# CS/240/Lab/2

In the next three labs we will look at creating a Twitter trend analyzer. We will write a program that pulls tweets and counts the frequency of each unique word. We start with a parser that will filter out frequently used but uninteresting words like `the`, `of`, and `to`.

Our parser will be made of three files. The main program is in a file called `parser.c`. This program will call several functions to parse the input. These functions are in a library file called `libwords.a`. The functions in `libwords.a` use several basic utility functions. These functions are defined in a library that you will write called `libutils.a`.

This lab is structured in two parts. In the first part we will write the simple string utility functions that compose `libutils.a`. In the second part we will link to this library and the `libwords.a` library (which we will provide you) to create a filter program that eliminates uninteresting words.

## Q1: String Utils

This part of the lab will walk you through the design, creation and testing of a library of reusable functions. Your job is to implement four utility functions (`isAlpha()`, `strLen()`, `strNCmp()` and `spaceOut()`) of `libutils.a`. The specification and interface of these functions is as follows:

```
int isAlpha(char ch)
```

*Description*: Test if `ch` is a letter (a - z or A - Z). Return 1 if yes, otherwise 0.

```
int strLen(char* s)
```

*Description*: Return the length of the zero terminated string `s`. The `\0` character is not counted, e.g. the length of `"abc"` is 3.

```
int strNCmp(char* s1, char* s2, int n)
```

*Description*: Compare the first `n` characters of string `s1` and `s2`. Return 1 if identical, 0 otherwise.

```
void spaceOut(char* s, int len)
```

*Description*: Replace every character of `s` with a white space. `len` indicates the length of `s`.

Download the skeleton file `utils.c` from Piazza, it has the stubs for these functions, you will only need to fill-in the bodies of the functions.

Testing a library is different than testing a complete program. Piazza has some useful goodies to help you out. The file `parser_ref.o` is a driver for your library, you can use it to check that your functions are correct. (The file `libwords.a` contains a library required by `parser_ref`.) The trouble with a driver is that if there is an error in your code, it will not tell you which of your functions was wrong. So, we encourage you to write unit tests for each of your functions.

**Compiling**

To compile and build your library, do the following:

```
gcc -std=c99 -c utils.c
ar rcu libutils.a utils.o
gcc -o parser_ref parser_ref.o -L. -lwords_ref -lutils
```

This gives you an executable `parser_ref` which can be used as follows:

```
cat msg.txt | ./parser_ref  the of in
```

The autograder used to evaluate your assignment will be based on unit tests of your functions. To beat it, try to think of all possible inputs that could be given to your functions. You can assume that the inputs are well formed, i.e. if the interface of the function requires a zero terminated string, it will be called with one.

**Unit Testing**

A unit test is a test of a single function or component in a larger system. We encourage you to create a `tester.c` file with unit tests for all of the funtions of your library. Here is a start, with a single test. You must add more.

```
#include<stdio.h>
extern int isAlpha(char);

testIsAlpha() {
    if(isAlpha('a')==0) printf("Error a is alpha\n");
    if(isAlpha('Z')==0) printf("Error Z is alpha\n");
    // more cases ...
}

int main(){
  testIsAlpha();
  // add other tests here
}
```

Compile it with:

```
gcc -std=c99 -o tester tester.c utils.c
```

# Q2: Parser

In this part of the lab you will write the parser. To make your life easier we provide you with a file `parser.c` that has the algorithm described, as well as reference version of the string utils library, `libutils.a`, and the words library, `libwords.a`.

The parser takes a list of uninteresting words from the command line and reads tweets from its standard input. For our purposes, you can assume that tweets come in as a sequence of lines terminated by a line break (`\n`). Each line will be at most a 140 characters long (`\n` included).

