Date due: Tuesday, March 15, 11:59:59 p.m.

1 Introduction and purpose

You will not write any C code in this project. Instead you will just write a makefile to compile some code that you are being given. This is a small project, which has only public tests and will not be graded for style. So there is just a short time to do it, in order to leave more time for other projects that are larger and more difficult.

We are supplying you with some small C functions in several source files (with associated header files), along with three different programs that use the functions. The programs call the functions and just print two numbers each. What you have to do is to look at the source files for the three programs, figure out the commands that would be needed to compile them and link the resuting object files to create the three executables from them, figure out which files each compiler–generated file depends on, and write a makefile that will do the compilation and linking.

Before starting to write anything, study the lecture example makefiles. If you aren't sure about anything that they're doing, ask in the TAs' office hours **in advance**.

2 Supplied code

The project tarfile `project05.tgz` contains the following files:

`add-elements.c`, `increment.c`, `init.c`, and `test-equal.c`: These source files contain simple utility functions that perform some operations on arrays of integers.

`add-elements.h`, `increment.h`, `init.h`, and `test-equal.h`: These are header files with the prototypes for the functions that are defined in the source files above.

`application.c`, `main.c`, and `tester.c`: These are the main programs that call the functions defined in the files mentioned above (the programs call different functions).

All the code mentioned above does exactly what we want. Do not change it in the slightest. Look at it only to figure out how to compile and link it to create the three executable programs.

3 `Makefile` to be written

- In the `project05` directory you must write a makefile, named `Makefile` (its name **must** begin with an uppercase `'M'`), which will build three programs `application.x`, `main.x`, and `tester.x` (so it must have these three targets) from the three files `application.c`, `main.c`, and `tester.c` and the other source files. You can tell which object files have to be linked together to form each executable by looking at what functions the main programs are calling. And you should be able to see what files have to depend on what other files in by looking at the source files.
- Your `Makefile` **must** also have a target `all`, which will build all three executable programs `application.x`, `main.x`, and `tester.x`. It should be the first target, so either running just `make`, or `make all`, will build all three programs.

  Note that an example makefile from lecture illustrated the correct way that an `all` target should be written.
- Your `Makefile` **must** use the five `gcc` compilation options that we are using this semester (which are `-ansi`, `-pedantic-errors`, `-Wall`, `-fstack-protector-all`, and lastly `-Werror`), which were mentioned in the first project, to build object files. (But it should **not** use these options when linking object files to form executable programs.)

  Your `Makefile` should not explicitly repeat all five options in every invocation of `gcc`; they should be added to compilation commands via the makefile macro `CFLAGS`, as long as the macro is added to every command that is compiling a source file. (When you compile code on Grace by typing `gcc` in a shell, these options are enforced via an alias for `gcc` that is added as a result of the account setup steps you made early in the semester. However, a makefile does not recognize shell aliases. This is why the options must be explicitly used in a makefile.) Your makefile should also use the macro `CC` for the name of the compiler to use.

- Your makefile **must** use separate compilation– for each of the three executable programs, each source file (e.g., `add-elements.c`) needed to form that program must be separately compiled, then the resulting object files must be linked together to form the executable. (So your `Makefile` must also have additional targets for all object files. Note that there are seven source files, so seven object files should be created by your makefile.)
- Your `Makefile` should **not** explicitly compile header files (i.e., do **not** pass them to `gcc` as arguments).
- Your `Makefile` must also have a target named `clean` that will not create any files but will instead delete all of the compiler–generated files (the object and executable files) that are created by the compilation commands in your makefile.
- Your makefile should have all needed dependencies, otherwise programs may not get built correctly (if some dependencies are missing it would not perform some necessary compilations). But it should not have any unnecessary dependencies, because that would lead to it performing compilations that are not needed.

A Development procedure review

A.1 Obtaining the project files, compiling, checking your results, and submitting

Log into the Grace machines and use commands similar to those from before:

```
cd ~/216
tar -zxvf ~/216public/project05/project05.tgz
```

This will create a directory `project05` that contains the necessary files for the project. You **must** have your coursework in your course disk space for this class. Your `Makefile` **must** be in the `project05` directory.

There are eleven public tests for this project, and no secret tests. These tests are written in languages most students don't know, and you don't need to understand the code for these tests in order to write your `Makefile`. So in order to minimize confusion, we haven't given you the test programs, so you cannot run the tests yourself on Grace. (There are also no output files for the tests.) But instead we explain here what the tests are testing, and below we explain how to test (**before** submitting) whether your `Makefile` passes the tests.

1. The first test tests whether your `Makefile` correctly builds `main.x`.
2. The second test tests whether your `Makefile` correctly builds `application.x`.
3. The third test tests whether your `Makefile` correctly builds `tester.x`.
4. The fourth test tests whether your `Makefile`'s `all` target works, is written correctly, and is in the correct place in your `Makefile`.
5. The fifth test tests whether your `Makefile` is using separate compilation everywhere.
6. The sixth test tests whether your `Makefile` is explicitly compiling header files, which should not be done.
7. The seventh test tests whether your `Makefile`'s `clean` target removes all compiler-generated (executable and object) files.
8. The eighth test tests whether your `Makefile` is using the macros `CC` and `CFLAGS`, instead of repeating the compiler name and the compilation options in every rule, and whether it is using the exact names `CC` and `CFLAGS`. (This test also expects that you are using the correct compilation options.)
9. The ninth test tests whether your `Makefile` has any missing dependencies.
10. The tenth test tests whether your `Makefile` has any extra unnecessary dependencies.
11. The eleventh test tests whether your `Makefile` is using the required compilation options for creating all object files.

