**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**
**факультет радиофизики и компьютерных технологий**
**кафедра информатики и компьютерных систем**

**Н.В. Серикова**

# ПРАКТИЧЕСКОЕ РУКОВОДСТВО

**к лабораторному практикуму**

# «СТРУКТУРЫ. ФАЙЛЫ»

**по курсу**
**«ПРОГРАММИРОВАНИЕ»**

**2020**
**МИНСК**

Практическое руководство к лабораторному практикуму «СТРУКТУРЫ. ФАЙЛЫ» по курсу «ПРОГРАММИРОВАНИЕ» предназначено для студентов, изучающих базовый курс программирования на языке C++, специальностей «Радиофизика», «Физическая электроника», «Компьютерная безопасность».

Руководство содержит некоторый справочный материал, примеры решения типовых задач с комментариями.

Все примеры протестированы в среде Microsoft Visual Studio 2005.

Автор будет признателен всем, кто поделится своими соображениями по совершенствованию данного пособия.

Возможные предложения и замечания можно присылать по адресу:
*E-mail: Serikova@bsu.by,*

# ОГЛАВЛЕНИЕ

# СТРУКТУРЫ

**Структура –** структурированный тип данных, состоящий из фиксированного числа компонентов одного или нескольких типов. Каждый компонент структуры должен иметь имя, чтобы можно было ссылаться на него. Определение структуры задает ее внутреннюю организацию.

Переменные, входящие в состав структуры, называются **полями** или **членами** структуры.

Компоненты структуры могут иметь любой тип, исключая тип void и тип этой же структуры (но может быть указателем на нее). Допускается вложенность структур.

Структуры в C++ обладают возможностью включать в себя не только переменные различных типов, но и функции. В языке C++ структура является видом класса и обладает некоторыми его свойствами.

Доступ к полю структуры осуществляется по имени переменной типа структуры и имени поля. Поля структуры могут иметь любой тип (кроме типа void и типа этой же структуры). Поля структуры располагаются в памяти последовательно друг за другом. Структура может содержать только такие поля, длина которых известна компилятору в момент определения структуры.

# ОПРЕДЕЛЕНИЕ ТИПА СТРУКТУРЫ. ОБЪЯВЛЕНИЕ ПЕРЕМЕННЫХ СТРУКТУРНЫХ ТИПОВ

Для определения типа структуры необходимо указать имена всех полей структуры и их типы. При определении структуры возможно указание имени структуры, но допускается и определение структур, не имеющих имен (анонимная структура).

**Синтаксис объявления структурного типа:**

**struct** <*Имя_структуры*>
**{**
*Тип_поля_1 Имя_поля1;*
*Тип_поля_2 Имя_поля2;*
**…**
**};**

Переменные типа структура объявляются также как переменные других типов. Возможны два способа.

1. При объявлении типа структуры можно сразу объявить одну или несколько переменных этого типа, не задавая имени для типа структуры.

2. Объявить (определить) переменную типа структура можно через имя типа структуры, определенного ранее.

**Примеры объявления переменных структурных типов:**

```
struct  emp
{
   int      empno;
   char     name [80];
   double   salary;
 };
emp engineer,  teacher,  professor;

struct         // Совмещение объявлений типа и переменных
{
   double re;
   double im;
} c1,  c2,  c3;
```

**Примеры объявления переменных структурных типов с инициализацией**

```
emp engineer  = {123, "Иванов",   650000},
    teacher   = {124, "Петров",   450000},
    professor = {127, "Сидоров",  790000};


struct          // Совмещение объявлений типа и переменных
{
     double re;
     double im;
} c1 = {0, 0},  c2 = {1, 0},  c3 = {1, 0};
```

# ДОСТУП К ЭЛЕМЕНТАМ СТРУКТУРЫ

Оператор «точка» используется для доступа к полям структуры. В контексте структур оператор «точка» называется оператором доступа к полям (членам) структуры. Для доступа к членам вложенной структуры оператор «точка» используется столько раз, какова вложенность структуры.

***Имя_переменной.Имя_поля***

**Примеры:**

```
strcpy(engineer.name,"Петров");
teacher.salary  =  professor.salary / 2;
c1.re = 12.5;
c1.im = -24;
c2.re = c1.re * c1.re + c1.im * c1.im;
```

# ВЛОЖЕННЫЕ СТРУКТУРЫ

Компоненты структуры могут иметь любой тип, исключая тип void и тип этой же структуры (но может быть указателем на нее). Допускается вложенность структур. Для доступа к членам вложенной структуры оператор «точка» используется столько раз, какова вложенность структуры.

**Пример.**

```
struct emp
{
  int empno;
  char name [80];
  double salary;
};

struct date
{
  unsigned short year;
  unsigned short month;
  unsigned short day;
};
struct  form_data
{
   date  birthday;
   emp employment;
};
```

**Объявление и инициализация переменных:**

```
form_data a, b =  {1933, 5, 19, 2345, "Petrova", 5678};

form_data a, b = { {1933, 5, 19}, {2345, "Petrova", 5678}};

a.birthday.year   = 1961;
a.birthday.month  =  4;
a.birthday.day    = 26;
a.employment       = b.employment;
```

# МАССИВ СТРУКТУР

```
const int m = 10,  n = 30

struct student
{
     unsigned int key;
     char     name [m];
};

student group [n];
```

| key | name | key | name | key | name | key | name | … | key | name |
|---|---|---|---|---|---|---|---|---|---|---|
| group[0] | | group[1] | | group[2] | | group[3] | | | group[9] | |

# ОБЪЕДИНЕНИЯ

**Объединение** – частный случай структуры, все поля которой располагаются по одному и тому же адресу. Это означает, что изменение одного поля, автоматически приводит к изменению всех остальных полей объединения. Основное достоинство объединений – возможность разных трактовок одного и того же содержимого (кода) памяти.

Данные, входящие в объединения, называются **полями** или **членами** объединения. Доступ к полям объединения осуществляется аналогично доступу к полям структур с помощью оператора доступа к членам структуры или класса или оператора разыменования указателя на структуру или класс.

Объединение применяют для экономии памяти, когда известно, что больше одного поля одновременно не требуется. **В каждый момент времени в переменной типа объединение хранится только одно значение**, и ответственность за его правильное использование лежит на программисте. В отличие от структур, все поля объединения используют одну и ту же область памяти.

Объединения часто используют в качестве поля структуры, для разной интерпретации одного и того же битового представления данных.

По сравнению со структурой на объединения налагаются некоторые **ограничения**:

- может инициализироваться только значением первого элемента;
- не может содержать битовые поля;
- не может содержать виртуальные методы, конструкторы, деструкторы;
- не может входить в иерархию классов.

Определение объединения задает ее внутреннюю организацию. **Формат описания** такой же, как у структуры, только вместо ключевого слова struct используется union. Для определения объединения необходимо указать имена всех полей объединения и их типы. При определении объединения возможно указать имя для типа объединения, но допускается определение объединения без имени (анонимное объединение).

**Длина объединения** (размер памяти, выделяемый объединению) равна наибольшей из длин его полей. Тип поля может быть любым, в том числе и структурой.

Пример:

```
union
{
     float          f;
     unsigned long  k;
};
```

Переменные типа объединение объявляются также как переменные других типов. Переменную типа объединение можно объявить через имя типа объединения, определенного ранее. Также можно объявить переменную при определении типа объединения , не задавая имени для типа объединения.

Особенностью инициализации переменной типа объединение в отличие от инициализации структуры является то, что переменная может быть проинициализирована только значением, которое имеет тип первого члена объединения.

# БИТОВЫЕ ПОЛЯ

Язык C++ предоставляет возможность задавать количество битов, в которых хранятся элементы целых типов. Это используется для плотной упаковки данных.

**Битовые поля** – особый вид полей структуры.

**Битовое поле** – это последовательность соседних двоичных разрядов (бит) внутри одного целого значения.

Битовые поля могут быть любого целого типа. Длина битового поля должна быть неотрицательным целым числом и не должна превышать длины базового типа данных битового поля.

Доступ к битовым полям осуществляется обычным способом – по имени. Адрес поля получить нельзя, в остальном битовые поля можно использовать как обычные поля структуры.

В отличие от обычного поля структуры при описании битового поля после имени переменной (через двоеточие) указывается длина поля в битах (целая положительная константа). Имя поля может отсутствовать, такие поля служат для выравнивания на аппаратную границу.

Операции с отдельными битами реализуются менее эффективно, чем с байтами и словами, и экономия памяти под переменные оборачивается увеличением объема кода программы. Размещение битовых полей в памяти зависит от компилятора и аппаратуры.

**Синтаксис описания битового поля:**

тип [имя]**:** ширина;

Члены битовых полей могут иметь значения типа signed или unsigned, char или int.

**Пример:**

```
struct
{
    unsigned  b1:1;
    unsigned  b2:2;
    unsigned  b4:4;
};
```

Массивы битовых полей недопустимы. К битовым полям не может быть применен оператор взятия адреса “&”, так как битовое поле может находиться внутри байта.

# СТРОКИ СТАНДАРТНОГО КЛАССА STRING

Язык С++ включает в себя новый класс, называемый ***string*.** Этот класс во многом улучшает традиционный строковый тип, позволяет обрабатывать строки также как данные других типов, а именно с помощью операторов. Он более эффективен и безопасен в использовании, не нужно заботиться о создании массива нужного размера для строковой переменной, класс *string* берет на себя ответственность за управлением памятью.

Если при создании приложения скорость выполнения не является доминирующим фактором, класс *string* предоставляет безопасный и удобный способ обработки строк.

