Черноморский национальный университет имени Петра Могилы

# Алгоритмы и структуры данных

Методические указания к выполнению лабораторных работ

Сост. к. пед. н. доцент Кирей Е. А.

Николаев 2017

# Содержание

**Лабораторная работа № 1. Сортировка**

Цель работы

- ознакомиться с основными приемами обработки одномерных массивов;
- ознакомиться с различными методами сортировки массивов;
- научиться применять полученные знания для решения практических задач.

## Теоретические сведения

Мы будем рассматривать методы сортировки файлов, которые состоят из элементов, обладающих ключами. Эти понятия являются естественными абстракциями в современных средах программирования. Ключи, которые являются лишь частью (зачастую очень небольшой частью) элементов, используются для управления сортировкой. Цель метода сортировки заключается в перемещении элементов таким образом, чтобы их ключи были упорядочены по некоторому заданному критерию (обычно это числовой или алфавитный порядок). Конкретные характеристики ключей и элементов в разных приложениях могут существенно отличаться друг от друга, однако абстрактное понятие размещения ключей и связанной с ними информации в определенном порядке и представляет собой суть задачи сортировки.

Если сортируемый файл полностью помещается в оперативной памяти, то метод сортировки называется **внутренним**. Сортировка файлов, хранящихся на диске, называется **внешней**. Основное различие между этими двумя методами заключается в том, что при внутренней сортировке возможен легкий доступ к любому элементу, а при внешней сортировке возможен только последовательный перебор элементов или, по крайней мере, большими блоками.

Из всех характеристик производительности алгоритмов сортировки в первую очередь интересует **время их выполнения**. Для выполнения сортировки $N$ элементов методом выбора, методом вставок и пузырьковым методом требуется время, пропорциональное $N^2$. Более совершенные методы могут упорядочить $N$ элементов за время, пропорциональное $NlogN$, однако эти методы не всегда столь же эффективны, как перечисленные три метода, для небольших значений $N$, а также в некоторых особых случаях.

Вторым по важности фактором является **объем дополнительной памяти**, используемой алгоритмом сортировки. По этому критерию все методы можно разбить на три категории:

1. те, которые выполняют сортировку на месте и не требуют дополнительной памяти, за исключением, возможно, небольшого стека или таблицы;
2. те, которые используют представление в виде связного списка или каким-то другим способом обращаются к данным с помощью $N$ указателей или индексов массивов, для которых нужна дополнительная память;
3. и те, которые требуют дополнительной памяти для размещения еще одной копии сортируемого массива.

Часто применяются методы сортировки элементов с несколькими ключами — иногда даже требуется упорядочение одного и того же набора элементов в разные моменты по разным ключам. В таких случаях очень важно знать, обладает ли выбранный метод сортировки свойством устойчивости.

Говорят, что метод сортировки **устойчив**, если он сохраняет относительный порядок размещения в файле элементов с одинаковыми ключами.

Например, если имеется список учеников, упорядоченный по алфавиту и году выпуска, то устойчивый метод сортировки выдаст список учеников, распределенный по классам, в том же алфавитном порядке, а неустойчивый метод, скорее всего, выдаст список без следов первоначальной упорядоченности.

## Технология выполнения работы

**Задание 1**

Реализуйте решение задачи 1.1 с использованием языка C++.

**Задача 1.1. Сортировка массива пузырьковым методом (Bubble sort)**

Данная программа служит иллюстрацией реализации базовых алгоритмов сортировки массивов.

*Листинг 1.1. Сортировка пузырьковым методом (Bubble sort)*

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int n; // количество элементов массива

void print(int *a);
int control();
void exch(int &a, int &b);
void sort_bubble(int *array);
int main() {
/////////////// Создание массива ///////////////
	n = control();
	int *a = new int[n];
	for (int i = 0; i < n; i++) a[i] = rand()%201-100; // от -100 до 100
/////////////// Сортировка ///////////////
	unsigned int start_time = clock(); // нчальное время
	sort_bubble(a);
	unsigned int end_time = clock(); // конечное время
	unsigned int search_time = end_time - start_time; // искомое время
	cout <<"\n"<< "Время работы программы: " << search_time << " мс"<< endl;

	system ("pause");
	return 0;
}
// Вывод элементов массива
void print(int *a){
	for(int i = 0; i < n; i++)
		cout << a[i] << '\t';
		cout << endl;
}
// Контроль ввода
int control(){
	while(true){
		cout << "Введите чило: ";
		cin >> n;
		if (cin.get() == '\n' && n > 0 ) break;
		else{
			cout <<"Ожидается целое число!";
			cin.clear();	// сбрасывает флаги ошибок
			cin.sync();	// извлекаются все символы, вплоть до '\n'
		}
	}
	return n;
}
// Перестановка
void exch(int &a, int &b){
	int tmp = a;
	a = b; b = tmp;
}
// Сортировка пузырьковым методом
void sort_bubble(int *array){
	for (int j = n-1; j >= 0; j--){
```

```
            for (int i = 0; i < j; i++){
                  if (array[i] > array[i+1]) exch(array[i], array[i+1]);
            }
      }
}
```

1. Проведите эксперимент и узнайте, как влияют на время работы элементы массива:
   - массив состоит из случайных чисел;
   - первоначально массив отсортирован по убыванию;
   - массив имеет два одинаковых значения расположенные в произвольном порядке;
   - массив имеет все одинаковые значения.
2. Модернизируйте код программы исходя из условий п. 1 и заполните сравнительную таблицу. Например, для пузырькового метода получены следующие данные.

**Время сортировки**

Количество элементов массива – **100 000**
ОС – **Windows 7**, процессор – **Intel Core i5-2500 CPU 3.30GHz**, ОЗУ – **4 ГБ**

| Название метода сортировки | Массив состоит из случайных чисел (**элементы массива сформированы функцией rand() в диапазоне от -1000 до 1000)** | Первоначально массив отсортирован по убыванию | Массив имеет два одинаковых значения, расположенные в произвольном порядке | Массив имеет все одинаковые значения |
|---|---|---|---|---|
| **пузырьком (Bubble sort)** | **46597 мс** | **46084 мс** | **39810 мс** | **23215 мс** |

3. Ознакомьтесь с методами сортировки массивов (edunow.su/site/content/algorithms/sortirovka_massiva):
   - пузырьком (Bubble sort);
   - вставками (Insertion sort);
   - выбором (Selection sort);
   - слиянием (Merge sort);
   - быстрой сортировкой (Quicksort).
4. Заполните таблицу для остальных четырех методов.

**Задание 2**

Напишите программу, которая показывает устойчивость или неустойчивость используемого метода сортировки.

**Примечание**

Для наглядной демонстрации устойчивости метода сортировки можно использовать сортировку строк двухмерного массива, ключом являются элементы выбранного столбца. Например, на рис. 1.1 показан результат сортировки по первому столбцу, а затем по второму пузырьковым методом. При этом порядок сортировки по первому столбцу в пределах одинаковых ключей не нарушен, т. е. этот метод является устойчивым.

.

```
Введите количество строк и столбцов: 8
3
Исходный массив массив
1       2       4
0       4       4
3       3       2
4       0       0
1       2       1
1       0       2
2       1       1
4       2       3
Введите номер столбца сортировки:
0
Отсортированный массив
0       4       4
1       2       4
1       2       1
1       0       2
2       1       1
3       3       2
4       0       0
4       2       3
Введите номер столбца сортировки:
1
Отсортированный массив
1       0       2
4       0       0
2       1       1
1       2       4
1       2       1
4       2       3
3       3       2
0       4       4
Введите номер столбца сортировки:
```

Рис. 1.1

**Задание 3**

Напишите программу для замера производительности сортировки, которая вызывает функцию сортировки для массивов различных размеров (от 1000 до 100 000, шаг 1000), замеряет время каждого выполнения и выводит в файл. По полученым данным постройте графики зависимоти времени сортировки от количества элементов для всех методов п. 3 задания 1. Например, на рис. 1.2 приведен график для сортировки встакми, построенный в MS Excel, добавлена линия тренда и выведено ее уравнение, из котрого видно, что зависимоть квадратичная (рис. 1.2).

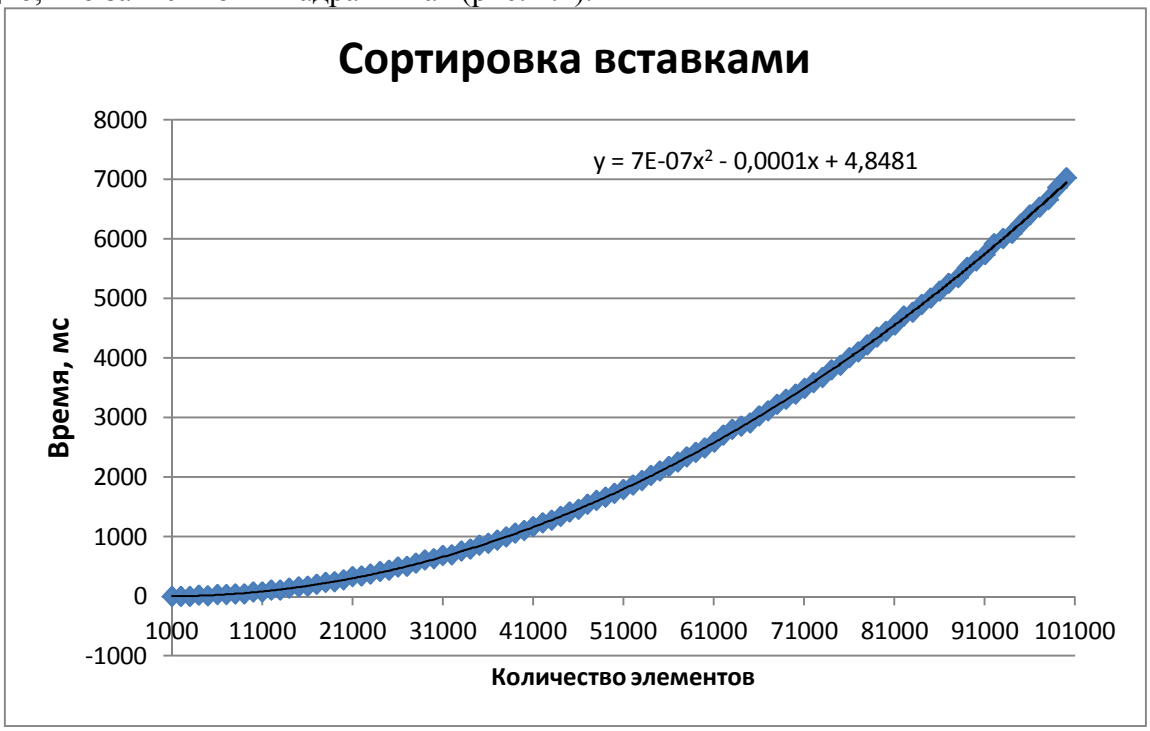


Рис. 1.2

## Лабораторная работа № 2. Строки и файлы

Цель работы

- ознакомиться со способами описания, ввода-вывода, обработки строк;
- научиться применять полученные знания для решения практических задач.

### Теоретические сведения

Любой язык программирования содержит средства представления и обработки текстовой информации. Другое дело, что обычно программист наряду с символами имеет дело с типом данных (формой представления) – строкой, причем особенности ее организации скрыты, а для работы предоставлен стандартный набор функций. В C++, наоборот, форма представления строки является открытой, а программист работает с ней «на низком уровне».

### Представление символов и строк в C++

Немного истории. Символы и строки в C++ реализованы в соответствии со сложившимися в 70-годы прошлого века стандартами представления текста. Тогдашний уровень технологии привел к тому, что в качестве единицы представления символа был выбран байт, информационная емкость которого равно $2^8$=256. Это означает, что одновременно в программе можно представить не более 256 различных символов. Этого хватает для стандартного набора символов и букв латинского алфавита. Для прочих символов (кириллица, национальные алфавиты, псевдо-графика, математические) используются различные кодовые таблицы, работа с которыми не включена в стандарты языка (т.е. попросту не оговаривается в нем). Аналогично, имеются различные варианты представления символа «конец строки» в различных ОС, что создает проблемы, связанные с переносом текстовых файлов.

**Примечание**

Исторически сложившееся «рыночное разнообразие» на момент появления стандарта привело к тому, что имеются несколько кодовых таблиц, представляющих кириллицу:

- кодовая таблица Windows CP-1251;
- кодовая таблица DOS CP-866;
- кодовая таблица Международной организации стандартизации (ISO), используемая семействами мобильных ОС UNIX, Linux, FreeBSD и т.п. - ISO-8859-5;
- кодовые таблицы «советских» стандартов кодов информационного обмена (KOI-8) - CP KOI-8U и CP KOI-8R.

### Стандартный ввод-вывод

Имеющийся в большинстве языков программирования стандартный символьный ввод-вывод соответствует работе программы в режиме командной строки (консольного приложения). Несмотря на то, что такой режим работы не имеет массового распространения, он остается актуальным для «внутреннего» программирования по следующим причинам:

- работа с текстовыми файлами «вписана» в стандартный ввод-вывод (например, в C++ потоки ввода-вывода могут быть перенаправлены как на текстовый файл, так и на консольный ввод-вывод (клавиатура – экран));
- текстовыми файлами являются файлы исходных текстов программ (C++ – cpp, Паскаль – pas, Бейсик – bas), значительная часть файлов с параметрами настройки различных приложений (ini), командных файлов (файлов последовательностей команд – bat);
- если приложения не работают с форматами данных друг друга (не совместимы по данным), то единственным форматом обмена является текстовый файл, в котором числовые (или символьные) данные разделены стандартными разделителями (пробел, табуляция, запятая, точка с запятой, конец строки). Обмен данными через

такие файлы называется экспортом-импортом. В C++ файлы такого формата читаются стандартными функциями форматного ввода;

- многие приложения (компиляторы, серверные приложения) наряду с оконными интерфейсами имеют возможность работы в режиме командной строки и чтения управляющих (текстовых) командных файлов.

**Символ текста**

Базовый тип данных char понимается трояко: как байт – минимальная адресуемая единица представления данных в компьютере, как целое со знаком (в диапазоне –127…+127) и как символ текста. Этот факт отражает общепринятые стандарты на представление текстовой информации, которые «зашиты» как в архитектуре компьютера (клавиатура, экран, принтер), так и в системных программах. Стандартом установлено соответствие между символами и присвоенными им значениями целой переменной (кодами). Любое устройство, отображающее символьные данные, при получении кода выводит соответствующий ему символ. Аналогично клавиатура (совместно с драйвером) кодирует нажатие любой клавиши с учетом регистровых и управляющих клавиш в соответствующий ей код.

Имеется ряд кодов неотображаемых символов, которым соответствуют определенные действия при вводе-выводе. Для их представления в программе используются символьные константы, начинающиеся с обратной косой черты (табл. 2.1).

Таблица 2.1

| Константа | Название | Действие |
|---|---|---|
| \a | bel | Звуковой сигнал |
| \b | bs | Курсор на одну позицию назад |
| \f | ff | Переход к началу (перевод формата) |
| \n | lf | Переход на одну строку вниз (перевод строки) |
| \r | cr | Возврат на первую позицию строки |
| \t | ht | Переход к позиции, кратной 8 (табуляция) |
| \v | vt | Вертикальная табуляция по строкам |
| \\ \' \" \? | | Представление символов \, ', ", ? |
| \nn | | Символ с восьмеричным кодом nn |
| \xnn | | Символ с шестнадцатеричным кодом |
| \0 | | Символ с кодом 0 |

**Строка**

Строкой называется последовательность символов, ограниченная символом с кодом 0, то есть '\0'. Из ее определения видно, что она является объектом переменной размерности. Местом хранения строки является массив символов. Работать с ней можно в цикле, ограниченном не размерностью массива, а условием обнаружения символа конца строки:

