3 – Просмотр файла

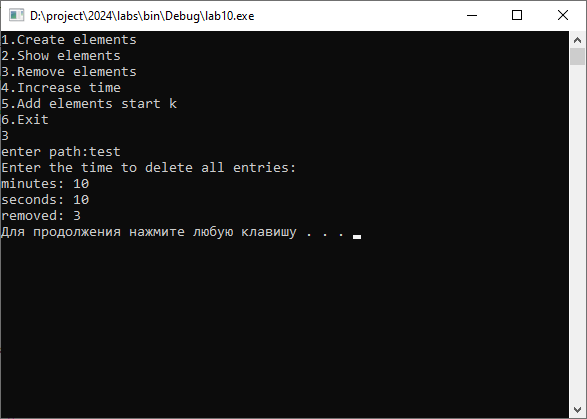


Рисунок 4 - Удаление из файла указанного элемента

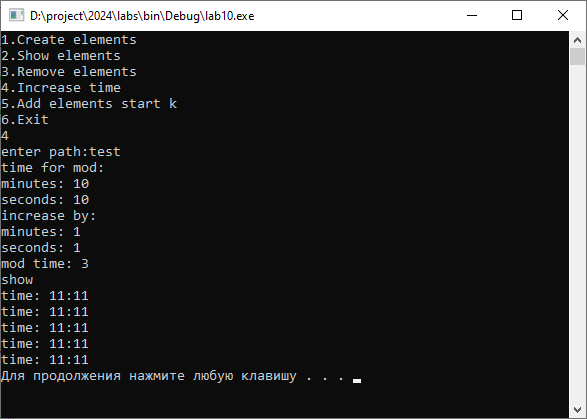


Рисунок 5 - Добавление времени к заданному

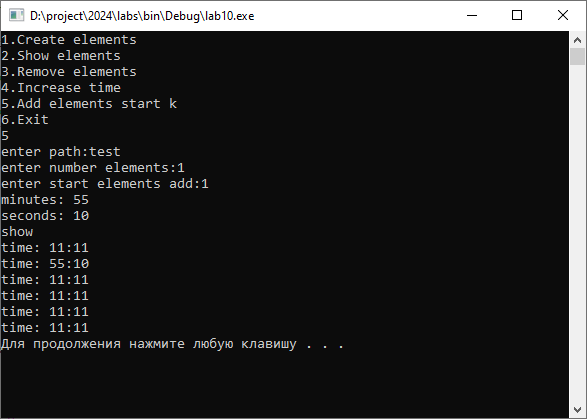


Рисунок 6 - Добавление элементов начиная с N

*Контрольные вопросы:*

1. Что такое поток?

Поток — это абстракция, представляющая последовательный поток данных. Потоки могут использоваться для ввода данных (например, с клавиатуры) или вывода данных (например, на экран).

std::cout << "Hello, World!" << std::endl; // Вывод в стандартный поток

1. Какие типы потоков существуют?

Существует три основных типа потоков:

- Стандартные потоки (std::cin, std::cout, std::cerr)

- Файловые потоки (std::ifstream, std::ofstream, std::fstream)

- Строковые потоки (std::istringstream, std::ostringstream)

1. Какую библиотеку надо подключить при использовании стандартных потоков?

Для использования стандартных потоков необходимо подключить библиотеку <iostream>.

1. Какую библиотеку надо подключить при использовании файловых потоков?

Для работы с файловыми потоками необходимо подключить библиотеку <fstream>.

1. Какую библиотеку надо подключить при использовании строковых потоков?

Для работы со строковыми потоками следует подключить библиотеку <sstream>

6. Какая операция используется при выводе в форматированный поток?

Операция вывода в форматированный поток — это оператор <<.

7. Какая операция используется при вводе из форматированных потоков?

Операция ввода из форматированных потоков — это оператор >>.

8. Какие методы используются при выводе в форматированный поток?

Методы std::cout могут включать std::endl, std::setw, std::setprecision и другие.

9. Какие методы используется при вводе из форматированного потока?

Для ввода обычно используется оператор >>, а также может использоваться метод .getline() для ввода строк.

10. Какие режимы для открытия файловых потоков существуют?

Существует несколько режимов:

- std::ios::in (для чтения)

- std::ios::out (для записи)

- std::ios::app (для добавления)

- std::ios::trunc (для обрезки файла)

- std::ios::binary (для двоичного режима)

11. Какой режим используется для добавления записей в файл?