Для работы со строками класса *string* существует множество **методов и операций.**

**Объявление и инициализация строк string.**

```
                    //создаем пустую строку
                    //вызов конструктора без аргументов
    string s1;

                  // создаем строку из С-строки
                  //вызов конструктора с одним аргументом
    string s2("aaaa");
//  string s2 = "aaaa";   // или так


              // создаем строку из С-строки 10 символов
              //вызов конструктора с двумя аргументами
    string s3("abcdefghijklmnopqrstuvwxyz",10);

              // создаем строку из 5 одинаковых символов
    string s4(5,'!');

                 // создаем строку-копию из строки s3
    string  s5(s3);

              // создаем строку-копию из строки s3
              //  начиная с индекса 5 не более 3 символов
    string  s6(s3,5,3);
```

# Методы для работы со строками класса string

| | МЕТОД | ЗАПИСЬ | ОПИСАНИЕ |
|---|---|---|---|
| 1 | **at** | **at** (unsigned n) | доступ к n-му элементу строки |
| 2 | **append** | **append** (string &str) | добавляет строку str к концу вызывающей строки (тоже, что оператор + ) |
| | | **append** (string &str, unsigned pos, unsigned n); | добавляет к вызывающей строке n символов строки str, начиная с позиции pos |
| | | **append** ( char *sr, unsigned n); | добавляет к вызывающей строке n символов С-строки s |
| 3 | **assign** | **assign** (string &str) | присваивает строку str вызывающей строке (тоже, что s2=s1) |
| | | **assign** ( string &str, unsigned pos, unsigned n); | присваивает вызывающей строке n символов строки str, начиная с позиции pos |
| | | **assign** ( char *sr, unsigned n); | присваивает вызывающей строке n символов С-строки s |
| 4 | **capacity** | unsigned int **capacity** (); | возвращает объем памяти, занимаемый строкой |
| 5 | **compare** | int **compare** (string &str); | сравнение двух строк, возвращает значение <0, если вызывающая строка лексикографически меньше str, =0, если строки равны и >0, если вызывающая строка больше |
| | | int **compare** (string &str, unsigned pos, unsigned n); | сравнение со строкой str n символов вызывающей строки, начиная с позиции pos; возвращает значение <0, если вызывающая строка лексикографически меньше str, =0, если строки равны и >0, если вызывающая строка больше |
| | | int **compare** (unsigned pos1, unsigned n1, string &str, unsigned pos2, unsigned n2); | n1 символов вызывающей строки, начиная с позиции pos1, сравниваются с подстрокой строки str длиной n2 символов, начиная с позиции pos2; возвращает значение <0, если вызывающая строка лексикографически меньше str, =0, если строки равны и >0, если вызывающая строка больше |

| | | | |
|---|---|---|---|
| **6** | **copy** | unsigned int **copy** (char *s, unsigned n, unsigned pos = 0); | копирует в символьный массив s n элементов вызывающей строки, начиная с позиции pos; нуль-символ в результирующий массив не заносится; метод возвращает количество скопированных элементов |
| **7** | **c_str** | char * **c_str**() | возвращает указатель на С-строку, содержащую копию вызываемой строки; полученную С-строку нельзя изменить |
| **8** | **empty** | bool **empty();** | возвращает истину, если строка пустая |
| **9** | **erase** | **erase (**unsigned pos **= 0,** unsigned n **= npos);** | удаляет n элементов, начиная с позиции pos (если n не задано, то удаляется весь остаток строки) npos-самое большое число >0 типа unsigned |
| **10** | **find** | unsigned int **find** (string &str, unsigned pos = 0); | ищет самое левое вхождение строки str в вызывающей строке, начиная с позиции pos; возвращает позицию вхождения, или npos(самое большое число >0 типа unsigned , если вхождение не найдено |
| | | unsigned **find** (char c, unsigned pos = 0); | ищет самое левое вхождение символа с в вызывающей строке, начиная с позиции pos; возвращает позицию вхождения, или npos(самое большое число >0 типа unsigned, если вхождение не найдено |
| | | unsigned **rfind** (char c, unsigned pos = 0); | ищет самое правое вхождение символа с в вызывающей строке, начиная с позиции pos; возвращает позицию вхождения, или npos(самое большое число >0 типа unsigned, если вхождение не найдено |
| **11** | **find_first_of** | unsigned **find_first_of** (string &str, unsigned pos = 0); | ищет самое левое вхождение любого символа строки str в вызывающей строке, начиная с позиции pos; возвращает позицию вхождения, или npos(самое большое число >0 типа unsigned, если вхождение не найдено |
| **12** | **find_last_of** | unsigned **find_last_of** (string &str, unsigned pos = 0); | ищет самое правое вхождение любого символа строки str в вызывающей строке, начиная с позиции pos; возвращает позицию вхождения, или npos(самое |

| | | | |
|---|---|---|---|
| | | | большое число >0 типа unsigned, если вхождение не найдено |
| **13** | **find_first_not_of** | unsigned **find_first_not_of** (string &str, unsigned pos = 0); | ищет самый левый символ не равный ни одному символу из строки str в вызывающей строке, начиная с позиции pos; возвращает позицию вхождения, или npos(самое большое число >0 типа unsigned, если вхождение не найдено |
| **14** | **insert** | **insert** (unsigned pos, string &str); | вставляет строку str в вызывающую строку, начиная с позиции pos |
| | | **insert** (unsigned pos1, string &str, unsigned pos2**,** unsigned n**);** | вставляет в вызывающую строку, начиная с позиции pos1 n символов строки str, начиная с позиции pos2 |
| | | **insert** (unsigned pos, char * sr, unsigned n); | вставляет в вызывающую строку n символов С-строки s**,** начиная с позиции pos |
| **15** | **length** | unsigned **length** (); | возвращает размер строки |
| **16** | **max_size** | unsigned **max_size**(); | возвращает максимальную длину строки |
| **17** | **replace** | **replace** (unsigned pos, unsigned n, string &str); | заменяет n элементов, начиная с позиции pos вызывающей строки, элементами строки str |
| | | **replace** (unsigned pos1, unsigned n1, string &str, unsigned pos2, unsigned n2); | заменяет n1 элементов, начиная с позиции pos1 вызывающей строки, n2 элементами строки str, начиная с позиции pos2 |
| | | **replace** (unsigned pos, unsigned n1, char *s, unsigned n2); | заменяет n1 элементов, начиная с позиции pos вызывающей строки, n2 элементами С-строки s |
| **18** | **size** | unsigned **size**(); | возвращает размер строки |
| **19** | **substr** | string **substr(**unsigned pos = **0,** unsigned n **= npos);** | выделяет подстроку длиной n из исходной строки, начиная с позиции pos |
| **20** | **swap** | **swap** (string &str) | обменивает содержимое вызывающей строки и строки str |

# Операции для работы со строками класса string

| | оператор | **ЗАПИСЬ**<br>**string s1,s2,s3;** | **ОПИСАНИЕ** |
|---|---|---|---|
| 1 | **+** | s3 = s1+ s2; | конкатенация (сцепление) строк, можно присоединить единичный символ к строке |
| 2 | **+=** | s1 += s2; | конкатенация (сцепление) строк с присвоением результата |
| 3 | **=** | s2 = s1; | присваивание |
| 4 | **==** | s2 == s1 | лексикографическое сравнение на равенство строк |
| 5 | **!=** | s2 != s1 | лексикографическое сравнение на неравенство строк |
| 6 | **>** | s2 > s1 | лексикографическое сравнение строк на > |
| 7 | **>=** | s2 >= s1 | лексикографическое сравнение строк на >= |
| 8 | **<** | s2 < s1 | лексикографическое сравнение строк на < |
| 9 | **<=** | s2 <= s1 | лексикографическое сравнение строк на <= |
| 10 | **[]** | s1[i] | индексация (обращение к элементу строки) |
| 11 | **>>** | cin >> s1; | ввод строки (лучше метод getline) |
| 12 | **<<** | cout << s2; | вывод строки |

# ФАЙЛЫ

Хранение данных в переменных и массивах является временным (до конца выполнения программы). Для постоянного хранения предназначены **файлы**. Посредством файлов обрабатываемые программой данные могут быть получены извне, а результаты сохранены для последующего использования.

**Файлом** называют именованную структуру данных, представляющую собой последовательность элементов одного типа, хранящихся на внешнем носителе информации.

**Физический файл** – файл с точки зрения операционной системы.

**Логический файл** – файл с точки зрения языка программирования.

У любого файла есть **имя**, что дает возможность работать одновременно с несколькими файлами.
В файле могут содержаться данные любых типов, за исключением файлов.

Под **доступом к файлу** понимают запись и чтение данных из файла.

По способу доступа файлы делятся на файлы **последовательного доступа** и файлы **произвольного (прямого) доступа.**

Файл может быть открыт

- **для чтения (входной файл),**
- **для записи (выходной файл),**
- **для чтения и записи (двунаправленный обмен).**

Операция **вывода данных в файл** означает пересылку их из рабочей области памяти в файл, а операция **ввода** – заполнение ячеек памяти данными, полученными из файла.

Различают **текстовые** и **двоичные (бинарные)** файлы.

При извлечении информации из файла программа должна уметь определять, когда данные в файле закончились. В любом языке программирование существует функция определения конца файла. Функция определяет конец файла либо через размер файла, читая его из служебной структуры файловой системы, либо через маркер конца файла EOF (End Of File, ASCII-код равен 26, комбинация клавиш <Ctrl><Z>), записываемый в конце файлов (конкретный способ зависит от типа используемой файловой системы).

