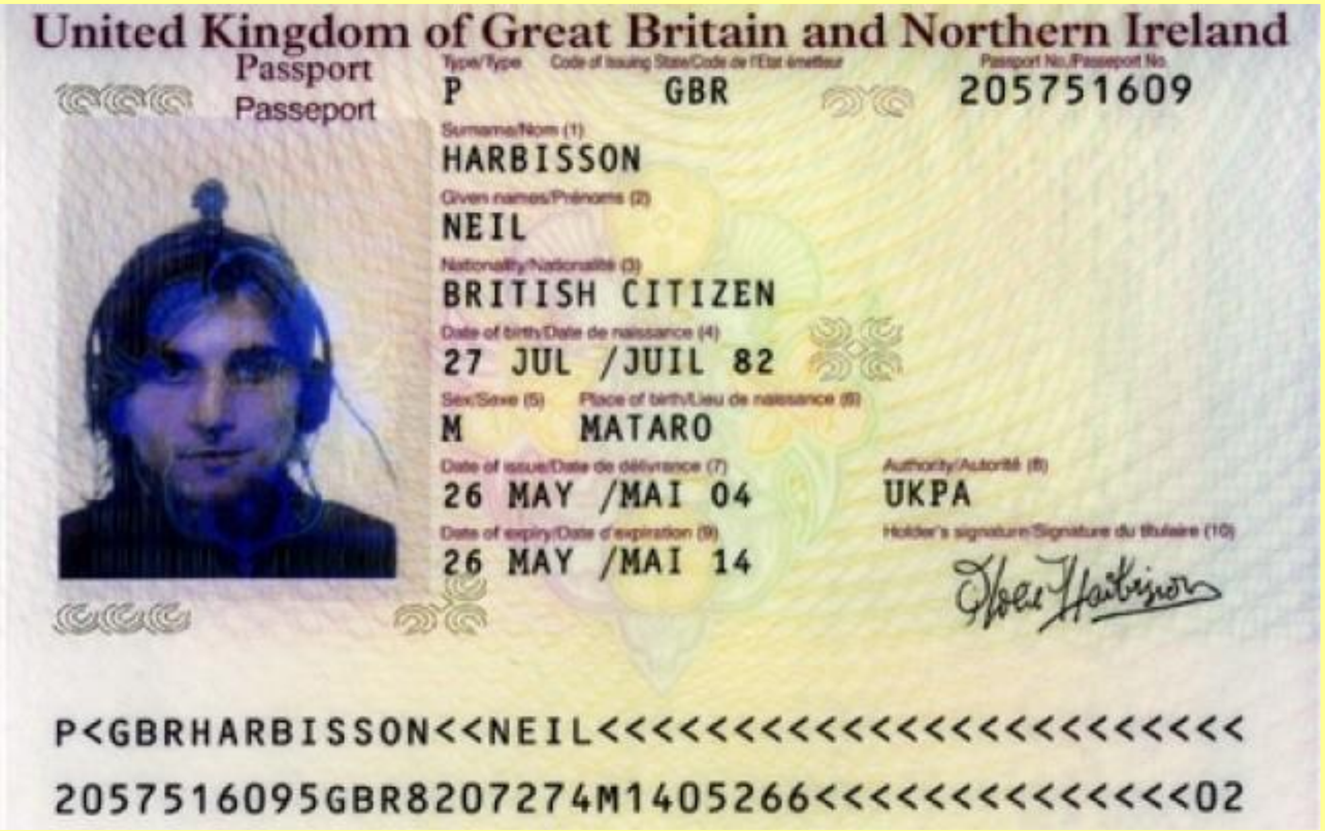
DASF004
Basic and Practice in Programming

Lecture 10
Character and String

Neil Harbisson

- First person to have an antenna implanted on a skull
- Formally being recognized by the British Government as a cyborg



