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

Книга по работе с WinAVR и AVR Studio

Продолжение. Начало в №1-11/2010

stdlib.h – Стандартные возможности

Этот модуль определяет базовые макросы и функции, требуемые по стандарту ISO C, а также некоторые специфичные для AVR расширения стандарта.

Определения модуля

В модуле определены следующие константы:

RANDOM_MAX – наибольшее случайное число, генерируемое функцией random()

DTOSTR_ALWAYS_SIGN – флаг для функции dtostre()

DTOSTR_PLUS_SIGN – флаг для функции dtostre()

DTOSTR_UPPERCASE – флаг для функции dtostre()

RAND_MAX – наибольшее случайное число, генерируемое функцией rand()

Введены описания следующих структур и типов:

div_t – возвращаемое значение функции div()

ldiv_t – возвращаемое значение функции ldiv()

typedef int(*) **__compar_fn_t (const void *, const void *)** – определение указателя на функцию, используемую при сортировке qsort(), введен лишь для удобства. Эта функция в качестве параметров получает два указателя, сравнивает объекты, на которые эти указатели указывают, и возвращает число меньше нуля, ноль или число больше нуля соответственно для случаев, когда первый объект меньше, равен или больше второго.

Обе структуры **div_t** и **ldiv_t** содержат по 2 поля данных: **quot** – частное и **rem** – остаток, только для первой структуры оба этих поля имеют тип **int**, а для второй – **long**.

Кроме того, определены глобальные переменные, позволяющие произвести «тонкую настройку» поведения менеджера динамической памяти (см. *Динамическое распределение памяти*).

size_t **__malloc_margin**
char * **__malloc_heap_start**
char * **__malloc_heap_end**

Стандартные функции

abort()
Определение:
void abort (void)
Параметры: нет.
Возвращаемое значение: нет.
Описание: функция вызывает аварийное завершение программы. В реализации для AVR это приводит к запрещению прерываний и вызову функции **exit(1)** или, что то же самое, переход к бесконечному пустому циклу. Это равносильно фактическому прекращению работы микроконтроллера.
Примечания: состояние портов микроконтроллера и прочей периферии не изменяется.

abs()
Определение:
int abs (int i)
Описание: функция возвращает абсолютное значение аргумента i, т.е. его модуль.
Примечание: данная функция – встроенная в GCC.

labs()
Определение:
long labs (long i)
Описание: функция возвращает абсолютное значение аргумента i, т.е. его модуль.
Примечание: данная функция – встроенная в GCC.

bsearch()
Определение:
void * bsearch (const void *key, const void *base, size_t nmemb, size_t size, int(*compar)(const void *, const void *))
Параметры:
const void ***key** – указатель на ключевой элемент поиска
const void ***base** – указатель на начало области поиска
size_t **nmemb** – количество просматриваемых элементов
size_t size – размер каждого элемента
int(*compar)(const void *, const void *) – указатель на функцию сравнения элементов
Возвращаемое значение: функция возвращает указатель на найденный элемент либо **NULL**, если ничего не найдено. Если имеется несколько одинаковых элементов, то будет возвращен указатель на один из них (какой именно – не определено стандартом[32]).
Описание: функция выполняет поиск элемента **key** среди последовательности из **nmemb** подобных элементов **base**, каждый из которых имеет размер **size**. Поиск осуществляется путем сравнения **key** с каждым из элементов в последовательности при помощи функции **compare**(), которая возвращает число меньше нуля, равное нулю или большее нуля, соответствующие результатам сравнения МЕНЬШЕ, ЭКВИВАЛЕНТНО или БОЛЬШЕ.

div()
Определение:
div_t div (int num, int denom)
Описание: функция целочисленного деления целых чисел. Возвращает результат в виде заполненных полей структуры **div_t** (см. *Определения модуля*).

ldiv()
Определение:
ldiv_t ldiv (long num, long denom)
Описание: функция целочисленного деления длинных целых чисел. Возвращает результат в виде заполненных полей структуры **ldiv_t** (см. *Определения модуля*).

