COMP 2401B – Final Project

Due: **Friday, December 11 at 11:59:00 pm**
Collaboration: **Collaboration is strictly disallowed.**

1. Goal

For this project, you will design and implement a program in C, in the Ubuntu Linux environment, that generates a set of reports based on data from the Organisation for Economic Co-operation and Development (OECD). Your code will be correctly separated into modular and reusable functions, and it will follow the principles of correct software design and development.

The grading of this project will be rubric-based. The **mandatory minimum criteria** for grading is that all the requirements must be implemented, they must meet all the listed constraints, they must execute correctly, the corresponding data must be printed to the screen, and you must participate in a scheduled demonstration of your project code. Submissions that are incomplete, or that don't work correctly, or that don't meet all the constraints, or that don't print the resulting report statistics to the screen, will earn a grade of zero.

2. Data Set and Reports

You will use the OECD data provided for you in the `grad.dat` file posted in *cuLearn*. This file contains data on the numbers of graduates from post-secondary institutions all over the world, who graduated with a degree in a technology field, including Computer Science.

Each record (i.e. each line) in the data file gives us statistics for a given country, year, degree, and gender (including the total numbers for all genders). That record tells us the number post-secondary graduates in technology, of that gender, who graduated in that country, during that year, with that degree.

The records are formatted as follows: `<countryCode countryName gender degree year numGrads>` where `countryCode` indicates the country name abbreviation where graduates studied; `countryName` indicates the corresponding long-form country name; `gender` tells us whether the numbers in that record are for all genders ("T" for "Total"), or males only ("M"), or females only ("F"); `degree` specifies a post-secondary degree, where "L6" is a Bachelor's degree, "L7" is a Master's, and "L8" is a Doctorate; `year` indicates a graduation year; and `numGrads` tells us the total number of post-secondary graduates in technology, of that gender, who graduated in that country, with that degree, during that year.

For example, the first record reads: `AUS Australia F L6 2010 1276`
and a later record tells us: `AUS Australia T L6 2010 7055`

The first record tells us that, in the year 2010, there were 1276 students who self-identified as female, who graduated with a Bachelor's degree in a technology field in Australia. The second record tells us that in the same year in Australia, a total of 7055 graduates (including all genders) graduated with a Bachelor's degree in those same fields.

Your program will use the OECD data provided in order to generate a total of five (5) reports, upon request by the user, including the following three (3) reports:

2.1. the graduate percentage for each country, by degree, for all years and all genders
- 2.1.1. each country will be a row, and there will be a column for each type of degree, plus one column for all degrees combined
- 2.1.2. each cell will show the proportion of graduates in that specific country, for that specific degree, compared to the total graduates in all countries combined, for that degree

2.2. the top 5 and bottom 5 countries for percentage of female graduates, for all years and all degrees
- 2.2.1. each country will be a row, and there will be one percentage column
- 2.2.2. each cell in the percentage column will show the proportion of female graduates compared to the total number of graduates in that specific country, for all years and all degrees combined
- 2.2.3. only the top 5 and bottom 5 countries in terms of percentages will be shown in the report

2.3. the graduate percentage for each country, by year from 2014 to 2017, for all degrees and all genders
   2.3.1. each country will be a row, and there will be a column for each year from 2014 to 2017, inclusively
   2.3.2. each cell will show the proportion of graduates in that specific country, for that specific year, compared to the total graduates in all countries combined, for that same year

3. Program Requirements

You will implement a program that reads in the graduate data from the `grad.dat` file posted in *cuLearn*, and you will present the user with a menu of possible reports. When the user selects a report, your program will compute the statistics required for the report using the graduate data, and it will print the resulting information to the screen.

Your program will implement the following requirements:

3.1. Report requirements

Your program will allow the user to execute the following reports:

3.1.1. the three (3) reports that are described in section 2, and

3.1.2. two (2) additional reports of your own creation

The reports of your own creation must meet all of the following requirements:

3.1.3. they must represent a substantial amount of new logic and new code

3.1.4. they must represent a different way of thinking from the existing reports described in section 2

3.1.5. they cannot be a simple variation of the existing reports
   (a) for example, the first report in section 2 produces the graduation percentages by country, by degree; it would **not** be acceptable to simply do the same for male graduates or female graduates only, since that would be the same logic as an existing report

3.1.6. they cannot be a simple inversion of the rows and columns of an existing report

3.1.7. your new reports must be thoroughly described in your README file; if your new reports are not explained there, the corresponding code will not be graded

3.2. Design requirements

3.2.1. You will design the structure data types that you need to represent the information of this program.

3.2.2. The program will be separated into modular, reusable functions that could be used in an extended version of this program, or potentially in other programs.

3.2.3. Your functions must be single-purpose and reusable, and they must have a clear interface with parameters that are identified as either input, output, or input-output parameters.

3.2.4. The graduate data must be stored in structures that you design, and all instances of these structures must be dynamically allocated. All dynamically allocated memory must be explicitly deallocated when no longer in use.

3.2.5. The initialization of structure instances of the graduate data must occur in separate functions that perform the dynamic memory allocation. Double pointers must be used for this purpose.

