Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

Семестр: 2

На тему: «Сохранение данных в файле с использованием потоков»

Выполнил студент ИВТ-22-2б:

Шилов Валерий Михайлович

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

Пермь 2023

**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.

**Описание класса**

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:

− вычитание временных интервалов (учесть, что в минуте не может быть более 60 секунд)

− сравнение временных интервалов (!=)

Задание:

• Удалить все записи не равные заданному значению.

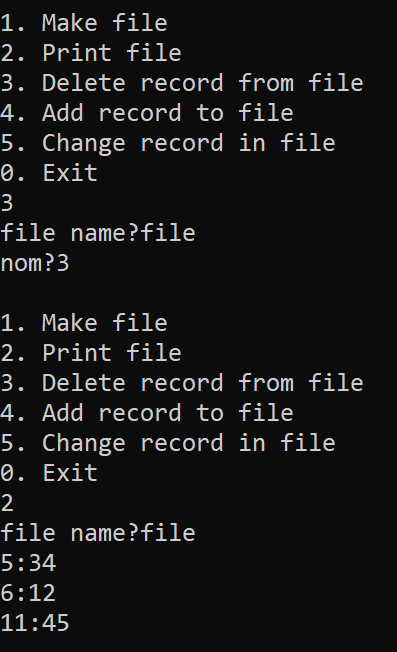
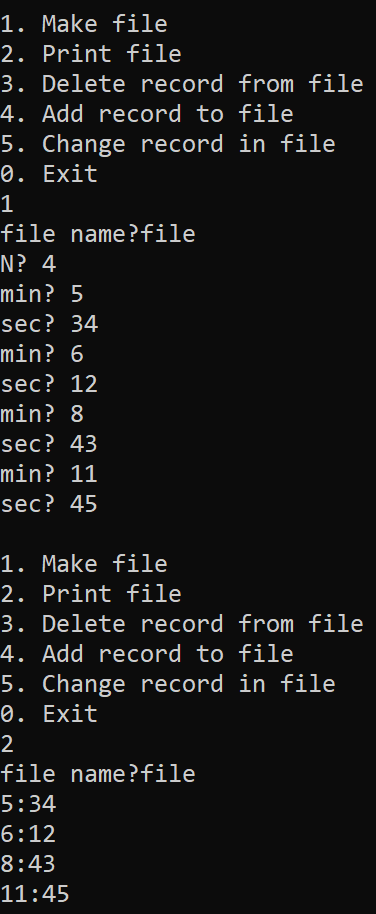
• Уменьшить все записи с заданным значением на 1 минуту 30 секунд. Значение интервала не должно быть меньше 0 минут 0 секунд.

• Добавить K записей в начало файла.

**UML-диаграмма**

****

Рисунок 1 – UML-диаграмма

**Результат работы программы**

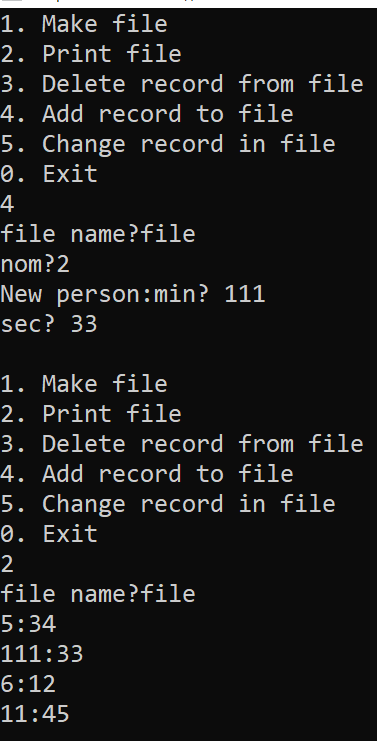
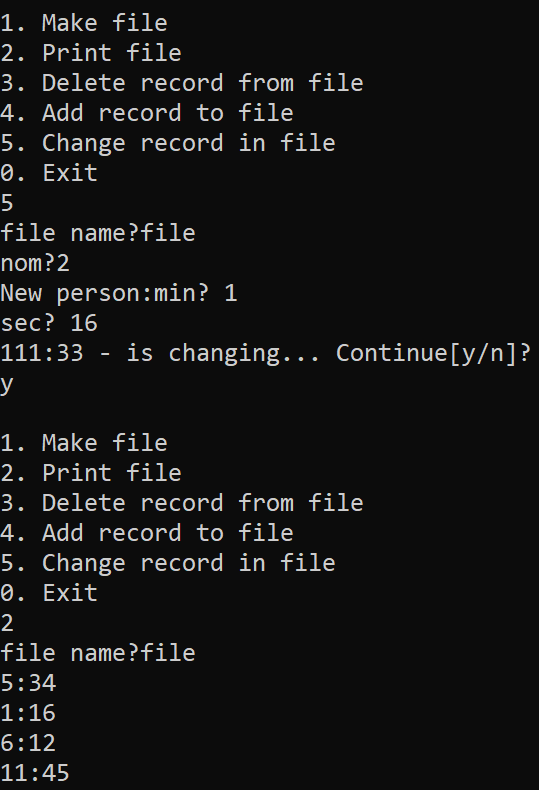


Рисунок 2 – результат работы программы

**Контрольные вопросы**

1. Что такое поток?

