Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (ПНИПУ)

Электротехнический факультет

Дисциплина: основы алгоритмизации и программирования, 2 семестр

ОТЧЁТ

«Лабораторная работа ООП № 10. Использование потоков»

Выполнил

Студент группы РИС-22-2Б

Хохряков А.С.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Пермь 2023

# Содержание отчета

1. Постановка задачи.
2. UML-Диаграмма
3. Ответы на вопросы
4. Текст программы

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 из 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.

Вариант 3

Создать класс Time для 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:

- добавление секунд (учесть, что в минуте не может быть более 60 секунд)

-вычитание секунд

Задание:

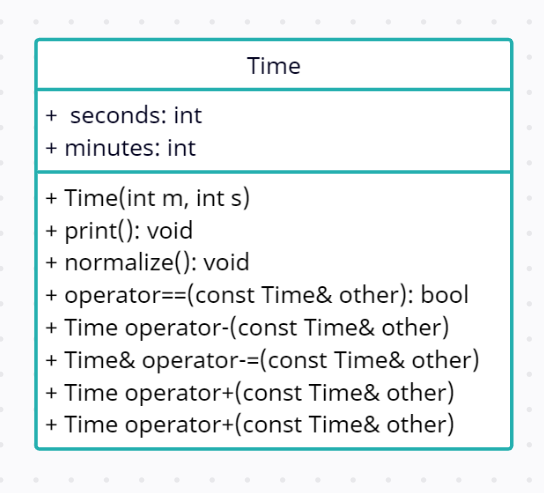
-Удалить все записи равные заданному значению.

-Уменьшить все записи с заданным значением на 1 минуту 30 секунд. Значение

интервала не должно быть меньше 0 минут 0 секунд.

-Добавить К записей после элемента с заданным значением.

UML – Диаграмма



Ответы на вопросы

1. Поток - это последовательный поток данных, который передает данные между программой и устройством ввода-вывода, например, между программой и консолью или файлом.
2. Существуют три типа потоков: ввода, вывода и ввода-вывода.
3. Для использования стандартных потоков не требуется подключать какие-либо дополнительные библиотеки, так как они уже встроены в стандартную библиотеку языка C++.
4. Для использования файловых потоков необходимо подключить библиотеку <fstream>.
5. Для использования строковых потоков необходимо подключить библиотеку <sstream>.
6. Для вывода в форматированный поток используется оператор вставки <<.
7. Для ввода из форматированных потоков используется оператор извлечения >>.
8. При выводе в форматированный поток используются методы, такие как width(), precision(), и fill().
9. При вводе из форматированного потока используются методы, такие как getline(), ignore() и peek().
10. Существуют различные режимы открытия файловых потоков, такие как ios::in, ios::out, ios::app, ios::trunc и ios::binary.
11. Режим ios::app используется для добавления записей в конец файла без удаления предыдущего содержимого.
12. Режим ios::in используется для чтения данных из файла, по умолчанию открытие файла происходит в режиме ios::in.
13. Режим ios::in | ios::out используется для одновременного чтения и записи в файл.
14. Режим ios::out используется для записи данных в файл, при этом предыдущее содержимое файла удаляется.
15. Для открытия потока в режиме ios::out | ios::app используется следующая команда:



1. Для открытия потока в режиме ios::out | ios::trunc используется следующая команда:



1. Для открытия потока в режиме ios::out | ios::in | ios::trunc используется следующая команда:



1. Для открытия файла для чтения используется команда:

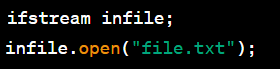


1. Для открытия файла для записи используется команда:

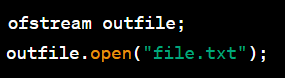


1. Примеры открытия файловых потоков в различных режимах:

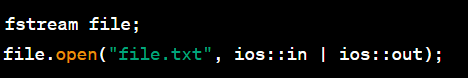
* Открытие файла для чтения:



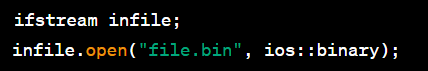
* Открытие файла для записи:



* Открытие файла для чтения и записи:

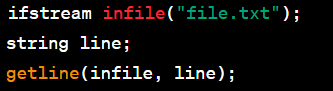


* Открытие файла в двоичном режиме:

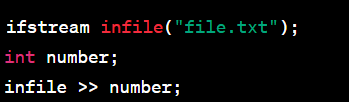


1. Примеры чтения объектов из потока:

* Чтение строки из файла:



* Чтение числа из файла:

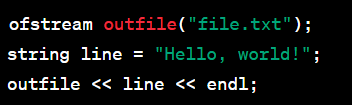


* Чтение пользовательской структуры из файла:

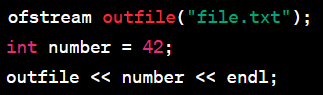


1. Примеры записи объектов в поток:

* Запись строки в файл:



* Запись числа в файл:



* Запись пользовательской структуры в файл:



1. Алгоритм удаления записей из файла:

* Открыть файл в режиме чтения и записи.
* Создать временный файл.
* Считать записи из исходного файла и записать их во временный файл, пропуская удаляемые записи.
* Закрыть исходный файл и временный файл.
* Удалить исходный файл.
* Переименовать временный файл в имя исходного файла.

1. Алгоритм добавления записей в файл:

* Открыть файл в режиме добавления.
* Записать новые записи в конец файла.
* Закрыть файл.

1. Алгоритм изменения записей в файле:

* Открыть файл в режиме чтения и записи.
* Создать временный файл.
* Считать записи из исходного файла, изменять необходимые записи и записывать их во временный файл.
* Закрыть исходный файл и временный файл.
* Удалить исходный файл.
* Переименовать временный файл в имя исходного файла.

Текст программы

#include <iostream>

using namespace std;

class Time {

public:

int minutes;

int seconds;

Time(int m, int s) {

minutes = m;

seconds = s;

}

void print() {

cout << minutes << ":" << seconds << endl;

}

void normalize() {

if (seconds >= 60) {

minutes += seconds / 60;

seconds %= 60;

}

if (seconds < 0) {

minutes += (seconds - 59) / 60;

seconds = 60 + (seconds % 60);

}

if (minutes < 0) {

minutes = 0;

seconds = 0;

}

}

bool operator==(const Time& other) const {

return minutes == other.minutes && seconds == other.seconds;

}

Time operator-(const Time& other) const {

int s = seconds - other.seconds;

int m = minutes - other.minutes;

return { m, s };

}

Time& operator-=(const Time& other) {

minutes -= other.minutes;

seconds -= other.seconds;

normalize();

return \*this;

}

Time operator+(const Time& other) const {

int s = seconds + other.seconds;

int m = minutes + other.minutes;

return { m, s };

}

Time& operator+=(const Time& other) {

minutes += other.minutes;

seconds += other.seconds;

normalize();

return \*this;

}

};

void deleteEntries(Time entries[], int& count, Time target) {

int i = 0;

while (i < count) {

if (entries[i] == target) {

for (int j = i; j < count - 1; j++) {

entries[j] = entries[j + 1];

}

count--;

}

else {

i++;

}

}

}

void decreaseEntries(Time entries[], int count, Time target) {

for (int i = 0; i < count; i++) {

if (entries[i] == target) {

entries[i] -= {1, 30}; // Вычитание интервала {1, 30}

entries[i].normalize(); // Нормализация интервала

}

}

}

void insertEntries(Time entries[], int& count, Time target, Time newEntries[], int newCount) {

for (int i = 0; i < count; i++) {

if (entries[i] == target) {

// Расширение массива на количество новых элементов

for (int j = count - 1; j >= i + 1; j--) {

entries[j + newCount] = entries[j];

}

// Вставка новых элементов

for (int j = 0; j < newCount; j++) {

entries[i + j + 1] = newEntries[j];

}

count += newCount;

i += newCount; // Пропускаем вставленные элементы

}

}

}

int main() {

const int MAX\_ENTRIES = 10;

Time entries[MAX\_ENTRIES] = { {2, 15}, {0, 45}, {1, 30}, {2, 0}, {1, 45}, {0, 30}, {1, 0}, {2, 45}, {0, 15}, {2, 30} };

int count = 10;

cout << "Initial entries: " << endl;

for (int i = 0; i < count; i++) {

entries[i].print();

}

Time target = { 1, 30 };

cout << endl << "Deleting entries equal to ";

target.print();

deleteEntries(entries, count, target);

cout << "Entries after deletion: " << endl;

for (int i = 0; i < count; i++) {

entries[i].print();

}

target = { 2, 0 };

cout << endl << "Decreasing entries equal to ";

target.print();

decreaseEntries(entries, count, target);

cout << "Entries after decrease: " << endl;

for (int i = 0; i < count; i++) {

entries[i].print();

}

return 0;

}