qsort()
Определение:
void qsort (void *base, size_t nmemb, size_t size, __compar_fn_t compar)
Параметры:
void ***base** – указатель на исходный массив элементов
size_t **nmemb** – количество элементов в массиве
size_t **size** – размер каждого элемента
__compar_fn_t **compare** – указатель на функцию сравнения элементов (см. *Определения модуля*)
Возвращаемое значение: нет.
Описание: функция выполняет «быструю» сортировку элементов в массиве **base**. Для сравнения элементов используется функция **compar**(), которая возвращает число меньше нуля, равное нулю или большее нуля, соответствующие результатам сравнения МЕНЬШЕ, ЭКВИВАЛЕНТНО или БОЛЬШЕ. Количество элементов определяется значением **nmemb**, размер каждого элемента равен **size**.

strtol()
Определение:
long strtol (const char *nptr, char **endptr, int base)
Параметры:
const char ***nptr** – указатель на строку
char ****endptr** – указатель на указатель на первый необработанный символ в строке
int **base** – основание для преобразования
Возвращаемое значение: результат преобразования строки в число.
Описание: функция осуществляет анализ символов строки **nptr** и их преобразование в длинное целое число, используя **base**, как основание системы счисления. Строка может начинаться с любого числа символов-разделителей, которые определяются функцией isspace(), после которых допустим один символ «–» или «+». base может принимать значения от 2 до 36 включительно или особое значение – ноль. В зависимости от значения base в строке nptr допустимы разные символы. Для **base**==16 число может начинаться с символов ”0x”. Если **base**==0, то происходит автоматическое распознавание системы счисления по первым символам

[32] Скорее всего это будет первый встреченный элемент.

числа: если обнаружена последовательность "0x", то принимается **base**=16, если число начинается с нуля – принимается **base**=8 (т.е. восьмеричная система), а в противном случае принимается **base**=10 (т.е. десятичная система).

Для систем счисления с основанием больше 10 в качестве допустимых символов-цифр принимаются буквы латинского алфавита (независимо от регистра) – аналогично шестнадцатеричной системе. Анализ строки прекращается, как только будет встречен первый символ, не попадающий в число допустимых для принятой системы счисления, при этом, если **endptr** не **NULL**, по его адресу заносится адрес этого символа. Если в строке не встретилось вообще никакой цифры, в **endptr** записывается исходное значение **nptr**.

Если при преобразовании не возникло ошибок переполнения, функция возвращает результат преобразования в виде длинного целого числа со знаком. Если было переполнение, то в переменную **errno** записывается значение **ERANGE** (см. *errno.h – Системные ошибки*)и возвращается LONG_MIN или LONG_MAX соответственно «направлению» переполнения.

strtoul()
Определение:
unsigned long strtoul (const char *nptr, char **endptr, int base)
Описание: функция полностью аналогична strtol(), кроме того, что возвращаемое значение – беззнаковое длинное (соответственно, символ «–» в строке недопустим).

atol()
Определение:
long atol (const char *s)
Параметры:
const char *s – указатель на строку
Описание: функция преобразует строку s (точнее, первые значащие символы строки) в длинное целое со знаком число, которое и возвращает. В отличие от strtol() не производится контроль переполнения (**errno** не устанавливается и результат функции неопределен), зато данная функция требует значительно меньше памяти (как по коду, так и по ОЗУ) и работает гораздо быстрее.
Примечания: в документации не указано, поддерживает ли эта функция автоматическое определение системы счисления, или требует только десятичной?

atoi()
Определение:
int atoi (const char *s)
Описание: функция во всем, кроме типа возвращаемого значения, аналогична atol().

exit()
Определение:
void exit (int status)
Параметры:
int **status** – код завершения
Возвращаемое значение: нет.
Описание: функция вызывает завершение программы. Так как в случае с микроконтроллером нет никакой внешней среды для программы, возвращаемое значение попросту игнорируется, а действие функции заключается в глобальном запрещении прерываний и переходу к бесконечному пустому циклу. В контексте C++ перед этим еще происходит вызов глобального деструктора.

malloc()
Определение:
void * malloc (size_t size)
Параметры:
size_t **size** – размер запрашиваемой области памяти
Возвращаемое значение: указатель на выделенную область или **NULL** в случае невозможности выделения.
Описание: функция выделяет (резервирует) область памяти запрошенного размера и возвращает указатель на эту область, если выделение произошло успешно.
Примечания: функция не производит инициализацию выделенной памяти каким-либо значением. Дополнительно см. *Динамическое распределение памяти*.