Поток - определяется как последовательность байтов и не зависит от конкретного устройства, с которым производится обмен (оперативная память, файл на диске, клавиатура или принтер). Обмен с потоком для увеличения скорости передачи данных производится, как правило, через специальную область оперативной памяти — буфер. Буфер накапливает байты, и фактическая передача данных выполняется после заполнения буфера. При вводе это дает возможность исправить ошибки, если данные из буфера еще не отправлены в программу.

2. Какие типы потоков существуют?

Потоки бывают

- Стандартные: только однонаправленные, либо входные, либо выходные.

- Строковые: могут быть и однонаправленными и двунаправленными

- Файловые: могут быть и однонаправленными и двунаправленными.

Стандартные потоки обозначаются стандартными именами. Эти имена привязаны к стандартным устройствам: клавиатуре и экрану; стандартные потоки можно перенаправить на другие устройства, например, на файл на диске.

Потоки других видов надо объявлять как переменные соответствующего типа. Переменная файлового потока связывается со стандартным файлом на диске.

Стандартные потоки - форматируемые, строковые тоже. Файловые потоки могут быть и форматируемыми, и не форматируемыми.

3. Какую библиотеку надо подключить при использовании стандартных потоков?

Для использования стандартных потоков надо задать в программе директиву

#include <iostream>

В заголовочном файле iostream содержатся описания классов ввода/вывода и четыре стандартных системных объекта:

cin – объект класса istream, по умолчанию связан с клавиатурой. cout- объект класса ostream, по умолчанию связан с экраном.

clog - объект класса ostream, соответствующий стандартному выводу для ошибок, по умолчанию связан с экраном.

cerr - объект класса ostream, соответствующий стандартному выводу для ошибок, по умолчанию связан с экраном.

Объект cout используется для обычного вывода, а clog и cerr для вывода сообщений об ошибок. Используются он также, как и cout. Специально объявлять в программе стандартные потоки не требуется.

4. Какую библиотеку надо подключить при использовании файловых потоков?

Для использования файловых потоков надо задать в программе директиву

#include <fstream>

После этого в программе можно объявлять объекты – файловые потоки трех типов:

- входной ifstream;

- выходной ofstream;

- двунаправленный fstream;

5. Какую библиотеку надо подключить при использовании строковых потоков?

Для использования строковых потоков надо задать в программе директиву

#include <sstream>

После этого в программе можно объявлять объекты – строковые потоки трех типов:

- входной istringstream;

- выходной ostringstream;

- двунаправленный stringstream;

6. Какая операция используется при выводе в форматированный поток?

Для форматируемых потоков вывод, как правило, осуществляется перегруженной операцией сдвига влево operator<<.

stream<<3.4;stream<<’\n’;//вывод константы

7. Какая операция используется при вводе из форматированных потоков?

Для форматируемых потоков ввод перегруженной операцией сдвига вправо operator>>.

stream>>a>>b;

Ввод в переменную завершается, если очередной символ в потоке не соответствует типу вводимого значения. Стандартными разделителями в потоке являются: ‘ ‘(пробел), ‘\n’, ‘\t’.

8. Какие методы используются при выводе в форматированный поток?

Для вывода символов можно использовать методы:



Методы вывода возвращают ссылку на поток. Их можно соединять в одно выражение-оператор.

stream.write(s,strlen(s)).put(‘\n);

Вывод из двоичных файлов (режим binary) выполняется методом write:

ostream& write(const char\* buf, streamsize size)

Метод записывает size символов символьного массива в поток данных. Символы разделители на вывод не влияют.

9. Какие методы используется при вводе из форматированного потока?

Поместить в символьную переменную любой символ (в том числе и символ разделитель) можно с помощью методов



Ввод символьных массивов и строк выполняется с помощью операции >> до первого символа-разделителя (обычно пробела). Для ввод строк с пробелами используют методы get() и getline().



