Файлы

Потоки

Основой понимания системы ввода/вывода Си являются концепции *потоков* и *файлов*.

Поток данных — абстракция, используемая для чтения или записи файлов в единой манере.

Поддержка потоков включена в большинство языков программирования (C++, C#, Java).

В Си поток представляет собой источник ввода и/или вывода данных (обычно байтов), связанный с файлом, устройством, либо другим процессом.

Потоки не зависят от устройств, поэтому одни и те же функции могут использоваться для записи информацию в файл на диске и на другие устройства.

Существует два типа потоков: текстовые и двоичные.

Текстовые потоки - это последовательность символов.

В текстовых потоках некоторые символы могут преобразовываться согласно требованиям среды. Поэтому может не быть однозначного соответствия между записываемыми или считываемыми символами и символами во внешнем устройстве.

Например, символ новой строки может преобразовываться в пару «возврат каретки - перевод строки».

Двоичный поток - это последовательность байт, имеющих однозначное соответствие с байтами во внешнем устройстве (преобразование символов не возникает).

Число байт, записанных или прочитанных из внешнего устройства, совпадает с числом во внешнем устройстве.

Может добавляться некоторое количество нулевых байт к двоичному потоку.

Файлы

В Си файлы - это логическая концепция, применимая ко всему, начиная от дисковых файлов оканчивая терминалами.

Если файл открыт, может осуществляться обмен между файлом и программой.

Не все файлы имеют одинаковые возможности. Т.е. в системе ввода/вывода Си - все потоки одинаковы, а файлы нет.

Основные определения

Файл – именованный набор байтов, который может быть сохранен на некотором накопителе. Файл, как и массив, - это совокупность данных, в этом они похожи друг на друга.

Файлы, в отличие от массивов, располагаются не в оперативной памяти,а на жестких дисках или на внешних носителях.

Файл имеет своё уникальное имя, например файл.txt.

В одной директории не могут находиться файлы с одинаковыми именами.

Под именем файла понимается не только его название, но и расширение, например: file.txt и file.dat - разные файлы, хоть и имеют одинаковые названия.

Полное имя файлов – это полный адрес к директории файла с указанием имени файла, например: D:\docs\file.txt.

Файл не имеет фиксированной длины, т.е. может увеличиваться и уменьшаться.

Перед работой с файлом его необходимо открыть, а после работы -закрыть.

Файловая система - это совокупность файлов и управляющей информации на диске для доступа к файлам.

Текстовые файлы могут быть просмотрены и отредактированы с клавиатуры любым текстовым редакторов и имеют простую структуру: последовательность ASCII-символов. Эта последовательность символов разбивается на строки, каждая строка заканчивается двумя кодами: 13, 10 (0xD, 0xA).

Бинарные файлы – это файлы, которые не имеют структуры текстовых файлов. Каждая программа для своих бинарных файлов определяет собственную структуру.

Открытие файла

Большинство функций для работы с файлами находятся в библиотеках stdio.h и io.h.

Поток связывается с конкретным файлом с помощью операции открытия.

Для открытия доступа к файлу необходимо воспользоваться конструкцией:

FILE* fopen(char * ID_файла, char *режим);

ID_файла и путь к нему, задается строкой:

“c:\\work\\file.txt”