free()
Определение:
void free (void *ptr)
Параметры:
void ***ptr** – указатель на область памяти
Возвращаемое значение: нет
Описание: функция освобождает память, на которую указывает указатель ptr, и тем самым делает ее доступной для последующего выделения. Если ptr == NULL, функция ничего не делает.
Примечания: Дополнительно см. *Динамическое распределение памяти*.

calloc()
Определение:
void * calloc (size_t nele, size_t size)
Параметры:
size_t **nele** – число элементов
size_t **size** – размер элемента
Возвращаемое значение: указатель на выделенную память или NULL, если выделение невозможно
Описание: функция выделяет (резервирует) область памяти для размещения **nele** элементов по **size** байт каждый. Полностью аналогична malloc(), кроме того, что выделенная память заполняется нулями.
Примечания: Дополнительно см. *Динамическое распределение памяти*.

realloc()
Определение:
void * realloc (void *ptr, size_t size)
Параметры:
void ***ptr** – указатель на «расширяемую» область памяти
size_t **size** – новый размер для области
Возвращаемое значение: указатель на новую область или NULL, если изменение размера невозможно.
Описание: функция изменяет размер области памяти, ранее выделенной для **ptr**, до размера **size**. Возвращаемое значение не обязательно будет совпадать с исходным значением **ptr**, однако содержимое области памяти будет сохранено (насколько это возможно для нового размера области).

Если изменение размера области невозможно, то значение и содержимое **ptr** не меняется.

strtod()
Определение:
double strtod (const char *nptr, char **endptr)
Параметры:
const char ***nptr** – указатель на строку
char ****endptr** – указатель на указатель на первый необработанный символ строки
Возвращаемое значение: результат преобразования строки в число с плавающей точкой.
Описание: аналогично strtol(), функция осуществляет анализ символов строки nptr (игнорируя начальные пробелы) и преобразование их в число с плавающей точкой. Подразумевается, что число представлено в виде общепринятой записи, в т.ч. с мантиссой, т.е. допустим первый символ «+» или «–», затем десятичные цифры, возможно с разделительной точкой, затем, возможно символ «e» или «E», после которого должно следовать число (возможно со знаком) без точки.

Функция возвращает результат, если не было переполнения в результате преобразования. Если же переполнение было, то переменная **errno** устанавливается в ERANGE (см. *errno.h – Системные ошибки*) и возвращается значение «бесконечность» с соответствующим знаком.

Если **endptr** не NULL, то туда записывается указатель на первый символ в строке, который не обработан (см. *strtol()*).

atof()
Определение:
double atof (const char *nptr)
Описание: функция полностью эквивалентна strtod()

rand()
Определение:
int rand (void)
Возвращаемое значение: псевдослучайное число
Описание: функция возвращает псевдослучайное число в пределах от 0 до RAND_MAX. Это число генерируется по специальному алгоритму, формирующему конечную последовательность псевдослучайных чисел, по достижению конца которой числа начинают повторяться в той же последовательности.

Функция srand() осуществляет «настройку» алгоритма так, что можно получать одинаковые или разные последовательности псевдослучайных чисел, при каждом вызове rand() автоматически предыдущее псевдослучайное значение используется для srand() (это происходит внутри реализации функции).

Примечания: Данная функция обеспечивает достаточно «короткую» последовательность чисел и введена для совместимости со стандартом. (См. более продвинутые варианты random() и srandom(), обладающие лучшим качеством «случайности» генерируемых чисел.)

srand()
Определение:
void srand (unsigned int seed)
Параметры:
unsigned int **seed** – коэффициент генератора случайных чисел
Возвращаемое значение: нет.
Описание: функция выполняет настройку генератора псевдослучайной последовательности чисел, используя коэффициент **seed** в качестве условно-стартового значения. Все это следует понимать лишь как то, что после вызова **srand(5)** и **srand(10)** функция rand() будет возвращать принципиально разные последовательности чисел, однако два независимых устройства после **srand(5)** будут получать абсолютно одинаковые последовательности чисел при помощи rand().
Примечания: по умолчанию генератор псевдослучайной последовательности настроен. Как будто был вызов **srand(1)**.