The tests do not test for some "stylistic" properties of makefiles. In later projects where you write makefiles, these stylistic properties may be checked manually. (If you have questions about whether your makefile is stylistically correct you can ask in the TAs' office hours.) Note that even if your makefile correctly builds the three programs, it might still fail tests other than the first three on the submit server. Even if your makefile is missing some rules entirely it could pass the first three tests but still fail later ones.

A.2 Testing your `Makefile`, and submitting

Before you submit you **must** test your `Makefile` yourself. (If it doesn't compile the programs on Grace it won't compile them on the submit server.) The first thing to test is whether the three separate commands `make main.x`, `make application.x`, and `make tester.x` create the three programs. Besides seeing the three programs in the directory, you should also see seven object files, one created for each source file.

The next thing to check is that your `clean` target removes all executable and object files, but nothing else. **First make a copy of your entire `project05` directory** using `cp -r`, then run `make clean` in the `project05` directory. You should still see all seven source and all four header files– none should have been removed– but you should not see any object or executable files. (The purpose of making a copy of the directory first is so you will have a backup in case anything is incorrectly removed by your `clean` target. For example, if you are in the main `~/216` directory, you can use `cp -r project05 project05-copy` or a similar command.)

After running `make clean`, run the three commands `make main.x`, `make application.x`, and `make tester.x` again, to ensure that your makefile will build the programs correctly when there are no executable or object files already in the directory. This is the situation in which your makefile has to work on the submit server– being able to build the programs when there are no executable or object files in the directory– so test it first and ensure it works right in the same circumstances. (If there are already some executable or object files in the directory when you try to build the programs your makefile may seem to be working correctly, even if it is missing dependencies or has other problems.) Then run `make clean` again and then `make all` and ensure that everything is also built correctly in that case as well. Then run `make clean` and `make` and you should get the exact same results.

Running `submit` from the `project05` will submit your makefile. After submitting you **must** then log into the submit server and verify your makefile worked correctly there. If your makefile fails some tests on the submit server, carefully identify what commands you would have to run manually for compiling each source file to an object file, and for linking the necessary object files to form each executable. Then run `make clean`, and run `make -n` on each target (object and executable file) in your makefile one at a time, and see if the commands that `make` says it would perform are the same ones that you just identified that you would have to run by hand to create that file. If not, something is wrong with your makefile.

If you are failing either the ninth or tenth test (missing dependencies or extra unnecessary dependencies) then for each source and header file: (a) carefully determine which object and executable files would have to be recreated if that file was changed; (b) then create all of the executable and object files so they're up to date; (c) then use `touch` on that source or header file to simulate it changing; (d) then run `make -n` to see whether your makefile will cause `make` to perform the right commands to create the object and executable files that you determined should be recreated when that file changes. You can ask for help in the TAs' office hours if needed, **after** you have already tried doing this, and can show the TAs your results. The TAs will **not** help you unless you have first at least tried testing your `Makefile` yourself.

If you fail other tests on the submit server and you're sure your makefile is correctly building the three executable programs, other things to check for are that your makefile has the right name (it must be named `Makefile`), that it is using the required `gcc` compilation options, that it is using separate compilation, that it is not explicitly compiling header files, and that its clean target is removing all object and executable files.

A.3 Grading criteria

Your grade for this project will be based on:

public tests	100 points

Since your Makefile is not being graded for style, and it is not C code, you are not required to write comments in it; of course you may if you like.

B Project–specific requirements, suggestions, and other notes

- Do **not** change any of the source or header files supplied in the project tarfile, or add any source or header files. Your `Makefile` will be tested on the submit server using our versions of these files.
- There are many features of make that were not explained in class, because they are difficult for a beginner to use correctly. ***Your* `Makefile` *is only allowed to use the features of make that were covered in class.*** If you use features not explained in class: (a) your submission may not compile on the submit server for technical reasons not worth

explaining here, (b) the TAs will not be able to help you fix your makefile in office hours, and (c) we will deduct significant credit.

- For this project you will **lose one point** from your final project score for every submission that you make in excess of ten submissions. (Since you are not writing any C code there won't be any such thing as a noncompiling submission.)

C Academic integrity

Please **carefully read** the academic honesty section of the syllabus. **Any evidence** of impermissible cooperation on projects, use of disallowed materials or resources, publicly providing others access to your project code online, or unauthorized use of computer accounts, **will be submitted** to the Office of Student Conduct, which could result in an XF for the course, or suspension or expulsion from the University. Be sure you understand what you are and what you are not permitted to do in regards to academic integrity when it comes to projects. These policies apply to all students, and the Student Honor Council does not consider lack of knowledge of the policies to be a defense for violating them. More information is in the course syllabus – please review it now.

The academic integrity requirements also apply to any test data for projects, which must be **your own original work**. Exchanging test data or working together to write test cases is also prohibited.