7.21 String handling `<string.h>`

7.21.1 String function conventions

1 The header **`<string.h>`** declares one type and several functions, and defines one macro useful for manipulating arrays of character type and other objects treated as arrays of character type.[268)] The type is **`size_t`** and the macro is **`NULL`** (both described in 7.17). Various methods are used for determining the lengths of the arrays, but in all cases a **`char *`** or **`void *`** argument points to the initial (lowest addressed) character of the array. If an array is accessed beyond the end of an object, the behavior is undefined.

2 Where an argument declared as **`size_t n`** specifies the length of the array for a function, **`n`** can have the value zero on a call to that function. Unless explicitly stated otherwise in the description of a particular function in this subclause, pointer arguments on such a call shall still have valid values, as described in 7.1.4. On such a call, a function that locates a character finds no occurrence, a function that compares two character sequences returns zero, and a function that copies characters copies zero characters.

3 For all functions in this subclause, each character shall be interpreted as if it had the type **`unsigned char`** (and therefore every possible object representation is valid and has a different value).

7.21.2 Copying functions

7.21.2.1 The `memcpy` function

Synopsis

1
```
#include <string.h>
void *memcpy(void * restrict s1,
     const void * restrict s2,
     size_t n);
```

Description

2 The **`memcpy`** function copies **`n`** characters 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.

Returns

3 The **`memcpy`** function returns the value of **`s1`**.

268) See “future library directions” (7.26.11).

7.21.2.2 The `memmove` function

Synopsis

1
```
#include <string.h>
void *memmove(void *s1, const void *s2, size_t n);
```

Description

2 The **`memmove`** function copies **`n`** characters from the object pointed to by **`s2`** into the object pointed to by **`s1`**. Copying takes place as if the **`n`** characters from the object pointed to by **`s2`** are first copied into a temporary array of **`n`** characters that does not overlap the objects pointed to by **`s1`** and **`s2`**, and then the **`n`** characters from the temporary array are copied into the object pointed to by **`s1`**.

Returns

3 The **`memmove`** function returns the value of **`s1`**.

7.21.2.3 The `strcpy` function

Synopsis

1
```
#include <string.h>
char *strcpy(char * restrict s1,
     const char * restrict s2);
```

Description

2 The **`strcpy`** function copies the string pointed to by **`s2`** (including the terminating null character) into the array pointed to by **`s1`**. If copying takes place between objects that overlap, the behavior is undefined.

Returns

3 The **`strcpy`** function returns the value of **`s1`**.

7.21.2.4 The `strncpy` function

Synopsis

1
```
#include <string.h>
char *strncpy(char * restrict s1,
     const char * restrict s2,
     size_t n);
```

Description

2 The **`strncpy`** function copies not more than **`n`** characters (characters that follow a null character are not copied) from the array pointed to by **`s2`** to the array pointed to by

s1.[269] If copying takes place between objects that overlap, the behavior is undefined.

3 If the array pointed to by **s2** is a string that is shorter than **n** characters, null characters are appended to the copy in the array pointed to by **s1**, until **n** characters in all have been written.

Returns

4 The **strncpy** function returns the value of **s1**.

7.21.3 Concatenation functions

7.21.3.1 The strcat function

Synopsis

1
```
#include <string.h>
char *strcat(char * restrict s1,
     const char * restrict s2);
```

Description

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

Returns

3 The **strcat** function returns the value of **s1**.

7.21.3.2 The strncat function

Synopsis

1
```
#include <string.h>
char *strncat(char * restrict s1,
     const char * restrict s2,
     size_t n);
```

Description

2 The **strncat** function appends not more than **n** characters (a null character and characters that follow it are not appended) from the array pointed to by **s2** to the end of the string pointed to by **s1**. The initial character of **s2** overwrites the null character at the end of **s1**. A terminating null character is always appended to the result.[270] If copying

269) Thus, if there is no null character in the first **n** characters of the array pointed to by **s2**, the result will not be null-terminated.

270) Thus, the maximum number of characters that can end up in the array pointed to by **s1** is **strlen(s1)+n+1**.

takes place between objects that overlap, the behavior is undefined.

Returns

3 The **strncat** function returns the value of **s1**.

Forward references: the **strlen** function (7.21.6.3).

7.21.4 Comparison functions

1 The sign of a nonzero value returned by the comparison functions **memcmp**, **strcmp**, and **strncmp** is determined by the sign of the difference between the values of the first pair of characters (both interpreted as **unsigned char**) that differ in the objects being compared.