Ввод в двоичные файлы производится методом read:

istream& read(char\*buf, streamsize size)

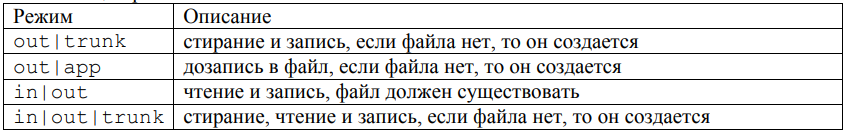
Метод читает size символов в массив buf. Символы разделители на ввод не влияют.

10. Какие режимы для открытия файловых потоков существуют?

Режимы открытия потока:



Комбинации режимов:



11. Какой режим используется для добавления записей в файл?

Для добавления записи в конец файла используется режим app.

Также, используется режим ate для чтения и/или записи в конец файла.

12. Какой режим (комбинация режимов) используется в конструкторе ifstream file(“f.txt”)?

Используется режим in - открыть поток для чтения (по умолчанию для ifstream).

13. Какой режим (комбинация режимов) используется в конструкторе fstream file(“f.txt”)?

Используется комбинация режимов in | out - чтение и запись, файл должен существовать.

14. Какой режим (комбинация режимов) используется в конструкторе ofstream file(“f.txt”)?

Используется режим out - открыть поток для записи в файл (по умолчанию для ofstream).

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

Поток открывается для дозапись в файл, если файла нет, то он создаётся.

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

Поток открывается для стирания и записи, если файла нет, то он создаётся.

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

Поток открывается для стирания, чтения и записи, если файла нет, то он создаётся.

18. Каким образом можно открыть файл для чтения?

Файл может быть открыт либо явно методом open(), либо неявно – конструктором при создании потока.

Операции обмена между файлом и программой зависят от типа связываемого с файлом потока.

Для открытия файла для чтения можно использовать различные режимы:

in - открыть поток для чтения (по умолчанию для ifstream);

ate - открыть поток для чтения и/или записи и встать в конец файла.

Также, можно комбинировать режимы:

in|out - чтение и запись, файл должен существовать;

in|out|trunk - стирание, чтение и запись, если файла нет, то он создается.

19. Каким образом можно открыть файл для записи?

Файл может быть открыт либо явно методом open(), либо неявно – конструктором при создании потока.

Операции обмена между файлом и программой зависят от типа связываемого с файлом потока.

Для открытия файла для записи можно использовать различные режимы:

out - открыть поток для записи в файл (по умолчанию для ofstream);

trunk - удалить старое содержимое файла (по умолчанию для оfstream);

app - открыть поток для записи в конец файла

ate - открыть поток для чтения и/или записи и встать в конец файла

Также, можно комбинировать режимы:

out|trunk - стирание и запись, если файла нет, то он создается;

out|app - дозапись в файл, если файла нет, то он создается;

in|out|trunk - стирание, чтение и запись, если файла нет, то он создается

20. Привести примеры открытия файловых потоков в различных режимах.

Пример 1.

В данном примере файл "example.txt" открывается для записи с помощью конструктора класса ofstream. Если файл успешно открыт, то в него записывается строка "Hello, world!". После окончания работы с файлом его необходимо закрыть с помощью метода close(). Если файл не удалось открыть, то выводится сообщение об ошибке.

#include <iostream>

#include <fstream>

int main() {

std::ofstream file("example.txt");

if (file.is\_open()) {

file << "Hello, world!" << std::endl;

file.close();

}

else {

std::cout << "Unable to open file" << std::endl;

}

return 0;

}

Пример 2:

В данном примере файл "example.bin" открывается для чтения в двоичном режиме с помощью конструктора класса ifstream и флага std::ios::binary. Если файл успешно открыт, то его данные читаются в буфер с помощью метода read(). После окончания работы с файлом его необходимо закрыть с помощью метода close(). Если файл не удалось открыть, то выводится сообщение об ошибке.

#include <iostream>

#include <fstream>

int main() {

std::ifstream file("example.bin", std::ios::binary);

if (file.is\_open()) {

// читаем данные из файла в буфер

char buffer[256];

file.read(buffer, sizeof(buffer));

// обрабатываем данные из буфера

file.close();

}

else {

std::cout << "Unable to open file" << std::endl;

}

return 0;

}

21. Привести примеры чтения объектов из потока.

Пример 1 (чтения строки из файла).

В данном примере файл "example.txt" открывается для чтения с помощью конструктора класса ifstream. Если файл успешно открыт, то его содержимое читается построчно с помощью функции getline(). Каждая прочитанная строка выводится на экран. После окончания работы с файлом его необходимо закрыть с помощью метода close(). Если файл не удалось открыть, то выводится сообщение об ошибке.

#include <iostream>

#include <fstream>

#include <string>

int main() {

std::ifstream file("example.txt");

if (file.is\_open()) {

std::string line;

while (std::getline(file, line)) {

std::cout << line << std::endl;

}

file.close();

}

else {

std::cout << "Unable to open file" << std::endl;

}

return 0;

}

Пример 2 (чтения числа из файла).

