C-Style Strings

CS 150 – C++ Programming I
Lecture 22

C-Strings

- A char array is used for C-style or traditional strings
  - `char greeting[] = "Hello";`
    ```
    // greeting[6] = "Hello";
    // greeting[] = {'H','e','l','l','o','\0'};
    ```
  - Don't need to use braces or commas as with traditional array
- Array occupies six bytes, not 5—one byte for each char followed by a binary zero or NUL terminator
  - NUL is the "name" of a character with the ASCII value `0`
  - NULL is the C representation of a pointer with the value `0`.

Where is the Data Stored?

- Let's look at two examples:
  - `char pet1[] = "Dog"; // equivalent to`
  - `char pet1[] = {'D','o','g','\0' };`
  - Memory is allocated for 4 characters in user space
  - `pet1` is an array or block of memory of size 4
- `char * pet2 = "Cat";`
  - `pet2` is a pointer to an array of 4 `chars`
  - Stored in read-only memory
  - Allowed in C. In C++ should be declared `const char *`
  - `pet2` may be reassigned, `pet1` not

Using C-Strings

- The standard library inherited a collection of functions in the header `<cstring>`
  - C and older C++ implementations use `string.h`
- Looping through strings? Use `strlen()`, not `size()`

```
string a = "Hello";
char b[] = "Goodbye";
size_t lenA = a.length();
size_t lenB = strlen(b);
for (size_t i = 0; i < lenB; i++) ...
for (auto& c : a) ...
```

Assignment or Copying

- C++ *string* assignment or copying

  -
    ```
    string a = "Hello";
    string b = a; // copies "Hello" into b
    ```
- C-string assignment or copying

  -
    ```
    const char *a = "Hello";
    char b[10];
    strcpy(b, a);
    ```

▪ Destination MUST have sufficient space to store all characters plus the NUL byte

String Concatenation

- C++ *string* concatenation

  -
    ```
    string a = "Hello";
    string b = a + " beautiful!";
    ```
- Here's the C-string version

  -
    ```
    const char *a = " beautiful!";
    char b[25] = "Hello";
    strcat(b, a);
    ```
- Destination MUST have sufficient space to store all characters from both strings, plus the NUL byte

String Comparison

- C++ *string* comparison

  -
    ```
    string a = "Hello", b = ???;
    if (a == b) ...
    if (a < b)...
    if (a > b)...
    ```
- C-string comparisons

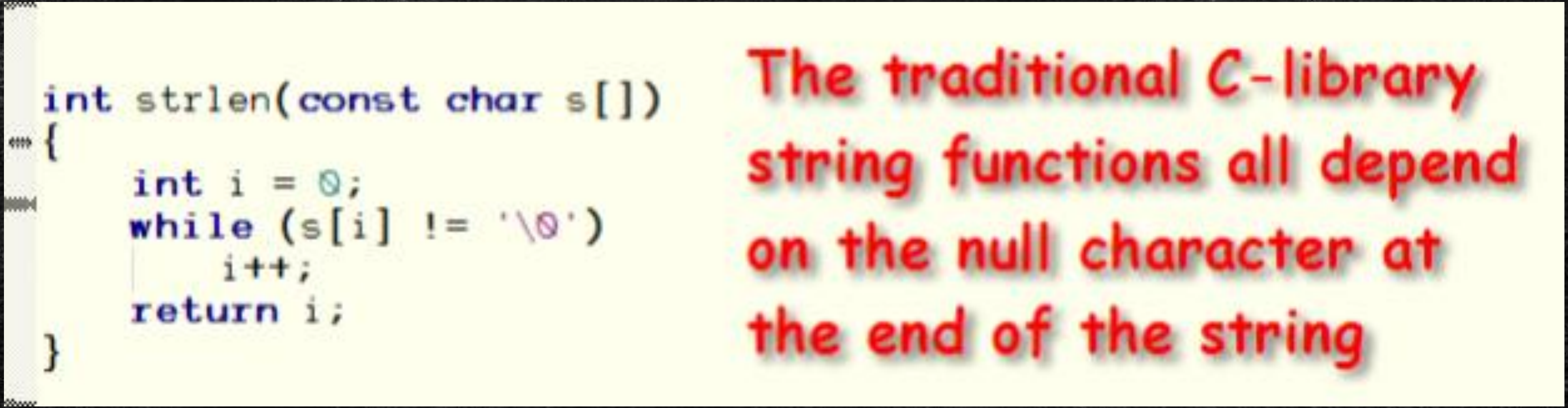
  -
    ```
    const char * a = "Hello", *b = ???;
    if (strcmp(a, b) == 0) ...
    if (strcmp(a, b) < 0)...
    if (strcmp(a, b) > 0)...
    ```
  - Common `strcmp()` bug; forgetting to check the return value

