

Севастопольский государственный университет  
Кафедра «Информационные системы»

Курс лекций по дисциплине

**«АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ»**  
(АиТ)

Лектор: Бондарев Владимир Николаевич

# Лекция 12

## Ввод-вывод

# Ввод-вывод

Возможности для ввода и вывода не являются частью самого языка Си.

Рассмотрим **стандартную библиотеку `stdio.h`**, содержащую набор функций, обеспечивающих ввод-вывод.

**Библиотечные функции ввода-вывода** точно определяются стандартом ANSI, так что они совместимы в любых реализациях Си.

# Ввод-вывод

## 1. Стандартный ввод-вывод

Библиотечные функции реализуют простую **модель текстового ввода-вывода**.

**Текстовый поток** состоит из последовательности строк; каждая строка заканчивается символом новой строки.

**Простейший механизм ввода** — это чтение одного символа из *стандартного потока ввода* (клавиатуры) функцией **getchar**:

**int getchar(void)**

В качестве результата функция **getchar** возвращает символ из потока ввода или **EOF ( -1)**, если обнаружен конец файла.

# Ввод-вывод

Пример:

```
#include <stdio.h>
```

```
main ( ) {
```

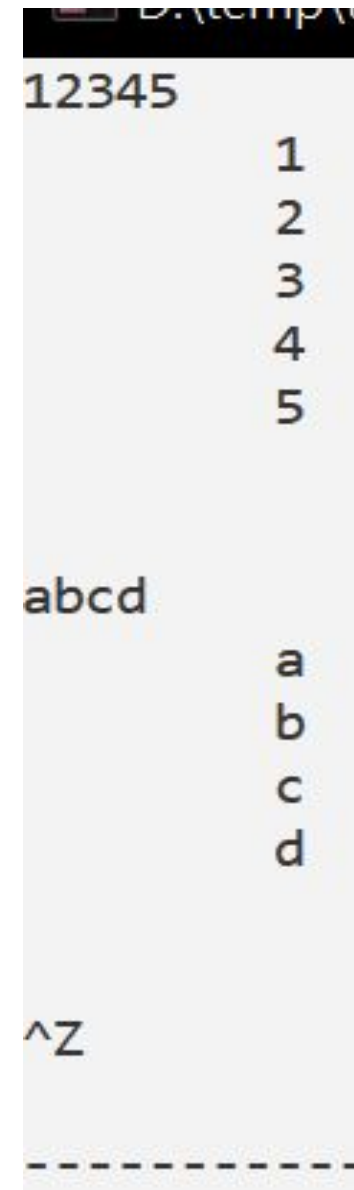
```
    int v;
```

```
    while ((v=getchar()) != -1)
```

```
        printf("\t%c\n", v);
```

```
    return 0;
```

```
}
```



```
D:\temp\c
12345
    1
    2
    3
    4
    5

abcd
    a
    b
    c
    d

^Z
-----
```

# Ввод-вывод

Во многих системах **клавиатуру можно заменить файлом**, перенаправив ввод с помощью значка <. Так, если программа **prog** использует **getchar**, то командная строка

**prog < infile**

предпишет программе читать символы из файла **infile**, а не с клавиатуры.

Функция **int putchar(int)** используется для **вывода**. Вызов **putchar(c)** отправляет символ **c** в *стандартный поток вывода* (дисплей).

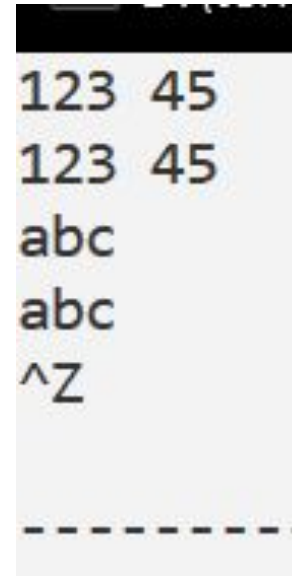
Функция **putchar** в качестве результата возвращает посланный символ или, в случае ошибки, **EOF**.

