UEE1302
Introduction to Computers and Programming

C_Lecture 09:

C Characters and Strings

C: How to Program 6th ed.
Chapter 8 C Characters and Strings

Agenda

- Fundamentals of Strings and Characters
- Character-Handling Library
- String-Conversion Functions
- Standard Input/Output Library Functions
- String-Handling Library

Special: `char` Array

- `char` array: an array consists of multiple characters char
  - Example:

'W'	'i'	'n'	\0

```
char name[] = "Win";
printf("%d\n", sizeof(name));
```

  - \0 in `name[3]` is called `NULL` character and automatically appended in the assignment.
  - Another example:

'W'	'i'	'n'

```
char name[] = {'W','i','n'};
printf("%d\n", sizeof(name));
```

Characters vs. Strings

- Characters are the fundamental building blocks of source programs.
  - A character constant is an `int` value represented as a character in single quotes.
- A string is a series of characters treated as a single unit.
  - A string in C is an array of characters ending in the null character (`'\0'`).
  - A string is access via a pointer to the first character in the string. => a string is a pointer.

```c
char *name = "Win";
char name[] = "Win";
```

'W'	'i'	'n'	\0

Characters vs. Strings (cont.)

- There is a difference between 'A' and "A"
  - 'A' is the character A
  - "A" is the string A
- Because strings are null terminated, "A" represents two characters, 'A' and `'\0'`
- Similarly, "Hello" contains six characters: 'H', 'e', 'l', 'l', 'o', and `'\0'`

Examples of C String

- Consider the statement

```
char name[16];
```

- Because C-String are null terminated and `name` has sixteen components
  - The largest string that can be stored in `name` is 15
- If you store a string of length, say 10 in `name`
  - the first 11 components of `name` (include `'\0'`) are used and
  - the last 5 components are left unused

C String Indexes

- A C-String is an array
- Can access indexed variables of:

```
char ourStr[6] = "ECE";
```

  - `ourStr[0]` is `'E'`
  - `ourStr[1]` is `'C'`
  - `ourStr[2]` is `'E'`
  - `ourStr[3]` is `'\0'`
  - `ourStr[4]` is unknown
  - `ourStr[5]` is unknown

'E'	'C'	'E'	'\0'	?	?
outStr[0]	outStr[1]	outStr[2]	outStr[3]	outStr[4]	outStr[5]

C-String Index Manipulation

- Can manipulate indexed variables

```
char happyString[7] = "DoBeDo";
happyString[6] = 'Z';
```

  - Be careful!
  - Here, `'\0'` (null) was overwritten by a `'Z'`
- If null overwritten, C-String no longer "acts" like C-String!
  - unpredictable results!

    => don’t know when to finish

Character-Handling Library

- The character-handling library (`<ctype.h>`) includes several functions that perform useful tests and manipulations of character data.
  - Each function receives a character as an argument.

<cctype> (ctype.h) : http://www.cplusplus.com/reference/cctype/

Manipulation in `<ctype.h>`

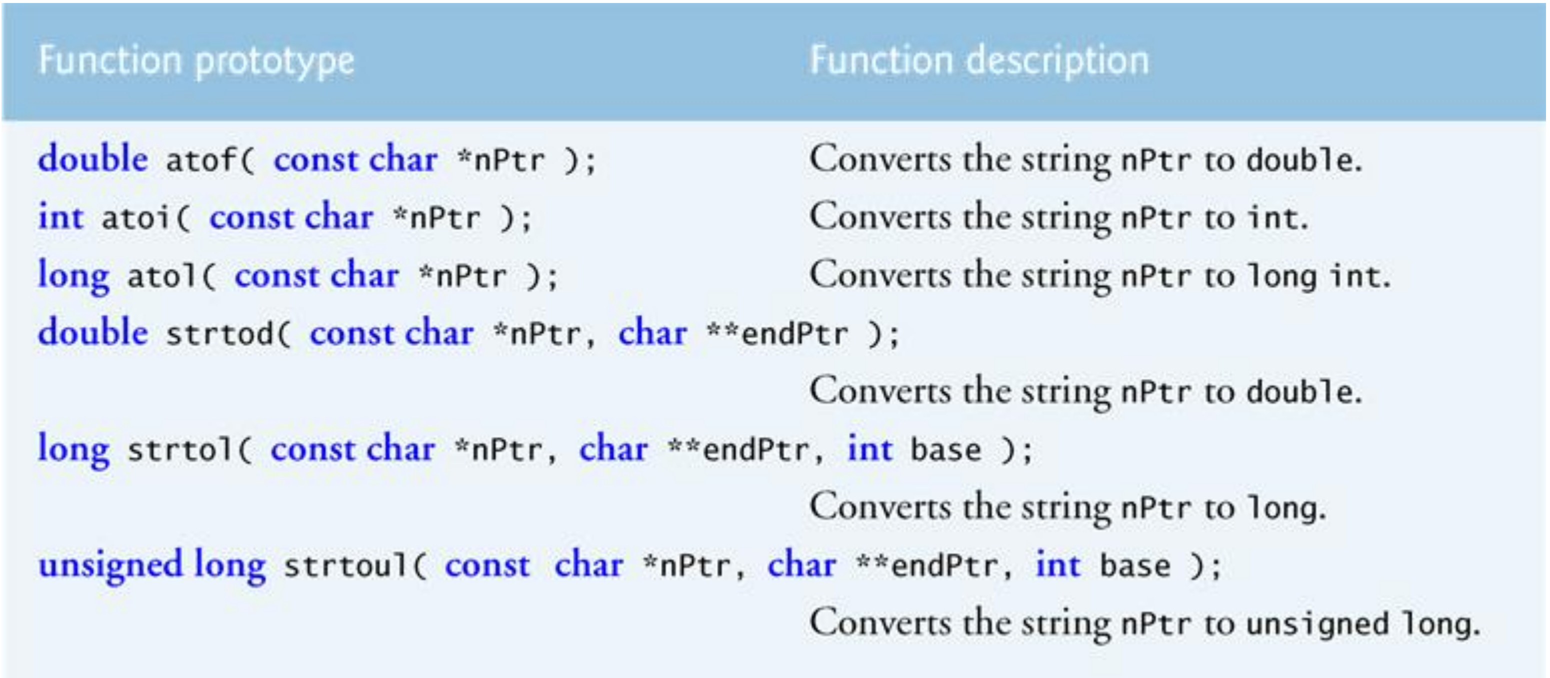
- `int toupper(int)`
  - return uppercase of a character
  - Ex: `char c = toupper('a'); //c = 'A'`
- `int tolower(int)`
  - return lowercase of a character
  - Ex: `char c = tolower('B'); //c = 'b'`
- `int isupper(int)`
  - true if the character is uppercase
  - Ex: `int cond = isupper('A'); //cond = 1`
- `int islower(int)`
  - true if the character is lowercase
  - Ex: `int cond = islower('A'); //cond = 0`

Manipulation in `<ctype.h>` (cont.)

