[1. Функции для работы с файлами.](#_Toc515770698)

[1.1. Полезные функции для работы с файлами](#_Toc515770699)

[1.1.1. clearerr()](#_Toc515770700)

[1.1.2. fclose()](#_Toc515770701)

[1.1.3. fcloseall()](#_Toc515770702)

[1.1.4. fdopen()](#_Toc515770703)

[1.1.5. feof()](#_Toc515770704)

[1.1.6. ferror()](#_Toc515770705)

[1.1.7. fflush()](#_Toc515770706)

[1.1.8. fgetc()](#_Toc515770707)

[1.1.9. fgetchar()](#_Toc515770708)

[1.1.10. fgetpos()](#_Toc515770709)

[1.1.11. fgets()](#_Toc515770710)

[1.1.12. filelength()](#_Toc515770711)

[1.1.13. fileno()](#_Toc515770712)

[1.1.14. flushall() (fflush())](#_Toc515770713)

[1.1.15. fopen()](#_Toc515770714)

[1.1.16. fprintf()](#_Toc515770715)

[1.1.17. fputc()](#_Toc515770716)

[1.1.18. fputchar()](#_Toc515770717)

[1.1.19. fputs()](#_Toc515770718)

[1.1.20. fread()](#_Toc515770719)

[1.1.21. freopen()](#_Toc515770720)

[1.1.22. fscanf()](#_Toc515770721)

[1.1.23. fseek()](#_Toc515770722)

[1.1.24. fsetpos()](#_Toc515770723)

[1.1.25. fstat()](#_Toc515770724)

[1.1.26. ftell()](#_Toc515770725)

[1.1.27. fwrite()](#_Toc515770726)

[1.1.28. getc()](#_Toc515770727)

[1.1.29. getchar()](#_Toc515770728)

[1.1.30. gets()](#_Toc515770729)

[1.1.31. getw()](#_Toc515770730)

[1.1.32. perror()](#_Toc515770731)

[1.1.33. printf()](#_Toc515770732)

[1.1.34. putc()](#_Toc515770733)

[1.1.35. putchar()](#_Toc515770734)

[1.1.36. puts()](#_Toc515770735)

[1.1.37. putw()](#_Toc515770736)

[1.1.38. remove()](#_Toc515770737)

[1.1.39. rename()](#_Toc515770738)

[1.1.40. rewind()](#_Toc515770739)

[1.1.41. scanf()](#_Toc515770740)

[1.1.42. setbuf()](#_Toc515770741)

[1.1.43. setvbuf()](#_Toc515770742)

[1.1.44. sprintf()](#_Toc515770743)

[1.1.45. sscanf()](#_Toc515770744)

[1.1.46. tmpfile()](#_Toc515770745)

[1.1.47. tmpnam()](#_Toc515770746)

[1.1.48. ungetc()](#_Toc515770747)

[1.1.49. vfprintf()](#_Toc515770748)

[1.1.50. vfscanf()](#_Toc515770749)

[1.1.51. vprintf()](#_Toc515770750)

[1.1.52. vsprintf()](#_Toc515770751)

[1.1.53. vsscanf()](#_Toc515770752)

[1.2. Низкоуровневый ввод и вывод](#_Toc515770753)

[1.2.1. close()](#_Toc515770754)

[1.2.1. lseek()](#_Toc515770755)

[1.2.2. open()](#_Toc515770756)

[1.2.1. read()](#_Toc515770757)

[1.2.1. unlink() (\_unlink())](#_Toc515770758)

[1.2.2. write()](#_Toc515770759)

1. Функции для работы с файлами.

Таблица 1.1. Функции потокового ввода и вывода

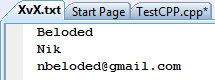
|  |  |
| --- | --- |
| **Функция** | **Описание** |
| clearer() | Сбрасывает индикатор ошибок потока и устанавливает в ноль индикатор конца файла (end-of-file) |
| fclose() | Закрывает поток |
| fcloseall() | Закрывает все потоки |
| fdopen() | Открывает поток, используя его описатель (handle), полученный при помощи creat, dup, dup2 или open |
| feof() | Проверяет поток на признак конца файла |
| ferror() | Проверяет поток на признак конца файла |
| fflush() | Проверяет поток на признак конца файла |
| fgetc() | Считывает символ из потока |
| fgetchar() | Считывает символ из потока stdin |
| fgetpos() | Возвращает текущий указатель файла |
| fgets() | Возвращает строку из потока |
| filelength() | Возвращает размер потока в байтах |
| fileno() | Возвращает описатель файла, связанного с потоком |
| flushall() | Очищает буферы всех потоков |
| fopen() | Открывает поток |
| fprintf() | Выполняет форматированную запись в поток |
| fputc() | Записывает символ в поток |
| fputchar() | Записывает символ в stdout |
| fputs() | Записывает строку в поток |
| fread() | Считывает неформатированные данные из потока |
| freopen() | Переназначает указатель потока |
| fscanf() | Считывает форматированные данные из потока |
| fseek() | Позиционирует указатель файла в указанную позицию |
| fsetpos() | Позиционирует указатель файла потока |
| fstat() | Возвращает информацию об открытом файле |
| ftell() | Возвращает положение текущего указателя файла |
| fwrite() | Записывает неформатированные данные в поток |
| getc() | Этот макрос считывает символ из потока |
| getchar() | Этот макрос считывает символ из stdin |
| gets() | Возвращает строку из stdin |
| getw() | Считывает целое число из потока |
| perror() | Выдает системную ошибку на stderr |
| printf() | Записывает форматированные данные в stdout |
| putc() | Этот макрос записывает символ в поток |
| putchar() | Этот макрос записывает символ в stdout |
| puts() | Записывает строку в stdout |
| putw() | Записывает целое число в поток |
| remove() | Удаляет файл |
| rename() | Переименует файл |
| rewind() | Позиционирует указатель файла на начало потока |
| scanf() | Сканирует и вводит форматированные данные из stdln |
| setbuf() | Отменяет автоматическую буферизацию, позволяя приложению определить собственный буфер для потока |
| setvbuf() | То же самое, что и setbuf(); позволяет также определить размер буфера |
| sprintf() | Записывает форматированные данные в строку |
| sscanf() | Сканирует и вводит форматированные данные из строки |
| tmpnam() | Создает в заданном подкаталоге файл с уникальным именем |
| ungetch() | Засылает символ в буфер клавиатуры |
| vfprintf() | Записывает форматированные данные в поток, используя указатель на форматирующую строку |
| vfscanf() | Сканирует и форматирует ввод из потока, используя указатель на форматирующую строку |
| vprintf() | Записывает форматированные данные в stdout, используя указатель на форматирующую строку |
| vscanf() | Сканирует и форматирует ввод из stdin, используя указатель на форматирующую строку |
| vsprintf() | Записывает форматированные данные в строку, используя указатель на форматирующую строку |
| vsscanf() | Сканирует и форматирует ввод строки, используя указатель на форматирующую строку |

Таблица 1.2. Функции низкоуровневого ввода и вывода

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| close() | Закрывает дисковый файл |
| lseek() | Поиск указанного байта в файле |
| open() | Открывает дисковый файл |
| read() | Читает данные в буфер |
| unlink() | Удаляет файл из подкаталога |
| write() | Записывает буфер данных |

## Полезные функции для работы с файлами

file:///D:/SUBJECTS/\_AP\_AlgorithmProgramming/AlgorithmProgramming%2004\_Aktive/x3\_%D0%A0%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%D1%81%20%D1%84%D0%B0%D0%B9%D0%BB%D0%B0%D0%BC%D0%B8/\_cstdio\_%20\_%20XoaX.net%20Video%20Tutorials.html



### clearerr()

void **clearerr(FILE\* qpStream);**

Эта функция очищает все индикаторы ошибок в объекте qpStream и состоянии конца файла.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

// Read from the output stream to create an error.

