**Задание:**

Переделать программу так, чтобы она:

1) Имела точно такое же меню, как и в задании (!);

2) Выводила таблицу как можно более приближенно к таблице задания (можно, например, использовать char-символы от 0 до 255). Если можно – то лучше в точности, как в задании, таблицу выводить(!);

3)В структуре не обязательны конструкторы, убрать, слово public тоже;

4) Если в классах используются закрытые или защищенные элементы, обязательно должны быть простые сет и гет методы доступа к ним(!);

5)Лучше вообще не использовать структуры, только классы, поля – марка машины и проч. – сделать закрытыми(защищенными)+см.пункт 4, т.е. должна быть инкапсуляция;

6)Вообще программу разбить. Класс – заголовочный файл и файл реализации, main отдельно, имя файла класса такое же, как и имя класса;

Создать абстрактный класс "Таблица". От данного абстрактного класса породить класс "Таблица с форматированием", в котором должны быть перегружены операции ввода из потока и вывода в поток. Данные должны выводиться в виде таблицы с автоматическим подбором ширины столбца и высоты строки по содержимому ячеек, использовать псевдографику для разлиновки таблицы в соответствии с заданием.

Написать программу, использующую разработанную иерархию классов и реализующую меню со следующими пунктами:

1.Указание имени текущего файла

2.Запись данных в файл с помощью функций C

3.Запись данных в файл с помощью функций C++

4.Чтение данных из файла с помощью функций C

5.Чтение данных из файла с помощью функций C++

6.Выход

Программа должна выводить данные в консоль и в файл. Необходимо предусмотреть проверку существования файла (если файл не существует, то при чтении выдается ошибка, при а при записи создается новый файл). При записи в существующий файл новые записи добавляются к существующим данным в файле. В меню должна отображаться информация было ли выбрано имя файла.

**Вариант 6.**



**Программа которую нужно переделать.**

#include <iostream> // Для потоковой работы с консолью (cin cout)

#include <iomanip> // для форматирования вывода (setw())

#include <string> // Для класса string

#include <clocale> // Для русского языка в консоле

#include <fstream> // Для потоковой работы с файлами

#include <cstdio> // Для работы с файлми в стиле Си

#include <io.h> // Для быстрой проверки создан файл или нет

using namespace std;

// Структура записи таблицы

struct Data

{

public:

 string brend; // Марка машины

 string number;// Гос. номер

 string owner; // Ф.И.О. владельца

 string date; // Год выпуска

 string color; // Цвет

 //Пустой конструктор

 Data() {}

 // Конструктор с параметрами, инициализирующий все поля записи

 Data(string brend, string number, string owner, string date, string color)

 {

 this->brend = brend;

 this->number = number;

 this->owner = owner;

 this->date = date;

 this->color = color;

 }

};

// Класс таблицы - абстрактный

class Table

{

public:

 int countOfData; // Количество записей в таблице

 Data\* dataArray; // Массив записей таблицы

 // Пустой конструктор

 //Table() {}

 // В с++ класс является абстрактным, если у него хотя бы одна чисто виртуальная функция (пример определени яниже)

 // Суть этой функции в производном классе будет очистка таблицы

 virtual void Clear() = 0;

};

// Класс таблицы с отформатированным выводом

class FormattedTable : public Table

{

public:

 //Конструктор без параметров

 FormattedTable()

 {

 countOfData = 0; // Таблица будет с 0 записями

 dataArray = new Data[countOfData]; // Выделяем память под 0 записей, для того, чтобы массив был инициализирован

 }

 //Конструктор с параметрами, data - массив записей, n - количество записей в массиве

 FormattedTable(Data\* data, int n)

 {

 countOfData = n; // Просто инициализируем поля таблиц

 dataArray = data;

 }

 // Добавление записей в таблицу

 // Добавление происходит в конец таблицы

 // addedData - добавленные записи

 // countOfNewData - их количество

 void AddData(Data\* addedData, int countOfNewData)