Your Turn

- Exercise: Open *`MinCat.cpp`*
  - Use the standard c-string functions to write a function that concatenates two c-string literals
    - `s1` and `s2` are the two strings
    - `out` is an array of `char` where the answer goes
    - `maxLen` is the size of the output array
  - Find the length of the shorter string: `len`
  - Concatenate only the last `len` characters from the longer string. See examples in problem.
- Exercise: *`penv.cpp`*

Processing C-Style Strings

- Process traditional C-style strings just like arrays, except, you assume that there is a terminating NUL character in the array
  - Use sentinel rather than counter-controlled loops

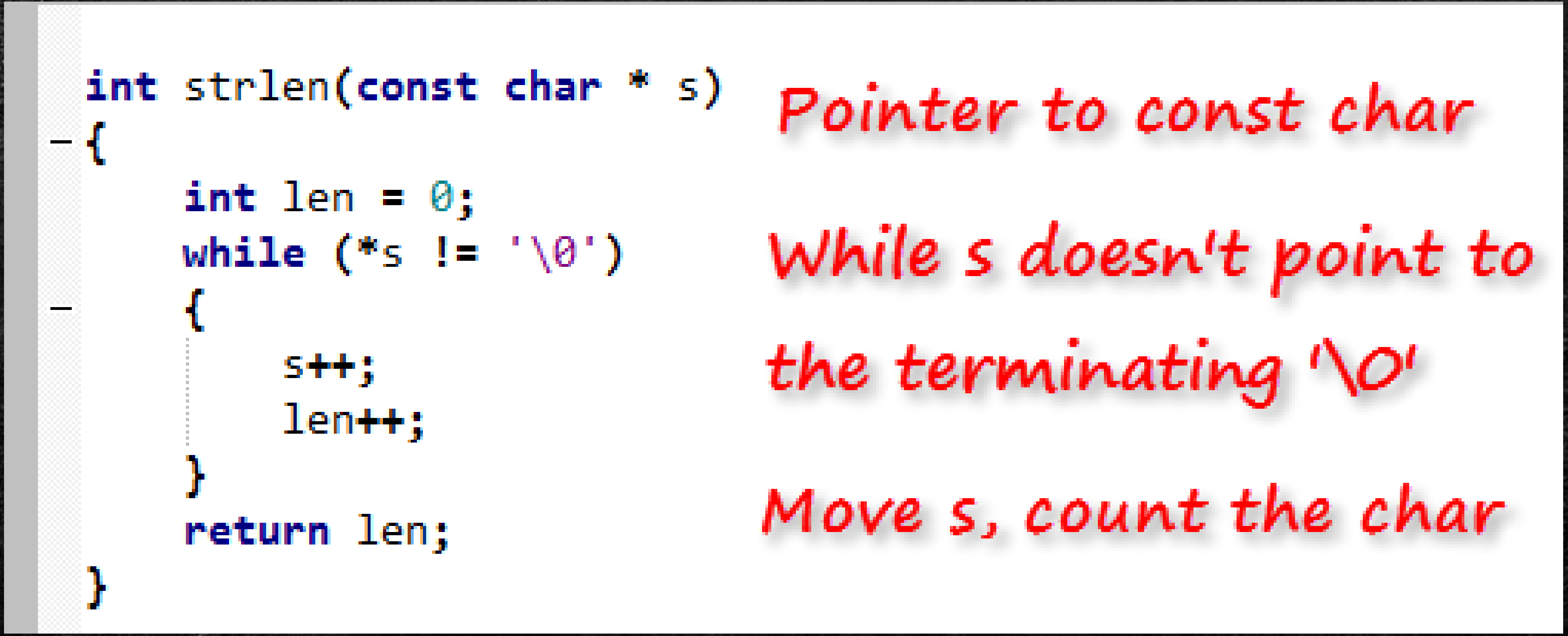


```
int strlen(const char s[])
{
    int i = 0;
    while (s[i] != '\0')
        i++;
    return i;
}
```

