String Processing

CS 1044

Strings

- We've been treating **strings** as opaque chunks of text
- Haven't really delved deeply into how they work or how they are built
- The C++ `string` datatype hides a lot of nasty low-level details from you

How Strings Are Built

- A `string` is an array of `char`s
- The word `"hello"` is actually **6** characters – the 5 letters, followed by a **null** character indicating the **end** of the string
- Using the `string` data type means you don’t have to deal with the underlying array or null character

Accessing a String

- Strings provide `[i]` notation to read or change individual characters, like an array or vector

```
string str = "hello";
char c = str[3];

str[0] = 'm';
```

Strings use double quotes

Characters use single quotes

Length of a String

- The `length` method (not `size`, like a vector!) returns the number of characters in the string

```
string str = "hello";
int n = str.length();     // n == 5
```

- The trailing null character is **not included** in the count

Joining Strings Together

- Joining strings together is easy – use the + operator

```
string m = "mello";
string y = "yello";
string not_mtdew = m + " " + y;
```

- Strings are joined verbatim – no space added unless you do so explicitly

Caution when Joining Strings

- Problems occur if you try to join string literals together

```
string s = "mello " + "yellow";
```

- Workaround: wrap the first one in an explicit call to `string`

```
string s = string("mellow ")
                              + "yellow";
```

Strings and Loops

- Let’s say we want to “double” each character in a string:

```
string orig = “Hello World”;
string doubled = "";

for(int i = 0; i < orig.length(); i++)
{
   doubled = doubled + orig[i] + orig[i];
}
cout << doubled << endl;
```

- This will print: “HHeelllloo WWoorrlldd”

Strings and Loops

- Let’s say we want to reverse the order of the characters in a string:

```
string orig = “Hello World”;
string reversed = "";

for(int i = 0; i < orig.length(); i++)
{
   reversed = orig[i] + reversed;
}
cout << reversed << endl;
```

- This will print: “dlroW olleH”

Strings and Loops

- Let’s say we want to grab every other character in a string:

```
string orig = “Hello World”;
string everyOther = "";

for(int i = 0; i < orig.length(); i+=2)
{
    //if ((i % 2) == 0)
    //{
    everyOther += orig[i];
    //}
}
cout << everyOther << endl;
```

- This will print: “HloWrd”

Finding Characters in a String

- Strings provide the `find` method (function) to find individual characters or strings.
- The `find` method (function) returns either the index or a special constant `string::npos`.

```
string str = “hello";

// pos should be 0, the beginning of “he”
int pos = str.find("he");

// pos should be string::npos since
// “z” isn’t in the string.
int pos = str.find("z");
```

Strings and Loops

- There several variants of the find function. There are also methods (functions) to make sub-strings.

```
// find all the “#” symbols in a tweet. “end” is the end
// of the last hashtag. So that’s where start searching.
while ((pos = tweet.find("#", end)) != string::npos)
{
    // hashtag ends with a space, so find the first one
    end = tweet.find_first_of(" \n\t", pos);
    // isolate just a hashtag using substring method
    string hashtag = tweet.substr(pos + 1, end);

    if (hashtag != "")
    {
        cout << hashtag << endl;
    }
}
```

String Streams `#include <sstream>`

- Streams we've seen so far:
  - Screen/keyboard (`cin`, `cout`)
  - Files on disk (`ifstream`, `ofstream`)
- Can also have streams that break apart strings or create new strings
- Useful for **parsing** or data **conversion**

Input String Streams

- istringstream lets you **read values from a string** using standard input operations like >>

```
string str = "21 Jump Street";
istringstream stream(str);
int num;
stream >> num;   // num == 21
```

Creates a stream that reads from the string `str`

- Very useful when you have data already in memory as a string and need to extract numbers from it

Output String Streams

- `ostringstream` lets you **generate strings** using standard output operations like <<

```
int num = 99;
ostringstream stream;
stream << num << " Luftballons";
string s = stream.str();
```

Gets what was written to the stream as a string

- Useful when you need to convert a number to a string, or include one in a larger string

String streams and Loops

- We can use string streams to isolate hashtags also:

```
// Tweet comes from somewhere.
stringstream split_tweet(tweet);

// Break a tweet down into words.
while (split_tweet >> word)
{
        if (word.find("#") == 0)
        {
            // Use substr to get everything after the ‘#’.
            string hashtag = word.substr(1);

            if (hashtag != "")
            {
                    cout << hashtag << endl;
            }
        }
}
```