Роман Абраш
г. Новочеркасск
E-mail: **arv@radioliga.com**

Книга по работе с WinAVR и AVR Studio

Продолжение. Начало в №1-10/2010

stdio.h – Стандартный ввод-вывод

В силу направленности **avr-libc** на микроконтроллеры, т.е. на среду без операционной системы и любых стандартных средств ввода-вывода, имеются большие отличия в реализации функций модуля по сравнению с требованиями стандарта. Реализован только необходимый минимум функций. Имеется ряд условностей и ограничений, связанных с аппаратной и архитектурной особенностью среды.

Для максимальной компактности генерируемого кода многие возможности усечены или исключены полностью, например, касающиеся форматированного ввода-вывода функциями printf() и scanf(). Часть возможностей этих функций может быть включена или выключена путем задания определенных опций (параметров) компилятора, чтобы получать код минимального размера с необходимым минимумом функциональности.

Стандартные потоки stdin, stdout и stderr поддерживаются «виртуально», т.к. нет никакой аппаратной их поддержки, нет и никакой инициализации этих потоков; т.к. нет реально существующих файлов вообще – нет и **fopen()** и т.п. Имеется функция fdevopen(), которая позволяет связать стандартные функции ввода-вывода с реализуемыми программистом средствами поддержки конкретной аппаратуры. В качестве альтернативы можно использовать макрос fdev_setup_stream(), при помощи которого производится инициализация средств ввода-вывода. В любом случае программист самостоятельно должен реализовать функцию ввода, вывода или обе только для одного символа (нет различий между символьным и двоичным потоками), которая затем и используется для работы со стандартными потоками. Более подробно об этом будет сказано далее, при рассмотрении соответствующих функций.

Так же с целью получения наиболее компактного кода, имеется ряд функций, способных работать сразу со строковыми константами, определенными в сегменте кода программы.

Реализация библиотеки, как было сказано ранее, не делает различий между текстовыми и двоичными операциями ввода-вывода, поэтому следует быть осторожными при использовании автоматических преобразований, реализуемых функциями ввода-вывода, например, автоматической вставки символа перевода строки. Этот символ будет нарушать нормальную двоичную последовательность, поэтому для двоичных потоков надо исключить любую «автоматику» преобразований. Если же после этого потребуется выводить текст в потоки, символы перевода строки и возврата каретки следует вставлять принудительно.

Первое обращение к fdevopen() для чтения приведет к фактической связи с потоком stdin. Обращение же к fdevopen() на запись приведет к тому, что результирующий поток будет связан с stdin и stderr, т.е. фактически оба этих потока будут на самом деле одним (stdin и stderr в данном контексте – просто синонимы). Соответственно закрытие функцией fclose() любого из этих потоков фактически закроет и его синоним. Данный подход позволил существенно оптимизировать результирующий код: во-первых, за счет исключения дублирования структур данных потоков; во-вторых, нет необходимости в функции fprintf(); в-третьих, можно отказаться от передачи через стек лишнего параметра (указателя на используемый поток). В сущности, это оправдано особенностями архитектуры микроконтроллера.

Имеется возможность связать с потоком дополнительные пользовательские данные при помощи fdev_set_udata(), которые затем могут быть извлечены в реализациях функций ввода-вывода символа при помощи fdev_get_udata() и обработаны, как требуется. Это, например, позволит реализовать одну функцию ввода-вывода, работающую с различными UART. В этом случае функции ввода-вывода сохраняют контекст различных UART в области данных пользователя между своими вызовами.

По умолчанию fdevopen() подразумевает использование malloc(), что часто очень нежелательно в ограниченных ресурсах микроконтроллера. Поэтому в библиотеке реализован опциональный вариант fdevopen() без использования malloc(). Макрос fdev_setup_stream() позволяет настроить стандартные функции ввода-вывода для работы с подготовленным заранее буфером файла. Пример ***см. на врезке***.

Врезка. Пример:

```
#include <stdio.h>   // подключение модуля ввода-вывода

static int uart_putchar(char c, FILE *stream);     // определяемая пользователем
                                                   // функция вывода символа

// подготовка при помощи макроса структуры «файла» для ввода-вывода
static FILE mystdout = FDEV_SETUP_STREAM(uart_putchar, NULL, _FDEV_SETUP_WRITE);

// реализация функции вывода символа
static int uart_putchar(char c, FILE *stream){
// принудительная вставка символа «возврат каретки» после символа
// «перевод строки» для нормального отображения в терминальной программе
if (c == '\n')
uart_putchar('\r', stream);

loop_until_bit_is_set(UCSRA, UDRE); // ожидание готовности UART

UDR = c; // вывод символа

return 0;
}

// основная функция
int main(void){
init_uart(); // инициализация UART
stdout = &mystdout; // связывание потока stdout с подготовленным «файлом»
printf(«Hello, world!\n»); // использование стандартной функции вывода
return 0;
}
```

Приведенный пример выводит в стандартный поток, связанный с аппаратно-реализованным UART микроконтроллера, текстовое сообщение.

