Содержание:

Введение

Приложению, написанному на любом языке программирования, необходимо взаимодействовать с пользователем и окружающим миром. Иначе пользы от такого приложения может и не быть. Как правило, такое взаимодействие осуществляется посредством ввода-вывода информации на терминал или в файл.

Стоит, конечно, отметить, что есть множество программ, которые не используют файловый или консольный ввод-вывод: это программы, осуществляющие низкоуровневое взаимодействие с аппаратной частью компьютера и периферией (ядро ОС, драйверы и пр.).

В стандартном C++ существует два основных пути ввода-вывода информации:

- посредством традиционной системы ввода-вывода, унаследованной от языка C;
- с помощью потоков, реализованных в STL (Standard Template Library).

Однако при ближнем рассмотрении оказывается, что и потоки, и традиционная система ввода-вывода для осуществления необходимых действий используют вызовы операционной системы.

1. Ввод и вывод в языке C

Ввод/вывод в языке C осуществляется с помощью функций из стандартных библиотек. Чтобы ими пользоваться, в программу надо включить соответствующие h-файлы: stdio.h, stdlib.h, conio.h и др. Главная библиотека, в которой содержатся основные функции ввода/вывода, в том числе и обеспечивающие стандартный ввод/вывод, – stdio.h.

1.1 Ввод/вывод файлов

Чтобы работать с файлом, его сначала следует открыть: связать со специальной структурой с именем FILE, которая описана в библиотеке stdio.h и в которой задаются характеристики файла (размер буфера ввода/вывода, состояние файла, последняя прочитанная запись и т. п.). Связь эта выполняется с помощью функции fopen(), которая тоже входит в библиотеку stdio.h и возвращает указатель на структуру FILE. Поэтому в программе, прежде всего, следует задать указатель на структуру FILE, а затем записать оператор открытия файла:

FILE *fp;

fp = fopen(<имя файла>, <способ открытия файла>);

Функция fopen() имеет два параметра: имя открываемого файла и способ открытия файла. Способ открытия файла определяет, как пользователь будет работать с файлом: читать его, писать в него данные или делать что-то еще. Способы открытия файла и их коды могут быть следующие:

- r – файл открывается только для чтения данных из него;
- w – файл открывается только для записи в него (если файл не существует, он создается автоматически и открывается для записи);
- a – файл открывается для дозаписи данных в конец файла (если файл не существует, он создается для записи);
- r+ – существующий файл открывается для обновления: можно и читать из файла, и записывать в него данные;
- w+ – создается новый файл для обновления: можно и читать, и записывать в него данные;
- a+ – файл открывается для дозаписи данных в конец файла (если файл не существует, он создается автоматически и открывается для записи).

Если по какой-либо причине открытия файла не произошло (например, в режиме r+ задано имя несуществующего файла), то функция fopen() возвращает значение NULL. Поэтому при открытии файла следует осуществлять проверку:

if((fp = fopen(name, mode)) == NULL)

{ операторы обработки ошибки открытия }

{ остальные операторы программы }

После того как программа с данным файлом отработала, следует "отвязать" структуру FILE от файла или, как говорят, закрыть файл. Это осуществляет

функция fclose(fp). Она не только разрывает связь структуры с файлом, но и записывает в память оставшееся содержимое буфера ввода/вывода, через который и организуется ввод/вывод. Только после закрытия файла с ним можно выполнять какие-либо действия, т. к. он "свободен", "не привязан". Например, его можно удалить или заново открыть в другом режиме открытия и т.д.

После того как файл открыт, то для чтения или записи данных используют специальные функции.

1. Функция fputc()

Формат функции: fputc(c, fp);

Функция выводит символ в файл: c – выводимый символ, fp – указатель на файл.

1. Функция fputs()

Формат функции: fputs(s, fp);

Функция выводит строку в файл: s – выводимая строка, fp – указатель на файл.

1. Функция fgetc()

Формат функции: char c = fgetc(fp);

Читает один символ из файла с указателем fp. В случае ошибки или достижения конца файла возвращает EOF.

1. Функция fgets()

Формат функции: fgets(s, maxline, fp);

Читает строку в s, где s – массив символов или указатель типа char (предварительно должна быть выделена память для чтения с использованием указа- теля), maxline – максимальное число символов, которое требуется читать из файла с указателем fp. В случае ошибки или достижения конца файла возвращает NULL.

1. Функция fread()

Формат функции: fread(buf, m, n, fp);

Эта функция читает из файла с указателем fp первые n элементов данных, каждый из которых имеет длину m байт. Чтение происходит в буфер, на который указывает указатель buf, например, char buf[50] или char *buf (но в последнем случае предварительно надо выделить память для буфера). Общее количество байт чтения составит (n*m). Функция возвращает количество прочитанных элементов, а по достижении конца файла или возникновении ошибки чтения возвращает NULL.

1. Функция fwrite()

Формат функции: fwrite(ptr, m, n, fp);

Функция добавляет n элементов в файл с указателем fp, каждый элемент длиной в m байт. Данные записываются из буфера, на который указывает указатель ptr (этот указатель указывает на некоторый объект, например, на структуру). Общее количество записанных байтов равно (n * m). В случае ошибки записи функция возвращает ноль, в противном случае — количество записанных элементов.

1. Функция fseek()

Формат функции: fseek(fp, n, m);

Эта функция устанавливает указатель в файле fp в позицию, отстоящую на n байт от текущей, а направление перемещения (вперед или назад) определяется параметром m, который может быть задан одним из трех значений: 0, 1, 2 или одной из трех символических констант, определенных в файле stdio.h:

- SEEK_SET = 0 – к началу файла;
- SEEK_CUR = 1 – указатель на текущую позицию в файле;
- SEEK_END = 2 — к концу файла.

