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: Лабораторная работа №10: «Сохранение данных в файле с использованием потоков».

**Вариант 13**

Выполнил студент ИВТ-22-2б:

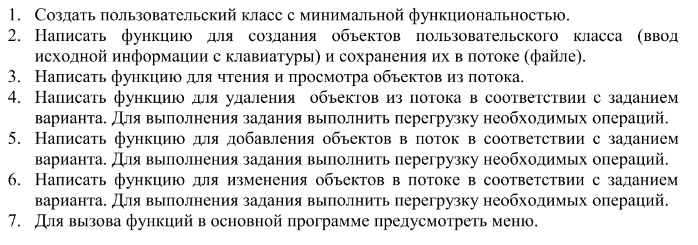
Коняев Александр Сергеевич

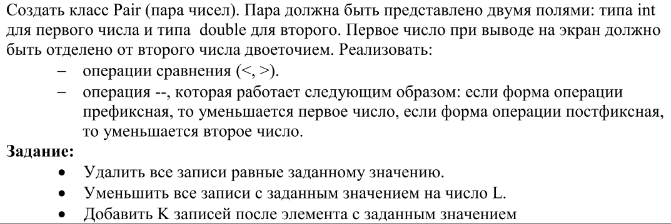
Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

Пермь 2023

**Постановка задачи**





**Анализ задачи**

Класс Pair

* Конструктор без параметров
* Конструктор с параметрами
* Конструктор копирования
* Метод get\_first
* Метод set\_first
* Метод get\_second
* Метод set\_second
* Перегруженные операторы: =, <, >, --, <<, >>

Класс LinkedList

* Конструктор с параметрами
* Конструктор без параметров
* Конструктор копирования
* Метод clear
* Метод push\_back
* Метод push\_front
* Метод pop\_back
* Метод pop\_front
* Метод get\_At
* Метод insert
* Метод erase
* Метод print
* Перегруженные операторы: =, [], +, <<, >>

Функции

* make\_file
* print\_file
* del\_file
* add\_file
* add\_end
* change\_file

**UML – диаграмма**

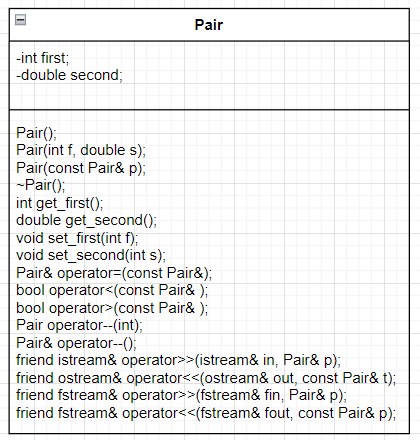


Рисунок 1 – UML-диаграмма.