В примере применена инициализация потока **mystdout** макросом FDEV_SETUP_STREAM() вместо макроса fdev_setup_stream(), поэтому вся инициализация потока происходит в момент инициализации переменных С-программы, т.е. совершенно прозрачно для пользователя.

Если инициализированный таким образом поток более не нужен, он может быть разрушен макросом fdev_close(). Не требуется использование fclose(), т.к. это потребует от компоновщика подключения модуля stdlib.h – Стандартные возможности для «освобождения» памяти, что в данном случае совершенно лишнее.

Определения модуля

В модуле определены следующие константы и макросы:

FILE

FILE – определение структуры «файл», т.е. структуры, передаваемой в различные функции ввода-вывода в качестве идентификатора файла.

stdin

stdin – поток, используемый упрощенными (т.е. не требующими параметра stream) функциями ввода.

stdout

stdout – поток, используемый упрощенными (т.е. не требующими параметра stream) функциями вывода.

stderr

stderr – поток, используемый для вывода сообщений об ошибках. По умолчанию соответствует **stdout**. Если необходимо направить сообщения об ошибках в другой, отличный от **stdout**, поток, необходимо присвоить ему другое значение, возвращенное **fdevopen()**, *не закрывая* имеющееся (т.к. иначе это привело бы к закрытию **stdout**).

EOF

EOF – константа «конец файла». Это значение возвращается функциями ввода в качестве индикатора ошибки. Так как в текущей реализации для AVR не существует реального понятия «файл», иное использование этой константы – бессмысленно.

fdev_set_udata()

fdev_set_udata(stream, u) – макрос вставки пользовательских данных (указатель) **u** во внутреннюю структуру файла **stream**. Использование этих данных возможно в функциях **get()** и **put()**.

fdev_get_udata()

fdev_get_udata(stream) – макрос возврата пользовательских данных (указателя) из внутренней структуры файла **stream**.

fdev_setup_stream()

fdev_setup_stream(stream, put, get, rwflag) – макрос, настраивающий буфер для использования стандартными функциями ввода-вывода.

Аргумент **stream** типа FILE должен быть заранее подготовлен пользователем, в отличие от аналогичной структуры, выделяемой fdevopen() динамически.

Аргументы **put** и **get** аналогичны передаваемым в fdevopen().

Аргумент **rwflag** – это флаги-признаки режима открытия потока. Может принимать одно из трех значений:

- **_FDEV_SETUP_READ** – для чтения
- **_FDEV_SETUP_WRITE** – для записи
- **_FDEV_SETUP_RW** – записи и чтения

_FDEV_ERR

_FDEV_ERR – константа ошибки чтения потока. Используется в функции чтения fdevopen().

_FDEV_EOF

_FDEV_EOF – константа ошибки при попытке чтения за концом потока. Используется в функции чтения fdevopen().

FDEV_SETUP_STREAM()

FDEV_SETUP_STREAM(put, get, rwflag) – макрос, аналогичный fdev_setup_stream(), только инициализирующий переменную потока путем присвоения ей «возвращаемого» значения макроса. Параметры макроса аналогичны параметрам fdev_setup_stream().

fdev_close()

fdev_close() – макрос, освобождающий любые ресурсы, связанные с открытым потоком. В текущей реализации он не делает ничего, но введен для совместимости с будущими реализациями.

putc()

putc(c, stream) – макрос «упрощенного» вывода в поток, полностью аналогичный (и являющийся синонимом) fputc().

putchar()

putchar(c) – макрос вывода символа в поток **stdout**.

getc()

getc(stream) – макрос «упрощенного» ввода из поток, полностью аналогичный (и являющийся синонимом) fgetc().

getchar()

getchar() – макрос, возвращающий символ, считанный из потока **stdin**.

Спецификация строки формата для функций форматированного вывода

Строка формата ввода-вывода состоит из последовательностей символов двух типов: обычных, которые копируются в поток непосредственно, и последовательностей символов-директив (или управляющих символов), которые в процессе обработки заменяются другими в соответствии с определенными правилами. Формат управляющей последовательности следующий (в [квадратных скобках] указаны элементы, которые могут отсутствовать, жирным выделены элементы-модификаторы, прочие символы – обязательные):

%[**префикс**][**ширина**][.**точность**][**суффикс**]**тип**

Префикс определяет общие характеристики преобразования формата. Список префиксов (может быть одновременно несколько):

– использовать *альтернативный* формат. Для типов **c**, **d**, **i**, и **s** игнорируется. Для типа **o** означает, что точность представления числа должна быть увеличена так, чтобы первый выводимый символ был нулем. Для типов **x** или **X** означает, что выводимое число будет предваряться символами "0x" или "0X" соответственно.

