

C-Style Strings

Solutions

- Describe the structure of a C-style string
 - An array of characters with a terminating null character
- Give two different ways to initialize a C-style string
 - Array initialization (needs terminating null)
`char abc = {'a', 'b', 'c', '\0'};`
 - Initialization from string literal
`const char *xyz = "xyz";`

- How is a string literal implemented?
 - C-style string with const characters

- Give some of the drawbacks of using C-style string manipulation functions
 - These rely on the terminating null to find the end of the string
 - If this is omitted, or overwritten, then a buffer overrun will occur (common security vulnerability)
 - Memory management errors, if allocated on the heap
 - `std::string` is much safer and easier to use (knows its own size, performs its own memory management)

- Give some advantages of C-style strings compared to `std::string`
 - Less overhead than `std::string`
 - Can use stack memory
 - Direct compatibility with C and older C++ code

- Give some disadvantages of C-style strings compared to `std::string`
 - Usual disadvantages of built-in arrays
 - Harder to use
 - Functions provided for manipulating C-style strings are highly unsafe

- The strcmp function compares two C-style strings
 - Defined in <cstring>, safe if given null-terminated inputs
- strcmp(s1, s2) will return
 - 1 if $s1 > s2$
 - 0 if $s1 == s2$
 - -1 if $s1 < s2$
 - Uses case-sensitive alphabetical ordering (strictly speaking lexicographical, which covers numbers, punctuation etc)

- Using strcmp, write a function my_max() which will compare two C-style strings and return the larger one
- Write a program to test your function which calls my_max with two string literals and prints out the returned string
- Amend your program to compare two array-initialized C-style strings
- What happens if the strings are not null terminated?
 - May see extra characters at the end of output, indicating that the end of the string has not been correctly detected