Small Group 0 Software Development Project for CS 4500, Fall 2025 2 or 3 person teams

Small Group 0 (SG0) is the semester's first CS 4500 group software development project. You will be in a group of 2 or 3 developers. You should log all the time you spend on this project in the time log described in the syllabus. In addition to working on your time log during this project, you will also be spending quite a bit of time communicating with other team members, and documenting that communication.

You must communicate with other members of your team each weekday after the assignment is mentioned in the schedule until your group submits the assigned software; and you must document that communication. Your first required communication is Friday, Sept. 5.

You are REQUIRED to communicate with your teammates EVERY WEEKDAY starting on Friday, Sept. 5, and ending when your group submits the assignment. This communication can be a phone call, an email, a Zoom session, a post on Discord, a face-to-face meeting, or any other means of communication your team decides is appropriate. You are allowed to communicate on the weekend, and you are not limited to one communication per weekday. But it is part of this assignment to communicate with your teammates at least once every weekday. No exceptions. If you don't, your semester grade will be slightly lowered for each day that you don't accomplish a documented communication with your team.

To enforce the communication requirement, you are required to notify me of at least one communication with your group each weekday. Although the communication between team members can take many forms, the notification to me must be via email. Do NOT use my usual UMSL email address for this notification. If you use my UMSL email for this notification, the notification will be ignored, and you will not get credit for the notification. Instead use this email for the notification:

CS4500instructor@yahoo.com

The subject heading should be "notification" without the quotes. The message in each notification email to me must be in this form:

X communicated with Y via Z

…where X is your name, Y is either a single name or a list of names, and Z describes how you communicated. Each name should have a first name and a last name, as they appear on our Canvas website. Here are three examples of legal notification messages, assuming that the named people were really in your small group:

Fred Zilphath communicated with Ethyl Mermen via email

Angie Doyle communicated with Larry Robin, Marie Koster, and Edna Lumbar via Discord

Mary Boyd communicated with Bonnie McNight via a face-to-face meeting

The notification shouldn't include a date because the notification should be sent on the day of the interaction. If the notification does not include the phrase "communicated with" or the word "via," it is not a valid notification. If you spell either of those words incorrectly, it is not a valid notification. If the message does not include your first name and last name, it is not a valid notification. If it does not include at least one of your team members' names, then it is not a valid notification. You only get credit for valid notifications. As you might guess, I am being picky about the format of these notifications because the notifications will be tabulated automatically, and the strict format facilitates that process. You only get credit for one notification per weekday, so don't bother to send multiples on the same day. Also, if you forget to send a notification, you may NOT make it up by sending the notification the next day.

I'm not going to lie to you; this notification system is a pain. Yes, it is a pain for you, but it is an even bigger pain for me. (There are over 40 students or so in this class, so you can imagine how many of these notifications I'll be dealing with.) However, there is a reason for all this pain. The number one most annoying thing about being in a student group project is having a team member who disappears during the project. Students HATE that, and I don't blame them. This notification system is my latest method of trying to enforce the idea that group members MUST communicate with each other daily (during the week) while they are on a team. In CS 4500, you will communicate with your team, or your grade will suffer. And you will document that communication using valid notifications (as detailed above), or your grade will suffer.

WHAT YOUR SG0 PROGRAM SHOULD DO

Your program (also known as SG0) should begin by printing to the screen a short explanation of what the program will do. The explanation should take up one screen or less.

Next, SG0 should ask the user for the name of a file. The file should be a text file, and its name should end in ".TXT" without the quote marks.

SG0 should check to make sure that the file name the user enters ends in .TXT. (I will use upper case only, but the user doesn't need to worry about upper or lower case, and SG0 should accept any mixture of upper and lower case for the TXT at the end of the filename. If the name that the user enters does not end in .TXT (or .txt or .TxT, or…), then SG0 should give an appropriate error message and reprompt for the filename. Later in the semester, you may have to ascertain if the file only contains valid ASCI text. But for SG0, you may assume that the file contains only text.

Once the user has entered a valid filename, SG0 should attempt to open the named file for reading. The named file should reside in the same directory as where SG0 resides. If the file does not exist in the directory where SG0 resides, then SG0 should give the user an appropriate error message, and reprompt for another valid file name. This should continue until SG0 successfully opens a .TXT file designated by the user. I will call that file F in this specification, even though the name could not be F. (Yes, it could be F.TXT, but it is so much easier to just use F in this specification.)

Under the assumption that F contains only text, your SG0 should read in that text, and break it up into a long list of words. A word is defined as a series of alphabetic characters, uninterrupted by a blank or a punctuation mark. The only punctuation mark allowed inside a “word” is a hyphen. Thus “first-base” is considered a word, whereas “first base” is considered two words. If the hyphen is used to separate words such as in the following sentence:

I always loved that – it was simple, but beautiful.

…then the hyphen is NOT used to make “that” and “it” into a single word. The sentence above has 9 words, by this definition.