**Код программы.**

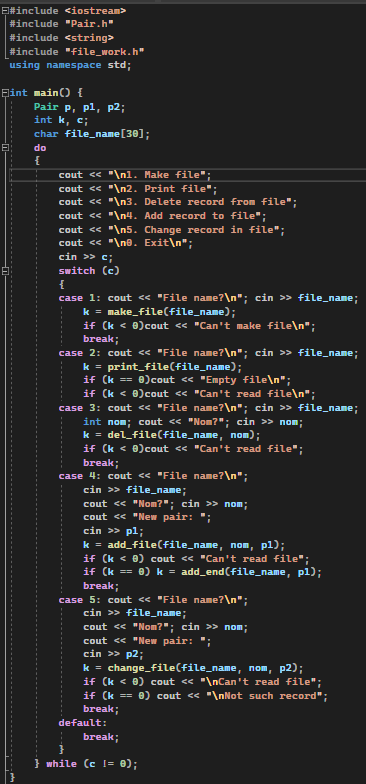


Рисунок 2 – Функция main

**Вывод программы.**

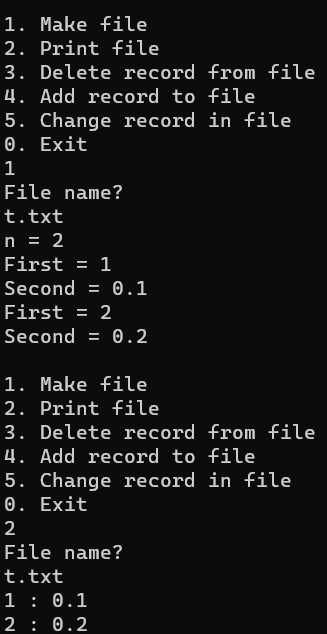
****

Рисунок 3 – Вывод программы

**Ответы на вопросы**

1. Поток в C++ - это абстракция, которая отображает последовательность байтов, позволяя программистам считывать данные из источников, таких как консоль, файлы или сеть, и записывать данные в целевые назначения.

По умолчанию C++ предоставляет три важных потока: cout, cin и cerr, которые соответственно представляют стандартный вывод, стандартный ввод и стандартный поток ошибок.

Для работы с потоками в C++ используются объекты классов std::ostream и std::istream. Потоки могут быть связаны с файлами, строками, устройствами и т.д.

Потоки в C++ можно использовать для форматированного ввода/вывода данных, двоичного ввода/вывода, а также для работы с буферизацией данных в памяти или на диске. потоки C++ предоставляют множество методов, позволяющих работать с потоками и манипулировать данными в них.

1. В C++ существуют два основных типа потоков:

Input Stream - поток для считывания данных из источника (например, std::cin для стандартного ввода или std::ifstream для файла)

Output Stream - поток для записи данных в целевое назначение (например, std::cout для стандартного вывода или std::ofstream для записи в файл).

Кроме того, существуют также и другие типы потоков:

Error Stream - поток для вывода сообщений об ошибках (например, std::cerr)

String Stream - поток, который хранит данные в строке в памяти (например, std::stringstream)

File Stream - поток, который работает с файлами (например, std::fstream для чтения и записи файлов).

Memory Stream - поток, который работает с буфером памяти (например, std::stringstream для работы с std::string в памяти).

Все эти потоки являются объектами классов из стандартной библиотеки C++ и могут использоваться для работы с данными из различных источников.

1. Для использования стандартных потоков в C++ необходимо подключить библиотеку iostream, которая содержит определения классов std::istream и std::ostream для работы с входными и выходными потоками соответственно.
2. Для работы с файловыми потоками в C++ необходимо подключить библиотеку fstream, которая определяет классы std::ifstream и std::ofstream для работы с потоками ввода-вывода из файлов.
3. При использовании строковых потоков в C++ необходимо подключить библиотеку sstream, которая определяет классы std::istringstream и std::ostringstream для работы со потоками ввода-вывода из строк.
4. Для вывода в форматированный поток в C++ используется оператор вставки << (двойное знакоменье "меньше"). Этот оператор позволяет "вставить" значение любого типа данных (числового, строкового и т.д.) в поток, который может быть направлен в стандартный вывод, файл или строку.
5. При вводе из форматированных потоков в языке C++ используется оператор извлечения >> (двойное знакоменье "больше"). Этот оператор позволяет извлекать данные разных типов из потока и присваивать их переменным.
6. При выводе в форматированный поток в языке C++ используются методы manipulator. Они представляют собой функции, которые изменяют формат вывода потока. Например:

std::setw(n) - устанавливает ширину поля вывода в n символов (если текст занимает меньше места, оставшееся место заполняется пробелами);

std::setprecision(n) - устанавливает количество знаков после запятой в числах с плавающей точкой (по умолчанию - 6);

std::fixed - выводит числа с фиксированной точкой (т.е. количество знаков после запятой будет постоянным);

std::scientific - выводит числа в экспоненциальной форме;

std::left - выравнивание по левому краю поля вывода;

std::right - выравнивание по правому краю поля вывода;

std::setfill(c) - устанавливает символ заполнения поля вывода (по умолчанию - пробел).