- `int isalpha(int)`
  - Check if character is alphabetic
  - Ex: `int cond = isalpha('a'); //cond = 1`
- `int isdigit(int)`
  - Check if character is a digit from '0' to ' 9'
  - Ex: `int cond = isdigit('M');//cond = 0`
- `int isalnum(int)`
  - Check if character is either a letter or a digit
  - Ex: `int cond = isalnum('A'); //cond = 1`
- `int isspace(int)`
  - Check if character is a white-space
  - Ex: `int cond = isspace('A'); //cond = 0`

String-Conversion Functions

- The string-conversion functions is from the general utilities library (`<stdlib.h>`) which convert strings of digits to integer and floating-point values.



Function prototype	Function description
`double atof( const char *nPtr );`	Converts the string `nPtr` to `double`.
`int atoi( const char *nPtr );`	Converts the string `nPtr` to `int`.
`long atol( const char *nPtr );`	Converts the string `nPtr` to `long int`.
`double strtod( const char *nPtr, char **endPtr );`	Converts the string `nPtr` to `double`.
`long strtol( const char *nPtr, char **endPtr, int base );`	Converts the string `nPtr` to `long`.
`unsigned long strtoul( const char *nPtr, char **endPtr, int base );`	Converts the string `nPtr` to `unsigned long`.

<cstdlib> (stdlib.h) : http://www.cplusplus.com/reference/cstdlib/

Function `atof`

```c
// Fig 8.6: fig08_06.c
// Using atof.
#include <stdio.h>
#include <stdlib.h>

int main ()
{
    double d;  // variable to hold converted string

    d = atof("99.0");
    printf("%s%.3f\n%s%.3f\n",
           "The string \"99.0\" converted to double is ",
           d, "The converted value divided by 2 is ",
           d/2.0 );
    return 0;
}
```

Function `atof` (cont.)

- screen output

```
The string "99.0" converted to double is 99.000
The converted value divided by 2 is 49.500
```

Function atoi

```
// Fig 8.7: fig08_07.c
// Using atoi.
#include <stdio.h>
#include <stdlib.h>

int main ()
{
    int i;  // variable to hold converted string

    i = atoi("2593");
    printf("%s%d\n%s%d\n",
           "The string \"2593\" converted to integer is ",
           i, "The converted value minus 593 is ",
           i - 593 );
    return 0;
}
```

Function `atoi` (cont.)

- screen output

```
The string "2593" converted to integer is 2593
The converted value minus 593 is 2000
```

Function `strtod`

```c
// Fig 8.9: fig08_09.c
// Using strtod.
#include <stdio.h>
#include <stdlib.h>

int main ()
{
    const char *string = "51.2% are admitted";
    double d;  // variable to hold converted sequence
    char *stringPtr;
    d = strtod( string, &stringPtr);
    printf("The string \"%s\" is converted to the\n",
           string);
    printf("double value %.2f and the string \"%s\"\n",
           d, stringPtr);
    return 0;
}
```

Function `strtod` (cont.)

- screen output

```
The string "51.2% are admitted" is converted to the
double value 51.20 and the string "% are admitted"
```

Function strtol

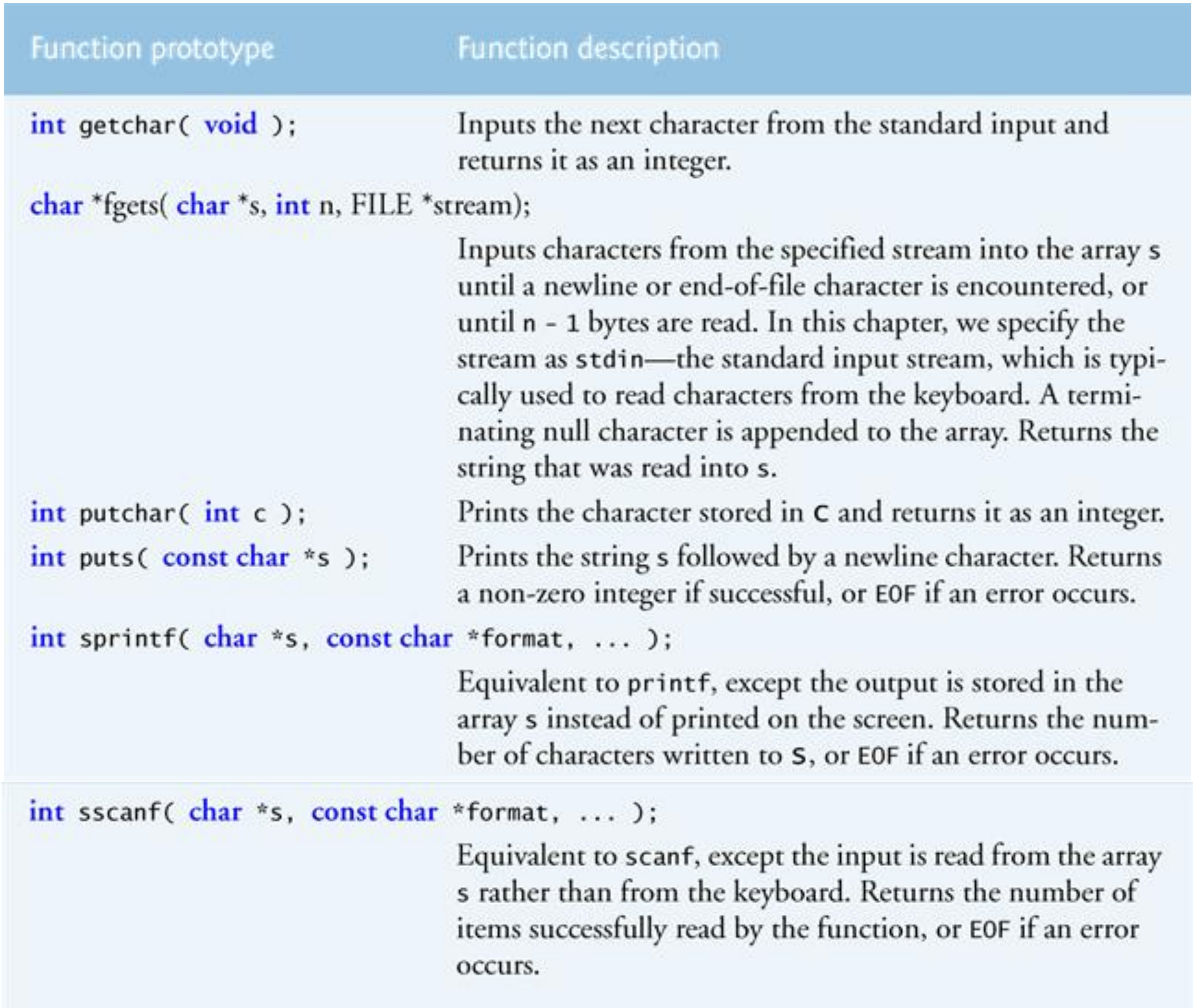
```
// Fig 8.10: fig08_10.c
// Using strtol.
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    const char *str = "-1234567abc";
    long x;  // variable to hold converted sequence
    char *remainderPtr;
    x = strtol( string, &remainderPtr, 0 );
    printf("%s\"%s\"\n%s%ld\n%s\"%s\"\n%s%ld\n",
           "The original string is ", string,
           "The converted value is ", x,
           "The remainder of the original string is ",
           remainderPtr,
           "The converted value plus 567 is ", x + 567);
    return 0;
}
```

