Язык программирования Си

Бикулов Д.А., Иваницкая Н.В., Иванов А.П.

Семинар 7. Ввод и вывод: форматированный и бесформатный. Работа с файлами. Строки Си. Функции работы со строками.

1 Форматированный вывод

Форматированный вывод выполняет функция **`printf()`**, которая описана в стандартной библиотеке **`<stdio.h>`**:

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

Эта функция преобразует, форматирует и печатает свои аргументы в стандартном выводе согласно заданной строке формата. Возвращает она количество напечатанных символов.

Любая спецификация формата начинается знаком «%» и заканчивается символом-спецификатором. Перечислим наиболее употребительные символы-спецификаторы:

Символ	Тип аргумента	Описание
d	int	десятичное целое
x	unsigned int	беззнаковое шестнадцатеричное целое
u	unsigned int	беззнаковое десятичное целое
c	char	одиночный символ
s	char*	строка символов
f	float	вещественное число
e	double	научный формат со строчной e
E	double	научный формат с прописной E
g	double	либо e, либо f — что короче
G	double	либо E, либо f — что короче
lf	double	вывод чисел с двойной точностью
p	void*	вывод адреса (указателя)

Функция **`printf()`** использует свой первый аргумент (форматирующую текстовую строку), чтобы определить, сколько еще ожидается аргументов и какого они будут типа. Правильный результат не удастся получить, если аргументов будет не хватать или их типы не будут совпадать с указанными в строке формата. В форматирующей строке можно писать любой текст – он будет просто напечатан, но кроме того, с помощью специальных форматирующих последовательностей символов можно управлять правилами вывода прочих аргументов. Символ «%» используется для обозначения начала форматирующей последовательности, для печати самого символа «процент» его надо удвоить: «%%».

Для печати переменных **`long`** и **`short`** типов к спецификаторам следует добавлять в начале **`l`** или **`h`** соответственно. Например, для печати переменной типа **`long int`** спецификатор формата выглядит так: **`%ld`**.

Для добавления табуляции в строке формата используется управляющая последовательность «**`\t`**», для перевода курсора на следующую строку — «**`\n`**».

Для каждого спецификатора формата можно задать минимальную ширину поля вывода. Так, для заполнения ведущими нулями целого числа до ширины в 6 знаков, надо написать «**`%06d`**». Если ширина исходного числа окажется больше указанной, то дополнительных нулей проставлено не будет, а ширина поля автоматически увеличится, чтобы

поместилось все число. Помимо заполнения нулями можно задать заполнение пробелами, для этого не надо указывать первый ноль в описании формата: «**%6d**». Модификаторы минимальной ширины чаще всего используются для форматирования таблиц.

Для каждого спецификатора формата можно задать модификатор точности. Например, чтобы вывести число с плавающей точкой с точностью до 4 знака, надо написать «**%.4f**». Поведение модификатора точности зависит от типа данных: для строк он задает максимальную ширину поля (если строка окажется длиннее, то последние символы будут отброшены), для вещественных чисел он задает число знаков после запятой, а для форматов «**%g**» и «**%G**» задает число значащих цифр. Для целых чисел модификатор точности обозначает минимальное число цифр, и если число оказывается меньше, то оно дополняется ведущими нулями.

Модификаторы точности и ширины поля можно комбинировать. Например, что задать ширину поля 20 и вывести число с плавающей точкой и четырьмя цифрами после запятой, надо написать «**%20.4f**». Чтобы вывести строку минимум из 8 символов и максимум из 12, надо написать (не очень интуитивно, что сначала идет меньшее число, но это не опечатка) «**%8.12s**».

2 Форматированный ввод

Форматированный ввод выполняет функция **scanf()**, которая описана в стандартной библиотеке **<stdio.h>** и является аналогом **printf()** для ввода:

```
int scanf(char *format, pointer1, pointer2, ...);
```

Функция **scanf()** читает символы из стандартного входного потока, интерпретирует их согласно спецификациям форматирующей строки и сохраняет результаты в свои остальные аргументы, каждый из которых должен быть указателем. Указатель определит, где будут запоминаться должным образом преобразованные данные.

Функция **scanf()** прекращает работу, когда оказывается, что исчерпался формат или водимая величина не соответствует управляющей спецификации.

В качестве результата **scanf()** возвращает количество успешно введенных элементов данных.