United Kingdom of Great Britain and Northern Ireland

Passport

Passeport

Type/Type P

Code of issuing State/Code de l'Etat émetteur GBR

Passport No./Passeport No. 205751609

Surname/Nom (1)
HARBISSON

Given names/Prénoms (2)
NEIL

Nationality/Nationalité (3)
BRITISH CITIZEN

Date of birth/Date de naissance (4)
27 JUL /JUIL 82

Sex/Sexe (5)
M

Place of birth/Lieu de naissance (6)
MATARO

Date of issue/Date de délivrance (7)
26 MAY /MAI 04

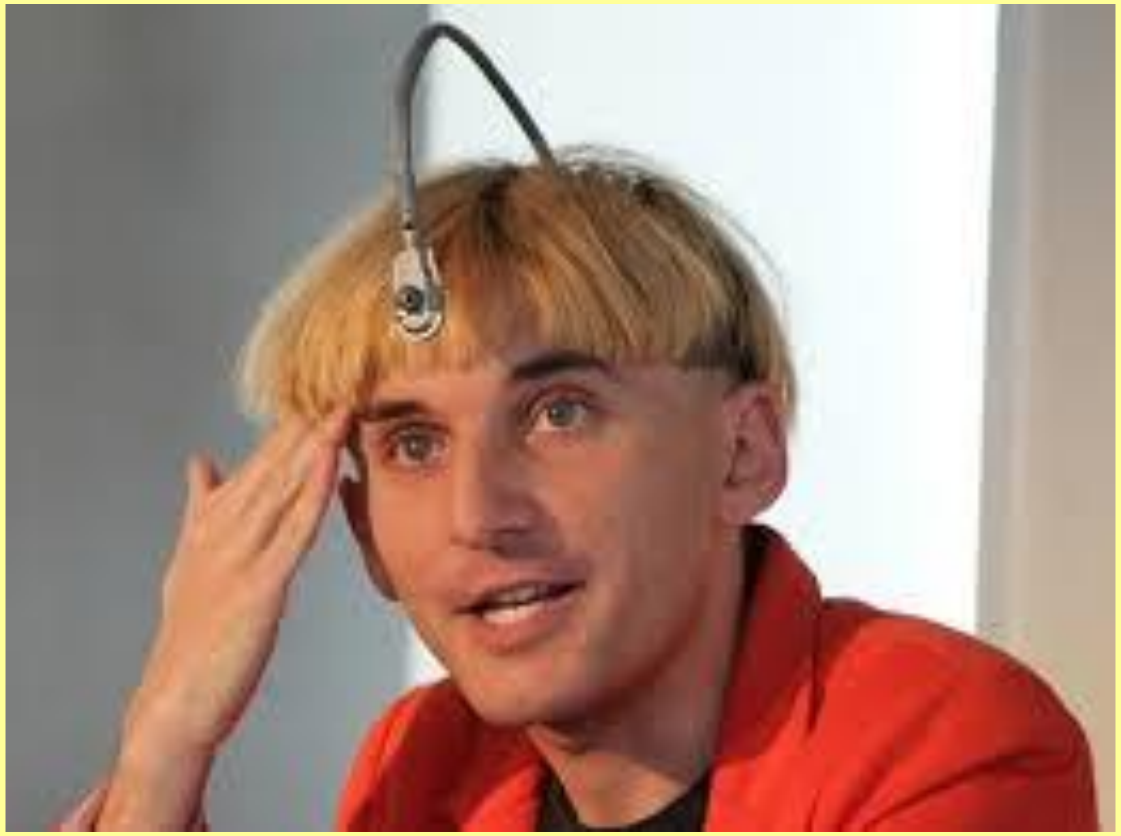
Authority/Autorité (8)
UKPA

Date of expiry/Date d'expiration (9)
26 MAY /MAI 14

Holder's signature/Signature du titulaire (10)

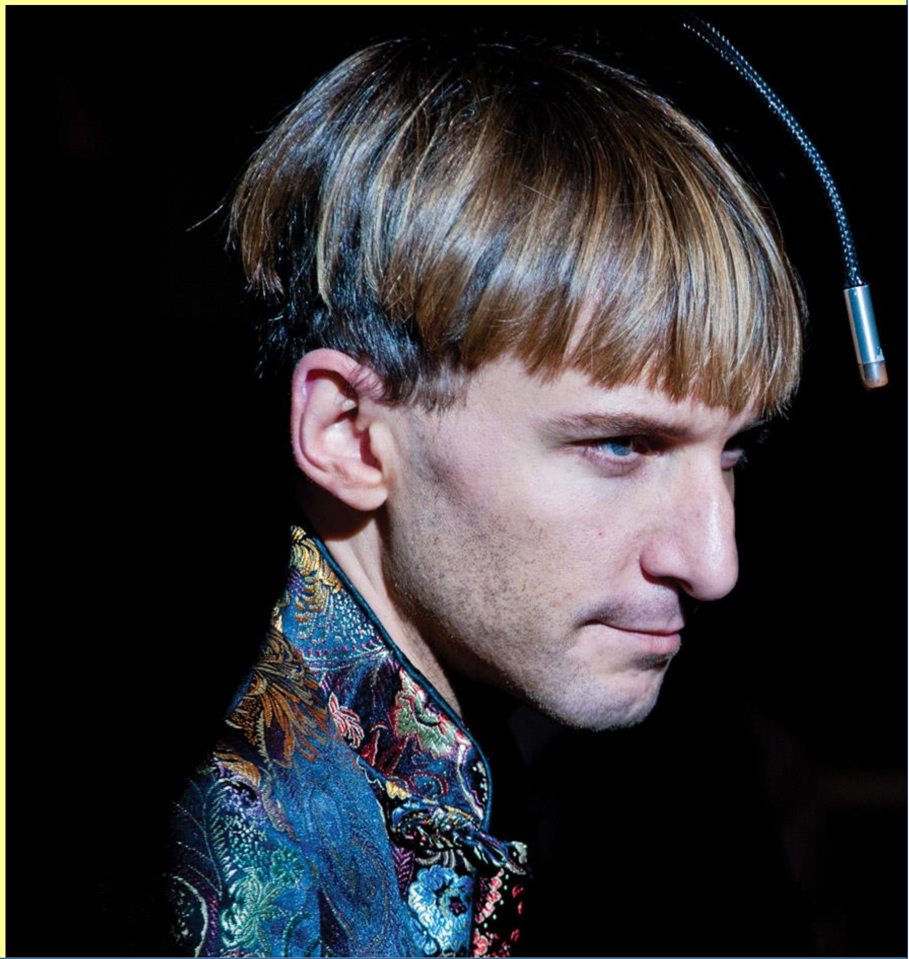
P<GBRHARBISSON<<NEIL<<<<<<<<<<<<<<<<<<<<<<<<