**getc(stdout);**

// Verify that the stream has an error.

if **(ferror(stdout)) {**

**perror(**"1.Error in stdout");

**}** else **{**

**printf(**"1.No error in stdout\n");

**}**

// Clear the stream

**clearerr(stdout);**

// Verify that the error was cleared.

if **(ferror(stdout)) {**

**perror(**"2.Error in stdout");

**}** else **{**

**printf(**"2.No error in stdout\n");

**}**

**getchar();**

return **0;**

}



### fclose()

int **fclose(FILE\* qpStream);**

Эта функция используется для закрытия потока, когда вы закончите использовать его. Аргумент «qpStream» - это поток, который закрывается. Функция возвращает 0, если поток был успешно закрыт и EOF указывает на ошибку.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

**FILE\* qpFile;**

char\* cpFileName = "XvX.txt";

**qpFile = fopen(cpFileName,** "r" **);**

// Verify that we opened the file.

if **(qpFile) {**

**printf(**"Opened the file %s for reading \n", cpFileName);

**}** else **{**

**printf(**"Could not open %s for reading \n", cpFileName);

**getchar();**

return **1;**

**}**

// Now close the file and verify that it closed.

int **iCloseResult = fclose(qpFile);**

if **(iCloseResult == 0) {**

**printf(**"Closed the file %s\n", cpFileName);

**}** else **{**

**printf(**"Could not close the file %s\n", cpFileName);

**}**

**getchar();**

return **0;**

}



### fcloseall()

int **fcloseall(**void);

Иногда мы открываем несколько файлов для обработки, тогда утомительная задача - закрыть все открытые потоки один за другим после использования. C обеспечивает мощную функцию для закрытия всех открытых потоков с использованием единого метода.

#include<cstdio>

usingnamespace **std;**

void **main()**

{

int **streams\_closed;**

**fopen(**"ONE.txt","w");

**fopen(**"TWO.txt","w");

**streams\_closed = fcloseall();**

if **(streams\_closed == EOF)**

**printf(**"Error");

else

**printf(**"%d Streams Were Closed", streams\_closed);

**getchar();**

}



### fdopen()

FILE\* fp = fdopen(fd, "w");

fdopen () использовать дескриптор файла для указателя на файл. Функция fdopen () связывает поток с файловым дескриптором. Дескрипторы файлов получают из open (), dup (), creat () или pipe (), которые открывают файлы, но не возвращают указатели на поток структуры FILE. Потоки необходимы для почти всех подпрограмм библиотеки stdio.

#include<fcntl.h>

#include<cstdio>

#include<io.h>

#include<stdio.h>

#include<stdlib.h>

#include<errno.h>

usingnamespace **std;**

int **main()**

{

int **fd;**

**FILE \*fp;**

**fd = open(**"my\_file.txt", O\_WRONLY | O\_CREAT | O\_TRUNC);

if(fd<0){

**printf(**"open call fail");

return **-1;**

**}**

**fp=fdopen(fd,**"w");

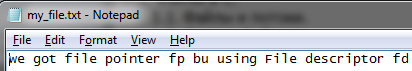
**fprintf(fp,**"we got file pointer fp bu using File descriptor fd");

**fclose(fp);**

**getchar();**

return **0;**

}



### feof()

int **feof(FILE\* qpStream);**

Эта функция проверяет «qpStream» для условия конца файла, сигнализируемого попыткой прочитать за пределами файла. Функция возвращает 0, чтобы сигнализировать, что никакая попытка не была сделана для чтения за пределами файла и ненулевого значения в противном случае.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

**FILE\* qpFile = fopen(cpFileName,** "r" **);**

// Check that the file was opened

if **(!qpFile) {**

**printf(**"Could not open %s for reading \n", cpFileName);

return **1;**

**}**

char **caReadBuffer[100];**

int **iIndex = 0;**

// Get characters until the we hit the end of the file.

while **(!feof(qpFile)) {**

**caReadBuffer[iIndex] = (**char)getc(qpFile);

**++iIndex;**

**}**

// Add a null terminator

**caReadBuffer[iIndex] = 0;**

// Output the read contents of the file

**printf(**"Text of %s:\n%s\n", cpFileName, caReadBuffer);

**fclose(qpFile);**

**getchar();**

return **0;**

}



### ferror()

int **ferror(FILE\* qpStream);**

Эта функция используется для проверки условий ошибок в потоке. Аргумент «qpStream» передается для тестирования. Если в «qpStream» произошла ошибка, тогда возвращаемое значение отличное от нуля; иначе функция возвращает 0.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

**FILE\* qpFile = fopen(cpFileName,** "r" **);**

// Check that the file was opened

if **(!qpFile) {**

**printf(**"Could not open %s for reading \n", cpFileName);

**getchar();**

return **1;**

**}**

// Create an error by trying to write to the read only file

**putc(**'a', qpFile);

// Get the error type

int **iResult = ferror(qpFile);**

if **(iResult != 0) {**

**printf(**"Reading Error: %i\n", iResult);

**getchar();**

return **1;**

**}**

**fclose(qpFile);**

**getchar();**

return **0;**

}



### fflush()

int **fflush(FILE\* qpStream);**

Эта функция очищает буфер буферизованного потока. Буфер для аргумента «qpStream» очищается при вызове этой функции. Если функция выполнена успешно и буфер сброшен, возвращается нулевое значение. В противном случае функция возвращает EOF для указания ошибки. Буфер для потока выходных файлов отправляется в файл, и буфер для входного потока просто очищается. Передача NULL в функцию сбрасывает все открытые выходные потоки. Буферы естественно сбрасываются, когда они заполняются, поток файлов закрывается или программа заканчивается.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

**FILE\* qpFile = fopen(cpFileName,** "r+" **);**

// Verify that the file was opened.

if **(!qpFile) {**

**printf(**"Could not open the file %s\n", cpFileName);

**getchar();**

return **1;**

**}**

// This writes over the first four bytes

**fputs(**"XoaX", qpFile);

// Without this flush, fgets writes garbage to our file

**fflush(qpFile);**

char **caBuffer[10];**

// Read the remaining bytes of the file into a buffer

**fgets(caBuffer, 5, qpFile);**

**printf(**"Read in: %s\n", caBuffer);

int **iResult = ferror(qpFile);**

if **(iResult != 0) {**

**printf(**"Reading Error: %i\n", iResult);

**getchar();**

return **1;**

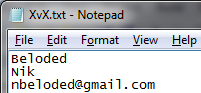
**}**

**fclose(qpFile);**

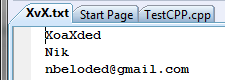
**getchar();**

return **0;**

}







### fgetc()

int **fgetc(FILE\* qpStream);**

Читает в одном символе из «qpStream» и возвращает его значение. Если чтение было успешным, возвращается символ, который был прочитан. В противном случае функция возвращает EOF для указания ошибки.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

**FILE\* qpFile = fopen(cpFileName,** "r" **);**

// Check that the file could be opened

if **(!qpFile) {**

**printf(**"Could not open %s for reading \n", cpFileName);

return **1;**

**}**

// Read the first byte of the file.

int **iFirstChar = fgetc(qpFile);**

if **(iFirstChar != EOF) {**

// Output it as a char.

**printf(**"Byte Read: %c\n", iFirstChar);

**}** else **{**

**printf(**"Read Error");

**}**

**fclose(qpFile);**

**getchar();**

return **0;**

}



### fgetchar()

int **fgetchar(** void **);**

Функция fgetchar () такая же, как fgetc () с аргументом stdin.