С помощью записи вида **> имя-файла вывод putchar** можно перенаправить **в файл**:

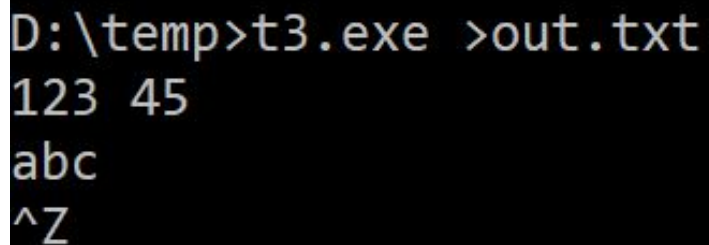
**prog >outfile**

# Ввод-вывод

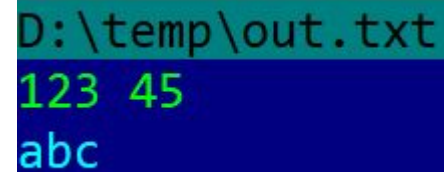
```
/* t3.cpp */  
#include <stdio.h>  
main ( ) {  
    int v;  
    while ((v=getchar()) != EOF)  
        putchar (v);  
    return 0;  
}
```



```
123 45  
123 45  
abc  
abc  
^Z  
-----
```



```
D:\temp>t3.exe >out.txt  
123 45  
abc  
^Z
```



```
D:\temp\out.txt  
123 45  
abc
```

# Ввод-вывод

Перенаправим ввод и вывод.

Ввод происходит не с клавиатуры, а из файла input.txt.

Вывод происходит не на экран, а в файл out.txt

```
D:\temp>copy con input.txt
abcdf
5678
^Z
Скопировано файлов:          1.

D:\temp>t3.exe <input.txt >out.txt
```

```
D:\temp\out.txt
abcdf
5678
```



# Ввод-вывод

## 2. Форматный вывод (printf)

Функция **printf** преобразует, форматирует и печатает свои аргументы в стандартном потоке вывода под управлением строки формата. Возвращает она количество напечатанных символов.

```
int printf(char *format, arg1, arg2, . . . );
```

**Строка формата** содержит два вида объектов: **обычные символы**, которые копируются в выходной поток, и **спецификации преобразования**, каждая из которых вызывает преобразование и печать очередного аргумента **printf**.

**Спецификация преобразования** начинается знаком **%** и заканчивается символом-формата:

**%[флаг][ширина][.точность][h|l]символ\_формата**

где **ширина** — минимальное количество позиций, отводимых под выводимое значение, **точность** — количество позиций, отводимых под дробную часть числа.

# Ввод-вывод

*Модификаторы* [ h|l ] : **h** - **short** или **unsigned short**; **l** - **long** или **unsigned long** (для целых) или **long double** (для вещественных).

**Флаг** – если равен минусу, то выравнивание по левому краю.

**Таблица** – Описание значений поля **символ\_формата**

символ_формата	тип выводимого объекта
<b>c</b>	<b>char</b> ; единичная литера
<b>s</b>	<b>char *</b> ; печатает символы до \0 или в кол-ве заданном точностью
<b>d,i</b>	<b>int</b> ; десятичное целое
<b>o</b>	<b>int</b> ; беззнаковое восьмеричное
<b>u</b>	<b>int</b> ; беззнаковое десятичное целое
<b>x,X</b>	<b>int</b> ; беззнаковая шестнадцатеричное
<b>f</b>	<b>double</b> ; вещественное число с фиксированной точкой
<b>e,E</b>	<b>double</b> ; вещественное число с плавающей точкой
<b>g,G</b>	<b>double</b> ; вещественное число в виде %f или %e в зависимости от значения
<b>p</b>	<b>void *</b> ; указатель (представление зависит от реализации)
<b>%</b>	знак процента %

# Ввод-вывод

**Ширину и точность** можно специфицировать с помощью **\***; значение ширины (или точности) в этом случае берется из следующего аргумента (который должен быть типа **int**). Например, печать не более **max** символов из строки **s** :

```
printf("%.*s", max, s);
```