Function `strtol` (cont.)

- screen output

```
The original string is "-1234567abc"
The converted value is -1234567
The remainder of the original string is "abc"
The converted value plus 567 is -1234000
```

Standard Input/Output Library Functions



Function prototype	Function description
int getchar(void);	Inputs the next character from the standard input and returns it as an integer.
char *fgets(char *s, int n, FILE *stream);	Inputs characters from the specified stream into the array s until a newline or end-of-file character is encountered, or until n - 1 bytes are read. In this chapter, we specify the stream as stdin—the standard input stream, which is typically used to read characters from the keyboard. A terminating null character is appended to the array. Returns the string that was read into s.
int putchar(int c);	Prints the character stored in c and returns it as an integer.
int puts(const char *s);	Prints the string s followed by a newline character. Returns a non-zero integer if successful, or EOF if an error occurs.
int sprintf(char *s, const char *format, ...);	Equivalent to printf, except the output is stored in the array s instead of printed on the screen. Returns the number of characters written to s, or EOF if an error occurs.
int sscanf(char *s, const char *format, ...);	Equivalent to scanf, except the input is read from the array s rather than from the keyboard. Returns the number of items successfully read by the function, or EOF if an error occurs.

Functions fgets and putchar

```
// Fig 8.13: fig08_13.c
// Using fgets and putchar.
#include <stdio.h>
void reverse( const char * const sPtr);
int main ()
{
    char sentence[80];
    printf("Enter a line of text: \n");
    // use fgets to read line of text
    fgets(sentence, 80, stdin);
    printf("\nThe line printed backward is:\n");
    reverse(sentence);

    return 0;
}
```

Functions fgets and putchar (cont.)

```
void reverse( const char * const sPtr)
{
    if (sPtr[0] == '\0') // base case
        return;
    else {
        reverse(&sPtr[1]); // recursion step
        // use putchar to display character
        putchar(sPtr[0]);
    }
}
```

- screen output

```
Enter a line of text:
Characters and Strings

The line printed backward is
sgngrtS dna sretcarahC
```

Functions getchar and puts

```
// Fig 8.14: fig08_14.c
// Using getchar and puts.
#include <stdio.h>
int main ()
{
    char c;
    char sentence[80];
    int i = 0;
    puts("Enter a line of text:");
    // use getchar to read each character
    while (( c = getchar() ) != '\n')
        sentence[i++] = c;
    sentence[i] = '\0';
    // use puts to display sentence
    puts("\nThe line entered was:");
    puts(sentence);
    return 0;
}
```

Functions `getchar` and `puts` (cont.)

- screen output

```
Enter a line of text:
This is a test.

The line entered was:
This is a test.
```

Functions sprintf

```c
// Fig 8.15: fig08_15.c
// Using sprintf.
#include <stdio.h>
int main ()
{
    char s[80];
    int i;
    double y;
    printf("Enter an integer and a double:\n");
    scanf("%d%lf", &x, &y);
    sprintf(s, "integer:%6d\ndouble:%8.2f", x, y);
    printf("%s\n%s\n", "The formatted output stored in
            array s is: ", s);

    return 0;
}
```

Functions `sprintf` (cont.)

- screen output

```
Enter an integer and a double:
298 87.375
The formatted output stored in array s is:
integer:   298
double:   87.38
```

Functions sscanf

```
// Fig 8.16: fig08_16.c
// Using sscanf.
#include <stdio.h>
int main ()
{
    char s[] = "31298 87.375";
    int x;
    double y;
    sscanf(s, "%d%lf", &x, &y);
    printf("%s\n%s%6d\n%s%8.3f\n",
           "The value stored in character array s are:",
           "integer:", x, "double:", y);

    return 0;
}
```

Functions sscanf (cont.)