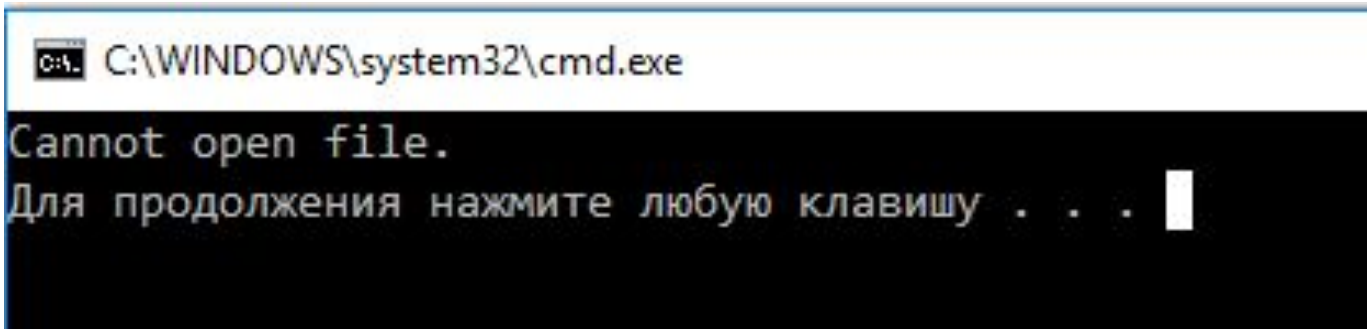
При успешном открытии происходит возвращение указателя на FILE.

Эта структура связана с физическим файлом и содержит всю необходимую информацию для работы с ним (указатель на текущую позицию в файле, тип доступа и др.)

Если при открытии файла произошла ошибка, то возвращается NULL.

Ошибка открытия файла

```
. . .
FILE *fp;

if ((fp = fopen("d:\@Work\test11.txt",“r")) == NULL) {
printf("Cannot open file.\n");exit(1);
}
. . .
```



Режим доступа к файлу

w – запись в текстовом режиме; (если файла с заданным именем нет, то он будет создан, если такой файл существует, то перед открытием прежняя информация уничтожается);

r – файл открывается только для чтения в текстовом режиме; если такого файла нет, то возникает ошибка;

a – файл открывается в текстовом режиме для добавления в его конец новой информации;

r+, *w+*, *a+* – файл открывается для редактирования данных, возможны и запись, и чтение информации;

t – файл открывается в текстовом режиме (используется по умолчанию);

b – файл открывается в бинарном режиме ;

rb, *wb*, *ab*, *r+b*, *w+b*, *a+b* – файл открывается в двоичном режиме.

При открытии файла в режимах *r*, *rb*, *r+*, *r+b* индикатор позиции устанавливается на начало файла, а в случае, если файл не существует – неудача;

При открытии файла в режимах *a*, *ab*, *a+*, *a+b* создается новый файл; если файл уже существует индикатор позиции устанавливается на конец файла;

При открытии файла в режимах *w*, *wb*, *w+*, *w+b* создается новый файл; если файл уже существует, то его содержимое стирается, а индикатор позиции устанавливается на начало файла.

При открытии файла лучше пользоваться конструкцией типа:

```
FILE *fp;

if ((fp = fopen("d:\\@Work\\test.txt","w")) == NULL) {
printf("Cannot open file.\n");exit(1);
}
```

При открытии файла этот метод контролирует возможность ошибок при открытии (проверяет наличие защиты от записи или отсутствие места на диске).

NULL используется потому, что указатели файлов никогда не принимают этого значения.

Закрытие файла

Связь потока с файлом уничтожается с помощью *операции закрытия*.

Закрытие потока вызывает принудительный сброс всего содержимого буфера во внешнее устройство. Данный процесс, как правило, называется очисткой буфера, и он гарантирует, что в буфере не останется информации.

Все файлы закрываются автоматически, когда программа нормальным образом завершает работу. Но, лучше самому закрыть файлы, используя fclose() в тот момент, когда файл уже не нужен, поскольку некоторые события могут помешать записи буфера на диск (например функция abort()).