2057516095GBR8207274M1405266<<<<<<<<<<<<<<02



Neil Harbisson

- Born with an extreme color blindness (achromatopsia)
  - He can only see in gray scale
- Since the age of 21 (2003), he started to “hear” color
- https://www.ted.com/talks/neil_harbisson_i_listen_to_color#t-32984



Agenda

Character and String

Character

Character and String

- Character is a 8-bit number
  - e.g. 00101101 = 45
- The number represent characters through a mapping table
- ASCII is the most commonly used mapping table
  - e.g. 00101101 = 45 = “-” in ASCII
  - Other commonly used table include EBCDIC

ASCII Table

Right	*ASCII*									
Left Digit(s) / *Digit*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	□	!	“	#	$	%	&	’
4	(	)	*	+	,	–	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

String

- String is an array of characters ending in the null character (`'\0'`).
- A string is accessed via a *pointer* to the first character in the string.
- The value of a string is the address of its first character.
- Thus, in C, it’s appropriate to say that a string is a pointer—in fact, a pointer to the string’s first character.
- In this sense, strings are like arrays, because an array is also a pointer to its first element.
- A *character array* or a *variable of type* *`char *`* can be initialized with a string in a definition.

String

The definitions

```
char color[] = "blue";
const char *colorPtr = "blue";
char color[] = { 'b', 'l', 'u', 'e', '\0' };
```

each initialize a variable to the string `"blue"`.

The above three lines of code are identical

These definition creates a 5-element array `color` containing the characters `'b'`, `'l'`, `'u'`, `'e'` and `'\0'`.

When you use the quotation to initialize the array (e.g. “blue”), a ‘\0’ is automatically attached as the last character of the array.

The first and second code definition automatically determines the size of the array based on the number of initializers in the initializer list.

The `scanf()` function

`scanf()` will read characters until a space, tab, newline or end-of-file indicator is encountered.

`scanf()` will will also automatically attach a ‘\0’ at the end of the array.

Consider the following code segment:

```
    char input[10];
    scanf(“%s”,input);
    printf(“%s”,input);
```

If user input is “`abc`”, the code segment will generate output of:

```
abc
```

The value of the string `input` will be ″`abc\0`″.

If user input is “`abcd defg`”, the code segment will generate output of:

```
abcd
```

The value of the string `input`will be ″`abcd\0`″.

The `scanf()` function

You can specify how many characters `scanf()` is reading in

```
scanf(“%9s”,input);
```

Consider the following code segment:

```
char input[10];
scanf(“%9s”,input);
printf(“%s”,input);
```

If user input is “`abc`”, the code segment will generate output of:

```
abc
```

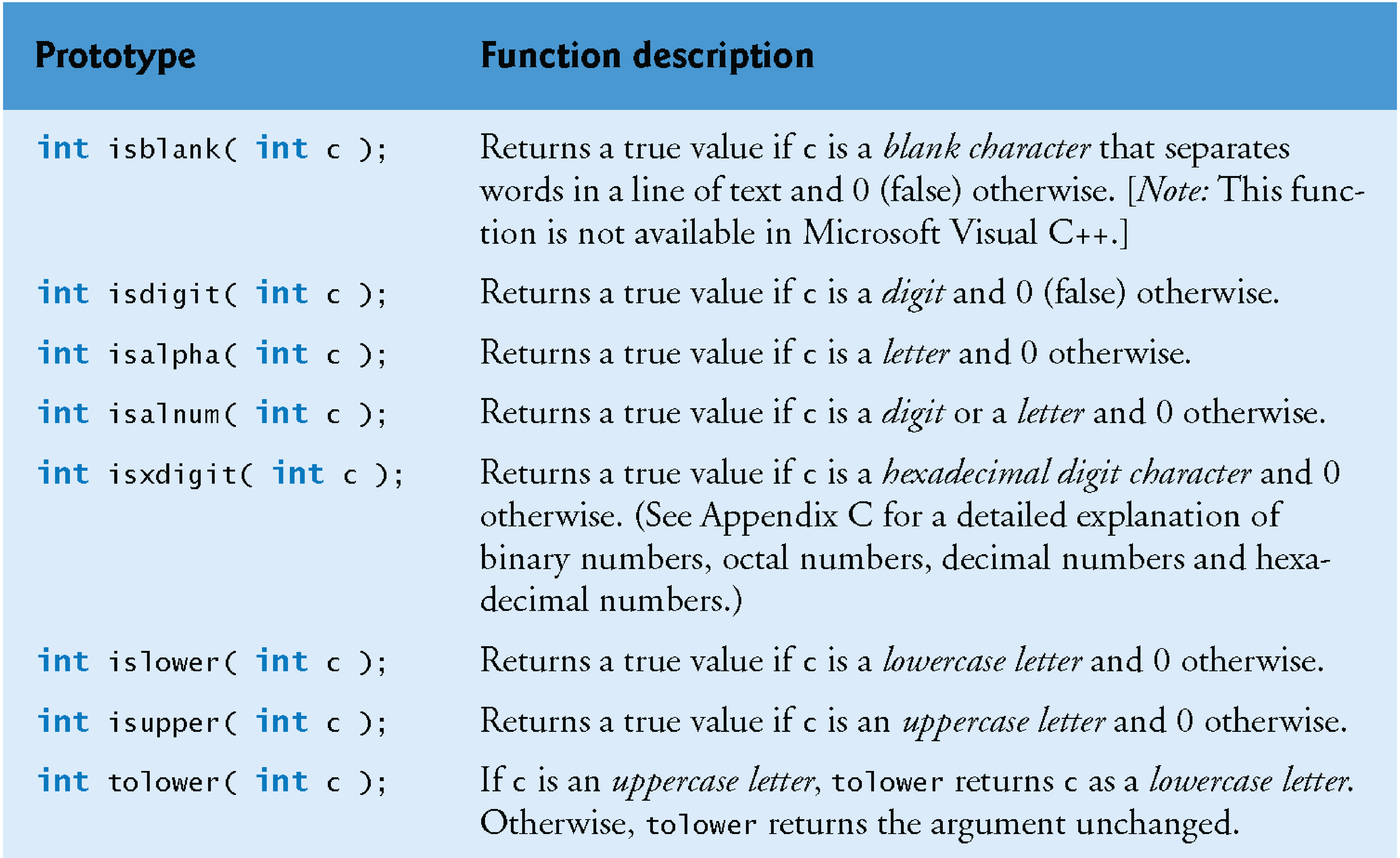
The value of the string `input` will be `”abc\0”`.