Режим для добавления записей в файл — это std::ios::app.

12. Какой режим (комбинация режимов) используется в конструкторе

ifstream file("f.txt")?

По умолчанию используется режим std::ios::in (чтение).

13. Какой режим (комбинация режимов) используется в конструкторе

fstream file("f.txt")?

По умолчанию используется режим std::ios::in | std::ios::out (чтение и запись).

14. Какой режим (комбинация режимов) используется в конструкторе

ofstream file("f.txt")?

По умолчанию конструктор ofstream открывает файл в режиме записи (ios::out) и с флагом ios::trunc, что означает обнуление файла, если он уже существует.

15. Каким образом открывается поток в режиме ios::out|ios::app?

В режиме добавления append.

16. Каким образом открывается поток в режиме ios::out |ios::trunc?

Убирает содержимое файла.

17. Каким образом открывается поток в режиме ios::out |ios::in|ios::trunk?

На запись, чтение и удаление содержимого файла.

18. Каким образом можно открыть файл для чтения?

Файл можно открыть для чтения с помощью ifstream или fstream с флагом ios::in.

#include <fstream>

std::ifstream file("f.txt", std::ios::in);

19. Каким образом можно открыть файл для записи?

Файл для записи можно открыть с помощью ofstream или fstream с флагом ios::out.

20. Привести примеры открытия файловых потоков в различных режимах.

#include <fstream>

// 1. Открытие файла только для записи (т trunc)

std::ofstream file1("file1.txt", std::ios::out | std::ios::trunc);

// 2. Открытие файла только для чтения

std::ifstream file2("file2.txt", std::ios::in);

// 3. Открытие файла для чтения и записи с trunc

std::fstream file3("file3.txt", std::ios::in | std::ios::out | std::ios::trunc);

// 4. Открытие файла для добавления

std::ofstream file4("file4.txt", std::ios::out | std::ios::app);

21. Привести примеры чтения объектов из потока.

#include <fstream>

struct Person {

std::string name;

int age;

};

int main() {

std::ifstream infile("people.txt");

Person person;

while (infile >> person.name >> person.age) {

// Используем данные

}

}

22. Привести примеры записи объектов в поток.

#include <fstream>

struct Person {

std::string name;

int age;

};

int main() {

std::ofstream outfile("people.txt");

Person person{"John", 25};

outfile << person.name << " " << person.age << std::endl;

}

23. Сформулировать алгоритм удаления записей из файла.

1. Открыть файл для чтения.

2. Считать записи в память (например, в вектор).

3. Удалить нужные записи из памяти.

4. Открыть файл для записи и перезаписать оставшиеся записи.

24. Сформулировать алгоритм добавления записей в файл.

1. Открыть файл для добавления (ios::app).

2. Записать новую запись.

25. Сформулировать алгоритм изменения записей в файле.

1. Открыть файл для чтения и чтения в память.

2. Изменить нужную запись в памяти.

3. Открыть файл для записи и перезаписать все записи.

**ПРИЛОЖЕНИЕ 1**

lab10.cpp

//-------------------------------------

#include <iostream>

#include <windows.h>

#include <filesystem>

#include "lab10\_io\_stream.hpp"

//-------------------------------------

#define IDENT\_PRINT printf("\n--------------------------\n\n")

//-------------------------------------

const char\* msg = "1.Create elements\n2.Show elements\n3.Remove elements\n4.Increase time\n5.Add elements start k\n6.Exit";

//-------------------------------------

enum Menu {

Create,

Show,

Remove,

Increase,

Add,

Exit

};

//-------------------------------------

void create(std::string& path) {

std::cout << "enter path:";

std::cin >> path;

int n;

std::cout << "enter n files:";

std::cin >> n;

std::ios\_base::openmode mode = std::ios::out;

if(std::filesystem::exists(path)) {

char yesno;

std::cout << "The file exists, should I delete the contents of the file? [y/n]:";

std::cin >> yesno;

if(yesno == 'n')

mode = std::ios::app;

}

IOTime::write(path.c\_str(), n, mode);

}

//-------------------------------------

void show(std::string& path) {

std::cout << "enter path:";

std::cin >> path;

IOTime::read(path.c\_str());

system("pause");

}

//-------------------------------------

void remove(std::string& path) {

Time time;

std::cout << "enter path:";

std::cin >> path;

std::cout << "Enter the time to delete all entries:\n";

std::cin >> time;

std::cout << "removed: " << IOTime::remove(path.c\_str(), time) << std::endl;

system("pause");