Функция fseek() используется для ввода/вывода потоком. Для работы не с потоковыми данными следует использовать функцию lseek(). После функции fseek() можно выполнять операции обновления в файлах, открытых для обновления. При удачном завершении работы fseek() возвращает ноль, в противном случае функция возвращает код ошибки. Тогда глобальная переменная errno принимает одно из следующих значений:

- EBADF — неверный указатель файла;
- EINVAL — неверный аргумент функции;
- ESPIPE — поиск на устройстве запрещен.

1. Функция ftell()

Формат функции: long int i = ftell(fp);

Возвращает текущее значение указателя файла fp (т. е. номер текущей позиции) в виде значения типа long int. Отсчет идет в байтах от начала файла. Возвращаемое значение может быть использовано затем в функции fseek(). Если обнаружены ошибки, функция возвращает значение -1L и присваивает глобальной переменной errno положительное значение.

1. Функция fscanf()

Формат функции: fscanf(fp, Control, arg1, arg2, ... , argn);

Функция читает данные из файла с указателем fp, преобразует их по форматам, записанным в управляющей строке Control, и отформатированные данные записывает в аргументы arg1, ... , argn.

1. Функция fprintf()

Формат функции: fprinf(fp, Control, arg1, arg2, ... , argn);

Выводит данные в файл с указателем fp, преобразует аргументы arg1, ... , argn к форматам, которые записаны в управляющей строке Control, и отформатированные данные записывает в файл.

1. Функция rewind()

Формат функции: rewind(fp);

Устанавливает указатель позиционирования в файле с указателем fp на начало потока. Вызов функции rewind(fp) эквивалентен вызову функции fseek(fp, 0L, SEEK_SET) за исключением того, что rewind() сбрасывает индикатор конца файла и индикаторы ошибок, а fseek() сбрасывает только индикатор конца файла. После функции rewind() можно выполнять операции обновления для файлов, открытых для обновления. Функция не возвращает никакого значения.

1. Функция remove()

Формат функции: remove(FILENameString);

Функция удаляет файл с именем FILENameString. Перед удалением файл должен быть закрыт. Строка FILENameString должна содержать полный путь к файлу. При нормальном завершении задачи функция возвращает ноль, а в случае ошибки – 1. При этом глобальная переменная errno принимает следующие значения:

- EACCES — запрещено удалять;
- ENOENT — нет такого файла или директории.

1. Функция FILElength()

Формат функции: long FILElength(fp);

Функция возвращает длину файла с указателем fp в байтах. Для работы функции требуется подключить файл io.h. В случае ошибки функция возвращает −1, и глобальная переменная errno принимает значение EBADF – неверный указатель файла.

1. Функция ferror()

Формат функции: ferror(fp);

Функция тестирует поток на ошибки чтения-записи. Если индикатор ошибки устанавливается, то он остается в таком положении, пока не будут вызваны функции clearerr() или rewind(), или до того момента, пока поток не закроется. Если в файле была обнаружена ошибка, то функция ferror() возвращает ненулевое значение.

1. Функция freopen()

Формат функции: FILE *freopen(const char *FILEname, const char *mode, FILE *stream);

Функция подставляет файл, заданный в первом параметре, вместо уже открытого потока. Она закрывает поток независимо от того, открыт ли он. Эта функция полезна для замены файла, связанного со стандартными устройствами ввода/вывода stdin, stdout или stderr. Способы открытия файла аналогичны функции fopen(). При успешном завершении функция возвращает указатель типа FILE (как и функция fopen()), при неуспешном — NULL.

1. Функция feof()

Формат функции: feof(fp);

Обнаруживает конец файла с указателем fp: тестирует поток на возникновение индикатора конца файла (который наступает после прочтения последней записи). Как только индикатор установлен, операции чтения файла возвращают индикатор до тех пор, пока не выполнена функция rewind() – "перемотка" в начало файла, или пока файл не будет закрыт. Индикатор конца файла переустанавливается с каждой операцией ввода. Функция возвращает ненулевую величину, если индикатор конца файла был обнаружен при последней операции чтения, в противном случае – ноль.

1. Функция ferror()

Формат функции: ferror(FILE *fp);

Функция выдает ненулевое значение, если операция с файловым потоком завершается с ошибкой (например, возникает ошибка чтения файла). Для обработки ошибок ввода/вывода следует записать эту функцию перед блоком работы с файлом в виде:

if (ferror(fp)) { команды обработки ошибок ввода/вывода }

Как только произойдет ошибка, выполнится эта функция, и начнут работать команды обработки ошибок.

1. Функция exit()

Формат функции: exit(int status);

Эта функция используется для срочного завершения работы программы при обработке ошибок открытия файла, а также для прерывания работы программы по каким-либо причинам. Перед завершением все файлы закрываются, остатки данных, находящиеся в буфере вывода, записываются в память и вызываются функции обработки ошибок, предварительно зарегистрированные специальной функцией atexit().

1.2 Стандартный ввод/вывод

При запуске любой программы автоматически открываются сразу три файла:

- файл стандартного ввода. Его указатель называется stdin;
- файл стандартного вывода. Его указатель называется stdout;

- файл стандартного вывода ошибок. Его указатель называется stderr.

При работе с файлами мы можем использовать эти указатели, чтобы направлять данные в стандартные потоки, в которых по умолчанию ввод идет с клавиатуры, а вывод — на экран. Например, чтобы ввести строку с клавиатуры, можно применить функцию fgets() в виде: fgets(s, maxline, stdin); а для вывода строки на экран — функцию fputs()в виде: fputs(s, stdout); Из приведенного ранее перечня функций, обслуживающих ввод/вывод, можно увидеть, что существуют функции getc(fp), putc(c,fp), которые соответственно вводят один символ из файла с указателем fp и пишут один символ в файл с указателем fp. Если вместо указателя fp, который имеет тип FILE, в эти функции поместить указатели стандартного потока, то они станут соответственно вводить один символ с клавиатуры и выводить его на экран.

