**Оглавление**

**Введение**5

**Локализация и настройка консоли**6

**Описание функций** 7

**Классы, структуры и методы** 11

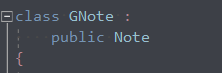
**Тестирование программы** 13

**Заключение**16

**Список литературы** 17

**Введение**

Целью курсовой работы является реализации программы для работ с заметками (создание, удаление, фильтрация, поиск, сохранение в файл и чтение из него) используя классы и методы. Для реализации был выбран язык программирования С++ в программной среде MS Visual Studio. Для написании проекта был выбран интерфейс консольного приложения. Для реализации заметок есть класс **Note** и его дочерние классы: **ANote**, **GNote**и **TNote.**



**Локализация и настройка консоли**

Программа разбита на семь файлов расширения “.cpp” и шесть заголовочных для удобства чтения и редактирования.

К сожалению сразу вывод данных в консоль на языке C++ позволяет выводить только английский текст. Для вывода информации на русском языке необходимо настроить некоторые параметры программы.

В основном файле “Note editor.cpp” производим процесс локализации консоли:

**SetConsoleCP(1251);** - задаем кодировку для вывода символов на экран

**SetConsoleOutputCP(1251);** - задаем кодировку для ввода символов с клавиатуры в консоль

После этого программа инициализирует структуры, в которых будут храниться данные заметок. После этого идёт главный цикл, который запускает меню выбора действий и с помощью оператора “**switch**”, он получает на вход команду, которую ввёл пользователь и вызывает нужную функцию.

Когда пользователь выбрал функцию выхода из программы, вызываются команды, очищающие консоль и удаляющие структуры, которые мы создавали. Потом программа выводит сообщение, что работа завершена, и останавливает консоль, чтобы она не закрылась.

**Описание функций**

Файл “Functions.cpp” самый большой, он хранит в себе практически все функции в программе. Первые ≈ 400 строк отвечают за меню.

Начнём с функции вывода заметок

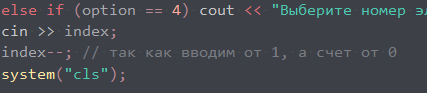
**voidPrint()** принимает за аргументы структуры для хранения заметок, выводит пользователю меню, где он выбирает какого типа заметки хочет увидеть.

После этого, консоль очищается, с помощью оператора **switch** функция выводит все заметки одного типа. После чего возвращает пользователя в главное меню.

**voidAdd()** тоже принимает за аргументы структуры для хранения заметок, выводит меню, проверяет правильность ввода команд, создаёт временные переменные, после чего просит пользователя ввести данные заметки и метки, очищает консоль и создаёт заметку.

**voidEdit()**тоже выводит меню, проверяет ввод, создаёт временные переменные, но потом просит ввести номер заметки (который можно узнать если вывести все заметки одного типа), проверяет ввод, очищает консоль и спрашивает пользователя что он хочет изменить, после чего просит ввести дату редактирования и настроить метки. После всего этого функция редактирует заметку.

**voidDel()** очень похожа на **voidEdit():** тут мы тоже работает с номером заметки, тоже уменьшаем его на 1 потому что считает программа от нуля:



Но в этот раз удаляем заметку.

**voidSearch()** и **voidFiltration()** выполняют поиск/фильтрацию заметок. Сами функции описаны в отдельных файлах для каждого типа заметки, о которых будет написано ниже.

**voidSave()** отвечает за сохранение заметок в текстовой файл. С помощью функции:

**ofstreamsave(fname, ios::out);**Чтобы использовать эти классы, в текст программы необходимо включить дополнительный заголовочный файл **fstream.h**. После этого в программе можно определять конкретные файловые потоки, соответствующих типов.

Программа создаёт поток для записи информации в текстовой файл, название которого пользователь ввёл. Такой файл показан на рис.1:

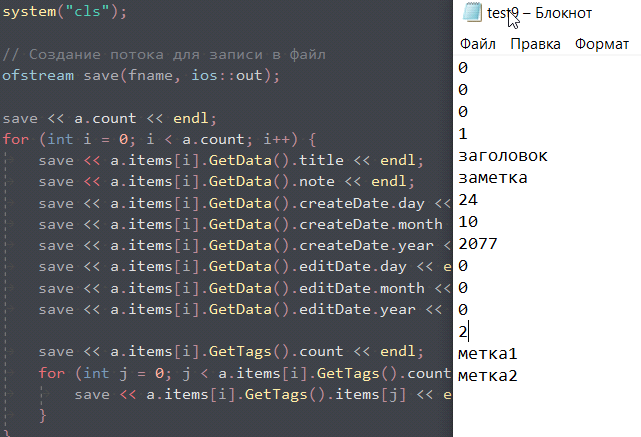
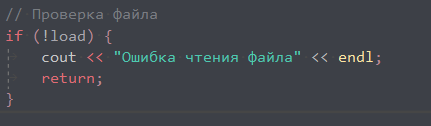


