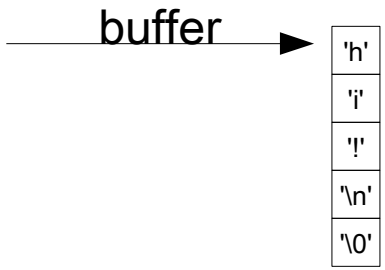
Strings and Overloading operator[]

C-strings

- Recall that a C-string is implemented as a NULL terminated array of type *char*

```
Char buffer[5];
strcpy(buffer,“hi!\n”);
cout<< buffer;
```



- When we use “” the compiler makes a NULL terminated const char array and fills it with the characters the programmer chose
- NOT every char array is a c-string, only those that are NULL terminated
  - Link to c-string review:
    - http://www.cs.fsu.edu/~myers/c++/notes/strings.html

C-string and c++

- We have some features in the standard C++ libraries available to help us work more easily with C-style strings
  - The <cstring> library
    - Contains functions for common string operations, such as copy, compare, concatenate, length, search, tokenization, and more
      - strlen(), strcpy(), strncpy(), strcat(), strncat(), strcmp(), strncmp(), strstr(), strtok()
  - Special features in <iostream>:
    - Special built-in functions for I/O handling of C-style strings, like the insertion and extraction operators, get(), getline(), etc
    - char str1[40];
    - cout << str1; // insertion operator for strings
    - cin >> str1; // extraction, reads up to white space
    - cin.get(str1, 40, ','); // reads to delimiter (comma)
    - cin.getline(sr1, 40); // reads to delimiter (default delimiter is newline), discards // delimiter

SAMPLE1.CPP

The Downside of C-strings

- Fixed length (when declared as static array)
- String name acts like a pointer
  - Much must be passed in and out of functions
- Array bounds are not automatically enforced
- Must use cumbersome functions instead of intuitive operators
  - strcpy(str1, str2); instead of str1 = str2;
  - (strcmp(str1, str2)) instead of (str1 == str2)
  - strcat(str1, str2) instead of str1 += str2;
- The NULL char can be tricky
  - See sample2.cpp, sample3.cpp, sample4.cpp

String Wish List

- We would like a more intuitive string interface
  - str1 + str2 //concatenation
  - str1 == str2 //compare str1 and str2
  - str1 = “Hello!\n” //store “hello!\n” in str1
- We would like to keep some of the legacy functionality
  - str1[4] // returns 4th char in str1
  - str1[4] = 'a' //sets 4th char in str1 to 'a'
  - &str1 returns the c-string (starting address) for str1

Overloading operator[]

- Usually done with two MEMBER functions
- Format: returntype operator[] (indextype index) const

  returntype& operator[](indextype index)
- The const member function allows us to read the element from a const object
- The non-const member function returns a reference to the element that can be modified
- See sample5.cpp
- The address operator can be overloaded just like any other operator (sample6.cpp)