

CSE 333 Section 9 - HW4 Tools: telnet and boost

Welcome back to section! We're glad that you're here :)

Boost Library

- Very useful for dealing with strings (HW4!), such as trimming, pattern matching, splitting, replacing, etc.
- `#include <boost/algorithm/string.hpp>`.
- Boost is template madness, so we have handy reference information here :)

```
using namespace boost;
```

```
// Trim the whitespace off the left and right of to_modify
// to_modify is input and output parameter
void boost::trim(string& to_modify);
```

```
// Replace all instances of to_find with to_replace.
// to_modify is input and output parameter
void boost::replace_all(string& to_modify,
                        const string& to_find,
                        const string& to_replace);
```

```
// Returns a predicate that matches on any of the characters in tokens
boost::PredicateT boost::is_any_of(const string& tokens);
```

```
// Split the string by the characters in match_on
void boost::split(vector<string>& output,
                  const string& input,
                  boost::PredicateT match_on,
                  boost::token_compress_mode_type compress);
```

Examples

```
TRIM:   string s(" \t HI \n ");
        boost::algorithm::trim(s); // s == "HI"
```

REPLACE_ALL:

```
string s1("ynrrrt");
boost::algorithm::replace_all(s1, "nr", "e"); // s1 ==
"yeet"

string s2("queue?");
boost::algorithm::replace_all(s2, "que", "q"); // s2 ==
"que?"
```

```
SPLIT:  string str1("hello abc-*--ABC-*--aBc goodbye");
        vector<string> SplitVec; // #2: Search for tokens
        split( SplitVec, str1, is_any_of("-*"), token_compress_on );
        // SplitVec == { "hello abc", "ABC", "aBc goodbye" }
```

Exercise 1

Write a function called `ExtractRequestLine` that takes in a well-formatted HTTP request as a string and returns a map with the keys as `method`, `uri`, `version` and the values from the corresponding request. For example,

Example Input:

```
"GET /index.html HTTP/1.1\r\nHost: www.mywebsite.com\r\nConnection: keep-alive\r\nUpgrade-Insecure-Requests: 1\r\n\r\n"
```

Map Returned:

```
{
    "method"    : "GET"
    "uri"       : "/index.html"
    "version"   : "HTTP/1.1"
}
```

```
map<string,string> ExtractRequestLine(const string& request) {
```

```
}
```

Exercise 2

Write a function that takes in a string that contains words separated by whitespace and returns a vector that contains all of the words in that string, in the same order as they show up, but with no duplicates. Ignore all leading and trailing whitespace in the input string.

Example:

`RemoveDuplicates(" Hi I'm sorry jon sorry hi\nhihi hi hi")` should return the vector `["Hi", "I'm", "sorry", "jon", "hi", "hihi"]`

```
vector<string> RemoveDuplicates(const string& input) {
```

```
}
```

Exercise 3 (Extra Practice)

Write a function called `CountHttpMethods` that takes in a `vector` of well-formatted HTTP request as a `string` and returns a `map` with all of the counts for each HTTP method used in the requests.

Example Input:

```
[  
"GET / HTTP/1.1\r\nHost: attu.cs.washington.edu:3333\r\nConnection: keep-alive\r\nUpgrade-Insecure-Requests: 1\r\n\r\n",  
  
"POST / HTTP/1.1\r\nHost: www.google.com\r\nConnection: close\r\nUpgrade-Insecure-Requests: 1\r\n\r\n"  
]
```

Example Map Returned:

```
{  
    GET    : 1  
    POST   : 1  
}
```

Exercise 4 (Extra Practice)

Write a function called `ParseHeaders` that takes in a well-formatted HTTP request as a string and returns a map of all the header-value mappings inside the request:

Example Input:

```
"GET / HTTP/1.1\r\nHost: attu.cs.washington.edu:3333\r\nConnection: keep-alive\r\nUpgrade-Insecure-Requests: 1\r\n\r\n"
```

Map Returned:

```
{  
  Host : attu.cs.washington.edu:3333  
  Connection : keep-alive  
  Upgrade-Insecure-Requests : 1  
}
```