If user input is “`abcdefghijkl`”, the code segment will generate output of:

```
abcdefjhi
```

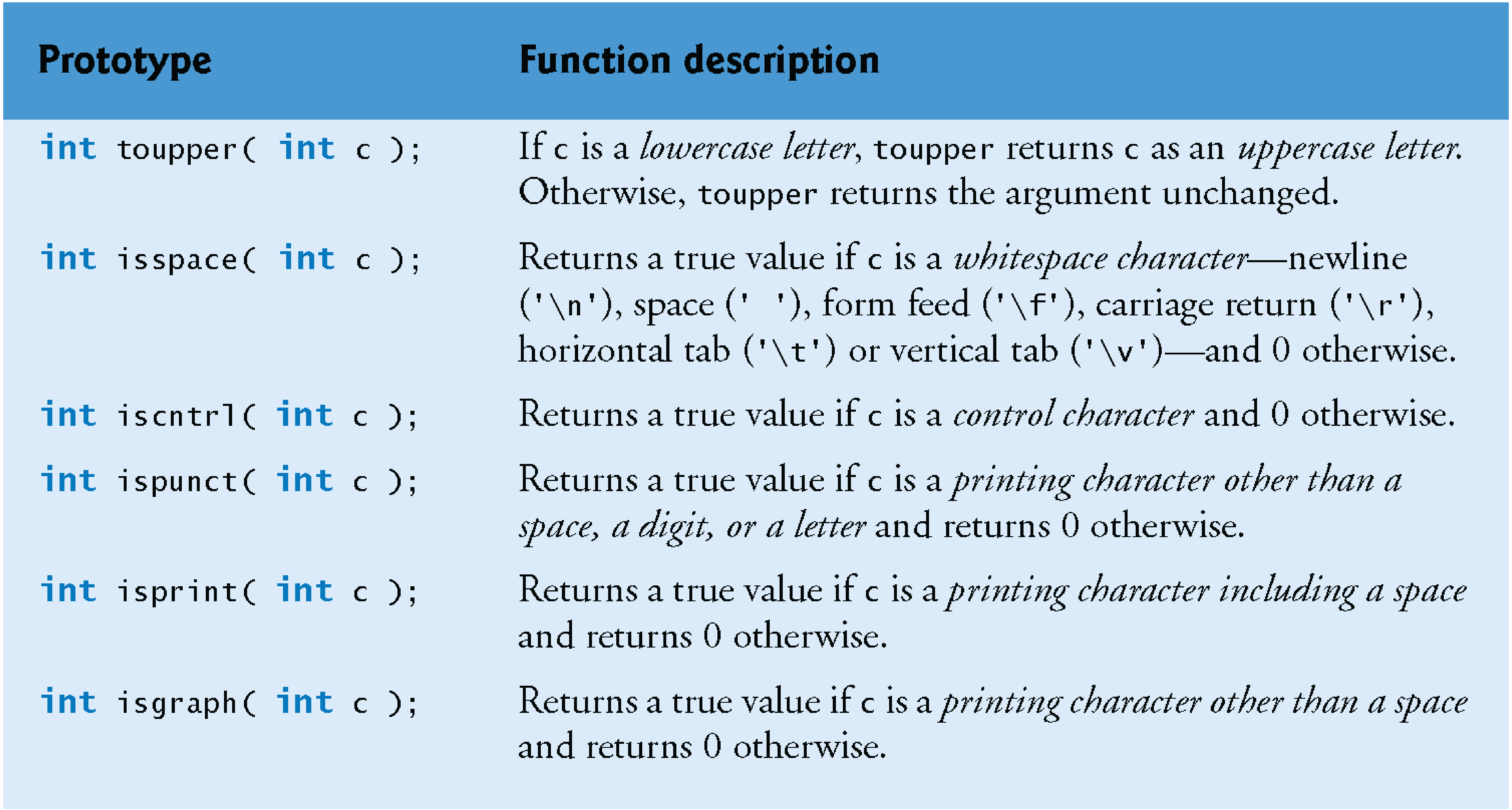
The value of the string `input` will be `”abcdefghi\0”`.

