Assignment 1 – ECS Opinion Survey

The Faculty of Engineering and Computer Science (ECS) at the University of Victoria (UVic) wants to know more about their students' satisfaction with their studies. In a pilot study, they invited SENG 265 students to create a software system to allow them to conduct a survey with students in their undergraduate programs.

Their idea is to do a pilot survey with adult students from some different fields so that, in the future, they can expand it with different surveys.

In the survey questionnaire, there are 12 statements. Each item is answered in a Likert scale with four levels of agreement to the statements: *disagree*, *partially disagree*, *partially agree* and *agree*. From each answer, researchers convert the answers into numbers in scale from 1 to 4. In doing so, they can easily compute both frequencies for each level and the average response for each question (a real number between 1 and 4).

Researchers asked the software developers to generate some statistics as output:

1. The relative frequency (percentage) of each level of agreement for each question;
2. The average response for each question.

Programming environment

For this assignment, please ensure your work executes correctly on the Linux machines in ELW B238. You are welcome to use your own laptops and desktops for much of your programming; if you do this, give yourself one or two days before the due date to iron out any bugs in the C programs you have uploaded to a lab workstation. (Bugs in this kind of programming tend to be platform specific, and something that works perfectly at home may end up crashing on a different hardware configuration.)

Git repo preparation

Configure your git info if you have not done it yet.

```
git config --global user.name "John Doe"

git config --global user.email johndoe@uvic.ca
```

Then, you will clone your **a1** repository – you will find it at https://gitlab.csc.uvic.ca/ .

You can browse your way until you find the **a1** project or you can go to the link:

https://gitlab.csc.uvic.ca/courses/2025011/SENG275_COSI/course_code/a1

There you can find the URL for cloning on ***Code > Clone with HTTPS***.

You can do this from the command line. Choose your project directory to start and recover the **a1** project directory there with **git clone**.

```
git clone
https://gitlab.csc.uvic.ca/courses/2025091/SENG265_COSI/course_code/a1.git
```

Notice that the repository you have just cloned is a read-only repo. You will first want to change the destination of your commits to a new student repo of yours on GitLab.

To do so, first, enter your working directory with **cd a1**

Rename the current remote repo to old-origin.

```
git remote rename origin old-origin
```

Then, add your new **a1** student repo as the remote origin. Replace <id> with your netlinkid.

```
git remote add origin
https://gitlab.csc.uvic.ca/courses/2025091/SENG265_COSI/assignments/<id>/a1.git
```

Finally, push all the contents to your new **a1** student repo.

```
git push –u origin --all
```

If it does not work (you may get a fatal error), you can also try:

```
git push --set-upstream origin main
```

Check that your push worked. On a web browser, open the following URL and then browse your **a1** student repo on GitLab. Replace <id> with your netlinkid.

```
https://gitlab.csc.uvic.ca/courses/2025091/SENG265_COSI/assignments/<id>
```

Now you can continue working with git the way you wish. You will write add, commit and push commands. All the next pushes can be done with a simple **git push**.

Committing practices

Commit your code frequently (**git add** and **git commit**), so you do not lose your work. Do not forget to **git push** into your remote repo from time to time.

We will look at the code commits you did in this assignment. **We require at least three different commits** (you should split your work into parts). The final grading will take that into account. Finally, when you finish Assignment 1, be sure to send the final version to the remote repo with a **git push** command.

Individual work

This assignment is to be completed by each individual student (i.e., no group work). Naturally you will want to discuss aspects of the problem with fellow students, and such discussion is encouraged. **However, sharing of code solutions is strictly forbidden without the express written permission of the course instructor (Roberto).** If you are still unsure regarding what is permitted or have other questions about what constitutes appropriate collaboration, please contact the course instructor as soon as possible. (Code-similarity analysis tools will be used to examine submitted work.) The URLs of significant code fragments you have found online and used in your solution must be cited in comments just before where such code has been used.

Description of the task

The survey questionnaire has the following questions/assertions.

