Preparing for CSE 340 programming projects

Mohsen Zohrevandi mohsen@asu.edu

Last Update: August 21, 2015

Abstract

We have an automated grading system for this course that evaluates your programs by running them on multiple test cases and comparing the output with the expected output for each case. This requires a single development platform to be used by all students and the grading software. In this semester we have chosen to work on CentOS 6.7 because the 2^{nd} floor lab has been equipped with machines running this flavor of Linux.[1] This document explains how to write code compatible with the automated grading system. It also provides instructions on how to install CentOS on a virtual machine so that you can develop and test programs on your own machine.

Contents

[1] The lab has been equipped with Red Hat Enterprise Linux which is binary compatible with CentOS. See the following page for more information: http://en.wikipedia.org/wiki/CentOS

1 Project Grading

We grade the programming projects with a shell script that automatically compiles and runs your programs on a set of test cases.

1.1 Test Cases

All assignments have a precise input/output specification that allows one to test the correctness of the program by comparing the program's output with the expected output. For each project we provide a description of the program's desired functionality along with input/output format and a set of sample test cases. Each test case consists of an *input* file and an *expected output* file. The grading script runs your program for each test case. It runs the program with the *input file* as input and compares the output generated by your program with the *expected output*. Each test case is either passed or failed depending on the result of the comparison. The files are compared with the following command:

```
$ diff -Bw file1 file2
```

If there are any differences between the two files, `diff` will output the differences in a special format, otherwise it will not output anything.[2]

Usually the set of test cases used for grading contains some additional test cases that are not available to students. The additional test cases might cover some special cases that are not represented in the sample set, so it is very important that you read and fully understand the project specification and not just rely on sample test cases. Your project grade will be a function of the number of test cases that your program correctly passes.

1.2 Standard I/O Redirection

Your programs should not access any files. In other words, you should not open any file in your program for reading or writing. Instead, your program should read the input from standard input and write the output to the standard output. Also, any output to standard error is ignored by the test scripts so you may use standard error for debugging messages. To feed the contents of a text file as input to a program, we use I/O redirection. Suppose we have an executable program `a.out`, we can run it by issuing the following command in a terminal (the dollar sign is not part of the command):

```
$ ./a.out
```

The above command runs the program and if the program uses standard I/O, it will print messages on the terminal screen and wait for input from keyboard. Now suppose we have a text file `some_input.txt` that contains all the input for our program. We can redirect the standard input to our input file:

```
$ ./a.out < some_input.txt
```

[2] If you are interested to learn more about `diff` and it's options, you can find more information in the following page: `http://linux.die.net/man/1/diff`

The < followed by a file name is used to redirect standard input to a file. This way the program will not wait for input from keyboard, instead, it will treat the contents of `some_input.txt` as if it were entered on the keyboard. The output is still printed to the terminal screen. To redirect the output to file `output.txt`, we issue the following command:

```
$ ./a.out < some_input.txt > output.txt
```

The > followed by a file name is used to redirect the standard output to a file. Note that the above command does not redirect standard error, and if the program prints anything to standard error, it will appear on the terminal screen while the normal output is written to the specified file.

To put everything in an example, let's assume that we need to write a program that reads any alphanumeric, space and newline characters from the input and prints `OK` or if it encounters any other character, it prints `ERROR` to the standard output. Here is a C program that does the job:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main() {
    char c;
    while (!feof(stdin)) {
        c = getchar();
        if (!isalnum(c) && c != '\n' && c != ' ' && c > 0) {
            printf("ERROR\n");
            exit(1);
        }
    }
    printf("OK\n");
    return 0;
}
```

Now we compile the program:

```
$ gcc -Wall sample.c -o sample.out
```

The executable file is `sample.out`, if we execute it normally:

```
$ ./sample.out
```

The program waits for input from keyboard and you can press `Ctrl+D` to simulate end of file. It prints `ERROR` or `OK` on the terminal screen. Now let's create a test case for it. Here is the contents of a text file named `test.txt`:

```
hello world this text file
only contains alphanumeric space and newline characters
here is a number 12345
```

We expect the program to generate an OK for this input:

```
$ ./sample.out < test.txt
OK
```

We can automate this by creating a text file named `test.txt.expected` with the following contents:

```
OK
```

Then to verify the program, we redirect both input and output, and then compare the output to the expected file:

```
$ ./sample.out < test.txt > output.txt
$ diff -Bw text.txt.expected output.txt
```

If the program produces output according to the expected file, we should see no output from the `diff` command. Otherwise `diff` will show the difference between the output generated by the program and the expected output.

1.3 Test Script

As mentioned before, the grading and testing are automated using shell scripts. In this section I give you details of the test script that you should use to test your programs during development. The test script is a Bash script and can be executed by:

```
$ ./test1.sh
```

The script file needs execution permission otherwise it won't run and you will get the following error message:

```
bash: ./test1.sh: Permission denied
```

To fix the problem, issue the following command:

```
$ chmod +x test1.sh
```

The test script assumes that there is a sub-folder named `tests` containing all the test cases. It also assumes that your executable program is named `a.out` and is in the current folder. It runs your program with I/O redirection for each test case in the tests folder and compares the output with the expected file using `diff` and reports any differences for failed cases. At the end it will print the number of tests passed and removes any temporary files created. Here is the script source code:

```
#!/bin/bash

let count=0;
for f in $(ls ./tests/*.txt); do
  ./a.out <$f > ./tests/‘basename $f .txt‘.output;
done;

for f in $(ls ./tests/*.output); do
  diff -Bw $f ./tests/‘basename $f .output‘.txt.expected > ./tests/‘basename $f .output‘.diff;
done

for f in $(ls tests/*.diff); do
```

```
  echo "==================================================";
  echo "FILE:" `basename $f .output`;
  echo "==================================================";
  if [ -s $f ]; then
    cat ./tests/`basename $f .diff`.txt;
    echo "------------------------------------------------------";
    cat $f
  else
    count=$((count+1));
    echo "NO ERRORS HERE!";
  fi
done

echo $count;

rm tests/*.output
rm tests/*.diff
```

The last two lines of the script remove the output files of your program and the results of `diff`. If you want to examine the output of your program, you can comment out the last two lines of the script by putting a # at the beginning of the lines:

```
#rm tests/*.output
#rm tests/*.diff
```

2 Development Tools

2.1 Platform

We will use the GCC compilers that come standard with CentOS 6.7 to compile your programs. To avoid any compiler compatibility issues, please use this version of GCC[3] for program development. Here is a list of most important things to keep in mind:

- Use CentOS 6.7 and its standard GCC compiler for all CSE 340 projects.
- The grading is automated, so stick to the input/output specification of the project.
- Only use standard input/output. Do NOT open any files in your code.
- The test cases are fed to your program with standard I/O redirection. See section 1.2 for details.
- Use the test script that will be provided later to test your program for all projects. The test script uses the same technique as the grading scripts. It tells you how many test cases are passed and gives you the result of `diff` for failed cases so that you can fix the problems.

[3] Currently version 4.4.7. Note that there are newer versions of GCC that might have additional features not present in this version of the compiler. Avoid using other versions of the GCC compiler.

2.2 Compiling Programs

You should compile your programs with the GCC compiler which is available in CentOS 6.7. The GCC compiler has separate commands for compiling C and C++ programs, use gcc to compile C programs and use g++ to compile C++ programs. Here is a simple command to compile a C program that is stored in file `test.c`:

```
$ gcc test.c
```

If the compilation is successful, GCC will generate an executable file named `a.out` in the same folder as the source file. You can change the output file name by `-o` switch:

```
$ gcc test.c -o test.out
```

To enable all warning messages of the GCC compiler, use `-Wall` switch:

```
$ gcc -Wall test.c -o test.out
```

Using `-Wall` switch will report all sorts of useful information about your program like unused variables, etc. The same options can be used with g++ to compile C++ programs.

If your program is written in multiple source files that should be linked together, you can compile and link all files together with one command:

```
$ gcc file1.c file2.c ...
```

Or you can compile them separately and then link:

```
$ gcc -c file1.c
$ gcc -c file2.c
$ ...
$ gcc file1.o file2.o ...
```

The files with `.o` extension are object files but are not executable. They are linked together with the last statement and the final executable will be `a.out` if there are no errors. To automate the compilation process you can write a Makefile, see section 2.4 for details.

2.3 Debugging

There are debugging tools available for CentOS that can be used to trace your program execution, see GDB and DDD for example:

```
http://www.gnu.org/software/ddd/
http://pkgs.org/centos-6/epel-x86_64/ddd-3.3.12-6.el6.x86_64.rpm.html
http://heather.cs.ucdavis.edu/~matloff/ddd.html
```

GDB is very powerful, but it does not have a nice user interface like other debuggers you might be familiar with. DDD provides a GUI frontend for GDB. The DDD installation is not straightforward, but you can find instructions in the second URL above in the *Install Howto* section. Remember that

to install packages you need to be in root mode, so switch to root by issuing `su` command and exit the root mode by `exit` command after you finish installing packages.

You can also debug your programs by printing meaningful messages at certain points in your code. As mentioned in the previous section, all output to `stderr` is ignored by the test scripts, so you may print your debugging messages to standard error. Here is a small C program that reads a character from standard input, then writes a message to standard output and another message to standard error:

```
#include <stdio.h>

int main() {
    char c;
    c = getchar();                          // Read 1 character from stdin
    printf("Input was: %c\n", c);           // Output to stdout
    fprintf(stderr, "debug message!\n");  // Output to stderr
    return 0;
}
```

However, standard error is not buffered while standard output *is* buffered, so using both might result in some confusion in the order of messages printed. Alternatively, you can avoid using `stderr` for debugging and use `stdout` with an extra flag to turn on/off debugging messages, here is an example:

```
#include <stdio.h>
#define DEBUG 1     // 1 => Turn ON debugging, 0 => Turn OFF debugging

void print_debug(char * msg) {
    if (DEBUG)
        printf("DEBUG: %s\n", msg);
}

int main() {
    char c;
    c = getchar();                          // Read 1 character from stdin
    printf("Input was: %c\n", c);           // Output to stdout
    print_debug("debug message!");          // Output to stdout if DEBUG != 0
    return 0;
}
```

A more complex `print_debug` function that accepts a format string and variable number of parameters and can be used like `printf` is listed below:

```
#include <stdio.h>
#include <stdarg.h>
#define DEBUG 1     // 1 => Turn ON debugging, 0 => Turn OFF debugging

void print_debug(const char * format, ...) {
    va_list args;
    if (DEBUG) {
        va_start (args, format);
        vfprintf (stdout, format, args);
        va_end (args);
    }
}

int main() {
    char c;
    c = getchar();                          // Read 1 character from stdin
    printf("Input was: %c\n", c);           // Output to stdout
    print_debug("%c was read\n", c);        // Output to stdout if DEBUG != 0
    return 0;
}
```

In C++, you can use `cin` to read from standard input, `cout` to write to standard output and `cerr` to write to standard error.

2.4 Makefiles

Makefiles are used to automate the compilation process. You can store all compiling instructions in a makefile and then just type `make` to compile the program. Our grading scripts compile your programs automatically by detecting the language (C or C++) based on file name extensions and running gcc or g++ with no additional options. However you may need to pass certain compiler options to gcc in order for your program to be compiled correctly in which case you can provide a Makefile with your project that tells us how to compile your program.

The compilation script automatically searches for a makefile in the files you submit. The script expects the name of your makefile, if you have one, to be `Makefile` (note the capital M in the name and the absence of an extension in the file name). No other file names are accepted.

The reference manual for `make` can be found at:

`http://www.gnu.org/software/make/manual/make.html`

If you need to write a Makefile for your project, you can use the following generic Makefile. Just remember to adjust it for your needs:

```
src = *.c
hdr = *.h
dep = $(hdr) $(src)
bin = a.out

$(bin): $(dep)
    gcc -Wall $(src) -o $(bin);

all: $(bin)

