Содержание:

Введение

Темой данной курсовой работы является «Способы ввода/вывода в C++». Данная тема касается различных способов ввода данных и их вывода в программу, без которых работа этой программы невозможна. Целью курсовой работы является выяснение функций и области применения различных способов ввода/вывода информации в C++.

Любой программе и приложению для работы необходимы донные которые поставляются устройствами и методами ввода/вывода. Эти методы позволяют программе и пользователю (или двум программам) передавать друг другу необходимые данные. Например внешними устройствами ввода/вывода считаются клавиатура, мышь, монитор и т.д.

Но также есть программы, не использующие файловый или консольный ввод-вывод: обычно это программы работающие на низком уровне с аппаратной частью компьютера и его периферийными устройствами (ядро,мат.плата,дравйверы и т.д.)

В стандартной версии C++ существует 2 основных пути ввода-вывода информации:

- С использованием обычного способа ввода-вывода доставшегося от языка C
- с помощью потоков, реализованных в STL (Standard Template Library).

Однако оба этих способа, ввода-вывода для осуществления необходимых действий используют вызовы ОС.

1. Ввод и вывод в языке C

Для того чтобы приступить к описанию ввода-вывода в C++, сперва обратим взор на его «предка» от которого ему и достались большинство функций и методов.

Ввод-вывод в обычном языке C производится с использованием команд и функций из различных библиотек. Для того чтобы их использовать, в программе необходимо подключить необходимые h-файлы: stdio.h, stdlib.h, conio.h и др. Основная библиотека - stdio.h, в ней содержатся стандартные и самые широко используемые

команды и функции.[1]

1.1 Ввод/вывод в работе с файлами

Для начала работы с различными видами файлов и данных в них, их надо связать со специализированной структурой с именем FILE, информация о которой находится в библиотеке stdio.h также в ней задаются некоторые свойства файла(размер буфера ввода/вывода, состояние файла, последняя прочитанная запись и т. п.).[2] Эта связь создается при использовании команды fopen(), которая также находится в библиотеке stdio.h, также она возвращает указатель на структуру FILE. Поэтому, прежде всего в программе, необходимо установить указатель на структуру FILE, а потом уже писать команду необходимую для открытия нужного файла:

FILE *fp;

fp = fopen(<точное имя файла>, <режим работы с файлом>); [3]

В команде fopen() есть две настройки: точное имя файла и режим работы с файлом. В строке режима работы указывается какие действия пользователь хочет делать с файлом: только прочесть его, записать, или и то и другое.[4] Существуют следующие обозначения параметров открытия файла:

- r – позволяет только считывать данные с файла;
- w – этот параметр может использоваться только для записи данных в файл, если указывается имя несуществующего файла то он создается и открывается автоматически.
- a – нужен для добавления данных в конце файла, также при использовании имени несуществующего файла он создается и открывается автоматически.
- r+ – открывает существующий файл и позволяет читать и записывать в нем данные;
- w+ – создается новый файл для обновления: можно и читать, и записывать в него данные;
- a+ – используется для добавления данных в конец файла, если файл с указанным именем отсутствует то он создается автоматически. [5]

Бывает что файл не получается открыть из-за некоторых причин (например, в свойстве r+ написано имя несуществующего файла), тогда функция fopen()

выставляет значение NULL. [6]Чтобы это предотвратить нужно выполнить проверку:

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

[операторы обработки ошибки открытия]

[остальные операторы программы]

[7]После окончания работы с нужным файлом, необходимо отключить от него функцию FILE. Это делается с помощью функции fclose(fp). Эта функция используется не только для разрыва с функцией FILE, но и записывает в память оставшееся содержимое буфера ввода/вывода, через который и организуется ввод/вывод. Только после отвязывания файла от FILE он становится доступен для дальнейших команд. Например, его можно удалить или опять открыть в другом режиме работы и т.д.

Приведем фрагмент программы, в которой используется функция fopen() для открытия файла под именем TEST.

FILE *fp;

fp = fopen("test", "w");

Можно заметить недостаточность такого кода в программе. Хотя приведенный код технически правильный, но его обычно пишут немного по-другому.

FILE *fp;

if ((fp = fopen("test", "w")==NUL)

{

printf("Ошибка при открытии файла.\n\r")"

exit(1);

}[8]