3.2.6. The graduate data must be stored in a linked list, as we saw in the course material, section 3.3.

3.3. Overall control flow

3.3.1. A menu of available reports will be presented to the end user. The user will select an option, and the program will compute the statistics for the corresponding report, and print them to the screen.

3.3.2. Once the report statistics have been printed to the screen, the menu will be displayed again, until the user chooses to exit.

3.4. Packaging

3.4.1. Your program must be separated into at least one (1) header file and at least four (4) source files. Each source file must contain functions that are functionally related to each other.

4. Constraints

Your program must comply with all the rules of correct software design and development that we have learned during the lectures, including but not restricted to:

4.1. The code must be written using the default C standard that is supported by the course VM, and it must compile and execute in that environment. It must not require the installation of libraries or packages or any software not already provided in the VM.

4.2. Your program must be correctly designed, and it must be separated into modular, reusable functions.

4.3. Do not use any global variables.

4.4. Your program must reuse the functions that you implemented, everywhere possible.

4.5. Your program must perform all basic error checking.

4.6. Compound data types must always be passed by reference, never by value.

4.7. Return values will be used only to indicate function status (success or failure). Except where permitted in the instructions, data must always be returned using parameters, and never using the return value.

4.8. All dynamically allocated memory must be explicitly deallocated.

4.9. All functions must be **fully documented**, with a block of comments before each function, documenting its purpose and each of its parameters, as indicated in the course material, section 1.2. **DO NOT** include comments in the body of the functions.

5. Submission

5.1. You will submit in *cuLearn*, before the due date and time, the following:

5.1.1. One `tar` or `zip` file that includes:
- (a) all source and header files
- (b) a Makefile
- (c) a README file that includes:
  - (i) a preamble (program author, purpose, list of source and header files)
  - (ii) compilation and launching instructions
  - (iii) a description of the new reports created for instruction 3.1

NOTE: Do not include object files, executables, hidden files, or duplicate files or directories in your submission.

5.2. Late submissions will be subject to the late penalty described in the course outline. No exceptions will be made, including for technical and/or connectivity issues. Do not wait until the last day to submit.

5.3. Only files uploaded into *cuLearn* will be graded. Submissions that contain incorrect, corrupt, or missing files will receive a grade of zero. Corrections to submissions will not be accepted after the due date and time, for any reason.

6. Grading

The grading of this project will be rubric-based, and the total grade will be out of 60 marks.

6.1. **Minimum criteria**: In order to be graded, your program must meet the following minimum criteria:

6.1.1. The program must be implemented in C, and it must compile and execute in the default course VM.

6.1.2. A correct Makefile must be provided.

6.1.3. The program must successfully complete at least three (3) executions that meet all requirements and constraints described in sections 3 and 4.

6.1.4. To be graded, the code must be invoked, and the corresponding data must be printed to the screen.

6.1.5. You must participate in a scheduled demo of your project code.

Submissions that do not meet this minimum criteria will not be graded and will earn a grade of zero.

6.2. Rubric-based grading

The project will be assessed in accordance with the grading categories listed in section 6.3. Each category will be graded based on a 6-point rubric. Some categories are worth double or triple the weight of other categories, so their value is 12 points or 18 points each. These will still be assessed according to the 6-point rubric, but the values will be doubled or tripled accordingly.

The 6-point rubric is defined as follows:

- [6 points] Excellent (100%)
- [5 points] Good (83%)
- [4 points] Satisfactory (67%)
- [3 points] Adequate (50%)
- [2 points] Inadequate (33%)
- [1 points] Poor (17%)
- [0 points] Missing (0%)

6.3. Grading categories

The project will earn an overall grade out of 60 marks. This will be the sum of the grades earned for each of the following categories, based on the 6-point rubric described in section 6.2:

6.3.1. **Function design [18 marks]:** This criteria evaluates your function design. This includes, but is not limited to, how well your structure data types are designed, how effectively the functionality of your implementation is separated into modular and reusable functions, how well your design is documented, and how well it follows correct design principles.

6.3.2. **Code quality [12 marks]:** This criteria evaluates the overall code quality. This includes, but is not limited to, how well your functions are implemented, how well the code is written, how well the code adheres to standard programming conventions, and how it complies with all constraints listed in section 4.

6.3.3. **Report generation [12 marks]:** This criteria evaluates the effective development and usage of the report generation features, including the usage of data, the correctness of the results, and the layout and clarity of the output.

6.3.4. **Innovation [6 marks]:** This criteria evaluates the innovative reports that you add to your program. The new reports will be evaluated based on how much additional code is implemented, and the creativity and the programming sophistication shown in selecting and implementing these reports. New reports that are not described in the README file will not be graded.

6.3.5. **Packaging [6 marks]:** This criteria evaluates the packaging of the code into a professional software submission. This includes, but is not limited to, the correct separation of the code into header and source files, the presence of complete and correctly formatted Makefile and README files, and the absence of duplicate, additional, and/or unneeded files.

6.3.6. **Demo [6 marks]:** This criteria evaluates the demonstration that you give of your project code. This includes, but is not limited to, how well you show the functionality of your code, and the depth and breadth of your answers to the questions you are asked.