The parser is a filter that identifies uninteresting words in its input and replaces them with spaces in the output. For example, if we gave the parser the following uninteresting words, `"a"`, `"an"`, `"i"`, `"in"`, then for the following tweets

```
$ cat msg.txt
In CS240, I got an A.
I learn C in CS240, while Java in CS180.
```

the expected output is:

```
$ cat msg.txt | ./parser a an i in
   cs240,   got     .
  learn c    cs240, while java    cs180.
```

The parser must turn the entire message to lower case before starts killing the keywords, so that, for example, with `"in"` specified as a keyword, all `"In"`, `"in"`, `"iN"`, and `"IN"` will be spaced out. Note that the length of the message as reported by `wc` should remain the same, every occurence of an uninteresting word is now replaced with a number of spaces that matches the length of the word.

**Command line arguments**

Here is how to access command line arguments. The parameter `argc` is the number of arguments, and `argv` is an array of zero-terminated argument strings. By convention, `argv[0]` is the name of the program and the rest are its command line arguments. Here is how to print command line arguments:

```
int main(int argc, char* argv[]){
  for(int i=0; i<argc; i++) printf("%s\n", argv[i]));
  return 0;
}
```

The parser skeleton is provided in `parser.c`. Fill in the blanks using the following helper functions (from `libwords.a`):

`int readMsg(char* buf)`

*Description*: Read one line from standard input into `buf`. (`\n` will be put in `buf` too if there is one.) Return the length of the message. Return EOF if there are no more lines to read.

`int getWord(char* msg, int len, int* start, int* end)`

*Description*: Find the next word. `msg` is a pointer to an array of characters. `len` is the length of the `msg` array. `end` is a pointer to an integer that holds the current position in the array of characters. The function will find the next word, starting from the position given by the initial value of `end`.
If a word was found the function returns 1, and updates `start` to point to the index of the first character in the word, `end` to point to the index of the first character after the word.
If no word was found, the function returns 0 and `end` points to the index of the last character in `msg`.

`void checkWord(char* word, int len, char** keywords, int num)`

*Description*: Check if `word` is contained in `keywords`. If yes, erase `word` with white spaces. `len` indicates the length of `word` and `num` indicates the length of the keyword list.

`void unCapitalize(char* c)`

*Description*: Change the character pointed by `c` to lower case if it is a capital.

**Compiling & Testing**
When testing your `parser.c`, do

```
gcc -std=c99 -o parser parser.c -L. -lwords_ref -lutils_ref
```

To test with messages in `msg.txt` and keywords from `keys.txt` (which you must create), use

```
cat msg.txt | ./parser $(cat keys.txt)
```

# Turning in

Go to the course web page ( `http://web.me.com/vitekj/240s12/` ), and under the "submissions" section click "web site". On the submission web page, enter your unique turnin ID in the box and press "log in". Under "currently open projects" will be listed "lab 2 utils" and "lab 2 parser". Click "lab 2 utils", use the file selector to choose your `utils.c` file, then click "submit". The page will tell you how you've done, and your total score. When you're satisfied, click "return to list of projects", then "lab 2 parser". Use the file selector to choose your `parser.c` file, then click "submit". When you're satisfied, close your browser window. You may resubmit at any time before the due date by the same process.

Lab is due Monday, January 30th before midnight. No late labs accepted.

## Grading criteria

### String Utils

- source file is named `utils.c`
- code compiles
- code runs without error
- code only includes `stdio.h`
- `isAlpha()` - returns 1 if given an alphabetical character, 0 otherwise
- `strLen()` - returns the correct length of a 0-terminated string (excludes `\0`)
- `strNCmp()` - returns 1 if the first n characters of two strings are identical, 0 otherwise
- `spaceOut()` - replaces every character of string `s`, up to length `n`, with a space character

### Parser

- source file is named `parser.c`
- code compiles
- code runs without error
- code only includes `stdio.h` and `words.h`
- program output has the same character count as the input
- program filters out all target words in the command line argument list from the standard input
- program filters out capitalized versions of the targets words