- screen output

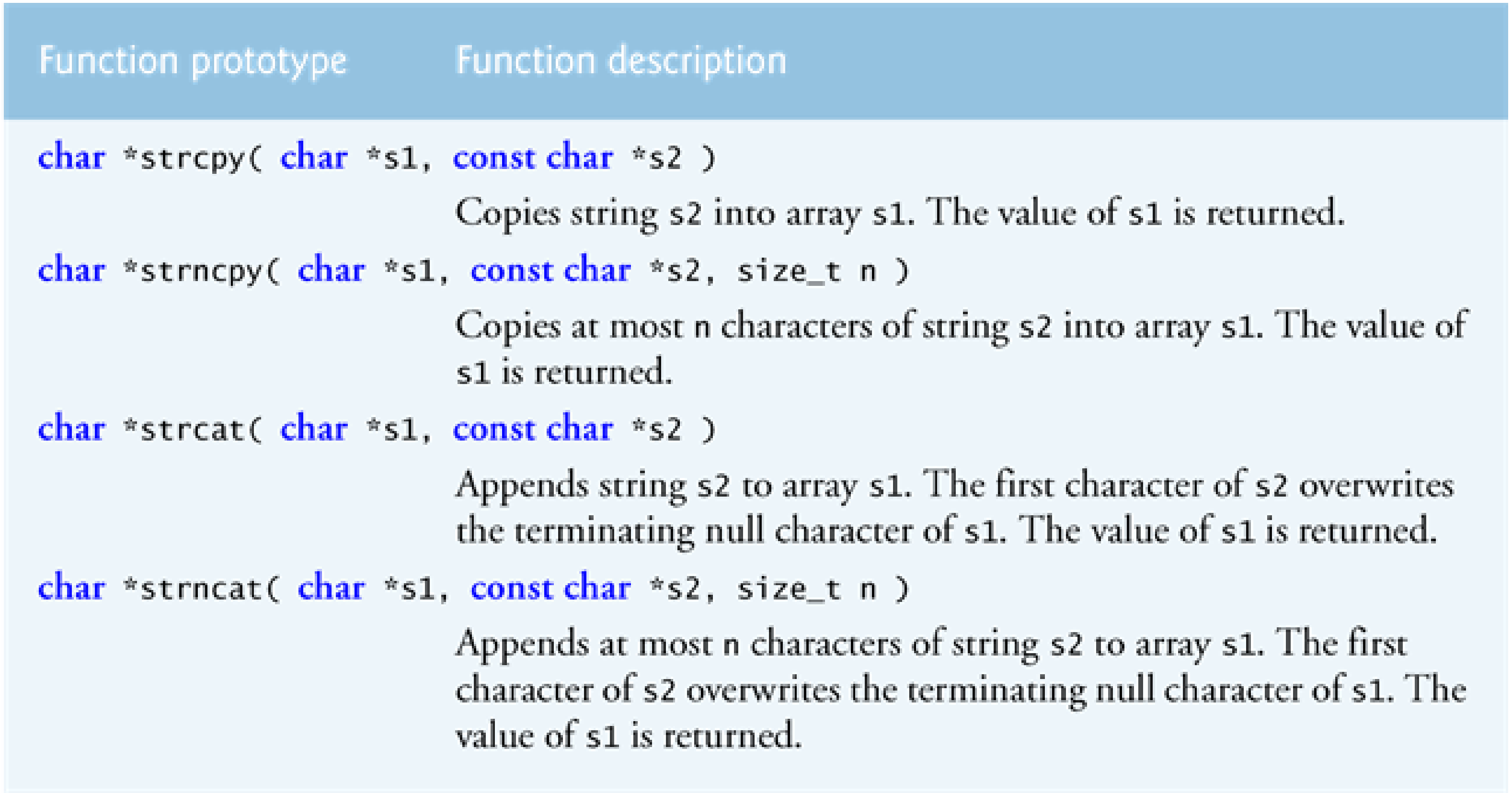
```
The value stored in character array s are:
integer: 31298
double:  87.375
```

String-Handling Library

- The string-handling library `(<string.h>)` provides useful functions for manipulating string data (copying strings and concatenating strings), comparing strings, searching strings for characters and other strings, tokenizing strings (separating strings into logical pieces) and determining the length of strings

<cstring> (string.h) : http://www.cplusplus.com/reference/cstring/

String-Manipulation Functions



Function prototype	Function description
`char *strcpy( char *s1, const char *s2 )`	Copies string s2 into array s1. The value of s1 is returned.
`char *strncpy( char *s1, const char *s2, size_t n )`	Copies at most n characters of string s2 into array s1. The value of s1 is returned.
`char *strcat( char *s1, const char *s2 )`	Appends string s2 to array s1. The first character of s2 overwrites the terminating null character of s1. The value of s1 is returned.
`char *strncat( char *s1, const char *s2, size_t n )`	Appends at most n characters of string s2 to array s1. The first character of s2 overwrites the terminating null character of s1. The value of s1 is returned.

Functions `strcpy` and `strncpy`

```c
// Fig 8.18: fig08_18.c
// Using strcpy and strncpy.
#include <stdio.h>
#include <string.h>
int main ()
{
    char x[] = "Happy Birthday to You";
    char y[25];
    char z[15];
    printf("%s%s\n%s%s\n",
           "The string in array x is: ", x,
           "The string in array y is: ", strcpy(y, x));
    strncpy(z, x, 14); // copy first 14 characters of x
    z[14] = '\0';
    printf("The string in array z is: %s\n", z);
    return 0;
}
```

Functions `strcpy` and `strncpy` (cont.)

- screen output

```
The string in array x is: Happy Birthday to You
The string in array y is: Happy Birthday to You
The string in array z is: Happy Birthday
```

Functions strcat and strncat

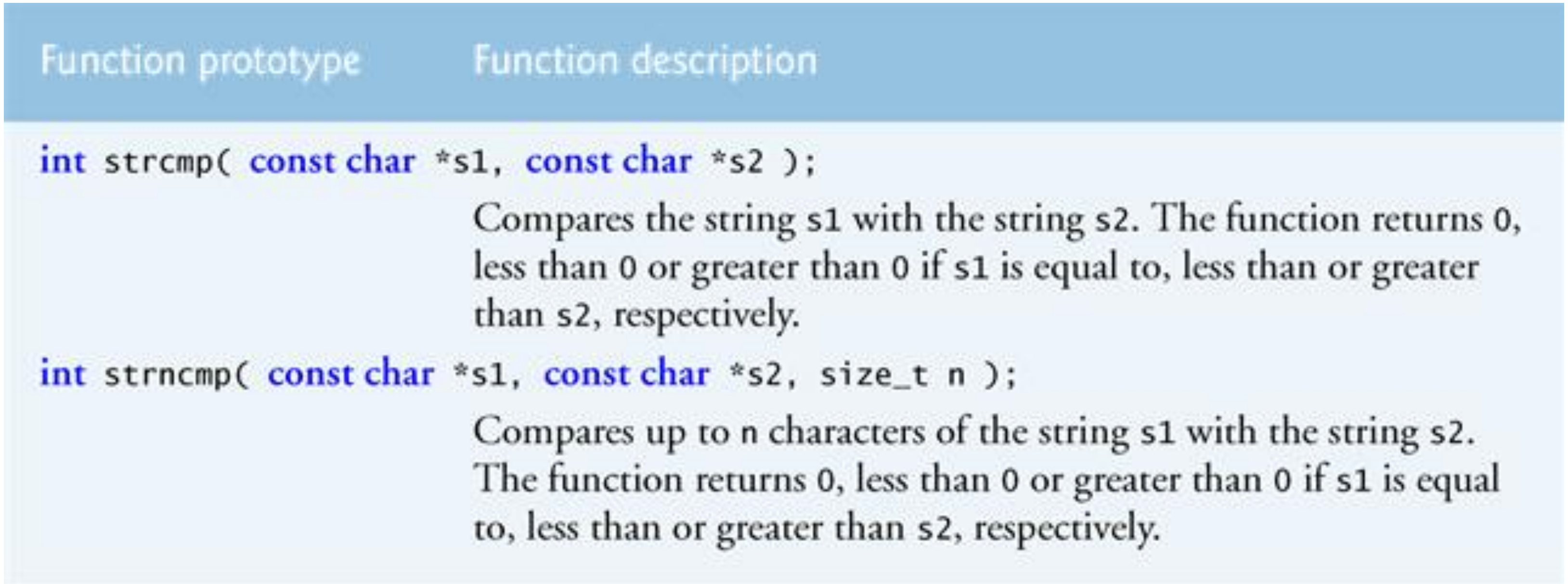
```c
// Fig 8.19: fig08_19.c
// Using strcat and strncat.
#include <stdio.h>
#include <string.h>
int main ()
{
    char s1[20] = "Happy ";
    char s2[] = "New Year ";
    char s3[40] = "";
    printf("s1 = %s\ns2 = %s\n", s1, s2);
    printf("strcat( s1, s2) = %s\n", strcat(s1,s2));
    printf("strncat( s3, s1, 6) = %s\n",
            strncat(s3, s1,6));
    printf("strcat( s3, s1 ) = %s\n", strcat(s3, s1));

    return 0;
}
```

Functions `strcat` and `strncat` (cont.)