 {

 int newCount = countOfData + countOfNewData; // Новый размер таблицы = сумме старой и добавленных записей

 Data\* newData = new Data[newCount]; // выделяем память под новую таблицу

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

 {

 newData[i] = dataArray[i]; // Сначала копируем старые записи

 }

 for (size\_t i = countOfData; i < newCount; i++)

 {

 newData[i] = addedData[i - countOfData]; // Потом копируем новые записи

 }

 Clear(); // Очищаем прошлую таблицу

 countOfData = newCount; // Обновляем количество записей

 dataArray = newData; // Перезаписываем массив записей на новый

 }

 // Очистка таблицы (освобождение памяти)

 void Clear()

 {

 countOfData = 0; // Обнуляем кол-во записей

 delete[] dataArray; // Освобождаем память массива записей

 dataArray = new Data[countOfData]; // Инициализируем пустым массивом, чтобы не был пустой указатель (так как мы же добавлять в этот массив будем)

 }

 // Перегрузка оператора << для вывода таблицы в поток

 friend ostream& operator<<(ostream& out, const FormattedTable& table)

 {

 // Выводим общую информацию о таблице

 out << "Таблица автомобилей" << endl;

 out << "Количество записей " << table.countOfData << endl;

 for (size\_t i = 0; i < 73; i++)

 {

 out << "="; // Делаем верхную границу таблицы

 }

 // ФОрмируем шапку таблицы (заголовки столбцов)

 // setw(n) в контексте setw(n) << "Марка" (к примеру) означает, что на слово "Марка" будет отведено n символов

 // то есть если слово "Марка" будет короче чем n символов, то слева от слова будут добавлены пробелы, если длиннее n то обрежется

 // Это нужно для красивоого отформатированного вывода

 // Установленые значения в setw() не менять, так как они подобраны, исходя из текста , который они форматируют а так же размеров экрана

 out << endl;

 out << "|" << setw(15) << "Марка"

 << "|" << setw(10) << "Гос.номер"

 << "|" << setw(20) << "ФИО владельца"

 << "|" << setw(12) << "Год выпуска"

 << "|" << setw(10) << "Цвет"

 << "|" << endl;

 // После шапкиформатируя по тем же соображения выводим информацию таблицы

 for (size\_t i = 0; i < table.countOfData; i++)

 {

 out << "|" << setw(15) << table.dataArray[i].brend

 << "|" << setw(10) << table.dataArray[i].number

 << "|" << setw(20) << table.dataArray[i].owner

 << "|" << setw(12) << table.dataArray[i].date

 << "|" << setw(10) << table.dataArray[i].color

 << "|" << endl;

 }

 // Низ таблицы

 for (size\_t i = 0; i < 73; i++)

 {

 out << "=";

 }

 out << endl;

 return out; // Возвращаем потом обратно (чтобы можно было написать например cout << table << endl)

 }

 // Перегрузка оператора >> для чтения из потока и записи в таблицу

 // Причем таблицу передаем по значению, так ее нужно изменить

 friend istream& operator>>(istream& in, FormattedTable& table)

 {

 // Здесь уже сложнее

 // Необходимо из строковой таблицы получить массив объектов таблицы (так как таблица храница массивом записей а не строк)

 // Небольшая справка

 // функция getline(in , s) берет из потока in считывает строку вплоть до символа переноса строки, и записывает прочитанное в s

 // после чего переходит на след. строку в потоке.