0 – выравнивание нулями. Означает, что если задана минимальная ширина представления числа, то все незначащие левые символы будут заменены нулями (вместо пробелов). Для типов **d**, **i**, **o**, **u**, **i**, **x**, и **X** данный модификатор игнорируется.

- – флаг отрицательного значения. Означает, что выводимое значение выравнивается влево в пределах заданной ширины, дополняясь справа пробелами. Если применяются оба модификатора **0** и **-**, последний имеет приоритет.

" " (пробел) – означает, что положительное число должно предваряться слева пробелом (т.е. резервирует место под возможный символ минуса для отрицательных чисел). Применяется для типов **d** или **i**.

+ – означает, что положительные числа всегда должны предваряться знаком плюс. Если использованы оба модификатора **+** и пробел, первый имеет приоритет.

Ширина – это десятичное число, которое определяет минимальное количество символов, требуемое для представления числа. Если число может быть представлено меньшим количеством символов – оно дополняется слева или справа (в зависимости от префикса) пробелами (или нулями – см. префикс **0**). Если для представления числа необходимо больше символов, то значение ширины игнорируется. Т.е. если указана ширина 3 и задан префикс 0, то число 1 будет выведено как "001".

Точность – это десятичное число, определяющее символов в представлении для достижения заданной точности. Для строковых аргументов определяет максимальную длину строки, для числовых – минимальное кол-во значащих цифр. Пару ширина-точность для чисел с плавающей точкой можно рассматривать как пару «общее кол-во знаков»-«кол-во знаков после точки».

Суффикс – это символ **h** или **l**. **h** – игнорируется, а **l** означает, что аргумент имеет тип **long**, а не **int**.

Тип – это символ, который определяет тип очередного элемента в списке выводимых аргументов. Определены следующие типы (должен указываться любой из перечисленных):

diouxX – целое число. Аргумент рассматривается, как целое число, и выводится в десятичном со знаком(***di***), десятичном без знака (***u***), восьмеричном (***o***) или шестнадцатеричном (***xX***) формате. Тип ***X*** означает, что шестнадцатеричные цифры-буквы должны выводиться в верхнем регистре (т.е. число **0x0a1b2c3d** будет выведено как "***A1B2C3D***"). Если задана точность, то представление числа дополняется слева нулями для ее достижения.

p – указатель. Аргумент трактуется, как **void *** и выводится, как обычное десятичное число в шестнадцатеричном формате, т.е. последовательность ***%p*** полностью аналогична управляющей последовательности ***%#x***

c – символ. Аргумент трактуется как **unsigned char** и выводится соответствующий символ.

s – строка. Аргумент трактуется как указатель на строку. В поток выводятся все символы вплоть до завершающего строку символа **NULL** (который не выводится). Если задана точность – выводится не более указанного количества символов, при этом наличие символа **NULL** в строке не требуется.

S – строка в сегменте кода. Аргумент трактуется как указатель на строку, размещенную в сегменте кода (см. тип ***s***), в отличие от стандартного размещения в ОЗУ.

% – символ **%**. Никаких преобразований не требуется, очередной аргумент так же не требуется – это просто вывод символа **%** в поток (т.е. управляющая последовательность **%%** предназначена для вывода в поток одного символа **%**).

eE – экспоненциальная форма записи числа с плавающей точкой. Аргумент типа **double** при необходимости округляется и выводится в формате ***[-]d.ddde±dd***, где ***d*** –

цифра. Число цифр после точки определяется точностью (если не задана – принимается равной 6), если точность равна 0, то дробная часть не выводится и десятичная точка так же не выводится. Если тип **E**, то в экспоненциальной форме используется этот символ (взамен **e**). Показатель степени – всегда 2 знака.

fF – простая форма записи числа с плавающей точкой. Аргумент типа **double** при необходимости округляется и выводится в формате ***[-]ddd.ddd***, где **d** – цифра. Число цифр после точки определяется точностью (если не задана – принимается равной 6), если точность равна 0, то дробная часть не выводится и десятичная точка так же не выводится. Если выводится точка, то минимум одна цифра всегда выводится перед ней.

gG – версия ***fF*** или ***eE*** (соответственно с учетом регистра) с некоторыми особенностями. Число цифр после точки определяется точностью (если не задана – принимается равной 6), если точность равна 0, то она принимается равной 1. выбор между вариантами ***fF*** и ***eE*** осуществляется автоматически: форма ***fF*** используется, если число не более 1000000, или показатель степени 10 для числа меньше или равен значению точности.