The `<ctype.h>` library



Prototype	Function description
`int isblank( int c );`	Returns a true value if `c` is a *blank character* that separates words in a line of text and 0 (false) otherwise. [*Note:* This function is not available in Microsoft Visual C++.]
`int isdigit( int c );`	Returns a true value if `c` is a *digit* and 0 (false) otherwise.
`int isalpha( int c );`	Returns a true value if `c` is a *letter* and 0 otherwise.
`int isalnum( int c );`	Returns a true value if `c` is a *digit* or a *letter* and 0 otherwise.
`int isxdigit( int c );`	Returns a true value if `c` is a *hexadecimal digit character* and 0 otherwise. (See Appendix C for a detailed explanation of binary numbers, octal numbers, decimal numbers and hexadecimal numbers.)
`int islower( int c );`	Returns a true value if `c` is a *lowercase letter* and 0 otherwise.
`int isupper( int c );`	Returns a true value if `c` is an *uppercase letter* and 0 otherwise.
`int tolower( int c );`	If `c` is an *uppercase letter*, `tolower` returns `c` as a *lowercase letter*. Otherwise, `tolower` returns the argument unchanged.

The `<ctype.h>` library



Prototype	Function description
`int toupper( int c );`	If `c` is a *lowercase letter*, `toupper` returns `c` as an *uppercase letter*. Otherwise, `toupper` returns the argument unchanged.
`int isspace( int c );`	Returns a true value if `c` is a *whitespace character*—newline (`'\n'`), space (`' '`), form feed (`'\f'`), carriage return (`'\r'`), horizontal tab (`'\t'`) or vertical tab (`'\v'`)—and 0 otherwise.
`int iscntrl( int c );`	Returns a true value if `c` is a *control character* and 0 otherwise.
`int ispunct( int c );`	Returns a true value if `c` is a *printing character other than a space, a digit, or a letter* and returns 0 otherwise.
`int isprint( int c );`	Returns a true value if `c` is a *printing character including a space* and returns 0 otherwise.
`int isgraph( int c );`	Returns a true value if `c` is a *printing character other than a space* and returns 0 otherwise.

iscntrl(), ispunct(), isprint() and isgraph()

- Function `iscntrl` determines whether a character is one of the following control characters: horizontal tab (`'\t'`), vertical tab (`'\v'`), form feed (`'\f'`), alert (`'\a'`), backspace (`'\b'`), carriage return (`'\r'`) or newline (`'\n'`).
- Function `ispunct` determines if a character is a printing character other than a space, a digit or a letter, such as `$`, `#`, `(`, `)`, `[`, `]`, `{`, `}`, `;`, `:` or `%`.
- Function `isprint` determines whether a character can be displayed on the screen (including the space character).
- Function `isgraph` is the same as `isprint`, except that the space character is not included.

String Conversion functions

- String-conversion functions are from the general utilities library (`<stdlib.h>`).
- These functions convert strings of digits to integer and floating-point values.
- The C standard also includes `strtoll` and `strtoull` for converting strings to `long long int` and `unsigned long long int`, respectively.
- Note the use of `const` to declare variable `nPtr` in the function headers (read from right to left as "`nPtr` is a pointer to a character constant"); `const` specifies that the argument value will not be modified.

Function prototype	Function description
`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`.

8.4.1 Function `strtod`

Function `strtod` (Fig. 8.6) converts a sequence of characters representing a floating-point value to double.

The function returns `0` if it’s unable to convert any portion of its first argument to double.

The function receives two arguments—a string (`char *`) and a pointer to a string (`char **`).

The string argument contains the character sequence to be converted to double—any whitespace characters at the beginning of the string are ignored.

8.4.1 Function `strtod` (Cont.)

The function uses the `char **` argument to modify a `char *` in the calling function (`stringPtr`) so that it points to the *location of the first character after the converted portion of the string* or to the entire string if no portion can be converted.

Line 14

```
    d = strtod( string, &stringPtr );
```

indicates that `d` is assigned the double value converted from `string`, and `stringPtr` is assigned the location of the first character after the converted value (`51.2`) in `string`.