 // Перегрузка getline(in, s, ' '); делает тоже самое, только читает вплоть до символа пробела

 // Вспоминаем как выглядит наша таблица

 string s;

 getline(in, s); // Сначала идет строка "Таблица автомобилей", пропускаем ее, она нам неинтересна

 int count;

 // Потом идет строка "Количество записей " << table.countOfData

 // Тут нам нужно получить количество записей

 // То есть подстроку после второго пробела в данной строке

 getline(in, s, ' '); // Пропускаем первое слово

 getline(in, s, ' '); // Второе слово

 getline(in, s, ' '); // Считываем третье слово

 // string - это не старый char\*, но у него есть метод c\_str(), которые преобразовывает string в char

 // Это нужно так как многие функции принимают старые строки , а не string

 // В данном случае atoi делает из строки число

 count = atoi(s.c\_str()); // Из третьего слова получили количество записей

 getline(in, s); // Пропускаем верхнюю границу таблицы

 Data\* data = new Data[count]; // Выделяем память под массив записей

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

 {

 // Дальше разделяем записи таблицы по |

 getline(in, s, '|'); // Так как любая запись в таблице начинается с | то первый такой символ надо пропустить

 getline(in, s, '|'); // А дальше по схеме, прочитали до палки, и записали это как очередная порция данных

 data[i].brend = s;

 getline(in, s, '|');

 data[i].number = s;

 getline(in, s, '|');

 data[i].owner = s;

 getline(in, s, '|');

 data[i].date = s;

 getline(in, s, '|');

 data[i].color = s;

 }

 table.AddData(data, count); // Добавляем найденные записи в таблицу

 return in;

 }

 // Заполняет таблицу несколькими строками

 void FillTable()

 {

 int n = 5;

 Data\* addedDataArray = new Data[n];

 addedDataArray[0] = Data("Ford Focus", "A001AA", "Ivanov A.A.", "14.10.1993", "Green");

 addedDataArray[1] = Data("Honda", "B666AA", "Petrov S.V.", "25.03.2016", "Black");

 addedDataArray[2] = Data("Lada Granta", "H546KH", "Sidorov T.E.", "05.11.2011", "Blue");

 addedDataArray[3] = Data("Mercedes Benz", "T999TT", "Putin V.V.", "20.12.2015", "Gold");

 addedDataArray[4] = Data("Lada Priora", "B423BA", "Burovinskiy E.A.", "01.02.2011", "White");

 AddData(addedDataArray, n);

 cout << "В таблицу добавлено " << n << " записей" << endl;

 }

 // Чтение файла в стиле СИ

 void ReadFromFile(string fileName)

 {

 FILE\* f; // Указатель на файл

 Data\* fileData; // считанные данные

 int fileCount; // количество считанных записей

 fopen\_s(&f, fileName.c\_str(), "r"); // Открываем файл на чтение

 char c[80]; // Заводим строку для построчного чтения информации из файла

 // Данная функция читает строку из файла, то есть все символы до символа конца строки. И переходит на следующую строку в файле.

 // Первый аргумент - куда записываем строку

 // Второй - максимальное количество символов которое записывается в эту строку (то есть либо до конца прочтем , либо ограничимся столькими символами)

 // Третий аргумент - указатель на файл

 fgets(c, sizeof(c), f); // Считываем и пропускаем строку "Таблица автомобилей"

 fgets(c, sizeof(c), f); // Считываем строку "Количество записей ..."

 // Из этой строки надо взять количество записей

 const char separator[] = " "; // Массив разделителей

 char\* word = NULL;

 // strtok\_s (берет строку в первом аргументе и разбивает с помощью разделителей(второй аргумент) на строки, заменяя разделители - символами конца строки)

 // Когда первым аргументом является 0 (пустой указатель), функция идет по строке из пред. вызова и берет следующее слова

 strtok\_s(c, separator, &word); // Разбили строку на несколько. В word лежит первое слово

 strtok\_s(0, separator, &word); // После этого вызова в word лежит число запиисей (строка)

 fileCount = atoi(word); // Преобразуем ее в целое число (стандартная функция)

 fileData = new Data[fileCount]; // Выделяем память под массив

 // Дальше операция похожая, будем считывать строку, и разбивать по символу | (разделитель колонк в таблице)

 char\* context = NULL;

 const char delimiter[] = "|"; // Разделитель теперь другой

 fgets(c, sizeof(c), f); // На очереди строка с границей таблицы (много ===), ее пропускаем