#include<fcntl.h>

#include<cstdio>

#include<io.h>

#include<stdio.h>

#include<stdlib.h>

#include<errno.h>

usingnamespace **std;**

int **main()**

{

**FILE \*fp;**

int **c;**

**fp = freopen(** "XvX.txt", "r", stdin );

if( fp != NULL ) {

while( (c = fgetchar()) != EOF ) {

**fputchar(c);**

**}**

**fclose( fp );**

**getchar();**

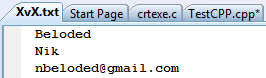
return **EXIT\_SUCCESS;**

**}**

**getchar();**

return **EXIT\_FAILURE;**

}





### fgetpos()

int **fgetpos(FILE\* qpStream, fpos\_t\* qpPos);**

Эта функция устанавливает объект, на который указывает «qpPos», на индикатор позиции файла «qpStream». Функция возвращает ноль, если она была успешной, и ненулевое значение в противном случае. Если функция не работает, она также устанавливает значение errno для указания типа ошибки: EBADF или EINVAL.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

**FILE\* qpFile = fopen(cpFileName,** "r" **);**

**fpos\_t iPos = 0;**

// Get the initial position

int **iResult = fgetpos(qpFile, &iPos);**

if **(iResult == 0) {**

**printf(**"Position: %i \n", iPos);

**}**

// Read the first byte of the file to advance the file pointer.

int **iFirstChar = fgetc(qpFile);**

// Get the position after the read

**iResult = fgetpos(qpFile, &iPos);**

if **(iResult == 0) {**

**printf(**"Position: %i \n", iPos);

**}**

**fclose(qpFile);**

**getchar();**

return **0;**

}



### fgets()

char\* fgets(char\* cpBuffer, int **iMaxRead, FILE\* qpStream);**

Эта функция считывает байты «iMaxRead» в «cpBuffer» из «qpStream». Чтение останавливается при возникновении любого из следующих событий; читается символ новой строки, заканчивается конец пара или когда (iMaxRead - 1) символы считываются в буфер. Функция возвращает указатель «cpBuffer», если чтение выполнено успешно и NULL (= 0), в противном случае.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

**FILE\* qpFile = fopen(cpFileName,** "r" **);**

char **caBuffer[50];**

// Read in the first and only line of text

char\* cpReturn = fgets(caBuffer, 50, qpFile);

if **(cpReturn) {**

**printf(**"Read in: %s\n", caBuffer);

**}**

**fclose(qpFile);**

**getchar();**

return **0;**

}



### filelength()

long **filelength(**int **handle);**

Функция filelengtn() не определена стандартом ANSI С. Она возвращает выраженную в байтах длину файла, ассоциированного с дескриптором файла handle. Надо помнить, что возвращаемая переменная имеет тип long. В случае ошибки возвращается — 1L и errno устанавливается в EBADF, что означает недействительный дескриптор файла.

#include<io.h>

#include<fcntl.h>

#include<cstdio>

#include<io.h>

#include<stdio.h>

#include<stdlib.h>

#include<errno.h>

usingnamespace **std;**

int **main()**

{

**FILE\* fp = fopen(**"XvX.txt", "r");

int **fn = fileno(fp);**

if( fp != NULL ) {

**printf(**"The file is %ld bytes long.", filelength(fn));

**fclose(fp);**

**getchar();**

return **EXIT\_SUCCESS;**

**}**

**getchar();**

return **EXIT\_FAILURE;**

}



### fileno()

int **fileno(FILE \*stream);**

Функция fileno () должна возвращать дескриптор целочисленного файла, связанный с потоком, на который указывает поток.

#include<io.h>

#include<fcntl.h>

#include<cstdio>

#include<io.h>

#include<stdio.h>

#include<stdlib.h>

#include<errno.h>

usingnamespace **std;**

int **main()**

{

**FILE\* fp = fopen(**"XvX.txt", "r");

int **fn = fileno(fp);**

if( fp != NULL ) {

**printf(**"The file 'XvX.txt' has the integer descriptor %d", fn);

**fclose(fp);**

**getchar();**

return **EXIT\_SUCCESS;**

**}**

**getchar();**

return **EXIT\_FAILURE;**

}



### flushall() (fflush())

int **flushall(** void **);**

Функция flushall() очищает все буферы, связанные с входными потоками, и записывает любые буферы, связанные с выходными потоками. Последующая операция чтения во входном файле заставляет новые данные считываться из соответствующего файла или устройства.

Вызов функции flushall() эквивалентен вызову fflush() для всех файлов открытого потока.

#include<io.h>

#include<fcntl.h>

#include<cstdio>

#include<io.h>

#include<stdio.h>

#include<stdlib.h>

#include<errno.h>

usingnamespace **std;**

int **main()**

{

int **n;**

char **c;**

**printf(**"Enter a number: ");

**scanf(**"%d",&n);

**printf(**"Enter a character: ");

**flushall();** //(or) fflush(stdin);

**scanf(**"%c",&c);

**printf(**"Number= %d\n", n);

**printf(**"Character= %c\n", c);

**printf(** "The number of open files is %d\n", flushall());

**getchar();**

return **0;**

}



### fopen()

FILE\* fopen(constchar\* kcpFilename, constchar\* kcpMode);

Эта функция используется для открытия файла для ввода и / или вывода. Аргумент «fcpFilename» обозначает имя файла, который нужно открыть, а «kcpMode» обозначает тип запрашиваемого доступа. Если файл успешно открыт, функция возвращает указатель на объект FILE. В противном случае возвращается указатель NULL.

|  |  |
| --- | --- |
| Строка режима | Описание режима |
| "a" | Это открывает файл для записи в конце файла (добавление) или создает новый файл, если он еще не существует. |
| "a+" | Это открывает файл для записи в конце файла (добавление) или создает новый файл, если он еще не существует. Он также открывает файл для чтения. |
| "r" | Это открывает файл только для чтения. Файл должен существовать до открытия. |
| "r+" | Это открывает файл для чтения и записи. Файл должен существовать до открытия. |
| "w" | Это открывает файл для записи только. Если файл существует, он удаляется при открытии. |
| "w+" | Это открывает файл для чтения и записи. Если файл существует, он удаляется при открытии. |
| "b" | Это открывает файл в двоичном режиме, в отличие от текстового режима по умолчанию ASCII. |
| "ccs" | Это открывает файл в текстовом режиме UNICODE. |

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for reading

**FILE\* qpFile = fopen(cpFileName,** "r" **);**

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open %s for reading \n", cpFileName);

return **1;**

**}**

char **caBuffer[50];**

// Read in the first and only line of text

char\* cpReturn = fgets(caBuffer, 50, qpFile);

if **(cpReturn) {**

**printf(**"Read in: %s\n", caBuffer);

**}**

**fclose(qpFile);**

**getchar();**

return **0;**

}



### fprintf()

int **fprintf(FILE\* qpStream,** constchar\* kcpFormatString, ...);

Эта функция записывает форматированную строку «kcpFormatString» в поток, указанный аргументом «qpStream». Информацию о форматировании строк см. На странице printf. В случае успеха функция возвращает количество записанных байтов. Если возникает ошибка, функция возвращает отрицательное число.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for appending

**FILE\* qpFile = fopen(cpFileName,** "a" **);**

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open %s for writing\n", cpFileName);

return **1;**

**}**

// Append the copyright to the file,

// with a decimal number and string

char **caBuf[] =** "XvX";

int **iReturn = fprintf(qpFile,** "\nCopyright %d %s", 2009, caBuf);

if **(iReturn < 0) {**

**printf(**"Write Error");

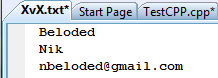
**}**

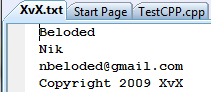
**fclose(qpFile);**

**getchar();**

return **0;**

}





### fputc()

int **fputc(**int **iChar, FILE\* qpStream);**

Эта функция записывает символ «iChar» в поток, указанный «qpStream». Если функция не удалась, возвращается значение EOF.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for appending

**FILE\* qpFile = fopen(cpFileName,** "a" **);**

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open %s for writing \n", cpFileName);

return **1;**

**}**

// Append a plus to the end of the file

int **iResult = fputc((**int)'+', qpFile);

if **(iResult == EOF) {**

**printf(**"Write Error");

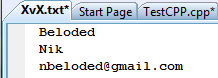
**}**

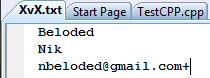
**fclose(qpFile);**

**getchar();**

return **0;**

}





### fputchar()

int **fputchar(** int **c );**

Функция fputchar () записывает символ, указанный аргументом c, в выходной поток stdout. Эта функция идентична функции putchar (). Это эквивалентно: fputc (c, stdout);

#include<io.h>

#include<fcntl.h>

#include<cstdio>

#include<io.h>

#include<stdio.h>

#include<stdlib.h>

#include<errno.h>

usingnamespace **std;**

int **main()**

{

**FILE \*fp;**

int **c;**

**fp = fopen(**"XvX.txt", "r");

if(fp != NULL) {

**c = fgetc(fp);**

while(c != EOF) {

**fputchar(c);**

**c = fgetc(fp);**

**}**

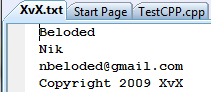
**fclose(fp);**

**}**

**getchar();**

return **0;**

}





### fputs()

int **fclose(FILE\* qpStream);**

Эта функция используется для закрытия потока, когда вы закончите использовать его. Аргумент «qpStream» - это поток, который закрывается. Функция возвращает 0, если поток был успешно закрыт и EOF указывает на ошибку.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for appending