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

int main( void )
{
   // initialize string pointer
   const char *string = "51.2% are admitted"; // initialize string

   double d; // variable to hold converted sequence
   char *stringPtr; // create char pointer

   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 );
} // end main
```

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

Fig. 8.6 | Using function `strtod`.

8.6 String-Manipulation Functions of the String-Handling Library

- The string-handling library (`<string.h>`) provides many 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.
- This section presents the string-manipulation functions of the string-handling library.
- The functions are summarized in Fig. 8.17.
- Every function—except for `strncpy`—appends the *null* character to its result.

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.

8.6 String-Manipulation Functions of the String-Handling Library (Cont.)

- Functions `strncpy` and `strncat` specify a parameter of type `size_t`, which is a type defined by the C standard as the integral type of the value returned by operator `sizeof`.
- Function `strcpy` copies its second argument (a string) into its first argument—a character array that *you must ensure is large enough* to store the string and its terminating null character, which is also copied.

8.6 String-Manipulation Functions of the String-Handling Library (Cont.)

- Function `strncpy` is equivalent to `strcpy`, except that `strncpy` specifies the number of characters to be copied from the string into the array.
- *Function `strncpy` does not necessarily copy the terminating null character of its second argument.*
- *This occurs only if the number of characters to be copied is at least one more than the length of the string.*

8.6 String-Manipulation Functions of the String-Handling Library (Cont.)

- For example, if `"test"` is the second argument, a terminating null character is written only if the third argument to `strncpy` is at least `5` (four characters in `"test"` plus a terminating null character).
- If the third argument is larger than `5`, null characters are appended to the array until the total number of characters specified by the third argument are written.

8.6.1 Functions `strcpy` and `strncpy`

- Figure 8.15 uses `strcpy` to copy the entire string in array `x` into array `y` and uses `strncpy` to copy the first `14` characters of array `x` into array `z`.
- A *null character* (`'\0'`) is appended to array `z`, because the call to `strncpy` in the program *does not write a terminating null character* (the third argument is less than the string length of the second argument).

```c
// Fig. 8.15: fig08_15.c
// Using functions strcpy and strncpy
#include <stdio.h>
#include <string.h>
#define SIZE1 25
#define SIZE2 15

int main( void )
{
   char x[] = "Happy Birthday to You"; // initialize char array x
   char y[ SIZE1 ]; // create char array y
   char z[ SIZE2 ]; // create char array z

   // copy contents of x into y
   printf( "%s%s\n%s%s\n",
      "The string in array x is: ", x,
      "The string in array y is: ", strcpy( y, x ) );

   // copy first 14 characters of x into z. Does not copy null
   // character
   strncpy( z, x, SIZE2 - 1 );

   z[ SIZE2 - 1 ] = '\0'; // terminate string in z
   printf( "The string in array z is: %s\n", z );
} // end main
```

Fig. 8.15 | Using functions `strcpy` and `strncpy`. (Part 1 of 2.)

```
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
```

Fig. 8.15 | Using functions `strcpy` and `strncpy`. (Part 2 of 2.)

8.6.2 Functions `strcat` and `strncat`

- Function `strcat` appends its second argument (a string) to its first argument (a character array containing a string).
- The first character of the second argument replaces the null (`'\0'`) that terminates the string in the first argument.
- *You must ensure that the array used to store the first string is large enough to store the first string, the second string and the terminating null character copied from the second string*.
- Function `strncat` appends a specified number of characters from the second string to the first string.
- A terminating null character is appended to the result.
- Figure 8.16 demonstrates functions `strcat` and `strncat`.

```c
// Fig. 8.16: fig08_16.c
// Using functions strcat and strncat
#include <stdio.h>
#include <string.h>

int main( void )
{
   char s1[ 20 ] = "Happy "; // initialize char array s1
   char s2[] = "New Year "; // initialize char array s2
   char s3[ 40 ] = ""; // initialize char array s3 to empty

   printf( "s1 = %s\ns2 = %s\n", s1, s2 );

   // concatenate s2 to s1
   printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );

   // concatenate first 6 characters of s1 to s3. Place '\0'
   // after last character
   printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );

   // concatenate s1 to s3
   printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
} // end main
```

Fig. 8.16 | Using functions strcat and strncat. (Part 1 of 2.)

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

Fig. 8.16 | Using functions `strcat` and `strncat`. (Part 2 of 2.)

8.7 Comparison Functions of the String-Handling Library

- This section presents the string-handling library's string-comparison functions, `strcmp` and `strncmp`.
- Fig. 8.17 contains their prototypes and a brief description of each function.

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.

Fig. 8.17 | String-comparison functions of the string-handling library.

8.7 Comparison Functions of the String-Handling Library (Cont.)

- Figure 8.18 compares three strings using `strcmp` and `strncmp`.
- Function `strcmp` compares its first string argument with its second string argument, character by character.
- The function returns 0 if the strings are equal, a *negative value* if the first string is less than the second string and a *positive value* if the first string is greater than the second string.
- Function `strncmp` is equivalent to `strcmp`, except that `strncmp` compares up to a specified number of characters.
- Function `strncmp` does *not* compare characters following a null character in a string.
- The program prints the integer value returned by each function call.