```
for (i=0; str[i]; i++) ...
```

Строковая константа – последовательность символов, заключенная в двойные кавычки. Допустимо использование неотображаемых символов. Строковая константа автоматически дополняется символом '\0', ею можно инициализироваться массив, в том числе такой, размерность которого определяется размерностью строки.

**Работа с файлами**

Файлы позволяют пользователю считывать большие объемы данных непосредственно с диска, не вводя их с клавиатуры. Существуют два основных типа файлов: *текстовые* и *двоичные* (*бинарные*).

Текстовыми называются файлы, состоящие из любых символов. Они организуются по строкам, каждая из которых заканчивается символом «конца строки». Конец самого файла обозначается символом «конца файла». При записи информации в текстовый файл,

просмотреть который можно с помощью любого текстового редактора, все данные преобразуются к символьному типу и хранятся в символьном виде.

В двоичных файлах информация считывается и записывается в виде блоков определенного размера, в которых могут храниться данные любого вида и структуры.

В языке C++ для организации работы с файлами используются классы потоков `ifstream` (файловый ввод), `ofstream` (файловый вывод) (рис. 2.1). Перечисленные классы являются производными от `istream`, `ostream` и `iostream` соответственно. Операции ввода-вывода выполняются так же, как и для других потоков, то есть компоненты-функции, операции и манипуляторы могут быть применены и к потокам файлов. Различие состоит в том, как создаются объекты и как они привязываются к требуемым файлам.

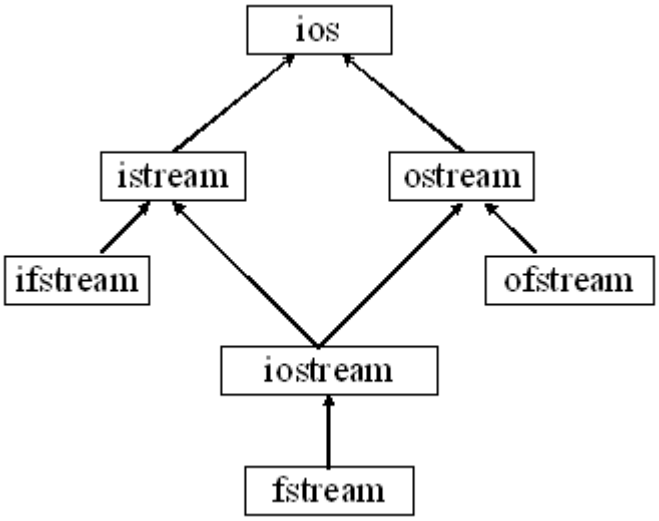


Рис. 2.1. Часть иерархии классов потоков ввода-вывода

В C++ файл открывается путем стыковки его с соответствующим потоком. Рассмотрим организацию связывания потока с некоторым файлом. Для этого используются конструкторы классов `ifstream` и `ofsream`:

```
ofstream(const char* Name, int nMode= ios::out, int nPot= filebuf::openprot);
ifstream(const char* Name, int nMode= ios::in, int nPot= filebuf::openprot);
```

Первый аргумент определяет имя файла (единственный обязательный параметр). Второй аргумент задает режим для открытия файла и представляет битовое ИЛИ ( | ) величин:

- `ios::app` при записи данные добавляются в конец файла, даже если текущая позиция была перед этим изменена функцией `ostream::seekp`;
- `ios::ate` указатель перемещается в конец файла. Данные записываются в текущую позицию (произвольное место) файла;
- `ios::in` поток создается для ввода, если файл уже существует, то он сохраняется;
- `ios::out` поток создается для вывода (по умолчанию для всех `ofstream` объектов), если файл уже существует, то он уничтожается;
- `ios::trunc` если файл уже существует, его содержимое уничтожается (длина равна нулю). Этот режим действует по умолчанию, если `ios::out` установлен, а `ios::ate`, `ios::app` или `ios:in` не установлены;
- `ios::nocreate` если файл не существует, функциональные сбои;
- `ios::noreplace` если файл уже существует, функциональные сбои;
- `ios::binary` ввод-вывод будет выполняться в двоичном виде (по умолчанию текстовой режим).

Например, возможны следующие комбинации перечисленных выше величин:

- `ios::out | ios::trunc` удаляется существующий файл и (или) создается для записи;
- `ios::out | ios::app` открывается существующий файл для дозаписи в конец файла;

Третий аргумент – данное класса `filebuf`, используется для установки атрибутов доступа к открываемому файлу.

Возможные значения `nProt`:

- `filebuf::sh_compat` совместно используют режим;
- `filebuf::sh_none` режим `Exclusive`: никакое совместное использование;
- `filebuf::sh_read` совместно использующее чтение;
- `filebuf::sh_write` совместно использующее запись.

Для комбинации атрибутов `filebuf::sh_read` и `filebuf::sh_write` используется операция логическое ИЛИ ( || ).

Для того чтобы записывать данные в текстовый файл, необходимо:

1. описать переменную типа `ofstream`.
2. открыть файл с помощью функции `open`.
3. вывести информацию в файл.
4. обязательно закрыть файл.

Для считывания данных из текстового файла, необходимо:

1. описать переменную типа `ifstream`.
2. открыть файл с помощью функции `open`.
3. считать информацию из файла, при считывании каждой порции данных необходимо проверять, достигнут ли конец файла.
4. закрыть файл.

## Технология выполнения работы

1. Реализуйте решение задач 2.1-2.2 с использованием языка C++.

**Задача 2.1. Поиск подстроки**

Написать программу, которая определяет, встречается ли в заданном текстовом файле **text_city.txt** заданная последовательность символов. В файле **text_city.txt** находится список крупных городов Англии. Длина строки текста не превышает 80 символов, текст не содержит переносов слов, последовательность не содержит пробельных символов.

**Решение**

*Исходные данные*

1. Текстовый файл **text_city.txt** неизвестного размера, состоящий из строк длиной не более 80 символов. Поскольку по условию переносы отсутствуют, можно ограничиться поиском заданной последовательности в каждой строке отдельно. Следовательно, необходимо помнить только одну текущую строку файла. Для ее хранения выделим строковую переменную длиной 81 символ (дополнительный символ требуется для завершающего нуля).
2. Последовательность символов для поиска, вводимая с клавиатуры. Поскольку по условию задачи она не содержит пробельных символов, ее длина также не должна быть более 80 символов, иначе поиск завершится неудачей. Для ее хранения также выделим строковую переменную длиной 81 символ.
3. Результатом работы программы является сообщение либо о наличии заданной последовательности, либо о ее отсутствии.

*Листинг 2.1. Поиск подстроки*

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;

int main() {
    const int len = 81; // длина строки файла и длина последовательности
    char word[len], line[len]; // 1
    cout <<"Введите слово для поиска: ";
    cin >> word;
    ifstream fin("text_city.txt"); // 2
    if (!fin){ // проверка успешности создания объекта fin
        cout << "Ошибка открытия файла!" << endl;
        system ("pause");
        return 1;
    }
    while(fin.getline(line, len)){ // 3
        if (strstr(line, word)){
            cout << "Присутствует!" << endl;
            system ("pause");
            return 1;
        }
    }
    cout << "Отсутствует!" << endl;
    system ("pause");
}
```

В операторе 1 описывается переменная `line` для размещения очередной строки файла и переменная `word` для хранения искомой последовательности символов. В операторе 2 определяется объект `fin` класса входных потоков `ifstream`. С этим объектом можно работать так же, как со стандартными объектами `cin` и `cout`, то есть использовать операции помещения в поток << и извлечения из потока >>, а также функции `get`, `getline` и др. Предполагается, что файл с именем **text_city.txt** находится в том же каталоге, что и текст программы, иначе следует указать полный путь, дублируя символ обратной косой черты, так как иначе он будет иметь специальное значение:

```
ifstream fin("c:\\prim\\cpp\\text_city.txt");    // 2
```

**Файлы, открываемые для чтения, нужно обязательно проверять!**

**Примечание**

В прошлом проверка успешности открытия файла выполнялась следующим образом:

```
if (!fin.fail()) …    // неудача открытия
if (!fin.good()) …    // неудача открытия
if (!fin) …           // неудача открытия
```

Объект `fin`, когда он используется в условии `if`, преобразуется в `false`, `fin.good()` возвращает `false`, и `true` – в остальных случаях, поэтому две приведенные формы эквивалентны. Однако эти тесты не могут правильно распознать ситуацию, когда предпринимается попытка открытия файла с неподходящим режимом файла. Метод `is_open()` перехватывает ошибки подобного рода, наряду с теми, которые перехватываются методом `good()`. Однако в старых реализациях C++ метод `is_open()` отсутствует.