If a line of text in F ends in a hypen, without a blank before the hyphen, then the word at the end of that line and the word that starts the next line are combined into a single word. Example:

```
This is a line that ends in a hyphenated-
word. That is strange, but it is how the rule works.
```

In that example, for this definition of “word,” “hyphenated-word” counts as a single word. In the next example:

```
That may be at least as strange –
that hyphen does NOT make a long word.
```

Because there is a blank between the word strange and the hyphen at the end of that line, the hyphen does NOT tie together two words.

All the words in the text file F should be placed into some sort of data structure in your SG0. I will call it a “list,” but you are not required to use a Phython 3 list data structure to hold those words, although you may. In this specification, I will call that list of words ListOfWords.

[POSSIBLE LOOP STARTS HERE:]

Next, SG0 should politely prompt the interactive user for a “word,” as defined above. It should contain only these characters:

LegalCharacters = 'abcdefghijklmnopqrstuvwxyz-'

However, the alphabetic characters can be upper or lower case. If the user types in a string that contains any other characters, SG0 should point out the first problem in the string, and politely reprompt. For example, no blanks are allowed in the word the user types in. Each time you prompt for a word, including the first time, make sure to clarify the rules for a legal word. I will call the legal word the user enters LegalWord in the rest of the specification.

Once the user enters a legal word, SG0 should search ListOfWords to find out how many times the LegalWord occurs in the ListOfWords. Upper or lower case should be ignored when making this count. For example, if LegalWord is UMSL, and if ListOfWords includes “UMSL” twice, and includes “UmSL” and “umsl” once each, and if there are no other variations of UMSL in the ListOfWords, then SG0 should count 4 as the number of times UMSL appears in the list of words.

SG0 should report to the interactive user how many times LegalWord appears in F. On the next line of screen output, SGO should ask if the interactive user wants to continue entering words to count. The user is allowed to answer Yes, No, yes, no, y, Y, n, or N (with the meaning of "yes" or "no," which I hope is obvious). If the user enters anything other than those 8 strings, your SG0 should give a polite error message and reprompt for one of those 8 strings. If the user indicates a Yes, then SG0 prompts for another LegalWord and repeats the process marked "POSSIBLE LOOP STARTS HERE" above. If the user indicated a No, then SG0 prints out a nicely formatted list of all the LegalWord entries the user entered during this session (there should be at least one, but there could be several), in a nicely formatted list, one LegalWord and its count on each line. Then SG0 announces that it is finished, and announces that the interactive user should push ENTER to end the program. When the user pushes ENTER, SG0 ends.

MORE INSTRUCTIONS ABOUT WHAT TO SUBMIT

This assignment is designed for a group of 2 or 3 people. (There will be one group of 3 if the class has an odd number of students.) Each group should submit one zipped file. Only one member of the group should submit the file. The zipped file should include three files:

1. Functioning Python 3 code (formally known as a script). More requirements for your code will be given below. Your code must be contained in a single file, not multiple files.
2. A separate design document that has a title page that lists the name of our class, the number of your group, and the people in your group. The document should have page numbers. There should be a list of any major revisions (or the date of first design, if there were no revisions). Starting on a new page, there should be a pseudo code design of the program. It should be at a high level of abstraction so that it fits on one page, single spaced. If you have never done a pseudocode design, or you would like a reminder of what it is exactly, please see https://www.geeksforgeeks.org/what-is-pseudocode-a-complete-tutorial/ . That website will be used as the grading standard for your design.
3. A separate test plan document that has a title page that lists the name of our class, the number of your group, and the people working on this program. There should be a list of any major revisions (or the date of first test plan, if there were no revisions.). Starting on a new page, there should be a list of tests that you planned and then executed before turning in your assignment. Each test should include, well-marked, the precise input that the test will use, and the precise behavior you expect to occur after that input. (NOTE: try to NOT just describe the input and output; specify it EXACTLY. For example, don't say "test with legal word;" instead, say "test with the word ExpeRt." There should also be a record of when the test was done, and what the result was. If the program does NOT produce the expected output, that failure should be recorded. If multiple tests executions are required before success, each should be recorded, including who did the test, when, and the result. The document should have page numbers. For this program, inputs will be either user responses to prompts, or a file that will be processed by SG0. When specifying a file in a test plan, it is NOT permitted to merely vaguely describe the file; instead, you must include everything in the file, item by item, line by line.
4. All of these deliverables should be collected into a single zipped file. Each 2 or 3 person team will submit one such zipped file.

The group should cooperate on such things as a common format for cover pages, pseudocode design, and for test plans. They should help each other to get everything submitted on time and in working order, because a part of everyone's grade depends on the whole group's accomplishments.

MORE ABOUT THE DOCUMENTATION IN YOUR SOURCE CODE

Start your Python 3 source code with an extensive opening comment. The first line of that opening comment must describe which programming language you are using. You should also mention what development system (or IDE) you used. (We'll be using Thonny to grade your program, but you can use a different IDE to develop if you want.)

The rest of the opening comment must include the names of all the team members working on the program, the date of submission (or the date you started the program, and major revision dates), the name and number of our class, an explanation of what the program is designed to do, a description of the central data structure or structures in your program, and an explanation of any external files used.

