Семестр 2 (2021), занятие 7

Функция snprintf

Прототип функции `snprintf` определен в заголовочном файле `stdio.h` и имеет следующий вид.

```
int snprintf(char       *str,
             size_t      size,
             const char *format,
             ...);
```

Функция `snprintf` наряду с более распространенными в учебном процессе функциями `printf` и `fprintf` является функцией печати. В отличие от последних символы печатаются не в поток вывода, а сохраняются в массив `str` длины `size`.

Правило формирования строки формата `format` у этих функций общее. Данная строка может содержать обычные символы, которые будут просто печататься, а также спецификаторы формата печати и специальные последовательности символов. Например, последовательность `\n` является специальной последовательностью и задает переход на новую строку. Спецификаторы формата печати начинаются с символа `%`. В вызове функции `snprintf` четвертый и последующий параметры задают значения, которые должны быть напечатаны при помощи соответствующего спецификатора.

Функция `snprintf` записывает не более `size` символов. В конец записываемой последовательности символов всегда добавляется символ признака завершения строки `\0`. Возвращает функция количество символов, не считая `\0`, которое могло быть записано, если бы массив `str` имел достаточный размер.

Функция sscanf

Прототип функции `sscanf` определен в файле `stdio.h` и имеет следующий вид.

```
int sscanf(const char *str,
           const char *format,
           ...);
```

Функция `sscanf` наряду с более распространенными в учебном процессе функциями `scanf` и `fscanf` является функцией сканирования. В отличие от последних символы считываются не из потока ввода, а из строки, на которую указывает параметр `str`. Правило формирования строки формата `format` у этих функций общее. Значения, полученные в результате обработки спецификаторов ввода из форматной строки, сохраняются по адресам, которые передаются в качестве параметров вслед за параметрами `str` и `format`.

Функция `sscanf` возвращает количество успешно обработанных спецификаторов ввода, отличных от спецификатора `%n`. Если до обработки первого спецификатора ввода произошла ошибка чтения, то будет возвращено значение `EOF`, которое обычно равно `-1`.

Спецификатор ввода вида `%[...]` осуществляет считывание символов в соответствии с шаблоном, помещенным между открывающей скобкой `[` и закрывающей скобкой `]`. Считанная последовательность символов, в конец которой добавляется признак завершения конца строки `\0`, сохраняется в массив элементов типа `char`. Шаблон состоит из перечисления множества допустимых символов. Например, у спецификатора ввода `%[MW]` допустимыми будут два символа `M` и `W`. В соответствии с этим спецификатором могут быть считаны следующие последовательности `M`, `WM`, `MMWMWWW`.

В шаблоне можно использовать диапазоны. Диапазон состоит из начального символа, знака «минус» и конечного символа. Диапазон соответствует множеству символов, номера которых больше или равны номеру начального символа и меньше или равны номеру конечного символа. Например, для спецификатора ввода `%[A-Z]` допустимыми символами будут все заглавные латинские буквы, а для `%[A-Za-z]` – все латинские буквы.

В спецификаторе ввода сразу после символа `%` может быть записано число, задающее ширину поля ввода. Это число задает максимальное количество символов, которые могут быть считаны. Например, в соответствии со спецификатором ввода `%1[MW]` может быть считано не более одного символа. В соответствии со спецификатором ввода `%9[A-Za-z]` может быть считано не более девяти символов.

Пробельный символ в форматной строке может соответствовать последовательности пробельных символов в обрабатываемом тексте. Работа каждого спецификатора ввода, за исключением спецификаторов вида `%c`, `%n` и `%[...]`, начинается с пропуска всех пробельных символов.

Выявлять «лишние» пробельные символы можно с помощью спецификатора ввода `%n`.

Этот спецификатор используется для получения количества прочитанных на текущий момент символов. Данное значение сохраняется в целочисленном объекте. Напомним, что обработка спецификатора `%n` не учитывается в числе успешно обработанных спецификаторов ввода, которое возвращается функцией `sscanf`.