Рис.1

Создание файлового потока (объекта соответствующего класса) связывает имя потока с выделяемым для него буфером и инициализирует переменные состояния потока.Открытие файла в самом общем смысле означает процедуру, информирующую систему о тех действиях, которые предполагается выполнять с файлом. Существуют функции стандартной библиотеки языка С для открытия файлов fopen(), open(). Но работая с файловыми потоками библиотеки ввода-вывода языка С++, удобнее пользоваться компонентными функциями соответствующих классов.

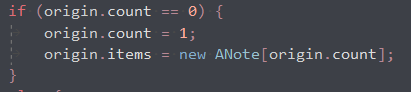
**voidLoad()** загружает в программу заметки из файла, нужно просто поместить файл в папку программы и в ней самой ввести его название. Если перед выходом из программы не сохранить заметки в файле, то при выходе они пропадут. То есть во время работы программы все написанные заметки хранятся в кэше, и чтобы их сохранить, нужно загрузить их в текстовой файл. Также есть проверка целостности файла:



На этом моменте функции из меню заканчиваются, и начинаются функции для работы с заметками, для каждого типа заметок они дублируются, т.е в программе есть несколько функций с одинаковым названием, но с разными типами аргументов.

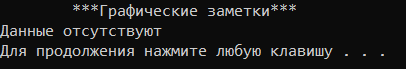
**voidCopy()** принимает как аргументы: количество элементов для копирования, исходный массив (из которого происходит копирование), ссылка на массив в который нужно скопировать элементы и элемент для того, чтобы его исключить. Благодаря последнему эту функцию можно использовать для перезаписи (удаления), что делает её универсальной.

**voidAdd()** добавляет заметку в массив. Если до этого в массиве не было ни одной заметки, создаёт массив с ней.



В ином случае перезаписывает массив (добавляет ещё один элемент). Я использовал временный массив потому что при команде “**origin.count++”** он теряет все свои элементы. После копирования временный массив удаляется.

**voidPrint()** принимает за аргумент структуру и выводит все заметки одного типа с помощью метода **Print** , который прописан в каждом классе. В случае если вывести нечего, программа это обозначит:



**voidEdit()** проверяет наличие заметок, правильность ввода индекса, и меняет значения.

**voidDel()** очень похожа на **voidAdd()** с одним лишь различием: при вызова функции **Copy()** программа уменьшит размер массива на 1 элемент - заметку, которую нужно удалить. Для этого в функции **Copy()** нужен был ещё один аргумент.

**voidSearch()** сначала создаёт флаг, который означает: нашла программа хоть одну заметку или нет. Потом проходит по всем элементам и вызывает для каждого метод **Search(),** ивыводит подходящий элемент (если таковой есть).

**voidFiltration()** работает почти также, но выводит все подходящие элементы.

**voidClear()** и **voidInit()** соответственно: очищают и инициализируют массив для хранения заметок.

### **Классы, структуры и методы**

Наследование является одним из основополагающих принципов ООП. В соответствии с ним, класс может использовать переменные и методы другого класса как свои собственные. Наследование полезно, поскольку оно позволяет структурировать и повторно использовать код, что, в свою очередь, может значительно ускорить процесс разработки.

В программе класс Note является родительским, у него есть свои дочерние классы, которые наследуют структуры данных вместе с конструктором и деструкторы, которые служат для создания и удаления объектов.

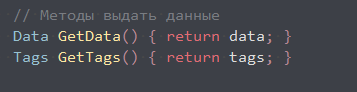
Классы **ANote, VNote, GNote** и **TNote** являются потомками класса **Note**. В каждом из них есть свои: **SetTags(), SetData(), Print()** и **Search()**, сеттеры и геттеры. Начнём с меток:

**SetTags()** с помощью цикла присваивает теги(метки)

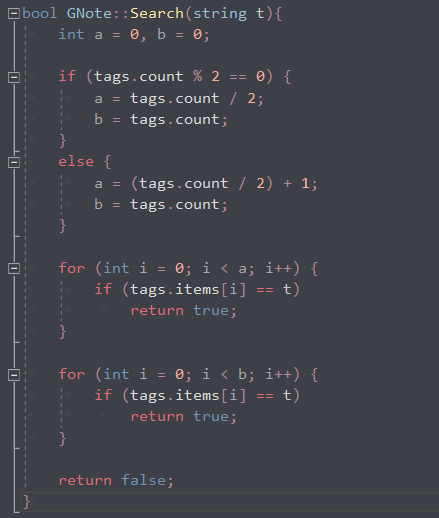
**SetData()** просто присваивает дату

**Print()** выводит одну заметку, даты и метки или же если меток нет, обозначает это.

Геттеры возвращают некоторые данные заметки, которые недоступны вне.