```c
// Fig. 8.18: fig08_18.c
// Using functions strcmp and strncmp
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *s1 = "Happy New Year"; // initialize char pointer
   const char *s2 = "Happy New Year"; // initialize char pointer
   const char *s3 = "Happy Holidays"; // initialize char pointer

   printf("%s%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
      "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,
      "strcmp(s1, s2) = ", strcmp( s1, s2 ),
      "strcmp(s1, s3) = ", strcmp( s1, s3 ),
      "strcmp(s3, s1) = ", strcmp( s3, s1 ) );

   printf("%s%2d\n%s%2d\n%s%2d\n",
      "strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ),
      "strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ),
      "strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 ) );
} // end main
```

Fig. 8.18 | Using functions `strcmp` and `strncmp`. (Part 1 of 2.)

```
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holidays

strcmp(s1, s2) =  0
strcmp(s1, s3) =  1
strcmp(s3, s1) = -1

strncmp(s1, s3, 6) =  0
strncmp(s1, s3, 7) =  6
strncmp(s3, s1, 7) = -6
```

Fig. 8.18 | Using functions `strcmp` and `strncmp`. (Part 2 of 2.)

8.7 Comparison Functions of the String-Handling Library (Cont.)

- To understand just what it means for one string to be “greater than” or “less than” another, consider the process of alphabetizing a series of last names.
- The reader would, no doubt, place “Jones” before “Smith,” because the first letter of “Jones” comes before the first letter of “Smith” in the alphabet.

8.7 Comparison Functions of the String-Handling Library (Cont.)

- But the alphabet is more than just a list of 26 letters—it’s an ordered list of characters.
- Each letter occurs in a specific position within the list.
- “Z” is more than merely a letter of the alphabet; “Z” is specifically the 26th letter of the alphabet.
- How do the string comparison functions know that one particular letter comes before another?
- All characters are represented inside the computer as numeric codes in character sets such as ASCII and Unicode; when the computer compares two strings, it actually compares the numeric codes of the characters in the strings.

8.8 Search Functions of the String-Handling Library

- This section presents the functions of the string-handling library used to search strings for characters and other strings.
- The functions are summarized in Fig. 8.19.
- The functions `strcspn` and `strspn` return `size_t`.
- [*Note*: Function `strtok` has a more secure version described in optional Annex K of the C11 standard. We mention this in the Secure C Programming section of this chapter and in Appendix F.]

Function prototypes and descriptions
`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.

Fig. 8.19 | Search functions of the string-handling library. (Part 1 of 2.)

Function prototypes and descriptions
`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.
`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.

Fig. 8.19 | Search functions of the string-handling library. (Part 2 of 2.)

8.8.1 Function strchr

- Function `strchr` searches for the *first occurrence* of a character in a string.
- If the character is found, `strchr` returns a pointer to the character in the string; otherwise, `strchr` returns `NULL`.
- Figure 8.20 searches for the first occurrences of `'a'` and `'z'` in `"This is a test"`.

```c
// Fig. 8.20: fig08_20.c
// Using function strchr
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *string = "This is a test"; // initialize char pointer
   char character1 = 'a'; // initialize character1
   char character2 = 'z'; // initialize character2

   // if character1 was found in string
   if ( strchr( string, character1 ) != NULL ) {
      printf( "\'%c\' was found in \"%s\".\n",
         character1, string );
   } // end if
   else { // if character1 was not found
      printf( "\'%c\' was not found in \"%s\".\n",
         character1, string );
   } // end else

```

Fig. 8.20 | Using function `strchr`. (Part 1 of 2.)

```c
   // if character2 was found in string
   if ( strchr( string, character2 ) != NULL ) {
      printf( "\'%c\' was found in \"%s\".\n",
         character2, string );
   } // end if
   else { // if character2 was not found
      printf( "\'%c\' was not found in \"%s\".\n",
         character2, string );
   } // end else
} // end main
```

```
'a' was found in "This is a test".
'z' was not found in "This is a test".
```

Fig. 8.20 | Using function `strchr`. (Part 2 of 2.)

8.8.2 Function strcspn

- Function strcspn (Fig. 8.21) determines the length of the initial part of the string in its first argument that does *not* contain any characters from the string in its second argument.
- The function returns the length of the segment.

```c
// Fig. 8.21: fig08_21.c
// Using function strcspn
#include <stdio.h>
#include <string.h>

int main( void )
{
   // initialize two char pointers
   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 ) );
} // end main
```

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

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

Fig. 8.21 | Using function `strcspn`.

8.8.3 Function strpbrk

Function strpbrk searches its first string argument for the *first occurrence* of any character in its second string argument.

If a character from the second argument is found, strpbrk returns a pointer to the character in the first argument; otherwise, strpbrk returns NULL.

Figure 8.22 shows a program that locates the first occurrence in string1 of any character from string2.

```c
// Fig. 8.22: fig08_22.c
// Using function strpbrk
#include <stdio.h>
#include <string.h>

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

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

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

Fig. 8.22 | Using function strpbrk.