1. При вводе из форматированного потока в языке C++ используются следующие методы manipulator:

std::setw(n) - устанавливает минимальное количество символов, которые должны быть прочитаны из потока; std::setprecision(n) - устанавливает количество знаков после запятой в числах с плавающей точкой, которые будут считаны из потока; std::fixed - устанавливает вывод чисел с фиксированной точкой в потоке; std::scientific - устанавливает вывод чисел в экспоненциальной форме в потоке; std::skipws - пропускает все пробельные символы в потоке; std::left - выравнивает данные по левому краю поля в потоке; std::right - выравнивает данные по правому краю поля в потоке; std::setfill(c) - устанавливает символ заполнения поля в потоке.

Например, чтобы считать из потока строку, содержащую 10 символов, можно использовать следующий код:

std::string str; std::cin >> std::setw(10) >> str;

1. В языке C++ существуют следующие режимы для открытия файловых потоков:

std::ios::in - открытие файла для чтения

std::ios::out - открытие файла для записи

std::ios::binary - открытие файла в двоичном формате

std::ios::ate - установка указателя в конец файла при его открытии

std::ios::app - открытие файла в режиме добавления (данные будут добавляться в конец файла)

std::ios::trunc - удаление содержимого файла при его открытии в режиме записи

std::ios::in | std::ios::out - открытие файла для чтения и записи одновременно

1. Режим для добавления записей в файл - это std::ios::app. При открытии файла в этом режиме, все данные будут записываться в конец файла, не удаляя предыдущее содержимое. Вот пример использования:

std::ofstream file("example.txt", std::ios::app);

Это открывает файл "example.txt" в режиме добавления записей.

1. В конструкторе ifstream file("f.txt") используется режим по умолчанию ios::in, который открывает файл только для чтения. Если нужно открыть файл на запись, то нужно добавить соответствующий режим ios::out или ios::app.
2. В конструкторе fstream file("f.txt") используется режим по умолчанию ios::in | ios::out, который открывает файл как для чтения, так и для записи. Если файл не существует, то он будет создан. Если файл существует, то его содержимое будет сначала прочитано, а затем можно производить как чтение, так и запись.
3. В конструкторе ofstream file("f.txt") используется режим по умолчанию ios::out, который открывает файл только для записи. Если файл не существует, то он будет создан. Если файл уже существует, то его содержимое будет удалено, а затем можно производить запись в файл.
4. Для открытия потока в режиме ios::out|ios::app необходимо в конструкторе объекта ofstream указать два флага: ios::out и ios::app. Флаг ios::out указывает на открытие файла только для записи, а ios::app на то, что запись будет производиться в конец файла, без удаления содержимого, если оно уже существует.
5. Для открытия потока в режиме ios::out|ios::trunc необходимо в конструкторе объекта ofstream указать два флага: ios::out и ios::trunc. Флаг ios::out указывает на открытие файла только для записи, а ios::trunc на то, что при открытии файла его содержимое будет удалено.
6. Для открытия потока в режиме ios::out | ios::in | ios::trunc необходимо в конструкторе объекта fstream указать три флага: ios::out, ios::in и ios::trunc. Флаг ios::out указывает на открытие файла только для записи, ios::in - на открытие для чтения, а ios::trunc на то, что при открытии файла его содержимое будет удалено.
7. Для открытия файла для чтения необходимо передать в конструктор объекта fstream один флаг ios::in. Пример кода:

std::fstream file("filename.txt", std::ios::in);

if (file.is\_open()) {

// чтение из файла

file.close();

} else {

// обработка ошибки открытия файла

}

1. Для открытия файла для записи необходимо передать в конструктор объекта fstream флаги ios::out и/или ios::app в зависимости от того, нужно ли перезаписывать файл или добавлять новые данные. Пример кода:

std::fstream file("filename.txt", std::ios::out);

if (file.is\_open()) {

// запись в файл

file.close();

} else {

// обработка ошибки открытия файла

}

1. Открытие файла для записи, существующий файл будет перезаписан:

std::fstream file("filename.txt", std::ios::out);

if (file.is\_open()) {

// Запись в файл

file.close();

} else {

// Обработка ошибки открытия файла

}

Открытие файла для добавления данных в конец файла:

std::fstream file("filename.txt", std::ios::app);

if (file.is\_open()) {

// Добавление данных в конец файла

file.close();