В языке С стандартный ввод можно перенаправлять на ввод из файла: если некоторая программа с именем p1.exe использует функцию getchar(), то с помощью выполнения командной строки:

p1.exe < anyFILE1

получим ввод не с клавиатуры, а из файла anyFILE1. Командную строку в С можно выполнить с помощью системной функции system() в виде:

system("P1.EXE < ANYFILE1");

причем символы должны быть в верхнем регистре, т.к. выполняется команда DOS. Точно так же, как и для ввода, можно перенаправить стандартный вывод в файл. Если имеем программу p2.exe, которая использует стандартный вывод, то с помощью выполнения командной строки:

p2.exe > anyFILE2

получим вывод в файл anyFILE2.

Рассмотрим основные функции стандартного ввода/вывода языка С.

1. Функция getchar()

Формат функции: getchar();

Функция вводит с клавиатуры один символ и возвращает его. Обращаться к этой функции можно так:

```
char c; // или int c;

c = getchar();
```

1. Функция putchar()

Формат функции: putchar(c);

Выводит значение переменной c (один символ) на стандартное выводное устройство.

1. Функция printf()

Формат функции: printf(Control, arg1, arg2, ... , argn);

Это функция форматного вывода. Выводит на экран содержимое arg1, arg2, ... , argn и возвращает количество выводимых байт. Control – управляющая символьная строка, в которой находятся форматы вывода на экран для соответствующих аргументов arg1, arg2, ... , argn, т. е. первый формат – для вывода arg1, второй — для arg2 и т. д. Все символы, находящиеся между форматами, выводятся один к одному, т. е. не форматируются. Это дает возможность вывода дополнительного текста для улучшения читаемости результата вывода. Форматы вывода задаются так: любой формат начинается с символа '%' и заканчивается одним из символов форматирования:

- d – аргумент преобразуется к десятичному виду (с учетом знака);
- x – аргумент преобразуется в беззнаковую шестнадцатеричную форму;
- U – аргумент преобразуется в беззнаковую десятичную форму;
- c – аргумент рассматривается как отдельный символ;
- s – аргумент рассматривается как строка символов; символы строки печатаются до тех пор, пока не будет достигнут нулевой символ или не будет напечатано количество символов, указанное в спецификации точности;
- e – аргумент рассматривается как переменная типа float или double и преобразуется в десятичную форму в экспонентном виде;
- f – аргумент рассматривается как переменная типа float или double иρ преобразуется в десятичную форму;
- n – указатель на целое со знаком;
- p – входной аргумент выводится как указатель.

Между границами формата вывода (от знака % до символа-спецификатора типа выводимого аргумента) находятся:

[флажки][ширина][.точность][F|N|h|l|L]

1. Функция scanf()

Формат функции: scanf(Control, arg1, arg2, ... , argn);

Это функция форматного ввода с клавиатуры. Осуществляет посимвольный ввод данных с клавиатуры, преобразует их в соответствии с форматом для каждого значения, указанном в управляющей (форматной) символьной строке Control, и результат преобразования записывает в аргументы arg1, arg2, ..., argn. Смысл строки Control тот же, что и для функции printf(). Так как arg1, arg2, ..., argn – это выходные параметры функции, то при обращении к функции они должны задаваться своими адресами: имена массивов задаются именами, т. к. имя массива — это указатель на его первый элемент, а те аргументы, которые не являются указателями, задаются как &arg. Форматная строка Control – это символьная строка, содержащая три типа объектов: незначащие символы, значащие символы и спецификации формата. Незначащие символы – это пробел, знак табуляции (\t), символ перехода на новую строку (\n). Как только функция встречает незначащий символ в строке формата, она считывает, но не сохраняет все последующие незначащие символы до тех пор, пока не встретится первый значащий символ (т. е. пропускает незначащие символы). Значащие символы – это все символы кода ASCII, кроме символа '%'. Если функция встречает в форматной строке значащий символ, она его считывает, но не сохраняет. Спецификация формата функции имеет вид:

%[*][ширина][F/N] [h/l] символ_формата

1. Функция sprintf()

Формат функции такой: sprintf(string, Control, arg1, arg2, ... , argn);

Эта функция аналогична printf(), за исключением того, что результат своей работы она выводит не на стандартное устройство вывода, а в строку string. Это позволяет собирать в одну строку данные совершенно разных типов. Функция sscanf() Формат функции: sscanf(string, Control, arg1, arg2, ... , argn); Эта функция аналогична scanf(), за исключением того, что входные данные для ее работы поступают не со стандартного устройства ввода, а из строки string. Это позволяет выделять в строке различные группы данных совершенно разных типов и помещать их в отдельные переменные.

1. Функция cprintf()

Формат этой функции совпадает с форматом функции printf(): cprintf(Control, arg1, arg2, ... , argn);

Параметры аналогичны параметрам printf(). Но для обеспечения работы этой функции следует подключить к программе файл conio.h, выполнив #include Если функция printf() выводит данные туда, куда назначен stdout, то функция cprintf() всегда выводит данные на консоль (на это указывает символ 'c' в начале ее имени), т.е. на экран. В отличие от printf(), функция cprintf() не переводит символ '\n' в пару "\r\n" – возврат каретки и перевод строки (вместо '\n' надо указывать оба этих символа). Кроме того, символы табуляции '\t' не преобразуются в пробелы. Эту функцию не рекомендуется использовать для приложений Win32 или Win32 GUI. Но ее можно использовать для выдачи на экран цветных сообщений. Для этого надо воспользоваться функциями textcolor() (установить цвет текста) и textbackground() (установить цвет фона).

1. Функция gets()

Формат функции: gets(s);

Вводит строку символов с клавиатуры и записывает ее в строку s, которая может быть объявлена как char *s или char s[].