**Текущая позиция** – место в файле, с которого будет выполняться следующая операция доступа к файлу.

# ПОТОКИ

**Поток** (**stream**) – абстрактное понятие, относящееся к любому переносу данных от источника к приемнику.

Чтение данных из потока называется **извлечением**, а вывод в поток – **помещением**.

Обмен с потоком производится через специальную область – **буфер**.

По направлению потоки делятся на **входные** (данные вводятся в память), **выходные** (данные выводятся из памяти) и **двунаправленные**.

По виду устройств, с которым работает поток, потоки делятся на **стандартные, файловые и строковые.** Характер поведения всех потоков одинаков, несмотря на различные физические устройства, с которыми они связываются.

Существует два вида потоков: **текстовый и двоичный**.

Текстовый используется для ввода-вывода символов, при этом могут происходить некоторые преобразования символов.

Двоичный поток можно использовать с данными любого типа, причем преобразования символов не выполняется: между тем, что посылается в поток , и тем, что реально содержится в файле существует взаимно-однозначное соответствие.

В C++ поток представляет собой объект некоторого класса.

Для поддержки потоков библиотека C++ имеет два базовых класса :

- **ios –** для ввода-вывода;
- **streambuf –** для обеспечения буферизации потоков и их взаимодействия с физическими устройствами.

Классы **istream** и **ostream** являются наследниками **ios** и предназначены **для ввода и вывода** соответственно. Класс **iostream** – наследник одновременно классов **istream** и **ostream** (пример множественного наследования).

# КЛАСС ISTREAM

Класс **istream** выполняет действия по вводу данных – извлечение, содержит следующие функции:

>> - форматированное извлечение данных всех основных (и перегружаемых) типов из потока;

для неформатированного чтения из потока:

**get()** – возвращает код извлеченного из потока символа или EOF;

**get(ch)** – извлекает один символ в ch и возвращает ссылку на поток;

**get(str)** – извлекает символы в символьный массив str до ограничителя ‘\n’;

**get(str, MAX)** – извлекает до MAX числа символов в символьный массив str;

**get(str, DELIM)** – извлекает символы в символьный массив str до указанного ограничителя (обычно ‘\n’); оставляет ограничитель в потоке;

**get(str, MAX, DELIM)** – извлекает в символьный массив str до MAX символов или до символа DELIM; оставляет ограничитель в потоке;

**getline(str, MAX, DELIM)** – извлекает в символьный массив str до MAX символов или до символа DELIM; извлекает ограничитель из потока;

**ignore(MAX, DELIM)** – извлекает и удаляет до MAX числа символов до ограничителя включительно (обычно ‘\n’); с извлеченными данными ничего не делает;

**peek ()** – возвращает следующий символ, оставляя его в потоке, или EOF, если достигнут конец файла;

**peek (ch)** – читает следующий символ, оставляя его в потоке;

**putback (ch)** – вставляет во входной поток символ, который становится текущим при извлечении из потока;

**gcount()** – возвращает число символов, считанных с помощью последнего вызова функ ций неформатированного ввода get(), getline(), read();

(наиболее часто применяемые для неформатированного чтения из файлов:)

**read (str, MAX)** – извлекает в символьный массив str MAX символов (или все символы до конца файла, если их меньше MAX);

**seekg(pos)** – устанавливает расстояние (в байтах) от начала файла до файлового указателя (т.е. устанавливает текущую позицию чтения в значение pos);

**seekg (pos, seek_dir)** – перемещает текущую позицию чтения на pos байтов, считая от одной из трех позиций, определяемых параметром seek_dir: ios::beg (от начала файла), ios::cur (от текущей позиции), ios::end (от конца файла);

**tellg()** – возвращает позицию (в байтах) указателя файла от начала файла.

**tellg(pos)** – возвращает позицию (в байтах) указателя файла от начала файла.

# КЛАСС OSTREAM

Класс **ostream** предназначен для вывода (вставки в поток) данных, содержит функции:

>> - форматированная вставка в поток данных всех основных (и перегружаемых) типов из потока;

для неформатированного вывода в поток:

**put(ch)** – выводит в поток один символ ch и возвращает ссылку на поток;

**flush()** – записывает содержимое потока вывода на физическое устройство (очистка буфера);

(наиболее часто применяемые для неформатированного вывода в файл:)

**write(str, SIZE)** – записывает SIZE символов из массива str в файл;

**seekp(pos)** – устанавливает текущую позицию записи в значение pos относительно начала файла;

**seekp(pos, seek_dir)** – перемещает текущую позицию файлового указателя на pos байтов, считая от одной из трех позиций, определяемых параметром seek_dir: ios::beg (от начала файла), ios::cur (от текущей позиции), ios::end (от конца файла);

**tellp()** – возвращает позицию указателя файла (в байтах).

# ОРГАНИЗАЦИЯ РАБОТЫ С ФАЙЛАМИ В С++

В С++ файл рассматривается как **последовательный поток байтов**.

От классов istream, ostream и iostream наследуются три класса **для работы с файлами**:

- **ifstream** – входной поток;
- **ofstream** – выходной поток;
- **fstream** – поток ввода-вывода.

Каждому файлу операционной системы в языке программирования ставится в соответствие **файловая переменная** определенного типа.

**Чтобы иметь возможность работать с файлом, необходимо связать переменную-поток с этим файлом** (процедура заключается в связывании объявленного потока с именем существующего или вновь создаваемого файла, а также в указании способа обмена информацией: чтение из файла, запись в него, чтение и запись). Обычно эта связь задается либо при открытии файла, либо при создании потока.

**При закрытии** файла связь разрывается (т.е. разрывается связь только переменной-потока с внешним файлом, сама переменная продолжает «жить» в соответствии с объявлением и может быть снова связана с тем же или с другим файлом). При закрытии файла буфер освобождается – выводится в файл на внешнее устройство. Если программа заканчивается аварийно, файлы остаются незакрытыми, и последняя порция информации на диск не попадает.

После создания потока, одним из способов связать его с файлом является компонентная функция **open()**.

Прототипы функции для каждого класса выглядят так:

```
void ifstream :: open (const char*  fileName, open_mode mode = ios::in);
void ofstream:: open (const char* fileName, open_mode mode = ios::out | ios::trunc);
void fstream  :: open (const char*  fileName, open_mode mode = ios::in | ios::out);
```

здесь fileName – имя файла, в которое может входить в спецификатор пути;
mode – режим – целая статическая константа размером в 1 бит; режимы можно объединять битовой операцией ‘ | ‘.

**Режимы обращения к файлам:**

| | |
|---|---|
| **ios::in** | открыть файл для ввода |
| **ios::out** | открыть файл для вывода |
| **ios::app** | записать данные в конец файла |
| **ios::ate** | переместиться в конец исходного открытого файла, данные могут быть записаны в любое место файла |
| **ios::trunc** | удалить содержимое файла, если он существует ( по умолчанию это делается и для ios::out) |
| **ios::nocreate** | если файл не существует, то операция его открытия не выполняется |
| **ios::noreplace** | если файл существует, то операция его открытия не выполняется |
| **ios::binary** | открыть файл в двоичном режиме (по умолчанию файлы открываются в текстовом режиме) |

**Режимы работы с файлами:**

- открыть файл для чтения;
- открыть файл для записи:
  - создание нового файла,
  - модификация существующего файла,
  - добавления информации в конец файла.

**Чтение из файла**

Операция **чтения из файла** означает заполнение ячеек памяти данными, полученными из файла.

Для чтения информации из файла необходимо:

- объявить файловую переменную,
- связать файловую переменную с физическим файлом или устройством,
- открыть файл для чтения,
- переместиться к той позиции в файле, с которой должно начаться считывание (определяется способом доступа к файлу: произвольный или последовательный),
- определить данные какого размера и в какую переменную будут считаны,
- вызвать функцию чтения данных из файла.

**Запись в файл**

Операция **записи данных в файл** означает пересылку их из рабочей области памяти в файл.

Для записи информации в файл необходимо:

- объявить файловую переменную,
- связать файловую переменную с физическим файлом или устройством,
- открыть файл для записи с указанием того будет ли файл перезаписан или изменен.
- переместиться к месту начала записи (определяется способом доступа к файлу: произвольный или последовательный),
- вызвать функцию записи данных в файл.

**Добавление данных в файл**

При добавлении информации данные будут записаны в конец файла.

Для добавления информации в файл следует:

- объявить файловую переменную,
- связать файловую переменную с физическим файлом или устройством,
- открыть файл для добавления (при этом указатель автоматически переместится в конец файла),
- вызвать функцию записи информации в файл.

# ТЕКСТОВЫЕ ФАЙЛЫ

Под доступом к файлу понимают запись и чтение данных из файла.

Текстовый файл является **файлом последовательного доступа**, т.е. любой элемент файла может быть прочитан или записан, только если перед ним был прочитан или записан элемент файла, непосредственно ему предшествующий.

**Текстовые файлы** предназначены для хранения текстовой информации.

**Текстовый файл рассматривается как последовательность символов, разбитая на строки.** Каждая строка завершается символом новой строки. В программе на C++ этот символ обозначается как ‘\n’.

Строки текстового файла могут иметь разную длину. Доступ к каждой строке возможен лишь последовательно, начиная с первой. Именно в файлах такого типа хранятся исходные тексты программ.