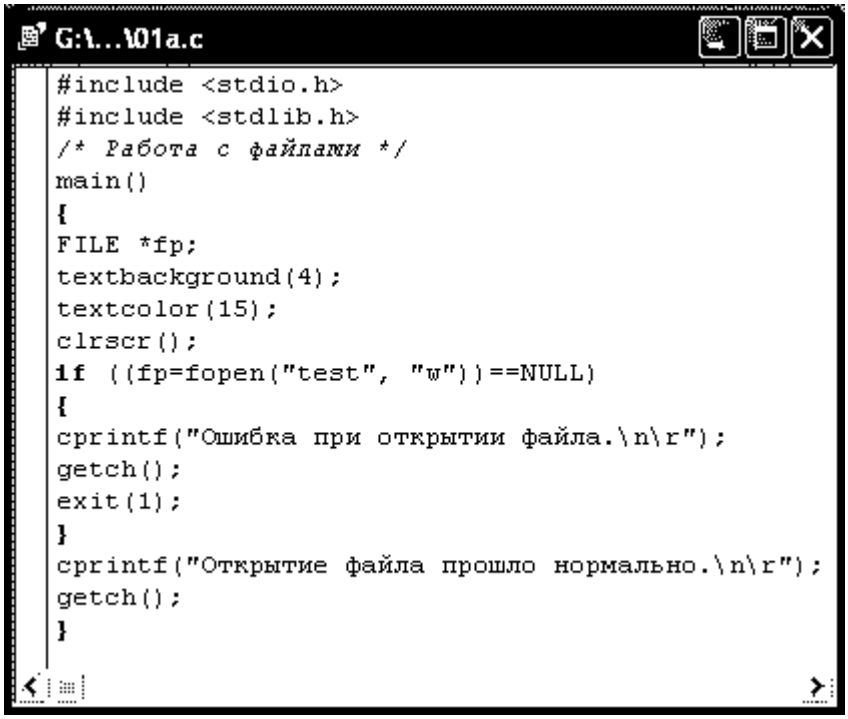
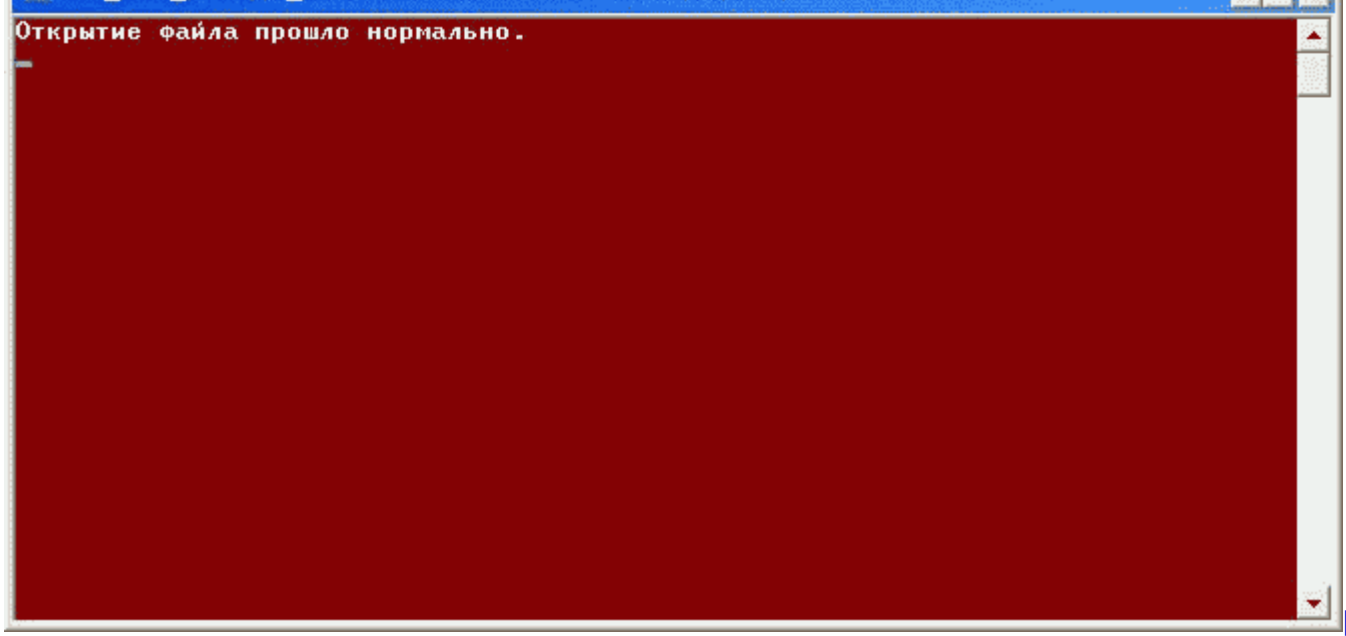


Рис. 1 Пример кода

Таким способом можно найти ошибки которые мешают открытию файла. [9]

Допустим ошибку защиты или записи на жесткий диск. Причем, обнаружить еще до того, как программа попытается в этот файл что-то записать. Поэтому всегда нужно вначале получить подтверждение, что функция fopen() выполнилась успешно, и лишь затем выполнять с файлом другие операции. Выше на Рис.1 приведена небольшаю часть кода программы, в которой подтверждается или нет открытие файла. Результат проверки показан на Рис.2 ниже. [10]



[11]

Рис. 2. Результат проверки

В отсутствие ошибок файл открывается. [12]После того как файл открыт, то для чтения или записи данных используют специальные функции. Они распространяются библиотеками в которые вложены множества функций упрощающих работы с файлами и увеличивающие количество возможных используемых функций при работе с ними.[13]

1. Функция fputc()

Вид функции: fputc(c, fp);

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

1. Функция fputs()

Вид функции: fputs(s, fp);

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

1. Функция fgetc()

Вид функции: char c = fgetc(fp);

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

1. Функция fgets()

Вид функции: fgets(s, maxline, fp);

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

1. Функция fread()

Вид функции: fread(buf, m, n, fp);

Эта функция считывает 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()

Вид функции: fwrite(ptr, m, n, fp);

Данная 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.[14]

1. Функция fseek()

Вид функции: fseek(fp, n, m);

Данная функция устанавливает указатель в файле fp в позицию, отстоящую на n количество байт от текущей, и также направление перемещения (вперед или назад) которое указывается в m, в нем может быть задано одно из трех значений: 0, 1, 2 или одной из трех символических констант, указанных в библиотеке stdio.h:

- SEEK_SET = 0 – указатель к началу файла;
- SEEK_CUR = 1 – указатель на текущую позицию в файле;
- SEEK_END = 2 — указатель к концу файла.[15]

Данная функция fseek() используется для ввода/вывода потоковых данных. 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()

Вид 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()

Вид 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.[16]

1. Функция fprintf()

Вид 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()

Вид 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()

Вид функции: remove(FILENameString);

Данная функция удаляет файл с именем FILENameString. Перед удалением удаляемый файл должен быть закрыт. Строка FILENameString должна содержать полный путь к файлу с сохранением регистров. При отсутствии ошибок и выполнении задания функция возвращает ноль, а в случае ошибки – 1. При этом глобальная переменная errno принимает следующие значения:

- EACCES — невозможно удалить;
- ENOENT — файл с таким именем отсутствует.