Функция fclose() имеет прототип:

int fclose(FILE *fp);

При успешном завершении возвращает 0, а в случае неудачи – EOF (End Of File, конец файла):

Для закрытия нескольких файлов введена функция int fcloseall(void);

Пример

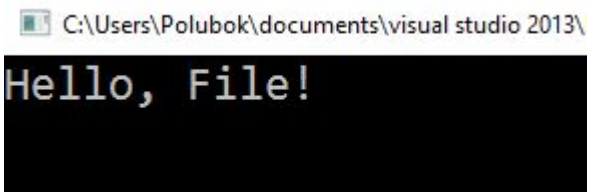
```
void main(){

int f;
char *str = new char[50];

FILE *fp;

if ((fp = fopen("d:\\@Work\\test.txt","w")) == NULL) {
printf("Cannot open file.\n");exit(1);
}

fflush(stdin);
gets(str);

fprintf(fp,str);

fclose(fp);

_getch();

}
```

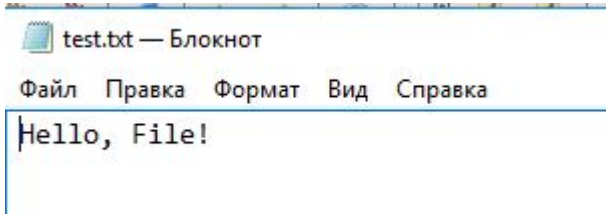
выделение памяти под строку, которую будем писать в файл

Открываем файл на запись

Функция записи строки в файл

Закрытие файла





Индикатор конца файла

Функция feof() проверяет, достигнут ли конец файла, связанного с потоком, через параметр filestream.

Возвращается значение, отличное от нуля, если конец файла был действительно достигнут.

Вызов данной функции выполняется после выполнения предыдущей операции с потоком, например операции считывания, которая постепенно двигает внутренний указатель файла в конец.

Дальнейшие операции с файлом, после достижения его конца не будут выполняться до тех пор, пока внутренний указатель не будет сдвинут назад, функциями fseek() или fsetpos().

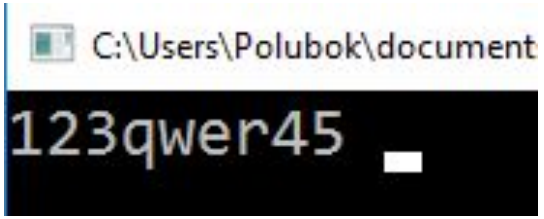
Состояние конца файла читается функцией

int feof(FILE *f);

Пример

```
. . .
void main(){

int f;
char *str = new char[50];

FILE *fp;

if ((fp = fopen("d:\\@Work\\test22.txt", "r")) == NULL) {
printf("Cannot open file.\n");exit(1);
}
fflush(stdin);

int i = 0;

if (fp == NULL) perror("Ошибка открытия файла");
else{
    while (!feof(fp)) {
        *(str+i)=fgetc(fp);
        printf("%c", *(str+i));
        i++;
    }
fclose(fp);
}
_getch();
}
```

Цикл чтения из файла посимвольно



Функции чтения / записи в файл

Все действия по записи/чтению данных из файла можно разделить на три группы:

- операции посимвольного ввода-вывода;
- операции построчного ввода-вывода;
- операции ввода-вывода по блокам.

Функции fgetc( ) и fputc( );

int fgetc(FILE *fp)

int fputc(int ch, FILE *fp)

Функция fgetc() возвращает следующий за текущей позицией символ из входного потока и дает приращение указателю положения в файле.

При достижении конца файла функция fgetc() возвращает EOF.

Так как EOF — действительное целое число, то при работе с двоичными файлами для обнаружения конца файла необходимо использовать feof().

Функция fputc() записывает символ ch в указанный поток в позицию, соответствующую текущему значению указателя положения в файле, а затем дает приращение указателю положения в файле.

Если ch был объявлен как int, он конвертируется функцией fputc() в unsigned char.

Функция fputc() возвращает значение записанного символа. В случае ошибки она возвращается EOF.