В данном примере файл "example.bin" открывается для чтения в двоичном режиме с помощью конструктора класса ifstream и флага std::ios::binary. Если файл успешно открыт, то из него читается целое число с помощью метода read(). Для корректного чтения числа необходимо использовать reinterpret\_cast для приведения указателя на буфер к типу char\*. После чтения числа оно выводится на экран. После окончания работы с файлом его необходимо закрыть с помощью метода close(). Если файл не удалось открыть, то выводится сообщение об ошибке.

#include <iostream>

#include <fstream>

int main() {

std::ifstream file("example.bin", std::ios::binary);

if (file.is\_open()) {

int number;

file.read(reinterpret\_cast<char\*>(&number), sizeof(number));

std::cout << "Number: " << number << std::endl;

file.close();

}

else {

std::cout << "Unable to open file" << std::endl;

}

return 0;

}

22. Привести примеры записи объектов в поток.

Пример записи объекта в текстовый файл:

#include <iostream>

#include <fstream>

#include <string>

class Person {

public:

Person(std::string name, int age) : m\_name(name), m\_age(age) {}

std::string getName() const { return m\_name; }

int getAge() const { return m\_age; }

private:

std::string m\_name;

int m\_age;

};

int main() {

std::ofstream file("example.txt");

if (file.is\_open()) {

Person person("John", 30);

file << person.getName() << " " << person.getAge() << std::endl;

file.close();

}

else {

std::cout << "Unable to open file" << std::endl;

}

return 0;

}

В данном примере создается класс Person, который содержит имя и возраст человека. Затем создается объект класса Person и его данные записываются в текстовый файл с помощью оператора <<.

Пример записи объекта в двоичный файл:

#include <iostream>

#include <fstream>

#include <string>

class Person {

public:

Person(std::string name, int age) : m\_name(name), m\_age(age) {}

std::string getName() const { return m\_name; }

int getAge() const { return m\_age; }

private:

std::string m\_name;

int m\_age;

};

int main() {

std::ofstream file("example.bin", std::ios::binary);

if (file.is\_open()) {

Person person("John", 30);

file.write(reinterpret\_cast<const char\*>(&person), sizeof(person));

file.close();

}

else

std::cout << "Unable to open file" << std::endl;

return 0;

}

В данном примере создается класс Person, который содержит имя и возраст человека. Затем создается объект класса Person и его данные записываются в двоичный файл с помощью метода write(). Для корректной записи объекта необходимо использовать reinterpret\_cast для приведения указателя на буфер к типу const char\*. После записи объекта файл закрывается с помощью метода close().

23. Сформулировать алгоритм удаления записей из файла.

Алгоритм удаления записей из файла в С++ может быть следующим:

1. Открыть файл для чтения и записи.

2. Создать временный файл.

3. Прочитать записи из исходного файла и проверять каждую запись на соответствие критерию удаления.

4. Записать нужные записи во временный файл.

5. Закрыть исходный файл и временный файл.

6. Удалить исходный файл с помощью функции.

7. Переименовать временный файл в имя исходного файла.

24. Сформулировать алгоритм добавления записей в файл.

1. Открыть файл для записи с помощью функции fstream::open().

2. Проверить, что файл успешно открыт с помощью функции fstream::is\_open().

3. Создать объект записи, который будет содержать данные для записи в файл.

4. Заполнить объект записи данными.

5. Записать объект записи в файл с помощью оператора << или функции fstream::write().

6. Проверить, что запись прошла успешно с помощью функции.

7. Повторять шаги 3-6 для каждой новой записи.

8. Закрыть файл с помощью функции fstream::close().

Пример:

#include <fstream>

#include <string>

struct Product {

std::string name;

int price;

int quantity;

};

int main() {

std::ofstream file("products.csv");

if (!file.is\_open()) {

return 1;

}

file << "Название товара,Цена,Количество\n";

Product milk{"Молоко", 50, 10};

file << milk.name << "," << milk.price << "," << milk.quantity << "\n";

Product bread{"Хлеб", 25, 20};

file << bread.name << "," << bread.price << "," << bread.quantity << "\n";

Product apples{"Яблоки", 30, 15};

file << apples.name << "," << apples.price << "," << apples.quantity << "\n";

file.close();

return 0;

}

25. Сформулировать алгоритм изменения записей в файле.

1. Открыть файл для чтения и записи с помощью функции std::fstream.

2. Проверить, что файл успешно открыт.

3. Считать данные из файла в структуру или массив данных.

4. Изменить необходимые данные в структуре или массиве.

5. Перезаписать измененные данные в файл.

6. Закрыть файл.