Примечания.

1. Ни при каких условиях не происходит усечения точности результата для вывода чисел: если заданная ширина недостаточна для представления числа – она игнорируется и выводится необходимое число знаков.

2. Для ширины и точности имеется ограничение – они не могут быть более 255.

3. Если использована не полная версия функции vprintf() при сборке проекта, а в списке аргументов присутствует число типа double, в потоке вывода на этом месте будет символ «?», т.е. краха не произойдет.

4. Так как суффикс h игнорируется (по стандарту он предназначен для принудительного приведения к типу char), эффективность кода повысится, если никогда его не использовать в строке формата.

5. Суффикс l приведет к прекращению вывода в поток. Т.к. текущая реализация не поддерживает тип long long.

6. Переменная ширина или точность не реализованы, поэтому символ * в поле ширины или точности приведет к прекращению вывода в поток.

Спецификация строки формата для функций форматированного ввода

Строка формата для функций ввода имеет строение, аналогичное строке формата вывода (см. Спецификация строки формата для функций форматированного вывода), но состоит из следующих частей:

%[**префикс**][**размер**] [**тип**]

Префикс – это одна или более из числа следующих последовательностей:

***** – означает, что преобразование должно быть выполнено, но результат должен быть проигнорирован, т.е. не должен помещаться в элемент списка аргументов

h – означает, что аргумент в списке имеет тип **short int** (а не **int**)

hh – означает, что аргумент имеет тип **char** (а не **int**)

l – означает, что аргумент имеет тип **long int** (а не **int**) для целочисленного ввода или что аргумент имеет тип **double** для ввода чисел с плавающей точкой.

Размер – это десятичное число от 1 до 255, которое показывает, какое минимальное количество символов потока должно быть считано для интерпретации соответствующего значения.

Тип – это один из следующих символов:

% – означает, что должен быть введен символ процента, преобразование не осуществляется

d – означает, что должно быть введено целое (возможно, со знаком) число в десятичном представлении, т.е. очередной аргумент в списке имеет тип **int**

i – означает, что должно быть введено целое (возможно со знаком) число в десятичном, восьмеричном или шестнадцатеричном формате, аргумент имеет тип **int**. Формат числа определяется по первым символам: если число начинается с **0x** или **0X** - принимается шестнадцатеричный формат, если число начинается с **0** (????) – восьмеричный, иначе принимается десятичный формат.

o – означает, что вводится восьмеричное целое число (аргумент имеет тип **unsigned int**)

u – означает, что вводится беззнаковое целое число, аргумент имеет тип **unsigned int**

x – означает, что вводится число в шестнадцатеричном представлении, аргумент имеет тип **unsigned int**

e или ***E***, или ***f***, или ***F***, или ***g***, или ***G*** – означает, что вводится число в формате с плавающей точкой, аргумент имеет тип **float**

s – означает, что вводится последовательность символов, не содержащая разрывов, аргумент представляет собой указатель на буфер достаточной длины, чтобы вместить всю последовательность вместе с завершающим символом **NULL**. Ввод строки прекращается, как только встретится любой символ разрыва строки, например, пробел, табуляция и т.п.

c – означает, что должна быть введена последовательность символов, количество которых определено полем размер. Аргумент, как и для ***s***, должен указывать на буфер достаточной длины, однако завершающий **NULL** не вводится. Вводятся все символы, включая разделительные. Чтобы пропустить начальные пробелы используйте принудительно вставленный в строку формата пробел перед управляющей последовательностью.

p – означает, что вводится значение указателя. Ожидается, что последовательность символов будет соответствовать той, что выводится в аналогичном случае функцией **vfprintf()**.

n – не вводится ничего, вместо этого в аргумент (типа int) заносится количество обработанных к настоящему моменту символов потока. Никаких преобразований не выполняется. Это действие может быть пропущено при использовании префикса *.

[множество] – означает, что должна быть введена последовательность, состоящая из символов, указанных между скобками ***[]***. Аргумент должен указывать на буфер достаточного объема. Чтобы вместить все символы, включая автоматически добавляемый **NULL** в конце. Множество формируется путем простого перечисления символов в виде строки, однако, есть особенности:

1. Если первый символ за открывающей скобкой «^» (птичка), то остальные символы образуют множество символов, исключаемых из числа вводимых.