Функция **scanf()** работает с теми же символами-спецификаторами, что и функция **printf()**.

При вводе строк функция **scanf()** со спецификатором «**%s**» читает до первого пробельного разделителя: перевода на новую строку, табуляции, пробела. Поэтому нельзя использовать один вызов **scanf()** для чтения строки «Привет, мир!», в переменную будет записано только «Привет,».

Функция **scanf()** допускает добавление модификаторов (ширина поля и др.) к спецификаторам формата. Так, для строк, если задан модификатор ширины поля «**%80s**», то будет прочитано не более 80 символов. Следующий ввод начнется с того места, где завершился предыдущий.

У функции **scanf()** есть также особая возможность, которой нет у **printf()**: т. н. спецификатор универсального формата — задание набора сканируемых символов. Например, если в качестве спецификатора формата указан «**%[ABC]**», то в соответствующий символьный массив будет считаны из входного потока только перечисленные символы: A, B и C. Как только **scanf()** находит символ, не перечисленный в наборе, он останавливает считывание в переменную, соответствующую набору сканируемых символов. Если первым символом стоит «**^**», то набор трактуется

«наоборот»: ввод символов, не входящих в набор. Набор символов чувствителен к регистру.

Пример.

```
#include <stdio.h>

int input_output()
{
    int nn;
    char cc;
    float ff;
    double dd;
    printf("INPUT nn, cc, ff, dd\n");

// в качестве указателей используются адреса соответствующих
// переменных:
    scanf("%d%c%f%lf", &nn, &cc, &ff, &dd);

    printf("nn=%d\tcc=%c\tff=%f\tdd=%lf", nn, cc, ff, dd);
    return 0;
}
```

Одна из самых распространенных ошибок случается, если вместо того, чтобы написать:

```
  scanf("%d", &nn);
```

указывают в аргументе не указатель на считываемую переменную, а ее саму:

```
  scanf("%d", nn);
```

Компилятор о подобной ошибке ничего не сообщает, но переменную **nn** введенное значение помещено не будет и программа может вообще аварийно завершиться во время исполнения.

3 Работа с файлами

Для того чтобы можно было читать из файла или писать в файл, он должен быть предварительно открыт с помощью функции **fopen()** (**<stdio.h>**), которая после вызова возвращает указатель на файл, используемый в дальнейшем для доступа к файлу.

Этот указатель ссылается на структуру, содержащую информацию о файле (адрес буфера, положение текущего символа в буфере, открыт файл на чтение или на запись, были ли ошибки при работе с файлом и не встретился ли конец файла). Определения, полученные из **<stdio.h>**, включают описание такой структуры, называемой **FILE.**

Для определения указателя файла требуется задать описания вида:

```
  FILE *fp;
  fp = fopen( "z:\\filename.txt", "r");
```

Здесь **fp** – указатель на **FILE**, а **fopen()** возвращает указатель на **FILE**. Первый аргумент функции **fopen()** – строка, содержащая имя файла. Второй аргумент несет информацию о режиме. Это тоже строка, в ней указывается, каким образом пользователь намерен работать с файлом. Возможны следующие режимы открытия файла:

- **"r"** (read) – файл открывается на чтение, ошибка, если файла нет;
- **"w"** (write) – файл открывается на запись, файл создается, если его нет;
- **"a"** (append) – добавление в конец файла, файл создается, если его нет;
- **"r+"** (r+w) – файл открывается на чтение и запись, ошибка, если файла нет;

- **"w+"** (w+r) – файл открывается на чтение и запись, файл создается, если его нет, если же файл есть – то его содержимое удаляется;
- **"a+"** (a+r) – файл открывается на чтение и запись в конец, файл создается, если его нет;
- **"t"** (text) – дополнительный флаг, который можно указывать в сочетании с другими флагами в одной строке, означает, что файл открывается в текстовом режиме, при котором из файла на месте конца строки «**\n**» будет прочитано два символа: «**\r\n**», а при записи, наоборот, на месте двух символов «**\r\n**» будет записан один «**\n**»; этот флаг считается установленным по умолчанию, поэтому указывать его явно – не обязательно;
- **"b"** (binary) – аналогичный предыдущему дополнительный флаг, он означает, что файл открывается в бинарном режиме, при котором никаких преобразований конца строк при чтении-записи не выполняется.