**Примеры** (печать строки **hello, world** – 12 литер, “:” условно показывает границы поля):

%s	:hello, world:
%10s	:hello, world:
%.10s	:hello, wor:
%-10s	:hello, world:
%.15s	:hello, world:
%-15s	:hello, world :
%15.10s	: hello, wor:
%-15.10s	:hello, wor :

# Ввод-вывод

Функция **sprintf** выполняет те же преобразования, что и **printf**, но вывод запоминает в строке

```
int sprintf(char *string, char *format, arg1, arg2, ...)
```

Заметим, что строка **string** должна быть достаточно большой, чтобы в ней поместился результат.

## 3. Форматный ввод (scanf)

Функция **scanf**, обеспечивающая **ввод**, является обратным аналогом **printf**; она выполняет многие из упоминавшихся преобразований, но в противоположном направлении. Объявление функции:

```
int scanf(char *format, arg1, arg2, ...)
```

Функция **scanf** читает символы из стандартного входного потока, интерпретирует их согласно спецификациям строки **format** и рассылает результаты в свои остальные **аргументы, которые являются указателями**. В качестве результата **scanf** возвращает количество успешно введенных элементов данных. По исчерпанию файла она выдает **EOF**.

# Ввод-вывод

Функция **scanf** прекращает работу, когда оказывается, что исчерпан формат или вводимая величина не соответствует управляющей спецификации.

Существует также функция **sscanf**, которая читает из строки (а не из стандартного ввода):

**int sscanf(char \*string, char \*format, *arg1*, *arg2*, . . . )**

Функция **sscanf** просматривает строку **string** согласно формату **format** и рассылает полученные значения в ***arg1***, ***arg2*** и т. д. Последние должны быть указателями.

Спецификация:

**%[\*][ширина][символ-формата]**

**\*** - поле ввода пропускается и присваивание не выполняется.

# Ввод-вывод

Таблица – Описание значений поля **символ\_формата**

символ_формата	тип поля; вводимые данные
<b>c</b>	<b>char *</b> ; единичная литера
<b>s</b>	<b>char *</b> ; строка
<b>d, u</b>	<b>int *</b> ; десятичное целое
<b>i, o, x</b>	<b>int *</b> ; целые ( i – p=8 или p=16)
<b>e, f, g</b>	<b>float *</b> ; вещественное число с плавающей точкой

Перед символами-формата **d, i, o, u** и **x** может стоять буква **h**, указывающая на то, что соответствующий аргумент должен иметь тип **short \*** (а не **int \***), или **l**, указывающая на тип **long \***.

Аналогично, перед символами-спецификаторами **e, f** и **g** может стоять буква **l**, указывающая, что тип аргумента – **double \*** (а не **float \***).

# Ввод-вывод

Пример:

```
#include <stdio.h>

main ( ) { /* программа-калькулятор */
    double sum, v;
    sum = 0;
    while (scanf ("%lg", &v) == 1)
        printf("\t%.2lg\n", sum += v);
    return 0;}
```

1	1.00
0.1	1.10
0.2	1.30
0.3	1.60
^Z	

Предположим, что нам нужно прочесть из потока ввода:

**26 декабря 1928**

Обращение к **scanf** выглядит следующим образом:

```
int day, year;           /* день, год */
char monthname[10];      /* название месяца */
scanf ("%d%s%d", &day, monthname, &year);
```

При вводе **scanf** игнорирует пробелы и табуляции.

# Доступ к файлам

Для того чтобы можно было читать из файла или писать в файл, он должен быть предварительно *открыт* с помощью библиотечной функции **fopen**. Функция **fopen** получает внешнее имя файла и **возвращает указатель**, используемый в дальнейшем для доступа к файлу. Этот указатель, называемый *указателем файла*, ссылается на структуру **FILE**, содержащую информацию о файле (адрес буфера, *счетчик положения текущего символа в буфере*, открыт файл на чтение или на запись, были ли ошибки при работе с файлом и не встретился ли конец файла). Структура **FILE** определена в заголовочном файле **<stdio.h>**.