1. Функция puts()

Формат функции: puts(s);

Выводит содержимое строки s на устройство стандартного вывода (экран). Строка s может быть объявлена как char *s или char s[].

1. Функция cputs()

Формат функции: cputs(s);

Выводит содержимое строки s на экран (s может быть объявлена как char *s или char s[]). Эту функцию можно использовать для вывода на экран цветных текстов. Цвет выводимых символов задается с помощью функции textcolor(), а цвет фона – функцией textbackground(). Для работы функции надо подключить файл conio.h.

1. Функция gotoxy()

Формат функции: gotoxy(x, y);

Переводит курсор в точку с координатами (x, y) в текущем окне на экране, где x – номер столбца экрана, y – номер строки экрана. Обе переменные должны быть описаны как int (не в пикселах). Для работы функции надо подключить файл conio.h.

1. Функция clrscr()

Формат функции: clrscr();

Очищает экран и закрашивает его цветом, заданным функцией textbackground().

2. Ввод и вывод в языке C++

Ввод и вывод в C++ организован с помощью так называемых поточных классов, содержащих данные и методы работы с файлами по вводу/выводу. Поточные классы происходят от общего предка – класса ios и потому наследуют его функциональность. Чтобы начать писать программу с использованием ввода/вывода на языке C++, следует обязательно выполнить в программе:

#include <fstream>

Класс fstream является потомком классов istream и ostream. Эти же классы являются родителями классов ifstream и ofstream. Класс fstream используется для организации ввода/вывода (т. е. чтения-записи) в один и тот же файл. Классы ifstream, ofstream – для организации соответственно ввода (чтения) файла и вывода (записи в файл). В свою очередь, экземплярами классов istream, ostream являются cin, cout, cerr, с помощью которых осуществляется так называемый стандартный ввод/вывод – ввод со стандартного вводного устройства, которым по умолчанию является клавиатура, и вывод на стандартное выводное устройство, которым по умолчанию является экран. Таким образом, включения в программу класса fstream оказывается достаточным для организации как стандартного, так и файлового ввода/вывода. Файловый ввод/вывод организован с помощью переопределенных в поточных классах операций включения (<>).

Чтобы работать с файлом, его сначала следует открыть – связать со специальной структурой, в которой задаются характеристики файла (размер буфера ввода/вывода, состояние файла, последняя прочитанная запись и т. п.). Связь эта выполняется с помощью функции open(), входящей в один из классов, который определяет ввод/вывод (fstream, istream, ostream). Поэтому, чтобы выполнить

такую функцию, следует сначала создать экземпляр соответствующего класса, чтобы получить доступ к этой функции. Если мы, например, хотим выполнять вывод в файл (т. е. запись в него), то следует создать экземпляр класса ostream: ostream exp; и затем выполнить функцию

exp.open().

В скобках должны быть указаны параметры этой функции: имя открываемого файла и способ открытия файла, в котором задаются сведения о том, как пользователь собирается работать с файлом: читать его, писать в него или делать что-то еще. После того как файл открыт для чтения или записи, используют операции включения извлечения (<>). Если использовать пример с экземпляром exp класса ostream, то можно записать, например:

exp << "строка текста" << i << j << endl;

Здесь i, j — некоторые переменные (например, int i; float j;), endl – конец вывода и переход на новую строку. После того как работа с файлом закончена, следует закрыть файл, чтобы разорвать связь с той структурой, с которой файл был связан при его открытии. Это необходимо, чтобы дать возможность другим файлам "открываться". Этот акт выполняется с помощью метода close() того же экземпляра класса, который мы создавали, чтобы выполнить функцию open(). В нашем случае следовало бы написать:

exp.close();

2.1 Файловый ввод/вывод с использованием разных классов

Поточные классы – это поставщики инструментов для работы с файлами. В поточных классах хранятся:

- структуры, обеспечивающие открытие/закрытие файлов;
- функции (методы) открытия/закрытия файлов;
- другие функции и данные, обеспечивающие собственно ввод/вывод.

При использовании поточных классов языка C++ в основной программе требуется писать директиву:

using namespace::std;

В противном случае программа не пройдет компиляцию. В листинге 1 приводится пример использования директив пространства имен.

Листинг 1.

```
#include <vcl.h>

#include <iostream>

#include <conio.h>

namespace F

{

float x = 9;

}

namespace G

{

using namespace F;

float y = 2.0;

namespace INNER_G

{

float z = 10.01;

}

}

int main()

{

using namespace G;
```

```
using namespace G::INNER_G;

float x = 19.1;

std::cout << "x = " << x << std::endl;

std::cout << "y = " << y << std::endl;

std::cout << "z = " << z << std::endl;

getch();

return 0;

}
```

В результате на экране появится:

x = 19.1

y = 2

z = 10.01

std::cout – это стандартный вывод. Здесь показано, что объект cout принадлежит пространству имен std.

1. Работа с классом fstream

Члены этого класса позволяют открыть файл, записать в него данные, переместить указатель позиционирования в файле (указатель, показывающий, на каком месте в файле мы находимся) в то или иное место, прочитать данные.

Этот класс имеет следующие основные функции (методы):

open() – открывает файл;

close() – закрывает файл;

is_open() – если файл открыт, то возвращает true, иначе — false;

rdbuf() – выдает указатель на буфер ввода/вывода.

Формат функции open(): open(char* file_name, open_mode);

где file_name – имя открываемого файла, open_mode – способ открытия файла. Способ открытия файла задается значением перечислимой переменной: enum open_mode {app, binary, in, out, trunc, ate}; Эта переменная определена в базовом классе ios, поэтому обращение к перечислимым значениям в классе fstream, с экземпляром которого мы работаем, должно идти с указанием класса-родителя: ios::app, ios::binary и т. д.