1. My studies contributed to develop skills and knowledge for employment
2. My studies contributed to develop effective study and learning skills
3. My studies contributed to develop time management skills
4. My studies contributed to thinking logically and analytically
5. My studies contributed to dealing successfully with obstacles to achieve an objective
6. My studies contributed to develop knowledge of career options
7. My studies contributed to develop ability to find and use information
8. My studies contributed to develop self-confidence
9. My studies contributed to develop ability to interact with people from backgrounds different from my own
10. My studies contributed to develop ability to evaluate my own strengths and weaknesses
11. My studies contributed to thinking creatively to find ways to achieve an objective
12. My studies contributed to develop persistence with difficult tasks

Someone else was hired to produce a user interface for the survey. You just need to know that survey questions, respondents' demographic data and respondents' answers to the survey questions are recorded in a text file following a simple format. You should use this file to read all the input data to your program. In this text file, lines that start with **#** are comments and should be ignored by your program. The other lines have data whose description is in the comment lines.

You should send your program's output to the standard output (usually the computer screen or console). By using Unix pipes and redirection, one will be able to send your program's output to a file. You will use this strategy yourself to test your program.

You must also develop the system source code in C. Your program will be run from the Unix command line. Input is expected from **`stdin`**, and output is expected at **`stdout`**.

You must NOT provide filenames as input parameters to the program, nor hardcode input and output file names. Code that does either one or the other will receive grade zero.

You must NOT hardcode strings with the phrasing of the questions and participants' responses. Instead, you should read that information from the `stdin`. Code that hardcoded strings with that information will receive grade zero.

Implementing your program

The C program you will develop in this assignment:

- Starts by reading the input from **`stdin`** with the questions and options, and stores them appropriately;
- Continues reading from **`stdin`** with the data from a sample of respondents.
- Computes the requested statistics according to the user's request expressed in the input file;
- finally, writes the requested results into **`stdout`**.

Assuming your current directory **`a1`** contains your **`survey.c`** source file as well as the compiled executable file (named **`survey`**), and a **`tests`** subdirectory containing the assignment's test files is also in the **`a1`** directory, then the command to run your script will be.

```
% cat tests/in01.txt | ./survey
```

In the command above, you will pipe the input query text from the **`tests/in01.txt`** to the **`survey`** script and the output will appear on the console screen.

```
% cat tests/in01.txt | ./survey | diff tests/out01.txt -
```

The **`diff`** command above allows comparing two files and showing the differences between them. Use **`man diff`** to learn more about this command. The ending `hyphen/dash` informs **`diff`** that it must compare **`tests/out01.txt`** contents with the text output from your program piped into **`diff`**'s **`stdin`**. This is how you should test your code.

The **`tests`** directory comes with your repository to guide your implementation effort. Inside **`tests`**, there are four input files and four output files: the ones finishing with **`01`** are related to the simpler tests; the ones finishing with **`02`** are related to more complex tests and so on. A script with four tests is available there for your final checks before submission.

Start with simple cases. In general, lower-numbered tests are simpler than higher-numbered tests. Refrain from writing the program all at once, and budget time to anticipate for when "things go wrong".

You should commit your code whenever you finish some functional part of your it. This helps you keep track of your changes and return to previous snapshots in case you regret a change. When you are confident that your code is working correctly, push it to the remote repository.

Exercises for this assignment

You may develop your code the way that suits you best. Our suggestions here are more for facilitating your learning than as a requirement for your work. You may not need to do the exercises below if you want to practice deeper problem solving skills. But, in case you get stuck, you may look at them as a reference. On the other hand, if C programming seems difficult to you, you may use them as a script to learn and practice.