В цикле 3 выполняется чтение из файла в переменную `line`. Метод `getline` при достижении конца файла вернет значение, завершающее цикл.

Для анализа строки применяется функция `strstr(line, word)`. Она выполняет поиск подстроки `word` в строке `line`. Обе строки должны завершаться нуль-символами. В случае успешного поиска функция возвращает указатель на найденную подстроку, в случае неудачи — `NULL` (пустой указатель). Если искомая подстрока пуста, функция возвращает указатель на начало строки `line`.

### Здание для самостоятельного выполнения

1.1. Создайте файл **text_city_ukr.txt** со списком городов Украины (на украинском или русском языке). Решите для этого набора значений предыдущую задачу.

**Примечание**

Для успешного поиска последовательностей, состоящих из русских букв, файл надо создать в текстовом редакторе с кодировкой ASCII — например, во встроенном редакторе оболочки Far. Если же текстовый файл уже существует и создан в кодировке ANSI, используемой в ОС Windows, для преобразования в эту же кодировку строки, вводимой с клавиатуры в переменную `word`, придется воспользоваться нестандартной функцией `OemToChar`, описанной в заголовочном файле `<windows.h>`. У нее два параметра: исходная строка и результирующая. Существует и функция `CharToOem`, выполняющая обратную перекодировку.

### Задача 2.2. Подсчет количества вхождений слова в текст

Написать программу, которая определяет, сколько раз встретилось заданное слово в текстовом файле **Shakespeare_Hamlet.txt**. Файл **Shakespeare_Hamlet.txt** содержит текст пьесы У. Шекспира «Гамлет», длина строки не превышает 100 символов. Текст не содержит переносов слов.

**Решение**

Определим слово как последовательность алфавитно-цифровых символов, после которых следует знак пунктуации, разделитель или признак конца строки. Слово может находиться либо в начале строки, либо после разделителя или знака пунктуации. Это можно записать следующим образом (фигурные скобки и вертикальная черта означают выбор из альтернатив):

```
слово = {начало строки | знак пунктуации | разделитель}
        символы, составляющие слово
        {конец строки | знак пунктуации | разделитель}
```

*Исходные данные и результаты.*

1. Текстовый файл **Shakespeare_Hamlet.txt** неизвестного размера, состоящий из строк длиной не более 100 символов. Поскольку по условию переносы отсутствуют, ограничимся поиском слова в каждой строке отдельно, поэтому выделим строку длиной 101 символ.

2. Слово для поиска, вводимое с клавиатуры (также выделим строку в 101 символ). Результатом работы программы является количество вхождений слова в текст. Представим его в программе в виде целой переменной. Для хранения длины строки будем использовать именованную константу, а для хранения фактического количества символов в слове — переменную целого типа.

*Листинг 2.2. Подсчет количества вхождений слова в текст*

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <cctype>
using namespace std;

int main() {
    //////////// Шаг 1. Открытие файла ////////////
    const int len = 101; //длина строки
    char word[len], line[len];
    cout << "Введите слово для поиска: ";
    cin >> word;
    int l_word = strlen(word); // длина искомого слова
    ifstream fin("Shakespeare_Hamlet.txt"); // поток чтения из файла
    if (!fin){
        cout << "Ошибка открытия файла!";
        system ("pause");
        return 1;
    }
```

```
    //////////// Шаг 2. Количество вхождений слова ////////////
    int count = 0;
    while(fin.getline(line, len)){ // считывание из файла строк
        char *p = line; // хранение в указателе p адреса считанной подстроки
        while(p=strstr(p, word)){/* поиск вхождения подстроки word в строку,
на которую указывает указатель p и запоминание адреса первого символа */
            cout << "Строка: " << p << endl; /* контрольный вывод подстроки с
первым найденным искомым словом */
            char *c = p; // начало найденной подстроки
            p += l_word; /* перемещаемся на слудующий символ после искомого
слова*/
    ////// Анализ символо до и после искомого слова //////
            if (c != line) // слово не в начале строки
                if (!ispunct(*(c-1)) && !isspace(*(c-1)) ) continue;/* перед
словом не знак пунктуации или разделитель*/
           // переход к поиску следующего вхождения слова
            if (ispunct(*p) || isspace(*p) || (*p == '\0') ) count++;
// после слова знак пунктуации, или разделитель, или конец строки
        }
    }
    cout <<"Количество вхождений слова " << word << ": " << count << endl;
    system ("pause");
}
```

Для многократного поиска вхождения подстроки в заголовке цикла используется функция `strstr`. Очередной поиск должен выполняться с позиции, следующей за найденной на предыдущем проходе подстрокой. Для хранения этой позиции определяется вспомогательный указатель `p`, который на каждой итерации цикла наращивается на длину подстроки. Также вводится счетчик количества совпадений. На данном этапе он считает не количество слов, а количество вхождений последовательности символов, составляющих слово.

Для анализа принадлежности символов, находящихся перед словом и после него, множеству знаков пунктуации и разделителей вводится служебная переменная `c` (хранение адреса начала вхождения подстроки). Символы, ограничивающие слово, проверяются с помощью функций `ispunct` и `isspace`, прототипы которых хранятся в заголовочном файле `<cctype>`. Символ, стоящий после слова, проверяется также на признак конца строки (для случая, когда искомое слово находится в конце строки).

### Здание для самостоятельного выполнения

1.2. Решите задачу 2.2 с использованием функции `strtok`.

**Примечание**

Функция `strtok` разбивает переданную ей строку на лексемы в соответствии с заданным набором разделителей. Если воспользоваться этой функцией, то не придется «вручную» выделять и проверять начало и конец слова.

Смешивать в одной программе ввод-вывод с помощью потоковых классов и функций библиотеки не рекомендуется. В целом программа, написанная с использованием функций библиотеки, может получиться более быстродействующей, но менее безопасной, поскольку программист должен заботиться о большем количестве деталей.

### Задания для самостоятельного выполнения

Выполните задания с использованием двух файлов, в которых содержится текст на разных языках.

**Вариант 1**

Написать программу, которая считывает из текстового файла три предложения и выводит их в обратном порядке.

**Вариант 2**

Написать программу, которая считывает текст из файла и выводит на экран только предложения, содержащие введенное с клавиатуры слово.

**Вариант 3**

Написать программу, которая считывает текст из файла и выводит на экран только строки, содержащие двузначные числа.

**Вариант 4**

Написать программу, которая считывает текст из файла и выводит на экран слова, начинающиеся с гласных букв.

**Вариант 5**

Написать программу, которая считывает текст из файла и выводит его на экран, меняя местами каждые два соседних слова.

**Вариант 6**

Написать программу, которая считывает текст из файла и выводит на экран только предложения, не содержащие запятых.

**Вариант 7**

Написать программу, которая считывает текст из файла и определяет, сколько в нем слов, состоящих из не более чем четырех букв.

**Вариант 8**

Написать программу, которая считывает текст из файла и выводит на экран только цитаты, то есть предложения, заключенные в кавычки.

**Вариант 9**

Написать программу, которая считывает текст из файла и выводит на экран только предложения, состоящие из заданного количества слов.

**Вариант 10**

Написать программу, которая считывает текст из файла и выводит на экран слова текста, начинающиеся и оканчивающиеся гласными букв.

**Вариант 11**

Написать программу, которая считывает текст из файла и выводит на экран только строки, не содержащие двузначных чисел.

**Вариант 12**

Написать программу, которая считывает текст из файла и выводит на экран только предложения, начинающиеся с тире, перед которым могут находиться только пробельные символы.

**Вариант 13**

Написать программу, которая считывает текст из файла и выводит его на экран, заменив каждую первую букву слов, начинающихся с гласной буквы, на прописную.

**Вариант 14**

Написать программу, которая считывает текст из файла и выводит его на экран, заменив цифры от 0 до 9 на слова «ноль», «один», ..., «девять», начиная каждое предложение с новой строки.

**Вариант 15**

Написать программу, которая считывает текст из файла, находит самое длинное слово и определяет, сколько раз оно встретилось в тексте.

**Вариант 16**

Написать программу, которая считывает текст из файла и выводит на экран сначала вопросительные, а затем восклицательные предложения.

**Вариант 17**

Написать программу, которая считывает текст из файла и выводит его на экран, добавляя после каждого предложения, сколько раз встретилось в нем введенное с клавиатуры слово.

**Вариант 18**

Написать программу, которая считывает текст из файла и выводит на экран все его предложения в обратном порядке.

**Вариант 19**

Написать программу, которая считывает текст из файла и выводит на экран сначала предложения, начинающиеся с однобуквенных слов, а затем все остальные.

**Вариант 20**

Написать программу, которая считывает текст из файла и выводит на экран предложения, содержащие максимальное количество знаков пунктуации.

**Лабораторная работа № 3. Двоичные файлы**

Цель работы

- ознакомиться с моделью двоичного файла;
- ознакомиться со способами доступа в двоичном файле;
- научиться применять полученные знания для решения практических задач.

## Теоретические сведения

Кроме текстовых файлов существует еще одна общая форма представления данных в файлах, на самом деле более многообразная – двоичный файл. Сюда относятся не только файлы с «мультимедийным» содержанием: документы, изображение, звук. Форматы двоичных файлов используются в базах данных и в самой файловой системе. Ведь файловая система – это не что иное, как большой двоичный файл, в котором построена сложная многоуровневая динамическая структура данных, включающая в себя все файлы и каталоги.

**Двоичный файл** – это файл, в котором используется двоичный поиск или файл, в котором данные представлены в двоичной системе счисления.

### Модель двоичного файла

Двоичный файл отличается от текстового тем, что данные в нем представлены во внутренней форме. А поскольку при внутреннем представлении используется двоичная система счисления, то «в честь ее» файлы и называются двоичными. По существу, двоичный файл является аналогом внутренней (оперативной, физической) памяти – неограниченным массивом байтов с возможностью непосредственного обращения (произвольного доступа) к любой его части.

Один из способов чтения и записи двоичных данных состоит в использовании функций `read()` и `write()`. Наиболее частый способ использования этих функций соответствует прототипу:

```
istream &read(unsigned char *buf, int num);
ostream &write(const unsigned char *buf, int num);
```

Функция `read()` читает `num` байт из ассоциированного потока и посылает их в буфер `buf`. Функция `write()` пишет `num` байт в ассоциированный поток из буфера `buf`. При использовании исключительно этих функций данные, записанные в определенной последовательности в файл, хранятся в нем и читаются в том же самом порядке. Этот неизменный порядок извлечения данных называется **последовательным доступом**, а файл – **последовательным двоичным файлом**. Естественно, что нас при этом не интересуют адреса размещения данных в файле. Однако существует и другой способ, позволяющий извлекать данные в любом произвольном порядке – **прямой (или произвольный) доступ**.

Произвольный доступ базируется на понятии адреса в двоичном файле. Поскольку на физическом уровне двоичный файл представляется как «неограниченно растущий» массив байтов, то под адресом понимается порядковый номер байта, начиная с 0. С каждым открытым файлом связывается такой параметр как **текущая позиция (текущий адрес)** – номер байта, начиная с которого будет выполняться очередная операция чтения-записи. При открытии файла текущая позиция устанавливается на начало файла, после чтения-записи порции данных перемещается вперед на размерность этих данных. Для дополнения файла новыми данными необходимо установить текущую позицию на конец файла и выполнить операцию записи.

**Примечание**

Текущая позиция в файле является адресом размещения переменной в нем, но получить этот адрес можно перед, а не после ее чтения оттуда.

## Технология выполнения работы

**Задание 1.**

Реализуйте решение задач 3.1-3.4 с использованием языка C++.

**Задача 3.1. Вывод вопросительных предложений**

Написать программу, которая считывает текст из файла **Shakespeare_Winter'sTale.txt** и выводит на экран только вопросительные предложения из этого текста. Файл **Shakespeare_Winter'sTale.txt** содержит текст пьесы У. Шекспира «Зимняя сказка».

*Исходные данные и результаты*

Исходные данные: текстовый файл **Shakespeare_Winter'sTale.txt** неизвестного размера, состоящий из неизвестного количества предложений. Предложение может занимать несколько строк, поэтому ограничиться буфером на одну строку в данной задаче нельзя. Примем решение выделить буфер, в который поместится весь файл.

Результаты являются частью исходных данных, поэтому дополнительное пространство под них не выделяется. Будем хранить длину файла в переменной длинного целого типа. Для организации вывода предложений понадобятся переменные того же типа, хранящие позиции начала и конца предложения.

*Листинг 3.1. Вывод вопросительных предложений*

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <windows.h>
using namespace std;

int main() {
    //////////// Шаг 1. Открытие файла ////////////
    ifstream fin("Shakespeare_Winter'sTale.txt ", ios::binary);
    if (!fin.is_open()){
        cout << "Ошибка открытия файла!";
        system ("pause");
        return 1;
    }
    //////////// Шаг 2. Определение размера файла ////////////
    fin.seekg(0, ios::end); // 1 перейти в конец файла
    // 2 текущая позизия в байтах, измеренная от начала файла
    long len = fin.tellg();
    cout << "Рзмер файла: " << len << endl; // контрольный вывод
    //////////// Шаг 3. Вывод вопросительных предложений ////////////
    char *buf = new char [len + 1]; // 3
    fin.seekg(0, ios::beg); // 4 перейти в начало файла
    fin.read(buf, len); // 5 чтение из файла
    buf[len] = '\0';
    long n = 0, i = 0; // 6
    while(buf[i]) { // 7
        if(buf[i] == '?') { // 8
            for (int j = n; j <= i; j++){
                if (j==i) cout << buf[j] << endl;
                else cout << buf[j];
            }
            n = i + 1;
        }
        if (buf[i] == '.' || buf[i] == '!' || buf[i] == '\n') n = i + 1;
        i++;
    }
    fin.close(); // 9
    system ("pause");
}
```

Для определения длины файла используются методы `seekg` и `tellg` класса `ifstream`. С любым файлом при его открытии связывается так называемая текущая позиция чтения или записи. Когда файл открывается для чтения, эта позиция устанавливается на начало файла. Для определения длины файла мы смещаем ее на конец файла с помощью метода `seekg` (оператор 1), а затем с помощью `tellg` получаем ее значение, запомнив его в переменной `len` (оператор 2).

Метод `seekg (offset, org)` смещает текущую позицию чтения из файла на `offset` байтов относительно параметра `org`, который может принимать значения: `ios::beg` — от начала файла, `ios::cur` — от текущей позиции, `ios::end` — от конца файла. Константы `beg`, `cur` и `end` определены в классе `ios`, предке `ifstream`, а символы `::` означают операцию доступа к этому классу.

В операторе 3 выделяется `len + 1` байтов под символьную строку `buf`, в которой будет храниться текст из файла. Мы выделяем на один байт больше, чем длина файла, чтобы после считывания файла записать в этот байт нуль-символ.

Для чтения информации требуется снова переместить текущую позицию на начало файла (оператор 4). Собственно чтение выполняется в операторе 5 с помощью метода `read(buf, len)`, который считывает из файла `len` символов (или менее, если конец файла встретится раньше) в символьный массив `buf`. В операторе 6 определяются служебные переменные. В переменной `n` будет храниться позиция начала текущего предложения, переменная `i` используется для просмотра массива.

Цикл просмотра массива `buf` (оператор 7) завершается, когда встретился нуль-символ. Если очередным символом оказался вопросительный знак (оператор 8), выполняется вывод символов, начиная с позиции `n` и заканчивая текущей, после чего в переменную `n` заносится позиция начала нового предложения. Оператор 9 (закрытие потока) в данном случае не является обязательным, так как он необходим, только когда требуется закрыть поток раньше окончания действия его области видимости.

При выполнении некоторых заданий может потребоваться гораздо менее эффективное посимвольное чтение из файла. При использовании потоков оно выполняется с помощью метода `get`. Например, для приведенной выше программы посимвольный ввод выглядит следующим образом:

```
while((buf[i] = fin.get()) != EOF) {
    // ... тело цикла i++;
}
```

**Здание для самостоятельного выполнения**

1.1. Организуйте вывод каждого вопросительного предложения задачи 3.1 в другой файл построчно (в столбик).

При отладке программ, работающих с двоичными файлами, иногда сложно установить, какой фрагмент – запись или чтение – содержит ошибку. Аналогично, при чтении уже известного формата необходимо проверять, насколько правильно читаются данные. Здесь не обойтись без навыков чтения дампа – двоичного содержимого файла. Все данные и адреса присутствуют в шестнадцатеричной системе счисления.

**Задача 3.2. Запись дампа файла**

Записать и просмотреть, как выглядит двоичный дамп файла, записанный простой программой. При создании файла **Z3.dat** в него записывается две переменные типа `double`. Затем пишется первый массив элементов типа `int`, а затем второй – типа `char`. Далее вычисляется размер записанного файла и дозаписывается в этот же файл.

*Листинг задачи 3.2. Запись дампа фала*

```
#include <iostream>
#include <fstream>
#include <locale.h>
#include <windows.h>
using namespace std;

int main() {
    setlocale(LC_ALL,"Russian");
    int arr1[]={0,10,4,25,1000,14,0,47,11,1};
    char arr2[]={'a','\0','b'};
    double m = 3.14, n = 0.0003;
    ofstream fin1("Z3.dat", ios::binary); // открытие для записи
            if (!fin1.is_open()){ // проверка
            cout << "Ошибка открытия файла!" << endl;
            system ("pause");
            return 1;
      }
    fin1.write((char *)&m,sizeof(m)); // Запись 8Б - размер m
    fin1.write((char *)&n,sizeof(n)); // Запись 8Б - размер n
// Запись 4Б*10 - значение элемента массива arr1[i]
    for(int i = 0; i < 10; i++) fin1.write((char *)&arr1[i],sizeof(arr1[i]));
// Запись 1Б*3 - значение элемента массива arr2[i]
      for(int i = 0; i < 3; i++){
      fin1.write((char *)&arr2[i],sizeof(arr2[i]));  }
      fin1.close(); // закрытие файла

      ifstream fout("Z3.dat", ios::binary); // открытие для чтения
            if (!fout.is_open()){ // проверка
            cout << "Ошибка открытия файла!" << endl;
            system ("pause");
            return 1;
      }
      fout.seekg(0, ios::end); // переход в конец файла
      long len = fout.tellg(); //текущая позиция в байтах, измеренная от
начала
      cout << len << endl;
      fout.close(); // закрытие файла
// открытие для дозаписи в коней файла
      ofstream fin2("Z3.dat", ios::binary|ios::app);
      if (!fin2.is_open()){
            cout << "Ошибка открытия файла!" << endl;
            system ("pause");
            return 1;
      }
      fin2.write((char *)&len,sizeof(len)); // Запись 4Б - размер файла
      fin2.close(); // закрытие файла
      system ("pause");
}
```

На рис. 3.1 показан файл **Z3.dat**, который открыт в текстовом редакторе в шестнадцатеричном коде. Область слева — смещение от начала файла (в шестнадцатеричном виде), в центре приведены шестнадцатеричные значения байтов файла, по 16Б в строке. Все данные пишутся «младшими байтами вперед», т.е. для получения значения машинного слова байты надо переписать в обратном порядке. В правой части указаны видимые ASCII символы, соответствующие шестнадцатеричным значениям (неотображаемые символы показаны точками).

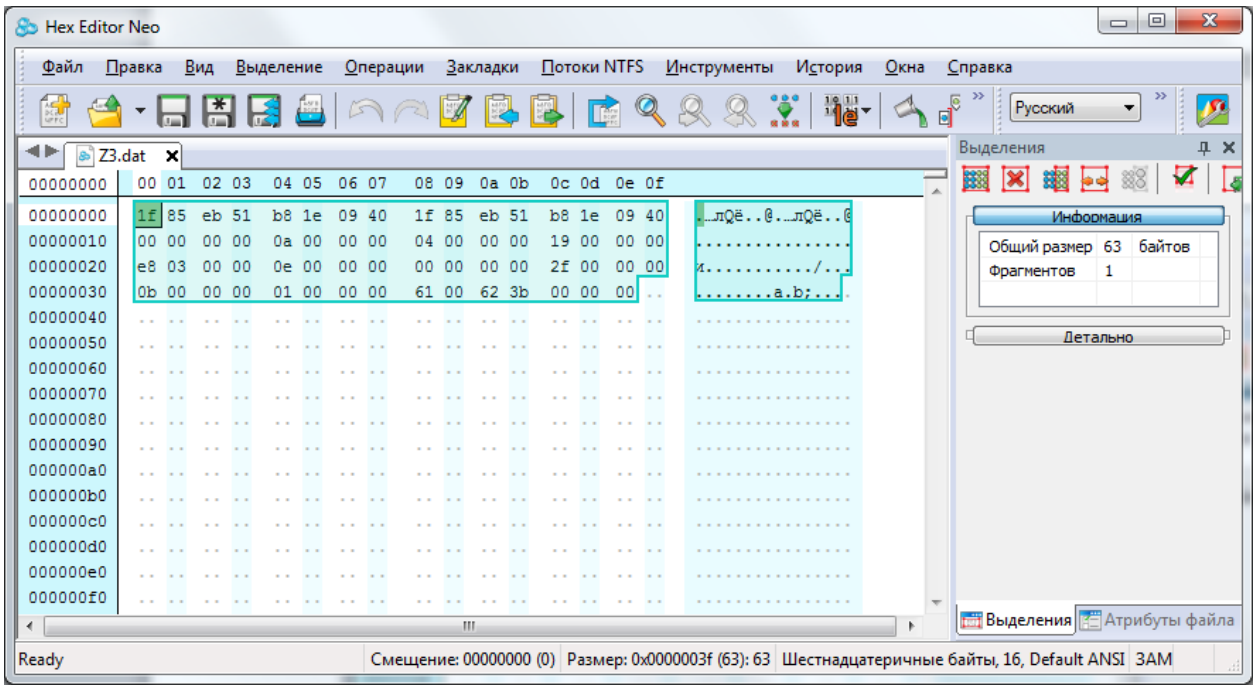


Рис. 3.1. Дамп двоичного файла (в шестнадцатиричной системе счисления)

**Здание для самостоятельного выполнения**

1.2. Запишите дамп файла задачи 3.1.