2.2 Назначение способов открытия файла

app – открыть файл для дозаписи в его конец;

binary – открыть файл в бинарном виде (такие файлы были записаны по определенной структуре данных и поэтому должны читаться по этой же структуре);

in — открыть файл для чтения;

out – открыть файл для записи в его начало. Если файл не существует, он будет создан;

trunc – уничтожить содержимое файла, если файл существует (очистить файл);

ate – установить указатель позиционирования файла на его конец.

При задании режимов открытия файла можно применять оператор логического ИЛИ (|), чтобы составлять необходимое сочетание режимов открытия. Приведем пример программы работы с классом fstream (листинг 2). Результат работы показан на рисунке 1.

Листинг 2.

```
#include <vcl.h>

#include <iostream>

#include <conio.h>

#include <fstream>

#include <stdio.h>
```

```
void main()
{
using namespace std;
fstream inout;
inout.open("fstream.out", ios_base::in | ios_base::out | ios_base::trunc);
inout << "This is the story1 of a man" << endl;
inout << "This is the story2 of a man" << endl;
inout << "This is the story3 of a man" << endl;
char p[100];
inout.seekg(0);
inout.getline(p, 100);
cout << endl << "String1 :" << endl;
cout << p;
fstream::pos_type pos = inout.tellg();
inout.getline(p, 100);
cout << endl << "String2 :" << endl;
cout << p;
inout.getline(p, 100);
cout << endl << "String3 :" << endl;
cout << p;
inout.seekp(pos);
inout << "This is the story2 of a man" << endl;
inout << "This is the story3 of a man" << endl;
```

inout.seekg(0);

cout << endl << endl << inout.rdbuf();

inout.close();

system("DEL FSTREAM.OUT");

getch();

}

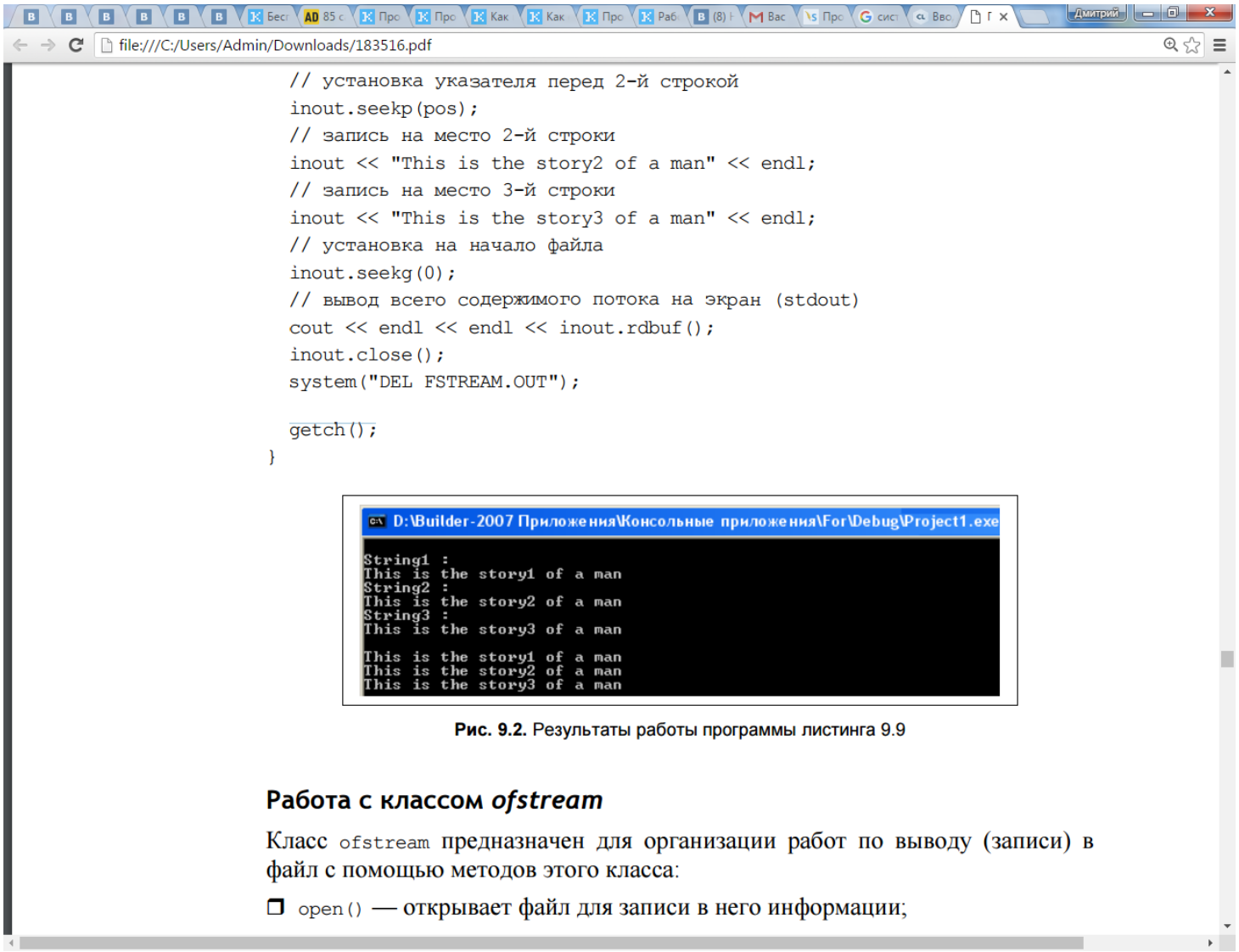

Рисунок 1.

1. Работа с классом ofstream

Класс ofstream предназначен для организации работ по выводу (записи) в файл с помощью методов этого класса:

open() – открывает файл для записи в него информации;

is_open() – возвращает true, если файл открыт, и false – в противном случае;

put() – записывает в файл один символ;

write() – записывает в файл заданное число символов;

skeep() – перемещает указатель позиционирования в заданное место файла;

tellp() – выдает текущее значение указателя позиционирования;

close() – закрывает файл;

rdbuf() – выдает указатель на буфер вывода (этот буфер находится в структуре, с которой связывается файл при его открытии).

