

You can use various functions that you learned in this chapter to fully validate such input. For example, you could

- use `fgets` to read the input as a line of text,
- convert the string to a number using `strtol` and ensure that the conversion was successful, then
- ensure that the value is in range.

For more information and techniques for converting input to numeric values, see CERT guideline INT05-C at <https://wiki.sei.cmu.edu/>.

✓ Self Check

1 (Fill-In) Among other things, the secure string-processing functions of Annex K help prevent buffer _____ by requiring additional parameters that specify the number of elements in a target array and by ensuring that pointer arguments are non-NULL.
Answer: overflows.

2 (True/False) It's important to validate the data you input into a program. You can use various string-and-character-processing functions to fully validate inputs. For example, you could use `fgets` to read the input as a line of text, convert the string to a number using `strtol`, ensure the conversion was successful, then ensure that the value is in range.
Answer: *True*.

Summary

Section 8.2 Fundamentals of Strings and Characters

- **Characters** are the fundamental building blocks of source programs. Every program is composed of a sequence of characters that—when grouped together meaningfully—is interpreted by the computer as instructions used to accomplish a task.
- A **character constant** (p. 388) is an `int` value represented as a character in single quotes. The value of a character constant is the character's integer value in the machine's **character set** (p. 389).
- A **string** (p. 389) is a series of characters treated as a single unit. A string may include letters, digits and various special characters (p. 389) such as `+`, `-`, `*`, `/` and `$`. String literals, or string constants, are written in double quotation marks.
- A string in C is an **array of characters** ending in the **null character** (p. 389; `'\0'`).
- A string is accessed via a **pointer** to its first character. The value of a string is the **address** of its first character.
- A **character array** or a **variable of type `char *`** can be initialized with a string in a definition.
- When defining a character array to contain a string, the array must be large enough to store the string and its terminating null character.
- A string can be stored in an array using `scanf`. Function `scanf` will read characters until a space, tab, newline or end-of-file indicator is encountered.
- For a character array to be printed as a string, the array must contain a terminating null character.

Section 8.3 Character-Handling Library

- Function **isdigit** (p. 391) determines whether its argument is a **digit** (0–9).
- Function **isalpha** (p. 391) determines whether its argument is an **uppercase letter** (A–Z) or a **lowercase letter** (a–z).
- Function **isalnum** (p. 391) determines whether its argument is an **uppercase letter** (A–Z), a **lowercase letter** (a–z) or a **digit** (0–9).
- Function **isxdigit** (p. 391) determines whether its argument is a **hexadecimal digit** (p. 391; A–F, a–f, 0–9).
- Function **islower** (p. 393) determines whether its argument is a **lowercase letter** (a–z).
- Function **isupper** (p. 393) determines whether its argument is an **uppercase letter** (A–Z).
- Function **toupper** (p. 393) converts a lowercase letter to uppercase and returns it.
- Function **tolower** (p. 393) converts an uppercase letter to lowercase and returns it.
- Function **isspace** (p. 394) determines whether its argument is one of the following **whitespace characters**: ' ' (space), '\f', '\n', '\r', '\t' or '\v'.
- Function **iscntrl** (p. 394) determines whether its argument is one of the following **control characters**: '\t', '\v', '\f', '\a', '\b', '\r' or '\n'.
- Function **ispunct** (p. 394) determines whether its argument is a **printing character** other than a space, a digit or a letter.
- Function **isprint** (p. 394) determines whether its argument is any printing character, including the space character.
- Function **isgraph** (p. 394) determines whether its argument is a printing character other than the space character.

Section 8.4 String-Conversion Functions

- Function **strtod** (p. 396) converts a sequence of characters representing a floating-point value to `double`. The location specified by its pointer to `char *` argument is assigned the remainder of the string after the conversion, or to the entire string if no portion of the string can be converted.
- Function **strtol** (p. 397) converts a sequence of characters representing an integer to `long`. This function works identically to `strtod`, but the third argument specifies the base of the value being converted.
- Function **strtoul** (p. 398) works identically to `strtol` but converts a sequence of characters representing an integer to unsigned `long int`.

Section 8.5 Standard Input/Output Library Functions

- Function **fgets** (p. 399) reads characters until a newline character or the end-of-file indicator is encountered. The arguments to `fgets` are an array of type `char`, the maximum number of characters to read and the stream from which to read. A null character ('\0') is appended to the array after reading terminates. If a newline is encountered, it's included in the input string.
- Function **putchar** (p. 399) prints its character argument.
- Function **getchar** (p. 401) reads a single character from the standard input and returns it as an integer. If the end-of-file indicator is encountered, `getchar` returns EOF.
- Function **puts** (p. 401) takes a string (`char *`) as an argument and prints the string followed by a newline character.
- Function **sprintf** (p. 401) uses the same conversion specifications as function `printf` to print formatted data into an array of type `char`.

