

# CSE 333 Section 8 - HW4 Tools: telnet and boost

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

## ***Boost Library***

### **Exercise 1**

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 hihi hi hi ")  
should return the vector ["Hi", "I'm", "sorry", "jon", "hi", "hihi"]

```
// There are other and better ways to solve this, but this
// solution uses only things from the final exam reference sheet
vector<string> RemoveDuplicates(const string& input){
    string copy(input);
    boost::algorithm::trim(copy);
    std::vector<string> components;
    boost::split(components, copy, boost::is_any_of(" \t\n"),
                 boost::token_compress_on);

    std::vector<string> result;
    for (uint i = 0; i < components.size(); ++i) {
        bool unique = true;
        for (uint j = 0; j < i && unique; ++j) {
            unique &= !(components[i] == components[j]);
        }
        if (unique) {
            result.push_back(components[i]);
        }
    }
    return result;
}
```

```
// Alternate Solution
vector<string> RemoveDuplicates(const string& input) {
    string copy(input);
    boost::algorithm::trim(copy);
    vector<string> components;
    boost::split(components, copy, boost::is_any_of(" \t\n"),
                 boost::token_compress_on);

    map<string, bool> aux;
    vector<string> res;
    for (auto comp : components) {
        if (aux.find(comp) == aux.end()) {
            res.push_back(comp);
            aux[comp] = true;
        }
    }
    return res;
}
```

## Exercise 2

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
}
```

```
#include <map>
#include <string>
#include <vector>
#include <boost/algorithm/string.hpp>

using std::map;
using std::string;
using std::vector;

map<string, string> ParseHeaders(const string& request) {
    vector<string> lines;
    boost::split(lines, request, boost::is_any_of("\r\n"),
        boost::token_compress_on);

    map<string, string> res;
    vector<string> components;
    for (uint i = 1; i < lines.size(); i++) {
        string currLine = lines[i];
        size_t index = currLine.find(":");
        if (index == string::npos)
            continue;

        string key = currLine.substr(0, index);
```

```
    string val = currLine.substr(index + 1);  
    res[key] = val;  
}  
return res;  
}
```

### Exercise 3

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\nConnect
ion: 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
}
```

```
#include <map>
#include <string>
#include <vector>
#include <boost/algorithm/string.hpp>

using std::map;
using std::string;
using std::vector;

map<string,int> CountHttpMethods(vector<string> requests) {
    map<string, int> res;
    for (string& request : requests) {
        vector<string> lines;
        boost::split(lines, request, boost::is_any_of("\r\n"),
                     boost::token_compress_on);

        string firstLine = lines[0];
        vector<string> components;
```

```
boost::split(components, firstLine, boost::is_any_of(" "),
             boost::token_compress_on);

string method = components[0];
if (res.find(method) == res.end()) {
    res[method] = 1;
} else {
    res[method]++;
}
}
return res;
}
```

#### **Exercise 4**

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.

#### **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"  
}
```

```
#include <map>  
#include <string>  
#include <vector>  
#include <boost/algorithm/string.hpp>  
  
using std::map;  
using std::string;  
using std::vector;  
  
map<string, string> ExtractRequestLine(const string& request) {  
    vector<string> lines;  
    boost::split(lines, request, boost::is_any_of("\r\n"),  
                 boost::token_compress_on);  
  
    vector<string> components;  
    string firstLine = lines[0];  
    boost::split(components, firstLine, boost::is_any_of(" "),  
                 boost::token_compress_on);  
  
    map<string, string> res;  
    res["method"] = components[0];  
    res["uri"] = components[1];  
}
```

```
    res["version"] = components[2];  
    return res;  
}
```