clean:
    rm $(bin);
```

Do not copy/paste the Makefile from this pdf, download the Makefile from the course website.

3 Installing CentOS on a virtual machine

If you prefer to work on your own machine rather than going to the lab, you can install CentOS 6.7 on your machine. Since it is highly unlikely that you would want to switch to CentOS for all your activities, it is better to keep your own operating system intact and install CentOS on a virtual machine. A virtual machine management (VMM) software allows you to create virtual machines and install all kinds of operating systems on them. There are many different VMM softwares available, some free of charge and some not. VirtualBox is one example that is free and open source. It can be installed on different platforms including Microsoft Windows and Mac OS X. You can use the following URL to download VirtualBox for your machine:

`https://www.virtualbox.org/`

In VMM parlance, the operating system that you already have on your machine is called the *host OS* and the operating system that you install on a virtual machine is called the *guest OS*. The following page contains an **installation guide** for different host operating systems:

`https://www.virtualbox.org/manual/ch02.html`

3.1 Creating a virtual machine in VirtualBox

Follow the instructions in the VirtualBox user manual listed below to create a new virtual machine:

`https://www.virtualbox.org/manual/ch01.html#gui-createvm`

Use the following settings when creating the virtual machine:

Name:	*centos*
Type:	*Linux*
Version:	*Red Hat* or *Red Hat (64 bit)* depending on your machine's architecture
Memory size:	*1024 MB*
Hard drive type:	*VDI*
Storage method:	*Dynamically allocated*
Disk size:	*8 GB*

After finishing the wizard, you will have a new entry in the VirtualBox window for the VM you created. Now you can start the VM and install CentOS 6.7 on it, but before that you need to download an ISO image of CentOS 6.7. **Don't start the VM yet**, follow the instructions in the next section to download and install CentOS on your VM.

3.2 Installing CentOS

You can download CentOS 6.7 from the following URL:

```
http://wiki.centos.org/Download
```

Click on *i386* or *x86_64* depending on your machine's architecture (unless your computer is very old, choose x86_64) and choose a mirror site to download the ISO image.

Note that CentOS 6.7 is not the latest version of CentOS. You need to download CentOS 6.7 not 7.0. To get higher download speed, look for mirror sites that are geographically closer to your location.

After downloading the appropriate ISO file, start the installation of CentOS by starting the VM in VirtualBox and specifying the downloaded ISO file location in the *Select start-up disk* window. The following page provides a step-by-step guide for installing CentOS:

```
http://www.tecmint.com/centos-6-5-installation-guide-with-screenshots/
```

There is a known issue in CentOS installation[4]:

> The message "Insufficient memory to configure kdump!" appears during install. This is a known issue which appears on systems with less than 2 GB RAM. This can be ignored.

When the installation is finished, make sure you remove the CD/DVD from the virtual drive by clicking on the following VirtualBox menu:

```
Devices > CD/DVD Devices > Remove disk from virtual drive
```

If you already have an older version of CentOS like 6.5, you can upgrade your existing installation to CentOS 6.7 by following simple instructions outlined in the following page:

```
http://www.tecmint.com/upgrade-centos/
```

In all cases, make sure you install programming packages and if you prefer to work with an IDE, CentOS comes with Eclipse which can be installed during the installation or afterwards. To make sure you have C/C++ compilers installed on the VM, open a terminal window in CentOS and type:

```
gcc --version
```

You should get an output similar to the following:

```
gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-16)
Copyright (C)...
```

[4] Known issues section in `https://wiki.centos.org/Manuals/ReleaseNotes/CentOS6.7`

If GCC is not installed, you can install it by issuing the following commands in a terminal:

```
$ su
$ yum groupinstall 'Development Tools'
```

The first command switches to the root user, so it will ask you to enter the root password which you set during CentOS installation. The second command will install a number of packages including GCC C/C++ compilers and `make`.

3.3 Installing Guest Additions

To have better integration with the host OS and enable some additional features, you need to install Guest Additions on the VM. You can do so by clicking on `Devices > Install Guest Additions...` then click OK then Run and enter your root password in the next screen.

If the Guest Additions installation fails, run the following commands in a terminal:

```
$ su
$ yum update -y
```

Then reboot the VM and try installing the Guest Additions again.

3.4 Sharing files with the VM

If you need to copy files between the host OS and the VM, you can either use conventional cloud solutions like Dropbox etc. or if you prefer to keep your files local, you can use VirtualBox shared folders capabilities. This requires the Guest Additions to be installed on the VM. To specify a folder on the host OS to be shared with the VM, open `Devices > Shared Folders...` then click on the plus button to open the Add Share window. Specify the folder path on the host OS and give it a name and check `Make Permanent` and `Auto-mount` so it automatically mounts the folder in the guest OS.

To be able to access the shared folder you need to become a member of the group `vboxsf`. Open a terminal window in the VM and enter the following commands to add your user to the group `vboxsf`:

```
$ su
$ usermod -a -G vboxsf user_name
```

Replace `user_name` with your user name. You can access your shared files by going to `/media/sf_share_name` in the file browser after rebooting the VM. See the following URL for more information about VirtualBox shared folders:

`http://www.virtualbox.org/manual/ch04.html#sharedfolders`