- screen output

```
s1 = Happy
s2 = New Year
strcat( s1, s2 ) = Happy New Year
strncat( s3, s1, 6) = Happy
strcat( s3, s1 ) = Happy Happy New Year
```

Comparison Functions



Function prototype	Function description
`int strcmp( const char *s1, const char *s2 );`	Compares the string s1 with the string s2. The function returns 0, less than 0 or greater than 0 if s1 is equal to, less than or greater than s2, respectively.
`int strncmp( const char *s1, const char *s2, size_t n );`	Compares up to n characters of the string s1 with the string s2. The function returns 0, less than 0 or greater than 0 if s1 is equal to, less than or greater than s2, respectively.

Compare C-String by `strcmp`

- String cannot use operator ==

```
char mystr_a[10] = "Hello";
char mystr_b[10] = "Goodbye";
mystr_a == mystr_b; // NOT allowed!
```

- Must use library function `strcmp(cstr1,cstr2)`
  - Ex:

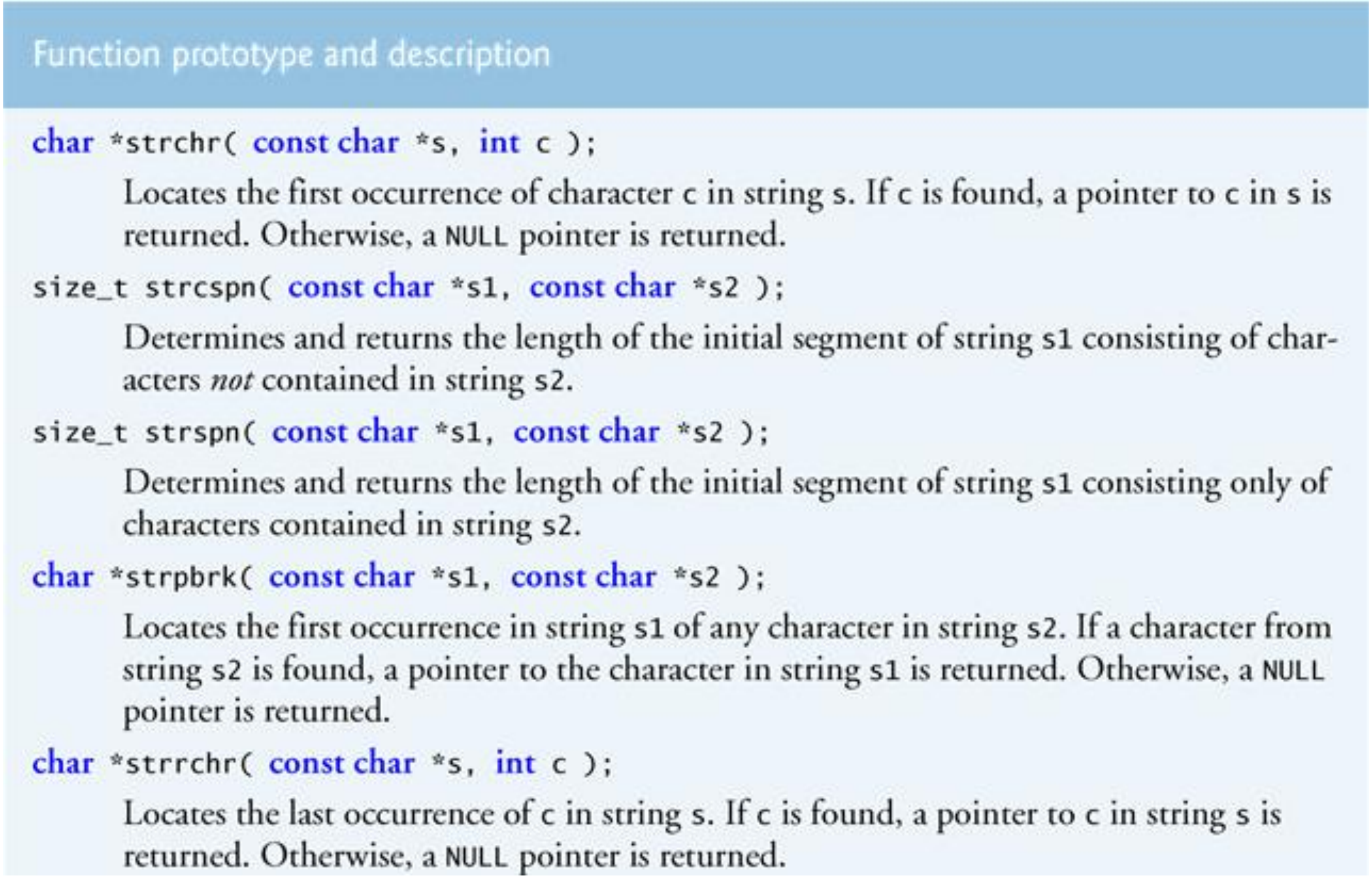
```
if (strcmp(mystr_a, mystr_b))
    printf("Strings are NOT the same.");
else
    printf("Strings are the same.");
```

Reading Results from `strcmp`

- C-Strings are compared character by character using the collating sequence of the system
- If we are using the ASCII character set