**FILE\* qpFile = fopen(cpFileName,** "a" **);**

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open %s for reading \n", cpFileName);

return **1;**

**}**

// Append a plus to the end of the file

int **iResult = fputs(**" C++ programming", qpFile);

if **(iResult == EOF) {**

**printf(**"Write Error");

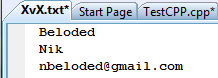
**}**

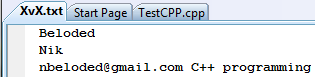
**fclose(qpFile);**

**getchar();**

return **0;**

}





### fread()

size\_t fread(void\* vpBuffer, size\_t qSize, size\_t qCount, FILE\* qpStream);

Эта функция считывает данные из «qpStream» в буфер «vpBuffer». Аргумент «qSize» определяет размер в байтах каждого прочитанного элемента и «qCount» указывает количество элементов. Функция возвращает это количество байтов, которые были фактически прочитаны. Это должно быть qSize \* qCount, но может быть меньше, если возникает ошибка или чтение достигает конца файла.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for reading.

**FILE\* qpFile = fopen(cpFileName,** "r" **);**

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open %s for reading \n", cpFileName);

return **1;**

**}**

char **caBuffer[10];**

// Read in the first four characters of the file.

**size\_t qReturn = fread(caBuffer, 1, 4, qpFile);**

// Add a null terminator.

**caBuffer[qReturn] =** '\0';

if **(qReturn > 0) {**

**printf(**"Read in: %s\n", caBuffer);

**}**

**fclose(qpFile);**

**getchar();**

return **0;**

}



### freopen()

FILE\* freopen(constchar\* kcpPath, constchar\* kcpMode, FILE\* qpStream);

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for reading and

// redirect stdin to it.

**FILE\* qpFile = freopen(cpFileName,** "r", stdin);

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open %s\n", cpFileName);

**getchar();**

return **1;**

**}**

// Use stdin to read the first char of the file

int **iReturn = getchar();**

if **(iReturn != EOF) {**

**printf(**"Read in: %c\n", (char)iReturn);

**}**

**fclose(qpFile);**

**getchar();**

return **0;**

}



### fscanf()

int **fscanf(FILE\* qpStream,** constchar\* kcpFormatString, ...);

Эта функция считывает форматированную строку «kcpFormatString» из потока, заданного аргументом «qpStream». Информацию о форматировании строк см. На странице scanf. В случае успеха функция возвращает количество полей, которые были успешно прочитаны и назначены. Если возникает ошибка или конец потока достигнут до первого назначения, функция возвращает EOF.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for reading.

**FILE\* qpFile = fopen(cpFileName,** "r" **);**

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open %s for reading \n", cpFileName);

return **1;**

**}**

char **cBuffer[50];**

// Read in the first and only string of the file.

int **iRet = fscanf(qpFile,** "%s", cBuffer);

if **(iRet != EOF) {**

**printf(**"Read in: %s\n", cBuffer);

**}**

**fclose(qpFile);**

**getchar();**

return **0;**

}



### fseek()

int **fseek(FILE\* qpStream,** long **lOffset,** int **iStart);**

Эта функция устанавливает положение внутреннего указателя файла для потока «qpStream». Указатель файла устанавливается в положение «iStart» плюс значение «lOffset». Значение «iStart» должно быть другим следующим постоянным значением: «SEEK\_SET», «SEEK\_CUR» или «SEEK\_END». Это значение соответствует началу, текущему местоположению указателя и конечным позициям в файле. Если функция выполнена успешно, она возвращает 0. В противном случае возвращается ненулевое значение, указывающее на ошибку.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for reading.

**FILE\* qpFile = fopen(cpFileName,** "r" **);**

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open %s for reading \n", cpFileName);

return **1;**

**}**

// Set the pointer to position 2 in the file.

int **iRet = fseek(qpFile, 2, SEEK\_SET);**

if **(iRet == 0) {**

// Read in 'l' at two places from the beginning

int **iRead = fgetc(qpFile);**

**printf(**"Read in: %c\n", (char)iRead);

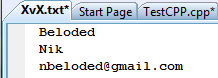
**}**

**fclose(qpFile);**

**getchar();**

return **0;**

}





### fsetpos()

int **fsetpos(FILE\* qpStream,** const **fpos\_t\* kqpPos);**

Эта функция устанавливает положение внутреннего указателя файла для потока «qpStream» в местоположение, указанное «kqpPos». Если функция успешна, она возвращает 0. В противном случае она возвращает ненулевое значение и устанавливает значение errno в соответствующее постоянное значение.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for reading.

**FILE\* qpFile = fopen(cpFileName,** "r" **);**

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open %s for reading \n", cpFileName);

return **1;**

**}**

**fpos\_t qPosition = 5;**

// Set the pointer to position 5 in the file.

int **iRet = fsetpos(qpFile, &qPosition);**

if **(iRet == 0) {**

// Read in 'e' at 5 places from the beginning

int **iRead = fgetc(qpFile);**

**printf(**"Read in: %c\n", (char)iRead);

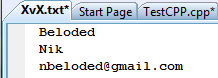
**}**

**fclose(qpFile);**

**getchar();**

return **0;**

}





### fstat()

int **fstat(**int **handle,** struct **stat \*statbuf)**

Эта функция не определена стандартом ANSI С. Функция fstat() заполняет структуру, на которую указывает statbuf, информацией о файле, связанном с дескриптором файла handle. Инфор­мация о содержимом stat может быть найдена в файле sys\stat.h.

При успешном заполнении структуры типа stat возвращается 0. При ошибке возвращается —1 и errno устанавливается в EBADF.

#include<stdio.h>

#include<sys\stat.h>

#include<stdlib.h>

#include<time.h>

//using namespace std;

//struct stat {

// dev\_t st\_dev; /\* ID of device containing file \*/

// ino\_t st\_ino; /\* inode number \*/

// mode\_t st\_mode; /\* protection \*/

// nlink\_t st\_nlink; /\* number of hard links \*/

// uid\_t st\_uid; /\* user ID of owner \*/

// gid\_t st\_gid; /\* group ID of owner \*/

// dev\_t st\_rdev; /\* device ID (if special file) \*/

// off\_t st\_size; /\* total size, in bytes \*/

// blksize\_t st\_blksize; /\* blocksize for file system I/O \*/

// blkcnt\_t st\_blocks; /\* number of 512B blocks allocated \*/

// time\_t st\_atime; /\* time of last access \*/

// time\_t st\_mtime; /\* time of last modification \*/

// time\_t st\_ctime; /\* time of last status change \*/

//};

int **main(**void)

{

**FILE \*fp;**

struct **stat buff;**

if **((fp=fopen(**"XvX.txt", "rb")) == NULL) {

**printf(**"Cannot open file.\n");

**exit(1);**

**}**

/\* заполнение структуры типа stat \*/

**fstat (fileno (fp), &buff);**

**printf(**"Size of the file: %ld\n", buff.st\_size);

**printf(**"I-node number: %ld\n", (long) buff.st\_ino);

**printf(**"Mode: %lo (octal)\n", (unsignedlong) buff.st\_mode);

**printf(**"Link count: %ld\n", (long) buff.st\_nlink);

**printf(**"Ownership: UID=%ld GID=%ld\n", (long) buff.st\_uid, (long) buff.st\_gid);

//printf("Preferred I/O block size: %ld bytes\n", (long) buff.st\_blksize);

//printf("File size: %lld bytes\n", (long long) buff.st\_size);

//printf("Blocks allocated: %lld\n", (long long) buff.st\_blocks);

**printf(**"Last status change: %s", ctime(&buff.st\_ctime));

**printf(**"Last file access: %s", ctime(&buff.st\_atime));

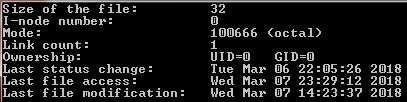
**printf(**"Last file modification: %s", ctime(&buff.st\_mtime));

**fclose(fp);**

**getchar();**

return **0;**

}



### ftell()

long **ftell(FILE\* qpStream);**

Эта функция возвращает позицию внутреннего указателя файла для потока «qpStream». Если функция не удалась, она возвращает EOF.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for reading.

