

# Programming Assignment 1

---

**Due** Oct 3 by 10pm      **Points** 100

---

## Overview

This assignment serves as an introduction to programming in C beyond the smaller exercises in the Lab assignments. Even so, as the first assignment, the programs that you will write are smaller than you may be used to and smaller than what is to come. Do not, however, let the size of the programs lead to complacency. Take the opportunity to emphasize a focused understanding, to explore the use of new tools to aid in your programming, and to reenforce concepts from prior coursework and projects. Challenge yourself to develop a quality solution that goes beyond a solution that just works.

## Course Learning Objectives

This assignment addresses the following course learning objectives.

### Direct


- Read and write complex programs in the C programming language.
- Transition to upper-division coursework where programming is a tool used in the exploration of course-specific content.
- Transition from following stepwise, prescribed decompositions to independent problem solving.

### Indirect

- Use standard UNIX user-level commands and the UNIX software development environment.

## Setup

Clone the repository.

- Accept the [GitHub Classroom Assignment 1 invitation](https://classroom.github.com/a/wOMtR306)    
(<https://classroom.github.com/a/wOMtR306>).
- Clone the repository.

## Tips

- Use a debugger to inspect your data or print the data to verify that your program has stored what you intended it to.
- Program incrementally. Verify that each component works before integrating them.

## The Tasks

There are two separate tasks listed below. Each task asks that you write a program to solve a "small" problem (each emulates a program commonly found in Unix installations) and while they are different programs, there is some (a little) common functionality between them. You are *strongly* encourage to identify this commonality and to move it into a source file shared between the two programs (a set of common utility or helper functions).

### Task 1 - word count

The `wc` program is a common utility available in many Unix installations. This program is rather simple in concept: it counts the number of characters, words, and lines in its input. Of course, to properly manage this we must adopt some definition of a "word" and of a "line" (the character count includes all characters in the input).

#### Definitions:

- word: whitespace delimited sequence of characters (use `isspace()` to check for such characters)
- line: sequence of characters terminated by a newline character

Write a program (with `main` in a file named `word_count.c`) that takes a single (optional) command-line argument specifying a file name. The program must read the contents of the file (read from `stdin` if no file is specified) and print (only) the number of lines, number of words, and number of characters read (the format of the output is up to you).

There should be only a single function that reads the input; do not duplicate code.

**Note:** your program cannot dynamically allocate memory (e.g., by using one of the `*alloc` functions). It should read a single character at a time.

### Task 2 - unique lines

The `uniq` program is a common utility available in many Unix installations. This program echoes (to `stdout`) its input, but will print only a single instance of consecutive matching lines. This is to say that if the input contained the following sequence of lines, then only a single instance of "This is a line." from the first group of matching lines will be printed, followed by "This is a different line.", and then "This is a line." (though it matches the first three, it was not part of the consecutive sequence).

This is a line.

This is a line.

This is a line.

This is a different line.

This is a line.

Write a program (with `main` in a file named `unique.c`) that takes a single (optional) command-line argument specifying a file name. The program must read the contents of the file (read from `stdin` if no file is specified) and print the contents of the input stream to `stdout` such that only a single instance of consecutive matching lines is printed.

There should be only a single function that reads the input; do not duplicate code.

**Note:** your program cannot explicitly dynamically allocate memory (e.g., by using one of the `*alloc` functions). It should use the `getline` function. A valgrind analysis of your program must not report any issues.

## Deliverables

- Source Code - Push all relevant source code to your repository.

### Programming Assignment 1 (1)

Criteria	Ratings		Pts
word count: functionality	45 pts Full Marks	0 pts No Marks	45 pts
word count: program style The source code follows general best practices in terms of line length, function decomposition (non-duplication), naming principles, etc.	5 pts Full Marks	0 pts No Marks	5 pts
unique: functionality Clean valgrind report on the department servers.	40 pts Full Marks	0 pts No Marks	40 pts
unique: program style The source code follows general best practices in terms of line length, function decomposition (non-duplication), naming principles, etc.	5 pts Full Marks	0 pts No Marks	5 pts
Valgrind Check valgrind reports no issues.	5 pts Full Marks	0 pts No Marks	5 pts
Total Points: 100			