2. В множество не включается символ закрывающей квадратной скобки. Чтобы все же добавить его в множество, следует поместить его сразу после открывающей скобкой или после «птички». В любом ином случае эта скобка обозначит конец множества.

3. Чтобы добавить в множество набор подряд идущих символов, следует указать первый символ, затем через дефис – последний.

Примеры использования множеств:

[[0-9] – ввод последовательности, состоящей из символов «0», «1», «2», «3», «4», «5», «6», «7», «8», «9» «[«

[^0-9] – ввод последовательности из любых символов, кроме цифр

[][] – ввод последовательности, состоящей из символов квадратных скобок

Функции модуля

fdevopen()

Определение:

FILE * fdevopen (int(*put)(char, FILE *), int(*get)(FILE *))

Параметры:

int(***put***)(char, FILE *) – указатель на функцию вывода символа в поток (если имеется намерение использовать создаваемый поток для вывода). Эта функция имеет два параметра: выводимый символ и указатель на открытый поток. Эта функция должна возвращать 0 в случае удачного вывода, и не ноль в случае ошибки.

int(***get***)(FILE *) – указатель на функцию для ввода символа (если имеется намерение использовать поток для ввода). Эта функция имеет один параметр – указатель на открытый поток, и должна возвратить введенный символ в случае успешного завершения. В случае попытки считать «за концом данных» функция должна вернуть _FDEV_EOF, в случае иных ошибок функция должна возвращать _FDEV_ERR.

Возвращаемое значение: в случае успеха – указатель на структуру открытого потока в динамической памяти, иначе **NULL**.

Описание: Эта функция заменяет **fopen()**. Она позволяет открыть поток для случая, когда необходимо реализовать собственную поддержку аппаратной части для ввода-вывода. В случае успешного завершения функция возвращает указатель на

структуру **FILE**, которая затем может использоваться, как соответствующий поток в других функциях. Если недостаточно динамической памяти, или не определена ни одна из функций **put** и **get** (т.е. попытка открыть поток вообще без намерения его использовать) – функция завершается неудачно.

Первый поток, открытый для ввода автоматически ассоциируется с потоком stdin, первый поток, открытый для вывода, автоматически ассоциируется с потоками stdout и stderr.

Примечания:

1. Функция использует динамическое выделение памяти при помощи calloc() (и/или malloc()).

2. Для совместимости с кодом, написанным для avr-libc версии 1.2 или более ранней, следует определить __STDIO_FDEVOPEN_COMPAT_12 до подключения модуля **stdio.h**. Это делается исключительно для обеспечения совместимости и упрощения миграции кода, в новых проектах не используйте это!

fclose()

Определение:

int fclose (FILE *stream)

Параметры:

FILE ***stream** – указатель на ранее открытый поток

Возвращаемое значение: всегда 0

Описание: функция закрывает поток, открытый функцией fdevopen() и тем самым делает невозможной любой ввод-вывод через него.

Примечания: эта функция не должна использоваться для потоков, открытых с помощью fdev_setup_stream() или FDEV_SETUP_STREAM().

vfprintf()

Определение:

int vfprintf (FILE *stream, const char *fmt, va_list ap)

Параметры:

FILE ***stream** – указатель на открытый для вывода поток

const char ***fmt** – указатель на строку формата

va_list **ap** – указатель на список выводимых аргументов

Возвращаемое значение: число выведенных в поток символов в случае успешного завершения или EOF в противном случае (это может быть, если поток не открыт для вывода).

Описание: это базовая функция форматированного файлового вывода для всего семейства **printf**-функций. Она реализует символьный вывод в поток **stream** списка аргументов, заданного как **ap** в соответствии с форматом, заданным строкой **fmt** (см. Спецификация строки формата для функций форматированного вывода).

Примечания: полная реализация всех возможностей форматирования требует больших затрат программной памяти. С целью минимизации ресурсов существует 3 варианта реализации функции **vfprintf()**, выбираемых компоновщиком во время сборки проекта:

- *обычная* (по умолчанию) – реализует все виды преобразований, кроме предназначенных для чисел с плавающей точкой;

- *усеченная* – реализует только базовые преобразования форматов целых чисел и строк, кроме того, может применяться только префикс **#** (см. Спецификация строки формата для функций форматированного вывода);

- *полная* – реализующая полную спецификацию форматов.