rand_r()
Определение:
int rand_r (unsigned long *ctx)
Параметры:
unsigned long ***ctx** – указатель на переменную для сохранения контекста
Возвращаемое значение: псевдослучайное число.
Описание: эта функция – реентерабельный вариант rand(). Для сохранения внутренних «настроек» генератора псевдослучайной последовательности она использует не глобальную переменную модуля, а локальную переменную пользовательской программы, на которую указывает **ctx**, таким образом возможен рекурсивный вызов функции, и при этом сохраняется «случайность» возвращаемых значений.

dtostre()
Определение:
char * dtostre (double val, char *s, unsigned char prec, unsigned char flags)
Параметры:
double **val** – преобразуемое число
char ***s** – указатель на строку-результат
unsigned char **prec** – точность представления
unsigned char **flags** – флаги-признаки преобразования
Возвращаемое значение: указатель на строку-результат
Описание: функция осуществляет преобразование **val** в его символьное представление, помещаемое в **s** (должно быть заранее выделено достаточно места). Формат представления числа [-]d.ddd***e***±dd, где количество цифр после точки определяется значением **prec**. Если **prec** не равно нулю, то перед точкой всегда одна цифра, а если **prc**==0, то точка не выводится. Показатель степени всегда содержит 2 знака.
Параметр **flags** может принимать одно или более (объединяемых по ИЛИ) следующих значений:
DTOSTRE_UPPERCASE – символ ***e*** будет заменен на ***E***
DTOSTRE_ALWAYS_SIGN – если число положительное, всегда будет добавлен пробел перед его представлением
DTOSTRE_PLUS_SIGN – всегда будет выведен знак перед представлением, в т.ч. «+» для положительных
Примечания: эта функция не входит в библиотеку **libc.a**, и поэтому должна использоваться лишь совместно с подключением к линкеру математической библиотеки **libm.a** директивой компилятора **–lm**.

dtostrf()
Определение:
char * dtostrf (double val, signed char width, unsigned char prec, char *s)
Параметры:
double **val** – преобразуемое число
signed char **width** – ширина результирующего представления
unsigned char **prec** – точность представления
char ***s** – указатель на строку-результат
Возвращаемое значение: указатель на строку-результат.
Описание: функция осуществляет преобразование числа в формате с плавающей точкой **val** в его символьное представление, помещаемое в строку **s** (достаточное пространство должно быть обеспечено заранее). Результат форматируется в «простой» форме с плавающей точкой, т.е. ***[-]d.dd***, причем **width** показывает, сколько символов должно занимать представление результата (включая десятичную точку и возможный минус для отрицательных чисел), а **prec** определяет число знаков после точки. **width** – это число со знаком, отрицательное значение означает, что результат должен быть выровнен «влево».
Примечания: эта функция не входит в библиотеку **libc.a**, и поэтому должна использоваться лишь совместно с подключением к линкеру математической библиотеки **libm.a** директивой компилятора **–lm**.

Дополнительные функции

itoa()
Определение:
char * itoa (int val, char *s, int radix)
Параметры:
int **val** – преобразуемое число
char ***s** – указатель на строку-результат
int **radix** – основание системы счисления
Возвращаемое значение: указатель на строку-результат.
Описание: функция преобразует целое число **val** в его символьное представление, помещаемое в **s**, в заданной системе счисления по основанию **radix**. Основание **radix** может принимать значения от 2 до 36 включительно, для цифр, «вес» которых более 9 используются символы латинского алфавита, начиная с “a”. Если число отрицательное, то символ «минус» выводится только для десятичной системы счисления (т.е. если **radix**==10).
Примечания: см. «обратную» функцию *atoi()*

ltoa()
Определение:
char * ltoa (long int val, char *s, int radix)
Параметры:
int **val** – преобразуемое число
char ***s** – указатель на строку-результат
int **radix** – основание системы счисления
Возвращаемое значение: указатель на строку-результат.
Описание: функция полностью аналогична itoa(), за исключением того, что **val** – длинное целое число.
Примечания: чем меньшее значение **radix** используется, тем большее пространство должно быть предусмотрено для хранения результата **s**, в противном случае возможен выход за пределы массива с непредсказуемыми последствиями.

utoa()
Определение:
char * utoa (unsigned int val, char *s, int radix)
Описание: функция полностью аналогична itoa(), за исключением того, что **val** – целое число без знака.

ultoa()
Определение:
char * ultoa (unsigned long int val, char *s, int radix)
Описание: функция полностью аналогична itoa(), за исключением того, что val – длинное целое число без знака.