В листинге 3 приведен пример использования класса ofstream.

Листинг 3.

```
ofstream FILE;

FILE.open("a.txt");

if(FILE == NULL) return(0);

for(int i = 0; i < 2; i++)

FILE << "string " << i << endl;

FILE.close();
```

1. Работа с классом ifstream

Класс ifstream предназначен для организации работ по вводу (чтению) из файла с помощью методов этого класса:

open() – открывает файл для чтения из него информации;

is_open() – возвращает true, если файл открыт, и false – в противном случае;

get() – читает из файла один символ;

read() – читает из файла заданное число символов;

eof() – возвращает ненулевое значение, когда указатель позиционирования в файле достигает конца файла;

peek() – выдает очередной символ потока, но не выбирает его (не сдвигает указатель позиционирования данного в файле);

seekg() – перемещает указатель позиционирования в заданное место файла;

tellg() – выдает текущее значение указателя позиционирования;

close() – закрывает файл;

rdbuf() – выдает указатель на буфер ввода (этот буфер находится в структуре, с которой связывается файл при его открытии).

Пример использования класса приведен в листинге 4.

Листинг 4.

```
ifstream FILE;

char p[100];

FILE.open("a.txt");

if(FILE == NULL) return(0);

while(!FILE.eof())

{

FILE >> p;

cout << p << endl;

}

FILE.close();
```

2.3 Стандартный ввод/вывод в C++

Стандартный ввод/вывод является частным случаем файлового ввода/вывода. При файловом вводе/выводе мы объявляли экземпляры соответствующих поточных классов и затем пользовались методами и операциями << и >>. Но классы istream, ostream, лежащие в основе поточных классов, содержат стандартные объекты-экземпляры классов с именами cout (экземпляр класса для стандартного ввода), cin (экземпляр класса для стандартного вывода) и cerr (экземпляр класса для стандартного вывода сообщений об ошибках).

При запуске любой программы на языке C+ + эти стандартные потоки определены (открыты) и по умолчанию назначены на стандартное вводное устройство — клавиатуру (cin), на стандартное выводное устройство — экран (cout и cerr). Причем все эти устройства синхронно связаны с соответствующими указателями stdin, stdout, stderr. Так что работа со стандартным вводом/выводом сводится к тому, что вместо задаваемых пользователем имен экземпляров соответствующих классов задаются имена стандартных экземпляров классов: cin, cout. Открывать ничего не нужно, надо только использовать операции <> и операции форматирования. Если мы пишем имена переменных, из которых выводятся или в которые вводятся данные, то по умолчанию для ввода/вывода используются определенные форматы. Например, запишем:

cout << i;

В этом случае значение i выведется на экран в формате, определенном по умолчанию для типа i и в минимальном поле.

Запишем:

cin >> i >> j >> s;

где i, j, s описаны соответственно как int, float, char. В записи мы не видим форматов, но при вводе значений этих переменных с клавиатуры (после ввода каждого значения надо нажимать клавишу) их форматы будут учтены.

1. Объект cout

Объект cout направляет данные в буфер-поток, связанный с объектом stdout, объявленным в файле stdio.h. По умолчанию стандартные потоки C и C++ синхронизированы. При выводе данные могут быть отформатированы с помощью

функций-членов класса или манипуляторов. Перечень их приведен в таблице 1.

Таблица 1.

Манипуляторы	**Функции-члены класса**	**Описание**
showpos	setf(ios::showpos)	Выдает знак плюс у выводимых положительных чисел
noshowpos	unsetf(ios::showpos)	—
showbase	setf(ios::showbase)	Выдает базу системы счисления в выводимом числе в виде префикса
noshowbase	unsetf(ios::showbase)	—
uppercase	setf(ios::uppercase)	Заменяет символы нижнего регистра на символы верхнего регистра в выходном потоке
nouppercase	unsetf(ios::uppercase)	—
showpoint	setf(ios::showpoint)	Создает символ десятичной точки в сгенерированном потоке с плавающей точкой (в выводимом числе)
noshowpoint	unsetf(ios::showpoint)	—
boolalpha	setf(ios::boolalpha)	Переводит булевый тип в символьный
noboolalpha	unsetf(ios::boolalpha)	—

unitbuf	setf(ios::unitbuf)	Сбрасывает буфер вывода после каждой операции вывода
nounitbuf	unsetf(ios::unitbuf)	—
internal	setf(ios::internal, ios::adjustfield)	Добавляет символы-заполнители к определенным внутренним позициям выходного потока (речь идет о выводе числа в виде потока символов). Если такие позиции не определены, поток не изменяется
left	setf(ios::left, ios::adjustfield)	Добавляет символы-заполнители с конца числа (сдвигая число влево)
right	setf(ios::right, ios::adjustfield)	Добавляет символы-заполнители с начала числа (сдвигая число вправо)
dec	setf(ios::dec, ios::basefield)	Переводит базу вводимых или выводимых целых чисел в десятичную (введенные после этого манипулятора данные будут выводиться как десятичные)
hex	setf(ios::hex, ios::basefield)	Переводит базу вводимых или выводимых целых чисел в шестнадцатеричную (введенные после этого манипулятора данные будут выводиться как шестнадцатеричные)
oct	setf(ios::oct, ios::basefield)	Переводит базу вводимых или выводимых целых чисел в восьмеричную (введенные после этого манипулятора данные будут выводиться как восьмеричные)

fixed	setf(ios::fixed, ios::floatfield)	Переводит выход с плавающей точкой в выход с фиксированной точкой
scientific	setf(ios::scientific, ios::floatfield)	Выдает числа с плавающей точкой в виде, используемом в научных целях: например, число 23450000 будет записано как: 23.45e6
fill(c)	setfill(char_type c)	Задает символ заполнения при выводе данных
precision(n)	setprecision(int n)	Задает точность вывода данных (количество цифр после точки)
setw(int n)	width(n)	Задает ширину поля для выводимых данных (количество символов)
endl		Вставляет символ новой строки ('\n') в выходную последовательность символов и сбрасывает буфер ввода
ends		Вставляет символ '\0' в выходную последовательность символов
flush	flush()	Сбрасывает буфер вывода
ws		Задает пропуск пробелов при вводе

Пример программы с применением объекта cout приведен на листинге 5. Результат работы представлен на рисунке 2.