Пример 1

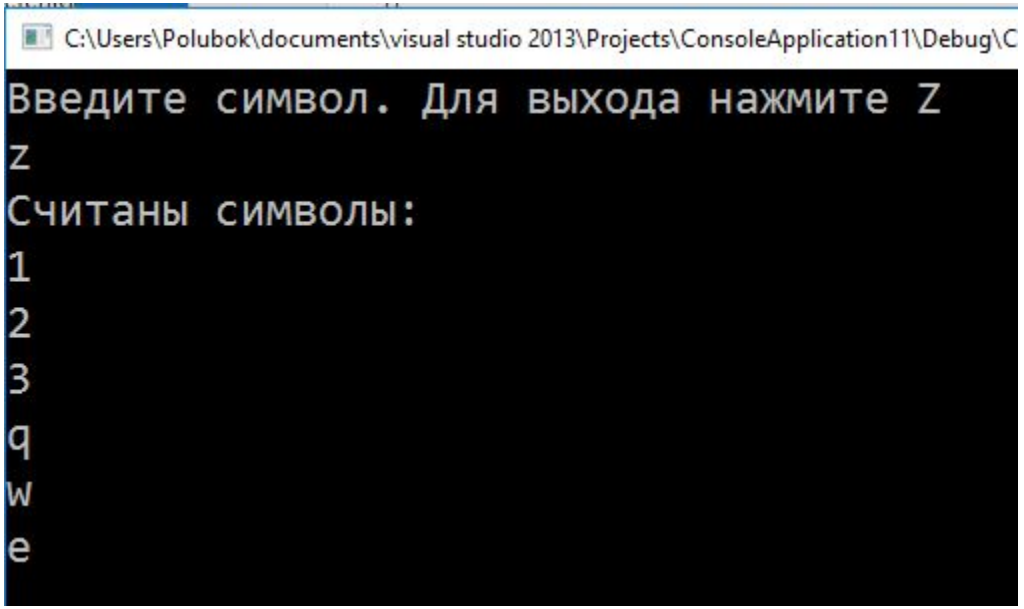
Посимвольный ввод/вывод

```
. . .
void main(){

setlocale(LC_CTYPE, "Rus");

char sym = '1', ch;
FILE *fp;

if ((fp = fopen("d:\\@Work\\test23.txt", "w")) == NULL) {
printf("Cannot open file.\n");exit(1);
}

while (sym !='Z' && sym != 'z') {
system("cls");
puts("Введите символ. Для выхода нажмите Z");
scanf("%c", &sym);
if (sym == 'z' || sym == 'Z');

else fputc(sym, fp);

}

fclose(fp);
```

Функция посимвольной записи в файл

```
if ((fp = fopen("d:\\@Work\\test23.txt", "r")) == NULL) {
printf("Cannot open file.\n");exit(1);
}
fflush(stdin);

puts("Считаны символы: ");

while ((ch= fgetc(fp) )!= EOF)

printf("%c", ch);

_getch();
}
```

Функция посимвольной чтения из файла



Функции fprintf() и fscanf()

Данные функции ведут себя так же, как и printf() и scanf(), за тем исключением, что работают с дисковыми файлами.

Прототипы функций:

int fprintf(FILE *fp, const char *форматная_строка, ...);

int fscanf(FILE *fp, const char * форматная_строка, ...);

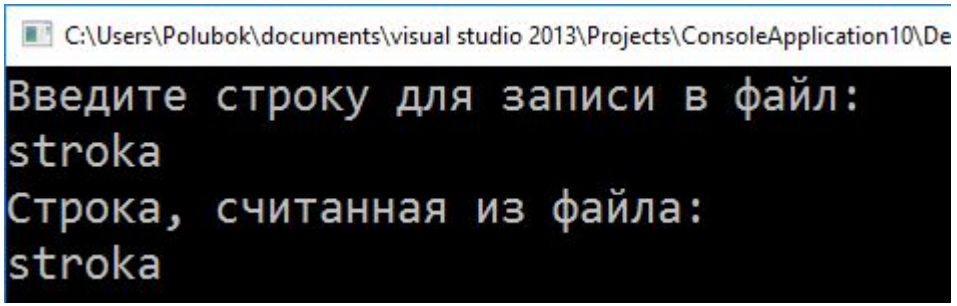
где fp - это указатель на файл.