1. Функция FILElength()

Вид 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()

Вид 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()

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

Эта функция выставляет файл, который задан в первом параметре, на место выбранного открытого потока. Также функция будет закрывать потом не взирая на его состояние открыт/закрыт. Такая функция может быть полезна для замены файлов связанных со стандартными средствами ввода/вывода stdin, stdout или stderr. Условия необходимые для открытия файла аналигичны функции fopen(). После удачного выполнения задачи функция возвращает указатель FILE (как и функция fopen()), при неуспешном — NULL).[17]

1. Функция feof()

Вид 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.[18]

1. Функция ferror()

Вид 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()

Вид функции: exit(int status);

Данная функция нужна для немедленного завершения всех процессов и работы программы при возникновении ошибки при открытии выбранного файла, и еще по каким-либо другим причинам. Перед тем как завершить программу все другие файлы отключаются и закрываются а данные которые остались в буфере вывода, будут записаны в память и вызываться функциями обработки ошибок, которые будут предварительно зарегистрированы функцией atexit().

Все вышеперечисленные функции являются основными и незаменимыми при работе с файлами. Данные функции предоставляют набор операций позволяющий редактировать, создавать, тестировать и т.д. необходимые пользователю файлы.

Стандартный ввод/вывод

файл операционный система ввод

Когда происходит открытие программы вместе с этим открываются сразу три различных файла:

- файл стандартного ввода.

Указатель для него называется stdin;

- файл стандартного вывода.

Указатель для него называется stdout;

- файл стандартного вывода ошибок.

Указатель для него называется stderr.

Во время работы с файлами можно использовать их для направления данных в стандартные потоки ввода/вывода(клавиатура/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 перечисленного выше списка функций используемых при операциях ввода/вывода, ясно, 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 можно переключить обычный ввод на ввод из файла: если некоторая программа с именем p1.exe использует функцию getchar(), то используя командную строку:

p1.exe < anyFILE1[19]

получим ввод не с клавиатуры, а из файла anyFILE1. Командную строку в С можно использовать 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[20]

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 с (один символ) на стандартное выводное устройство.

1. Функция printf()

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

Это функция форматного вывода. Она 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 перечисленных ниже:

- d – аргумент преобразуется к десятичному виду (с учетом знака);
- x – аргумент преобразуется в беззнаковую шестнадцатеричную форму;
- U – аргумент преобразуется в десятичную форму без знака;
- 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 и 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);

[21]Это функция форматного ввода с клавиатуры. Она осуществляет ввод символов с клавиатуры и преобразует их в соответствии с установленным форматом для каждого значения, указанном в управляющей (форматной) символьной строке Control, а результаты форматирования записываются в аргументы arg1, arg2, ...,

argn. Назначение строки Control тот же, что и в функции printf(). Так как arg1, arg2, ..., argn – это выходные параметры функции, то при обращении к функции они должны задаваться своими адресами: поскольку имя массива — это указатель на его первый элемент, а аргументы, которые не являются указателями, задаются как &arg.[22] Форматная строка Control – является символьной строкой, которая содержит три типа объектов: незначащие символы, значащие символы и спецификации формата. [23]Незначащие символы – это пробел, знак табуляции (\t), символ перехода на новую строку (\n). Когда функция натыкается на незначащий символ в строке формата, она считывает его, но не сохраняет все незначащие символы которые следуют за ним до того, пока не встретится первый значащий символ. 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);

Осуществляет ввод символов с клавиатуры и записывает их в строку s, которую можно объявить 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()[24]

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.[25]

1. Функция clrscr()

Формат функции: clrscr();

Очищает экран и окрашивает задний фон цветом который был указан в функции textbackground().

2. Ввод и вывод в языке C++

После рассмотрения назначений, правил и функций отвечающих за операции ввода/вывода в обычном языке С, можно приступить к теме работы «Ввод и вывод в языке С++». Далее мы будем описывать роль и значение функций отвечающих за ввод и вывод в С++, а также правила их использования.