random()
Определение:
long random (void)
Параметры: нет
Возвращаемое значение: псевдослучайное число
Описание: функция возвращает псевдослучайное число в пределах от 0 до RANDOM_MAX. Кроме того, что тип возвращаемого числа – длинное целое, различий в функционировании по сравнению с rand() нет никаких.
Примечания: благодаря большей разрядности возвращаемого числа данная функция образует более «случайную» последовательность, по сравнению с rand(), и ее применение совместно с srandom() и random_r() более предпочтительно.

srandom()
Определение:
void srandom (unsigned long seed)
Описание: за исключением того, что данная функция выполняет «настройку» улучшенного алгоритма для работы функции random(), и имеет аргумент **seed** типа длинное целое число, действие функции полностью аналогично стандартной функции srand().

random_r()
Определение:
long random_r (unsigned long ***ctx**)
Описание: за исключением того, что данная функция возвращает значение и использует аргумент **ctx** типа длинное целое число, действие функции полностью аналогично стандартной функции rand_r().

string.h – Строки

Модуль реализует набор функций поддержки различных действий над строками, оканчивающимися символом NULL (так называемые ASCIIZ-строки).

Модуль предназначен для работы со строками, находящимися в ОЗУ. Если используются строки, хранимые в сегменте кода, следует использовать функции, определенные в модуле avr/pgmspace.h – Поддержка обращения к сегменту кода AVR.

Модуль определяет макрос **_FFS(x)**, который по смыслу полностью аналогичен функции ffs(). Этот макрос вычисляется на этапе компиляции, что порождает быстрый и компактный код, но в качестве своего аргумента должен принимать только константные значения. Применение этого макроса с аргументом-переменной не рекомендуется, так как генерируемый код может быть объемным и/или неверным в принципе.

Функции модуля

ffs()
Определение:
int ffs(int val)
Параметры:
int **val** – анализируемое число
Возвращаемое значение: номер бита.
Описание: функция возвращает номер младшего значащего бита в анализируемом числе **val**. Биты нумеруются с 1-го, которому соответствует младший разряд числа. Если нет ни одного установленного бита в числе, функция возвращает ноль.
Примечания: для вычисления константных значений более предпочтительно применение макроса **_FFS()**

ffsl()
Определение:
int ffsl(long val)
Описание: действие функции полностью аналогично ffs(), за исключением того, что анализируемое число **val** имеет тип длинное целое.

ffsll()
Определение:
int ffsll(long long val)
Описание: действие функции полностью аналогично ffs(), за исключением того, что анализируемое число **val** имеет тип большое длинное целое.

memccpy()
Определение:
void * memccpy(void *dest, const void *src, int val, size_t len)
Параметры:
void ***dest** – указатель на область-приемник
const void ***src** – указатель на область-источник
int **val** – значение для поиска
size_t **len** – ограничитель количества копируемых байт
Возвращаемое значение: указатель на следующий за найденным в области **src** байт или NULL, если не найдено.
Описание: функция выполняет копирование из области **src** в область **dest** не более **len** байт последовательно, прекращая копирование, если очередной байт равен **val**.

memchr()
Определение:
void * memchr(const void *src, int val, size_t len)
Параметры:
const void ***src** – указатель на начало области поиска
int **val** – искомое значение
size_t **len** – длина области
Возвращаемое значение: указатель на найденный байт или NULL, если не найдено.
Описание: функция осуществляет поиск байта **val** (обрабатываемого как unsigned char) в области памяти, начинающейся с адреса **src** и содержащей **len** байт. Поиск прекращается на первом встреченном байте, значение которого совпадает с **val**.

memcmp()
Определение:
int memcmp(const void *s1, const void *s2, size_t len)
Параметры:
const void ***s1** – указатель на первую область памяти
const void ***s2** – указатель на вторую область памяти
size_t **len** – длина областей
Возвращаемое значение: число меньше нуля, равное нулю или большее нуля в случае, если первая область соответственно меньше, равна или больше второй по содержимому.
Описание: функция сравнивает **len** первых байтов в областях **s1** и **s2**, возвращая результат сравнения.
Примечания:
1. Сравнение байтов ведется как беззнаковых (unsigned char).
2. Если сравниваются области, заполненные 16-битными числами (или более) результат сравнения может быть некорректным.
3. Эта функция не совместима с опцией компилятора **–mint8**, однако если важна проверка только на равенство, она может корректно использоваться и с этой опцией.