Пример 2
Построчный ввод/вывод

```
. . .
void main(){

setlocale(LC_CTYPE, "Rus");

char *str = new char[50];
FILE *fp;

if ((fp = fopen("d:\\@Work\\test22.txt","r+")) == NULL) {
printf("Cannot open file.\n");exit(1); }

fflush(stdin);
puts("Введите строку для записи в файл: ");
gets(str);

fprintf(fp,str);

puts("Строка, считанная из файла: ");

fscanf(fp, str);

puts(str);

fclose(fp);

_getch();
}
```

Функция записи в файл

Функция чтения из файла



Пример 3
Построчный ввод/вывод

```
. . .
void main(){

setlocale(LC_CTYPE, "Rus");

int number;
char *str1 = new char[50];
char *str2 = new char[50];
FILE *fp;

if ((fp = fopen("d:\\@Work\\test22.txt", "w")) == NULL) {
printf("Cannot open file.\n");exit(1);
}

puts("Введите ФИО пользователя, город и номер телефона: ");

fscanf(stdin, "%s%s%d", str1, str2, &number);

fprintf(fp, "%s %s %d", str1, str2, number);

fclose(fp);
```

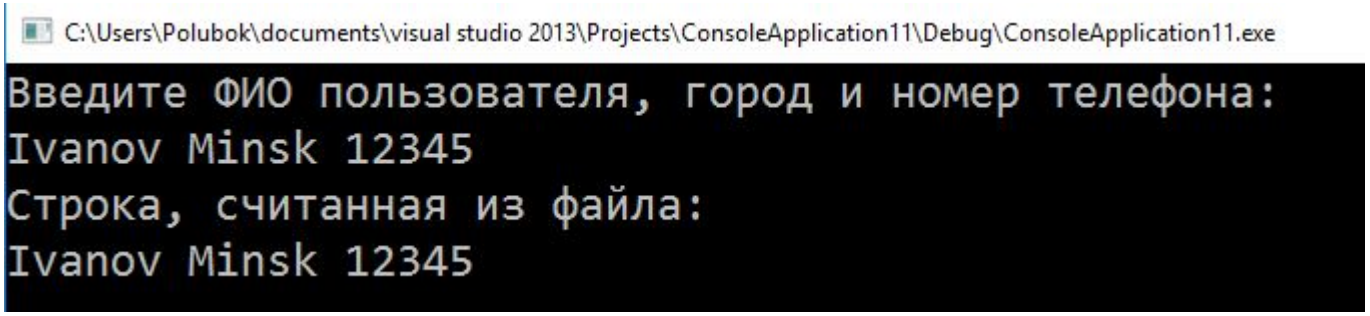
Считывание с клавиатуры

Запись считанного в файл с пробелами в качестве разделителя

```
if ((fp = fopen("d:\\@Work\\test22.txt", "r")) == NULL) {
printf("Cannot open file.\n");exit(1);
}
fflush(stdin);

puts("Строка, считанная из файла: ");

fscanf(fp, "%s %s %d", str1, str2, &number);

printf("%s %s %d",str1, str2, number);

fclose(fp);

_getch();

}
```

Чтение трех параметров, разделенных пробелами, и запись значений в соответствующие переменные

Вывод на печать



Функции fgets() и fputs()

Могут читать и писать строки в поток. Они имеют следующие прототипы:

int fputs(const char *str, FILE *fp);

char *fgets(char *str, int длина, FILE *fp);

Функция fputs() во многом подобна puts(), за тем исключением, что она записывает строку в указанный поток.

Функция fgets() читает строку из указанного потока, пока не встретится символ новой строки или не будет прочитано (длина - 1) символов.

Функция fgets() считывает символы до тех пор, пока не встретится символ «новая строка», EOF или до достижения указанного предела.

По окончании считывания в массив str сразу после последнего считанного символа помещается нулевой символ.