Целые и вещественные значения в текстовом файле имеют внешнее (текстовое) представление и отделяются друг от друга пробельным символом (чтобы узнать, где кончается одно число и начинается другое, при выводе в файл их надо их разделять пробельными символами). При чтении чисел из файла, они автоматически преобразуются из текстового представления во внутреннее, машинное. При выводе чисел в текстовый файл, они преобразуются из внутреннего представления в символьный (текстовый) вид.

В каждый момент времени позиции в файле, откуда выполняется чтение и куда производится запись, определяются значениями указателей позиций записи и чтения файла. Позиционирование указателей записи и чтения (т.е. установка на нужные байты) выполняется либо автоматически, либо за счет явного управления их положением.

Текстовые файлы могут создаваться с помощью тестового редактора и содержать данные разных типов.

Длина создаваемого файла никак не оговаривается при его объявлении и ограничивается только ёмкостью устройств внешней памяти.

# БИНАРНЫЕ ФАЙЛЫ

Под доступом к файлу понимают запись и чтение данных из файла.

**Произвольный (прямой) доступ к файлу** означает, что чтение и/или запись данных может производиться в любую указанную позицию.

В файлах произвольного доступа должен быть механизм вычисления местонахождения любого элемента файла (например, если все элементы имеют одинаковую фиксированную длину). Элементы могут вставляться в файл без разрушения других элементов файла. Элементы, которые в нем хранятся, могут быть изменены или удалены.

В C++ **двоичные (или бинарные) файлы** являются файлами прямого доступа.

Произвольный доступ выполняется только для файлов, открытых в двоичном режиме, тогда файлы **- бинарные или двоичные.**

При работе с файлами необходимо знать:

- элементы двоичных файлов хранятся во внутреннем представлении и поэтому должны создаваться программно;
- двоичные файлы также могут содержать элементы разных типов, для большей надежности их следует формировать из элементов только определенного типа;
- для последовательной обработки однотипных данных двоичный файл предпочтительнее текстового как в отношении надежности, так и в отношении экономичности (скорости выполнения операций);
- благодаря возможности прямого доступа к своим элементам двоичные файлы представляются привлекательными при чередовании операций чтения и записи данных.

**ПРИМЕР 1. Структура «комплексное число»**

Создать тип для представления комплексного числа. Реализовать ввод-вывод комплексного числа, определить сумму двух комплексных чисел.

```
#include <iostream>  // for cin cout
using namespace std;

struct complex   // объявление структуры комплексное число
{
     float re,  im;
};

complex read ();            // ввод комплексного числа
void print (complex c);     // вывод комплексного числа
complex add (complex, complex); // сумма комплексных чисел

void main()
{
   complex c1, c2, c3; // определение трех комплексных чисел

     c1 = read ();                 // ввод 1 числа
     c2 = read ();                 // ввод 2 числа
     c3 = add (c1, c2);           //сложение двух компл. чисел
     print(c3);                   // вывод результата на экран
}

complex read()                    // ввод комплексного числа
{    complex c;
     cout<<" re = ";        cin>> c.re;
     cout<<" im = ";        cin>> c.im;
     return c;
}
void print (complex c) // вывод на экран комплексного числа
{
     cout<<c.re<< " +i " << c.im <<endl;;
}
                        // сложение двух комплексных чисел
complex add (complex c1, complex c2)
{
     complex c3;
     c3.re  = c1.re  + c2.re;
     c3.im = c1.im + c2.im;
     return c3;
}
```

**ПРИМЕР 2. Массив структур**

Заполнить массив записей, содержащих поля: ФИО, ключ. Вывести значения записей массива на экран.

```
#include <iostream>  // for cin cout
using namespace std;

int main()
{
     const int m = 40, n = 3;
     int i;
     struct
     {
          unsigned int key;
          char name [m];
     } group [n];              // объявление массива записей

     for (i = 0; i < n; i++)
                    // заполнение полей массива записей
     {
          cout<<" FIO = ";
                 // ввод ФИО для i-го элемента массива
          cin.getline(group[i].name, m);
          cout<<" key = ";
            // ввод значения key для i-го элемента массива
          cin >> group[i].key;
          cin.get();
     }

     for (i = 0; i < n; i++)
           // вывод значений элементов массива на экран
     {
          cout<<" FIO = "<<group[i].name;
          cout<<" key = "<<group[i].key<<endl;
     }
     return 0;
}
```

**ПРИМЕР 3. Передача структуры в качестве аргумента. Структура как возвращаемое значение**

Заполнить массив записей, содержащих поля: ФИО, ключ. Вывести значения записей массива на экран.

Пример аналогичен предыдущему, но оформлен с использованием функций.

```
#include <iostream>  // for cin cout
using namespace std;

const int m = 40;
struct student           // объявление структуры student
{    unsigned int key;
     char name [m];
};

student read();            //  ввод значений полей структуры
void print(student);      //  вывод значений полей структуры

void main()
{    const int n = 3;
     int i;
     student group[n];      // объявление массива записей
                          // заполнение полей массива записей
     for (i = 0; i < n; i++)
          group[i] = read();

               // вывод значений элементов массива на экран
     for (i = 0; i < n; i++)
          print(group[i]);
}

student read()       //  ввод значений полей структуры
{    student stud;
     cout<<" FIO = ";
     cin.getline(stud.name,m);        // ввод ФИО
     cout<<" key = ";
     cin>>stud.key;                      // ввод значения key
     cin.get();
     return stud;
}
void print(student stud) //  вывод значений полей структуры
{    cout<<" FIO = "<<stud.name;
     cout<<" key = "<<stud.key<<endl;
}
```

**ПРИМЕР 4. Побайтный вывод значения вещественного числа (float)**

Написать функцию для вывода на экран побайтного представления в ЭВМ вещественного числа в формате float.

```
#include <iostream>  // for cin cout
using namespace std;

union  Uflc             // объявление объединения
{
    float f;            // веществ. число - 4 байта
    unsigned long l;    // длинное целое без знака - 4 байта
    unsigned char c[4];       // массив из 4 байт
};

void print(Uflc);       // вывод полей объединения

void main()
{
    Uflc FLC = {1.0};              // инициализация

    print(FLC);                  // вывод полей

    cout<<" input float number =  ";
    cin>>FLC.f;                  // ввод вещ. числа
    print(FLC);  // вывод по байтам представления вещ. числа

    FLC.l = 1;                    // ввод длинного целого
    print(FLC);                 // вывод полей
}

                          //  вывод полей объединения
void print(Uflc FLC)
{
    cout<<" float= "<<FLC.f<<endl;
    cout<<" long = "<<dec<<FLC.l<<endl;
    cout<<" char = ";

    for(int i = 0; i < 4; i++)
        cout<<hex<<(unsigned(FLC.c[i])&0xff)<<" ";
    cout<<endl;
}
```

**ПРИМЕР 5. Побайтный вывод значения вещественного числа (double) в двоичном представлении**

Написать функцию для вывода на экран побайтного представления в ЭВМ вещественного числа в формате double.

```
#include <iostream>  // for cin cout
using namespace std;

union  bits                          // объявление объединения
{
    double  d;                                  // веществ. число
    unsigned char c[sizeof (double)];    // массив байт
    bits (double n);          // инициализация числа
    void show_bits();         // побайтный вывод на экран
};

bits::bits(double n)
{
  d = n;
}

void bits::show_bits()       // побайтный вывод на экран
{
    for(int j = sizeof (double)- 1; j >= 0; j--)
    {
        cout<<" byte  "<< j << " = ";
        for(int i = 128; i ; i >>= 1)
            if (i & c[j])cout<<"1";
            else cout<<"0";
        cout<<endl;
    }
}

void main()
{
    double x;
    cout<<" input double number =  ";
    cin>>x;                  // ввод вещ. числа

    bits ob(x);              // инициализация
    ob.show_bits();          // вывод
}
```

**ПРИМЕР 6. Код символа через объединение**

Формирование кода символа с помощью битовых полей объединения:

```
#include <iostream>  // for cin cout
using namespace std;

void main( void )
{

     union
     {
          char simb;
          struct
          {
               int x : 5;
               int y : 3;
          };
     } cod;

     cod.x = 4;                          //  x = 001002
     cod.y = 2;                          //  y = 0102
     cout<<cod.simb;       // 'D' (код =  44 16 = 010001002)
}
```

**ПРИМЕР 7. Битовое представление целого числа**

Написать функцию для вывода на экран битового представления в ЭВМ числа в формате unsigned char.

```
#include <iostream>  // for cin cout
using namespace std;
    //вывод на экран двоичного представления байта-параметра
void binary (unsigned char ch)
{
    union
    {
        unsigned char ss;
        struct
        {           unsigned char a0:1;
            unsigned char a1:1;
            unsigned char a2:1;
            unsigned char a3:1;
            unsigned char a4:1;
            unsigned char a5:1;
            unsigned char a6:1;
            unsigned char a7:1;
        };
    } cod;

    cod.ss = ch;//занесение в объединение значения параметра
    cout <<"  bity =  ";

                                          // вывод бит
    cout <<" "<<cod.a7<<" "<<cod.a6
        <<" "<<cod.a5<<" "<<cod.a4
        <<" "<<cod.a3<<" "<<cod.a2
        <<" "<<cod.a1<<" "<<cod.a0<<endl;
}


void main()
{
    unsigned c;
    cin>>c;                  // вводим число 0..255
    if (c < 256)
        binary(c);           // битовое представление числа
}
```

