`string.h` Library

Source: http://en.wikibooks.org/wiki/C_Programming/Strings

strlen()

```
size_t strlen(const char *s);
```

The strlen() function shall compute and return the number of bytes in the string to which s points, not including the terminating null byte. No value is used to indicate an error.

```
size_t (strlen)(const char *s)
{
    const char *p = s;
    while (*p != '\0')
        p++;
    return (size_t)(p - s);
}
```

strcmp()

```
int strcmp(const char *s1, const char *s2);
```

The strcmp() function takes two strings s1 and s2 as arguments and returns a value less (greater) than zero if the s1 is lexographically less (greater) than s2 or zero if the two strings are equal.

```
int (strcmp)(const char *s1, const char *s2)
{
    unsigned char uc1, uc2;
    while ((*s1 != '\0') && (*s1 == *s2)) {
        s1++;
        s2++;
    }
    uc1 = (*(unsigned char *) s1);
    uc2 = (*(unsigned char *) s2);
    return ((uc1 < uc2) ? -1 : (uc1 > uc2));
}
```

See also: **strncmp()**

strcpy()

```
char *strcpy(char *s1, const char *s2);
```

The strcpy() function shall copy the C string pointed to by s2 (including the terminating null byte) into the array pointed to by s1. If copying takes place between objects that overlap, the behavior is undefined. The function returns s1. There is no value used to indicate an error: if the arguments to strcpy() are correct, and the destination buffer is large enough, the function will never fail.

```
char * (strcpy)(char *s1, const char *s2)
{
    char *dst = s1;
    const char *src = s2;
    while ((*dst++ = *src++) != '\0')
        ;
    return s1;
}
```

See also: **strncpy()**

strcat()

```
char *strcat(char * s1, const char * s2);
```

The strcat() function shall append a copy of the string pointed to by s2 (including the terminating null byte) to the end of the string pointed to by s1. The initial byte of s2 overwrites the null byte at the end of s1. If copying takes place between objects that overlap, the behavior is undefined. The function returns s1.

```
char * (strcat)(char *s1, const char *s2)
{
    char *s = s1;
    while (*s != '\0')
        s++;
    strcpy(s, s2);
    return s1;
}
```

See also: **strncat()**

strchr()

```
char *strchr(const char *s, int c);
```

The strchr() function shall locate the first occurrence of c (converted to a char) in the string pointed to by s. The terminating null byte is considered to be part of the string. Upon completion, strchr() shall return a pointer to the byte, or a null pointer if the byte was not found.

```
char * (strchr)(const char *s, int c)
{
    while ((*s != '\0') && (*s != (char) c))
        s++;
    return ((*s == c) ? (char *) s : NULL);
}
```

See also: **strrchr()**

strstr()

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

The strstr() function shall locate the first occurrence in the string pointed to by s1 of the sequence of bytes (excluding the terminating null byte) in the string pointed to by s2. The function returns the pointer to the matching string in s1 or a null pointer if a match is not found. If s2 is an empty string, the function returns s1.

```
char * (strstr)(const char *s1, const char *s2)
{
    size_t s2len;
    if (*s2 == '\0')
        return (char *) s1;
    s2len = strlen(s2);
    for (; (s1 = strchr(s1, *s2)) != NULL; s1++)
        if (strncmp(s1, s2, s2len) == 0)
            return (char *) s1;
    return NULL;
}
```

strspn()

```
size_t strspn(const char *s1, const char *s2);
```

The strspn() function computes the length of the maximum initial segment of the string pointed to by s1 which consists entirely of characters from the string pointed to by s2.

```
size_t (strspn)(const char *s1, const char *s2)
{
    const char *sc1;
    for (sc1 = s1; *sc1 != '\0'; sc1++)
        if (strchr(s2, *sc1) == NULL)
            return (sc1 - s1);
    return sc1 - s1;
}
```