При возникновении любой ошибки **fopen()** возвращает **NULL**, а получить код ошибки можно, воспользовавшись функцией **ferror(fp)**.

Функция:

```
int fclose(FILE *fp);
```

является обратной по отношению к **fopen()**, она разрывает связь между указателем на файл и внешним именем, которая раньше была установлена с помощью **fopen()**, освобождая тем самым этот указатель для других файлов.

После того, как файл открыт на запись, в него можно записать данные при аналога функции **printf()**:

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

А если файл открыт на чтение, то из него можно читать аналогом функции **scanf()**:

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

В любой программе автоматически открыты три стандартных потока ввода-вывода:

- **stdin** (аналог **cin** в Си++) – стандартный поток ввода, связан с консолью;
- **stdout** (аналог **cout** в Си++) – стандартный поток вывода, связан с консолью;
- **stderr** (аналог **cerr** в Си++) – стандартный поток вывода сообщений об ошибках, связан с консолью;

С ними можно работать сразу, указывая их в соответствующих параметрах функций ввода и вывода, в том числе – бесформатных, о которых речь пойдет дальше.

Если нужно гарантировать немедленный вывод данных в файл (например, для немедленного отображения выводимых данных в консоли), следует воспользоваться функцией опустошения буфера ввода-вывода для данного файла:

```
int fflush(FILE* fp);
```

Признак достигнутого конца файла можно проверить так:

```
while( !feof(fp) ) { ... /* любые операции чтения */ };
```

4 Бесформатный вывод и ввод

```
size_t fread( void *ptr, size_t size, size_t nobj, FILE* fp );
```

fread() читает из потока **stream** в массив **ptr** не более **nobj** объектов размера **size**.

Она возвращает количество прочитанных объектов, которое может быть меньше заявленного.

Для индикации состояния после чтения следует использовать **feof** и **ferror**.

```
size_t fwrite( const void* ptr, size_t size, size_t nobj, FILE* fp );
```

fwrite() пишет из массива **ptr** в stream **nobj** объектов размера **size**; возвращает число записанных объектов, которое в случае ошибки меньше **nobj**.

Среди прочих функций бесформатного вывода отметим функции чтения и записи строки:

```
int   fputs( const char* str, FILE* fp );     // запись
char* fgets( char* str, int size, FILE* fp ); // чтение
```

чтения-записи символа:

```
int   fputc( int chr, FILE* fp );             // запись
int   fgetc( FILE* fp );                      // чтение
```

и функцию возврата зря прочитанного символа обратно в поток, из которого он был прочитан:

```
int ungetc ( int chr, FILE* fp );
```

Эта последняя функция может вернуть в поток чтения файла не обязательно тот самый символ, который был прочитан перед этим – вернуть можно любой символ! После выполнения этой функции следующая операция чтения из файла прочитает именно этот возвращенный символ.

Пример.

```
#include <stdio.h>

// функция создает файл с именем f1.txt для записи
// заполняет его данными: массив, таблица, символ, строка
void create_file()
{
    int i=123, j;
    char ch, cm[33];
    FILE *fp;

    //"w" - файл открыт для записи,
    // маркер указывает на начало файла, при записи
    // старое содержимое затирается

    if( (fp=fopen("f1.txt","w")) != NULL ) {
        for(i=0; i<10; i++)
            fprintf(fp,"%d\t",i*i*i);
        fprintf(fp,"\n");

        // пишем таблицу чисел:
        for(i=0; i<4; i++) {
            for(j=0; j<7; j++)
                fprintf(fp,"%d\t",i+j);
            fprintf(fp,"\n");
        }

        puts("INPUT SYMBOL:"); // пишем строку в консоль
        ch=getchar();          // читаем один символ с консоли
```

```
        fputc(ch,fp);           // пишем символ в файл
        fputc('\n',fp);         // пишем символ конца строки в файл

        puts("INPUT STRING:"); // пишем строку в консоль
        gets(cm);              // читаем строку из консоли
        puts(cm)               // пишем строку в консоль;

        fputs(cm,fp);           // пишем строку в файл
        fputc('\n',fp);         // пишем символ конца строки в файл
        fclose(fp);             // закрываем файл

    }                           // конец if, где файл был открыт
    else printf("ERROR WHILE OPENING FILE!!\n");
}

////////////////////////////////////////////////
// имя файла для чтения передается функции в качестве
// входного параметра
void read_from_file(char* filename)
{
    int i, j, matr[4][7], mass[10];
    char cc, str[34];

    FILE *fp;
    if( (fp=fopen(filename,"r")) != NULL) {

        for(i=0; i<10; i++) {
            fscanf(fp,"%d",&mass[i]);
            printf("%d\t",mass[i]);
        }
        printf("\n");

        for(i=0; i<4; i++) {
            for(j=0; j<7; j++) {
                fscanf(fp,"%d",&matr[i][j]);
                printf("%d\t",matr[i][j]);
            }
            printf("\n");
        }

        // считываем символы из файла и тут же их пишем в консоль:
        while((cc=fgetc(fp)) != '\n')
            putchar(cc);

        fgets(str,33,fp);       // читаем строку из файла
        puts(str);              // пишем строку в консоль

        fclose(fp);             // закрываем файл
    }                           // конец if, где файл был открыт
    else printf("ERROR WHILE OPENING FILE FOR READING!!\n");
}

////////////////////////////////////////////////
void main()
{
    char filename[21];

    create_file();
    puts("INPUT FILENAME:");
    gets(filename);
    puts(filename);
    read_from_file(filename);
}
```