**Search()**реализовать алгоритм поиска по меткам можно через односвязные и двусвязные списки, но проблема такого решения в том, что проход по элементам такого списка мягко говоря не быстрый, особенно если нужен последний элемент. Поэтому алгоритм поиска реализован через обычный одномерный массив. Сам алгоритм следующий:

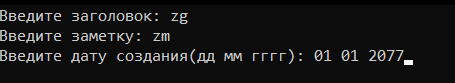


Разбиваем массив на две части (если количество элементов нечётное, то одна часть будет больше) и проходим по первой половине, потом по второй.

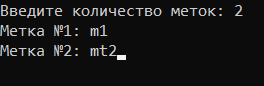
Такое решение будет быстрее, чем если бы программа проходила через весь массив, тем более если бы он был динамический.

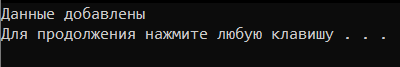
## **Тестирование программы**

Создадим заголовок, заметку и дату:



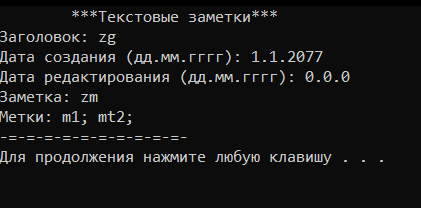
Добавим две метки:



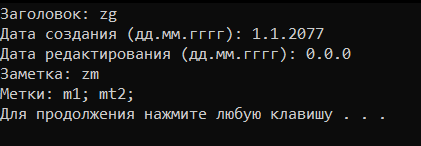


Теперь попробуем её сохранить в файл:

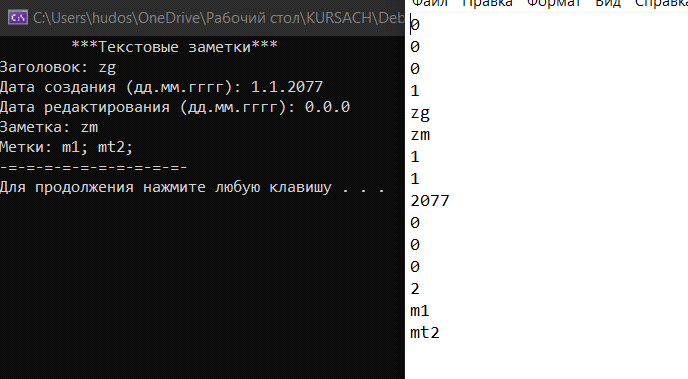
И теперь выводим её:



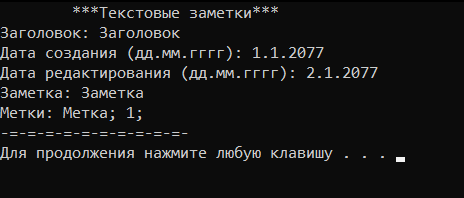
Теперь попробуем её найти по первой метке:



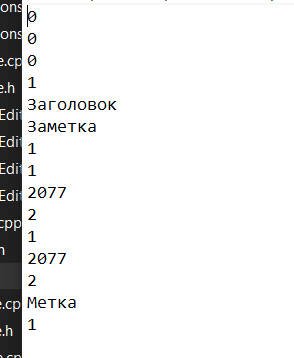
Теперь откроем в текстовом редакторе файл с заметкой:



Отредактируем её:



Теперь в текстовом файле есть изменённая заметка с датой изменения:



# **Заключение**

В ходе выполнения курсовой работы было создано консольное приложение по работе с заметками на языке C++. При разработке программы было изучены разные способы поиска и их особенности. Была применена на практике программа второго семестра по объектно-ориентированному программированию. Выполнение курсовой работы помогло закрепить изученный материал.

# **Список литературы**

**1.Бьерн Страуструп «Язык программирования C++. Специальное издание. Пер. с англ». — М.: Издательство Бином, 2011 г. — 1136 с: ил.**

**2.Уильям Топп, Уильям Форд “Структуры данных в C++”**

**3.“Алгоритмы: построение и анализ”, авторы: Кормен Т.Х., Лейзерсшон Ч.И., Ривест Р.Л.**

**4.Revesli - программирование для начинающих.**

[**5.Работаем с потоками на C++ ofstream и ifstream**](https://www.makak.ru/2008/03/30/rabotaem-s-potokami-na-c-ofstream-i-ifstream/)**.**