Функция ввода и вывода выполнена в С++ с помощью классов, в которых содержатся методы и данные необходимые для работы. Все поточные классы происходят от общего предка – класса ios и потому наследуют его функциональность.[26] Для того чтобы начать создание программы с вводом/выводом необходим класс:

#include <fstream>

Данный 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). Также, классы istream, ostream являются cin, cout, cerr, и с их помощью осуществляется стандартный ввод и вывод – стандартным вводом считается ввод данных с клавиатуры, а стандартным выводом вывод информации на 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 (<>).[27]

Для начала работы с файлом необходима сперва его открыть а затем связать с функцией в которой задаем характеристики файла. Например: размер буфера ввода/вывода, состояние файла, последняя прочитанная запись и т. п. Связывание файла выполняет функция open(), входящая в один из классов, который определяет ввод/вывод (fstream, istream, ostream). Поэтому, чтобы выполнить эту функцию, следует сначала создать экземпляр соответствующего класса, чтобы получить доступ к этой функции. Например мы хотим сделать какую-либо запись в файле, тогда следует создать экземпляр класса ostream: ostream exp; и затем

выполнить функцию

exp.open().[28]

В скобках обязательно должны быть указаны параметры 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 необходимая работа с файлом закончена, надо отвязать файл от той структуры к которой его привязали при открытии, закрыв его. Если мы этого не сделаем, то открыть и работать с другим файлом не получится. Для закрытия файла нужно воспользоваться методом close() того же экземпляра класса, который мы создавали, чтобы выполнить функцию open().[29] В нашем случае следовало бы написать:

exp.close();

Файловый ввод/вывод с использованием разных классов

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

- структуры, необходимые для открытия и закрытия файлов;
- функции (методы) открытия и закрытия файлов;
- другие функции и данные, производящие различные виды ввода-вывода;

В языке C++ для использования проточных классов необходимо написать команду:

using namespace::std;

В противном случае программа не пройдет компиляцию. В листинге 1 приводится пример использования директив пространства имен.[30]

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;
}[31]
}
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
```
[32]

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

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 уже открыт, то возвращает true, иначе — false;

rdbuf() – ставит 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. Режим работы с файлом после открытия устанавливается с помощью различных переменных: 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.[33]

Назначение способов открытия файла:

app – открывает файл и позволяет дописать данные в конец файла;

binary – открывает файл в бинарном виде (такие файлы были записаны по определенной структуре данных и поэтому чтобы из прочесть необходимо использовать туже структуру);

in — открыть файл для чтения;

out – открывает файл и позволяет записать данные в его начале, если открываемый файл не существует автоматически создает новый;

trunc – удаляет данные в существующем файле;[34]

ate – утанавливает указатель позиционирования файла в его конце.

При выборе режимов открытия файла можно применять оператор логического ИЛИ (|), чтобы сделать несколько режимов открытия, если требуется. 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()

[35]{

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();

}
```
[36]

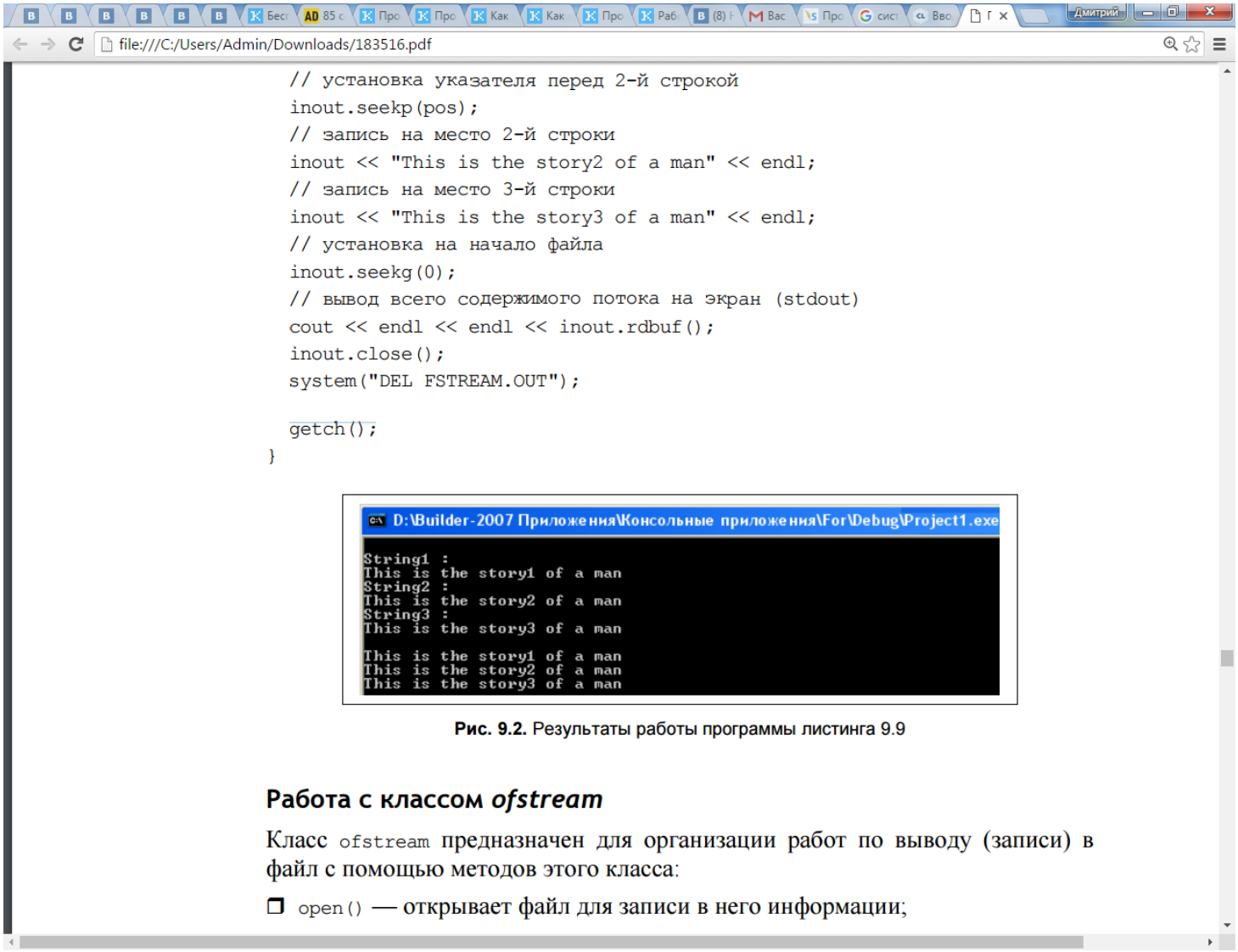


Рис. 1 Результат работы кода

Работа с классом ofstream

Класс ofstream используется для организации задач по выводу (записи) в файл с помощью различных методов входящих в класс:

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

is_open() –проверяет открыт ли файл, 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.[37]

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();
```

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.[38]

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();
```

Стандартный ввод/вывод в C++

Стандартный ввод/вывод является частным случаем файлового ввода/вывода. Во время файлового ввода-вывода мы используем различные поточные классы с их операциями и методами << и >>. Но классы istream, ostream, которые лежат 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).

Во время запуска программы в C++ эти стандартные потоки открыты и по умолчанию назначены на стандартные вводное и выводное устройства —

клавиатуру (cin) и экран (cout и cerr). Причем 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. [39]Например, запишем:

cout << i;

i выводится на экран в определенном формате, который определен по умолчанию для типа i и в минимальном поле.

Запишем:

cin >> i >> j >> s;

здесь i, j, s описаны соответственно как int, float, char. В записи не указаны форматы, но во время ввода с клавиатуры они будут укзаны (после ввода каждого значения надо нажимать клавишу) и их форматы будут учтены.

Объект cout

Этот объект cout отправляет данные в буфер-поток, связанный с объектом stdout, объявленным в файле stdio.h. По умолчанию стандартные потоки C и C++ синхронизированы. Во время вывода, данные могут быть изменены или отформатированы с помощью функций-членов класса или манипуляторов. Перечень их приведен в таблице 1.[40]

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

[41]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();