See also: **strcspn()**

strpbrk()

```
char *strchr(const char *s, int c);
```

The strpbrk() locates the first occurrence in the string pointed to by s1 of any character from the string pointed to by s2, returning a pointer to that character or a null pointer if not found.

```
char * (strpbrk)(const char *s1, const char *s2)
{
    const char *sc1;
    for (sc1 = s1; *sc1 != '\0'; sc1++)
        if (strchr(s2, *sc1) != NULL)
            return (char *)sc1;
    return NULL;
}
```

strtok()

```
char *strtok(char *s1, const char *delimiters);
```

A sequence of calls to strtok() breaks the string pointed to by s1 into a sequence of tokens, each of which is delimited by a byte from the string pointed to by delimiters. The first call in the sequence has s1 as its first argument, and is followed by calls with a null pointer as their first argument. The separator string pointed to by delimiters may be different from call to call.

```
char * (strtok_r)(char *s, const char *delimiters, char **lasts)
{
    char *sbegin, *send;
    sbegin = s ? s : *lasts;
    sbegin += strspn(sbegin, delimiters);
    if (*sbegin == '\0') {
        *lasts = "";
        return NULL;
    }
    send = sbegin + strcspn(sbegin, delimiters);
    if (*send != '\0')
        *send++ = '\0';
    *lasts = send;
    return sbegin;
}
char *(strtok)(char *restrict s1, const char *restrict delimiters)
{
    static char *ssave = "";
    return strtok_r(s1, delimiters, &ssave);
}
```

memchr()

```
void *memchr(const void *s, int c, size_t n);
```

The memchr() function shall locate the first occurrence of c (converted to an unsigned char) in the initial n bytes (each interpreted as unsigned char) of the object pointed to by s. If c is not found, memchr() returns a null pointer.

```
void * (memchr)(const void *s, int c, size_t n)
{
    const unsigned char *src = s;
    unsigned char uc = c;
    while (n-- != 0) {
        if (*src == uc)
            return (void *) src;
        src++;
    }
    return NULL;
}
```

memcmp()

```
int memcmp(const void *s1, const void *s2, size_t n);
```

The memcmp() function shall compare the first n bytes (each interpreted as unsigned char) of the object pointed to by s1 to the first n bytes of the object pointed to by s2. The sign of a **non-zero return value** shall be determined by the sign of the difference between the values of the first pair of bytes (both interpreted as type unsigned char) that differ in the objects being compared.

```
int (memcmp)(const void *s1, const void *s2, size_t n)
{
    const unsigned char *us1 = (const unsigned char *) s1;
    const unsigned char *us2 = (const unsigned char *) s2;
    while (n-- != 0) {
        if (*us1 != *us2)
            return (*us1 < *us2) ? -1 : +1;
        us1++;
        us2++;
    }
    return 0;
}
```

memcpy()

```
void *memcpy(void * s1, const void * s2, size_t n);
```

The memcpy() function shall copy n bytes from the object pointed to by s2 into the object pointed to by s1. If copying takes place between objects that overlap, the behavior is undefined. The function returns s1.

```
void * (memcpy)(void * s1, const void * s2, size_t n)
{
    char *dst = s1;
    const char *src = s2;
    while (n-- != 0)
        *dst++ = *src++;
    return s1;
}
```

memmove()

```
void *memmove(void *s1, const void *s2, size_t n);
```

The memmove() function shall copy n bytes from the object pointed to by s2 into the object pointed to by s1. Copying takes place as if the n bytes from the object pointed to by s2 are first copied into a temporary array of n bytes that does not overlap the objects pointed to by s1 and s2, and then the n bytes from the temporary array are copied into the object pointed to by s1. The function returns the value of s1. The following code is not completely portable. (Why?)

```
void * (memmove)(void *s1, const void *s2, size_t n)
{
    char *p1 = s1;
    const char *p2 = s2;
    if (p2 < p1 && p1 < p2 + n) {
        p2 += n;
        p1 += n;
        while (n-- != 0)
            *--p1 = *--p2;
    } else
        while (n-- != 0)
            *p1++ = *p2++;
    return s1;
}
```

memset()

```
void *memset(void *s, int c, size_t n);
```

The memset() function converts c into unsigned char, then stores the character into the first n bytes of memory pointed to by s.

```
void *(memset)(void *s, int c, size_t n)
{
    unsigned char *us = s;
    unsigned char uc = c;
    while (n-- != 0)
        *us++ = uc;
    return s;
}
```