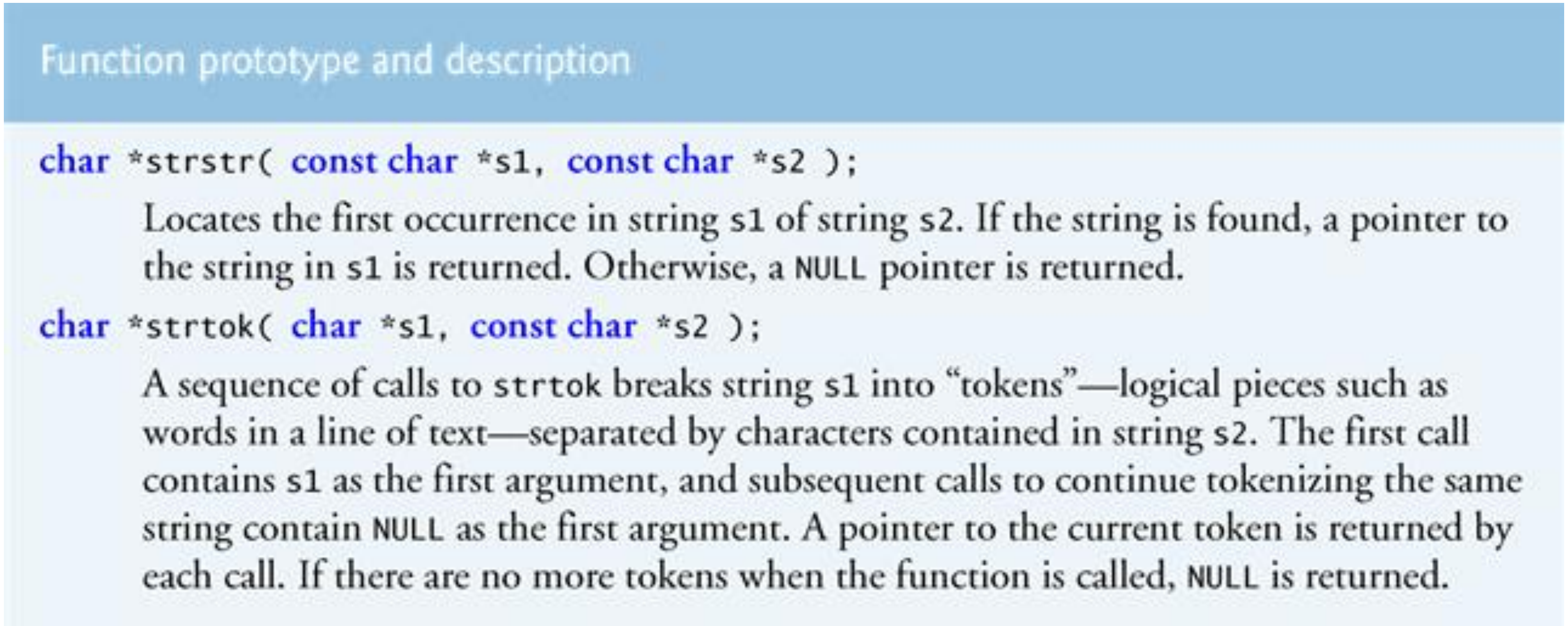
1. The string "`Air`" is smaller than the string "`Boat`" => ∵ '`B`' > '`A`'
2. The string "`Air`" is smaller than the string "`An`" => ∵ '`n`' > '`i`'
3. The string "`Billy`" is larger than the string "`Bill`" => ∵ '`y`' > '`\0`'
4. The string "`Hello`" is smaller than "`hello`" => ∵ '`h`' > '`H`'

Search Functions



Function prototype and description
`char *strchr( const char *s, int c );` Locates the first occurrence of character `c` in string `s`. If `c` is found, a pointer to `c` in `s` is returned. Otherwise, a `NULL` pointer is returned.
`size_t strcspn( const char *s1, const char *s2 );` Determines and returns the length of the initial segment of string `s1` consisting of characters *not* contained in string `s2`.
`size_t strspn( const char *s1, const char *s2 );` Determines and returns the length of the initial segment of string `s1` consisting only of characters contained in string `s2`.
`char *strpbrk( const char *s1, const char *s2 );` Locates the first occurrence in string `s1` of any character in string `s2`. If a character from string `s2` is found, a pointer to the character in string `s1` is returned. Otherwise, a `NULL` pointer is returned.
`char *strrchr( const char *s, int c );` Locates the last occurrence of `c` in string `s`. If `c` is found, a pointer to `c` in string `s` is returned. Otherwise, a `NULL` pointer is returned.

Search Functions (cont.)



Function prototype and description
char *strstr(const char *s1, const char *s2);
Locates the first occurrence in string s1 of string s2. If the string is found, a pointer to the string in s1 is returned. Otherwise, a NULL pointer is returned.
char *strtok(char *s1, const char *s2);
A sequence of calls to strtok breaks string s1 into "tokens"—logical pieces such as words in a line of text—separated by characters contained in string s2. The first call contains s1 as the first argument, and subsequent calls to continue tokenizing the same string contain NULL as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, NULL is returned.

Functions strchr

```
// Fig 8.23: fig08_23.c
// Using strchr.
#include <stdio.h>
#include <string.h>

int main ()
{
    const char *string = "This is a test";
    char character = 'a';
    if (strchr(string, character) != NULL)
        printf ("\'%c\' was found in \"%s\".\n",
                character, string);
    else
        printf ("\'%c\' was not found in \"%s\".\n",
                character, string);

    return 0;
}
```

Functions `strchr` (cont.)

- screen output

```
'a' was found in "This is a test"
```

Functions strcspn

```
// Fig 8.24: fig08_24.c
// Using strcspn.
#include <stdio.h>
#include <string.h>

int main ()
{
    const char *string1 = "The value is 3.14159";
    const char *string2 = "1234567890";

    printf("%s%s\n%s%s\n\n%s\n%s%u\n",
           "string1 = ", string1, "string2 = ", string2,
           "The length of the initial segment of string1",
           "containing no characters from string2 = ",
           strcspn(string1, string2);

    return 0;
}
```

Functions strcspn (cont.)

- screen output

```
string1 = The value is 3.14159
string2 = 1234567890

The length of the initial segment of string1
containing no characters from string2 = 13
```

Functions strpbrk

```
// Fig 8.24: fig08_24.c
// Using strpbrk.
#include <stdio.h>
#include <string.h>

int main ()
{
    const char *string1 = "This is a test";
    const char *string2 = "beware";

    printf("%s\"%s\"\n\'%c\'%s\n\"%s\"\n",
           "Of the characters in ", string2,
           *strpbrk(string1, string2),
           " appears earliest in ", string1);

    return 0;
}
```

Functions strpbrk (cont.)