memcpy()
Определение:
void * memcpy(void *dest, const void *src, size_t len)
Параметры:
void ***dest** – указатель на область-приемник
const void ***src** – указатель на область-источник
size_t **len** – ограничитель количества копируемых байт
Возвращаемое значение: указатель на область-приемник.
Описание: функция копирует **len** байт из области-источника в область-приемник.
Примечания: области **dest** и **src** не должны пересекаться! Для правильного копирования пересекающихся областей следует использовать memmove()

memmem()
Определение:
void * memmem(const void *s1, size_t len1, const void *s2, size_t len2)
Параметры:
const void ***s1** – область поиска
size_t **len1** – длина области поиска
const void ***s2** – «искомая» область
size_t **len2** – длина «искомой» области
Возвращаемое значение: указатель на начало первого вхождения области **s2** внутри **s1** или NULL, если не найдено.
Описание: функция ищет первое полное вхождение последовательности **s2** из **len2** байт внутри области **s1** из **len1** байт. Если **len2==0**, функция возвращает указатель на **s1**.

memmove()
Определение:
void * memmove(void *, const void *, size_t)
Параметры:
void ***dest** – указатель на область-приемник
const void ***src** – указатель на область-источник
size_t **len** – ограничитель количества копируемых байт
Возвращаемое значение: указатель на область-приемник.
Описание: функция копирует **len** байт из области-источника в область-приемник. Причем эти области могут пересекаться.

memrchr()
Определение:
void * memrchr(const void *src, int val, size_t len)
Параметры:
const void ***src** – указатель на начало области поиска
int **val** – искомое значение
size_t **len** – длина области
Возвращаемое значение: указатель на найденный байт или NULL, если не найдено.
Примечания: функция, выполняющая поиск байта, как и memchr(), только в обратном направлении, т.е. от конца области **src** к ее началу.

memset()
Определение:
void * memset(void *dest, int val, size_t len)

Параметры:
const void ***dest** – указатель на начало области
int **val** – значение для записи
size_t **len** – длина области
Возвращаемое значение: **dest**
Описание: функция осуществляет заполнение **len** первых байтов области **dest** значением **val**, рассматриваемым как байт.

strcasecmp()
Определение:
int strcasecmp(const char *s1, const char *s2)
Параметры:
const char ***s1** – указатель на первую строку
const char ***s2** – указатель на вторую строку
Возвращаемое значение: число меньше нуля, равное нулю или большее нуля в случае, если первая строка соответственно меньше, равна или больше второй по содержимому.
Описание: функция производит сравнение двух ASCIIZ-строк без учета регистра символов. Сравнение ведется с учетом фактической длины **s1** и **s2**: если одна из строк совпадает по символам со второй, но короче – она считается меньше другой.

strcasestr()
Определение:
char * strcasestr(const char *s1, const char *s2)
Параметры:
const char ***s1** – указатель на первую строку
const char ***s2** – указатель на вторую строку
Возвращаемое значение: указатель на начало первого вхождения строки **s2** в строке **s1** или NULL, если не найдено.
Описание: функция ищет подстроку **s2** внутри строки **s1** без учета регистра символов. Если **s2** равна **s1** или наоборот, пуста, то возвращается **s1**.
Примечания: см. также *strstr()*

strcat()
Определение:
char * strcat(char *dest, const char *src)
Параметры:
char ***dest** – указатель на строку-приемник
const char ***src** – указатель на строку-источник
Возвращаемое значение: указатель на строку-приемник.
Описание: функция осуществляет конкатенацию двух строк, присоединяя к **dest** строку **src**. Строки не должны пересекаться и в **dest** должно быть предусмотрено достаточное пространство.

strchr()
Определение:
char * strchr(const char *str, int val)
Параметры:
const char ***str** – указатель на строку
int **val** – искомый символ
Возвращаемое значение: указатель на найденный символ в строке или NULL, если не найдено.
Описание: функция ищет первый символ **val** в строке **str**, возвращая указатель на него. Не смотря на тип переменной **val**, функция не может использоваться для поиска мультибайтных символов, т.е. «символ» в данном случае означает «байт».

