```
Array b displayed with:

Array subscript notation
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40
```

Fig. 8.17 | Using subscripting and pointer notations with built-in arrays. (Part 3 of 4.)

```
Pointer/offset notation where the pointer is the array name
*(b + 0) = 10
*(b + 1) = 20
*(b + 2) = 30
*(b + 3) = 40

Pointer subscript notation
bPtr[0] = 10
bPtr[1] = 20
bPtr[2] = 30
bPtr[3] = 40

Pointer/offset notation
*(bPtr + 0) = 10
*(bPtr + 1) = 20
*(bPtr + 2) = 30
*(bPtr + 3) = 40
```

Fig. 8.17 | Using subscripting and pointer notations with built-in arrays. (Part 4 of 4.)

8.10 Pointer-Based Strings

- This section introduces C-style, pointer-based strings, which we'll simply call C strings.
- *C++'s `string` class is preferred for use in new programs, because it eliminates many of the security problems that can be caused by manipulating C strings.*
- We cover C strings here for a deeper understanding of arrays.
- Also, if you work with legacy C and C++ programs, you're likely to encounter pointer-

8.10 Pointer-Based Strings (cont.)

Characters and Character Constants

- Characters are the fundamental building blocks of C++ source programs.
- Character constant
  - An integer value represented as a character in single quotes.
  - The *value* of a character constant is the integer value of the character in the machine’s character set.

8.10 Pointer-Based Strings (cont.)

Strings

- A string is a series of characters treated as a single unit.
  - May include letters, digits and various special characters such as `+`, `-`, `*`, `/`and `$`.
- String literals, or string constants, in C++ are written in double quotation marks

Pointer-Based Strings

- A pointer-based string is a built-in array of characters ending with a null character (`'\0'`).
- A string is accessed via a pointer to its first

8.10 Pointer-Based Strings (cont.)

String Literals as Initializers

- A string literal may be used as an initializer in the declaration of either a built-in array of `char`s or a variable of type `const char *`.
- String literals have *static storage duration* (they exist for the duration of the program) and may or may not be *shared* if the same string literal is referenced from multiple locations in a program.

Error-Prevention Tip 8.6

If you need to modify the contents of a string literal, store it in a built-in array of `char`s first.

8.10 Pointer-Based Strings (cont.)

Character Constants as Initializers

- When declaring a built-in array of `char`s to contain a string, the built-in array must be large enough to store the string *and* its terminating null character.

Common Programming Error 8.7

Not allocating sufficient space in a built-in array of `char`s to store the null character that terminates a string is a logic error.

Common Programming Error 8.8

Creating or using a C string that does not contain a terminating null character can lead to logic errors.

Error-Prevention Tip 8.7

When storing a string of characters in a built-in array of `char`s, be sure that the built-in array is large enough to hold the largest string that will be stored. C++ allows strings of any length. If a string is longer than the built-in array of `char`s in which it's to be stored, characters beyond the end of the built-in array will overwrite data in memory following the built-in array, leading to logic errors and potential security breaches.

8.10 Pointer-Based Strings (cont.)

Accessing Characters in a C String

- Because a C string is a built-in array of characters, we can access individual characters in a string directly with array subscript notation.

8.10 Pointer-Based Strings (cont.)

Reading Strings into `char` Built-In Arrays with `cin`

- A string can be read into a built-in array of `char`s using stream extraction with `cin`.
- The `setw` stream manipulator can be used to *ensure* that the string read into `word` *does not exceed the size of the built-in array*.
  - Applies *only* to the next value being input.

8.10 Pointer-Based Strings (cont.)

Reading Lines of Text into `char` Built-In Arrays with `cin.getline`

- In some cases, it's desirable to input an *entire line of text* into a built-in array of `char`s.
- For this purpose, the `cin` object provides the member function `getline`, which takes three arguments—a *built-in array of `char`s* in which the line of text will be stored, a *length* and a *delimiter character*.
- The function stops reading characters when the delimiter character `'\n'` is encountered, when the *end-of-file indicator* is entered or when the number of characters read so far is one less than the length specified in the second argument.
- The third argument to `cin.getline` has `'\n'` as a default value.

8.10 Pointer-Based Strings (cont.)

Displaying C Strings

- A built-in array of `char`s representing a null-terminated string can be output with `cout` and `<<`.
- The characters are output until a *terminating null character* is encountered; the null character is *not* displayed.
- `cin` and `cout` assume that built-in arrays of `char`s should be processed as strings terminated by null characters; `cin` and `cout` do not provide similar input and output