The traditional C-library string functions all depend on the null character at the end of the string

Processing C-Strings Using Pointers

- While you can use array syntax (with no loss of efficiency), it is more common to use pointer syntax



```
int strlen(const char * s)
{
    int len = 0;
    while (*s != '\0')
    {
        s++;
        len++;
    }
    return len;
}
```

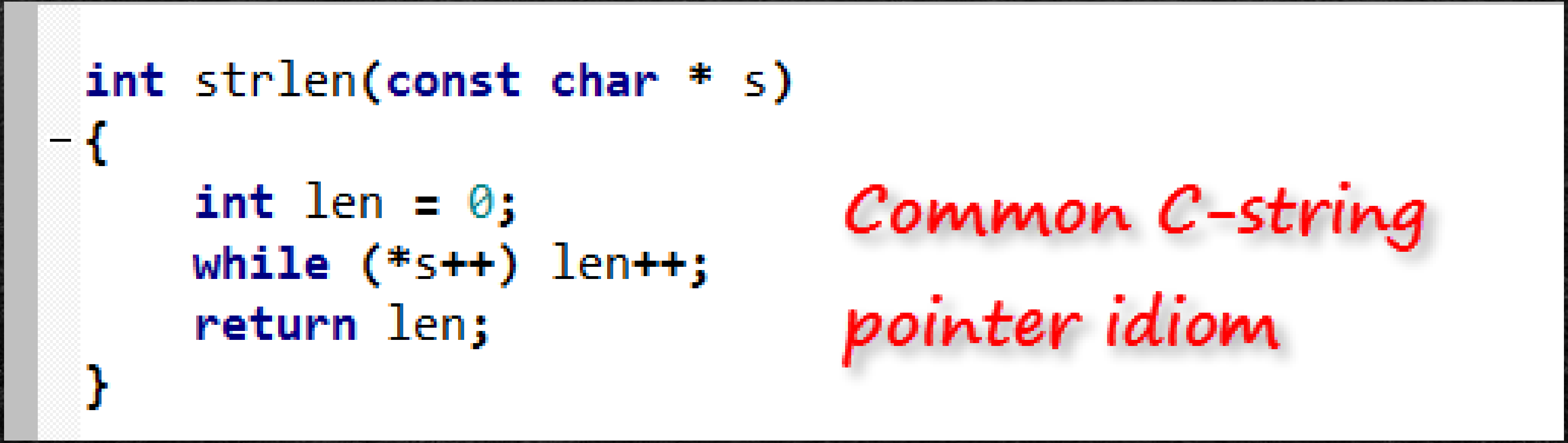
Pointer to const char

While s doesn't point to the terminating '\0'

Move s, count the char

Processing C-Strings Using Pointers

- It is also very common to use a much more concise syntax when processing C-strings with pointers
  - I don't encourage this kind of code, but it is a very common idiom, so you should recognize and understand it when you see it



```
int strlen(const char * s)
{
    int len = 0;
    while (*s++) len++;
    return len;
}
```

Your Turn

- This is from the online C strings practice problems
  - Unlike the online version ours will not use any library functions. (This is similar to PE08)
- Open `countmatches.cpp`
  - Look through an array
  - Count the number strings that contain the first string
  - Let's solve it using array notation, and then using pointer notation.