 fgets(c, sizeof(c), f); // Далее считываем и пропускаем шапку таблицы

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

 {

 Data data;

 fgets(c, sizeof(c), f); // Тут уже пошло считывание очередной записи

 // Далее вызовами strtok\_s(0, delimiter, &context) поэлементно расщепляем строку на нужные данные и записываем в структуру

 word = strtok\_s(c, delimiter, &context);

 data.brend = word;

 word = strtok\_s(0, delimiter, &context);

 data.number = word;

 word = strtok\_s(0, delimiter, &context);

 data.owner = word;

 word = strtok\_s(0, delimiter, &context);

 data.date = word;

 word = strtok\_s(0, delimiter, &context);

 data.color = word;

 fileData[i] = data; // Добавляем в массив

 }

 fclose(f); // Закрываем файл

 AddData(fileData, fileCount); // Добавляем считанные файлы а таблицу

 }

 // Запись в файл в стиле СИ

 void WriteInFile(string fileName)

 {

 FILE\* f; // Указатель на файл

 fopen\_s(&f, fileName.c\_str(), "w"); // Открываем файл на запись

 // В стиле СИ есть несколько способов записывать строки в файл

 // функция fputs("строка", файл) - самая простая, записывает строку из первого аргумента в файл

 fputs("Таблица автомобилей\n", f); // Записываем и добавляем перенос строки

 fputs("Количество записей ", f); // Теперь надо добавить число записей

 char countString[5]=""; // Выделяем строку под число

 // atoi - функция делает число из строки

 // itoa - наоборо, из числа строку

 // Первый аргумент - число для преобразования, второй строка куда записываем, и третий - система счисления, в которой интерпретировать наше число

 \_itoa\_s(countOfData, countString, 10);

 fputs(countString, f); // Записываем число записай в файл

 fputs("\n", f); // Перенос строки

 for (size\_t i = 0; i < 73; i++)

 {

 fputs("=", f); // ФОрмируем границу таблицы

 }

 // Переходим на новую строку

 fputs("\n", f);

 // Далее воспользуемся другой функцией записи в файл

 // Так как нам данные нужно записать отформатировано и красиво. fputs это не поддерживает

 // fprintf - пишет в файл(первый аргумент), в формате(второй аргуемент), то что лежит в третьем аргументе

 // В данном случае %s - формат строки, %32s, %2s означает сколько мы символов отводим для печати данной строки

 // Все что вне %s, будет напечатано, например | сразу после столбца таблицы

 // Заполняем шапку

 fprintf(f, "|%15s|", "Марка");

 fprintf(f, "%10s|", "Гос.номер");

 fprintf(f, "%20s|", "ФИО владельца");

 fprintf(f, "%12s|", "Год выпуска");

 fprintf(f, "%10s|\n", "Цвет");

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

 { // В таком же формате заполняем данные записей

 fprintf(f, "|%15s|", dataArray[i].brend.c\_str());

 fprintf(f, "%10s|", dataArray[i].number.c\_str());

 fprintf(f, "%20s|", dataArray[i].owner.c\_str());

 fprintf(f, "%12s|", dataArray[i].date.c\_str());

 fprintf(f, "%10s|\n", dataArray[i].color.c\_str());

 }

 for (size\_t i = 0; i < 73; i++)

 {

 fputs("=", f); // Формируем нижнюю границу таблицы

 }

 fclose(f); // Закрываем файл

 }

};

bool FileExists(string fileName)

{

 return (\_access(fileName.c\_str(), 0) != -1);

}

int main()

{

 setlocale(0, ""); // Подключаем русский язык в консоле

 FormattedTable table; // Заводим таблицу

 bool fileIsChoosen = false; // Признак того, что файл уже выбран

 string fileName; // Имя файла

 int choice = -1; // Переменная, которая будет хранить команду пользователя

 while (choice != 0)