**ПРИМЕР 8. Объявление и инициализация строки string**

```
#include <iostream>
#include <string>                    // строковый класс
using namespace std;

void main()
{
     string s1;              //создаем пустую строку

     string s2("aaaa");    // создаем строку из С-строки

              // создаем строку из С-строки 10 символов
     string s3("abcdefghijklmnopqrstuvwxyz",10);

              // создаем строку из 5 одинаковых символов
     string s4(5,'!');

                  // создаем строку-копию из строки s3
     string  s5(s3);

              // создаем строку-копию из строки s3
              //  начиная с индекса 5 не более 3 символов
     string  s6(s3,5,3);

     cout<<"s1= "<<s1<<endl;
     cout<<"s2= "<<s2<<endl;
     cout<<"s3= "<<s3<<endl;
     cout<<"s4= "<<s4<<endl;
     cout<<"s5= "<<s5<<endl;
     cout<<"s6= "<<s6<<endl;
}
```

**ПРИМЕР 9. Инициализация строки string. Оператор =. Метод assign**

```
#include <iostream>
#include <string>                    // строковый класс
using namespace std;

void main()
{
     string s1, s2, s3;

// в классе string определены три оператора присваивания:
// string & operator = (const string& str);
// string & operator = (const char* s);
// string & operator = (char c);

     s1 = '1';
     s2 = "bbbbbb";
     s3 = s2;

     cout<<"s1= "<<s1<<endl;
     cout<<"s2= "<<s2<<endl;
     cout<<"s3= "<<s3<<endl;

                                  //   метод assign
     s2.assign("ccccccc");        //  s2="cccccccc"
     s3.assign(s2);               //  s3=s2

     cout<<"s1 = "<<s1<<endl;
     cout<<"s2 = "<<s2<<endl;
     cout<<"s3 = "<<s3<<endl;

     s2.assign("1234");           //  s2="1234"
             // в s3 из s2 3 символа, начиная с 1 позиции
     s3.assign(s2,1,3);

     cout<<"s2= "<<s2<<endl;
     cout<<"s3= "<<s3<<endl;

     char s[]="56789";
                 //  присваивает  s3  3 символа С-строки
     s3.assign(s,3);
     cout<<"s= "<<s<<endl;
     cout<<"s3= "<<s3<<endl;
}
```

**ПРИМЕР 10. Ввод строки string. Оператор >>**

```
#include <iostream>
#include <string>
using namespace std;

void main()
{
     string s1;

     cout << "   Enter a string: ";
       //ввод выполняется до первого пробельного символа
     cin >> s1;
     cout << "You entered: " << s1 << endl;
}
```

**ПРИМЕР 11. Ввод строки string. Метод  getline**

```
#include <iostream>
#include <string>
using namespace std;

void main()
{
     string s1;

     cout << "Enter a string: ";
     getline(cin,s1);              //ввод   строки
     cout << "You entered: " << s1 << endl;
     cin.get();       // удаление из потока символа '\n'.

     cout << "   Enter a string: ";
               // свой разделитель для ввода строки
     getline(cin,s1,'&');
     cout << "You entered: " << s1 << endl;
     cin.get();       // удаление из потока символа '&'.
}
```

**ПРИМЕР 12. Длина строки string. Методы length, size**

```
#include <iostream>
#include <string>
using namespace std;

void main()
{
     string st("********************");

     cout << " " << st.length()<< " " << st.size() <<
             "  "<<st.max_size()<<endl;

     st = "Good Morning";
     cout << " " << st.length()<< " " << st.size() <<
             "  "<<st.max_size()<<endl;

     st = "Hello";
     cout << " " << st.length()<< " " << st.size() <<
             "  "<<st.max_size()<<endl;
}
```

**ПРИМЕР 13. Доступ к элементу строки string. Оператор []. Метод at**

```
#include <iostream>
#include <string>
using namespace std;

void main()
{
     string st("********************");
     int i;

     for (i = 0; i < st.length(); i++)
          cout << st[i];
                    // если i выходит за пределы строки,
                    // то поведение не определено
     st = "Good Morning";
     for (i = 0; i < st.length(); i++)
          cout << st.at(i);

          // если i выходит за пределы строки,
          // метод возвращает исключение типа out_of_rang
     st = "Hello";
     for (i = 0; i < st.length(); i++)
          cout << st[i];
}
```

**ПРИМЕР 14. Сравнение строк. Операторы сравнения**

Вводим строки в цикле, выход – ввод “пустой” сроки.
Вывод на экран результатов сравнения двух строк.

```
#include <iostream>
#include <string>
using namespace std;

void main()
{
     string st1, st2;
     cout << "Enter a string \n";
     getline(cin,st1);           //  ввод 1 строки

     while (true)
     {
          cout << "Enter a string \n";
          getline(cin,st2);      //  ввод 2 строки
          cin.get();

          if ( st2 == "")
              break;              //  выход из цикла

          cout  << endl<< st2 ;

          // операторы лексикографического сравнения строк
          if (st2 == st1)
             cout << "  =  ";
          else
              if (st2 < st1)
                 cout <<"  <  ";
              else
                 cout  << " > ";
          cout  << st1<<endl ;
          st1 = st2;
     }
}
```

**ПРИМЕР 15. Сравнение строк. Метод compare**

Вводим строки в цикле, выход – ввод “пустой” сроки.
Вывод на экран результатов сравнения двух строк.

```
#include <iostream>
#include <string>
using namespace std;

void main()
{
     string st1, st2;
     cout << "Enter a string \n";
     getline(cin,st1);           //  ввод 1 строки

     while (true)
     {
          cout << "Enter a string \n";
          cin.get();
          getline(cin,st2);           //  ввод 2 строки

          if ( !st2.compare("")) break; // выход из цикла
          cout  << endl<< st2 ;

               // лексикографическое сравнение строк
          if (!st2.compare(st1)) cout <<"  =  ";
          else
               if (st2.compare(st1)<0) cout <<"  <  ";
               else       cout  << " > ";
          cout  << st1<<endl ;
          cout  << endl <<st2[1]<<st2[2];

                    // лексикографическое сравнение подстрок
          if (!st2.compare(1,2,st1,2,3))
            cout <<"  =  ";
          else
               if (st2.compare(1,2,st1,2,3)<0)
                 cout <<"  <  ";
                else
                   cout  << " > ";
          cout  << st1[2]<<st1[3]<<st1[4]<<endl ;
          st1 = st2;
     }
}
```

**ПРИМЕР 16. Объединение строк. Оператор +. Метод append**

```
#include <iostream>
#include <string>
using namespace std;

void  main()
{
     string s1("11");;
     string s2("2222");
     string s3 ("333333");
     string s4("44444444");
     s1 += s2;                          // добавить s2 к s1
     cout<<"s1 = s1 + s2 = "<<s1<<endl;

     s4 = s1 + s2;                     // добавить s1 к s2
     cout<<"s4 = s1 + s2 = "<<s4<<endl;

     s4 = s4 + '!';                   // добавить s1 к s2
     cout<<"s4 = s4 + ! = "<<s4<<endl;

     s2.append(s1);                 // добавить s1 к s2
     cout<<"s2 + s1 = "<< s2<<endl;

      //  добавить к  s3  2 символа строки s1 со 1 позиции
     s3.append(s4,1,2);
     cout<<"s3 + s4 = "<< s3<<endl;

     char s[] = "56789";
                 //  добавить к  s3  2 символа С-строки s
     s3.append(s,2);
     cout<<"s3 + s = "<< s3<<endl;
}
```

**ПРИМЕР 17. Вставка строки (подстроки) в строку. Метод insert**

```
#include <iostream>
#include <string>
using namespace std;

void  main()
{
     string s1("1111111");;
     string s2("23456789");

     s1.insert(3,s2);    // вставить s2 в s1 с 3 позиции
     cout<<"s1="<< s1<<endl;

     s1 = "1111111";
     s2 = "23456789";
   // вставить 4 символа s2 со 2 позиции в s1 с 3 позиции
     s1.insert(3,s2,2,4);
     cout<<"s1="<< s1<<endl;

     s1 = "1111111";
     char s[] = "23456789";
           // вставить 4 символа s  в s1 с 3 позиции
     s1.insert(3,s,4);
     cout<<"s1 = "<< s1<<endl;
}
```

**ПРИМЕР 18. Замена строки (подстроки) в строке. Метод replace**

```
#include <iostream>
#include <string>
using namespace std;

void  main()
{
     string s1("1111111");;
     string s2("23456789");

  // замена  2 символов в s1 с 3 позиции элементами s2
     s1.replace(3,2,s2);
     cout<<"s1="<< s1<<endl;

     s1 = "1111111";
     s2 = "23456789";
     // замена  2 символов в s1 с 3 позиции 1 символом
     //  из 4 позиции строки  s2
     s1.replace(3,2,s2,4,1);
     cout<<"s1="<< s1<<endl;

     s1 = "1111111";
     char s[] = "23456789";
     // замена  2 символов в s1 с 3 позиции 4 символами s
     s1.replace(3,2,s,4);
     cout<<"s1="<< s1<<endl;
}
```

**ПРИМЕР 19. Удаление подстроки в строке. Метод erase**

```
#include <iostream>
#include <string>
using namespace std;

void  main()
{
     string s1("123456789");

          // удаление   2 символов в s1 с 3 позиции
     s1.erase(3,2);
     cout<<"s1="<< s1<<endl;

          // удаление   всех символов в s1 с 3 позиции
     s1 = "123456789";
     s1.erase(3);
     cout<<"s1="<< s1<<endl;

     s1 = "123456789";
     s1.erase();          // удаление   всех символов  s1
     cout<<"s1="<< s1<<endl;
}
```