8.8.4 Function strrchr

- Function strrchr searches for the *last occurrence* of the specified character in a string.
- If the character is found, strrchr returns a pointer to the character in the string; otherwise, strrchr returns NULL.
- Figure 8.23 shows a program that searches for the last occurrence of the character 'z' in the string "A zoo has many animals including zebras."

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

int main( void )
{
   // initialize char pointer
   const char *string1 = "A zoo has many animals including zebras";

   int c = 'z'; // character to search for

   printf( "%s\n%s'%c'%s\"%s\"\n",
      "The remainder of string1 beginning with the",
      "last occurrence of character ", c,
      " is: ", strrchr( string1, c ) );
} // end main
```

```
The remainder of string1 beginning with the
last occurrence of character 'z' is: "zebras"
```

Fig. 8.23 | Using function strrchr.

8.8.5 Function `strspn`

- Function `strspn` (Fig. 8.24) determines the length of the *initial part* of the string in its first argument that contains only characters from the string in its second argument.
- The function returns the length of the segment.

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

int main( void )
{
   // initialize two char pointers
   const char *string1 = "The value is 3.14159";
   const char *string2 = "aehi lsTuv";

   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 only characters from string2 = ",
      strspn( string1, string2 ) );
} // end main
```

```
string1 = The value is 3.14159
string2 = aehi lsTuv

The length of the initial segment of string1
containing only characters from string2 = 13
```

Fig. 8.24 | Using function strspn.

8.8.6 Function strstr

- Function strstr searches for the *first occurrence* of its second string argument in its first string argument.
- If the second string is found in the first string, a pointer to the location of the string in the first argument is returned.
- Figure 8.25 uses strstr to find the string "def" in the string "abcdefabcdef".

```c
// Fig. 8.25: fig08_25.c
// Using function strstr
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *string1 = "abcdefabcdef"; // string to search
   const char *string2 = "def"; // string to search for

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

```
string1 = abcdefabcdef
string2 = def

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

Fig. 8.25 | Using function `strstr`.

8.8.7 Function strtok

- Function `strtok` (Fig. 8.26) is used to break a string into a series of tokens.
- A token is a sequence of characters separated by delimiters (usually *spaces* or *punctuation marks*, but a delimiter can be *any character*).
- For example, in a line of text, each word can be considered a token, and the spaces and punctuation separating the words can be considered delimiters.

```c
// Fig. 8.26: fig08_26.c
// Using function strtok
#include <stdio.h>
#include <string.h>

int main( void )
{
   // initialize array string
   char string[] = "This is a sentence with 7 tokens";
   char *tokenPtr; // create char pointer

   printf( "%s\n%s\n\n%s\n",
      "The string to be tokenized is:", string,
      "The tokens are:" );

   tokenPtr = strtok( string, " " ); // begin tokenizing sentence

   // continue tokenizing sentence until tokenPtr becomes NULL
   while ( tokenPtr != NULL ) {
      printf( "%s\n", tokenPtr );
      tokenPtr = strtok( NULL, " " ); // get next token
   } // end while
} // end main
```

Fig. 8.26 | Using function `strtok`. (Part 1 of 2.)

```
The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:
This
is
a
sentence
with
7
tokens
```

Fig. 8.26 | Using function `strtok`. (Part 2 of 2.)

8.8.7 Function strtok (Cont.)

- Multiple calls to `strtok` are required to tokenize a string—i.e., break it into tokens (assuming that the string contains more than one token).
- The first call to `strtok` contains two arguments: a string to be tokenized, and a string containing characters that separate the tokens.
- In line 16, the statement

  -
    ```
    // begin tokenizing sentence
    tokenPtr = strtok( string, " " );
    ```

  assigns `tokenPtr` a pointer to the first token in `string`.

8.8.7 Function strtok (Cont.)

- The second argument, `" "`, indicates that tokens are separated by spaces.
- Function `strtok` searches for the first character in `string` that’s not a delimiting character (space).
- This begins the first token.
- The function then finds the next delimiting character in the string and *replaces it with a null (* `'\0'`*) character* to terminate the current token.
- Function `strtok` saves a pointer to the next character following the token in `string` and returns a pointer to the current token.

8.8.7 Function strtok (Cont.)

- Subsequent `strtok` calls in line 21 continue tokenizing `string`.
- These calls contain *`NULL` as their first argument*.
- The `NULL` argument indicates that the call to `strtok` should continue tokenizing from the location in `string` saved by the last call to `strtok`.
- If no tokens remain when `strtok` is called, `strtok` returns `NULL`.

8.8.7 Function strtok (Cont.)

- You can change the delimiter string in each new call to `strtok`.
- Figure 8.26 uses `strtok` to tokenize the string `"This is a sentence with 7 tokens"`.
- Each token is printed separately.
- Function `strtok` *modifies the input string* by placing `'\0'` at the end of each token; therefore, a *copy* of the string should be made if the string will be used again in the program after the calls to `strtok`.

Q&A?

Last, but not least

Check that you have submit your attendance.

If you don’t have your phone or computer and could not submit your attendance, you should talk to me NOW!

Attendance cannot be made up.