}
```
[42]

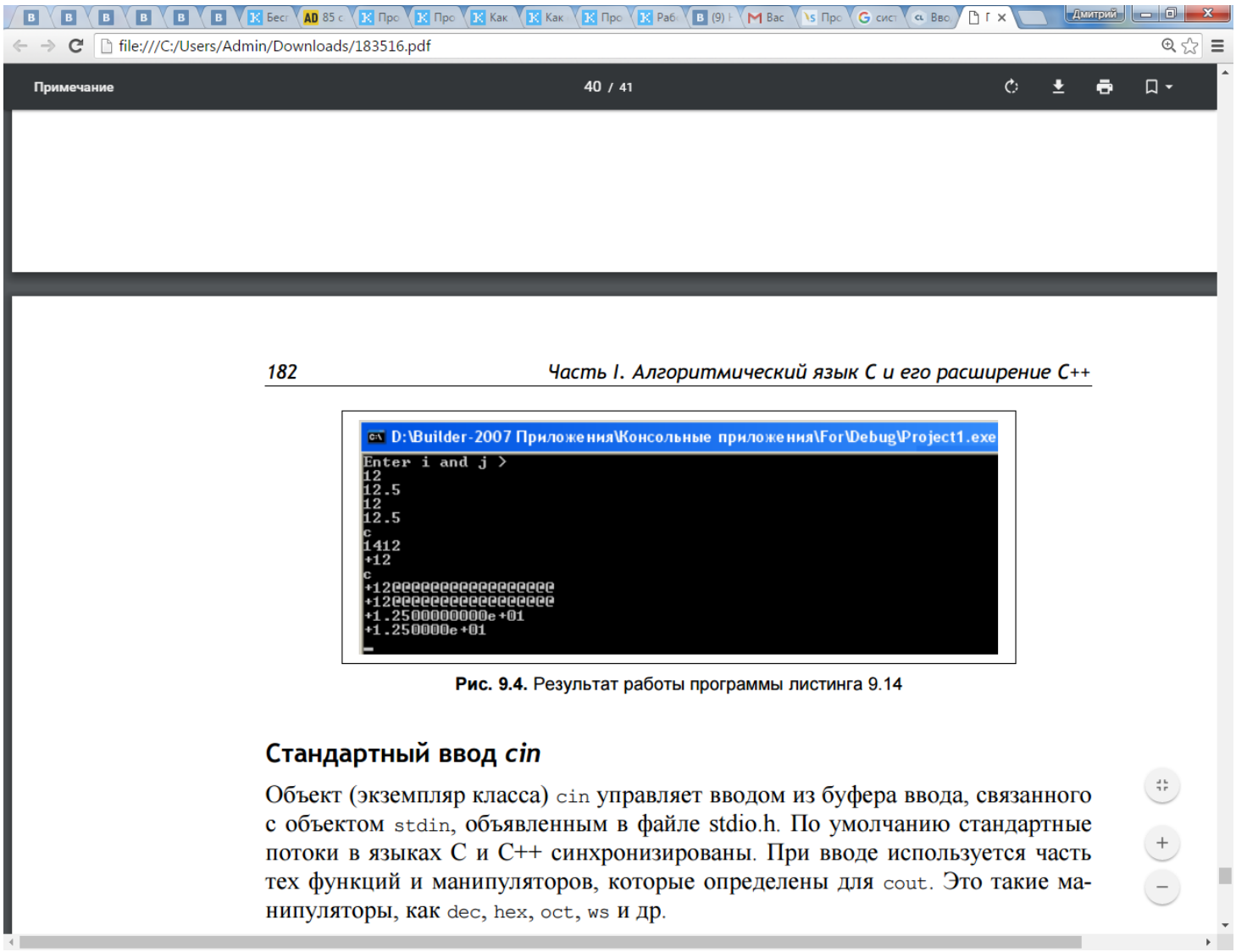


Рис. 2 Результат работы кода

Стандартный ввод cin

Это объект cin который управляет вводом из буфера ввода, связанного с объектом stdin, объявленным в файле stdio.h. По умолчанию стандартные потоки в языках С и С++ 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();

}[43]

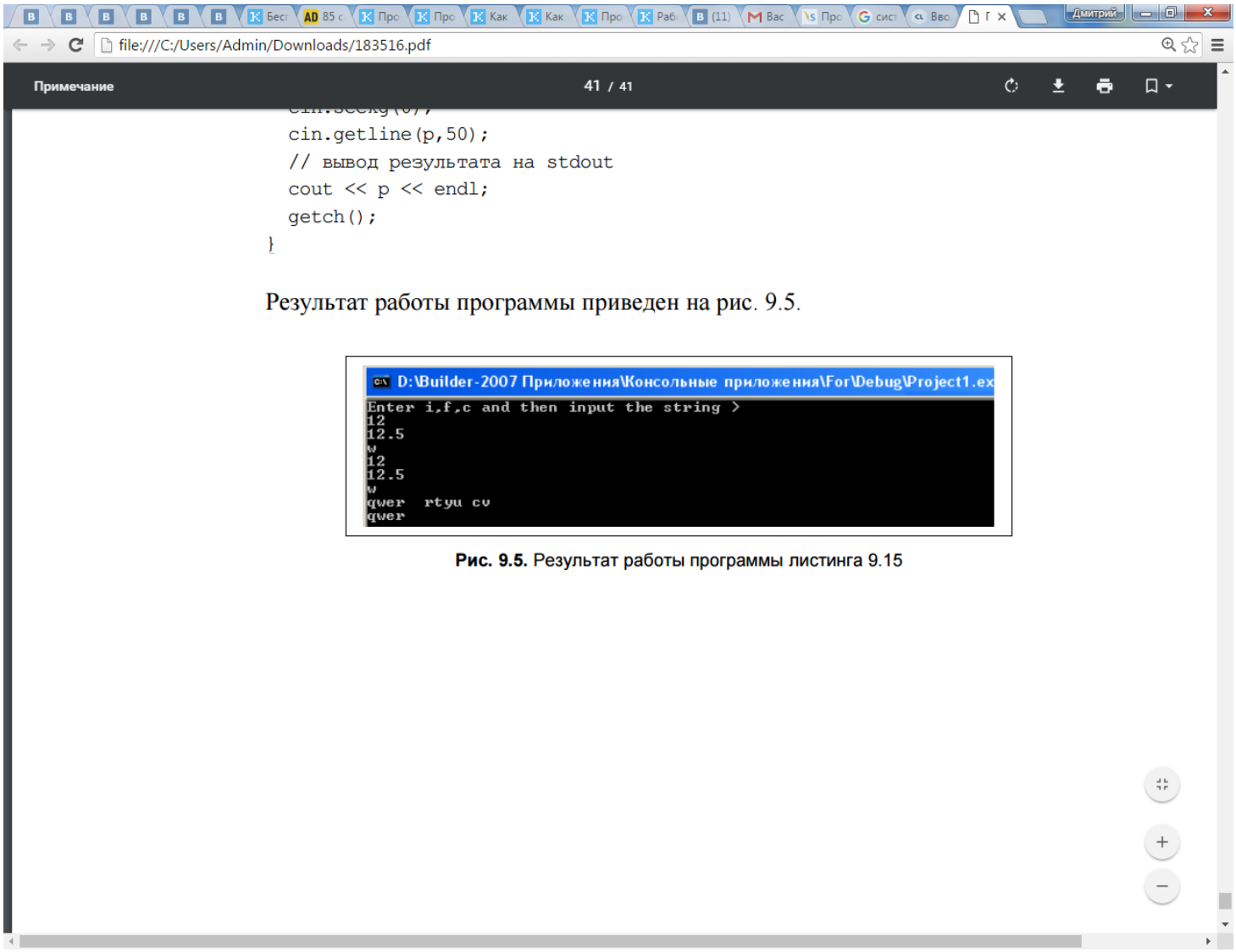


Рис. 3 Результат работы кода

3. Форматный ввод-вывод в C++

Во время работы со стандартными потоками ввода-вывода определены две функции:

printf( ) - форматный вывод;

scanf( ) - форматный ввод.

Прототип функции printf() имеет вид:

int printf(const char *format,...);

При обращении к функции printf() возможно установить два вида задания:

int printf (*форматная строка, список_аргументов);

int printf (указателъ_на_форматную_строку,. список_аргументов); [44]

В обох из них printf() преобразует данные из внутреннего представления в символьный вид в соответствии с форматной строкой и выводит их в выходной поток. Данные, которые преобразуются и выводятся, задаются как аргументы функции printf().

Возвращаемое значение функции printf() – количество напечатанных знаков; а в случае ошибки - отрицательное число.

Также есть форматная строка которая ограничена двойными кавычками куда можно ввести произвольный текст, символы управления и вид работы с данными. Текст и управляющие символы из форматной строки просто копируются в выходной поток.[45] Форматная строка обычно размещается в списке фактических параметров функции, что соответствует первому варианту вызова функции printf(). Во втором же варианте, первый фактический параметр - это указатель типа char *, и форматная строка определена в программе как обычная строковая константа или переменная.

В списке аргументов функции printf() находятся выражения, значения которых надо вывести из программы. Частные случаи этих выражений - переменные и константы. Количество аргументов и их типы должны соответствовать последовательности спецификаций преобразования в форматной строке. Для каждого аргумента должна быть указана точно одна спецификация преобразования. [46]

Если аргументов недостаточно для данной форматной строки, то результат зависит от реализации (от операционной системы и от системы программирования). Если аргументов больше, чем указано в форматной строке, "лишние" аргументы игнорируются. Гарантируется, что при любом количестве параметров и любом их типе после выполнения функций printf() дальнейшее выполнение программы будет корректным. [47]

Спецификация преобразования имеет следующую форму:

% флаги ширина_поля. точностъ спецификатор

Символ % является признаком спецификации преобразования. В спецификации преобразования обязательными являются только два элемента: признак % и спецификатор.

Спецификатор Тип аргумента Формат вывода

d int, char,unsigned Десятичное целое со знаком

u int, char,unsigned Десятичное целое без знака

o int, char,unsigned Восьмеричное целое без знака

x int, char,unsigned Шестнадцатеричное целое без знака; при выводе используются символы “0..9a..f”

X int, char,unsigned Шестнадцатеричное целое без знака; при выводе используются символы "0...9A...F”

f double, float Вещественное значение со знаком в виде:

Знак_числа dddd.dddd

где dddd - одна или более десятичных цифр. Количество цифр перед десятичной точкой зависит от величины выводимого числа, а количество цифр после десятичной точки зависит от требуемой точности. Знак числа при отсутствии модификатора '+' изображается только для отрицательного числа.

Форматный ввод из входного потока.

Форматный ввод из входного потока осуществляется функцией scanf(). Прототип функции scanf( ) имеет вид:

int scanf(const char * format, ...);

При обращении к функции scanf() возможны две формы задания первого параметра:

int scanf (форматная строка, список аргументов);

int scanf(указатель_на_форматную _строку, список_аргументов); [48]