- screen output

```
Of the characters in "beware"
'a' appears earliest in
"This is a test"
```

Functions strstr

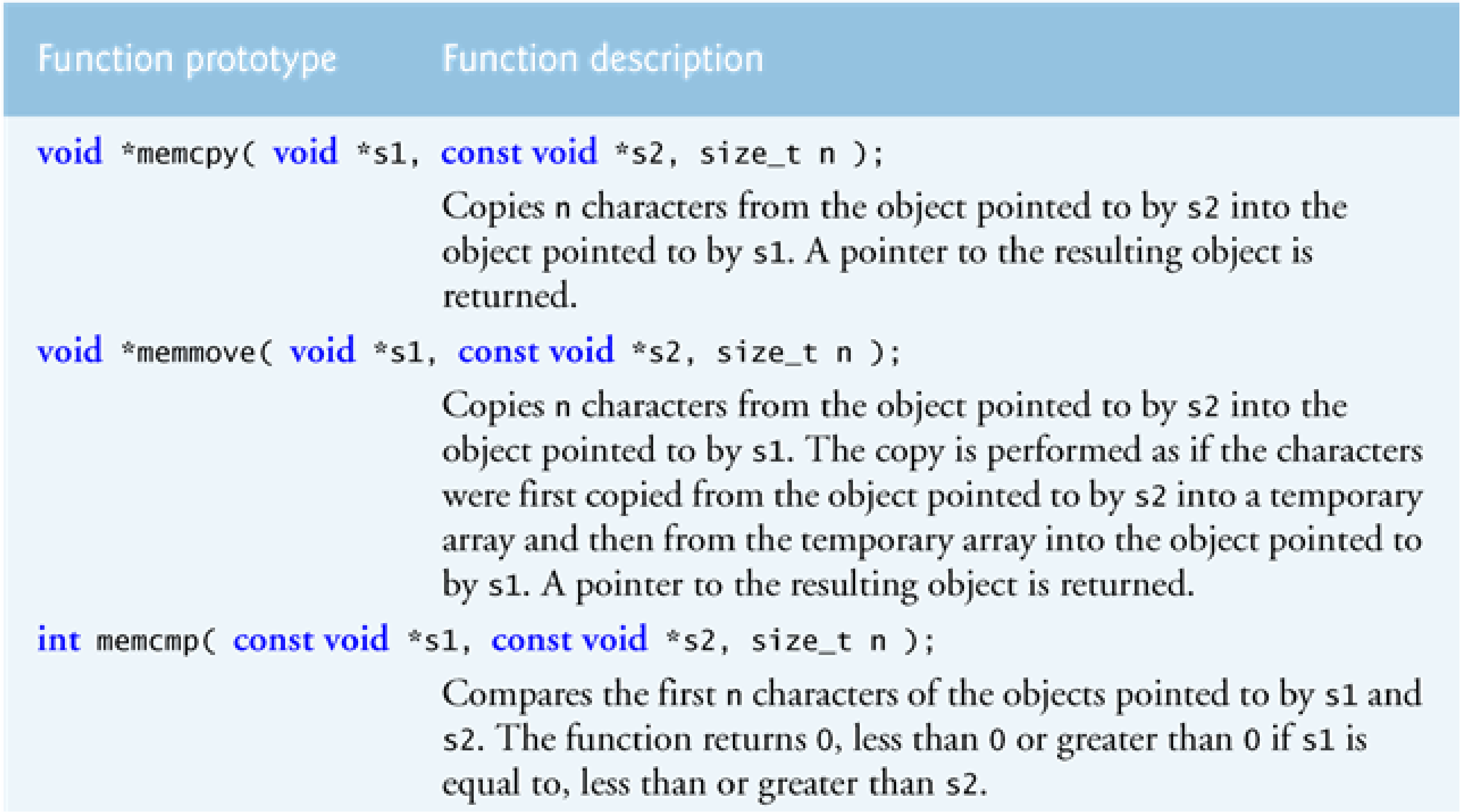
```c
// Fig 8.28: fig08_28.c
// Using strstr.
#include <stdio.h>
#include <string.h>

int main ()
{
    const char *string1 = "abcdefabcdef";
    const char *string2 = "def";

    printf("%s%s\n%s%s\n\n%s\n%s%u\n",
           "string1 = ", string1, "string2 = ", string2,
           "The remainder of string1 beginning with the",
           "first occurrence of string2 is: ",
           strstr(string1, string2);

    return 0;
}
```

Functions `strstr` (cont.)

- screen output

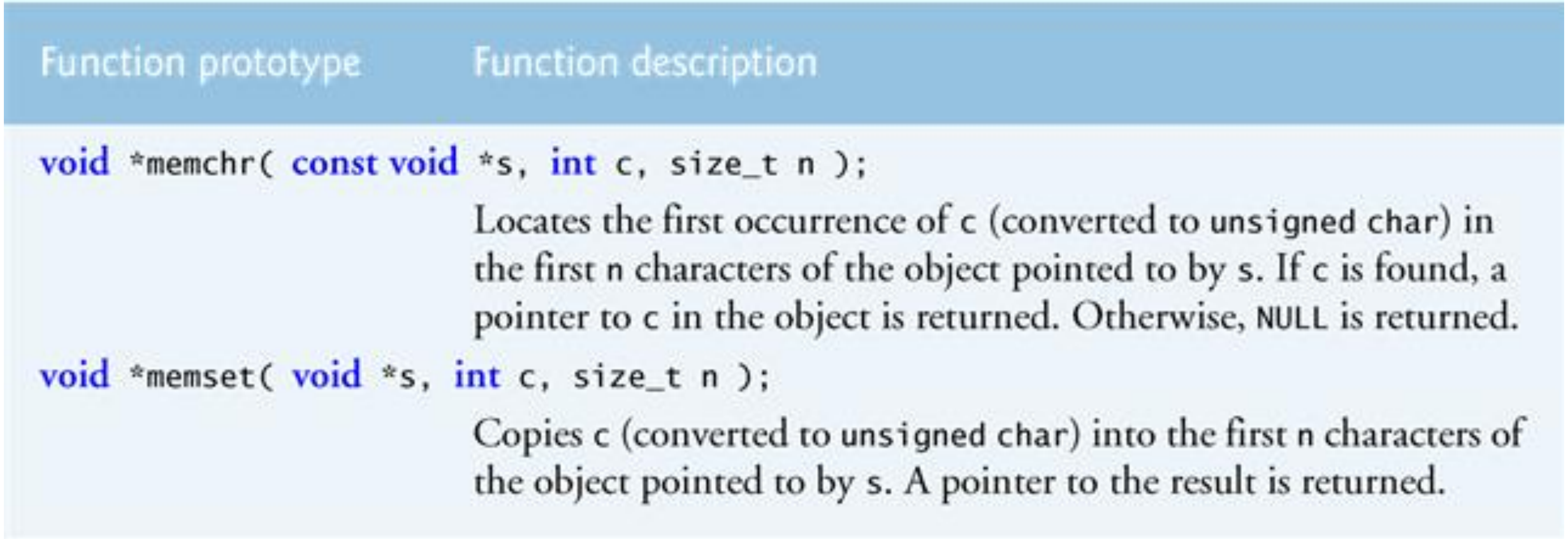
```
string1 = abcdefabcdef
string2 = def

The remainder of string1 beginning with the
first occurrence of string2 is: defabcdef
```

Memory Functions



Function prototype	Function description
`void *memcpy( void *s1, const void *s2, size_t n );`	Copies `n` characters from the object pointed to by `s2` into the object pointed to by `s1`. A pointer to the resulting object is returned.
`void *memmove( void *s1, const void *s2, size_t n );`	Copies `n` characters from the object pointed to by `s2` into the object pointed to by `s1`. The copy is performed as if the characters were first copied from the object pointed to by `s2` into a temporary array and then from the temporary array into the object pointed to by `s1`. A pointer to the resulting object is returned.
`int memcmp( const void *s1, const void *s2, size_t n );`	Compares the first `n` characters of the objects pointed to by `s1` and `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`.

