$Id: lab0-unix-c-make.mm,v 1.59 2022-03-29 18:06:09-07 - - $
/afs/cats.ucsc.edu/courses/cse111-wm/Assignments/lab0-unix-c-make
https://www2.ucsc.edu/courses/cse111-wm/:/Assignments/lab0-unix-c-make/

1. Introduction

This lab will not be graded and there is no credit for it. However, it is important you understand the Unix command line and submit procedure before you actually use it in the first assignment. Learn how to submit files. Submit these files to **`lab0`**. Verify that the submit actually worked by looking in the submit directory for your username. The **`find`**(1) command is very useful for things like this.

The submit command copies your files into a directory

`/afs/cats.ucsc.edu/class/cse111-wm.s22/lab0/$USER`

and prefixes each file with a sequence number. You can see the names of the files, but not their contents. The actual syntax of the **`submit`** command is:

`submit cse111-wm.s22 lab0` *files*...

2. Prior experience

Prior experience with Unix is assumed. You should know the following:

- The C language, from a prerequisite course.
- General use of Linux commands, and specifically **`submit`** and **`checksource`**.
- Use of environment variables and setting the **`$PATH`** variable.
- Construction and use of **`Makefile`**s.
- A code archival system such as **`rcs`**, **`cvs`**, **`svn`**, **`git`**, etc.

Attend a lab section to ask questions if you don't understand how submit and the Unix command line in general works. The TA can explain such things. Also read the submit checklist: **`Syllabus/submit-checklist`**.

There are some scripts that are available, such as **`checksource`**, **`cpplint.py`**, **`cpplint.py.perl`**, **`cid`**, and others. These can be made available by adding the following to your **`$PATH`** environment variable:

`/afs/cats.ucsc.edu/courses/cse111-wm/bin`

This can be done with the following line in your **`$HOME/.bash_profile`**:

```
export PATH=$PATH:/afs/cats.ucsc.edu/courses/cse111-wm/bin
```

3. Version of C++

```
-bash-1$ which g++
/opt/rh/devtoolset-11/root/usr/bin/g++
-bash-2$ g++ --version | grep -i g++
g++ (GCC) 11.2.1 20210728 (Red Hat 11.2.1-1)
-bash-3$ uname -npo
unix2.lt.ucsc.edu x86_64 GNU/Linux
```

Be sure that any textbook or other reference to C++ includes at least C++11 on its cover documentation, and preferably C++14 or C++17. Whenever you need to look up information about C++, or look at **`http://www.cplusplus.com/`**.

Any code you develop must compile and run on **`unix.ucsc.edu`**. That is the only environment the graders will be using. If you are developing on your own system, verify that the compiler you are using is at least C++14 compliant. Frequently port your code to **`unix.ucsc.edu`** and verify that it compiles and runs there. If it does not work there, then it does not work.

4. Review of C and make

The prerequisite for this course is a knowledge of C, the use of the **`make`**(1) utility, and the use of Unix in general. As a review, write the following program and submit to **`lab0`**. There is no credit for this lab and it will not be graded, but it is useful to verify that you have the necessary knowledge of C. Alternatively, write the following program in C++.

5. Program specification

Write a program **countwords** that will read in words from all files specified on the command line, or from the standard input if no files are specified on the command line. After all of the files have been read in, print each word followed by a count of the number of times it occurred. A word is any sequence of alphabetical (**isalpha**(3)) characters, and words are converted to lower case (**tolower**(3)). The following Perl program (**countwords.perl**) shows the required functionality. You do not need to know Perl in order to run the program.

```
#!/usr/bin/perl
# $Id: countwords.perl,v 1.1 2020-01-02 14:32:14-08 - - $
map {++$words{lc $_}} m/([[:alpha:]]+)/g while <>;
print "$_ $words{$_}\n" for sort keys %words;
```

6. Program implementation

Write a program in C consisting of three modules as shown here. Also construct a **Makefile** with the usual targets **all**, **clean**, and **spotless**, and recipies to build the object files and binary from source files. Do not assume anything about the maximum length of a word, the maximum number of words in the file, the maximum number of files, or any other maxima. Suggested files :

(a) **vstring.h**, **vstring.c** — implements a variable string into which characters can be accumulated, using array doubling when the length of the string is exceeded.

(b) **wordlist.h**, **wordlist.c** — implements a list sorted lexicographically with each element of the list containing a word and a count.

(c) **main.c** — iterates over all of the files specified on the command line, or the standard input if none. For each word found in each line, insert into the list, updating the count. At end of file, print out all of the words in lexicographic order along with the count of the number of times each word appears.

(d) **Makefile** targets :

- **all** : builds the executable binary.
- **${EXECBIN}** : builds the executable from the object files.
- **%.o** : builds object files from source files.
- **clean** : removes the object files.
- **spotless** : first does **clean**, then removes any files not part of the source.
- **Makefile.deps** : builds the dependency file.

As mentioned, this program will not be graded, so coding it is optional, but you should verify that you know enough C to do it. Exact specification details depend on variations on implementation preferences.

7. Alternative versions

Use a binary search tree instead of a simple linear list.

Use a hash table to hold the information, then sort the table before printing.

Code in C++, but do not use any of **<map>**, **<regex>**, **<string>**, **<vector>**. Use only raw pointers, **struct**s, **malloc**(3), and **free**(3) for the data structures.

8. Sample output

```
-bash-4$ countwords.perl countwords.perl mk Makefile | column -c80
a 1             do 1            keys 1          pe 1            touch 1
afs 1           done 1          lab 1           perl 4          tps 1
all 6           echo 3          label 3         pkill 2         true 1
alpha 1         edu 1           lc 1            popd 1          tt 5
b 1             etc 1           letterbbox 1    print 1         ttopts 4
basename 1      f 1             lis 1           ps 6            u 1
bash 1          filter 1        ls 1            psopts 4        ucsc 1
bin 2           firstword 1     m 2             pushd 1         unix 1
call 2          font 1          make 5          rref 2          usr 1
cats 1          for 2           makefile 2      s 1             v 3
cbuo 1          g 1             map 1           shell 1         version 1
ci 1            groff 2         mgm 1           sort 1          while 1
cid 3           groffdir 5      mk 1            spte 1          wm 1
countwords 3    groffmm 5       mkpdf 1         squeeze 2       words 3
courses 1       gv 3            mm 5            st 1            ww 1
cse 1           i 1             n 1             stars 3         z 2
d 1             id 3            new 4           t 1
deps 3          in 1            newest 1        tar 1
dir 2           intro 1         p 1             tlatin 1
diropts 2       is 2            pdf 6           tmac 4
```

9. Simple C++ programs

Some simple programs you might want to consider as practice for an introduction to C++:

(a) Write a program which reads words and at end of file, prints the number of words found in `cin`. If `string word` is a variable, `cin >> word` will read in the next white-space delimited word.

(b) Write a program which used `cin >>` to read numbers. At end of file, print the average.

(c) Any other simple program you have implemented in your first programming course, re-implement it in C++.

Submit these to lab 0. Reminder: lab 0 will not be graded — it is just a review of Unix and the `submit` command.