7.21.4.1 The memcmp function

Synopsis

1
```
#include <string.h>
int memcmp(const void *s1, const void *s2, size_t n);
```

Description

2 The **memcmp** function compares the first **n** characters of the object pointed to by **s1** to the first **n** characters of the object pointed to by **s2**.[271)]

Returns

3 The **memcmp** function returns an integer greater than, equal to, or less than zero, accordingly as the object pointed to by **s1** is greater than, equal to, or less than the object pointed to by **s2**.

7.21.4.2 The strcmp function

Synopsis

1
```
#include <string.h>
int strcmp(const char *s1, const char *s2);
```

Description

2 The **strcmp** function compares the string pointed to by **s1** to the string pointed to by **s2**.

Returns

3 The **strcmp** function returns an integer greater than, equal to, or less than zero, accordingly as the string pointed to by **s1** is greater than, equal to, or less than the string

271) The contents of "holes" used as padding for purposes of alignment within structure objects are indeterminate. Strings shorter than their allocated space and unions may also cause problems in comparison.

pointed to by **s2**.

7.21.4.3 The **strcoll** function

Synopsis

1
```
#include <string.h>
int strcoll(const char *s1, const char *s2);
```

Description

2 The **strcoll** function compares the string pointed to by **s1** to the string pointed to by **s2**, both interpreted as appropriate to the **LC_COLLATE** category of the current locale.

Returns

3 The **strcoll** function returns an integer greater than, equal to, or less than zero, accordingly as the string pointed to by **s1** is greater than, equal to, or less than the string pointed to by **s2** when both are interpreted as appropriate to the current locale.

7.21.4.4 The **strncmp** function

Synopsis

1
```
#include <string.h>
int strncmp(const char *s1, const char *s2, size_t n);
```

Description

2 The **strncmp** function compares not more than **n** characters (characters that follow a null character are not compared) from the array pointed to by **s1** to the array pointed to by **s2**.

Returns

3 The **strncmp** function returns an integer greater than, equal to, or less than zero, accordingly as the possibly null-terminated array pointed to by **s1** is greater than, equal to, or less than the possibly null-terminated array pointed to by **s2**.

7.21.4.5 The **strxfrm** function

Synopsis

1
```
#include <string.h>
size_t strxfrm(char * restrict s1,
     const char * restrict s2,
     size_t n);
```

Description

2 The **strxfrm** function transforms the string pointed to by **s2** and places the resulting string into the array pointed to by **s1**. The transformation is such that if the **strcmp** function is applied to two transformed strings, it returns a value greater than, equal to, or

less than zero, corresponding to the result of the **strcoll** function applied to the same two original strings. No more than **n** characters are placed into the resulting array pointed to by **s1**, including the terminating null character. If **n** is zero, **s1** is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior is undefined.

Returns

3 The **strxfrm** function returns the length of the transformed string (not including the terminating null character). If the value returned is **n** or more, the contents of the array pointed to by **s1** are indeterminate.

4 EXAMPLE The value of the following expression is the size of the array needed to hold the transformation of the string pointed to by **s**.

```
1 + strxfrm(NULL, s, 0)
```

7.21.5 Search functions

7.21.5.1 The memchr function

Synopsis

1
```
#include <string.h>
void *memchr(const void *s, int c, size_t n);
```

Description

2 The **memchr** function locates the first occurrence of **c** (converted to an **unsigned char**) in the initial **n** characters (each interpreted as **unsigned char**) of the object pointed to by **s**.

Returns

3 The **memchr** function returns a pointer to the located character, or a null pointer if the character does not occur in the object.

7.21.5.2 The strchr function

Synopsis

1
```
#include <string.h>
char *strchr(const char *s, int c);
```

Description

2 The **strchr** function locates the first occurrence of **c** (converted to a **char**) in the string pointed to by **s**. The terminating null character is considered to be part of the string.

Returns

3 The **strchr** function returns a pointer to the located character, or a null pointer if the character does not occur in the string.

7.21.5.3 The `strcspn` function

Synopsis

1
```
        #include <string.h>
        size_t strcspn(const char *s1, const char *s2);
```

Description

2 The **`strcspn`** function computes the length of the maximum initial segment of the string pointed to by **`s1`** which consists entirely of characters *not* from the string pointed to by **`s2`**.