Усеченная версия выбирается следующими параметрами компилятора:

-Wl,-u,vfprintf -lprintf_min

Полная версия выбирается следующими параметрами компилятора:

-Wl,-u,vfprintf -lprintf_flt -lm

vfprintf_P()

Определение:

int vfprintf_P (FILE *stream, const char *fmt, va_list ap)

Параметры:

FILE ***stream** – указатель на открытый для вывода поток

const char ***fmt** – указатель на строку формата в сегменте кода

va_list **ap** – указатель на список выводимых аргументов

Возвращаемое значение: число выведенных в поток символов в случае успешного завершения или EOF в противном случае (это может быть, если поток не открыт для вывода).

Описание: это версия функции vprintf(), которая использует строку формата, размещаемую не в ОЗУ, а в сегменте кода программы.

fputc()

Определение:

int fputc (int c, FILE *stream)

Параметры:

int c – символ для вывода

FILE *stream – открытый для вывода поток.

Возвращаемое значение: в случае успеха возвращает выведенный символ, в противном случае возвращает EOF.

Описание: Функция выводит в поток символ (1 байт, хотя параметр имеет тип int).

Примечания:

printf()

Определение:

int printf (const char *fmt,...)

Описание: данная функция осуществляет форматированный вывод в поток stderr. Параметры и подробности см. в описании **vfprintf()**.

printf_P()

Определение:

int printf_P (const char *fmt,...)

Описание: данная функция осуществляет форматированный вывод в поток stderr, используя строку формата в сегменте кода. Параметры и подробности см. в описании **vfprintf_P()**.

vprintf()

Определение:

int vprintf (const char *fmt, va_list ap)

Описание: данная функция осуществляет форматированный вывод в поток stdout. Параметры и подробности см. в описании **vfprintf()**.

sprintf()

Определение:

int sprintf (char *s, const char *fmt,...)

Описание: данная функция осуществляет форматированный вывод в строку **s** (область памяти для строки-результата должна иметь достаточный размер!). Подробности об остальных параметрах функции и ее особенностях см. в описании **vfprintf()**.

sprintf_P()

Определение:

int sprintf_P (char *s, const char *fmt,...)

Описание: данная функция осуществляет форматированный вывод в строку **s** (область памяти для строки-результата должна иметь достаточный размер!), используя строку формата в сегменте кода. Подробности об остальных параметрах функции и ее особенностях см. в описании **vfprintf_P()**.

snprintf()

Определение:

int snprintf (char *s, size_t n, const char *fmt,...)

Описание: вариант функции **sprintf()** с контролем размера строки. Параметр n определяет максимальное количество символов (включая завершающий NULL), которое будет помещено в строку **s**. В остальном (по параметрам и поведению) полностью аналогична функции **sprintf()**.

snprintf_P()

Определение:

int snprintf_P (char *s, size_t n, const char *fmt,...)

Описание: вариант функции **sprintf_P()** с контролем размера строки. Параметр n определяет максимальное количество символов (включая завершающий NULL), которое будет помещено в строку **s**. В остальном (по параметрам и поведению) полностью аналогична функции **sprintf_P()**.

vsprintf()

Определение:

int vsprintf (char *s, const char *fmt, va_list ap)

Описание: аналог функции **snprintf()**, но со списком аргументов, передаваемых переменной **ap**.

vsprintf_P()

Определение:

int vsprintf_P (char *s, const char *fmt, va_list ap)

Описание: вариант **vsprintf()**, но для строки формата в сегменте кода.

vsnprintf()

Определение:

int vsnprintf (char *s, size_t n, const char *fmt, va_list ap)

Описание: вариант **snprintf()** с контролем количества выводимых символов, как в **snprintf()** и списком аргументов, передаваемых переменной **ap**.

vsnprintf_P()
Определение:
int vsnprintf_P (char *s, size_t n, const char *fmt, va_list ap)
Описание: вариант **snprintf()** с контролем количества выводимых символов, как в **snprintf()** и списком аргументов, передаваемых переменной **ap**. Строка формата размещается в сегменте кода.

fprintf()
Определение:
int fprintf (FILE *stream, const char *fmt,...)
Описание: функция осуществляет форматированный вывод в поток **stream**. См. подробности об особенностях и параметрах в **vfprintf()**.

fprintf_P()
Определение:
int fprintf_P (FILE *stream, const char *fmt,...)
Описание: функция осуществляет форматированный вывод в поток **stream**, причем строка формата размещается в сегменте кода. См. подробности об особенностях и параметрах в **vfprintf()**.

fputs()
Определение:
int fputs (const char *str, FILE *stream)
Параметры:
const char ***str –** указатель на строку
FILE ***stream** – указатель на открытый для вывода поток
Возвращаемое значение: возвращает 0 в случае успешного завершения, в противном случае возвращает EOF.
Описание: Функция выводит в поток **stream** строку **str**. Завершающий символ NULL не выводится.

