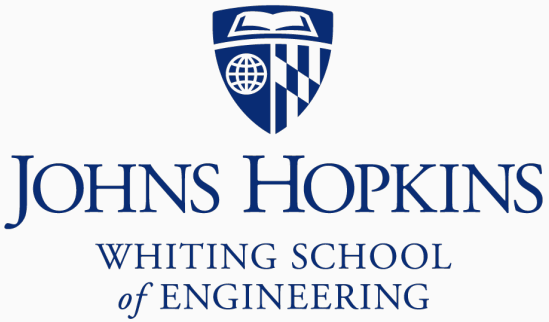
C++ Strings

Ben Langmead

ben.langmead@gmail.com

www.langmead-lab.org



Source markdown available at github.com/BenLangmead/c-cpp-notes

C++ Strings

C++ strings have similar user-friendliness of Java/Python strings

Spare us from details like null terminators

(We will still need C strings sometimes, e.g. `char *argv[]`)

C++: `string`

Use `#include <string>` to use C++ strings

Full name is `std::string`; or put `using std::string;` at the top of `.cpp` file

C++: `string`

`s[5]` accesses 6th character in string

`s.at(5)` does the same, additionally doing a "bounds check"

- Like Java's `ArrayIndexOutOfBounds` or Python's `IndexError`

C++: string

```cpp
#include <iostream>
#include <string>

using std::cout;
using std::endl;
using std::string;

int main() {
    string s("Nobody's perfect");
    for(size_t pos = 0; pos <= s.length(); pos++) { // too far
        cout << s.at(pos);
    }
    cout << endl;
    return 0;
}
```

C++: string

```
$ g++ -c string_at.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o string_at string_at.o
$ ./string_at
terminate called after throwing an instance of 'std::out_of_range'
  what():  basic_string::at: __n (which is 16) >= this->size() (which is 16)
```

Better to use `assert(...)` to check that you're in-bounds. But `s.at(x)` (instead of `s[x]`) is another way to be cautious.

C++: string

Some ways to initialize a new string variable:

```
string s1 = "world";  // initializes to "world"
string s2("hello");   // just like s2 = "hello"
string s3(3, 'a');    // s2 is "aaa"
string s4;            // empty string ""
string s5(s2);        // copies s2 into s5
```

C++: `string`

`strings` can be arbitrarily long

The C++ library worries about the memory

- Dynamically allocated and adjusted as needed
- When `string` goes out of scope, memory is freed

Automatic handling of heap memory is a major advantage of C++

- We will leverage it for our own `classes` later

C++: string

Assuming s, s1 and s2 are std::string's:

```
s = "wow"          // assign literal to string
cin >> s           // put one whitespace-delimited input word in s
cout << s          // write s to standard out
getline(cin, s)    // read to end of line from stdin, store in s
s1 = s2            // copy contents of s2 into s1
s1 + s2            // return new string: s1 concatenated with s2
s1 += s2           // same as s1 = s1 + s2, also same as s1.append(s2)
== != < > <= >=    // relational operators; alphabetical order
```

C++: string

```
string s = "hello";
cout << s.length() << endl; // prints 5

// prints bytes of memory allocated
cout << s.capacity() << endl;

// s.substr(offset, howmany) gives substring of s
cout << s.substr(1, 3) << endl; // prints "ell"

// s.c_str() returns C-style "const char *" version
cout << strlen(s.c_str()) << endl; // prints 5
```

C++: `string`

See C++ reference for more `string` functionality

- www.cplusplus.com/reference/string/string/

Commonly used member functions (click for links):

- `length` – return # of characters (ignoring terminator)
- `empty` – return true when there is at least 1 character
- append – like +=
- push_back – like append for a single character
- `clear` – set to empty string
- `insert` – insert one string in middle of another
- `erase` – remove stretch of characters from string
- `replace` – replace a substring with a given string