CS 2230 | **Programming Assignment #2**
Bruce Bolden | **10 Points**
October 30, 2025 | **Due:** November 6, 2025

Objective: The purpose of this assignment is to write a function, in C, that will *parse* a string into *tokens* (or *words*), similar to what the shell is required to do.

Description: For this assignment, you are to write a function with the following prototype:

```
    int makearg(char *s, char ***args);
```

or

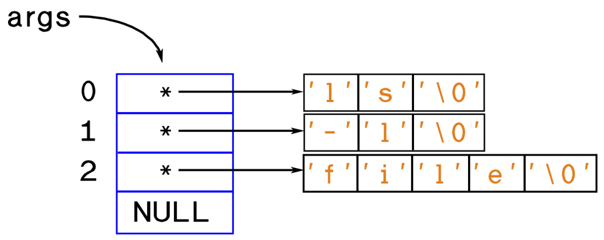
```
    int makearg(char s[], char **args[]);
```

(Yes, this *is* the correct prototype—don't change it!) This function should accept a (C-type) string and a pointer to a pointer to `char` (or, if you prefer, a pointer to an array of pointers to char) (i.e., a pointer to the same type as `argv` in a C program), and should return the pointer defined to point to an array pointing to the separate tokens on the command line, as well as the number of tokens found as the function return value. If some problem occurred during the operation of the function, the value returned should be `-1`. **NOTE:** No C library string functions may be used, e.g., `strlen()`, `strcat()`, `strtok()`, etc.

For example, if you were to call the function using the following code:

```
    char **args, str[] = "ls -l file";
    int argc;
    argc = makearg(str, &args);
```

it should produce the following configuration in memory:



The important part of this exercise is the function you write. However, for consistency in grading, your main program should be written to input a line, call the function, then print out the number of arguments found, and finally, display the arguments, one per line.

Note that, as in the example above, the value of the pointer argument is undefined when it is passed to the `makearg()` function. `makearg()` must allocate any necessary dynamic memory necessary for use by the program. (The array used for the unparsed string need not be dynamically allocated.) **Note:** `strtok()` may not be used for parsing.

Deliverables:

- A program design sheet. Describe all functions needed to implement your program.
- Programming Log:
  - Record the time required to design and implement your program.
  - Record of things you encountered/learned while implementing your program.
- Output—proof that your program worked.
- Program—fully documented.