Содержание

stdio.h

stdio.h (от англ. standard input/output header — стандартный заголовочный файл ввода/вывода) заголовочный файл стандартной библиотеки языка Си, содержащий определения макросов, константы и объявления функций и типов, используемых для различных операций стандартного ввода и вывода. Функциональность унаследована от «портативного пакета ввода/вывода» («portable I/O package»), написанного Майком Леском из Bell Labs в начале 1970-х. C++ ради совместимости, также использует stdio.h наряду со схожим по функциональности заголовочным файлом cstdio.

Функции, объявленные в stdio.h, являются весьма популярными благодаря тому, что являясь частью Стандартной библиотеки языка Си, они гарантируют работу на любой платформе, поддерживающей Си. Приложения на отдельных платформах могут, тем не менее, иметь причины для использования функций ввода/вывода самой платформы вместо функций stdio.h.

```
#include <stdio.h>

typedef struct FILE FILE;          /* абстракция потока ввода/вывода */

extern FILE *stdin;                /* стандартный ввод */
extern FILE *stdout;               /* стандартный вывод */
extern FILE *stderr;               /* стандартный вывод ошибок */

int printf(const char *format, ...);                    /* форматированный вывод
в stdout */
int fprintf(FILE *stream, const char *format, ...);     /* форматированный вывод
в поток */
int sprintf(char *str, const char *format, ...);        /* форматированный вывод
в буфер (без огранич.) */
int snprintf(char *str, size_t size, const char *format, ...); /*
форматированный вывод в буфер с ограничением */

int scanf(const char *format, ...);                     /* форматированный ввод
из stdin */
int fscanf(FILE *stream, const char *format, ...);      /* форматированный ввод
из потока */
int sscanf(const char *str, const char *format, ...); /* форматированный ввод
из строки */

FILE *fopen(const char *pathname, const char *mode);  /* открыть файл */
int fclose(FILE *stream);                             /* закрыть файл */
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream); /* чтение
блоков */
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream); /*
запись блоков */

int fseek(FILE *stream, long offset, int whence);     /* установить позицию в
файле */
long ftell(FILE *stream);                              /* получить текущую
```

```
позицию */
void rewind(FILE *stream);                                  /* установить позицию в
начало и сброс ошибок */

int fflush(FILE *stream);                                   /* сбросить буфер потока */

int fgetc(FILE *stream);            /* получить символ из потока (int, EOF) */
int getc(FILE *stream);             /* макро/функция, как fgetc */
int getchar(void);                  /* получить символ из stdin */
char *fgets(char *s, int size, FILE *stream); /* чтение строки в буфер */

int fputc(int c, FILE *stream);     /* записать символ в поток */
int putc(int c, FILE *stream);      /* макро/функция, как fputc */
int putchar(int c);                 /* записать символ в stdout */
int fputs(const char *s, FILE *stream); /* записать строку (без добавления '\n')
*/
int puts(const char *s);            /* записать строку и добавить перевод строки */

/* дополнительные полезные функции и макросы (не исчерпывающе) */
int feof(FILE *stream);              /* проверка конца файла */
int ferror(FILE *stream);            /* проверка флага ошибки */
void clearerr(FILE *stream);         /* сброс флагов EOF и ошибки */
int perror(const char *s);           /* вывести сообщение об ошибке на stderr */
int remove(const char *pathname);    /* удалить файл */
int rename(const char *old, const char *new); /* переименовать файл */
int setvbuf(FILE *stream, char *buf, int mode, size_t size); /* настроить
буферизацию */
FILE *tmpfile(void);                 /* создать временный файл */
char *tmpnam(char *s);               /* сгенерировать временное имя */
```

спецификаторы

- `%d, %i` - целое со знаком в десятичной системе.
- `%u` - целое без знака в десятичной системе.
- `%o` - целое без знака в восьмеричной системе.
- `%x, %X` - целое без знака в шестнадцатеричной (буквы в нижнем/верхнем регистре).
- `%f` - число с плавающей запятой в фиксированном десятичном виде.
- `%F` - то же, что %f, с возможной отличной локалью представления.
- `%e, %E` - число с плавающей запятой в экспоненциальной форме (малые/большие E).
- `%g, %G` - выбор между %f и %e/%E в зависимости от представления (короткое).
- `%a, %A` - шестнадцатеричное представление числа с плавающей запятой (C99).
- `%c` - одиночный символ (передаётся как int, выводится как unsigned char).
- `%s` - строка символов (массив char, завершающийся '\0').
- `%p` - указатель (адрес) в реалистичном форматированном виде.
- `%n` - записать в соответствующий аргумент (указатель на int/long) количество выведенных символов; поведение опасно при внешнем вводе формата.
- `%%` - литеральный символ '%' в выводе.
- `h, hh, l, ll, j, z, t, L` (модификаторы) - изменяют размер/тип сопровождающего аргумента (h/hh для short/char, l/ll для long/long long, j для intmax_t, z для size_t, t для ptrdiff_t, L для long double).

- Поле ширины (например %10d) - минимальная ширина выводимого поля; может задаваться числом или '*' (ширина из аргумента).
- Точность (например %.3f or %10.4s) - для чисел: количество знаков после точки; для строк: максимальное число выводимых символов.
- Флаги:
  1. - (левое выравнивание),
  2. + (всегда показывать знак для чисел),
  3. (пробел для положительных чисел),
  4. 0 (дополнять нулями до ширины),
  5. # (альтернативная форма: для %o обеспечивает ведущий 0, для %x/%X добавляет 0x/0X, для %f/%e/%g оставляет десятичную точку).

понятия:

- FILE — поток ввода/вывода.
- stdin, stdout, stderr — стандартные потоки.
- Буферизация — режимы: полная, построчная, безбуферная.
- Форматированные функции — управление представлением данных через спецификаторы формата.
- Блочное чтение/запись — эффективная работа с большими объёмами данных.
- Позиционирование — перемещение и получение смещения в файле.
- Флаги состояния — EOF и ошибка доступны через feof/ferror/clearerr.
- Безопасность — избегать небезопасных функций и неверного использования форматных строк.

особенности:

- Используйте snprintf и fgets вместо sprintf и gets.
- Форматные спецификаторы должны соответствовать типам аргументов.
- Учитывайте различия текстового/бинарного режимов на разных платформах.
- Перед fork выполняйте fflush для корректного управления буферами.
- При работе в многопоточном окружении синхронизируйте последовательные операции при необходимости.

пример использования:

```c
#include <stdio.h>

int main(void) {
    FILE *f = fopen("example.txt", "w");     /* открыть для записи */
    if (!f) {
        perror("fopen");                     /* сообщение об ошибке */
        return 1;
    }

    const char *s = "Пример строки\n";
    fputs(s, f);                             /* записать строку */

    fprintf(f, "Число: %d\n", 42);           /* форматированный вывод */

    fflush(f);                               /* сброс буфера */
```

```
    fclose(f);                                  /* закрыть файл */

    f = fopen("example.txt", "r");              /* открыть для чтения */
    if (!f) return 1;
    char buf[128];
    while (fgets(buf, sizeof buf, f)) {         /* читать построчно */
        fputs(buf, stdout);                     /* выводить в stdout */
    }
    fclose(f);
    return 0;
}
```