Name: _____

CS 0449 – Multifile Development Lab

Background

Make is a utility that automatically builds executable programs and libraries from source code by reading files called makefiles which specify how to derive the target program. Besides building programs, Make can be used to manage any project where some files must be updated automatically from others whenever the others change.

In this lab, you will learn how to divide a single monolithic source file into multiple source and header files, and write a makefile for the project. Refer to your lecture notes and the GNU Makefile tutorial: http://www.gnu.org/software/make/manual/make.html#Introduction.

Part 1: Modularizing Source Code / Applying Scoping

1. Login via SSH to thoth.cs.pitt.edu

2. Create a new directory for this lab under your work directory and change into it.

3. First try compiling and running the monolithic file mallocdrv.c using the below directions. Incidentally, this is the malloc driver given to you to test project 3.

```
cp ~wahn/public/cs449/mallocdrv.c ./
gcc –o mallocdrv ./mallocdrv.c
```

4. Now, divide mallocdrv.c into 4 logically modular files: tree.c, tree.h, malloc.h, and main.c. The files should contain the following declarations or definitions.

   main.c: the main() function and the test functions (test1, test2, comp)
   tree.c: the definitions for the tree manipulation functions (freetree, randominsert, printtree)
   tree.h: the declarations for the functions and struct type needed for main.c to use tree.c
   malloc.h: the declarations for the malloc macros MALLOC, FREE, and DUMP_HEAP()

   Insert #include "…" directives as necessary. Try not to have duplicate declarations anywhere. The purpose of headers is to prevent duplication. After you are done, you will notice that headers also provide the benefit of summarizing the interface of a C code (the functions and variables that can be accessed externally and the type declarations for them). So a programmer need only to look at tree.h to figure out what tree.c is able to do. In spirit, it is similar to the

concept of how a Java interface exposes only the parts of an implementation class that are visible to the outside world.

5. Follow below directions to compile and link the multiple files, then run program. **Show the TA.**

```
gcc –o mallocdrv ./tree.c ./main.c
./mallocdrv
```

Part 2: Writing and Using a Makefile

6. Write a makefile named 'Makefile' in your working directory using your favorite editor. The makefile should produce intermediate .o object files for each of the .c source files and produce the executable mallocdrv when typing 'make'. Remember, the first target in the makefile is the default target and will be what is built when no target is given to 'make'. Also, insert an additional target 'clean:' in the makefile that will remove mallocdrv and all intermediate files (the .o files) when typing 'make clean'. All the rules should have correct dependencies in terms of source files, header files, or object files.

7. Test your makefile by building mallocdrv and then cleaning. **Show the TA.**

```
make
make clean
```

8. At below, draw the dependency graph for the makefile you created. Refer to the lecture notes.

What to Hand In

Hand in this worksheet to the TA and submit your project files.

```
tar -zcvf USERNAME_lab_makefile.tar.gz <lab directory name>
cp USERNAME_lab_makefile.tar.gz ~wahn/submit/449/RECITATION_CLASS_NUMBER
```

Ask any questions about writing makefiles for project 3 to the TA.