**Распределение памяти в двоичном файле**

Если содержимое двоичного файла при каждом запуске программы переписывается от начала до конца, то он всегда будет содержать только необходимые данные. Иначе обстоит дело, если двоичный файл при очередном запуске редактируется. Тогда при удалении данных в файле остаются неиспользованные участки. То же самое происходит при изменении (обновлении, в терминологии баз данных – `update`) данных, если их размерность увеличивается: они переписываются в конец файла. Эта свободные участки называются «мусором» и со временем они приводят к значительному превышению объема файла относительно его полезного содержимого. Программа может придерживаться разных стратегий «утилизации мусора»:

- свободные участки объединяются в отдельную структуру данных, например, список, и повторно используются. Недостатком является сложность поддержания в файле дополнительной структуры данных;
- периодически выполняется процедура «сбора мусора», обычно выполняемая в виде сжатия данных: в новом двоичном файле создается аналогичная структура данных и в нее переписываются только актуальные (достижимые) данные из исходного файла. Процедура сжатия может быть вызвана пользователем явно, либо автоматически, на основании косвенных характеристик структур данных в файле.

Изменение данных в файле не может быть выполнено непосредственно. Необходимо создать в памяти переменную, прочитать туда значение из файла, а после изменения записать обратно (обновить).

**Задача 3.3. Редактирование массива целых чисел в двоичном файле**

Написать программу, которая считывает в бинарный файл статический массив целых чисел размернойстью `n`. Затем пользователь должет иметь возможность выбрать редактируемый элемент массива и указать новое значение, которое записывается на место старого.

*Листинг задачи 3.3. Редактирование массива целых чисел в двоичном файле*

```
#include <iostream>
#include <fstream>
#include <cstdio>
#include <cstdlib>
using namespace std;

int main() {
    const int n = 10;
    int arr[n];
    int arr_new, i, pos;
    for(i = 0; i < n; i++){
        arr[i] = rand()%10;
        cout << arr[i] << '\t';
    }
    ofstream fin("Z3_3.txt", ios::binary); // открытие для записи
    if (!fin.is_open()){ // проверка
        cout << "Ошибка открытия файла!" << endl;
        return 1;
    }
    fin.write((char *)&arr,sizeof(arr)); // Запись массива arr
    fin.close(); // закрытие файла
    for(i = 0; i < n; i++){ // обнуление массива
        arr[i] = 0;
        cout << arr[i] << '\t';
    }
      /////// Редактирование ///////
    cout << "\nВведите номер элемента (от 0 до " << n << "): ";
    cin >> i;
    cout << "\nВведите новое значение элемента: ";
    cin >> arr_new;
    // открытие для чтения и записи
    fstream fout("Z3_3.txt", ios::binary|ios::in|ios::out);
    if (!fout.is_open()){ // проверка
        cout << "Ошибка открытия файла!" << endl;
        return 1;
    }
    // конроль записанных данных - отладка
    fout.seekg(0, ios::end); // переход в конец файла
    long len = fout.tellg(); //текущая позиция в байтах, измеренная от начала
    cout << "Размер файла: " << len << "Б" << endl;
    pos =  i*sizeof(int); // адрес редактируемого элемента
    cout << "Позиция редактируемого элемента: " << pos << endl; // отладка
    fout.seekp(pos, ios::beg);
    fout.write((char *)&arr_new,sizeof(arr_new));;    // редактирование arr[i]
    fout.seekg(0); // переход в начало - абсолютное позиционирование
    fout.read((char *)&arr, sizeof(arr)); // отладка
    for(i = 0; i < n; i++){
        cout << arr[i] << '\t';
    }
    fout.close(); // закрытие файла
    system ("pause");
}
```

**Здания для самостоятельного выполнения**

1.3. Написать программы аналогичные задачи 3.3 для:
- динамического массива;
- двухмерного массива.

1.4. Добавить возможность не только редактирования, но и удаления элементов массива.

**Задача 3.4. Сравнение файлов**

Написать программу, которая сравнивает два файла по размеру или по содержимому. Программа должна определить: полную идентичность двух файлов или отличие по размеру или содержимому.

*Листинг задачи 3.4. Сравнение файлов*

```
#include <locale.h>
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
using namespace std;
int const N = 1024;

int main() {
   setlocale(LC_ALL,"Russian");
   ///// выбор файлов для сравнения /////
   ifstream *fin = new ifstream[2];
   for(int i = 0; i < 2; i++){
      cout << "Введите имя файла: ";
      string s;
      cin >> s;
      fin[i].open(s.c_str(), ios::binary | ios::in);
      if (!fin[i].is_open()){ // проверка
         cout << "Ошибка открытия файла!" << endl;
         i--;
      }
   }
   unsigned char buf1[N] , buf2[N];
   cout << "Сравнение файлов...  " << endl;
   do{
      fin[0].read((char *)buf1, sizeof buf1);
      fin[1].read((char *)buf2, sizeof buf2);
      int f1 = fin[0].gcount();// количество прочитанных символов
      int f2 = fin[1].gcount();
      if(f1 != f2){
         cout << "Файлы имею разные размеры." << endl;
         fin[0].close();
         fin[1].close();
         return 0;
      }
      // сравнение содержимого
      for(int j = 0; j < f1; j++){
         if(buf1[j] != buf2[j]){
            cout << "Файлы имею различное содержимое." << endl;
            fin[0].close();
            fin[1].close();
            return 0;
         }
      }
      }while (!fin[0].eof() || !fin[1].eof());
    cout << "Файлы одинаковые." << endl;
    fin[0].close(); // закрытие файла
    fin[1].close();
    system ("pause");
    return 0;
}
```

**Здания для самостоятельного выполнения**

1.5. Проверьте, как работает программа для файлов разных форматов.

1.6. Измените программу так, что бы выполнялось сравнение и по размеру и по содержимому.

### Задания для самостоятельного выполнения

**Вариант 1**

Написать программу, которая сравнивает два текстовых файла и помещает одинаковые фрагменты в третий файл. Кроме этого, программа должна вычислять и выводить на экран процент идентичности первого файла по отношению ко второму.

**Вариант 2**

Написать программу, которая сравнивает по словам (словом считать непрерывную последовательность букв) два текстовых файла и помещает одинаковые слова в результирующий файл. Слова в результирующем файле не должны повторяться. Кроме этого, программа должна вычислять и выводить на экран процент идентичности первого файла по отношению ко второму.

**Вариант 3**

Написать программу, которая сравнивает по предложениям (предложением считать последовательность символов до точки, знаков вопроса или восклицания) два текстовых файла и помещает одинаковые предложения в третий файл. Предложения в результирующем файле не должны повторяться. Кроме этого, программа должна вычислять и выводить на экран процент идентичности первого файла по отношению ко второму.

**Вариант 4**

Написать программу, которая составляет словарь терминов. Термином считать слово, написанное заглавными буквами. Например, в приведенном фрагменте текста термины выделены жирным шрифтом:

> **СИСТЕМЫ ПРОГРАММИРОВАНИЯ** – это комплексы программ и прочих средств, предназначенных для разработки и их эксплуатации на конкретном языке программирования для конкретного вида **ЭВМ**.
>
> Выделяют два вида трансляторов: интерпретаторы и компиляторы.
>
> **ИНТЕРПРЕТАТОР** переводит на язык машинных кодов поочередно каждый оператор исходной программы, проверяет правильность записи оператора и немедленно выполняет его.
>
> В отличие от интерпретатора **КОМПИЛЯТОР** осуществляет перевод на машинный язык всей исходной программы.
>
> Преимуществом компиляторов по сравнению с интерпретаторами является быстродействие, а недостатком – громоздкость. Большинство современных компиляторов работают в режиме трансляции.
>
> В некоторых языках, вместо машинного кода генерируется интерпретируемый двоичный код "виртуальной машины", также называемый **БАЙТ-КОД (BYTE-CODE)**. Такой подход применяется в Forth, Lisp, Java , Perl, Python, а также в языках платформы Microsoft .NET. Например: Программы на Java выполняются в два этапа. Сначала исходный текст компилятором переводится на промежуточный аппаратно-независимый язык. В таком виде полуфабрикат программы (байт-код) хранится на интернет-сервере, откуда по запросу клиента пересылается ему по сети. У клиента байт-код исполняется специальным интерпретатором, этот интерпретатор называется **ВИРТУАЛЬНАЯ JAVA-МАШИНА**, он встроен во все современные браузеры.
>
> **СРЕДА ВИЗУАЛЬНОЙ РАЗРАБОТКИ** – среда разработки программного обеспечения, в которой наиболее распространённые блоки программного кода представлены в виде графических объектов. Применяются для создания прикладных программ и любительского программирование…

Термины должны быть отсортированы по алфавиту.

**Вариант 5**

Написать программу, которая сравнивает два текстовых файла и формирует результирующий файл из несовпадающих фрагментов. Кроме этого, программа должна вычислять и выводить на экран процент первого и второго файла в результирующем файле.

**Вариант 6**

Написать программу, которая сравнивает по словам (словом считать непрерывную последовательность букв) два текстовых файла и помещает слова, которые присутствуют только в одном из файлов, в результирующий файл. Слова в результирующем файле не должны повторяться. Кроме этого, программа должна вычислять и выводить на экран процент слов первого и второго файла в результирующем файле.

**Вариант 7**

Написать программу, которая сравнивает по предложениям (предложением считать последовательность символов от начала (файла, строки, символа разделителя предложений) до точки, знаков вопроса или восклицания) два текстовых файла и помещает предложения, которые присутствуют только в одном из файлов, в результирующий файл. Предложения в результирующем файле не должны повторяться. Кроме этого, программа должна вычислять и выводить на экран процент предложений первого и второго файла в результирующем файле.

**Вариант 8**

Написать программу, которая считывает из файла все даты в формате `дд.мм.гггг` (например, 16.01.2000) и помещает в результирующий файл строку: `<количество повторений даты – дата>`. Кроме этого программа должна вывести на экран дату с наибольшим числом повторений.

**Вариант 9**

Написать программу, которая считывает текст из файла и определяет, сколько каждое из слов первого файла встретилось во втором. Результат должен сохраняться в результирующем файле в отсортированном виде по количеству повторений.

**Вариант 10**

Написать программу, которая считывает из файла все даты в формате `дд месяц гггг` (например, 16 января 2000 г.) и помещает в результирующий файл строку: `<количество повторений даты – дата>`. Кроме этого программа должна вывести на экран дату с наибольшим числом повторений.

**Вариант 11**

Написать программу, которая считывает из файла все века, написанные римскими цифрами (например, XIX в., V в.) и помещает в результирующий файл строку: `<количество повторений даты – дата>`. Кроме этого программа должна вывести на экран дату с наибольшим числом повторений.

**Вариант 12**

Написать программу, которая считывает из файла все даты в формате `<день недели, дд месяц гггг>` (например, "Tue, 14 Nov 2000") и помещает в результирующий файл строку: `<количество повторений дня недели – день недели>`. Кроме этого программа должна на экран вывести день с наибольшим числом повторений.

**Вариант 13**

Написать программу, которая считывает из файла все данные с размерностью килограмм (например, 5 кг, 1 кг, 124 кг) и помещает эти данные в результирующий файл в порядке возрастания, находит сумму все чисел и вносит ее в файл в итоговую строку.

**Вариант 14**

Написать программу, которая считывает из файла все даты в формате <день недели, дд месяц гггг> (например, "Tue, 14 Nov 2000") и помещает в результирующий файл строку: <количество повторений месяца – месяц>. Кроме этого программа должна вывести на экран месц с наибольшим числом повторений.

**Вариант 15**

Написать программу, которая считывает из файла все даты в формате дд.мм.гггг (например, 16.01.2000) и помещает в результирующий файл строку: <количество повторений даты – год>. Кроме этого программа должна вывести на экран год с наибольшим числом повторений.

**Вариант 16**

Исходный текстовый файл содержит некий текст. Файл отправляют получателю. Написать программу, которая сравнивает исходный файл с полученным и определяет, были ли внесены в него изменения. Если изменения были внесены, то выводит в файл отчета внесенные изменения в формате <№ строки: исходный символ – измененный символ>.