strchrnul()
Определение:
char * strchrnul(const char *str, int val)
Параметры:
const char ***str** – указатель на строку
int **val** – искомый символ
Возвращаемое значение: указатель на найденный символ в строке или на пустую строку, если не найдено.
Описание: функция, как и strchr(), осуществляет поиск символа **val** в строке **str**, однако всегда возвращает не NULL-указатель: если символ не найден, возвращается указатель на символ NULL, завершающий **str**.

strcmp()
Определение:
int strcmp(const char *s1, const char *s2)
Параметры:
const char ***s1** – указатель на первую строку
const char ***s2** – указатель на вторую строку
Возвращаемое значение: число меньше нуля, равное нулю или большее нуля в случае, если первая строка соответственно меньше, равна или больше второй по содержимому.
Описание: функция производит сравнение двух ASCIIZ-строк. Сравнение ведется с учетом фактической длины **s1** и **s2**: если одна из строк совпадает по символам со второй, но короче – она считается меньше другой.
Примечания: см. так же функцию *strcasecmp()*

strcpy()
Определение:
char * strcpy(char *dest, const char *src)
Параметры:
char ***dest** – указатель на строку-приемник
const char ***src** – указатель на строку-источник
Возвращаемое значение: указатель на строку-приемник.
Описание: функция копирует в буфер **dest** строку **src** (включая завершающий NULL).
Примечания: если буфер **dest** не содержит достаточного пространства для размещения целиком строки **src** (что бывает, если программист глуп или просто ленив, и не смог проверить размеры строк перед копированием)[33], может произойти выход за допустимые границы области памяти (переполнение буфера) – любимая техника хакеров.

[33] В скобках приведен **дословный перевод** соответствующего текста из оригинальной документации WinAVR.

strcspn()
Определение:
size_t strcspn(const char *s, const char *reject)
Параметры:
const char ***s** – указатель на строку
const char ***reject** – указатель на строку-множество
Возвращаемое значение: число символов.
Описание: функция вычисляет количество подряд идущих первых символов в строке **s**, не совпадающих ни с одним из символов строки **reject** (завершающий ноль не рассматривается как часть строки).

strlcat()
Определение:
size_t strlcat(char *dest, const char *src, size_t siz)
Параметры:
Возвращаемое значение:
Описание:
Примечания:

strlcpy()
Определение:
size_t strlcpy(char *dest, const char *src, size_t siz)
Параметры:
Возвращаемое значение:
Описание:
Примечания:

strlen()
Определение:
size_t strlen(const char *str)
Параметры:
const char *str – указатель на строку
Возвращаемое значение: количество символов в строке.
Описание: функция вычисляет длину строки. Завершающий ноль не участвует в подсчете.

strlwr()
Определение:
char * strlwr(char *s)
Описание: переводит «буквенные» символы строки **s** в нижний регистр. Возвращает **s**.

strncasecmp()
Определение:
int strncasecmp(const char *s1, const char *s2, size_t len)
Описание: функция полностью аналогична strcasecmp(), за исключением того, что сравниваются только **len** первых байтов строки **s1**.

strncat()
Определение:
char * strncat(char *dest, const char *src, size_t len)
Описание: функция полностью аналогична strcat(), за исключением того, что будут присоединены только **len** первых байтов строки **src**.

strncmp()
Определение:

int strncmp(const char *s1, const char *s2, size_t len)

Описание: функция полностью аналогична strcpy(), за исключением того, что сравниваются только **len** первых байтов строки **s1**.

strncpy()

Определение:

char * strncpy(char *dest, const char *src, size_t len)

Параметры:

char ***dest** – указатель на строку-приемник

const char ***src** – указатель на строку-источник

size_t **len** – количество копируемых байт

Возвращаемое значение: указатель на строку-приемник.

Описание: функция осуществляет копирование не более **len** байтов из строки **src** в строку **dest**. Если среди **len** первых байтов **src** не будет встречен NULL, то **dest** не будет иметь завершающего нуля. Если же ноль встретится раньше, чем скопируются **len** байтов, остаток будет дополнен нулями.

strnlen()

Определение:

size_t strnlen(const char *str, size_t len)

Параметры:

const char ***str** – указатель на строку

size_t **len** – ограничитель

Возвращаемое значение: длина строки, но не более **len**

Описание: функция возвращает количество символов в строке **str** (не включая завершающий ноль), если это количество меньше **len**, или значение **len** в противном случае.

strpbrk()

Определение:

char *strpbrk(const char *s, const char *accept)

Параметры:

const char ***s** – указатель на строку

const char ***accept** – указатель на строку-множество

Возвращаемое значение: указатель на первый встреченный символ в строке s или NULL, если не найдено.

