Faculty of Computer Science, Dalhousie University *22-Nov-2018*

CSCI 2132 — Software Development

Lab 9: Assignments and Make Utility

Location: Teaching Labs   Instructor: Vlado Keselj
Time: Thursday

Lab 9: Assignments and Make Utility

Lab Overview

- If you still need to work on Assignment 6, it should be your highest priority to finish it
- Assignment 7 is out, as well as Practicum 5, and you can work on them
- One of the questions in Assignment 7 is finishing material in this lab, which you can find further, and you can work on it as well.

Step 1: GitLab Project and SSH Setup. You should have created you GitLab account by now, as a part of Lab 7. If you did not do it, you should go to the instructions at the end of Lab 7 and do it now.

Open your Web browser and check you GitLab account. Remember that URL for the GitLab account is:
`https://git.cs.dal.ca`
You can login using your CSID and take a look at your `csci2132` project.

Login to your bluenose account as before. Rather than going to SVN, this time, we are going to use your GitLab repository to save and submit work on this lab. On your bluenose accout, make the directory `~/csci2132/git` and go to that directory. You can get a working copy of your GitLab project there using the following command:
`git clone https://git.cs.dal.ca/`*CSID*`/csci2132.git`
where *CSID* is your CSID. Try this command.

If you are asked the question about "The authenticity of host git.cs.dal.ca", answer 'yes'.

You will be asked to enter your CSID and password.

Go to the directory `csci2132` and make directory `lab9`. Go to the directory `lab9` and copy the file for this lab using the command:

```
cp ~prof2132/public/lab-make/* .
```

If you type 'ls' you should see the following files in your directory:

```
bit.h  dec2bin.c  hello.c  Makefile  sortlist.c  stack.c  stack.h
```

Now we are going to commit these files to the GitLab repository. Go back to your parent directory with:

```
cd ..
```

stage the directory 'lab9' for git using the command:

```
git add lab9
```

commit the directory locally using the command:

```
git commit -m'Initial commit of lab9
```

and 'push' the change to the GitLab repository using:

```
git push origin master:master
```

You will be asked to enter your CSID and password. After this you should be able to see your changes in the GitLab repository if you refresh the page.

Step 2: Running make commands. Execute the following sequence of commands to familiarize yourself with the `make` utility. These commands were discussed in class. After you enter each command, pay attention to the commands that the `make` utility used to create the targets, and run the target executable files created. Note that there may also be error messages when running `make clean`; think about why.

1. `make`
2. `make clean`
3. `make all`
4. `make clean`
5. `make dec2bin`
6. `make clean`
7. `make hello`
8. `make clean`

Step 3: Using make variables. In class, we learned that in order to use `gdb` to debug the executable file generated from multiple source files, we have to use `-g` as an option of all the `gcc` commands in the Makefile. It is however tedious to edit the makefile to change all the `gcc` commands, especially if your program contains a large number of modules.

To be more productive, we can define a variable and use it as the option for `gcc` commands in the Makefile. More precisely, edit the `Makefile` using `emacs` so that its content becomes:

```
# Makefile for the project dec2bin
FLAGS=-g -std=c99

.PHONY: help
help:
	@echo 'make all            will produce all'
	@echo 'make dec2bin will produce only dec2bin'
	@echo 'make clean          will clean'

all: dec2bin hello

dec2bin: dec2bin.o stack.o
	gcc $(FLAGS) -o dec2bin dec2bin.o stack.o

dec2bin.o: dec2bin.c stack.h bit.h
	gcc $(FLAGS) -c dec2bin.c

stack.o: stack.c stack.h bit.h
	gcc $(FLAGS) -c stack.c

hello: hello.c
	gcc $(FLAGS) -o hello hello.c

clean:
	rm dec2bin dec2bin.o stack.o hello
```

Then, make sure that there are no object or executable files (if there are, enter `make clean`). Let us learn one option (among many) of the command `make`. Enter:
`make -n dec2bin`
The make command will show what it would do to make the file `dec2bin`, however if you use `ls` to check the

contents of the directory, you will see that the target file was not created. This option is useful when we want to check what certain 'make target' command would do. Now, run the make for real:
`make dec2bin`

Step 4: Using `gdb`. Look at the lecture slides to remind your self about using `gdb` on a multi-source-file program, and what you would need to do to use the `gdb` command to set a breakpoint at line 12 of the source file `dec2bin.c`.

Open the executable file `dec2bin` using `gdb`, set a breakpoint at this line, and print the value of the variable `decimal` when the program pauses at this breakpoint.

Step 5: Other variations of the `FLAGS` variable. We can use other values for the variable `FLAGS` in the `Makefile`. Try the following two values: `FLAGS=-std=c99` and `FLAGS=-O3 -std=c99` (make sure that you use letter 'O' and not digit '0' when typing `-O3`). When you are changing the value, it is useful to comment out the old value so that in future you can quickly choose option that you need. For example, you can use:

```
# FLAGS=-g -std=c99
FLAGS=-std=c99
# FLAGS=-O3 -std=c99
```

Each time after you change the value of `FLAGS`, save and exit `emacs`, run `make clean` and then `make all`. Observe what commands `make` invokes to compile your program. Observe the size of the executable of the program.

Step 6: Program `sortlist.c`. You can find the program `sortlist.c` in your lab directory, which we covered in class. Your task is to break into program into several files (modules) and update the `Makefile` to be able to compile the program. Pieces of the program `sortlist.c` should be copied into the following files:

– `sortlist.h` — header file including standard NAME_LEN macro definition, `struct node` definition, and all function prototypes. Do not forget to protect the file from double-inclusion.
– `main.c` — the C file containing the main function,
– `mergesort.c` — the C file containing two merge sort functions,
– `list.c` — the C file containing the linked list functions, and
– `aux.c` — the C file containing the auxiliary functions (clearly marked in the `sortlist.c` program).

You must also update the `Makefile` so that it includes the target `listdb`, which is the executable program build from these files. Each C file must first compile into the appropriate `.o` (object) file, and the object files are used to build the final `listdb` file (similarly to the `dec2bin` example used in class).

Step 7: Test cases. Create two files `testlist1.in` and `testlist1.out` which can be used for testing the program `listdb`. Namely, you need to put content into the file `testlist1.in` such that when you run the command:
`./listdb < testlist1.in`
the following operations are executed on the program:

1. Five students are added to the list, in a random non-sorted way,
2. Display all students on the list,
3. Sort students by the number,
4. Display again all students on the list,
5. Delete one student from the list,
6. Display all students on the list, and
7. Exit the program.

Once you check that the test input file works correctly, create the corresponding output file `testlist1.out`. You can use the following command to create it:

```
./listdb < testlist1.in > testlist1.out
```

Update the `Makefile` with the new target `testlist` so that when we execute the command:

```
make testlist
```

the following commands are executed:

```
./listdb < testlist1.in > testlist1.new
diff testlist1.new testlist1.out
```

GitLab submission: Add all the files in the 'lab9' directory to git, commit it, and push it to the GitLab repository. Check in the browser that they are there.

You have finished the lab.