Функция scanf() читает последовательности кодов символов (байты) из входного потока и интерпретирует их в соответствии с форматной_строкой как целые числа, вещественные числа, одиночные символы, строки. В первом варианте вызова функции форматная строка размещается непосредственно в списке фактических параметров. Во втором варианте вызова предполагается, что первый фактический параметр - это указатель типа char *, адресующий собственно форматную строку. Форматная строка в этом случае должна быть определена в программе как обычная строковая константа или переменная. [49]

После преобразования во внутреннее представление данные записываются в области памяти, определенные аргументами, которые следуют за форматной строкой. Каждый аргумент должен быть указателем на переменную, в которую будет записано очередное значение данных и тип которой соответствует типу, указанному в спецификации преобразования из форматной строки.

Если для форматной строки недостаточно аргументов, то результат зависит от реализации (от операционной системы и от системы программирования). Если аргументов больше, чем требуется в форматной строке, "лишние" аргументы игнорируются. [50]

Последовательность кодов символов, которую функция scanf() читает, из входного потока, как правило, состоит из полей (строк), разделенных символами промежутка или обобщенными пробельными символами. Поля просматриваются и вводятся функцией scanf() посимвольно. Ввод поля прекращается, если встретился пробельный символ или в спецификации преобразования точно указано количество вводимых символа.

Функция scanf() завершает работу, если исчерпана форматная строка. При успешном завершении scanf() возвращает количество преобразованных и введенных полей (точнее, количество объектов, получивших значения при вводе). Значение EOF возвращается при возникновении ситуации "конец файла"; значение -1 - при возникновении ошибки преобразования данных.

Рассмотрим форматную строку функции scanf(): "code: %d %*s %c %s" [51]

Строка "code:" присутствует во входном потоке для контроля вводимых данных и поэтому указана в форматной строке. Спецификации преобразования задают следующие действия:

%d - ввод десятичного целого;

%*s - пропуск строки;

%c - ввод одиночного символа;

%s - ввод строки.

Приведем результаты работы программы для трех различных наборов входных данных.

1.Последовательность символов исходных данных:

code: 5 поле2 D asd

Результат выполнения программы:

i=5 c=D s=asd ret=3

Значением переменной ret является код возврата функц»-scanf(). Число 3 говорит о том, что функция scanf() ввела данные без ошибки и было обработано 3 входных поля (строки "code:" и "поле2" пропускаются при вводе). [52]

Заключение

Подведем итоги данной работы. В ходе повествования были рассмотрены функции и особенности операций ввода/вывода в языках C и C++. Были описаны понятия «функции» и «потока», их разновидности и особенности использования и функционирования в этих языках программирования.

Более пристальное внимание было уделено потоковому вводу-выводу символов, особенностям его работы и правильного использования. В работе в качестве примеров приведены фрагменты кода программ, которые показывают особенности реализации тех или иных функций ввода-вывода.

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.

1. Миков А. И., Королев Л.Н., Информатика. Введение в компьютерные науки. — Абрис, Высшая школа, 2012. ↑

2. Ахо Альфред В., Хопкрофт В., Ульман Джеффри Д. Структуры данных и алгоритмы — М.: Вильямс, 2016. ↑

3. Кортмен Т.Х Алгоритмы: построение и анализ — М.: Вильямс, 2013 ↑

4. Миков А. И., Королев Л.Н., Информатика. Введение в компьютерные науки. — Абрис, Высшая школа, 2012. ↑

5. Ахо Альфред В., Хопкрофт В., Ульман Джеффри Д. Структуры данных и алгоритмы — М.: Вильямс, 2016. ↑

6. Кортмен Т.Х Алгоритмы: построение и анализ — М.: Вильямс, 2013 ↑

7. Кортмен Т.Х Алгоритмы: построение и анализ — М.: Вильямс, 2013 ↑

8. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. ↑

9. Кортмен Т.Х Алгоритмы: построение и анализ — М.: Вильямс, 2013 ↑

10. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. ↑

11. ↑