1. Write your program **`survey.c`** program in the **`a1`** directory within your git repository.
    a. Practice with **`stdin`** by reading it line by line with loops and **`fgets()`** and printing the output with **`printf()`** or **`fprintf()`**.
    b. Play with standard input redirection, by redirecting input to read from a file instead of reading from the keyboard.
    c. Learn how to tokenize a line (i.e., split a string line into a group of string tokens) stored in a string using the **`strtok()`** (you may wish to look at the example in the course slides or any good web tutorial that explains how to tokenize a sentence into words using the C language and the **`strtok()`** function).
    d. Learn how to loop over string tokens by printing them on the terminal screen.
    e. Learn how to store questions and answer options in an array of strings.
    f. Learn how to store answer options in an array of strings.
    g. Learn to use **`strcmp()`** or **`strncmp()`** to search for a particular string.
    h. Now search for any of the answer options stored in an array of strings, looping over it.
    i. Convert a respondent's answer using a string array of answer options. Use a function to encapsulate this operation.
    j. Play with the standard output, using **`printf()`** to print on your terminal screen a respondent's answers converted into a numerical scale.
    k. Read all the students' answers and store the absolute frequencies of each question-answer pair appropriately (using two-dimensional arrays may be a good idea).
    l. Compute the relative frequency of each answer (again, using two-dimensional arrays may be a good way of doing this).
    m. Print the relative frequencies (percentages) of each level of agreement for each question.
    n. Format the relative frequencies appropriately to converge with the format in the tests.
    o. Play with standard output redirection, by redirecting **`stdout`** to save data in a file instead of sending them to the console.
    p. Test your program using pipes and the **`diff`** tool with the first test file as explained in the previous section.
    q. You may realize that just printing the words with **`printf()`** to your terminal screen may lead to some disorganized output (spaces missing or in excess). Those issues are related to the limitations of the **`strtok()`** function (particularly when a line ends). You may bring organization to your output by checking the separators you used for splitting the tokens (you may add **`\n`** as a separator together with the standard separator).
    r. Compute the average (real value between 1 and 4) for each question and store it in a one-dimensional array.
    s. Print the average results for all the questions into **`stdout`**, using the format as in the tests.
    t. Play with standard output redirection, by redirecting **`stdout`** to save data in a file instead of sending them to the console.
    u. Test your program using pipes and the **`diff`** tool with the second test file as explained in the previous section.
    v. Have you thought about modularizing your work during the development? If not, now it would be a good time to separate parts of your code into different functions, in case you want an "A" grade.
2. Use the **`-std=c11`** flag when compiling to ensure your code meets the ISO/IEC 9899:2011 standard for the C language.
3. Keep all of your code in one file for this assignment. In later assignments we will use the separable compilation features of C.

4. Commit your code frequently (**`git add`** and **`git commit`**), so you do not lose your work. For instance, you may combine some of the exercises above or each partial solution to a separate test case into a commit, and describe it with a commit message.
5. From time to time, do not forget to **`git push`** them into your repo (you can also do this immediately after **`git add`** and **`git commit`**, if you prefer). Of course, our final grading of the code functionalities will not analyze any initial commits and pushes, just the final push.
6. Use the test files given in the starter repo to guide your implementation effort. Start with simple cases. In general, lower-numbered tests are simpler than higher-numbered tests. Refrain from writing the program all at once, and budget time to anticipate for when "things go wrong".

What you must submit

- You must submit 1 (one) single C source file named **`survey.c`**, within your **`git`** repository (and within its **`a1`** subdirectory) containing a solution to this assignment. Ensure your work is committed to your local repository and pushed to the remote before the due date/time. (You may keep extra files used during development within the repository, there is no problem doing that. But notice that the graders will only analyze your **`survey.c`** file). Do not forget to send your final commit to the remote repo with **`git push`**.
- On Brightspace, you must send your Netlink ID and your final **commit hash** until the due date.

Evaluation

Our grading scheme is relatively simple and is out of 100 points. Assignment 1 grading rubric is split into nine parts.

1) Modularization - 10 points - the code should have appropriate modularization, dividing the larger task into simpler tasks (and subtasks, if needed);
2) Documentation - 10 points - code comments (enough comments explaining the hardest parts (loops, for instance), no need to comment each line), function comments (explain function purpose, parameters and return value if existent), adequate indentation;
3) Version control - 10 points - Appropriate committing practices will be evaluated, i.e., your work cannot be pushed in just one single commit, evolution of your code must happen gradually (at least three commits are expected for this assignment);
4) Appropriate use of arrays to store data - 10 points - Your code must store the data read from input and intermediate processed data in adequate data structures such as one-dimensional and two-dimensional arrays.
5) Tests: Part 1 - 30 points - passing test 1 in **`tests`** directory: **`in01.txt`** as input and **`out01.txt`** as the expected output;
6) Tests: Part 2 - 20 points - passing test 2 in **`tests`** directory: **`in02.txt`** as input and **`out02.txt`** as the expected output;
7) Tests: Part 3 - 10 points - passing test 3 in **`tests`** directory: **`in03.txt`** as input and **`out03.txt`** as the expected output.

We will only assess your final submission sent up to the due date (previous submissions will be ignored). On the other hand, late submissions after the due date will not be assessed.