}

//-------------------------------------

void increase(std::string& path) {

Time mod,

time;

std::cout << "enter path:";

std::cin >> path;

std::cout << "time for mod:\n";

std::cin >> mod;

std::cout << "increase by:\n";

std::cin >> time;

std::cout << "mod time: " << IOTime::increase\_specified\_time(path.c\_str(), mod, time) << std::endl;

std::cout << "show\n";

IOTime::read(path.c\_str());

system("pause");

}

//-------------------------------------

void add(std::string& path) {

std::cout << "enter path:";

std::cin >> path;

int n, shift;

std::cout << "enter number elements:";

std::cin >> n;

std::cout << "enter start elements add:";

std::cin >> shift;

IOTime::add(path.c\_str(), n, shift);

std::cout << "show\n";

IOTime::read(path.c\_str());

system("pause");

}

//-------------------------------------

int main() {

bool isLoop = true;

int num = 0;

std::string path;

while(isLoop) {

std::cout << msg << std::endl;

std::cin >> num;

switch((Menu)(num-1)) {

case Menu::Create: create(path); break;

case Menu::Show: show(path); break;

case Menu::Remove: remove(path); break;

case Menu::Increase: increase(path); break;

case Menu::Add: add(path); break;

case Menu::Exit:

isLoop = false;

break;

}

system("cls");

}

return 0;

}

//-------------------------------------

lab10\_io\_stream.hpp

//-------------------------------------

#ifndef LAB10\_IO\_STREAM\_H\_INCLUDED

#define LAB10\_IO\_STREAM\_H\_INCLUDED

//-------------------------------------

#include <fstream>

#include "lab10\_time.hpp"

//-------------------------------------

namespace IOTime {

//-------------------------------------

bool write(const char\* path, size\_t n, std::ios\_base::openmode mode = std::ios::out);

//-------------------------------------

bool read(const char\* path);

//-------------------------------------

int remove(const char\* path, const Time& time);

//-------------------------------------

int increase\_specified\_time(const char\* path, const Time& cpecified, const Time& time);

//-------------------------------------

bool add(const char\* path, size\_t n, size\_t shift);

//-------------------------------------

}

//-------------------------------------

#endif // LAB10\_IO\_STREAM\_H\_INCLUDED

//-------------------------------------

lab10\_io\_stream.cpp

//-------------------------------------

#include "lab10\_io\_stream.hpp"

//-------------------------------------

void create\_time(const size\_t& n, std::fstream& temp) {

Time time;

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

std::cin >> time;

temp << time;

}

}

//-------------------------------------

void path\_editor(std::string& path) {

std::string::iterator it = path.end();

size\_t pos = 0;

while(it != path.begin()) {

if(\*it == '/' || \*it == '\\') {

pos = it - path.begin();

break;

}

--it;

}

path = pos == 0 ? "./temp" : path.substr(0, pos+1) + "temp";

}

//-------------------------------------

bool IOTime::write(const char\* path, size\_t n, std::ios\_base::openmode mode) {

std::fstream stream(path, mode);

if(!stream.is\_open())

return false;

Time time;

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

std::cin >> time;

stream << time;

}

stream.close();

return true;

}

//-------------------------------------

bool IOTime::read(const char\* path) {

std::fstream stream(path, std::ios::in);

if(!stream.is\_open())

return false;

Time time;

while(stream >> time)

std::cout << "time: " << time;

stream.close();

return true;

}

//-------------------------------------

int IOTime::remove(const char\* path, const Time& time) {

std::string tempPath = path;

path\_editor(tempPath);

std::fstream stream(path, std::ios::in),

temp(tempPath, std::ios::out);

int removed = 0;

Time buffer;

while(stream >> buffer) {

if(time == buffer) {

++removed;

continue;

}

temp << buffer;

}

temp.close();

stream.close();

if(removed == 0) {

std::remove(tempPath.c\_str());

return removed;

}

std::remove(path);

std::rename(tempPath.c\_str(), path);

return removed;

}

//-------------------------------------

int IOTime::increase\_specified\_time(const char\* path, const Time& cpecified, const Time& time) {

int edit = 0;

std::string tempPath = path;

path\_editor(tempPath);

std::fstream stream(path, std::ios::in),

temp(tempPath, std::ios::out);

Time buffer;

while(stream >> buffer) {

if(cpecified == buffer) {

buffer = buffer + time;

++edit;

}

temp << buffer;