- Function **sscanf** (p. 402) uses the same conversion specifications as function **scanf** to read formatted data from a string.

Section 8.6 String-Manipulation Functions of the String-Handling Library

- Function **strcpy** copies its second argument string into its first argument char array. You must ensure that the array is large enough to store the string and its terminating null character.
- Function **strncpy** (p. 404) is equivalent to **strcpy**, but specifies the maximum number of characters to copy from the string into the array. The terminating null character will be copied only if the number of characters to be copied is one more than the string's length.
- Function **strcat** (p. 405) appends its second argument string—including its terminating null character—to its first argument string. The first character of the second string replaces the null ('\0') character of the first string. You must ensure that the array used to store the first string is large enough to store both the first string and the second string.
- Function **strncat** (p. 404) appends a specified number of characters from the second string to the first string. A terminating null character is appended to the result.

Section 8.7 Comparison Functions of the String-Handling Library

- Function **strcmp** (p. 406) compares its first string argument to its second string argument, character by character. It returns 0 if the strings are equal, a negative value if the first string is less than the second or a positive value if the first string is greater than the second.
- Function **strncmp** (p. 406) is equivalent to **strcmp**, except that **strncmp** compares a specified number of characters. If one of the strings is shorter than the number of characters specified, **strncmp** compares characters until the null character in the shorter string is encountered.

Section 8.8 Search Functions of the String-Handling Library

- Function **strchr** (p. 409) searches for the first occurrence of a character in a string. If found, **strchr** returns a pointer to the character in the string; otherwise, **strchr** returns NULL.
- Function **strcspn** (p. 410) 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 segment's length.
- Function **strpbrk** (p. 410) searches for the first occurrence in its first argument of any character in its second argument. If a character from the second argument is found, **strpbrk** returns a pointer to the character; otherwise, **strpbrk** returns NULL.
- Function **strrchr** (p. 411) searches for the last occurrence of a character in a string. If found, **strrchr** returns a pointer to the character in the string; otherwise, **strrchr** returns NULL.
- Function **strspn** (p. 412) 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.
- Function **strstr** (p. 412) 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.
- A sequence of calls to **strtok** (p. 413) breaks its first string argument into tokens (p. 413) that are separated by characters contained in the second string argument. The first call contains the string to tokenize as the first argument. Subsequent calls to continue tokenizing that 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, it returns NULL.

Section 8.9 Memory Functions of the String-Handling Library

- Function **memcpy** (p. 415) copies a specified number of bytes from the object to which its second argument points into the object to which its first argument points. The function can receive a pointer to any type of object.
- Function **memmove** (p. 416) copies a specified number of bytes from the object pointed to by its second argument to the object pointed to by its first argument. Copying is accomplished as if the bytes were copied from the second argument to a temporary array and then copied from the temporary array to the first argument.
- Function **memcmp** (p. 416) compares the specified number of bytes of its first and second arguments.
- Function **memchr** (p. 417) searches for the first occurrence of a byte, represented as unsigned char, in the specified number of bytes of an object. If the byte is found, a pointer to the byte is returned; otherwise, a NULL pointer is returned.
- Function **memset** (p. 417) copies its second argument, treated as an unsigned char, to a specified number of bytes of the object pointed to by the first argument.

Section 8.10 Other Functions of the String-Handling Library

- Function **strerror** (p. 419) maps an integer error number into a full text string in a locale-specific manner. A pointer to the string is returned.
- Function **strlen** (p. 419) takes a string as an argument and returns the **number of characters** in the string—the terminating null character is not included in the length of the string.

Self-Review Exercises

8.1 Write a single statement to accomplish each of the following. Assume variable *c* is a char, variables *x*, *y* and *z* are ints, variables *d*, *e* and *f* are doubles, variable *ptr* is a char * and *s1* and *s2* are 100-element char arrays.

- Convert the character stored in variable *c* to an uppercase letter. Assign the result to variable *c*.
- Determine whether the value of variable *c* is a digit. Use the conditional operator as shown in Figs. 8.1–8.3 to print " is a " or " is not a " when the result is displayed.
- Determine whether the value of variable *c* is a control character. Use the conditional operator to print " is a " or " is not a " when the result is displayed.
- Read a line of text into array *s1* from the keyboard. Do not use `scanf`.
- Print the line of text stored in array *s1*. Do not use `printf`.
- Assign *ptr* the location of the last occurrence of *c* in *s1*.
- Print the value of variable *c*. Do not use `printf`.
- Determine whether the value of *c* is a letter. Use the conditional operator to print " is a " or " is not a " when the result is displayed.
- Read a character from the keyboard and store the character in variable *c*.
- Assign *ptr* the location of the first occurrence of *s2* in *s1*.
- Determine whether the value of variable *c* is a printing character. Use the conditional operator to print " is a " or " is not a " when the result is displayed.