Returns

3 The **`strcspn`** function returns the length of the segment.

7.21.5.4 The `strpbrk` function

Synopsis

1
```
        #include <string.h>
        char *strpbrk(const char *s1, const char *s2);
```

Description

2 The **`strpbrk`** function locates the first occurrence in the string pointed to by **`s1`** of any character from the string pointed to by **`s2`**.

Returns

3 The **`strpbrk`** function returns a pointer to the character, or a null pointer if no character from **`s2`** occurs in **`s1`**.

7.21.5.5 The `strrchr` function

Synopsis

1
```
        #include <string.h>
        char *strrchr(const char *s, int c);
```

Description

2 The **`strrchr`** function locates the last occurrence of **`c`** (converted to a **`char`**) in the string pointed to by **`s`**. The terminating null character is considered to be part of the string.

Returns

3 The **`strrchr`** function returns a pointer to the character, or a null pointer if **`c`** does not occur in the string.

7.21.5.6 The `strspn` function

Synopsis

1
```
#include <string.h>
size_t strspn(const char *s1, const char *s2);
```

Description

2 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`**.

Returns

3 The **`strspn`** function returns the length of the segment.

7.21.5.7 The `strstr` function

Synopsis

1
```
#include <string.h>
char *strstr(const char *s1, const char *s2);
```

Description

2 The **`strstr`** function locates the first occurrence in the string pointed to by **`s1`** of the sequence of characters (excluding the terminating null character) in the string pointed to by **`s2`**.

Returns

3 The **`strstr`** function returns a pointer to the located string, or a null pointer if the string is not found. If **`s2`** points to a string with zero length, the function returns **`s1`**.

7.21.5.8 The `strtok` function

Synopsis

1
```
#include <string.h>
char *strtok(char * restrict s1,
     const char * restrict s2);
```

Description

2 A sequence of calls to the **`strtok`** function breaks the string pointed to by **`s1`** into a sequence of tokens, each of which is delimited by a character from the string pointed to by **`s2`**. The first call in the sequence has a non-null first argument; subsequent calls in the sequence have a null first argument. The separator string pointed to by **`s2`** may be different from call to call.

3 The first call in the sequence searches the string pointed to by **`s1`** for the first character that is *not* contained in the current separator string pointed to by **`s2`**. If no such character is found, then there are no tokens in the string pointed to by **`s1`** and the **`strtok`** function

returns a null pointer. If such a character is found, it is the start of the first token.

4 The **strtok** function then searches from there for a character that *is* contained in the current separator string. If no such character is found, the current token extends to the end of the string pointed to by **s1**, and subsequent searches for a token will return a null pointer. If such a character is found, it is overwritten by a null character, which terminates the current token. The **strtok** function saves a pointer to the following character, from which the next search for a token will start.

5 Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above.

6 The implementation shall behave as if no library function calls the **strtok** function.

Returns

7 The **strtok** function returns a pointer to the first character of a token, or a null pointer if there is no token.

8 EXAMPLE

```
#include <string.h>
static char str[] = "?a???b,,,#c";
char *t;

t = strtok(str, "?");    // t points to the token "a"
t = strtok(NULL, ",");   // t points to the token "??b"
t = strtok(NULL, "#,");  // t points to the token "c"
t = strtok(NULL, "?");   // t is a null pointer
```

7.21.6 Miscellaneous functions

7.21.6.1 The memset function

Synopsis

1
```
#include <string.h>
void *memset(void *s, int c, size_t n);
```

Description

2 The **memset** function copies the value of **c** (converted to an **unsigned char**) into each of the first **n** characters of the object pointed to by **s**.

Returns

3 The **memset** function returns the value of **s**.

7.21.6.2 The `strerror` function

Synopsis

1
```
#include <string.h>
char *strerror(int errnum);
```

Description

2 The **`strerror`** function maps the number in **`errnum`** to a message string. Typically, the values for **`errnum`** come from **`errno`**, but **`strerror`** shall map any value of type **`int`** to a message.

3 The implementation shall behave as if no library function calls the **`strerror`** function.

Returns

4 The **`strerror`** function returns a pointer to the string, the contents of which are locale-specific. The array pointed to shall not be modified by the program, but may be overwritten by a subsequent call to the **`strerror`** function.

7.21.6.3 The `strlen` function

Synopsis

1
```
#include <string.h>
size_t strlen(const char *s);
```

Description

2 The **`strlen`** function computes the length of the string pointed to by **`s`**.

Returns

3 The **`strlen`** function returns the number of characters that precede the terminating null character.