Описание: функция ищет в строке **s** первый символ, совпадающий с любым символом из строки **accept** (завершающий ноль не входит в число проверяемых). Если оба или любой из параметров пусты, возвращается NULL.

strrchr()

Определение:

char *strrchr(const char *str, int val)

Параметры:

const char ***str** – указатель на строку

int **val** – искомый символ

Возвращаемое значение: указатель на найденный символ или NULL, если не найдено.

Описание: функция ищет последний символ в строке **str**, равный **val**, или, что то же самое, первый символ **val** от конца строки. Как и strchr(), эта функция не может использоваться для поиска многобайтных символов, т.е. **val** рассматривается как байт.

strrev()

Определение:

char *strrev(char *str)

Параметры:

char *str – указатель на строку

Возвращаемое значение: str

Описание: функция изменяет порядок следования символов в строке str на противоположный (т.е. “ABCD” превращается в “DCBA”).

strsep()

Определение:

char *strsep(char **sp, const char *delim)

Параметры:

char ***sp** – указатель на указатель на строку

const char ***delim** – указатель на строку-разделитель

Возвращаемое значение: указатель на исходное значение ***sp**

Описание: функция разбивает строку на отдельные элементы. Функция ищет первое вхождение любого из символов **delim** (в том числе и завершающий ноль) в строке, на которую указывает **sp**, и заменяет этот символ нулем, а в **sp** запоминается указатель на следующий за найденным символ.

Примечания: если сравнить значение символа, на который указывает **sp** после завершения функции, с нулем, то можно выяснить, остались ли в исходной строке еще другие символы, отличные от символов-разделителей, или нет.

strspn()

Определение:

size_t strspn(const char *s, const char *accept)

Параметры:

const char ***s** – указатель на строку

const char ***accept** – указатель на строку-множество

Возвращаемое значение: количество символов.

Описание: функция определяет, сколько первых подряд идущих символов в строке **s** совпадает с любым из символов строки **accept**, и возвращает это количество.

strstr()

Определение:

char *strstr(const char *s1, const char *s2)

Параметры:

const char ***s1** – указатель на первую строку

const char ***s2** – указатель на вторую строку

Возвращаемое значение: указатель на начало первого вхождения строки **s2** в строке **s1** или NULL, если не найдено.

Описание: функция ищет подстроку **s2** внутри строки **s1**. Если **s2** равна **s1** или наоборот, пуста, то возвращается **s1**.

strtok_r()

Определение:

char *strtok_r(char *str, const char *delim, char **last)

Параметры:

char ***str** – указатель на строку

const char ***delim** – указатель на строку-разделитель

char ****last** – указатель на указатель на строку-остаток

Возвращаемое значение: NULL, если вся строка разобрана, или указатель очередной найденный элемент.

Описание: функция осуществляет разбор (парсинг) строки **str** по отдельным элементам, разделенным символами из строки **delim**. Парсинг строки ведется следующим образом.

Первый вызов **strtok_r()** обязательно происходит с параметром **str**, указывающим на разбираемую строку, все последующие вызовы делаются с **str=NULL**. Переменная-указатель **last** должна оставаться не измененной во всем процессе парсинга одной и той же строки, а **delim** может меняться от вызова к вызову. При каждом вызове функция ищет в разбираемой строке **str** очередное вхождение любого из символов строки **delim**, заменяет его на ноль и возвращает указатель на начало выделенного таким образом элемента, а указатель на следующий символ сохраняется в переменной **last** до следующего вызова функции. Если возвращенное значение не NULL, можно продолжить парсинг строки следующим вызовом функции.

Примечания: переменная для хранения указателя **last** – это локальная переменная, определенная пользователем, она используется для обеспечения реентерабельности функции **strtok_r()** (т.е. эта функция может вызываться из рекурсивной функции в программе).

strupr()

Определение:

char *strupr(char *)

Описание: переводит «буквенные» символы строки **s** в верхний регистр. Возвращает **s**.

Продолжение в №1/2011