Листинг 5.

```
#include <vcl.h>
#include <iostream>
#include <conio.h>
#include <iomanip>
#include <stdio.h>
void main()
{
using namespace std;
int i;
float f;
cout << "Enter i and f >" << endl;
cin >> i >> f;
cout << i << endl;
cout << f << endl;
cout << hex << i << endl;
cout << oct << i << dec << i << endl;
cout << showpos << i << endl;
cout << setbase(16) << i << endl;
cout << setfill('@') << setw(20) << left << dec << i;
cout << endl;
cout.fill('@');
cout.width(20);
cout.setf(ios::left, ios::adjustfield);
```

```
cout.setf(ios::dec, ios::basefield);

cout << i << endl;

cout << scientific << setprecision(10) << f << endl;

cout.precision(6);

cout << f << fixed << endl;

getch();

}
```

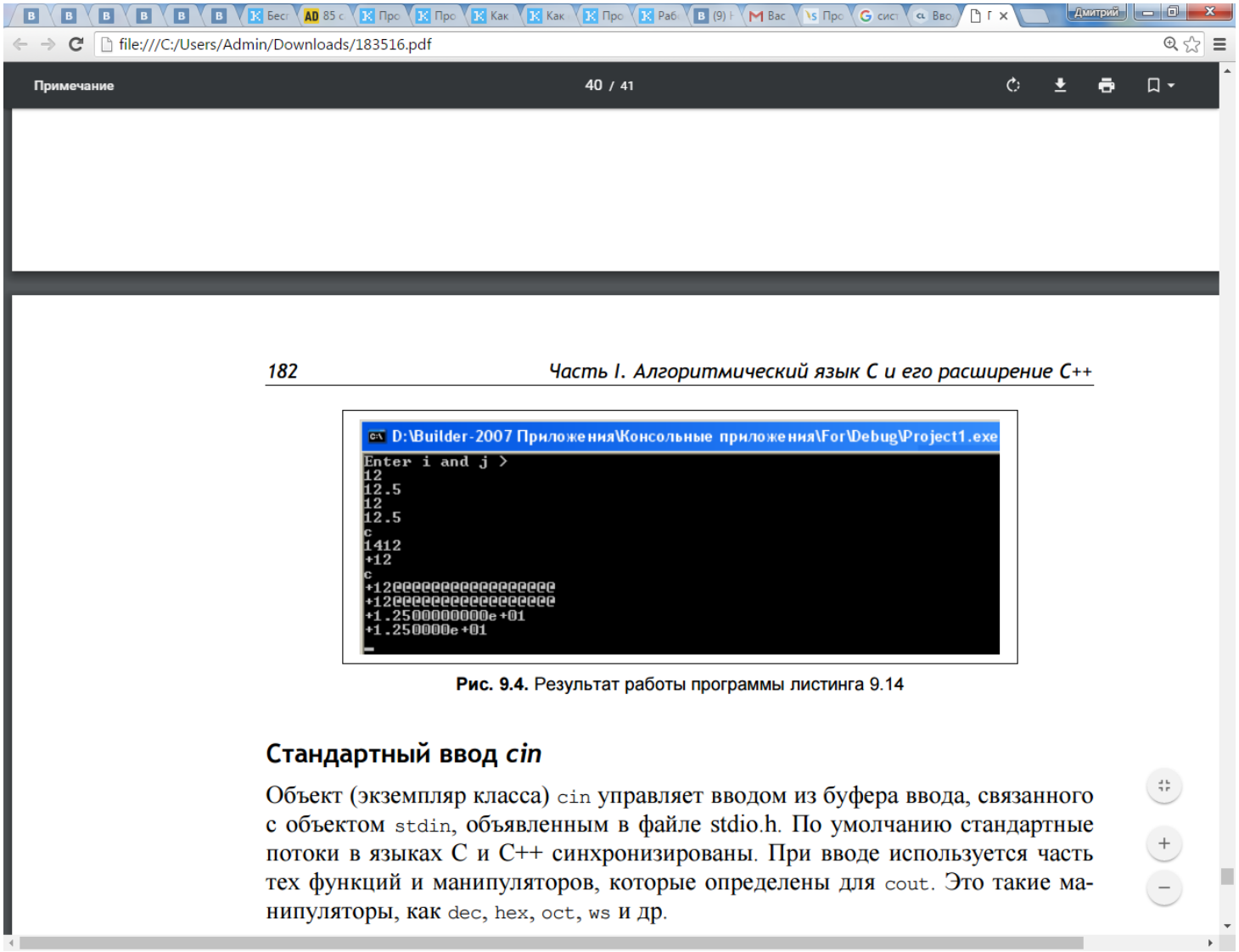

Рисунок 2.

1.

2.4 Стандартный ввод cin

Объект (экземпляр класса) cin управляет вводом из буфера ввода, связанного с объектом stdin, объявленным в файле stdio.h. По умолчанию стандартные потоки в языках C и C++ синхронизированы. При вводе используется часть тех функций и манипуляторов, которые определены для cout. Это такие манипуляторы, как dec, hex, oct, ws и др.

Пример программы с использованием объекта cin приведен в листинге 6. Результат работы представлен на рисунке 3.