Функция **fopen** имеет следующий прототип:

**FILE \*fopen(const char \*filename, const char \*mode);**

Функция открывает файл с **именем** **filename**. Параметр **mode** задаёт **режим**, в котором открывается файл.



# Доступ к файлам

Таблица – Режимы открытия файлов

Режим	Описание режима
<b>r</b>	Файл открывается только для чтения
<b>w</b>	Файл создается только для записи. Если файл с этим именем уже существует, он будет перезаписан. Чтение из файла не разрешено.
<b>a</b>	Режим добавления записей (append). Файл открывается только для записи в конец или создается только для записи, если он еще не существует. Чтение не разрешено.
<b>r+</b>	Существующий файл открывается для обновления (считывания и записи)
<b>w+</b>	Создается новый файл для обновления. Перезаписывается любой существующий файл с тем же именем.
<b>a+</b>	Файл открывается для добавления в конец; если файл не существует, он создается, и любой существующий файл с тем же именем перезаписывается.

# Доступ к файлам

Чтобы указать, что данный файл открывается или создается как **текстовый**, добавьте символ **t** в строку режима (например, “**rt**”, “**w+t**” и т.п.).

Аналогично можно сообщить, что файл открывается или создается как **бинарный**. Для этого добавьте в строку режима символ **b** (например, “**wb**”, “**a+b**”).

В случае успеха функция **fopen** возвращает указатель на открытый поток, в случае ошибки – **NULL**.

Например:

```
FILE *fptr = fopen(“mytxt.txt”, “rt”);
```

Здесь объявляется **указатель на файл fptr** и выполняется его инициализация с помощью функции **fopen()**.

Для завершения работы с файлом он должен быть закрыт с помощью функции **fclose()**:

```
fclose(fptr);
```

# Доступ к файлам

Функция **fgetc( )** имеет следующий прототип:

```
int fgetc(FILE *fptr);
```

Она осуществляет **ввод символа** из файлового потока **fptr**. В случае успеха функция возвращает код символа. Если делается попытка прочесть конец файла или произошла ошибка, то возвращается **EOF**. Имеется аналогичная функция **getc** (оптимизирована и реализована в виде макроса).

Функция **fputc()** имеет следующий прототип:

```
int fputc(int c, FILE *fptr);
```

Она осуществляет **вывод символа в поток**. При ошибке возвращает **EOF**, иначе — записанный символ. Имеется аналогичная функция **putc**.

При запуске Си-программы операционная система всегда открывает **три файла** и обеспечивает **три файловые ссылки** на них: **stdin**, **stdout** и **stderr**; они описаны в **<stdio.h>**. Обычно **stdin** соотнесен с клавиатурой, а **stdout** и **stderr** — с экраном.

# Доступ к файлам

С помощью **getc**, **putc**, **stdin** и **stdout** функции **getchar** и **putchar** теперь можно определить следующим образом:

```
#define getchar( ) getc(stdin)
```

```
#define putchar(c) putc((c), stdout)
```

Функция **fgets()** имеет следующий прототип:

```
char *fgets(char *s, int n, FILE *fptr);
```

Она осуществляет **чтение строки символов** из файлового потока в строку **s**. Функция прекращает чтение, если прочитано **n-1** символов или встретится символ перехода на новую строку **'\n'**. Если этот символ встретился, то он сохраняется в переменной **s**. В обоих случаях в переменную **s** добавляется символ **'\0'**. В случае успеха функция возвращает указатель на считанную строку **s**. Если произошла ошибка или считана метка **EOF**, то возвращается **NULL**.

# Доступ к файлам

Для записи строки в файл можно использовать функцию **fputs**.

**int fputs(const \*char, FILE \*fptr);**

В случае успеха функция **fputs()** возвращает неотрицательное значение. В противном случае она возвращает **EOF**.

