Faculty of Computer Science, Dalhousie University *25-Oct-2018*

CSCI 2132 — Software Development

Lab 7: Practicum 2 and Simple Makefile

Location: Teaching Labs Instructor: Vlado Keselj
Time: Thursday

Lab 7: Practicum 2 and Simple Makefile

Lab Overview

- You can start working on problems for Practicum 2, and finish the rest of the lab after, or
- You can first work on the lab and later submit problems to the Practicum
- If you want a good time on the practicum questions, then it may be a good idea to do practicum questions first

---

Step 1: Login and Practicum 2. You can login and work on the Practicum 2 first if you want. You should login to bluenose and go to the directory '`~/csci2132/svn/`*CSID*`/`' where *CSID* is your CSID. You should create the directory `a4` using the command:

```
svn mkdir a4
```

and in this directory you can work on your practicum files `C.c` and `D.c`. Once you develop the programs you can submit them to the practicum web site, which can be found under the link 'P2' on the course web site, or directly at: `https://web.cs.dal.ca/~dpc/csci2132-p2`

You can also submit the programs directly from bluenose using the command:

```
~dpc/public/dpc-submit csci2132-p2 C.c
```

or

```
~dpc/public/dpc-submit csci2132-p2 D.c
```

Remember that you will need to submit the programs `C.c` and `D.c` in the SVN directory *CSID*`/a4` and also to the practicum web site. To upload the problem to the practicum web site, you should first copy it to your local computer using a command that may look like this on Mac or Linux:

```
scp CSID@bluenose.cs.dal.ca:csci2132/svn/CSID/a4/C.c .
```

or like this on Windows if you use PuTTY:

```
pscp CSID@bluenose.cs.dal.ca:csci2132/svn/CSID/a4/C.c .
```

Then you can go to the practicum web site and submit the program. You may find it easier to submit directly from bluenose. If you are developing programs on your computer, you can submit them to the practicum from there, and copy them later to bluenose to submit them via SVN.

Step 2: Lab Directory Setup. If you are done with the practicum, or if you want to finish the lab first, you can continue with this step. You should be logged in your bluenose account as before, and now you should prepare your lab directory on SVN. First, go to your SVN directory:

```
cd ~/csci2132/svn/CSID
```

where *CSID* is your CSID. It is always a good idea to run an 'svn update' in case there are some changes in the repository:

```
svn update
```

Then, create the lab directory:

```
svn mkdir lab7
```

and go to that directory:

```
cd lab7
```

Step 3: Getting program and test files. You should copy your program `A.c` from the first practicum to the current directory. If your program did not work correctly, you can copy the sample solution available in `~prof2132/public/practicums/A.c`. Remember that you can use the 'cp' command for this as follows:

```
cp ~prof2132/public/practicums/A.c .
```

There are also some test files in the above directory that you should copy. They all start with the uppercase 'A' and they end either with '`.in`' or '`.out`' so you can copy them with the commands:

```
cp ~prof2132/public/practicums/A*.in .
cp ~prof2132/public/practicums/A*.out .
```

Step 4: Simple Makefile. If you type 'ls' command, you should see the following files in your directory: `A.c`, `A.in`, `A.out`, `A-2.in`, and `A-2.out`. Add these files to SVN. You do not have to commit the files right away. You can do that later. Using Emacs, create a simple textual file named 'Makefile' with the following content. Have in mind that the second line **must** start with the Tab character, just before '`gcc`', which is obtained using the Tab key on the keyboard:

```
A.exe: A.c
        gcc -o A.exe A.c
```

Save the Makefile and run the command:

`make` The command should compile the program `A.c` into the executable program `A.exe` without any errors. If you get an error such as:

```
Makefile:2: *** missing separator (did you mean TAB instead of 8 spaces?).  Stop.
```

it means that you did not use proper tab character at the beginning of the line that continues with '`gcc`' command. You should fix this. If everything was done correctly, then the 'make' command should produce the following output:

```
gcc -o A.exe A.c
```

without any further messages. Try running the 'make' command once more. Nothing should happen. In this way we can use the Makefile to record the recipe of how to compile a program. When we run make, it will read the Makefile and interpret the recipe in the following way:

The line '`A.exe: A.c`' means that we want to create the target file named '`A.exe`' and it is made from the 'dependency', i.e., the file it depends on, called '`A.c`'. The next line contains the command that needs to run. We can have several commands following this command, and they would all be run. It is important that they all start with the tab character. When we run the 'make' command next time, based on the dependencies it will realize that the file `A.exe` is newer that the file `A.c`, so it is not going to repeat the process. If we change the file `A.c`, then the compilation will be repeated.