**ПРИМЕР 20. Выделение подстроки в строке. Метод substr**

```
#include <iostream>
#include <string>
using namespace std;

void  main()
{
     string s1("123456789");
     string s2;

          // s2 – подстрока  s1 из 2 символов  с 3 позиции
     s2 = s1.substr(3,2);
     cout<<"s2="<< s2<<endl;

     s1 = "123456789";
           // s2 – подстрока s1 всех символов  с 3 позиции
     s2 = s1.substr(3);
     cout<<"s2="<< s2<<endl;

     s1 = "123456789";
     s2 = s1.substr();                        // s2=s1
     cout<<"s2="<< s2<<endl;
}
```

**ПРИМЕР 21. Обмен содержимого строк. Метод swap, reverse**

```
#include <iostream>
#include <string>
using namespace std;

void  main()
{
     string s1("123456789");
     string s2("abcdef");

     s1.swap(s2);
     cout<<"s1="<< s1<<endl;
     cout<<"s2="<< s2<<endl;

     string s3("12345678");

     reverse(s3.begin(),s3.end());
     cout<< s3 <<endl;

}
```

**ПРИМЕР 22. Поиск подстроки в строке. Метод find**

```
#include <iostream>
#include <string>
using namespace std;

void  main()
{
     string s1("123123123");
     string s2("12");
     unsigned k;

       //  поиск подстроки s2 в строке s1 с 4 позиции
     k = s1.find(s2,4);
     cout<<"   "<< k<<endl;

       //  поиск подстроки s2 в строке s1 с 7 позиции
     k = s1.find(s2,7);
     cout<<"   "<< k<<endl;

         //  поиск символа '1'  в строке s1 с 4 позиции
     k = s1.find('1',4);
     cout<<"   "<< k<<endl;

        //  поиск символа '1'  в строке s1 с 4 позиции
      k = s1.rfind('1',4);
     cout<<"   "<< k<<endl;
}
```

**ПРИМЕР 23. Поиск символа подстроки в строке. Метод find_first_of**

```
#include <iostream>
#include <string>
using namespace std;

void  main()
{
     string s1("1234561728");
     string s2("12");
     char s3[100]="124";
     unsigned k;

// поиск первого любого символа из подстроки s2 в строке s1
// с 4 позиции
     k = s1.find_first_of(s2,4);
     cout<<"   "<< k<<endl;             // 6

       //  поиск первого любого символа из count первых
       // символов подстроки s3 в строке s1  с 4 позиции
     k = s1. find_first_of (s3,4,2);
     cout<<"   "<< k<<endl;                  // 6

    // поиск первого любого символа из подстроки s3 в
    // строке s1 с 4 позиции

     k = s1. find_first_of (s3,4);
     cout<<"   "<< k<<endl;          // 6

        //  поиск символа '1'  в строке s1 с 4 позиции
      k = s1. find_first_of ('1',4);
     cout<<"   "<< k<<endl;                   // 6
}
```

**ПРИМЕР 24. Поиск символа подстроки в строке. Метод find_first_not_of**

```
#include <iostream>
#include <string>
using namespace std;

void  main()
{
     string s1("1234561728");
     string s2("12");
     char s3[100]="124";
     unsigned k;

// поиск первого символа отличного от любого символа
// из подстроки s2 в строке s1 с 6 позиции
     k = s1.find_first_not_of(s2,6);
     cout<<"   "<< k<<endl;             // 7

    // поиск первого символа отличного от любого символа
    // из count первых символов из подстроки s3
    // в строке s1 с 6 позиции
     k = s1. find_first_not_of (s3,6,2);
     cout<<"   "<< k<<endl;                 // 7

    // поиск первого любого символа отличного от любого
символа // из подстроки s3 в строке s1 с 6 позиции

     k = s1. find_first_not_of (s3,6);
     cout<<"   "<< k<<endl;          // 7

        //  поиск первого символа отличного от символа '1'
        //  в строке s1 с 6 позиции
      k = s1. find_first_not_of ('1',6);
     cout<<"   "<< k<<endl;                   // 7
}
```

**ПРИМЕР 25. Выделение лексем. Методы find_first_not_of, find_first_of**

```
#include <iostream>
#include <string>
using namespace std;
#define OUT

void lexem(string str, string delim, OUT string& res)
{
  unsigned int wordBegin = 0, wordEnd = 0;

                      // позиция начала лексемы
  wordBegin = str.find_first_not_of(delim, wordEnd);
                      // позиция конца лексемы
  wordEnd = str.find_first_of(delim, wordBegin);

  if (wordEnd >= str.length())
    wordEnd = str.length();

  while (wordBegin < str.length())
  {               // выделение лексемы
    string word = str.substr(wordBegin, wordEnd - wordBegin);
    if (res.length())
      res += " ";
    res += word;                          // результирующая строка

                      // позиция начала лексемы
    wordBegin = str.find_first_not_of(delim, wordEnd);
                      // позиция конца лексемы
    wordEnd = str.find_first_of(delim, wordBegin);

    if (wordEnd >= str.length())
      wordEnd = str.length();
  }
}

void  main()
{
  string delim(" .,;!?-:");        // строка разделителей
  string s1, s2;
  cout << "Enter a string \n";
  getline(cin,s1);                //  ввод  строки
  lexem(s1, delim, OUT s2);
  cout <<  s2 << endl;           //  вывод результатов
}
```

**ПРИМЕР 26. Выделение лексем. Чтение из потока**

```
#include <iostream>
#include <string>
using namespace std;
#define OUT

void lexem(string str, string delim, OUT string& res)
{
     // заменяем все символы в строке str из delim на пробелы
     for (int i = 0; i < str.length(); i++)
     {
          char c = str[i];
          if (delim.find(str[i]) < delim.length())
            str[i] = ' ';
     }
     stringstream stream(str);

     string word;
     while (stream >> word)  // читаем из потока следующую лексему
     {
          if (res.length())
               res += ' ';
          res += word;
     }
}

void  main()
{
     string delim(" .,;!?-:");        // строка разделителей
     string s1, s2;
     cout << "Enter a string \n";
     getline(cin,s1);                 //  ввод  строки
     lexem(s1, delim, OUT s2);
     cout <<  s2 << endl;            //  вывод результатов
}
```

**ПРИМЕР 27. Строки С и С++. Методы copy, c_str**

```
#include <iostream>
#include <string>
using namespace std;

void  main()
{
     string str("1234567890");
     char s[80];
     int k;

        // копируем 3 символа str с 5 позиции в s
     k = str.copy(s,3,5);
     cout<<s<<"   "<<k<<endl;

          // копируем 3 символа str с 5 позиции в s
     k = str.copy(s,3,5);
     s[3]='\0';
     cout<<s<<"   "<<k<<endl;

     cout<<str.c_str()<<endl;
}
```

**ПРИМЕР 28. Связывание логического и физического файлов**

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

void main()
{
    // в каждом из трех  потоковых классов связывание
    // логического и физического файлов можно выполнить
    // либо при создании потока,либо при открытии файла

                          // при создании потоков
    // создать объект ofstream
    // открыть физический файл file.bin с програмным
    // именем  outfile  в режиме записи
    ofstream outfile("file.bin");

    //  создать объект ifstream
     // открыть физический файл file.txt с программным
     // именем infile  в режиме считывания
    ifstream infile("file.txt");

    // создать объект fstream
    //  открыть физический файл name.txt с программным
    //  именем filename  и  для записи и для чтения
    fstream filename ("name.txt",ios::in | ios::out);


                             //  при открытии файлов
    ofstream out_file;    // создать объект ofstream
    //  открыть физический файл file.bin с програмным
    // именем  utfile  в режиме записи
    out_file.open("file.bin");

    ifstream in_file;       //  создать объект ifstream
      // открыть физический файл file.txt с программным
      // именем unfile  в режиме считывания
    in_file.open("file.txt");

    fstream file_name;       // создать объект fstream
    //  открыть физический файл name.txt с программным
    //   именем filename   и  для записи и для чтения
    file_name.open("name.txt",ios::in | ios::out);
}
```

**ПРИМЕР 29. Открытие файла для чтения**

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

int main()
{
    // Режимы открытия файлов
    // ios::in              открыть файла для ввода
    // ios::nocreate        если файл не существует, то
    //                операция его открытия не выполняется
    // ios::binary       открыть файл в двоичном режиме

                                                  // неявно
    //  открыть файл file.txt с программным именем infile
    //   в режиме считывания  ios::in - по умолчанию

    ifstream infile("file.txt");
                // проверка на успешность выполнения
    if (!infile)
             { cout<<"File error"<<endl;
               return 1;
             }

                                                  // явно
    ifstream infile1;         // создать объект ifstream

    //  открыть файл file1.txt с программным именем infile1
    infile1.open("file1.txt");

    //   функцией is_open  можно проверить открыт ли файл
    if (infile1.is_open())
        cout<<"file open"<<endl;
    return 0;
}
```