Пример 4
Построчный ввод/вывод

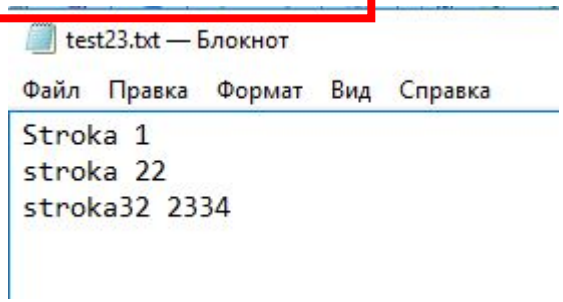
```
. . .
void main(){

setlocale(LC_CTYPE, "Rus");

char *str1 = new char[255];
char *str2 = new char[255];
char key;

FILE *fp;

if ((fp = fopen("d:\\@Work\\test23.txt", "w")) == NULL) {
printf("Cannot open file.\n");exit(1);
}

do {
system("cls");
puts("Введите строку: ");

fflush(stdin);

gets(str1);
str1 = strcat(str1, "\n");

fputs(str1, fp);

puts("Продолжим? y/n");
```

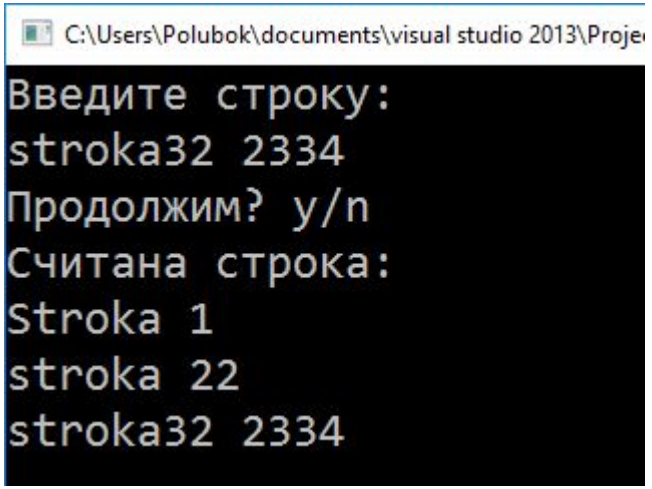
Добавляем в конце строки переход на новую сроку

Запись в файл

```
do {
key = _getch();
} while (key != 'n' && key != 'N'&&key != 'y' && key != 'Y');

}

while (key == 'y' || key == 'Y');

fclose(fp);

if ((fp = fopen("d:\\@Work\\test23.txt", "r")) == NULL) {
printf("Cannot open file.\n");exit(1);
}
fflush(stdin);

puts("Считана строка: ");

while (fgets(str2, 255, fp)!=NULL)
printf("%s", str2);

_getch();

}
```

Цикл чтения из файла по строкам





Функции fwrite() и fread()

Функции, позволяющие читать и писать блоки данных.
Блочный ввод-вывод целесообразно использовать с бинарными файлами.
Они имеют следующие прототипы:

size_t fread(void *str, unsigned size, unsigned n, FILE *fp);

size_t fwrite(void *str, unsigned size, unsigned n, FILE *fp);

***str** - указатель на буфер;
size - размер блока;
n - количество блоков;
***fp** - указатель на структуру FILE открытого файла.

Позиционирование в файле

Каждый открытый файл имеет, так называемый указатель на текущую позицию в файле.

При каждом выполнении функции чтения или записи, указатель смещается на количество прочитанных или записанных байт, то есть устанавливается сразу за прочитанным или записанным блоком данных в файле. *Это так называемый последовательный доступ к данным.*

При необходимости чтения или записи данных в произвольном порядке применяется функцией fseek().

Функция fseek() перемещает указатель позиции в потоке.

int fseek(FILE * fp, long int offset, int origin);

offset - количество байт для смещения, относительно положения указателя.
origin - позиция указателя, относительно которой будет выполняться смещение.