**FILE\* qpFile = fopen(cpFileName,** "r" **);**

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open %s for reading \n", cpFileName);

return **1;**

**}**

// Set the file pointer to two from the end

int **iRet = fseek(qpFile, 2, SEEK\_END);**

// Find the position of the pointer in the file.

long **lPos = ftell(qpFile);**

if **(lPos != EOF) {**

**printf(**"File pointer is at %i\n", lPos);

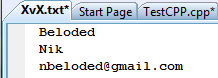
**}**

**fclose(qpFile);**

**getchar();**

return **0;**

}





### fwrite()

size\_t fwrite(constvoid\* kvpBuf, size\_t qSize, size\_t qCount, FILE\* qpStream);

Эта функция записывает записи «qCount» размером «qSize» байтов из буфера «kvpBuf» в поток, указанный «qpStream». Функция возвращает количество записей, фактически записанных в поток, которые могут быть меньше, чем «qCount», если возникает ошибка.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for appending.

**FILE\* qpFile = fopen(cpFileName,** "a" **);**

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open the file %s\n", cpFileName);

return **1;**

**}**

char **cpBuffer[] =** "\nC++ Reference";

**size\_t qBytes = 0;**

// Write the entire buffer without the null terminator.

**qBytes = fwrite(cpBuffer, 1,** sizeof(cpBuffer) - 1, qpFile);

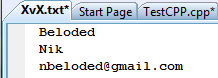
**printf(**"Bytes Written = %i\n", qBytes);

**fclose(qpFile);**

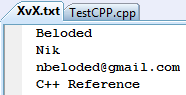
**getchar();**

return **0;**

}







### getc()

size\_t getc(constvoid\* kvpBuf, size\_t qSize, size\_t qCount, FILE\* qpStream);

Эта функция возвращает символ из потока «qpStream» в текущем местоположении указателя файла и продвигает указатель файла. Если функция не удалась, она возвращает EOF.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for reading.

**FILE\* qpFile = fopen(cpFileName,** "r" **);**

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open %s for reading \n", cpFileName);

return **1;**

**}**

// Get the first character in the file.

int **iRet = getc(qpFile);**

**printf(**"Read in: %c\n", (char)iRet);

**fclose(qpFile);**

**getchar();**

return **0;**

}



### getchar()

int **getchar();**

Эта функция считывает символ из стандартного потока ввода, «stdin», возвращает символ чтения и продвигает указатель потока. Если функция не удалась, она возвращает EOF.

#include<cstdio>

#include<cstdarg>

int **main()**

{

// Read the first typed in character.

int **iRet = getchar();**

**printf(**"You typed: %c\n", (char)iRet);

**getchar();**

return **0;**

}



### gets()

char\* gets(char\* cpBuffer);

Эта функция считывает строку текста в буфер «cpBuffer» из стандартного входного потока «stdin». Важно убедиться, что линия не слишком велика для буфера. В противном случае произойдет переполнение буфера. Строка текста включает в себя все символы до и включая символ «\» конечной линии, который помещается в поток как символ «null».

#include<cstdio>

#include<cstdarg>

int **main()**

{

char **caBuffer[50];**

// Get the first character in the file.

**gets(caBuffer);**

**printf(**"You typed: %s\n", caBuffer);

**getchar();**

return **0;**

}



### getw()

int **int\_variable=getw(FILE \*fp);**

Функция считывает целочисленные данные из файла

#include<stdio.h>

#include<sys\stat.h>

#include<stdlib.h>

#include<time.h>

#include<conio.h>

//using namespace std;

int **main()**

{

int **data;**

**FILE \*fp;**

**printf (**"Opening file in write mode to write integer data:");

**fp=fopen (**"testfile.txt","w"); //opening file in write mode

if(fp == NULL){

**printf (**"\ncan't open file");

**exit(0);**

**}**

else{

for(int **i=0; i<10; i++){**

**putw(i, fp);**

**}**

**}**

**fclose(fp);**

**printf(**"\nInteger data from file:\n");

**fp=fopen (**"testfile.txt","r"); //opening file in read mode

while((data=getw(fp))!=EOF) //read integer data

**{**

**printf(**"%d\n", data);

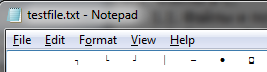
**}**

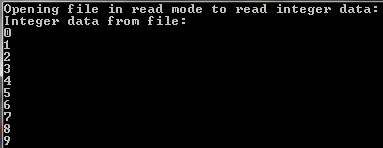
**fclose(fp);**

**getchar();**

return **0;**

}





### perror()

void **perror(**constchar\* kcpString);

Эта функция выводит строку «kcpString» в поток «stderr» для сообщения об ошибках. Результатом является строка «kcpString», за которой следует двоеточие, а затем последнее системное сообщение об ошибке. Последняя ошибка сохраняется в переменной errno.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.net";

// Try to open a file that doesn't exist

**FILE\* qpFile = fopen(cpFileName,** "r" **);**

// Check that the file was opened successfully.

if **(!qpFile) {**

**perror(**"Could not open the file");

**getchar();**

return **1;**

**}**

**fclose(qpFile);**

**getchar();**

return **0;**

}



### printf()

int **printf(**constchar\* kcpFormatString, ...);

Эта функция выводит форматированную строку «kcpFormatString» в стандартный поток вывода «stdout». В случае успеха функция возвращает количество выводимых символов. В противном случае функция возвращает отрицательное значение, указывающее на ошибку.

#include<cstdio>

#include<cstdarg>

int **main()**

{

char **caString[] =** "formatted";

// Write a formatted string to the stdout.

int **iRet = printf(**"Writing a %s string.\n", caString);

// Check whether or not some characters were written

if **(iRet > 0) {**

**printf(**"Characters written:%i\n", iRet);

**}** else **{**

**perror(**"printf() call failed");

**}**

**getchar();**

return **0;**

}



### putc()

int **putc(**int **iChar, FILE\* qpStream);**

Эта функция записывает символ ASCII «iChar» в поток «qpStream» в месте расположения текущего внутреннего указателя файла. В случае успеха функция возвращает значение написанного символа. В противном случае он возвращает EOF, чтобы указать условие ошибки или конечного fo-файла.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for appending.

**FILE\* qpFile = fopen(cpFileName,** "a" **);**

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open the file %s\n", cpFileName);

return **1;**

**}**

// Append a plus sign to the file

int **iRet = putc(**'+', qpFile);

// Check the put was successful

if **(iRet != EOF) {**

**printf(**"Character appended: %c\n", (char)iRet);

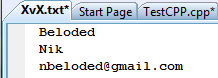
**}**

**fclose(qpFile);**

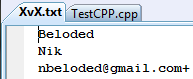
**getchar();**

return **0;**

}







### putchar()

int **putchar(**int **iChar);**

Эта функция записывает символ ASCII «iChar» в стандартный выходной поток «stdout». В случае успеха функция возвращает значение написанного символа. В противном случае он возвращает EOF, чтобы указать условие ошибки или конечного fo-файла.

#include<cstdio>

int **main()**

{

// Write an ASCII character to "stdout"

int **iRet = putchar(**'X');

// Check the put was successful

if **(iRet != EOF) {**

**printf(**"\nCharacter written: %c\n", (char)iRet);

**}**

**getchar();**

return **0;**

}



### puts()

int **puts(**constchar **\*kcpString);**

Эта функция выводит строку с нулевым завершением в стандартный выходной поток «stdout». Конечный нулевой символ '' заменяется конечной строкой '\' в выходном потоке. Функция возвращает неотрицательное значение в случае успеха и отрицательное значение, если вызов функции не выполняется.

#include<cstdio>

int **main()**

{

// Write to stdout.

int **iRet = puts(**"Write a string to stdout");

// Check whether or not some characters were written

if **(iRet == EOF) {**

**perror(**"puts() call failed");

**}**

**getchar();**

return **0;**

}



### putw()

int **putw(**int **i, FILE \*stream)**

Функция putw() записывает целое i в stream в текущую позицию внутри файла и дает соответ­ствующее приращение указателю положения в файле.