Пример 1.

```
const char *ss[] = {
    "",
    "abcd",
    " 1.1",
    " 1.1 1.2",
    " 1.1 1.2 abcd"
};
int    i, c;
double x, y;

for(i = 0; i < 5; i ++) {
    c = sscanf(ss[i], "%lf%lf", &x, &y);
    printf("%-5d %-30s % d\n", i, ss[i], c);
}
```

Результат.

```
0                                -1
1     abcd                        0
2      1.1                        1
3      1.1 1.2                    2
4      1.1 1.2 abcd               2
```

Пример 2.

```
const char *ss[] = {
    "",
    "abcd",
    "abcd ",
    "abcd x",
    "abcd 1.1",
    "abcd1.1",
    "abcd 1.1 1.2",
    "abcd1.1 1.2",
    "abcd 1.1 1.2 abcd",
    "abcd1.1 1.2 abcd"
};
int    i, c;
double x, y;

for(i = 0; i < 10; i ++) {
    c = sscanf(ss[i], "abcd%lf%lf", &x, &y);
    printf("%-5d %-30s % d\n", i, ss[i], c);
}
```

Результат.

```
0                                -1
1     abcd                       -1
2     abcd                       -1
3     abcd x                      0
4     abcd 1.1                    1
5     abcd1.1                     1
6     abcd 1.1 1.2                2
7     abcd1.1 1.2                 2
8     abcd 1.1 1.2 abcd           2
9     abcd1.1 1.2 abcd            2
```

Функция `isblank`

Прототип функции `isblank` определен в заголовочном файле `ctype.h` и имеет следующий вид.

```
int isblank(int c);
```

Эта функция возвращает ненулевое значение, если символ `c` является пробелом или знаком табуляции.

В дальнейшем нам понадобится следующая вспомогательная функция `skip`. Эта функция определяет число пробельных символов в начале заданной строки.

```
size_t skip(const char *txt) {
    size_t i;
    for(i = 0; isblank(txt[i]); i ++)
        ;
    return i;
}
```

Функция `fgets`

Прототип функции `fgets` определен в заголовочном файле `stdio.h` и имеет следующий вид.

```
char *fgets(char *s, int size, FILE *stream);
```

Функция `fgets` считывает не более чем `size - 1` символов (байтов) из потока `stream` и сохраняются в буфер `s`. Чтение прекращается если достигнут конец файла, или был считан символ перехода на новую строку `\n`. Если `\n` считывается, то он также сохраняется в буфер `s`. Символ признак конца строки `\0` (нулевой байт) дописывается в буфер вслед за последним считанным символом. Функция `fgets` возвращает либо `s` (успех), либо `0` (ошибка или не был считан ни один символ по причине достижения конца файла).

Пример программы использования функции `fgets`. В «бесконечном» цикле пользователю предлагается ввести строку. Программа считывает введенную строку и печатает ее длину на экран.

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

static void repl(void);

int main(void) {
    repl();
    return 0;
}

#define LEN 10

static void repl(void) {
    char buf[LEN];
```

```
    int  len;

    for(;;) {
        printf("> ");

        if(fgets(buf, LEN, stdin)) {
            len = strlen(buf);
            printf("%d\n", len);
        }
        else
            break;
    }
}
```

Функция fputs

Прототип функции `fputs` определен в заголовочном файле `stdio.h` и имеет следующий вид.

```
int fputs(const char *s, FILE *stream);
```

Функция `fputs` печатает строку символов в поток `stream` без завершающего нулевого байта `\0`. В случае успеха возвращает неотрицательное число, а в случае ошибки возвращает константу `EOF`.

Функция fgetc

Прототип функции `fgetc` определен в заголовочном файле `stdio.h` и имеет следующий вид.

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

Функция `fgetc` считывает очередной символ из потока `stream` и возвращает его в качестве значения приведенного к типу `int`. В случае конца файла или ошибки возвращается константа `EOF`.

Ниже приведен фрагмент программы, извлекающей из текстового файла вещественные числа и печатающей их на экран.