**ПРИМЕР 30. Открытие файла для записи**

```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{    // Режимы открытия файлов
     // ios::out    открыть файл для вывода
     // ios::app    записать данные в конец файла
     // ios::ate    переместиться в конец исходного
     //             открытого файла данные могут быть записаны
     //             в любое место файла
     // ios::trunc  удалить содержимое файла, если он
     // существует по умолчанию это делается и для ios::out)
     // ios::_Nocreate          если файл не существует,
     //  то операция его открытия  не выполняется
     // ios::noreplace   если файл существует, то операция
     //                       его открытия   не выполняется
     // ios::binary          открыть файл в двоичном режиме

                                 //  1  неявно
     //  открыть файл file.bin с программным именем outfile
     //  в режиме записи    ios::out - по умолчанию
     ofstream outfile("file.bin", ios::binary);
                  // проверка на успешность выполнения
     if (!outfile)
               {  cout<<"File error"<<endl;
                   return 1;
               }
                                 //  2 или так  неявно
     //  открыть файл file1.txt с программным именем outfile1
     //  при повторном обращении новая запись прибавляется
     //   к существующей
     ofstream outfile1("d:\\file1.txt", ios::app );
                  // проверка на успешность выполнения
     if (!outfile1)
              {  cout<<"File error"<<endl;
                 return 2;
              }
                                 // 3   явно
     ofstream outfile2;      // создать объект ofstream
     //  открыть файл file1.txt с программным именем
     //  outfile1 в режиме записи   ios::out - по умолчанию
     Outfile2.open("file1.txt");
     return 0;
}
```

**ПРИМЕР 31. Открытие файла для добавления информации**

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

int main()
{
                                        //  1  неявно
    //  открыть файл name.txt с программным именем filename
    //  для записи и для чтения
    fstream filename ("name.txt",ios::in | ios::out);
             // проверка на успешность выполнения
    if (!filename)
            { cout<<"File error"<<endl;
               return 1;
               }

                                       // 2   явно
    fstream filename1;                 // создать объект fstream
    //  открыть файл name.txt с программным именем filename1
    //  для записи и для чтения
    filename1.open("name.txt",ios::in | ios::out);
    return 0;
}
```

**ПРИМЕР 32. Чтение информации из файла прямого доступа**

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

// Для считывания и записи блоков двоичных данных в C++
// используются функции read() и write(), которые являются
// членами потоковых классов для ввода и вывода. Прототипы:
//   istream &read (char *str, streamsize  count);
//   ostream &write (const char *str, streamsize  count);

                    // вывод на экран компонент файла
void read_file(ifstream &infile);

int main()
{
 // нужно открыть файл file.bin с программным именем unfile
 //   в режиме считывания      ios::in - по умолчанию
 ifstream infile ("file.bin", ios::binary | ios::_Nocreate);

  if (!infile)
    {  cout<<"error2"<<endl;
       return 1;
     }

    read_file(infile);// вывод на экран значений из file.bin
    infile.close();

    return 0;
}

                  // вывод на экран компонент файла
void read_file(ifstream &infile)
{
   int v;
   infile.read(reinterpret_cast <char*> (&v), sizeof (int));
   while( !infile.eof() )   //  до достижения конца файла
    {
        cout<<v<<" "<<"\n";
            //  чтение элемента из файла  infile.read
    infile.read(reinterpret_cast <char*>(&v), sizeof (int));
    }
}
```

**ПРИМЕР 33. Чтение информации из текстового файла. Оператор >>**

```
#include <fstream>
#include <iostream>
using namespace std;
                            // вывод на экран строк файла
void read_file(ifstream &infile);

int main()
{
// нужно открыть файл file.txt с программным именем unfile
//   в режиме считывания  ios::in - по умолчанию
   ifstream infile ("file.txt");

   if (!infile)
     { cout<<"error2"<<endl;
       return 1;
     }

    read_file(infile); //вывод на экран значений из file.txt
    infile.close();

    return 0;
}

                                  // вывод на экран строк файла
void read_file(ifstream &infile)
{
    const unsigned MAX = 80;
    char v[MAX];
    infile>>v;                                          //  >>;
    while( !infile.eof() )//  до достижения конца файла
//  можно  while(infile) !!!
//  можно  while( !infile.fail() )
//       более корректно для текстового файла
    {
        cout<<v<<endl;
        infile>>v;       //  чтение строки из файла  infile
    }

}
```

**ПРИМЕР 34. Чтение информации из текстового файла. Метод getline**

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

                  // вывод на экран строк файла
void read_file(ifstream &infile);

int main()
{
     ifstream infile ("file.txt");
// нужно открыть файл file.txt с программным именем unfile
//    в режиме считывания  ios::in - по умолчанию

     if (!infile)
       {  cout<<"error2"<<endl;
          return 1;
       }

     read_file(infile); //вывод на экран значений из file.txt
     infile.close();
     return 0;
}


                        // вывод на экран строк файла
void read_file(ifstream &infile)
{
     const unsigned MAX = 80;
     char v[MAX];

     infile.getline(v, MAX);
              //  можно infile.get(v[i])  можно  infile>>v;
     while( !infile.eof() ) //  до достижения конца файла
     {
          cout<<v<<endl;
                     //  чтение строки из файла  infile
          infile.getline(v, MAX);
     }
}
```

**ПРИМЕР 35. Запись информации в файл прямого доступа**

```
#include <stdlib.h>
#include <time.h>                       // for time()
#include <fstream>
#include <iostream>
using namespace std;

// Для считывания и записи блоков двоичных данных в C++
// используются  функции read() и write(), которые являются
// членами потоковых классов для ввода и для вывода.
//   istream &read (char *str, streamsize  count);
//   ostream &write (const char *str, streamsize  count);

                  // создание файла случайных  целых чисел
void create(ofstream &outfile, int n);

int main()
{
     srand((unsigned)time(NULL));
     int n;
     cout<<"enter n"<<endl;
     cin>>n;                 // количество чисел файла

                             // создадим файл сл. чисел file.bin
    ofstream outfile("file.bin", ios::binary);
     if (!outfile)
       { cout<<"error1"<<endl;
         return 1;
       }

     create(outfile, n);
     outfile.close();
     return 0;
}
                // создание файла случайных целых чисел
void create(ofstream &outfile, int n)
{
 for (int i = 1; i <= n; i++)
 {
  int  v = rand() % 100;
  outfile.write(reinterpret_cast <char*>(&v), sizeof (int));
  }
}
```

**ПРИМЕР 36. Запись информации в текстовый файл**

```
#include <stdlib.h>
#include <time.h>                  // for time()
#include <fstream>
#include <iostream>
using namespace std;

                    // создание файла случайных  целых чисел
void create(ofstream &outfile, int n);

int main()
{
     srand((unsigned)time(NULL));
     int n;
     cout<<"enter n"<<endl;
     cin>>n;                 // количество чисел файла

                          // создадим текстовый файл сл. чисел
          ofstream outfile("file.txt");
     if (!outfile)
        {cout<<"error1"<<endl;
         return 1;
        }

     create(outfile, n);
     outfile.close();

     return 0;
}


          // создание текстового файла случайных целых чисел
void create(ofstream &outfile, int n)
{
     for (int i = 1; i <= n;  i++)
     {
          int v = rand()%100;
          outfile<<v<<endl;     // запись целых чисел в файл
                                // каждое число в новой строке
     }
}
```

**ПРИМЕР 37. Определение текущей позиции указателя**

```
// Только для двоичных файлов
// Для потока istream - функция pos_type tellg()
// Для потока ostream - функция pos_type tellp ()
// тип pos_type позволяет хранить самое большое допустимое
// значение, которое может возвратить любая из этих функций

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

                          // определение текущей позиции
int main()
{
     int n, k, v;

     ifstream filename ("file.bin", ios::binary | ios::in);
     if (!filename)
       {cout<<"error1"<<endl;
        return 1;
       }

     cout<<" k= "<<endl;
     cin>>k;

     for (int i = 0; i < k; i++)
      filename.read(reinterpret_cast<char*>(&v),sizeof(int));

     n = filename.tellg(); //определяем номер текущей позиции
     cout<<" n= "<<n<<endl;

     filename.close();

     return 0;
}
```

**ПРИМЕР 38. Смещение указателя в указанную позицию. Запись в позицию**

```
//  Только для двоичных файлов
//  Прототипы функций для произвольного доступа  к файлу
//  istream &seekg (off_type offset, seekdir origin);
//  ostream &seekp (off_type offset, seekdir origin);
//  тип off_set позволяет хранить самое большое допустимое
//  значение для  параметра offset
//  тип seekdir может иметь три значения
//  ios::beg   начало файла
//  ios::cur    текущая позиция
//  ios::end    конец файла
//  Для потока istream - функция seekg () перемещает текущий
//  указатель на offset  байт  относительно позиции,
//  заданной параметром origin .
//  Для потока ostream - функция seekp () перемещает текущий
//  указатель на offset  байт  относительно позиции,
//   заданной параметром origin .
                        // запись в n позицию файла числа v
#include <fstream>
#include <iostream>
using namespace std;

int main()
{   int n, v;
// нужно открыть файл file.bin с программным именем filename
//  для записи и для чтения
 fstream filename ("file.bin",ios::binary|ios::in|ios::out);
 if ( !filename)
   { cout<<"error1"<<endl;
     return 1;
    }

    cout<<" n = "<<endl;
    cin>>n;                                    // номер позиции
    cout<<" number = "<<endl;
    cin>>v;                                     // новое число
                      // устанавливаем указатель на позицию
                      // n*sizeof(тип компоненты)
    filename.seekp(n * sizeof(int), ios::beg);
                                          // запись числа
    filename.write(reinterpret_cast<char*>(&v),sizeof(int));

    filename.close();
    return 0;
}
```

**ПРИМЕР 39. Смещение указателя в указанную позицию. Чтение из позиции**

```
//  Только для двоичных файлов
//  Прототипы функций для произвольного доступа  к файлу
//  istream &seekg (off_type offset, seekdir origin);
//  ostream &seekp (off_type offset, seekdir origin);
//  тип off_set позволяет хранить самое большое допустимое
//  значение для параметра offset
//  тип seekdir может иметь три значения
//  ios::beg   начало файла
//  ios::cur    текущая позиция
//  ios::end    конец файла
//  Для потока istream - функция seekg () перемещает текущий
//  указатель на offset  байт  относительно позиции,
//  заданной параметром origin .
//  Для потока ostream - функция seekp () перемещает текущий
//  указатель  на offset  байт  относительно позиции,
//   заданной параметром origin .