**Форматированный ввод-вывод** текстовых файлов организуется с помощью функций **fscanf()** и **fprintf()**. Эти функции аналогичны функциям **scanf()** и **printf()** с той лишь разницей, что их первым аргументом является указатель на файл, открытый в соответствующем режиме:

**int fscanf(FILE \*f, const char \*format, arg1, arg2, ...)**

**int fprintf(FILE \*f, const char \*format, arg1, arg2, ...)**

Функция **feof()** распознаёт **конец файла**. Она имеет прототип:

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

Функция возвращает **0**, если конец файла не достигнут.

# Доступ к файлам

Программа копирует целые числа из входного файла **old.txt** в выходной файл **new.txt**.

```
#include <stdio.h>
```

```
int main() {
```

```
    int i, x;
```

```
    FILE *in,*out; // описание указателей на файлы
```

```
    if ((in = fopen("c:\\1\\old.txt","rt"))== NULL) {
```

```
        fprintf(stderr," Не могу открыть входной файл \n");
```

```
        return 1;}
```

```
    if ((out = fopen("c:\\1\\new.txt","wt"))== NULL) {
```

```
        fprintf(stderr," Не могу открыть выходной файл \n");
```

```
        return 1;}
```

```
    i = 0;
```

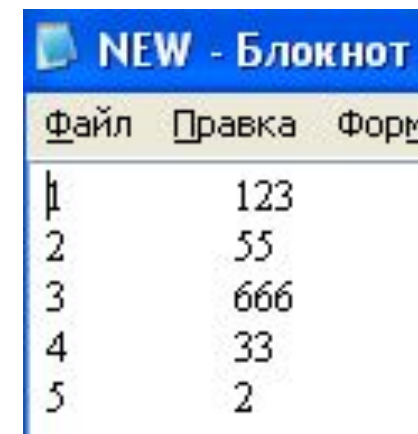
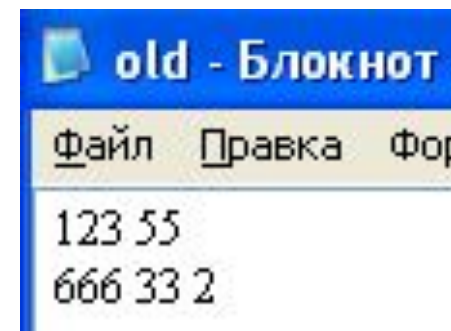
```
    while (fscanf(in,"%d",&x)!=EOF)
```

```
        { i++;
```

```
          fprintf(out,"%d\t%d\n", i, x);}
```

```
    fclose(in); fclose(out);
```

```
    return 0; }
```



# Пример

Программа копирует символы из входного файла **old.txt** в выходной файл **new.txt**.

```
#include <stdio.h>

int main() {
    char ch;
    FILE *in,*out;
    if ((in = fopen("old.txt","rt"))== NULL) {
        fprintf(stderr,"error1\n "); return 1;}

    out = fopen("new.txt","wt");
    if (out == NULL) { fprintf(stderr, "error2\n"); return 1;}

    while (fscanf(in,"%c",&ch)!=EOF)
        {   fprintf(out,"%c", ch);}

    fclose(in); fclose(out);
    return 0;
}
```

# Пример

**// 1 версия**

```
while (fscanf(in,"%c",&ch)!=EOF)
    fprintf(out,"%c", ch);
```

**// 2 версия**

```
while (1) {
    if(fscanf(in,"%c",&ch)==EOF) break;
    fprintf(out,"%c", ch);
}
```

**// 3 версия**

```
while (1) {
    fscanf(in,"%c",&ch);
    if (feof (in)) break;
    fprintf(out,"%c", ch);
}
```

**// 4 версия**

```
while (1) {
    ch=fgetc(in);
    if (feof (in)) break;
    fputc(ch,out); }
```



# Пример

```
char *s=(char*)malloc(80*sizeof(char));  
while (1) {  
    s=fgets(s,80,in);  
    if (feof (in)) break;  
    for (int i=0;s[i]!='\0';i++)  
        printf("%c",s[i]);  
        fputs(s,out);  
}  
free(s);
```

# Доступ к файлам

Для осуществления *неформатированного ввода-вывода* (без преобразований) применяются функции **fread()** и **fwrite()**. Эти функции имеют следующие прототипы:

```
size_t fread(void *ptr, size_t size, size_t n, FILE *fptr);
```

```
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *fptr);
```

Функция **fread()** **считывает блоки данных** из файлового потока, на который указывает **fptr**, в буфер, доступ к которому выполняется через указатель **ptr**, а функция **fwrite()** выполняет обратную операцию, то есть **записывает блоки данных** из буфера в файловый поток.

