Семестр 4 (2021), занятие 3

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`.

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;
    }
}
```