12. Кортмен Т.Х Алгоритмы: построение и анализ — М.: Вильямс, 2013 ↑

13. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. ↑

14. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. ↑

15. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. ↑

16. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. ↑

17. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. ↑

18. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. ↑

19. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. Стр 148. ↑

20. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. ↑

21. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. ↑

22. Ахо Альфред В., Хопкрофт В., Ульман Джеффри Д. Структуры данных и алгоритмы — М.: Вильямс, 2016. ↑

23. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. ↑

24. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. ↑

25. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. ↑

26. Мейерс С. Эффективное использование C++. 55 верных советов улучшить структуру и код ваших программ: Пер. с англ. — 3-е изд. — М.: ДМК пресс, 2006. ↑

27. Мейерс С. Эффективное использование C++. 55 верных советов улучшить структуру и код ваших программ: Пер. с англ. — 3-е изд. — М.: ДМК пресс, 2006. ↑

28. Страуструп Б. Язык программирования C++: Пер. с англ. — 3-е спец. изд. — М.: Бином, 2003. ↑

29. Мейерс С. Эффективное использование C++. 55 верных советов улучшить структуру и код ваших программ: Пер. с англ. — 3-е изд. — М.: ДМК пресс, 2006. ↑

30. Страуструп Б. Язык программирования C++: Пер. с англ. — 3-е спец. изд. — М.: Бином, 2003. ↑

31. Джосьютис Н. М. C++. Стандартная библиотека. Для профессионалов: Пер. с англ. — СПб.: Питер, 2004. ↑

32. Страуструп Б. Язык программирования C++: Пер. с англ. — 3-е спец. изд. — М.: Бином, 2003. ↑

33. Джосьютис Н. М. C++. Стандартная библиотека. Для профессионалов: Пер. с англ. — СПб.: Питер, 2004. ↑

34. Джосьютис Н. М. C++. Стандартная библиотека. Для профессионалов: Пер. с англ. — СПб.: Питер, 2004. ↑

35. Страуструп Б. Язык программирования C++: Пер. с англ. — 3-е спец. изд. — М.: Бином, 2003. ↑

36. Страуструп Б. Язык программирования C++: Пер. с англ. — 3-е спец. изд. — М.: Бином, 2003. ↑

37. Страуструп Б. Язык программирования C++: Пер. с англ. — 3-е спец. изд. — М.: Бином, 2003. ↑

38. Страуструп Б. Язык программирования C++: Пер. с англ. — 3-е спец. изд. — М.: Бином, 2003. ↑

39. Страуструп Б. Язык программирования C++: Пер. с англ. — 3-е спец. изд. — М.: Бином, 2003. ↑

40. Страуструп Б. Язык программирования C++: Пер. с англ. — 3-е спец. изд. — М.: Бином, 2003. ↑

41. Джосьютис Н. М. C++. Стандартная библиотека. Для профессионалов: Пер. с англ. — СПб.: Питер, 2004. ↑

42. Джосьютис Н. М. C++. Стандартная библиотека. Для профессионалов: Пер. с англ. — СПб.: Питер, 2004. ↑

43. Джосьютис Н. М. C++. Стандартная библиотека. Для профессионалов: Пер. с англ. — СПб.: Питер, 2004. ↑

44. Мейерс С. Эффективное использование C++. 55 верных советов улучшить структуру и код ваших программ: Пер. с англ. — 3-е изд. — М.: ДМК пресс, 2006. ↑

45. Страуструп Б. Язык программирования C++: Пер. с англ. — 3-е спец. изд. — М.: Бином, 2003. ↑

46. Мейерс С. Эффективное использование C++. 55 верных советов улучшить структуру и код ваших программ: Пер. с англ. — 3-е изд. — М.: ДМК пресс, 2006. ↑

47. Страуструп Б. Язык программирования C++: Пер. с англ. — 3-е спец. изд. — М.: Бином, 2003. ↑

48. Страуструп Б. Язык программирования C++: Пер. с англ. — 3-е спец. изд. — М.: Бином, 2003. ↑

49. Мейерс С. Эффективное использование C++. 55 верных советов улучшить структуру и код ваших программ: Пер. с англ. — 3-е изд. — М.: ДМК пресс, 2006. ↑

50. Мейерс С. Эффективное использование C++. 55 верных советов улучшить структуру и код ваших программ: Пер. с англ. — 3-е изд. — М.: ДМК пресс, 2006. ↑

51. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. ↑

52. Мейерс С. Эффективное использование C++. 55 верных советов улучшить структуру и код ваших программ: Пер. с англ. — 3-е изд. — М.: ДМК пресс, 2006. ↑