Значения, которые может принимать параметр **origin :**

SEEK_SET - Смещение выполняется от начала файла
SEEK_CUR - Смещение выполняется от текущей позиции указателя
SEEK END - Смещение выполняется от конца файла

В случае успеха, функция возвращает нулевое значение, иначе - ненулевое.

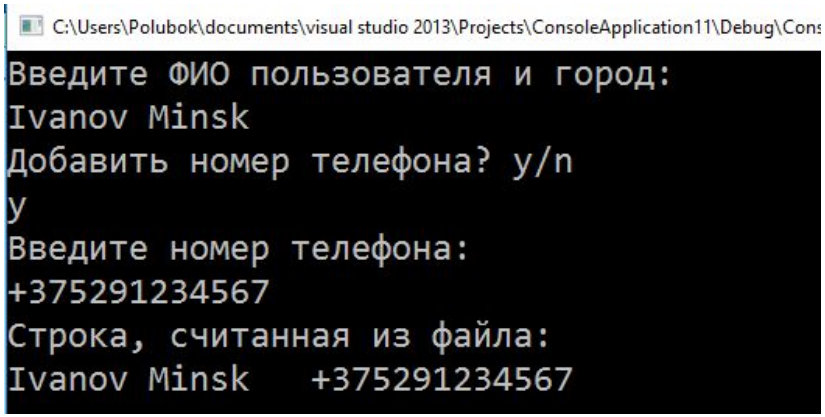
При использовании функции fseek() в текстовых файлах со смещением на величину значения, отличного от нуля, на некоторых платформах, в связи с нестандартным форматом преобразования текстовых файлов, может возникнуть ситуация с некорректного позиционирования указателя.

Функция
fseek()

```
. . .
void main(){

setlocale(LC_CTYPE, "Rus");

int number;
char *str1 = new char[50];
char *str2 = new char[50];
char *str3 = new char[50];
char ch;
FILE *fp;

if ((fp = fopen("d:\\@Work\\test22.txt", "w+")) == NULL) {
printf("Cannot open file.\n");exit(1);
}

puts("Введите ФИО пользователя и город: ");

fflush(stdin);

gets(str1);
str1 = strcat(str1, "   ");
fputs(str1, fp);

puts("Добавить номер телефона? y/n");

scanf("%c", &ch);
```

```
if (ch == 'y'){

puts("Введите номер телефона: ");
fflush(stdin);
gets(str2);

fseek(fp, strlen(str1), SEEK_SET);
fputs(str2, fp);
}

fclose(fp);

if ((fp = fopen("d:\\@Work\\test22.txt", "r")) == NULL) {
printf("Cannot open file.\n");exit(1);
}

fflush(stdin);
puts("Строка, считанная из файла: ");

fgets(str3,255,fp);

puts(str3);

fclose(fp);

_getch();
}
```