 {

 system("cls"); // Очищаем экран

 cout << "Таблица содержит " << table.countOfData << " записей" << endl; // Выводим количество записей в таблице

 if (fileIsChoosen) cout << "Выбранный файл " << fileName << endl; // Выводим имя файла если он выбран

 else cout << "Файл для операций записи / чтения не выбран" << endl;

 cout << "0. Выход из программы" << endl; // Вывод основного меню

 cout << "1. Выбрать файл" << endl;

 cout << "2. Добавить записи в таблицу (несколько заготовленных записей)" << endl;

 cout << "3. Очистить таблицу" << endl;

 cout << "4. Показать таблицу (печать на консоль)" << endl;

 cout << "5. Сохранить таблицу в файл (в стиле с++ потоковый ввод/вывод)" << endl;

 cout << "6. Считать таблицу из файла (в стиле с++ потоковый ввод/вывод)" << endl;

 cout << "7. Сохранить таблицу в файл (в стиле Cи с помощью указателей на файл и базовых операций)" << endl;

 cout << "8. Считать таблицу из файла (в стиле Cи с помощью указателей на файл и базовых операций)" << endl;

 cout << endl << "Введите команду: ";

 cin >> choice; // Вводим команду, и в зависимости от нее уже плящем

 switch (choice)

 {

 case 0:

 break;

 case 1:

 cout << "Введите название файла. Файл должен быть текстовым (например table.txt): ";

 cin >> fileName;

 fileIsChoosen = true;

 break;

 case 2:

 table.FillTable(); // Заполняем таблицу

 break;

 case 3:

 table.Clear(); // Очищаем таблицу

 cout << "Таблица очищена" << endl;

 break;

 case 4:

 cout << table; // Печатаем таблицу

 break;

 case 5:

 {

 if (!fileIsChoosen) // Если файл не выбран выбираем его

 {

 cout << "Сначала нужно выбрать файл" << endl;

 break;

 }

 ofstream out(fileName); // Заводим поток на запись в файл

 out << table; // Выводим туда таблицу

 cout << "Таблица записана в файл " << fileName << endl;

 out.close(); // Не забываем закрыть за собой поток

 break;

 }

 case 6:

 {

 if (!fileIsChoosen)// Если файл не выбран выбираем его

 {

 cout << "Сначала нужно выбрать файл" << endl;

 break;

 }

 if (!FileExists(fileName))

 {

 cout << "Указанный файл не существует. Введите название существующего файла" << endl;

 break;

 }

 ifstream in(fileName);// Заводим поток на чтение из файла в таблицу

 in >> table;

 cout << "Таблица считана из файла " << fileName << endl;

 cout << "Замечание: данные, считанные из файла, не перезаписали таблицу, а добавили записи в конец" << endl;

 in.close();// Не забываем закрыть за собой поток

 break;

 }

 case 7:

 {

 if (!fileIsChoosen)// Если файл не выбран выбираем его

 {

 cout << "Сначала нужно выбрать файл" << endl;

 break;

 }

 table.WriteInFile(fileName);

 cout << "Таблица записана в файл " << fileName << endl;

 break;

 }

 case 8:

 {

 if (!fileIsChoosen)// Если файл не выбран выбираем его

 {

 cout << "Сначала нужно выбрать файл" << endl;

 break;

 }

 if (!FileExists(fileName))

 {

 cout << "Указанный файл не существует. Введите название существующего файла" << endl;

 break;

 }

 table.ReadFromFile(fileName);

 cout << "Таблица считана из файла " << fileName << endl;

 cout << "Замечание: данные, считанные из файла, не перезаписали таблицу, а добавили записи в конец" << endl;

 break;

 }

 default:

 cout << "Введите корректную команду из предложенных" << endl;

 break;

 }

 system("PAUSE"); // Делаем паузу, чтобы пользователь видел результат операций

 }

 cout << "Работа программы завершена" << endl;

 return 0;

}