Функция putw() возвращает записанное значение. Возврат EOF означает ошибку в потоке, если последний открыт в текстовом режиме. Поскольку маркер EOF также является целым числом, то  
в случае двоичного потока надо использовать ferror(), чтобы выяснить, действительно ли про­изошла ошибка.

#include<stdio.h>

#include<sys\stat.h>

#include<stdlib.h>

#include<time.h>

#include<conio.h>

//using namespace std;

int **main()**

{

int **data;**

**FILE \*fp;**

**printf (**"Opening file in read mode to read integer data:");

**fp=fopen (**"testfile.txt","w"); //opening file in write mode

if(fp == NULL){

**printf (**"\ncan't open file");

**exit(0);**

**}**

else{

for(int **i=0; i<10; i++){**

**putw(i, fp);**

**}**

**}**

**fclose(fp);**

**printf(**"\nInteger data from file: \n");

**fp=fopen (**"testfile.txt","r"); //opening file in read mode

while((data=getw(fp))!=EOF) //read integer data

**{**

**printf(**"%d\n", data);

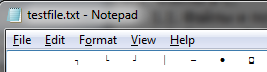
**}**

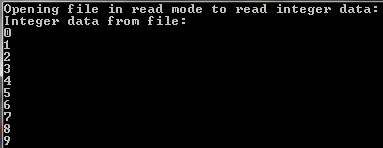
**fclose(fp);**

**getchar();**

return **0;**

}





### remove()

int **remove(**constchar\* kcpFilePath);

Эта функция удаляет файл, указанный в пути к файлу, заданном строкой «kcpFilePath». В случае успеха функция удаляет файл и возвращает 0. В противном случае функция возвращает -1 и устанавливает значение errno для указания ошибки.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

// Delete the local file

int **iRet = remove(**"XvX2.txt");

// Check whether or not some characters were written

if **(iRet == -1) {**

**perror(**"Failed to remove file XvX2.txt");

**}** else **{**

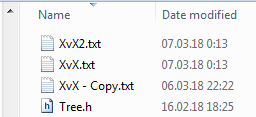
**printf(**"The file XvX2.txt was deleted.\n");

**}**

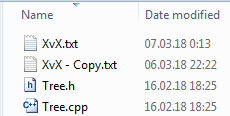
**getchar();**

return **0;**

}







### rename()

int **rename(**constchar\* kcpOldName, constchar\* kcpNewName);

Эта функция переименовывает файл или каталог, указанные строкой с нулевым завершением «kcpOldName», в имя, указанное «kcpNewName». Функция возвращает 0 в случае успеха. В противном случае функция возвращает отрицательное значение и устанавливает значение errno для указания ошибки.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

// Delete the local file

int **iRet = rename(**"XvX2.txt", "XvX2\_dotnet.txt");

// Check whether or not some characters were written

if **(iRet != 0) {**

**perror(**"Failed to rename file XvX2.txt");

**}** else **{**

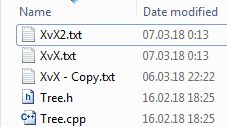
**printf(**"The file XvX2.txt was renamed.\n");

**}**

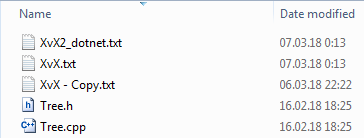
**getchar();**

return **0;**

}







### rewind()

void **rewind(FILE\* qpStream);**

Эта функция позиционирует внутренний указатель файла для потока «upStream» в начале файла. Функция очищает все индикаторы ошибок и конца файла для потока.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for reading.

**FILE\* qpFile = fopen(cpFileName,** "r");

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open the file %s\n", cpFileName);

return **1;**

**}**

// Set the file pointer position to 4

**fseek(qpFile, 4, SEEK\_SET);**

long **lPos = ftell(qpFile);**

**printf(**"File pointer is at %d\n", lPos);

// Set the file pointer position back to 0

**rewind(qpFile);**

**lPos = ftell(qpFile);**

**printf(**"File pointer is at %d\n", lPos);

**fclose(qpFile);**

**getchar();**

return **0;**

}



### scanf()

int **scanf(**constchar\* kcpFormatString, ...);

Эта функция считывает форматированные данные формата, заданного форматированной строкой «kcpFormatString» из стандартного входного потока «stdin». Функция возвращает поля, успешно прочитанные и назначенные.

#include<cstdio>

int **main()**

{

int **iInteger = 0;**

// Read a value into the integer

int **iRet = scanf(**"%i", &iInteger);

**printf(**"You typed: %i\n", iInteger);

**getchar();**

return **0;**

}



### setbuf()

void **setbuf(FILE\* qpStream,** char\* cpBuffer);

Эта функция устанавливает буфер для потока «qpStream» в переданный в буфере «cpBuffer». Функция должна ссылаться на поток, который был открыт, но это не было прочитано или записано. Кроме того, буферный указатель «cpBuffer» должен указывать на буфер длиной байтов BUFSIZ. Обратите внимание, что эта функция устарела в пользу setvbuf.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for reading.

**FILE\* qpFile = fopen(cpFileName,** "r");

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open the file %s\n", cpFileName);

return **1;**

**}**

// The buffer size must be BUFSIZ

char **caBuffer[BUFSIZ];**

// Set the buffer for the file stream

**setbuf(qpFile, caBuffer);**

// Read from the file.

char **caRead[50];**

**fscanf(qpFile,** "%s", caRead);

**printf(**"String read: %s\n", caRead);

**fclose(qpFile);**

**getchar();**

return **0;**

}



### setvbuf()

int **setvbuf(FILE\* qpStream,** char\* cpBuffer, int **iMode, size\_t qSize);**

Эта функция устанавливает буфер для потока «qpStream» в переданный в буфере «cpBuffer». Аргумент «qSize» определяет размер буфера в диапазоне [2, INT\_MAXM] в байтах, округленных до ближайшего четного числа. Параметр режима «iMode» должен быть одним из следующих значений: \_IOFBF, \_IOLBF или \_IONBF, которые обозначают полный буфер, буферизацию строки и отсутствие буферизации. В режимах полной буферизации и линейного буфера используется буфер с заданным размером буфера, и буферизация не используется, чтобы указать, что буфер не используется, независимо от того, что передается. Функция возвращает 0 для указания успеха и -1 для указания ошибки.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for reading.

**FILE\* qpFile = fopen(cpFileName,** "r");

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open the file %s\n", cpFileName);

return **1;**

**}**

// The can be whatever size we like

const **size\_t kqBufferSize = 100;**

char **caBuffer[kqBufferSize];**

// Set the buffer for the file stream

**setvbuf(qpFile, caBuffer, \_IOFBF, kqBufferSize);**

// Read from the file.

char **caRead[50];**

**fscanf(qpFile,** "%s", caRead);

**printf(**"String read: %s\n", caRead);

**fclose(qpFile);**

**getchar();**

return **0;**

}



### sprintf()

int **sprintf(**char\* cpBuffer, constchar\* kcpFormatString, ...);

Эта функция записывает форматированную строку с нулевым завершением, определенную «kcpFormatString», и ее аргументы в буфер «cpBuffer». В случае успеха функция возвращает количество байтов, помещенных в буфер, исключая нулевой ограничитель. В противном случае функция возвращает -1, чтобы указать ошибку.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char **caBuffer[100];**

// Fill the buffer with a formatted string.