```
void readNums(FILE *fi) {
    double x;
    for(;;)
        switch(fscanf(fi, "%lf", &x)) {
            case 1:
                printf("%f\n", x);
                break;
            case EOF:
                return;
            default:
                fgetc(fi);
        }
}
```

Например, для файла со следующим содержанием

```
qwerty1.2|3.qw2.3.4.5w
1.5e-1q
```

будет напечатано

```
1.200000
3.000000
2.300000
0.400000
0.500000
0.150000
```

Функция fputc

Прототип функции `fputc` определен в заголовочном файле `stdio.h` и имеет следующий вид.

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

Функция `fputc` печатает символ `c` в поток `stream`. В случае успеха возвращает напечатанный символ, а в случае ошибки константу `EOF`.

Приведем фрагмент программы, осуществляющей копирование одного файла в другой файл с попощью функций `fgetc` и `fputc`.

```
int copyFile(FILE *fi, FILE *fo) {
    int c;
    for(;(c = fgetc(fi)) != EOF;)
        if(fputc(c, fo) == EOF)
            return -1;
    return 0;
}
```

Функции fwrite и fread

Прототип функции `fwrite` определен в заголовочном файле `stdio.h` и имеет следующий вид.

```
size_t fwrite(const void *ptr,
              size_t      size,
              size_t      nmemb,
              FILE       *stream);
```

Файл должен быть открыт с ключем `wb`. Функция `fwrite` записывает `nmemb` элементов данных, каждый из которых имеет размер `size` байтов, в поток `stream`. Записываемые данные располагаются в области памяти по адресу `ptr`.

Прототип функции `fread` определен в заголовочном файле `stdio.h` и имеет следующий вид.

```
size_t fread(void   *ptr,
             size_t  size,
             size_t  nmemb,
             FILE   *stream);
```

Файл должен быть открыт с ключем `rb`. Функция `fread` считывает `nmemb` элементов данных, каждый из которых имеет размер `size` байтов, из потока `stream`. Считанные данные сохраняются в области памяти по адресу `ptr`.

Функции `fread` и `fwrite` возвращают количество успешно считанных или записанных элементов. Функция `fread` не различает ситуации конца файла и ошибки. Для получения подобной информации нужно использовать функции `feof` и `ferror`.

Функция `nftw`

Программа получения информации о содержимом каталога файловой системы.

```
#define _XOPEN_SOURCE 500
#include <stdio.h>
#include <ftw.h>

int fn(const char        *fpath,
       const struct stat *sb,
       int                typeflag,
       struct FTW        *ftwbuf);

int main(int argc, char *argv[]) {
    if(argc != 2) {
        fprintf(stderr,
                "Usage: %s <path>\n",
                argv[0]);
        return -1;
    }
    else {
        nftw(argv[1], fn, 0, 0);
        return 0;
    }
}

int fn(const char        *fpath,
       const struct stat *sb,
       int                typeflag,
       struct FTW        *ftwbuf) {
    if(typeflag == FTW_F)
        printf("File: %s\n", fpath);

    if(typeflag == FTW_D)
        printf("Dir : %s\n", fpath);

    if(typeflag == FTW_F || typeflag == FTW_D)
        printf("    (%d) %s (%d)\n",
               ftwbuf->level,
               fpath + ftwbuf->base,
               (int)sb->st_size);

    return 0;
}
```

Предположим, что каталог `mydir` содержит файл `main.c`.

```
$ ./a.out mydir
Dir : ./mydir
    (0) mydir (4096)
File: ./mydir/main.c
    (1) main.c (214)
```

Кодировка UTF-8

Кодировка UTF-8 использует следующую структуру кодирующих последовательностей.

```
0xxxxxxx
110xxxxx 10xxxxxx
1110xxxx 10xxxxxx 10xxxxxx
...
```

Слово *Мата*, записанное латинскими буквами.

```
77 97 109 97
01001101 01100001 01101101 01100001
```

Слово *Мама*, записанное русскими буквами.

```
208 156 208 176 208 188 208 176
11010000 10011100 ...
```