Every function in your code should include a short opening comment. That comment should include what the function accomplishes, any global variables it uses, what each parameter is and does, and what, if anything, the function returns to the caller.

If you use any outside resources to develop your program, for example looking up programming language details on the Web, then your opening comment should explicitly list those sources, and give at least a little information about what you found at each resource. If you don't use ANY outside resources during development (and that would greatly surprise me), then indicate that no outside resources were used. Remember, even looking up a syntax detail, or checking out a debug problem in StackOverflow, are examples of using an outside resource. There is absolutely nothing wrong with doing that, but you must document it. Give credit where credit is due. Also, if you adapt or adopt a function or code segment from the Web or perhaps from a previous program you wrote, document it twice: once in your opening comment, and again in an internal comment wherever the borrowed code appears in your program.

Documentation is a central concern when developing high quality software. I am convinced that the best software engineering professionals document their work in excruciating detail. That's exactly what I want you to do for any assignments in CS 4500.

This documentation is going to be *particularly* important to you and your teammates in CS 4500. There are several phases to the SG programs, and they build on each other. You will be in different teams, one for each phase of the project. When you move from your first team to your second team (and from your second team to your third team and so on) you will take your source code and other deliverables with you. These will prove vital in building the next phase of the project, and any documentation you do in Phase X is likely to help you in adapting the software to Phase X+1.

In addition to the extensive opening comment, internal comments should explain important data structures where they are declared and/or used, delineate large sections of code ("paragraphing comments"), and clarify anything clever or hard to understand in your code. We grade documentation critically, so I recommend that you spend some time on your opening and internal comments.

PEER EVALUATIONS ARE SUBMITTED SEPARATELY USING A SEPARATE DROPBOX IN CANVAS

After SG0 has been submitted, each member of your team will send me a confidential assessment of the other team members' contributions to the project. These are peer evaluations. You should NOT assess yourself, only others in your group. The assessments are straightforward. Each team member will submit, via Canvas, a list of the names of the other team members, giving each member a rank of either +1, +2, or -1. After each name (first and last name, please), put a colon and the number. If you only know about the contributions of one other member of the team, then only evaluate that one person. Here are what the three possible peer evaluation ratings mean:

- +1 will indicate that the team member made a good faith effort to contribute to the project. We expect that most of the time, most team members will get a 1 from the rest of the team.
- +2 will indicate that, in your opinion, this particular team member made an extraordinary effort to contribute to the project. You can only award one 2 in each of your peer assessments, and you are not required to award any 2 ranking.
- -1 will indicate that, in your opinion, a team member did NOT make a good faith contribution to the group project.

I hope that there will be very few -1 rankings. But if you think someone does not make a good faith effort to contribute to the group effort, then I want to know. Please submit your evaluations carefully. Don't use a hyphen to separate the name and the evaluation, because that could make a PLUS one look like a MINUS one.

If you or your team have questions about the SG0 project, or about anything else in CS 4500, please email me at millerkei@umsl.edu. Don't be shy… I get paid for this.

Take care, and best of luck on SG0.

Keith

PS. The following page constitutes a grading rubric for SG0. This tells you what's important to me, and helps you understand how we will grade your project.

IF I WERE YOU, I WOULDN'T TURN IN MY DELIVERABLES BEFORE I'D READ THE RUBRIC!

SG0 RUBRICS

Code for the SG0 program : 100 points total

1. Does the opening comment identify all the programming language and IDE used, the programmers, the date, and the class, an explanation of what the program does, a description of the central data structure(s), any external files used, and any external sources used in preparing the program? [15]
2. Does the code include appropriate internal comments? [10]
3. When the program begins, does it print to the screen an explanation that is no more than 1 screen, and tells concisely what the program does? [5]
4. Does the program skillfully lead the user to enter the filename for F, including appropriate error messages and reprompts when necessary? [20]
5. Does the program skillfully lead the user to enter one or more words? [10]
6. Does the program correctly implement the Yes/No question? [5]
7. Does the program correctly count the number of times a LegalWord entered by the user occurs in the ListOfWords? [20]
8. When the user has finished, does the program print out a nicely formatted list of each LegalWord and its count? [10]
9. Does the program finish correctly? [5]

Pseudo Code Design for SG0: 50 points

1. Is there a title page, and does it follow the specification? Are there page numbers? [5]
2. Is there a revision history? [5]
3. Is the design pseudo-code? [10]
4. Is the code at a high level (less than a page long)? [10]
5. Is the design consistent with the code and the specification? [20]

The Test Plan for SG0: 50 points

1. Is there a title page, and does it follow the specification? Are there page numbers? [5]
2. Is there a revision history and a table of contents? [5]
3. Does each test include explicit, detailed inputs (or actions) and the expected behavior? [20]
4. Have all the tests been executed at least once? Is each execution dated and attributed to a team member? [10]
5. Will the specified test each required task in the program at least once, and uncover common errors? [10]

There are 200 points (source code, 100 points; design, 50 points; and test plan, 50 points), for a total of 200 points for SG0 Each member of the group will receive the same team total.