**Вариант 17**

Исходный файл содержит некий набор чисел. Файл отправляют получателю. Написать программу, которая сравнивает исходный файл с полученным и определяет, были ли внесены в него изменения. Если изменения были внесены, то выводит в файл отчета внесенные изменения в формате <индекс (№ числа): исходное число – измененное число>.

**Вариант 18**

Файл-ключ содержит элементы замены в формате <ключ – значение>. Написать программу, которая во всем тексте указанного файла заменяет все ключи их значениями.

**Вариант 19**

Файл-ключ содержит элементы замены в формате <символ – текст>. Написать программу, которая во всем тексте указанного файла заменяет все символы соответствующим текстом.

**Вариант 20 (*)**

В файле **shakespeare_plays.json** (архив **shakespeare_plays.json.zip**) содержатся пьессы Шекспира в json-формате. Написать программу, которая помещает в результирующий файла название пьессы – номер строки. Затем пользователю выводится на экран список пьесс в формате <№ – название пьессы>. Пользователь выбирает по номеру пьессу и ему выводится в файл отчета слудующая информация:

- название;
- список персонажей пьессы – количество реплик (ключ "says").

**Примечание**

Под репликой понимается весь текст персонажа, после которого должен говорить свою роль другой.

**Лабораторная работа № 4. Задание для самостоятельного выполнения**

Цель работы

- научится вести поиск необходимой информации.

**Технология выполнения работы**

**Задание 1**

Напишите программы для анализа содержимого фала **mails.json** (архив **mails.zip**) и получите ответы на следующие вопросы. Результаты оформите в виде реферата.

Переписка сотрудников компании Enron (файл **mails.zip**). Этот файл содержит переписку примерно 150 пользователей, в основном руководителей компании Enron. Поскольку полный датасет занимает в распакованном виде около 1,5 Гбайт, из него удалили поле `body` для уменьшения размера.

Файл содержит строки следующего вида:


```
{"_id" : ObjectId("4f16fc97d1e2d32371003e27"),
  "subFolder" : "notes_inbox",
  "mailbox" : "bass-e",
  "filename" : "450.",
  "headers" : {
      "X-Folder" : "\\Eric_Bass_Dec2000\\Notes Folders\\Notes inbox",
      "X-bcc" : "",
      "To" : "eric.bass@enron.com",
      "X-From" : "Michael Simmons",
      "Message-ID" : "<6884142.1075854677416.JavaMail.evans@thyme>",
      "Mime-Version" : "1.0",
      "From" : "michael.simmons@enron.com",
      "X-To" : "Eric Bass",
      "X-cc" : "",
      "Content-Transfer-Encoding" : "7bit",
      "X-Origin" : "Bass-E",
      "Subject" : "Re: Plays and other information",
      "X-FileName" : "ebass.nsf",
      "Date" : "Tue, 14 Nov 2000 08:22:00 -0800 (PST)",
      "Content-Type" : "text/plain; charset=us-ascii"}
}
```


**Строка:**


```
{ "_id" : { "$oid" : "4f16fc97d1e2d32371003e27" }, "subFolder" :
"notes_inbox", "mailbox" : "bass-e", "filename" : "450.", "headers" :
{ "X-cc" : "", "From" : "michael.simmons@enron.com", "Subject" : "Re:
Plays and other information", "X-Folder" : "\\Eric_Bass_Dec2000\\Notes
Folders\\Notes inbox", "Content-Transfer-Encoding" : "7bit", "X-bcc" :
"", "To" : "eric.bass@enron.com", "X-Origin" : "Bass-E", "X-FileName"
: "ebass.nsf", "X-From" : "Michael Simmons", "Date" : "Tue, 14 Nov
2000 08:22:00 -0800 (PST)", "X-To" : "Eric Bass", "Message-ID" :
"<6884142.1075854677416.JavaMail.evans@thyme>", "Content-Type" :
"text/plain; charset=us-ascii", "Mime-Version" : "1.0" } }
```


1. Определите общее количество писем (ключ "_id ").
2. Определите количество уникальных отправителей (ключ "From").
3. Для каждого дня недели посчитайте, сколько писем было отправлено на адрес ebass@enron.com (ключ "To")?
4. Проанализируйте переписку Shanna Husser и Eric Bass. Сколько писем каждый из них отправил другому?
5. Laurie Ellis иногда посылает письма с одинаковыми темами (ключ "Subject"). Посчитайте для каждой темы письма, сколько раз она была использована в 2000 году.
6. В какой день недели отправлено максимальное количество писем?

**Лабораторная работа № 5. Структуры**

Цель работы

- ознакомиться со способами описания структур;
- ознакомиться со способами ввода-вывода содержимого структур;
- научиться применять полученные знания для решения практических задач.

## Теоретические сведения

Структуры в C++ обладают практически теми же возможностями, что и классы, но чаще их применяют исключительно для группировки (объединения) данных. В отличие от массива, в структуру можно объединять данные различных типов. Например, требуется обрабатывать информацию о расписании работы конференц-зала, и для каждого мероприятия надо знать время, тему, фамилию организатора и количество участников. Поскольку все это относится к одному событию, логично дать ему общее имя. Для этого описывается новый тип данных, например, с именем `Event`. Переменные этого типа можно описать так же, как переменные встроенных типов:

```
struct Event {
   int hour, min, num;
   char theme[100], name[100];
}; // обратите внимание: точка с запятой после описания обязательна!
Event e1, e2[10]; // описание структуры и массива структур
```

Если структура используется только в одном месте программы, можно совместить описание типа с описанием переменных, при этом имя типа можно не указывать:

```
struct {
   int hour, min, num;
   char theme[100], name[100];
} e1, e2[10];
```

Переменные структурного типа можно размещать и в динамической области памяти, для этого надо описать указатель на структуру и выделить под нее место:

```
Event *pe = new Event; // структура
Event *pm = new Event[m]; // массив структур
```

Элементы структуры называются полями. Поля могут быть любого основного типа, массивом, указателем, объединением или структурой. Для обращения к полю используется операция выбора («точка» для переменной и `->` для указателя):

```
e1.hour = 12;
e1.min = 30;
strncpy(e2[0].theme, "Проблемы использования решений класса «Big
Data» в бизнесе", 99);
pe->num = 30; // или (*pe).num = 30;
pm[2].hour = 14;    // или (*(pm + 2)).hour = 14;
```

Структуры (не-динамические) можно инициализировать:

```
Event e3 = {12, 30, "Проблемы использования решений класса «Big
Data» в бизнесе", 25};
```

Структуры одного типа можно присваивать друг другу:

```
*pe = e1;
pm[1] = e1;
pm[4] = e2[0];
```

*Присваивание* – это все, что можно делать со структурами целиком. Другие операции, например, сравнение на равенство и вывод, не определены. Впрочем, можно задать их самостоятельно, поскольку в структурах, как и в классах, разрешается определять свои операции.

Ввод-вывод структур выполняется поэлементно. Вот, например, как выглядит ввод и вывод описанной выше структуры `e1`:

```
cin >> e1.hour >> e1.min; /* с использованием классов: подключить
<iostream> */
```

```
cin.getline(e1.theme, 100);
cout << e1.hour << ' ' << e1.min << ' ' << e1.theme << endl;

// в стиле C: подключить <cstdio>:
scanf("%d%d", &e1.hour, &e1.min);
gets(e1.theme);
printf("%d %d %s", e1.hour, e1.min, e1.theme);
```

## Технология выполнения работы

1. Реализуйте решение задач 5.1-5.4 с использованием языка C++.

**Задача 5.1. Поиск в простой базе (массиве структур)**

В текстовом файле хранится база отдела кадров предприятия. На предприятии 100 сотрудников. Каждая строка файла содержит запись об одном сотруднике. Формат записи: ФИО (30 позиций, фамилия должна начинаться с первой позиции), год рождения (5 позиций), оклад (10 позиций). Написать программу, которая по заданной фамилии выводит на экран сведения о сотруднике, подсчитывая средний оклад всех запрошенных сотрудников.

*I. Исходные данные и результаты.*

База сотрудников находится в текстовом файле. Поиск по базе будет выполняться многократно, поэтому всю информацию желательно хранить в оперативной памяти, поскольку многократное чтение из файла крайне нерационально. Количество строк файла по условию задачи ограничено, поэтому можно выделить для их хранения массив из 100 элементов. Каждый элемент массива будет содержать сведения об одном сотруднике. Поскольку эти сведения разнородные, удобно организовать их в виде структуры.

**Примечание**

Строго говоря, для решения этой конкретной задачи запись о сотруднике может быть просто строкой символов, из которой при необходимости выделяется подстрока с окладом, преобразуемая затем в число, но мы для общности и удобства дальнейшей модификации программы будем использовать структуру.

В результате работы программы необходимо вывести на экран требуемые элементы исходного массива. Поскольку эти результаты являются выборкой из исходных данных, дополнительная память для них не отводится. Для подсчета среднего оклада опишем переменную вещественного типа.

*Листинг 5.1. Поиск в массиве структур*

```
#include <iostream>
#include <locale.h>
#include <fstream>
#include <cstring>
#include <windows.h>
#include <cstdlib>
using namespace std;

int main() {
    setlocale(LC_ALL,"Russian");
    const int l_name = 35, l_year = 5, l_pay = 5;// длины полей строки файла
    // длина буфера - сумма длин полей struct
    const int l_buf = l_name + l_year + l_pay;
    struct Sotr { // структура для хранения сведений об одном сотруднике
        char name[l_name + 1];      // длина поля задана с учетом нуль-символа
        int birth_year;
        float pay;
    };
    const int l_dbase = 100;
    Sotr dbase[l_dbase]; // массив структур для хранения всей базы
    char buf[l_buf+1]; // буфер для ввода строки из файла
    char name[l_name+1]; // строка для ФИО запрашиваемого сотрудника
```

*Продолжение листинга 5.1*

```
    //////////// Шаг 1. Открытие файла и чтение данных ////////////
    ifstream fin("db_otd-kadr.txt");
    if (!fin.is_open()){
        cout << "Ошибка открытия файла!" << endl;
        system ("pause");
        return 1;
    }
    int i = 0;
    while (!fin.eof()){      // пока не конец файла
        fin.getline(buf, l_buf);
        strncpy(dbase[i].name, buf, l_name);
        dbase[i].name[l_name] = '\0';
        dbase[i].birth_year = atoi(&buf[l_name]);
        dbase[i].pay = atof(&buf[l_name + l_year]);
        i++;
        if (i > l_dbase){
            puts("Файл слишком длинный!\n");
            system ("pause");
            return 1;
        }
    }
    fin.close();
    //////////// Шаг 2. Поиск сотрудников ////////////
    int n_record = i, n_sotr = 0; /* n_record - количество записей о
сотрудниках, n_sotr - количество сотрудников, о которых выводятся данные */
    float sum_pay = 0; // сумма окладов
    while (true){
        cout << "Введите фамилию для поиска (end - для завершения поиска): ";
        cin >> name;
    //  преобразует строку с заданного разработчиком набора символов в ANSI
        OemToChar(name, name);
        if (strcmp(name, "end")==0) break;
        bool not_found = true;
        for (int i = 0; i < n_record; i++){
            if (strstr(dbase[i].name, name))
                if(dbase[i].name[strlen(name)] == ' '){
                    cout << dbase[i].name << dbase[i].birth_year << "\t" <<
dbase[i].pay << endl;
                    n_sotr++;
                    sum_pay += dbase[i].pay;
                    not_found = false;
                }
        }
        if (not_found) cout << "Такого сотрудника нет!" << endl;
    }
    if (n_sotr > 0) cout << "Средний оклад: " << sum_pay/n_sotr << endl;
    system ("pause");
}
```

Необходимо предусмотреть случай, когда одна фамилия является частью другой (например, Иванов и Ивановский). В конце файла не должно быть пустых строк. Не забудьте проверить, выдается ли диагностическое сообщение, если файл не найден. Для формирования полей структуры используются функции `strncpy`, `atoi` и `atof`. Обратите внимание, что завершающий нуль-символ в поле фамилии заносится «вручную», поскольку функция `strncpy` делает это только в случае, если строка-источник короче строки-приемника. В функцию `atoi` передается адрес начала подстроки, в которой находится год рождения.