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

int main()
{
     int n, v;

     ifstream filename ("file.bin", ios::binary | ios::in);
     if (!filename)
       { cout<<"error1"<<endl;
         return 1;
       }

     cout<<" n = "<<endl;
     cin>>n;

                  //  устанавливаем указатель на позицию
                  // n*sizeof(тип компоненты)
     filename.seekg(n*sizeof(int),ios::beg);
                                     // чтение числа
     filename.read(reinterpret_cast<char*>(&v),sizeof (int));

     cout<<" number= "<<v<<endl;
     filename.close();
     return 0;
}
```

**ПРИМЕР 40. Определение числа компонент в файле**

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

int main()
{

     ifstream infile ("file.txt", ios::binary);
     if ( !infile)
       { cout<<"error1"<<endl;
         return 1;
       }

     infile.seekg(0, ios::end);  //  указатель в конец файла
                                 //  определяем текущую позицию указателя
     int n = infile.tellg();
     n = n / sizeof(int);      //  количество компонент
                               //  sizeof(тип компонент файла)

     cout<<n<<endl;
     infile.close();

     return 0;
}
```

**ПРИМЕР 41. Удаление файла**

```
#include <stdio.h>                              // for remove()
#include <iostream>
using namespace std;


void main( void )
{
             //  Удаление файла
             //  функция  int remove( const char *path );

     if( remove( "file.txt" ) == -1 )
          cout<<"Could not delete 'file.txt'"<<endl;
     else
          cout<<"Deleted 'file.txt'"<<endl;
}
```

**ПРИМЕР 42. Уничтожение информации от текущей позиции указателя до конца**

```
#include <stdio.h>          //  for remove()  rename()
#include <fstream>
#include <iostream>
using namespace std;

int main()
{    int v, rezult;
     unsigned n;
    // открыть файл file.bin с именем infile для чтения
     ifstream infile ("file.bin", ios::binary);
     if (!infile)
       { cout<<"error1"<<endl;
         return 1;
        }
     cout<<" n = "<<endl; cin>>n;       // номер позиции
        //открыть временный файл file1.bin  для записи
     ofstream outfile ("file1.bin",ios::binary);
     if (!outfile)
       { cout<<"error1"<<endl;
         return 2;
       }
  //  чтение элемента из файла infile до необходимой позиции
    while(!infile.eof() && infile.tellg() <= n * sizeof(int))
    {
     infile.read(reinterpret_cast<char*>(&v),sizeof (int));
                     // запись элементов в outfile
     outfile.write(reinterpret_cast<char*>(&v),sizeof (int));
     }
     infile.close();
     outfile.close();

     rezult=remove( "file.bin");  // удаление исходного файла
     if (rezult != -1)
     {    // переименование временного файла в исходный файл
          rezult = rename("file1.bin", "file.bin");
          if (rezult)   {cout<<"error3"<<endl; return 3;}
     }
     else
        { cout<<"error4"<<endl;
          return 4;
    }
     return 0;
}
```

**ПРИМЕР 43. Работа с бинарным файлом**

```
//   Задача: создание бинарного файла случайных целых чисел,
//   вывод чисел на экран, создание нового файла только из
//   четных чисел исходного файла

#include <ctime>
#include <stdlib.h>
#include <fstream>
#include <iostream>
using namespace std;

void create(ofstream&,int);       // создание файла сл. чисел
void read_file(ifstream &);       // вывод на экран
          // создание файла четных чисел из файла исходного
void create_chet(ofstream &, ifstream &);

int main()
{    srand((unsigned)time(NULL));
     int n;
     cout<<"enter n"<<endl; cin>>n; // количество чисел файла
     // открыть file.bin  с именем outfile  для записи
     ofstream outfile("file.bin",ios::binary);
     if (!outfile)
       { cout<<"error1"<<endl;
         return 1;
       }
                // создание файла сл. чисел  file.bin
     create(outfile,n);
     outfile.flush();

          // открыть file.bin  с именем infile  для чтения
  ifstream infile ("file.bin", ios::binary);
  if (!infile)
    { cout<<"error2"<<endl;
      return 2;
     }

     read_file(infile);       // вывод на экран file.bin
     infile.close();
 // открыть file1.bin  с  именем outfile_new  для записи
     ofstream outfile_new ("file1.bin", ios::binary);
     if (!outfile_new)
       { cout<<"error3"<<endl;
         return 3;
        }
```

```
 // открыть file.bin с программным именем infile  для чтения
    infile.clear();
    infile.open("file.bin", ios::binary);
    if (!infile)
      { cout<<"error4"<<endl;
        return 4;
       }
                 // создание файла четных чисел file1.bin
                 // из файла исходного file.bin
    create_chet(outfile_new,infile);
    outfile_new.flush();        // outfile_new.close();
    infile.close();
//открыть file1.bin  с программным именем infile  для чтения
    infile.clear();
    infile.open("file1.bin", ios::binary);
    if (!infile)
      { cout<<"error5"<<endl;
        return 5;
      }
            // вывод на экран файла четных чисел file1.bin
    read_file(infile);
    infile.close();
    return 0;
}

                     // создание файла из n сл. чисел
void create(ofstream &outfile, int   n)
{
    for (int i = 1; i <= n;  i++)
    {
        int v = rand() % 100;
 outfile.write(reinterpret_cast <char*> (&v), sizeof (int));
    }
}
                     // вывод на экран чисел файла
void read_file(ifstream &infile)
{
    int v;
    infile.read(reinterpret_cast<char*>(&v),sizeof (int));
    while( !infile.eof())
    {
        cout<<v<<" ";
    infile.read(reinterpret_cast<char*>(&v),sizeof (int));
    }
    cout<<endl;
}
```

```
          // создание файла четных чисел из файла исходного
void create_chet(ofstream &outfile_new,  ifstream &infile)
{
     int ch;

     infile.read(reinterpret_cast<char*>(&ch), sizeof(int));

     while (infile)
     {
          if(ch % 2 == 0)
         outfile_new.write(reinterpret_cast<char*>(&ch),
                           sizeof (int));
          infile.read(reinterpret_cast<char*>(&ch),
                       sizeof(int));
     }
}
```

**ПРИМЕР 44. Работа с текстовым файлом**

```
//  Задача: создание текстового файла случайных целых чисел,
//  вывод чисел на экран, создание нового файла только из
//  четных чисел исходного файла

#include <ctime>
#include <stdlib.h>
#include <fstream>
#include <iostream>
using namespace std;

void create(ofstream&,int);    // создание файла сл. чисел
void read_file(ifstream &);  // вывод на экран
          // создание файла четных чисел из файла исходного
void create_chet(ofstream &, ifstream &);

int main()
{    srand((unsigned)time(NULL));
     int n;
     cout<<"enter n"<<endl; cin>>n;// количество чисел файла

// открыть file.txt с программным именем outfile  для записи
     ofstream outfile("file.txt");
     if (!outfile)
       { cout<<"error1"<<endl;
         return 1;
        }
                 // создание файла сл. чисел  file.txt
     create(outfile,n);
     outfile.flush();
// открыть file.txt  с программным именем infile  для чтения
     ifstream infile ("file.txt");
     if (!infile)
       { cout<<"error2"<<endl;
         return 2;
       }
     read_file(infile);
                            // вывод на экран file.txt
     infile.close();
// открыть file1.txt  с  именем outfile_new  для записи
     ofstream outfile_new ("file1.txt");
     if (!outfile_new)
       { cout<<"error3"<<endl;
          return 3;
        }
```

```
// открыть file.txt  с программным именем infile  для чтения
     infile.clear();
     infile.open("file.txt");
     if (!infile)
       { cout<<"error4"<<endl;
         return 4;
        }
     // создание файла четных чисел file1.txt  из файла
     // исходного file.txt
     create_chet(outfile_new,infile);
     outfile_new.flush();              // outfile_new.close();
     infile.close();
 // открыть file1.txt  с  именем infile  для чтения
     infile.clear();
     infile.open("file1.txt");
     if (!infile)
       { cout<<"error5"<<endl;
         return 5;
        }
            // вывод на экран файла четных чисел file1.txt
     read_file(infile);
     infile.close();
     return 0;
}
                          // создание файла n сл. чисел
void create(ofstream &outfile,  int   n)
{
     for (int i = 1; i <= n; i++)
     {
          int v = rand() % 100;
          outfile<<v<<endl;
     }
}

                  // вывод на экран чисел файла
void read_file(ifstream &infile)
{
     int v;
     infile>>v;
     while( !infile.fail())  //while( !infile.eof())
     {
          cout<<v<<" ";
          infile>>v;
     }
     cout<<endl;
}
```

```
        // создание файла четных чисел из файла исходного
void create_chet(ofstream &outfile_new, ifstream &infile)
{
     int v;
     infile>>v;
     while (!infile.eof())
     {
          if(v % 2 == 0)
               outfile_new<<v<<endl;
          infile>>v;
     }
}
```

# СЛОВАРЬ ПОНЯТИЙ, ИСПОЛЬЗУЕМЫХ В ЗАДАНИЯХ