Memory Functions (cont.)



Function prototype	Function description
`void *memchr( const void *s, int c, size_t n );`	Locates the first occurrence of `c` (converted to `unsigned char`) in the first `n` characters of the object pointed to by `s`. If `c` is found, a pointer to `c` in the object is returned. Otherwise, `NULL` is returned.
`void *memset( void *s, int c, size_t n );`	Copies `c` (converted to `unsigned char`) into the first `n` characters of the object pointed to by `s`. A pointer to the result is returned.

Functions memcpy

```c
// Fig 8.31: fig08_31.c
// Using memcpy.
#include <stdio.h>
#include <string.h>

int main ()
{
    char s1[17];
    char s2[] = "Copy this string";

    memcpy(s1, s2, 17);
    printf("%s\n%s\"%s\"\n",
           "After s2 is copied into s1 with memcpy,",
           "s1 contains ", s1);

    return 0;
}
```

Functions `memcpy`(cont.)

- screen output

```
After s2 is copied into s1 with memcpy,
S1 contains "Copy this string"
```

Functions memchr

```c
// Fig 8.34: fig08_34.c
// Using memchr.
#include <stdio.h>
#include <string.h>

int main ()
{
    const char *s = "This is a string";

    printf("%s\'%c\'%s\"%s\"\n",
           "The remainder of s after character ", 'r',
           " is found is ", memchr(s, 'r', 16) );

    return 0;
}
```

Functions `memchr`(cont.)

- screen output

```
The remainder of s after character 'r' is found is "ring"
```

Functions memset

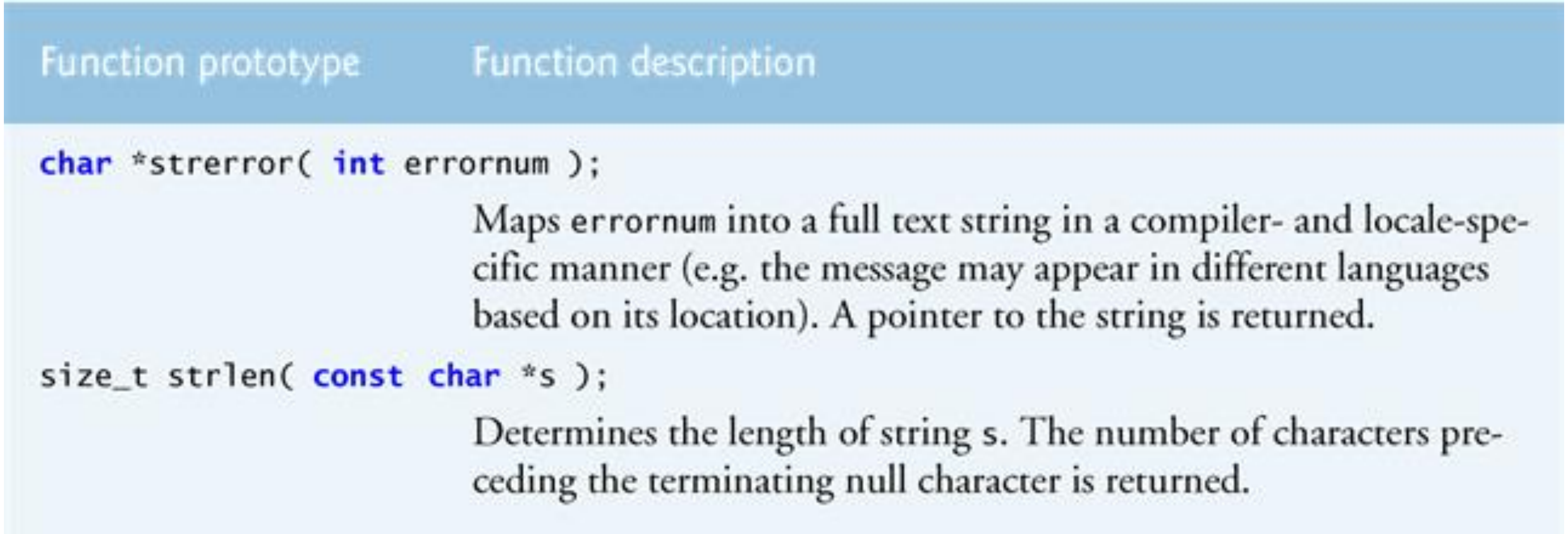
```
// Fig 8.35: fig08_35.c
// Using memset.
#include <stdio.h>
#include <string.h>

int main ()
{
    char string1[15] = "BBBBBBBBBBBBBB";

    printf("string1 = %s\n", string1);
    printf("string1 after memset = %s\n",
           memset(string1, 'b', 7);

    return 0;
}
```

Functions memset(cont.)

- screen output

```
string1 = BBBBBBBBBBBBBB
string1 after memset = bbbbbbbBBBBBBB
```

Other Functions



Function prototype	Function description
`char *strerror( int errornum );`	Maps `errornum` into a full text string in a compiler- and locale-specific manner (e.g. the message may appear in different languages based on its location). A pointer to the string is returned.
`size_t strlen( const char *s );`	Determines the length of string `s`. The number of characters preceding the terminating null character is returned.

main() Function

```
int main(int argc, char *argv[]);
```

- where the program starts
- An int is returned indicate the result status; typically return 0
- argc gives the number of arguments passed to the program
- argv contains those arguments

- Ex: `>./prog 5 4.4 test1.txt`
  - argc = 4
  - argv[0] is "prog", argv[1] is "5", argv[2]is "4.4" and argv[3] is "test1.txt"

Summary

- C String variable is "array of characters"
    - with addition of null character, '\o'
- C Strings act like arrays
    - cannot assign, compare like simple variables
- Libraries `<ctype.h>`, `<string.h>` & `<stdio.h>` have useful manipulating functions.