**Примечание**

При заполнении массива из файла обязательно контролируйте выход за границы массива и при необходимости выдавайте предупреждающее сообщение.

### Здание для самостоятельного выполнения

1.1. Организуйте выход из бесконечного цикла ввода фамилий по нажатию клавиши *Esc*. Цикл поиска сотрудников по фамилии организован как бесконечный с принудительным выходом (`end - для завершения поиска`). Некоторые специалисты считают, что такой способ является плохим стилем и для выхода из цикла следует определить переменную-флаг, но нам кажется иначе. Впрочем, в данном случае выход из цикла действительно организован не лучшим образом. Более удобным был бы выход по нажатию, например, клавиши Esc. К сожалению, в рамках стандарта это сделать невозможно, однако в состав библиотек Visual Studio.NET входит библиотека консольного ввода-вывода, описанная в заголовочном файле `<conio.h>`, и в ней есть функция `_getch`, позволяющая анализировать код нажатой клавиши.

### Здание на дополнительные баллы

1.2. Крупным недостатком нашей программы является то, что вводить фамилию сотрудника требуется именно в том регистре, в котором она присутствует в базе. Для преодоления этого недостатка необходимо перед сравнением фамилий переводить все символы в один регистр. Для символов латинского алфавита в библиотеке есть функции `tolower` и `toupper`, переводящие переданный им символ в нижний и верхний регистры соответственно. Напишите аналогичные функции для символов русского алфавита и устраните указанный недостаток.

## Задача 5.2. Сортировка массива структур

Написать программу, которая упорядочивает описанный в задаче 5.1 файл по году рождения сотрудников.

Изменим предыдущую программу таким образом, чтобы она вместо поиска упорядочивала массив, а затем записывала его в исходный файл. Для сортировки применим метод выбора (Selection sort). При всей своей простоте он достаточно эффективен. В листинге 5.2 опущено считывание базы, совпадающее с листингом 5.1 (*Шаг 1. Открытие файла и чтение данных*) и приводится только фрагмент, выполняющий сортировку и вывод.

**Примечание**

Будьте аккуратны при отладке программ, изменяющих входные файлы: следует либо перед запуском программы создать копию исходного файла, либо открывать выходной файл с другим именем, а заменять его на имя, требуемое по заданию, только после того, как удалось убедиться в полной работоспособности программы.

*Листинг 5.2. Сортировка массива структур*

```
…
//////////// Шаг 2. Сортировка по дате ////////////
    ofstream fout( "db_otd-kadr.txt" );
    if (!fout) {
        cout << "Ошибка открытия файла!" << endl;
        return 1;
    }
    for (int i = 0; i < l_dbase-1; i++){
        int min_i = i;
        for (int j = i+1; j < l_dbase; j++)
            if (dbase[j].birth_year < dbase[min_i].birth_year) min_i = j;
        // обмен элементами структур
        Sotr tmp = dbase[i];
        dbase[i] = dbase[min_i];
        dbase[min_i] = tmp;
  }
```

*Продолжение листинга 5.2*

```
    for (int i = 0; i < l_dbase; i++){
        fout << dbase[i].name << dbase[i].birth_year << "\t" << dbase[i].pay
<< endl;
    }
    fout.close();
    …
```

Элементами массива в данной задаче являются структуры. Для структур одного типа определена операция присваивания, поэтому обмен двух элементов массива структур выглядит точно так же, как для основных типов данных.

Чтобы записать результаты в тот же файл, файл, открытый для чтения, закрывается, и открывается файл с тем же именем для записи (точнее, создается объект выходного потока `ostream` с именем `fout`). При этом старый файл на диске уничтожается и создается новый пустой файл, куда и производится запись массива.

**Здание для самостоятельного выполнения**

1.3. Написать программу, которая упорядочивает описанный в задаче 5.1 файл по году рождения и окладу.

**Задача 5.3. Структуры и бинарные файлы**

Написать две программы. Первая считывает информацию из файла, формат которого описан в задаче 5.1, и записывает ее в бинарный файл. Количество записей в файле не ограничено. Вторая программа по номеру записи корректирует оклад сотрудника в этом файле.

Обе программы не представляют алгоритмических сложностей, поэтому мы сразу приведем тексты (листинги 5.3-1 и 5.3-2). Для краткости работа с файлами выполнена «в стиле С». Хранить в памяти весь файл нет необходимости, вполне достаточно одной переменной структурного типа, в которой будет содержаться в каждый момент времени запись об одном сотруднике.

*Листинг 5.3-1. Создание бинарного файла из текстового*

```
#include <locale.h>
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;

int main() {
    setlocale(LC_ALL,"Russian");
    const int l_name = 35;    // длины поля ФИО
    struct {      // структура для хранения сведений об одном сотруднике
        char name[l_name + 1];      // длина поля задана с учетом нуль-символа
        int birth_year;
        float pay;
    } Sotr;
    /////////////// Шаг 1. Открытие файлов ////////////
    FILE *fin, *fout; // указатели на файлы
    fin = fopen("db_otd-kadr.txt", "r");
    if (fin == NULL){
        puts("Ошибка открытия файла db_otd-kadr.txt!");
        return 1;
    }
    // создание нового файла db_otd-kadr.bin, b - бинарный режим
    fout = fopen("db_otd-kadr.bin", "wb");
    if (fout == NULL){
        puts("Ошибка открытия файла db_otd-kadr.bin!");
        return 1;
    }
```

*Продолжение листинга 5.3-1*

```
//////////// Шаг 2. Запись в бинарный файл ////////////
    while (!feof(fin)) { // читаем файл, пока не встретится конец файла
        fgets(Sotr.name, l_name, fin); /* читаем строку из текстового файла и
записываем в структуру
        читаем информацию из текстового файла, преобразуем ее в числовую
форму и записываем в структуру:*/
        fscanf(fin, "%i%f\n", &Sotr.birth_year, &Sotr.pay);
        // отладка
        printf("%s\t%i\t%f\n", Sotr.name, Sotr.birth_year, Sotr.pay);
        fwrite(&Sotr, sizeof(Sotr), 1, fout); // вывод в бинарный файл
    }
    fclose(fout);
    fclose(fin);
    puts("Бинарный файл записан!");
    system ("pause");
}
```

Функция `feof()` возвращает не 0, если достигнут конец файла, в противном случае она возвращает 0. Для чтения из файла строки используются функции `fgets()`. Функция `fscanf()` читает информацию из текстового файла и преобразует ее во внутреннее представление данных в памяти компьютера. Информация о количестве читаемых элементов, их типах и особенностях представления задается с помощью формата. В случае функции ввода формат – это строка, содержащая описания одного или нескольких вводимых элементов.

Для формирования записи в бинарный файл здесь применяется функция:

```
size_t fwrite(const void *p, size_t size, size_t n, FILE *f)
```

`fwrite()` буфер – это указатель на информацию, записываемую в файл. Она записывает `n` элементов длиной `size` байт из буфера, заданного указателем `p`, в поток `f`. Возвращает число записанных элементов.

Для чтения из бинарного файла будет использоваться функция `fread`:

```
size_t fread( void *p, size_t size, size_t n, FILE *f )
```

Она считывает `n` элементов длиной `size` байт в буфер, заданный указателем `p`, из потока `f`. Возвращает число фактически считанных элементов.

*Листинг 5.3-2. Корректировка бинарного файла*

```
#include <locale.h>
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;

int main() {
    setlocale(LC_ALL,"Russian");
    const int l_name = 35;    // длины поля ФИО
    struct {      // структура для хранения сведений об одном сотруднике
        char name[l_name + 1]; // длина поля задана с учетом нуль-символа
        int birth_year;
        float pay;
    } Sotr;
    //////////// Шаг 1. Открытие файла ////////////
    FILE *fout; // указатель на файл
    // открытие файла db_otd-kadr.bin, r+ - чтение и запись, b - бинарный режим:
    fout = fopen("db_otd-kadr.bin", "r+b");
    if (fout == NULL){
        puts("Ошибка открытия файла db_otd-kadr.bin!");
        return 1;
    }
```

*Продолжение листинга 5.3-2*

```
//////////// Шаг 2. Определение количества записей в файле ////////////
    fseek(fout, 0, SEEK_END); // перейти в конец файла
    // текущая позизия в байтах, измеренная от начала файла
    int n_record = ftell(fout)/sizeof(Sotr);
    // контрольный вывод
    cout << "Количество струдников: " << n_record << endl;
    //////////// Шаг 3. Обновление ////////////
    int num; // номер записи, вводит пользователь
    int y; // вспомогательная переменная для проверки ввода
    puts("Введите номер сотрудника (для выхода из программы введите -1): ");
    while(y!=1){
        y = scanf("%i", &num); // если ввод успешный, то y = 1, если нет то y = 0
        fflush(stdin); // функция fflush очищает содержимое буфера
        if (num != -1 && (y == 0 || num < 0 || num >= n_record)){
            puts("Ожидается номер сотрудника:");
            y = 0;
        }
        else if (num == -1) break;
        // установка текущей позиции в файле
        fseek(fout, num*sizeof(Sotr), SEEK_SET);
        // считывание одной записи в структуру Sotr
        fread(&Sotr, sizeof(Sotr), 1, fout);
        // контрольный вывод
        printf("%s\t%i\t%f\n", Sotr.name, Sotr.birth_year, Sotr.pay);
        puts("Введите новый оклад:");
        scanf("%f", &Sotr.pay); // ввод нового оклада
        // установка позиции в файле в начало считанной записи:
        fseek(fout, num*sizeof(Sotr), SEEK_SET);
        // запись в файл данных из структуры Sotr
        fwrite(&Sotr, sizeof(Sotr), 1, fout);
        // вывод результата на экран
        printf("%s\t%i\t%f\n", Sotr.name, Sotr.birth_year, Sotr.pay);
        y = 0; // возврат для ввода нового номера сотрудника
        puts("Введите номер сотрудника (для выхода из программы введите -1): ");
    }
    fclose(fout);
    puts("Корректировка завершена!");
    system ("pause");
}
```

В шаге 1 открывается сформированный в предыдущей задаче бинарный файл. Обратите внимание на режим открытия: `r+` обозначает возможность чтения и записи. Чтобы проконтролировать введенный номер записи, в переменную `n_record` заносится длина файла в записях.

Для прерывания непрерывного цикла при организации проверки ввода используется функция `fflush(stdin)` (*Шаг 3. Обновление*). Если поток открыт для ввода, то функция `fflush` очищает содержимое буфера. После вызова функции поток остается открытым. Для небуферизованного потока применение этой функции не эффективно.

В цикле корректировки оклада текущая позиция в файле устанавливается дважды, поскольку после первого чтения она смещается на размер считанной записи.

**Здание для самостоятельного выполнения**

1.4. Установите контроль вводимого нового оклада – не должен превышать или быть меньше старого более чем на 50%.

1.5. Организуйте возможность повторения выбора сотрудников, пока пользователь сам не пожелает завершить редактирование.

1.6. Объедините оба решения задачи в одну программу с возможностью выбора пользователем действий:
   1. редактирование оклада;
   2. редактирование ФИО;
   3. редактирование всей записи.

**Задача 5.4. Структуры в динамической памяти**

Вывести на экран содержимое бинарного файла, сформированного в задаче 5.1, упорядочив фамилии сотрудников по алфавиту.