5 Строки Си. Функции работы со строками.

Строка – это массив символов. Во внутреннем представлении строки в конце обязательно присутствует нулевой символ `'\0'`, поэтому памяти для строки требуется на один байт больше, чем число символов, расположенных между двойными кавычками:

```
"ARS LONGA, VITA  BREVIS."
```

Функции, работающие со строками, находятся в библиотеке **<string.h>**. Вот некоторые из них (везде: **s**, **t** – указатели на строки (тип **char***), **n** – целая переменная):

strlen(s) – вычисляет длину строки **s** без учета завершающего ее символа `'\0'`;

strcpy(s, t) – копирует строку **t** в строку **s**;

strncpy(s, t, n) – то же для **n** символов;

strcat(s, t) – дописывает строку t в конец строки **s**;

s = strdup(t) – создает новый дубль строки **t**;

i = strcmp(s, t) – возвращает **-1**, если **s** лексикографически меньше **t**

+1, если **s** лексикографически больше **t**

0, если **s** совпадает с **t**

В последней функции производится сравнение в лексикографическом смысле, а именно: сравнение посимвольное, до первого несовпадающего символа, при этом символы сравниваются по их алфавитному порядку, более короткая строка считается меньше более длинной, даже, если все ее символы совпадают с началом более длинной строки: строка «**sort**» будет меньше строки «**sort it**», но больше строки «**song**». Иными словами, лексикографический порядок строк – это такой порядок, в котором строки записываются в оглавлении словаря или телефонной книги.

s = strchr(t, c) – ищет символ **c** в строке **t**;

s = strrchr(t, c) – то же, но поиск ведется от конца строки к ее началу;

z = strstr(s, t) – ищет строку **t** в **s**;

i = strcspn(s, strCharSet) – ищет в строке **s** любой из символов, не входящих в строку **strCharSet**;

strupr(s) – переводит все буквы строки **s** в верхний регистр;

strlwr(s) – переводит все буквы строки **s** в нижний регистр;

strset(s, c) – все символы строки **s** устанавливает равными символу **c**.

Библиотека работы со строками включает и другие полезные функции, которые можно посмотреть в справочной системе. В частности, там можно найти варианты приведенных функций, которые принимают дополнительный параметр – ограничение на длину обрабатываемой строки.

Пример.

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

void main()
{
    int n=0, len1, len2;


    fputs("input n: ", stdout); fflush(stdout);
    scanf("%d", &n); // ввести число символов

    // строка p1 - символьный массив:
    char *p1 = (char*)malloc(n);

    printf("input string of %d symbols\n", n-1); fflush(stdout);
    fgets(p1, n-1, stdin);
    // здесь надо ввести (n-1) символ, последний символ
    // введенной строки будет '\n' (соответствует нажатию ENTER)

    fputs(p1,stdout);
    free(p1);

//////////////////////////////////////////////////////
// сложение двух строк

    char *ps;
    char ch1[] = "ARS LONGA,";
    char ch2[] = "VITA BREVIS.";

