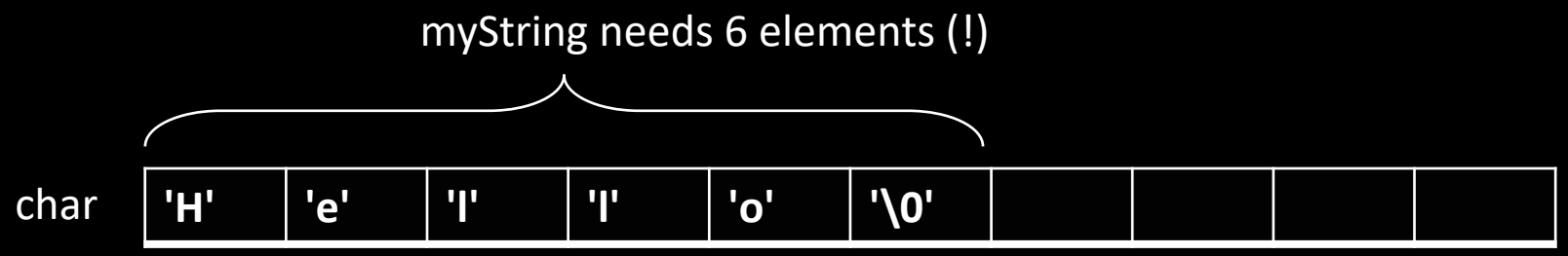
c-strings or cstrings or C-style strings

C-string:

- Early C language: C-string is an array of characters.
- End of string is indicated by a 0-value: '\0' (terminating null character)
  - `char myString[] { "Hello" };`



C-string:

- C-style string is an array > all rules do apply:
  - `char mystring[] { "Hello" };`
  - `mystring[0] = 'Y';`
  - `cout << mystring;`
  - `-> Yello is printed`

C-string:

- cout prints characters until a '\0' is found. If by accident the '\0' is not present, cout keeps printing everything in the adjacent memory slot until it happens to be a 0:
  - `char mystring[5] { "Hello" };`
  - `cout << mystring;`
  - `-> Hello is printed, followed by undefined characters`

String literal

- What? Anything between quotation marks. Example: "Hello"
- Static storage duration > exist in memory for the lifetime of the program.
- Type > `const char[]`
- Example: "Hello" is holding the characters 'H', 'e', 'l', 'l', 'o', and '**\0**'.
- String literals can be used to initialize c-strings and std::string.

```
char name[] { "Hello" };
```

- The null character '\0' is always appended to a string literal:
  - The size of the name array is 6.

Unicode

- computing industry standard
- encoding, representation and handling text
- Implemented by UTF-8, UTF-16 and UTF-32, (others)
- UTF-8: dominant >90% websites

Unicode

- UTF-8:
  - encoding unicode using 8, 16 or 32 bits
- UTF-16:
  - encoding unicode using 16 or 32 bits
- UTF-32:
  - encoding unicode using 32 bits

Unicode

- UTF-8:
  - ascii compatible, used by html, xml
- UTF-16:
  - not ascii compatible, used by windows, java
- UTF-32:
  - not ascii compatible

UTF-8

8 bit characters

- 8 bits are used to encode a Unicode character
- limited to western character sets

16 bit characters

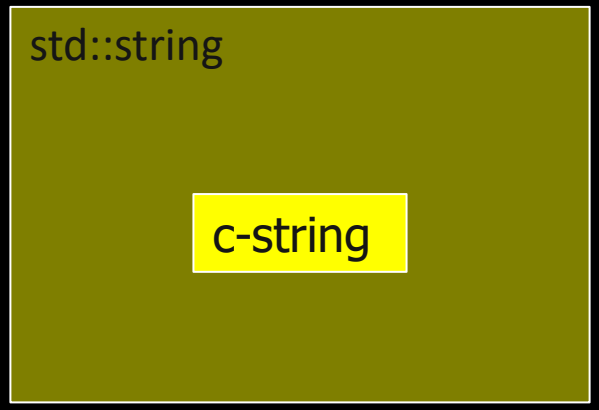
- 16 bits are used to encode a Unicode character.
- worldwide use: eg. Chinese, Korean

C-string: 8 vs 16 bits/character

single byte: char or char8_t	two bytes (wide): wchar_t or char16_t
➢ `sizeof(char)` -> 1 byte	➢ `sizeof(wchar_t)` -> 2 bytes
➢ `char name[] { "Tibo" }`	➢ `wchar_t name[] { L"Tibo" };`
➢ `name[2] = 'k';`	➢ `name[2] = 'k';`
➢ c-string functions start with "str"	➢ Wide c-string functions > "Wide Character String" > wcs
➢ Standard Library functions for c-strings: ➢ strlen() strcpy() strcmp()	➢ Standard Library functions for wide c-strings: ➢ wcslen() wcscpy() wcscmp()

std::string

- Is a class that encapsulates a c-string.
- Is safer to work with than a c-string > no worries about terminating 0.
- Is preferred over c-string
- c-string is still being used > compatibility with c language
- Has a huge set of member functions: cppreference



std::string vs std::wstring

std::string	std::wstring
➢ standard ASCII – UTF-8	➢ Wide character and Unicode UTF-16
➢ `std::string name { "Tibo" };`	➢ `std::wstring name { L"Tibo" };`
➢ `std::cout << name << endl;`	➢ `std::wcout << name << endl;`

Both implementations are derived from the same base class and have identical methods. (see later)

from std::string to c-string

std::string

```
std::string name { "Tibo" };
const char *s = name.c_str();
name[2] = 'k';
s[2] = 'b'; // error
```

std::wstring

```
std::wstring name { L"Tibo" };
const wchar_t *s = name.c_str();
name[2] = 'k';
s[2] = 'b'; // error
```

References

- http://www.learncpp.com/cpp-tutorial/66-c-style-strings/