*Листинг 5.4. Структуры в динамической памяти*

```
#include <locale.h>
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
using namespace std;
const int l_name = 35;  // длины поля ФИО
struct Sotr{      // структура для хранения сведений об одном сотруднике
    char name[l_name + 1];    // длина поля задана с учетом нуль-символа
    int birth_year;
    float pay;
};
int compare(const void *str1, const void *str2); // сравнение строк
int main() {
    setlocale(LC_ALL,"Russian");
    //////////// Шаг 1. Открытие файла ////////////
    FILE *fbin; // указатель на файл
    // открытие файла db_otd-kadr.bin, r - чтение, b - бинарный режим:
    fbin = fopen("db_otd-kadr.bin", "rb");
    if (fbin == NULL){
        puts("Ошибка открытия файла db_otd-kadr.bin!");
        return 1;
    }
    //////////// Шаг 2. Определение количества записей в файле ////////////
    fseek(fbin, 0, SEEK_END);
    int n_record = ftell(fbin)/sizeof(Sotr);
    cout << "Количество сотрудников: " << n_record << endl;
    //////////// Шаг 3. Сортировка и печать ////////////
    // выделение памяти под весь массив структур
    Sotr *st = new Sotr[n_record];
    fseek(fbin, 0, SEEK_SET); // установка текущей позиции в начало файла
    // считывание файла целиком в массив
    fread(st, sizeof(Sotr), n_record, fbin);
    fclose(fbin);
    // сортировка массива структур
    qsort(st, n_record, sizeof(Sotr), compare);
    for(int i = 0; i < n_record; i++)  // вывод на экран
        printf("%s\t%i\t%f\n", st[i].name, st[i].birth_year, st[i].pay);
    system ("pause");
}
int compare(const void *str1, const void *str2) { // сравнение строк
    // лексиграфическое сравнение двух строк
    return strcmp(((Sotr *)str1)->name, ((Sotr *)str2)->name);
}
```

Для сортировки использовалась стандартная функция `qsort`, прототип которой находится в заголовочном файле `<cstdlib>`. Она может выполнять сортировку массивов любых размеров и типов. У нее четыре параметра: указатель на начало области, в которой размещается сортируемая информация; количество сортируемых элементов; размер каждого элемента в байтах; имя функции, которая выполняет сравнение двух элементов.

Так как функция qsort универсальна, ее надо снабдить информацией о том, как сравнивать сортируемые элементы и какие выводы делать из сравнения. Необходимо написать функцию, которая сравнивает два любых элемента, и передать ее в qsort. Имя функции может быть любым. Мы назвали ее compare. Для работы qsort требуется, чтобы наша функция имела два параметра — указатели на сравниваемые элементы типа void. Функция должна возвращать значение, меньшее нуля, если первый элемент меньше второго, равное нулю, если они равны, и большее нуля, если первый элемент больше. При этом массив будет упорядочен по возрастанию. Если нужно упорядочить массив по убыванию, следует изменить возвращаемые значения так: если первый элемент меньше второго, возвращать значение, большее нуля, а если больше — меньшее.

Внутри функции надо привести указатели на void к типу указателя на структуру Sotr. Для этого мы использовали операцию приведения типа в стиле C (Sotr *).

Чтобы описание структуры было известно в функции compare, описание структуры и необходимой ей константы l_name перенесено в глобальную область.

Для упорядочивания массива по другому полю необходимо изменить функцию сравнения. Вот, например, как она выглядит при сортировке по возрастанию года рождения:

```
int compare(const void *by1, const void *by2) {
   int p;
   if (((Sotr *)by1)->birth_year < ((Sotr *) by2)->birth_year)
      p = -1;
   else if (((Sotr *) by1)->birth_year == ((Sotr *) by2)->birth_year)
      p = 0;
   else p = 1;
   return p;
}
```

Можно записать то же самое более компактно с помощью тернарной условной операции:

```
int compare(const void *by1, const void *by2) {
   return ((Sotr *)by1)-> birth_year > ((Sotr *)by2)-> birth_year ? -1 :
   ((Sotr *)by1)-> birth_year == ((Sotr *)by2)->pay ? 0 : 1;
}
```

Как видите, использование стандартных функций сокращает размер программы, но требует некоторых усилий по изучению интерфейса функций.

### Здание для самостоятельного выполнения

1.7. Добавьте в решение задачи 5.4 возможность пользователю выбирать требуемую сортировку по любому полю, по возрастанию или убыванию с последующим выводом результата на экран. Возможность выбора – многократная.

**Примечание**

Для избегания утечки памяти при многократном выделении памяти под динамические массивы необходимо ее освобождать после использования:

```
delete [] myArray;
```

## Задания для самостоятельного выполнения

**Вариант 1**

Описать структуру с именем STUDENT, содержащую следующие поля: ФИО; номер группы; успеваемость (массив из пяти элементов). Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по номеру группы, по ФИО студента, по среднему баллу;
- вывод на экран ФИО и номеров групп для всех студентов, средний балл которых больше 4 (если таких студентов нет, выводится соответствующее сообщение).

**Вариант 2**

Описать структуру с именем STUDENT, содержащую следующие поля: ФИО; номер группы; успеваемость (массив из пяти элементов). Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по номеру группы, по ФИО студента, по среднему баллу;
- вывод на экран ФИО и номеров групп для всех студентов, имеющих только оценки 4 и 5, (если таких студентов нет, выводится соответствующее сообщение).

**Вариант 3**

Описать структуру с именем STUDENT, содержащую следующие поля: ФИО; номер группы; успеваемость (массив из пяти элементов). Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по номеру группы, по ФИО студента, по среднему баллу;
- вывод на экран ФИО и номеров групп для всех студентов, имеющих хотя бы одну оценку 2, (если таких студентов нет, выводится соответствующее сообщение).

**Вариант 4**

Описать структуру с именем AEROFLOT, содержащую следующие поля: название пункта назначения рейса; номер рейса; тип самолета. Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по номеру рейса, по пункту назначения, по типу самолета;
- вывод на экран информации об имеющихся рейсах, вылетающих в пункт назначения, название которого введено пользователем с клавиатуры (если таких рейсов нет, выводится соответствующее сообщение).

**Вариант 5**

Описать структуру с именем AEROFLOT, содержащую следующие поля: название пункта назначения рейса; номер рейса; тип самолета. Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по номеру рейса, по пункту назначения, по типу самолета;
- вывод на экран информации об имеющихся рейсах, которые обслуживаются выбранным пользователем типом самолета (если таких рейсов нет, выводится соответствующее сообщение).

**Вариант 6**

Описать структуру с именем WORKER, содержащую следующие поля: ФИО работника; название занимаемой должности; год поступления на работу. Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по должности, по ФИО, по году поступления на работу;
- вывод на экран информации о работниках, чей стаж работы в организации превышает значение, введенное с клавиатуры пользователем (если таких работников нет, выводится соответствующее сообщение).

**Вариант 7**

Описать структуру с именем TRAIN, содержащую следующие поля: название пункта назначения; номер поезда; время отправления. Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по названиям пунктов назначения, по номеру поезда, по времени отправления;
- вывод на экран информации о поездах, отправляющихся после введенного пользователем с клавиатуры времени (если таких поездов нет, выводится соответствующее сообщение).

**Вариант 8**

Описать структуру с именем TRAIN, содержащую следующие поля: название пункта назначения; номер поезда; время отправления. Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по названиям пунктов назначения, по номеру поезда, по времени отправления;
- вывод на экран информации о поездах, направляющихся в пункт, название которого введено пользователем (если таких поездов нет, выводится соответствующее сообщение).

**Вариант 9**

Описать структуру с именем TRAIN, содержащую следующие поля: название пункта назначения; номер поезда; время отправления. Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по названиям пунктов назначения, по номеру поезда, по времени отправления;
- вывод на экран информации о поезде, номер которого введен пользователем (если таких поездов нет, выводится соответствующее сообщение).

**Вариант 10**

Описать структуру с именем MARSH, содержащую следующие поля: название начального пункта маршрута; название конечного пункта маршрута; номер маршрута. Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по названиям начальных пунктов маршрута, по названиям конечных пунктов маршрута, по номеру маршрута;
- вывод на экран информации о маршруте, номер которого введен пользователем (если таких маршрутов нет, выводится соответствующее сообщение).

**Вариант 11**

Описать структуру с именем MARSH, содержащую следующие поля: название начального пункта маршрута; название конечного пункта маршрута; номер маршрута. Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по названиям начальных пунктов маршрута, по названиям конечных пунктов маршрута, по номеру маршрута;
- вывод на экран информации о маршрутах, которые начинаются или оканчиваются в пункте, название которого введено пользователем (если таких маршрутов нет, выводится соответствующее сообщение).

**Вариант 12**

Описать структуру с именем NOTE, содержащую следующие поля: фамилия, имя; номер телефона; дата рождения (массив из трех чисел). Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по ФИО, по номерам телефонов, по датам рождения;
- вывод на экран информации о человеке, номер телефона которого введен пользователем (если такого нет, выводится соответствующее сообщение).

**Вариант 13**

Описать структуру с именем NOTE, содержащую следующие поля: фамилия, имя; номер телефона; дата рождения (массив из трех чисел). Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по ФИО, по номерам телефонов, по датам рождения;
- вывод на экран информации о людях, чьи дни рождения приходятся на месяц, значение которого введено пользователем (если таких нет, выводится соответствующее сообщение).

**Вариант 14**

Описать структуру с именем NOTE, содержащую следующие поля: фамилия, имя; номер телефона; дата рождения (массив из трех чисел). Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по ФИО, по номерам телефонов, по датам рождения;
- вывод на экран информации о человеке, чья фамилия введена пользователем (если такого нет, выводится соответствующее сообщение).

**Вариант 15**

Описать структуру с именем ZNAK, содержащую следующие поля: фамилия, имя; знак Зодиака; дата рождения (массив из трех чисел). Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по ФИО, по знаку Зодиака (по календарю), по датам рождения;
- вывод на экран информации о человеке, чья фамилия введена пользователем (если такого нет, выводится соответствующее сообщение).

**Вариант 16**

Описать структуру с именем ZNAK, содержащую следующие поля: фамилия, имя; знак Зодиака; дата рождения (массив из трех чисел). Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по ФИО, по знаку Зодиака (по календарю), по датам рождения;
- вывод на экран информации о людях, родившихся под знаком, название которого введено пользователем (если таких нет, выводится соответствующее сообщение).

**Вариант 17**

Описать структуру с именем ZNAK, содержащую следующие поля: фамилия, имя; знак Зодиака; дата рождения (массив из трех чисел). Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по ФИО, по знаку Зодиака (по календарю), по датам рождения;
- вывод на экран информации о людях, родившихся в месяц, значение которого введено пользователем (если таких нет, выводится соответствующее сообщение).

**Вариант 18**

Описать структуру с именем PRICE, содержащую следующие поля: название товара; название магазина, в котором продается товар; стоимость товара в гривнах. Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по названию магазина, по названию товара, по стоимости;
- вывод на экран информации о товаре, название которого введено с клавиатуры (если таких товаров нет, выводится соответствующее сообщение).

**Вариант 19**

Описать структуру с именем PRICE, содержащую следующие поля: название товара; название магазина, в котором продается товар; стоимость товара в гривнах. Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по названию магазина, по названию товара, по стоимости;
- вывод на экран информации о товарах, продающихся в магазине, название которого введено пользователем (если такого магазина нет, выводится соответствующее сообщение).

**Вариант 20**

Описать структуру с именем ORDER, содержащую следующие поля: расчетный счет плательщика; расчетный счет получателя; перечисляемая сумма в гривнах. Написать программу, позволяющую выполнять следующие действия (выбираются пользователем в соответствующем меню):

- ввод данных с клавиатуры и последующее сохранение введенных данных в файл (если в файле уже сеть записи, то новые дозаписываются в конец файла);
- упорядочивание записей по расчетным счетам плательщиков, по расчетным счетам покупателей, по суммам;
- вывод на экран информации о сумме, снятой с расчетного счета плательщика, ФИО которого введено пользователем (если такого расчетного счета нет, выводится соответствующее сообщение).