    // так можно посчитать число символов в строке:
    len1 = 0;
    while(ch1[len1]!='\0') ++len1;

    printf("len1=%d\n", len1);

    // или с помощью функции:
    len2 = strlen(ch2);

    printf("len1=%d, len2=%d\n", len1, len2);

    char *ps = (char*)malloc(len1+len2+1);
    strcpy(ps,ch1);
    strcat(ps,ch2);

    // эта функция присоединила бы только 4 символа:
    // strncat(ps,ch2,4);

    fputs(ps,stdout);
    fflush(stdout);
    free(ps);

//////////////////////////////////////////////////////
// сравниваем две строки

    char ch3[] = "gaz";
    char ch4[] = "garbage";
    char ch5[] = "gad";

    printf( "compare \'%s\' and \'%s\' => %d\n", ch3, ch5, strcmp(ch3,ch5) );
```

```
    printf( "compare \'%s\' and \'%s\' => %d\n", ch4, ch3, strcmp(ch4,ch3) );

    // сравнение только двух символов:
    printf( "compare \'%s\' and \'%s\' (2 chr) => %d\n", ch5, ch4,
            strncmp(ch5,ch4,2) );

    fflush(stdout);
}
```

Типовое задание: написать программу, которая читает данные (например, матрицу чисел произвольной размерности) из одного файла, как-то преобразует прочитанные данные (например, транспонирует прочитанную матрицу) и записывает получившийся результат в другой файл.

1. Вариант

В файле содержится информация типа **«химический элемент»** в следующем виде: полное название на латинском языке, символ элемента из двух букв, атомная масса, символ, обозначающий принадлежность к металлам, неметаллам, газам. Считать информацию из файла, в другой файл записать только информацию о металлах.
Размер исходного файла неизвестен.

2. Вариант

В файле содержится информация типа **«студент физик»** в следующем виде: фамилия, имя, год рождения, массив оценок, полученных в первую сессию. Считать информацию из файла, в другой файл записать только информацию о круглых отличниках.
Размер исходного файла неизвестен.

3. Вариант

В файле содержится информация типа **«кинофильм»** в следующем виде: название фильма, год выпуска, бюджет, награды. Считать информацию из файла, в другой файл записать информацию только о фильмах, получивших премию «Оскар».
Размер исходного файла неизвестен.

4. Вариант

В файле содержится информация типа **«музыкальная группа»** в следующем виде: название, год создания, год распада, страна происхождения. Считать информацию из файла, в другой файл записать информацию только о группах из России.
Размер исходного файла неизвестен.

5. Вариант

В файле содержится информация типа **«великий физик»** в следующем виде: фамилия, имя, год рождения, годы великих открытий, страна. Считать информацию из файла, в другой файл записать только информацию об ученых из Великобритании.

Размер исходного файла неизвестен.

6. Вариант

В файле содержится информация типа **«футбольный клуб»** в следующем виде: название, страна происхождения, годы побед. Размер файла неизвестен. Считать информацию из файла, в другой файл записать информацию только о клубах из России.

Размер исходного файла неизвестен.

7. Вариант

В файле содержится информация типа **«олимпийские игры»** в следующем виде: город, страна, год проведения, зимняя или летняя. Размер файла неизвестен. Считать информацию из файла, в другой файл записать информацию только о летних олимпиадах.

Размер исходного файла неизвестен.

8. Вариант

В файле записан массив натуральных чисел. Считать массив до конца файла, в другой файл записать только простые числа из исходного массива.
Размер исходного массива неизвестен. Необходимо использовать функции выделения динамической памяти.

9. Вариант

В файле записаны пары натуральных чисел. Количество пар заранее неизвестно. Для каждой пары чисел требуется определить наибольший общий делитель. Соответствующие тройки чисел записать в другой файл. Например:

```
6        8          2
64      100        4
```

Требуется использовать функции выделения динамической памяти.

10. Вариант

В файле записаны несколько строк чисел, по три целых числа в строке. Это – коэффициенты квадратного уравнения. Сформировать новый файл следующего вида: коэффициенты квадратного уравнения, значения корней. Например:

```
3     -24   45          x1=3        x2=8
1     -4    8           x1=2-2i     x2=2+2i
```

Количество троек чисел заранее неизвестно. Требуется использовать функции выделения динамической памяти.

11. Вариант

В файле находится фрагмент текста неизвестной длины, содержащий слова, пробелы, знаки препинания. Посчитать, сколько раз каждая буква латинского алфавита (регистр не учитывать) встречается в тексте. Результат записать в другой файл.

12. Вариант

В файле находится фрагмент текста неизвестной длины, содержащий слова, пробелы, знаки препинания. Сформировать массив уникальных слов из этого текста, отсортировать его по алфавиту. Результат записать в другой файл.

13. Вариант

В файле записана символьная строка неизвестной длины. Среди букв, пробелов и знаков препинания встречаются числа неизвестной разрядности. Найти сумму чисел с учетом разрядности и знака. Числа и их сумму записать в другой файл. Пример строки:

```
Asdf123lkj-23.,.,.,yuyu+256mn>>n
123   -23   256   summa=356
```

14. Вариант

В файле находятся две одинаковых по размеру таблицы целых чисел.
Размер таблиц заранее неизвестен. Это рабочие матрицы. Требуется считать матрицы из файла, найти их сумму и разность. Матрицы и результаты сложения и вычитания поместить в новый файл.

15. Вариант

Создать функцию, удаляющую комментарии из файлов C и C++.
Функция принимает в качестве параметра название файла, и удаляет из файла все строки, начинающиеся с символов **//**, и все куски теста, заключенные в символы `/* ... */`.

16. Вариант

Создать функцию, которая зашифровывает фразу на основании страницы текста, хранящейся в ключевом файле.

Функция принимает в качестве параметра название файла и шифруемую фразу. Функция должна возвращать шифр – массив целых чисел (например, положение букв шифруемой фразы в текстовом файле), выбирая нужные буквы из ключевого файла.

17. Вариант

Создать функцию, которая расшифровывает фразу на основании текста, хранящегося в ключевом файле и шифра (массива целых чисел).

Функция принимает в качестве параметра название файла и шифр и возвращает расшифрованную фразу. Ключом может быть, например, положение букв фразы в ключевом файле.

18. Вариант

Создать функцию, которая транспонирует матрицы.

Матрица считывается из текстового файла, транспонируется и записывается в другой файл. Размеры матрицы заранее не известны.

19. Вариант

Создать функцию поиска подстроки в текстовом файле.

Функция принимает название входного файла и искомую подстроку. Размеры подстроки заранее не известны, подстрока вводится с консоли. Функция должна вернуть номер строки файла, где была первый раз встречена искомая строка.

20. Вариант

Создать функцию, подсчитывающую количество строк, слов и символов в текстовом файле.

Функция принимает название входного файла и печатает результаты в консоль.

21. Вариант

Создать функцию для определения частоты встречаемости символов в тексте. Функция принимает название входного и выходного файла, обрабатывает входной файл и записывает результат в выходной файл в виде:

```
. - 100, ! - 4, a - 250, b - 75 и т.д.
```

22. Вариант

Создать функцию архивации файлов. Имеется несколько файлов, необходимо слить их в один следующим образом: сначала идут названия всех файлов и их размеры, затем данные из этих файлов, разделенные, например, строками вида:

```
##########################
```

23. Вариант

Создать функцию разархивации файлов. Несколько файлов были слиты в один следующим образом: сначала идут названия всех файлов и их размеры, затем данные из этих файлов, разделенные, например, строками вида:

```
##########################
```

Надо разделить этот файл обратно на исходные файлы.

Функция принимает название входного файла.

24. Вариант

Создать функцию, обрабатывающую текстовый файл, в котором записаны арифметические выражения вида:

```
5+7/2 =
3-3*2 =
```

Подсчитать их значение и создать новый файл, вида:

```
5-7/2= 2.5
3-3*2=-3
```

Функция принимает название входного и выходного файла.

25. Вариант

Создать функцию, обрабатывающую текстовый файл. В файле записаны выражения вида:

```
sin(3.5) =
ctg(1.2) =
```

Функция должна вычислить результат каждого выражения и создать новый файл вида:

```
sin(3.5) = -0.3508
ctg(1.2) = 0.3888
```

В файле могут быть записаны только тригонометрические функции `sin()`, `cos()`, `tg()`, `ctg()`. Функция принимает имена входного и выходного файлов. Функция должна возвращать `NULL`, если файл не существует или данные записаны неправильно, т.е. если функция не может распознать записанное выражение.