При этом копируется **n** блоков данных, каждый из которых содержит **size** байтов. В случае успеха функции **возвращают число скопированных блоков**. В случае ошибки возвращается **0** или число полностью скопированных блоков. Если **n** больше значения, которое вернула функция **fread**, то это говорит о том, что встретилась метка конец файла.

# Доступ к файлам

Сразу после открытия файла *счетчик положения текущего байта файла* равен 0. Каждая операция ввода-вывода вызывает приращение этого счетчика на число записанных или считанных байтов из файла.

**Функции позиционирования** – `fseek()`, `ftell()` и `rewind()` позволяют изменять или получать значение счетчика, связанного с файлом.

Функция `long int ftell(FILE *fptr)` **возвращает текущую позицию в файле**. В случае ошибки возвращает `-1L`.

Функция `int fseek(FILE *fptr, long offset, int from)` **перемещает указатель позиции в файле `fptr` на `offset` байтов относительно позиции `from`**, где `from` может иметь значения:

`SEEK_SET (=0)`    начало файла;

`SEEK_CURR (=1)`    текущая позиция в файле;

`SEEK_END (=2)`    конец файла.

Функция возвращает 0, если счетчик текущей записи успешно изменен.

# Доступ к файлам

Функция **void rewind(FILE \*fptr)**

устанавливает счетчик текущей позиции на начало файла.

## Пример:

Написать программу, которая считывает текст из файла и выводит в выходной файл только вопросительные предложения из этого текста.

### Алгоритм решения задачи:

1. Открыть файл.
2. Определить его длину.
3. Выделить в динамической памяти соответствующий буфер.
4. Считать файл с диска в буфер.
5. Анализируя буфер посимвольно, выделять предложения. Если предложение оканчивается “?” выводить его в файл.

В программе будем использовать функции **чтения блоков данных** из входного файла, так как применение функций посимвольного чтения неэффективно.

# Доступ к файлам

```
#include <stdio.h>

int main(){
    //открытие входного файла
    FILE *fin;
    fin=fopen("d:\\temp\\input.txt","r");
    if (!fin) {
        puts("Can't open input file");
        return 1;
    }
    fseek(fin,0,SEEK_END);    //указатель в конец файла
    long len=ftell(fin);      //запомнить длину файла

    //выделить память под буфер
    char *buf= (char *) calloc(len+1, sizeof(char));
```

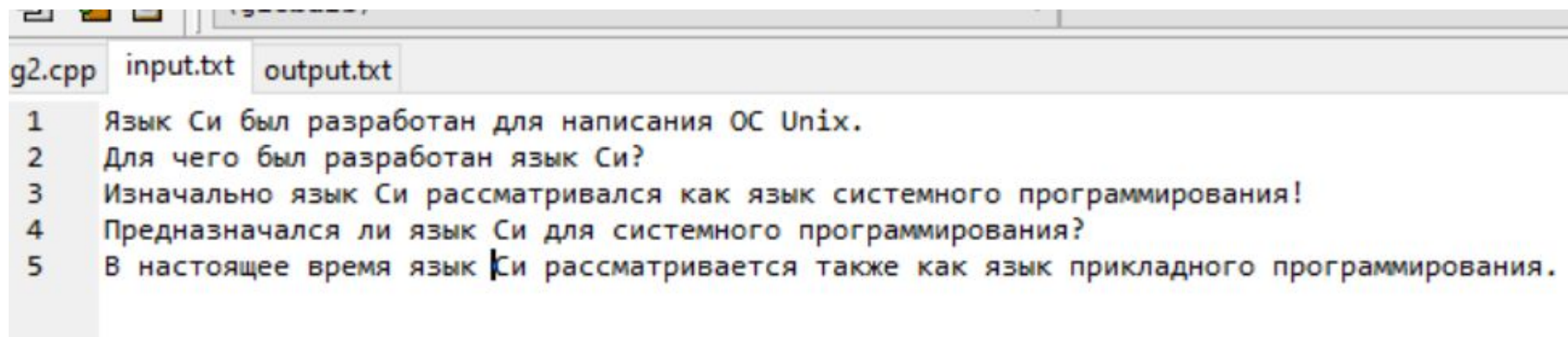
# Доступ к файлам

```
//неформатированное чтение текстового файла (поблочное)
const int l_block=1024;      //задать длину блока для чтения
int num_block=len/l_block; //определить число блоков
rewind(fin);                 //указатель в начало файла
fread(buf, l_block, num_block+1, fin); //чтение блоков из файла
buf[len]='\0';               //поместить в буфер нуль-литеру
//создание выходного файла
FILE *fout;
fout = fopen("d:\\temp\\output.txt","w");
if (!fout) {
    puts("Can't open output file");
    return 1;
}
```