fputs_P()
Определение:
int fputs_P (const char *str, FILE *stream)
Описание: вариант **fputs()** для вывода строки из сегмента кода.

puts()
Определение:
int puts (const char *str)
Параметры:
const char ***str** – указатель на строку
Возвращаемое значение: возвращает 0 в случае успешного завершения, в противном случае возвращает EOF.
Описание: функция выводит в поток stdout строку **str**, дополняя ее символом перевода строки.

puts_P()
Определение:
int puts_P (const char *str)
const char ***str** – указатель на строку
Возвращаемое значение: возвращает 0 в случае успешного завершения, в противном случае возвращает EOF.
Описание: функция выводит в поток stdout строку **str** из сегмента кода, дополняя ее символом перевода строки.

fwrite()
Определение:
size_t fwrite (const void *ptr, size_t size, size_t nmemb, FILE *stream)
Параметры:
const void ***ptr** – указатель на первый выводимый элемент
size_t **size** – размер выводимого элемента
size_t **nmemb** – число выводимых элементов
FILE ***stream** – указатель на поток, открытый для записи
Возвращаемое значение: в случае успеха возвращает значение **nmemb**
Описание: функция выводит в поток **stream nmemb** элементов, каждый из которых имеет размер **size** байт. Указатель **ptr** должен указывать на первый выводимый элемент.

fgetc()
Определение:
int fgetc (FILE *stream)
Параметры:
FILE ***stream** – указатель на поток, открытый для чтения
Возвращаемое значение: введенный символ или EOF в случае ошибки
Описание: функция считывает символ (байт) из потока **stream**. В случае ошибки или попытки чтения «за концом потока» возвращается значение EOF, поэтому следует использовать функции **feof()** и **ferror()** для определения истинной причины ошибки.

ungetc()
Определение:
int ungetc (int c, FILE *stream)
Параметры:
int **c** – символ
FILE ***stream** – указатель на поток, открытый для чтения
Возвращаемое значение: в случае успеха возвращает символ c, в противном случае возвращает **EOF**.
Описание: функция «заталкивает» в поток, открытый для чтения, один символ c (преобразуемый в **unsigned char**), заставляя его стать следующим, считываемым при вводе. Функция возвращает этот самый символ, если не было ошибки.
Примечания: если «заталкиваемый» символ равен EOF, функция завершается с ошибкой, а поток остается неизмененным.

fgets()
Определение:
char * fgets (char *str, int size, FILE *stream)
Параметры:
char ***str** – указатель на строку-результат
int **size** – количество вводимых байт
FILE ***stream** – указатель на поток, открытый для чтения
Возвращаемое значение: в случае успешного завершения возвращает указатель на строку-результат, в противном случае возвращает **NULL**.
Описание: функция считывает из потока **stream** не более **size**-1 байтов, размещая их в строке-результате **str**. Считывание символов продолжается до тех пор, пока не встретится символ перевода строки (который в результат не заносится). Если не было ошибок, строка-результат завершается символом **NULL**, а функция возвращает указатель на строку-результат. В случае ошибок возвращается **NULL**, и в потоке устанавливается признак ошибки, чтобы можно было использовать функцию **ferror()**.

gets()
Определение:
char * gets (char *str)
Параметры:
char ***str** – строка-результат
Возвращаемое значение: в случае успешного завершения возвращает указатель на строку-результат, в противном случае возвращает **NULL**
Описание: функция действует аналогично **fgets()**, только строка читается из потока stdin и количество считанных байт не контролируется.
Примечания: отсутствие контроля количества вводимых символов налагает большую ответственность на программиста.

fread()
Определение:
size_t fread (void *ptr, size_t size, size_t nmemb, FILE *stream)
Параметры:
void ***ptr** – указатель на буфер-приемник
size_t **size** – размер каждого элемента
size_t **nmemb** – количество элементов
FILE ***stream** – указатель на поток. Открытый для чтения
Возвращаемое значение: в случае успешного завершения возвращает значение nmemb
Описание: функция считывает из потока **stream nmemb** элементов, каждый из которых имеет размер **size** байт и помещает их в буфер **ptr**.
Примечания: буфер должен иметь достаточный объем!

clearerr()
Определение:
void clearerr (FILE *stream)
Параметры:
FILE ***stream** – указатель на поток
Возвращаемое значение: нет
Описание: функция сбрасывает признаки любых ошибок потока **stream**.

feof()
Определение:
int feof (FILE *stream)
Параметры:
FILE ***stream** – указатель на поток
Возвращаемое значение: не нулевое значение, если достигнут конец потока **stream**

Описание: функция тестирует состояние флага EOF для указанного потока.