**sprintf(caBuffer,** "XvX%s\n", ".net");

// Output the buffer's contents

**printf(**"Buffer holds: %s\n", caBuffer);

**getchar();**

return **0;**

}



### sscanf()

int **sscanf(**constchar\* kcpBuffer, constchar\* kcpFormatString, ...);

Эта функция считывает и присваивает форматированные данные из строки «kcpBuffer». Функция возвращает количество успешно присвоенных полей. Если поля не были прочитаны или произошла ошибка, функция возвращает EOF.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

char **caBuffer[] =** "Copyright 2018 XvX";

// Read from the buffer

char **caCopyright[20];**

int **iYear = 0;**

char **caXoaX[20];**

**sscanf(caBuffer,** "%s %i %s", caCopyright, &iYear, caXoaX);

// Output the read contents

**printf(**"Read: %s %i %s\n", caCopyright, iYear, caXoaX);

**getchar();**

return **0;**

}



### tmpfile()

FILE\* tmpfile();

Эта функция создает и открывает временный файл для чтения и записи в двоичном режиме. Функция возвращает указатель на открытый файл, если он успешный, и NULL, если возникает ошибка. Файл автоматически удаляется, когда он закрыт или программа завершается

#include<cstdio>

usingnamespace **std;**

int **main()**

{

// Open a temp file in binary mode.

**FILE\* qpFile = tmpfile();**

if **(!qpFile) {**

**perror(**"Could not open temp file:");

return **1;**

**}**

char **caWrite[] =** "XvX.net";

char **caRead[20];**

int **iSize =** sizeof(caWrite);

// A binary write and read

**fwrite(caWrite, 1, iSize, qpFile);**

**rewind(qpFile);**

**fread(caRead, 1, iSize, qpFile);**

// Add null-terminator before we output

**caRead[iSize] =** '/0';

// Output the read contents

**printf(**"Read: %s\n", caRead);

**fclose(qpFile);**

**getchar();**

return **0;**

}



### tmpnam()

char\* tmpnam(char\* cpString);

Эта функция создает имя (файла, которого не существует в текущем каталоге) в строке «cpString», которая может использоваться для создания временных файлов. Функция возвращает указатель на строку, содержащую имя файла, если вызов был успешным и NULL в противном случае.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

// Get unique file name, using NULL for the argument

char\* cpTempName = tmpnam(0);

if **(cpTempName) {**

**printf(**"Use %s as a temp file name.\n", cpTempName);

**}** else **{**

**printf(**"No temp file name was found.\n");

return **1;**

**}**

**getchar();**

return **0;**

}



### ungetc()

int **ungetc(**int **iChar, FILE\* qpStream);**

Эта функция помещает символ «iChar» в поток «qpStream» в текущее местоположение указателя внутреннего файла после того, как символ был прочитан. В случае успеха функция возвращает символ, помещенный в поток, или возвращает EOF, если возникает ошибка.

#include<cstdio>

usingnamespace **std;**

int **main()**

{

for **(**int **i = 0; i < 8; ++i) {**

int **iChar = getchar();**

// Put the second 'X' back so that

// we get it next time.

if **(i == 3) {**

**ungetc(iChar, stdin);**

**}**

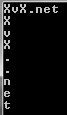
**printf(**"%c\n", (char)iChar);

**}**

**getchar();**

return **0;**

}



### vfprintf()

int **vfprintf(FILE\* qpStream,** constchar\* kcpFormatString, va\_list qVarArgList);

Эта функция записывает форматированную строку «kcpFormatString» в поток, указанный «qpStream», используя аргументы в списке аргументов «qVarArgList». Функция возвращает количество символов, которые были записаны в поток, или возвращает отрицательное число, чтобы указать, что произошла ошибка.

#include<cstdio>

#include<cstdarg>

usingnamespace **std;**

int **main()**

{

char\* cpFileName = "XvX.txt";

// Open an ASCII text file for appending.

**FILE\* qpFile = fopen(cpFileName,** "a");

// Check that the file was opened successfully.

if **(!qpFile) {**

**printf(**"Could not open the file %s\n", cpFileName);

return **1;**

**}**

// An argument list - the first entry is just a count.

int **iaArguments[] = {2, 10, 20};**

**va\_list qVarArg = 0;**

**va\_start(qVarArg, iaArguments[0]);**

// Use a var arg list to output two integers

**vfprintf(qpFile,** "\n%d %d", qVarArg);

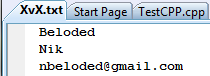
**va\_end(qVarArg);**

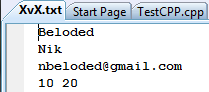
**fclose(qpFile);**

**getchar();**

return **0;**

}





### vfscanf()

int vfscanf(FILE \*fp, const char \*format, va\_list arg);

Функция vfscanf() сканирует входные данные из файла, обозначенного fp, под управлением формата аргумента. Функция vfscanf () является «varargs» версией fscanf (). Возвращает: количество входных аргументов, для которых значения были успешно сканированы и сохранены, или EOF, когда сканирование остановлено, дойдя до конца входного потока перед сохранением любых значений.

fp поток, который вы хотите прочитать.

format строка, определяющая формат ввода. Для получения дополнительной информации смотри scanf(). Строка форматирования определяет, какие дополнительные аргументы вам необходимо предоставить.

arg список дополнительных аргументов с переменными аргументами, которые вы должны инициализировать с помощью макроса va\_start ().

#include <stdio.h>

#include <stdarg.h>

FILE \*stream;

int call\_vfscanf(FILE \* istream, char \* format, ...)

{

int result;

va\_list arglist;

va\_start(arglist, format);

result = vfscanf(istream, format, arglist);

va\_end(arglist);

return result;

}

int main(void)

{

long l;

float fp;

char s[81];

char c;

if (fopen\_s(&stream, "vfscanf.txt", "w+") != 0)

{

printf("The file vfscanf.out was not opened\n");

}

else

{

fprintf(stream, "%s %ld %f%c", "a-string",

65000, 3.14159, 'x');

// Security caution!

// Beware loading data from a file without confirming its size,

// as it may lead to a buffer overrun situation.

// Set pointer to beginning of file:

fseek(stream, 0L, SEEK\_SET);

// Read data back from file:

call\_vfscanf(stream, "%s %ld %f%c", s, &l, &fp, &c);

// Output data read:

printf("%s\n", s);

printf("%ld\n", l);

printf("%f\n", fp);

printf("%c\n", c);

fclose(stream);

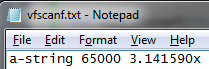
}

getchar();

return 0;

}





### vprintf()

int **vprintf(**constchar\* kcpFormatString, va\_list qVarArgList);

Эта функция записывает форматированную строку «kcpFormatString» в стандартный поток вывода «stdout», используя аргументы в списке аргументов «qVarArgList». Функция возвращает количество символов, которые были записаны в поток, или возвращает отрицательное число, чтобы указать, что произошла ошибка.

#include<cstdio>

#include<cstdarg>

int **main()**

{

// An argument list - the first entry is just a count.

int **iaArguments[] = {2, 10, 20};**

**va\_list qVarArg = 0;**

**va\_start(qVarArg, iaArguments[0]);**

// Use a var arg list to output two integers

**vprintf(**"%d %d\n", qVarArg);

**va\_end(qVarArg);**

**getchar();**

return **0;**

}



### vsprintf()

int **vsprintf(**char\* cpBuffer, constchar\* kcpFormatString, va\_list qVarArgList);

Эта функция записывает форматированную строку с нулевым завершением, определенную «kcpFormatString», и ее аргументы «qVarArgList» в буфер «cpBuffer». В случае успеха функция возвращает количество символов, которые были написаны, не включая завершающий NULL. В противном случае функция возвращает отрицательное значение, чтобы указать, что произошла ошибка.

#include<cstdio>

#include<cstdarg>

void **PutArgsInBuffer(**char\* cpBuffer, char\* cpFormat, ...)

{

**va\_list qVarArg = 0;**

**va\_start(qVarArg, cpFormat);**

// Put the arguments into the buffer

**vsprintf(cpBuffer, cpFormat, qVarArg);**

**va\_end(qVarArg);**

}

int **main()**

{

char **caBuffer[100];**

**PutArgsInBuffer(caBuffer,** "%s %d", "XvX.net", 2018);

**printf(**"Buffer Holds: %s\n", caBuffer);

**getchar();**

return **0;**

}



### vsscanf()

int vsscanf(const char \* s, const char \* format, va\_list arg);

s c строка, которую функция обрабатывает как источник для извлечения данных.

format c строка, которая содержит строку формата, которая соответствует тем же спецификациям, что и формат в scanf (подробности см. scanf).

arg значение, определяющее список переменных аргументов, инициализированных с помощью va\_start. va\_list - это особый тип, определенный в <cstdarg>.

