# CIS*2750
# Assignment 1 grading instructions

Important notes:
- Make sure you compile and run all code on linux.socs.uoguelph.ca!
- Please review the rubric and apply the deductions consistently (warnings, leaks, etc.).
- A perfect assignment would get 90 as the test score, 10 for the makefile (4 targets, 2.5% each) and will have no deductions
- Please **do not** hit "Publish" when saving the feedback - use "Save Draft" instead. I need to review the grades before I release them.

## 1 Grading rubric

| | | |
|---|---|---|
| Functions (graded using an automated test harness): | | 90% |
| • `createGEDCOM()`: | 50% | |
| • Correct GEDCOMobject creation from different valid files: | 35% | |
| • Correct handling of invalid GEDCOM files: | 15% | |
| • `printGEDCOM()`: | 5% | |
| • `deleteGEDCOM()`: | 10% | |
| • `printError()`: | 5% | |
| • `findPerson()`: | 5% | |
| • `getDescendants()`: | 15% | |

Correct makefile (has all four rules, creates correctly named files): 10%
- `make list` creates a static library `liblist.a` in `assign1/bin`
- `make parser` creates a static library `libparser.a` in `assign1/bin`
- `make` or `make all` creates `liblist.a`,and `libparser.a` in `assign1/bin`
- `make clean` removes all `.o` and `.a` files

The submission must have the following directory structure:
- `assign1/` contains the `README` file and the `Makefile`
- `assign1/bin` should be empty, but this is where the `Makefile` will place the static lib files
- `assign1/src` contains `GEDCOMparser.c`, `LinkedListAPI.c`, and `GEDCOMutilities.c` (if you need it)
- `assign1/include` - contains `GEDCOMparser.h`, `LinkedListAPI.h`, and any additional header files that you might have.

**Deductions:**

You will lose mark for run-time errors ad incorrect functionality. Additional deductions include:

| | |
|---|---|
| • Any compiler warnings: | -15% |
| • Any memory leaks: | -15% |
| • Incorrect directory structure: | -5% |
| • Incorrect output filenames created by makefile: | -5% |
| • Any additional failures to follow submission instructions: | -5% |
| • Failed list tests: | up to -10% |

**Any compiler errors: automatic grade of zero (0) on the assignment.**

## 2. Test harness instructions

*Running the main test harness*

The harness tests calendar creation/deletion, the two print... functions, as well as findIndividual and get Descendants. It also tests error handling. It does not test for memory leaks - those scripts are separate (see below). As a result, GEDCOM deletion tests only really test for segfaults and other crashes during deletion.

We don't want to rely on potentially broken makefiles/static libs, so the test harness only uses student .h and .c files.

The test harness directory structure is:
- `bin` - will contain all executable files
- `src` - contains test cases. Do not modify these in any way.
- `include` - contains test harness headers and `GEDCOMparser.h`. Do not modify these in any way.
- `studentCode` - all student .c files go here.
- `studentInclude` - all student .h files go here. Do not use the student version of `GEDCOMparser.h`!
- `testFiles` - contains various broken and valid GEDCOM files

For every student, copy their .c and .h files into the appropriate directories, and run the harness.

To run:
- compile by typing `make`
- run `bin/GEDCOMtests`

If the code does not compile, student gets a 0 for the assignment, and you do not have to do anything else. However, make sure you paste the exact compiler error in the feedback, and include a note saying that the student got a 0 due to compiler errors.

The output of the test harness contains all the information about the passed/failed tests, as well as the total score (out of 90). Paste all of this information into the "feedback" portion.

*Checking for memory leaks*

- Compile ty typing `make memTest`
- run `make valgrind`

This will execute valgrind 4 times (with 4 different .ged files). Each test must show
`" in use at exit: 0 bytes in 0 blocks".`

If there are more than 0 bytes at exit - i.e. if there are leaks - you will see something like:

```
LEAK SUMMARY:
==31556==    definitely lost: 112 bytes in 1 blocks
==31556==    indirectly lost: 20,929 bytes in 681 blocks
==31556==      possibly lost: 0 bytes in 0 blocks
==31556==    still reachable: 0 bytes in 0 blocks
==31556==         suppressed: 0 bytes in 0 blocks
```

In this case, deduct 15% from the overall grade, add a note saying that they lost marks for memory leaks, and include the leak summary. If there are multiple leaks, include just one summary. The deduction is flat 15%, regardless of the number of leaks.

The student code will probably be buggy, so the valgrind output might be messy. Ignore all the errors - just look for memory leak summaries.

If one of the valgrind cases crashes, ignore the leaks for that case - crashed code will leak by its nature. However, make sure you look at the output of the other cases. If all 4 test cases crash, do not assign any penalties for the leaks.

*Marking the makefile*
Run the student makefile from their main directory. Use make to create each of the 4 targets described in the grading rubric. Students get 2.5% for every command that executes successfully.