}

temp.close();

stream.close();

if(edit == 0) {

std::remove(tempPath.c\_str());

return edit;

}

std::remove(path);

std::rename(tempPath.c\_str(), path);

return edit;

}

//-------------------------------------

bool IOTime::add(const char\* path, size\_t n, size\_t shift) {

if(n == 0)

return false;

std::string tempPath = path;

path\_editor(tempPath);

std::fstream stream(path, std::ios::in),

temp(tempPath, std::ios::out);

if(!stream.is\_open())

return false;

Time buffer;

size\_t num = 0;

bool isAdd = false;

while(stream >> buffer) {

if(num == shift) {

create\_time(n, temp);

isAdd = true;

}

temp << buffer;

++num;

}

if(!isAdd)

create\_time(n, temp);

temp.close();

stream.close();

std::remove(path);

std::rename(tempPath.c\_str(), path);

return true;

}

//-------------------------------------

lab10\_time.hpp

//-------------------------------------

#ifndef LAB10\_TIME\_HPP\_INCLUDED

#define LAB10\_TIME\_HPP\_INCLUDED

//-------------------------------------

#include <iostream>

#include <fstream>

#include <cstdint>

//-------------------------------------

#define MAX\_SECONDS 60

//-------------------------------------

class Time {

public:

Time();

Time(int minutes, int seconds);

Time(const Time& time);

~Time();

int getMinutes() { return m\_minutes; }

int getSeconds() { return m\_seconds; }

void setMinutes(int minutes) { m\_minutes = minutes; }

void setSeconds(int seconds) { m\_seconds = seconds % MAX\_SECONDS; }

void reset() { m\_minutes = 0; m\_seconds = 0; }

Time& operator=(const Time& time);

Time operator+(const Time& time);

bool operator==(const Time& time) const;

friend std::istream& operator>>(std::istream& input, Time& time);

friend std::ostream& operator<<(std::ostream& output, Time& time);

friend std::fstream& operator>>(std::fstream& input, Time& time);

private:

int m\_minutes,

m\_seconds;

};

//-------------------------------------

#endif // LAB10\_TIME\_HPP\_INCLUDED

//-------------------------------------

lab10\_time.cpp

//-------------------------------------

#include "lab10\_time.hpp"

//-------------------------------------

Time::Time() {

m\_minutes = 0;

m\_seconds = 0;

}

//-------------------------------------

Time::Time(int minutes, int seconds) {

this->setMinutes(minutes);

this->setSeconds(seconds);

}

//-------------------------------------

Time::Time(const Time& time) {

m\_minutes = time.m\_minutes;

m\_seconds = time.m\_seconds;

}

//-------------------------------------

Time::~Time() {

//

}

//-------------------------------------

Time& Time::operator=(const Time& time) {

if(this == &time)

return \*this;

m\_minutes = time.m\_minutes;

m\_seconds = time.m\_seconds;

return \*this;

}

//-------------------------------------

Time Time::operator+(const Time& time) {

uint32\_t newtime = time.m\_minutes\*MAX\_SECONDS + time.m\_seconds;

newtime += (m\_minutes\*MAX\_SECONDS + m\_seconds);

return Time(newtime/MAX\_SECONDS, newtime%MAX\_SECONDS);

}

//-------------------------------------

bool Time::operator==(const Time& time) const {

return time.m\_minutes == m\_minutes && time.m\_seconds == m\_seconds;

}

//-------------------------------------

std::istream& operator>>(std::istream& input, Time& time) {

#ifdef LOG

std::cout << "object: " << &time << std::endl;

#endif //LOG

std::cout << "minutes: ";

input >> time.m\_minutes;

std::cout << "seconds: ";

input >> time.m\_seconds;

time.m\_seconds %= MAX\_SECONDS;

return input;

}

//-------------------------------------

std::ostream& operator<<(std::ostream& output, Time& time) {

return (output <<

#ifdef LOG

"object: " << &time << "\n" <<

#endif // LOG

time.getMinutes() << ":" << (int)time.getSeconds() << "\n");

}

//-------------------------------------

std::fstream& operator>>(std::fstream& input, Time& time) {

std::string buffer;

input >> buffer;

size\_t pos = buffer.find(":");

try {

time.m\_minutes = std::stoi(buffer.substr(0, pos));

time.m\_seconds = std::stoi(buffer.substr(pos+1, buffer.size()));

} catch(const std::exception& e) { }

time.m\_seconds %= MAX\_SECONDS;

return input;

}

//-------------------------------------