Функции vscanf(), vfscanf() и vsscanf() должны быть эквивалентны функциям scanf(), fscanf() и sscanf(), за исключением того, что вместо вызова с переменным числом аргументов они вызываемый с помощью списка аргументов, определенного в заголовке <stdarg.h>. Эти функции не должны вызывать макрос va\_end. Поскольку эти функции вызывают макрос va\_arg, значение ap после возврата не указывается.

#include <stdio.h>

#include <stdarg.h>

char buffer[80] = "6 7.0 forty-two";

int my\_vfscanf(char \*fmt, ...)

{

va\_list argptr;

int cnt;

fflush(stdin);

va\_start(argptr, fmt);

cnt = vsscanf(buffer, fmt, argptr);

va\_end(argptr);

return(cnt);

}

int main(void)

{

int inumber;

float fnumber;

char string[80];

my\_vfscanf("%d %f %s", &inumber, &fnumber, string);

printf("%d \* %f = %s\n", inumber, fnumber, string);

getchar();

return 0;

}



## Низкоуровневый ввод и вывод

### close()

int **close(**int **fildes);**

close - системный вызов, который используется для закрытия дескриптора открытого файла.

#include<fcntl.h>

#include<cstdio>

#include<io.h>

#include<stdio.h>

#include<stdlib.h>

#include<errno.h>

usingnamespace **std;**

externint **errno;**

int **main()**

{

**size\_t filedesc = open(**"testfile.txt", O\_WRONLY | O\_CREAT);

if(filedesc < 0){

**printf(**"Error Number % d\n", errno);

// print program detail "Success or failure"

**perror(**"Program");

**getchar();**

return **1;**

**}**

else{

**printf(**"File testfile.txt OK\n");

**}**

if(close(filedesc) < 0){

// print program detail "Success or failure"

**perror(**"Program");

**getchar();**

return **1;**

**}**

else{

**printf(**"File testfile.txt closed\n");

**}**

**getchar();**

return **0;**

}



### lseek()

long **lseek(**int **handle,** long **offset,** int **origin);**

Функция lseek() устанавливает указатель положения в файле, указанном дескриптором handle, в положение, указанное аргументами offset и origin.

Действия lseek() зависят от значений аргументов origin и offset. Аргумент origin может принимать значения 0, 1 или 2:

0 Смещение отсчитывается от начала файла

1 Смещение отсчитывается от текущей позиции

2 Смещение отсчитывается от конца файла

В файле io.h определены перечисленные ниже макросы. Они могут использоваться в качестве зна­чения origin в порядке от 0 до 2

SEEK\_SET

SEEK\_CUR

SEEK\_END

В случае успеха функция lseek() возвращает offset, т.е. целое число типа long. При неудаче возвращается — 1L и errno устанавливается в одно из следующих значений:

EBADF Недействительный номер файла

EINVAL Недействительный аргумент

#include<fcntl.h>

#include<cstdio>

#include<io.h>

#include<stdio.h>

#include<stdlib.h>

usingnamespace **std;**

int **main(){**

int **file = open(**"XvX.txt", O\_RDWR); //open file with contents

**lseek(file, 0, 0);** //positions at first char at beginnging of file

char **buffer;**

**read(file, &buffer,** sizeof(buffer));

**printf(**"%c\n", buffer); //text file

if **(buffer ==** 'B'){

char **copy =** '?';

**lseek(file, 0, 0);** //positions at first char at beginnging of file

**write(file, &copy, 1);** //text file containing 'B' puts '?' were 'B' is.

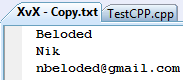
**}**

**close(file);**

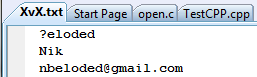
**getchar();**

return **0;**

}







### open()

int **open (**constchar\* Path, int **flags [,** int **mode ]);**

Используется для открытия файла для чтения, записи или и того, и другого.

Path путь к файлу, который вы хотите использовать.

* используйте абсолютный путь, начинающийся с «/», если вы не работаете в том же каталоге файла.
* используйте относительный путь, который является только именем файла с расширением, когда вы работаете в том же каталоге файла.

Flags на Ваше усмотрение, как вы любите использовать

O\_RDONLY только чтение,

O\_WRONLY только запись,

O\_RDWR чтение и запись,

O\_CREAT создание файла, если он не существует,

O\_EXCL предотвратить создание, если оно уже существует

#include<fcntl.h>

#include<cstdio>

#include<io.h>

#include<stdio.h>

#include<stdlib.h>

#include<errno.h>

usingnamespace **std;**

externint **errno;**

int **main()**

{

// if file does not have in directory

// then file XvX.tmp is created.

int **fd = open(**"XvX.tmp", O\_RDONLY | O\_CREAT);

if **(fd ==-1){**

// print which type of error have in a code

**printf(**"Error Number % d\n", errno);

// print program detail "Success or failure"

**perror(**"Program");

**}**

else{

**printf(**"fd = %d\n", fd);

**}**

**getchar();**

return **0;**

}



### read()

size\_t read (int **fd,** void\* buf, size\_t cnt);

Чтение данных из одного буфера в дескриптор файла, чтение size байт из файла, указанного fd, в память.

#include<fcntl.h>

#include<cstdio>

#include<io.h>

#include<stdio.h>

#include<stdlib.h>

usingnamespace **std;**

int **main()**

{

int **fd, sz;**

char **\*buf = (**char **\*) calloc(100,** sizeof(char));

**fd = open(**"XvX.txt", O\_RDONLY);

if **(fd < 0) { perror(**"r1"); exit(1); }

**sz = read(fd, buf, 10);**

**printf(**"called read(% d, c, 10). returned that"

" %d bytes were read.\n", fd, sz);

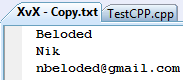
**buf[sz] =** '\0';

**printf(**"Those bytes are as follows: % s\n", buf);

**getchar();**

return **0;**

}





### unlink() (\_unlink())

int **unlink(**constchar **\*fname)**

int **\_unlink(**constchar **\*filename);**

Функция unlink() удаляет указанный файл из каталога. В случае успеха функция возвращает 0, а при неудаче —1, причем errno устанавливается в одно из следующих значений:

ENOENT Недействительный путь или имя файла

EACCESS Доступ запрещен

#include<fcntl.h>

#include<cstdio>

#include<io.h>

#include<stdio.h>

#include<stdlib.h>

usingnamespace **std;**

int **main()**

{

if( \_unlink( "XvX2.tmp" **) == -1 )**

**perror(** "Could not delete 'XvX2.tmp'" **);**

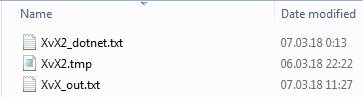
else

**printf(** "Deleted 'XvX2.tmp'\n" **);**

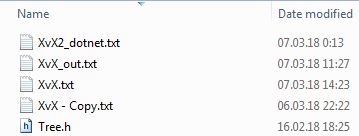
**getchar();**

return **0;**

}









### write()

ssize\_t write(int **fd,** constvoid **\*buf, size\_t count);**

write () записывает до подсчета байтов из буфера, обозначенного buf, в файл, на который ссылается файловый дескриптор fd.

Количество записанных байтов может быть меньше, чем count, если, например, недостаточно места на базовом физическом носителе или встречается ограничение ресурса RLIMIT\_FSIZE, или вызов был прерван обработчиком сигнала после написания меньше, чем количество байт.

Для файла с возможностью поиска (то есть, к которому может применяться lseek, например, обычный файл), запись происходит при текущем смещении файла, а смещение файла увеличивается на количество фактически записанных байтов. Если файл был открыт с помощью O\_APPEND, смещение файла сначала устанавливается в конец файла перед записью. Коррекция смещения файла и операция записи выполняются как атомный шаг.

#include<fcntl.h>

#include<cstdio>

#include<io.h>

usingnamespace **std;**

int **main(**void)

{

int **filedesc = open(**"testfile.txt", O\_WRONLY | O\_APPEND);

if **(filedesc < 0) {**

return **-1;**

**}**

if **(write(filedesc,** "This will be output to testfile.txt\n", 36) != 36) {

**write(2,** "There was an error writing to testfile.txt\n", 43);

return **-1;**

**}**

**getchar();**

return **0;**

}