Примечания: функция не сбрасывает тестируемый флаг, для этого следует использовать **clearerr()**.

ferror()

Определение:

int ferror (FILE *stream)

Параметры:

FILE ***stream** – указатель на поток

Возвращаемое значение: не нулевое значение, если установлен флаг ошибки в потоке **stream**

Описание: функция тестирует состояние флага ошибки для указанного потока.

Примечания: функция не сбрасывает тестируемый флаг, для этого следует использовать **clearerr()**.

vfscanf()

Определение:

int vfscanf (FILE *stream, const char *fmt, va_list ap)

Параметры:

FILE ***stream** – указатель на поток, открытый для чтения

const char ***fmt** – указатель на строку-спецификатор формата

va_list **ap** – список вводимых аргументов

Возвращаемое значение: функция возвращает количество введенных элементов списка аргумента в случае успешного завершения или EOF в противном случае.

Описание: это базовая функция форматированного ввода для всего семейства **scanf**-функций. Функция считывает символы из потока **stream** и преобразовывает их в соответствии с заданным форматом **fmt**, помещая результирующие значения в аргументы из списка **ap**. Все символы, не попадающие под спецификацию управляющей последовательности строки формата (см. Спецификация строки формата для функций форматированного ввода) считаются текстом и проверяются на совпадение с соответствующими символами строки формата непосредственно. При этом пробел в строке формата считается совпадающим с любым символом «пустого места» в потоке, прочие же должны совпадать абсолютно. Если обнаруживается несовпадение символов, либо достигается конец потока – функция завершается по ошибке.

Функция возвращает количество считанных элементов в списке аргументов, причем 0 так же допустим. Ноль возвращается, если поступавшие символы не могли быть интерпретированы в соответствии с заданным форматом. Функция возвращает EOF только в том случае, если конец потока был достигнут прежде, чем завершено последнее преобразование формата.

Примечания: полная реализация всех возможностей форматирования ввода требует больших затрат программной памяти. С целью минимизации ресурсов существует 3 варианта реализации функции **vfscanf()**, выбираемых компоновщиком во время сборки проекта:

- *обычная* (по умолчанию) – реализует все виды преобразований, кроме предназначенных для чисел с плавающей точкой;

- *усеченная* – реализует только базовые преобразования форматов целых чисел и строк, кроме того, исключается управляющая последовательность **%[** (см. Спецификация строки формата для функций форматированного ввода);

- *полная* – реализующая полную спецификацию форматов, кроме того, снимающая ограничение на размер в 255 символов (размер ограничивается 65535).

Усеченная версия выбирается следующими параметрами компилятора:

-Wl,-u,vfscanf -lscanf_min –lm

Полная версия выбирается следующими параметрами компилятора:

-Wl,-u,vfscanf -lscanf_flt -lm

vfscanf_P()

Определение:

int vfscanf_P (FILE *stream, const char *fmt, va_list ap)

Описание: вариант функции vfscanf(), использующий строку формата в сегменте кода.

fscanf()

Определение:

int fscanf (FILE *stream, const char *fmt,...)

Описание: вариант функции vfscanf(), использующий прямо указанный список вводимых аргументов.

fscanf_P()

Определение:

int fscanf_P (FILE *stream, const char *fmt,...)

Описание: вариант функции **fscanf()**, использующий строку формата в сегменте кода.

scanf()

Определение:

int scanf (const char *fmt,...)

Описание: вариант **fscanf()** для ввода из потока **stdin**.

scanf_P()

Определение:

int scanf_P (const char *fmt,...)

Описание: вариант **scanf()**, использующий строку формата в сегменте кода.

vscanf()

Определение:

int vscanf (const char *fmt, va_list ap)

Описание: вариант **vfscanf()** для ввода из потока **stdin**.

sscanf()

Определение:

int sscanf (const char *buf, const char *fmt,...)

Параметры:

const char ***buf** – буфер для обработки

const char ***fmt** – строка формата

... – список вводимых аргументов

Описание: функция полностью аналогичная **vfscanf()**, с той лишь разницей, что вместо чтения из потока берутся символы из буфера **buf**.

sscanf_P()

Определение:

int sscanf_P (const char *buf, const char *fmt,...)

Описание: вариант **sscanf()**, использующий строку формата в сегменте кода.

fflush()

Определение:

int fflush (FILE *stream)

Параметры:

FILE ***stream** – указатель на поток

Возвращаемое значение:

Описание: вызывает немедленный «сброс» буферов чтения-записи для указанного потока **stream**.

Примечание: эта функция введена только для совместимости, т.к. никакой буферизации ввода-вывода не реализовано. Функция не делает ничего

Продолжение в №12/2010