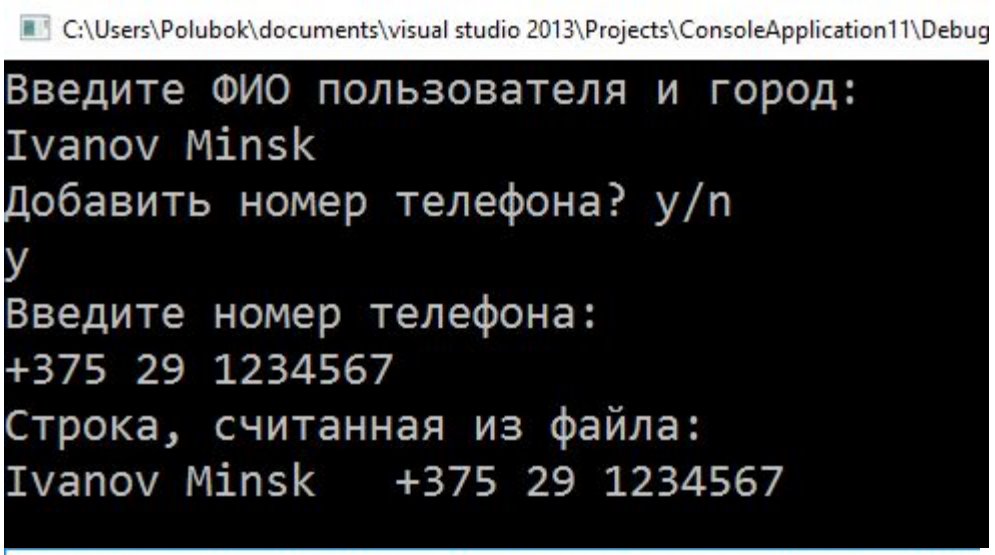
Установка указателя в конец файла



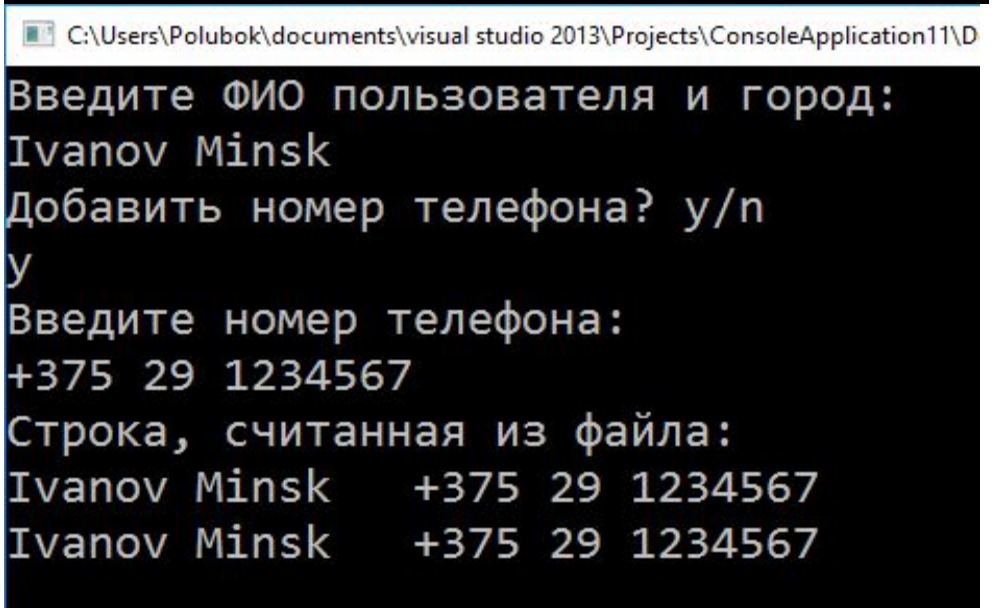
Функции библиотеки Си для работы с файлами

void rewind(FILE *fp); – устанавливает индикатор позиции на начало указанного потока.

```
...
fgets(str3,255,fp);
puts(str3);
*str3 = 0;
fgets(str3, 255, fp);
puts(str3);
...
```




```
...
fgets(str3,255,fp);
puts(str3);
*str3 = 0;
rewind(fp);
fgets(str3, 255, fp);
puts(str3);
...
```



long filelength (int fp);– возвращает длину файла, имеющего дескриптор **fp** , **в байтах**; библиотека **io.h**

```
FILE *fp;

. . .
long int len = _filelength(_fileno(fp));

printf("%d", len);
. . .
```

Определение размера файла

Возвращает дескриптор файла указанного потока.

Дескриптор - это целое число, которое система ставит в соответствие открытому файлу и в дальнейшем использует в операциях работы с файлом

Переименование файла

int rename (const char *oldname, const char *newname);
Успех – 0, неудача – ненулевое значение.

Если файл с **newname** уже существует, то работа функции зависит от реализации.

```
. . .
if ( rename("text.txt", "emp.txt")  )
    printf("This is no such a file.\n");
else printf("This file was renamed.\n");
. . .
```

Удаление файла

int remove(const char *filename);

Успех – 0, неудача – ненулевое значение.