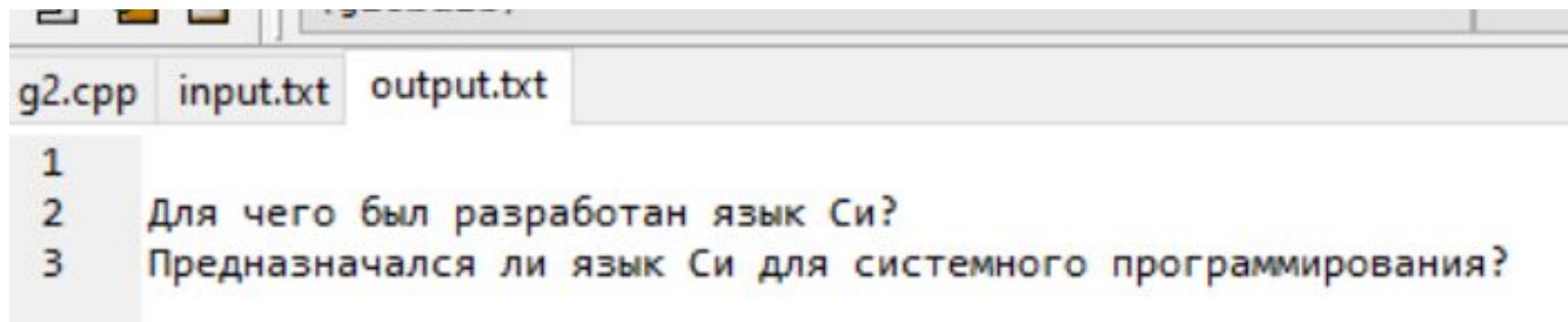
# Доступ к файлам

```
long n=0,      //индекс символа начала предложения
    i=0,      //индекс символа конца предложения
    j=0;      //текущий индекс символа вопросит. предл.
while(buf[i]) { //просмотр символов в буфере
    if(buf[i]=='?') { //если i-ый символ – вопрос,
        for(j=n; j<=i; j++)
            putchar(buf[j], fout); //то вывод предложения в поток,
        n=i+1; //обновление индекса начала предл.
    }
    if (buf[i]=='.'||buf[i]=='!') //если предл. не вопросительное,
        n=i+1; //то обновление только индекса n
    i++;
}
fclose(fin); fclose(fout);
printf("\n");
return 0;}
```

# Результаты работы программы



```
g2.cpp input.txt output.txt
1  Язык Си был разработан для написания ОС Unix.
2  Для чего был разработан язык Си?
3  Изначально язык Си рассматривался как язык системного программирования!
4  Предназначался ли язык Си для системного программирования?
5  В настоящее время язык Си рассматривается также как язык прикладного программирования.
```



```
g2.cpp input.txt output.txt
1
2  Для чего был разработан язык Си?
3  Предназначался ли язык Си для системного программирования?
```