Листинг 6.

```
#include <vcl.h>

#include <iostream>

#include <conio.h>

#include <iomanip>

#include <stdio.h>

void main()

{

using namespace std;

int i;

float f;

char c;

cout << "Enter i,f,c and then input the string >" << endl;

cin >> i >> f >> c;

cout << i << endl << f << endl << c << endl;
```

```
char p[50];

cin >> ws >> p;

cout << p << endl;

cin.seekg(0);

cin.getline(p,50);

cout << p << endl;

getch();

}
```

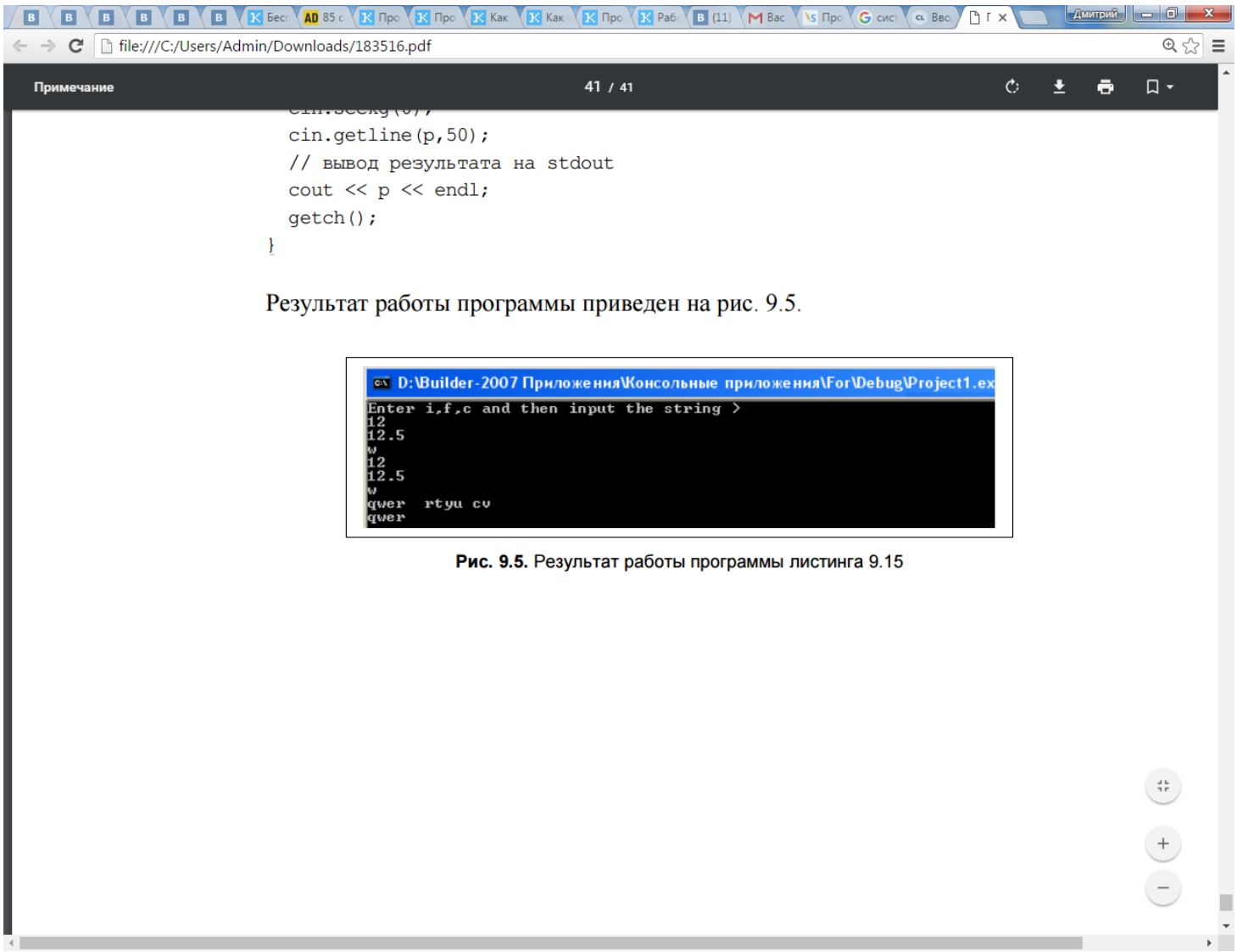

Рисунок 3.

Заключение

Специалисты C++ рекомендуют использовать для ввода-вывода только потоки STL и отказаться от использования традиционного ввода-вывода в духе C. Однако, ничего не мешает, по крайней мере пока, использовать традиционную систему ввода-вывода. Более того, предусмотрена специальная функция для синхронизации ввода-вывода, выполненного посредством потоков и посредством старых функций.

Какой механизм использовать – вопрос предпочтений программиста, если работодателем явно не предписано использование конкретного механизма. В любом случае для физического ввода-вывода используются вызовы операционной системы. Всё остальное – обёртка, набор более или менее удобных функций или классов для взаимодействия с ОС.

Библиографический список

1. Ахо Альфред В., Хопкрофт В., Ульман Джеффри Д. Структуры данных и алгоритмы — М.: Вильямс, 2016.
2. Кортмен Т.Х Алгоритмы: построение и анализ — М.: Вильямс, 2013
3. Миков А. И., Королев Л.Н., Информатика. Введение в компьютерные науки. — Абрис, Высшая школа, 2012.
4. Мейерс С. Эффективное использование C++. 55 верных советов улучшить структуру и код ваших программ: Пер. с англ. — 3-е изд. — М.: ДМК пресс, 2006.
5. Страуструп Б. Язык программирования C++: Пер. с англ. — 3-е спец. изд. — М.: Бином, 2003.
6. Джосьютис Н. М. C++. Стандартная библиотека. Для профессионалов: Пер. с англ. — СПб.: Питер, 2004.
7. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001.