Step 5: Testing with Makefile. The existing Makefile contains a recipe for making the executable '`A.exe`'. A Makefile is not limited by one recipe—we can have several recipes. For example, we want the create a recipe for testing the program A. To add this recipe, add the following lines at the end of the Makefile using Emacs:

```
test-A: A.exe
        ./A.exe < A.in > A.new
        diff A.out A.new
        ./A.exe < A-2.in > A-2.new
        diff A-2.out A-2.new
```

Again, it is important that you type Tab characters at the beginning of each line after the first line. This new recipe does not create a file named 'test-A' but this name is just a virtual target name, and we will see how it is used. The dependency '`A.exe`' means that this file must be created before we run the test. If the file is not created, the 'make' will create it using the previously provided recipe. After that, 'make' will execute the four lines in the recipe, which will test the program by feeding input into it, saving the output, and then comparing the output with the expected output. The 'diff' command should return no output if the program is correct.

We can try to run 'make' but it will only make '`A.exe`' file, since this was the first recipe in the Makefile. If we want to choose to run tests, we need to issue command:

```
make test-A
```

an with this command we choose the target that we want to execute.

Step 6: Target 'help' in Makefile. One Makefile can have many targets, and it would not be convenient for a user to check for all possible targets by reading through Makefile. For this reason, it is a convention to set a virtual target 'help' at the beginning of the Makefile, which prints a helpful message about targets that can be called from the command line. Open the Makefile in Emacs and type the following lines at the **beginning** of the Makefile:

```
help:
        @echo make A.exe - to make executable
        @echo make test-A - to test program
```

Again, be careful to use tab character at the beginning of each line with the recipe command. If you save the Makefile and try the command 'make' you will see that the command prints the two help lines that we added, because 'help' was the first target that is specified in the Makefile. The reason why we added the character '`@`' at the beginning of the 'echo' lines is that we want to prevent the 'echo' commands to printed themselves. If you are not sure what this means, you can try removing the 'at' characters ('`@`') and run make again to see what it produces. We can also run 'make' with any of the targets:

```
make help
make A.exe
make test-A
```

Step 7: Recipes for programs C and D. Prepare similar recipes for creating `C.exe` and `D.exe`, and for testing them. You can copy sample input and output files for these programs from the practicum web site, or, more easily, from the directory '`~prof2132/public/practicums`' on bluenose. Add the new targets for 'C' and 'D' to the help message in Makefile as well. You should use these new recipes to test your programs for Practicum 2. For example, when you develop '`C.c`' you can type '`make test-C`' to test the program and see if it perfectly matches the expected sample output.

Step 8: SVN Submission. You can commit now your files to SVN. Make sure that you have all the files, and do not submit any additional files. The following files, and **only** these files should be submitted to SVN: `A.c`, `A.in`, `A.out`, `A-2.in`, `A-2.out`, `Makefile`, `C.c`, `C.in`, `C.out`, `C-2.in`, `C-2.out`, `D.c`, `D.in`, `D.out`, `D-2.in`, `D-2.out`. The files `C.c` and `D.d` can be submitted later, when you finish them. Similarly `C-2.in`, `C-2.out`, `D-2.in`, and `D-2.out` are your own test files that can be submitted later. For start, you could just copy `A.c` to be `C.c` and `D.c` and copy `C.in` to `C-2.in` and similarly create `C-2.out`, `D-2.in`, and `D-2.out`. In this way, you will have all necessary files submitted, and once you edit them to be correct files, you can do 'svn commit' again.

Step 9: SVN Creation of GitLab account. In this final phase of the lab, you are asked to create a GitLab account on FCS Dal GitLab server. This server is very similar with the popular site GitHub, with a difference that it is not a third-party site, but data is stored in one of the local Faculty of Computer Science servers. You are encouraged to open the an account on GitHub as well, where you can show-case publicly your projects.

To open the GitLab account, you need to go the following URL: `https://git.cs.dal.ca/`
You should use your CSID and password to login (Sign In).
- Once you are logged in, click on the button 'New Project' to open a course project.
- Under Project Name, enter: `csci2132`
- Do not change the Project URL and Project slug, which should look as follows:
`https://git.cs.dal.ca/`*CSID*`/` and `csci2132` where *CSID* is your CSID.
- You can enter some project description if you want, maybe "This is a course project..." or similar
- **Keep** visibility "Private"
- Choose to initialize repository with a README
- and click "Create project"

The project is now created, and you will be on the project page.

On the bottom of the left-hand side, you will see a menu item called 'Settings'. If you move mouse over it, you will see sub-item 'Members' — click on it. A new form will open. Go to the field 'Select members to invite' add 'Vlado Keselj' (`@vlado`). Set 'Choose role permission' to Developer. After this you can go to the main project page and logout from GitLab, by clicking on the pop-down menu in the top right corner, and choosing 'Sign out' option.

Step 10: The work on this lab is finished. You can continue working on the practicum and the rest of the assignment. Do not forget to commit all required files to SVN.