} else {

// Обработка ошибки открытия файла

}

Открытие файла для чтения:

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

if (file.is\_open()) {

// Чтение данных из файла

file.close();

} else {

// Обработка ошибки открытия файла

}

Открытие файла для чтения и записи:

std::fstream file("filename.txt", std::ios::in | std::ios::out);

if (file.is\_open()) {

// Чтение и запись данных в файл

file.close();

} else {

// Обработка ошибки открытия файла

}

1. Конкретный пример чтения объекта зависит от типа объекта и способа его записи в поток. Однако в общем случае, для чтения объекта из потока можно использовать операторы извлечения данных (<<) и методы класса потока. Например, если объект был записан в поток с помощью оператора вставки (>>), его можно прочитать следующим образом:

#include <iostream>

#include <fstream>

#include <string>

struct Person {

std::string name;

int age;

};

std::ostream& operator<<(std::ostream& os, const Person& p) {

os << p.name << " " << p.age;

return os;

}

std::istream& operator>>(std::istream& is, Person& p) {

is >> p.name >> p.age;

return is;

}

int main() {

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

out << Person{"Alice", 25} << "\n";

out << Person{"Bob", 30} << "\n";

out.close();

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

if (in.is\_open()) {

Person p1, p2;

in >> p1 >> p2;

std::cout << p1.name << " " << p1.age << "\n";

std::cout << p2.name << " " << p2.age << "\n";

in.close();

} else {

std::cout << "Error opening file

";

}

return 0;

}

В этом примере создается структура Person, которая имеет поля name и age. Затем определяются операторы вставки и извлечения, которые позволяют записывать и читать объекты Person из потока. В функции main создается файл "people.txt", в который записываются два объекта Person. Затем файл открывается для чтения, и из него извлекаются объекты, которые выводятся на экран.

Важно заметить, что в этом примере объекты Person не были записаны и прочитаны как бинарные данные, а были представлены в текстовом виде с помощью операторов вставки и извлечения. Если требуется записывать или читать бинарные данные, используйте другие методы класса потока, такие как write и read.

1. Примеры записи объектов в поток:

Запись базовых типов данных:

int x = 10;

double y = 3.14;

std::ofstream out("data.txt");

out << x << " " << y << "\n";

out.close();

Запись пользовательского класса:

class Person {

public:

std::string name;

int age;

Person(std::string n, int a) : name(n), age(a) {}

};

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

Person p1("Alice", 25);

out.write((char\*)&p1, sizeof(Person));

out.close();

Запись в контейнер (вектор):

std::vector<int> vec{1, 2, 3, 4, 5};

std::ofstream out("data.txt");

out.write((char\*)&vec[0], vec.size() \* sizeof(int));

out.close();

Запись объектов в поток может происходить как в текстовом, так и в бинарном формате. Важно использовать соответствующие методы записи, чтобы получить ожидаемый результат.

1. Открыть файл в режиме чтения и записи.

Читать записи из файла, пока не будет достигнут конец файла, и проверять каждую запись на соответствие условию, по которому необходимо удалить запись.

Если запись соответствует условию, то продолжить чтение следующей записи.

Если запись не соответствует условию, то записать её во временный файл или в буфер памяти и продолжить чтение следующей записи.

После окончания чтения записей из файла, закрыть файл, открыть его заново в режиме записи и перезаписать его оставшимися записями из временного файла или из буфера памяти.

Если записей в файле не осталось, удалить файл или создать пустой файл вместо него.

1. Открыть файл в режиме добавления (append).

Проверить, существует ли файл. Если нет, то создать новый файл.

Подготовить данные записи.

Записать данные в конец файла.

Закрыть файл.

1. Открыть файл в режиме чтения и записи (r+).

Используя функцию чтения, найти нужную запись в файле.

Если запись была найдена, то обновить данные этой записи.

Если запись не была найдена, вывести сообщение об ошибке.

Записать измененные данные в файл.

Закрыть файл.