University of Reading
Department of Computer Science
Pat Parslow, Rhian Taylor

Exercise Week 5
CS1PC20 / Autumn 2021
120 Minutes
Discussion date: Week 7

---

1. The exercises are formative – they are there to learn – but also assessed as part of your portfolio of work submitted for course works.
2. Exercises will be discussed on the Week indicated on the header. Additional (non-discussed) training tasks may be provided, and will be shown in red.
3. Make sure to schedule some time each week to keep up with the progress. It is not necessary to complete every task perfectly. If you are struggling, then reach out for help on Teams.

Exercise Introduction

Using an iterative algorithm to convert between decimal and binary, merging our git branches, and 'compiling' our report

Remember:

You should have done exercises 1 (bash) and 2 (git) and 3 (programming) and 4 before this exercise – we are creating a real program!

If you are finding you do not get the expected results, it is almost always (except where otherwise noted) because of a typing mistake or failure to follow the steps in order. Do check!

Be careful to distinguish between characters that look similar:

1 is not l is not I is not | for instance (these are one, lower case L, capital i and the pipe symbol, respectively)
O is not 0
~ is not – and ~ is called the "tilde"
_ is called underscore
is called an 'octothorpe' but almost nobody actually calls it that, it is 'hash'
" is (on UK keyboards) shift 2. If you are cheating and using copy/paste, you may end up with a 'smart quote' which is not the right character. Do not use copy/paste.

Contents

Task 5: Programming, git and report generation exercise (120 min)

We don't expect for you to be writing long and complex C programs straight away. We also appreciate that moving to a CLI (Command Line Interface), while powerful, can be challenging especially when you are accustomed to interacting with primarily graphical interfaces on Operating Systems like Windows, and that C can be a lot less forgiving than that other snake-related language you may have used before (python).

Below we have included several steps for creating simple C programs of varying complexity that also cover more advanced concepts.

We want you to *experiment with them*, try running them and observe the result! Think about what you expect their output to be before you run them.

See what happens if you change the code (program) slightly – does it have the effect you expect?

Make your own changes to the program code. Use this as an opportunity to learn about C and programming practice.

We would like you to produce a markdown text file providing a short descriptions of your findings, i.e. all you need to provide is a brief description of what the command does (if possible write what you expected) and a short explanation of how it works where appropriate. Marks do not depend on you having the right 'prediction' – so avoid the temptation to run it and then record what happened as what you think will happen!

You should have completed the week 1 & 2 & 3 & 4 exercises first. Programming courses often start with "Hello World" (we did, in fact, in week 1) written as a 'main' function in C. We want to avoid that, and write our programs in a way that makes it easy to test them, so we will be writing functions in libraries, and using those libraries in a test program.

This week we will be writing a simple conversion from decimal to binary, merging our git branches and generating a report for the first coursework submission.

You do not need to type in the $ at the beginning of each line (in fact, you need to not type it in…)

Where the instructions say to edit a file to say something, you can use nano, vim, sed, echo commands… you'll probably want to stick with nano or vim though! If you are using nano, the commands for saving and exiting are listed along the bottom of the screen (ctrl-O to write the file, ctrl-X to exit nano). If you use vim (and git may put you in to vim if you are not using our standard virtual machine), you need to know it has two "modes" – command and insert mode. To enter insert mode, press "i" (or a, or o… or actually a number of others – i is for insert, a for append, o inserts a new line below the cursor and enters insert mode on it). To get in to command mode you press the Esc (escape) key. To save and quit, when you are in command mode, use ":wq"

If the instructions have a term in angle brackets, like <this> you should replace that entire term with the appropriate content.
So, for instance, if the instructions say <tab> you should press the tab key rather than typing in < t a b >. Similarly if it asks for your <id> then you should use e.g. xy123645

1. `$ cd ~`

Do make sure you have the proper folder structure, and that your portfolio folder is a git repository.

If you have a folder called cs1pc20_portfolio instead of one called portfolio, you can rename it:

```
mv cs1pc20_portfolio portfolio
```

Portfolio should have week1, week2 and week3 folders in it, depending on which branch your git repository is currently on, and using

```
git status
```

should ***not*** say "fatal: not a git repository" if you are in a git repository. Assuming you have done exercises 2 and 3 you can recover your work from https://csgitlab.reading.ac.uk if necessary (see the end of Exercise 2 for the git clone command)

If your virtual machine is running out of disk space, follow the instructions on Blackboard to increase the disk size and the partitions.

1. `$ cd portfolio`
2. `$ git switch master`

Before we do anything else, let's merge the previous weeks' work together. If you have followed the instructions in previous sessions, this will be easy. If you have created 'spare' directories within your folder structure, you might need to clean it up somewhat.

Let's remind ourselves what branches we have in our git repository.
In week 1 we did our work without having a git repository, but we included that work in the repository at the start of week 2. This was on the master branch.

git branch

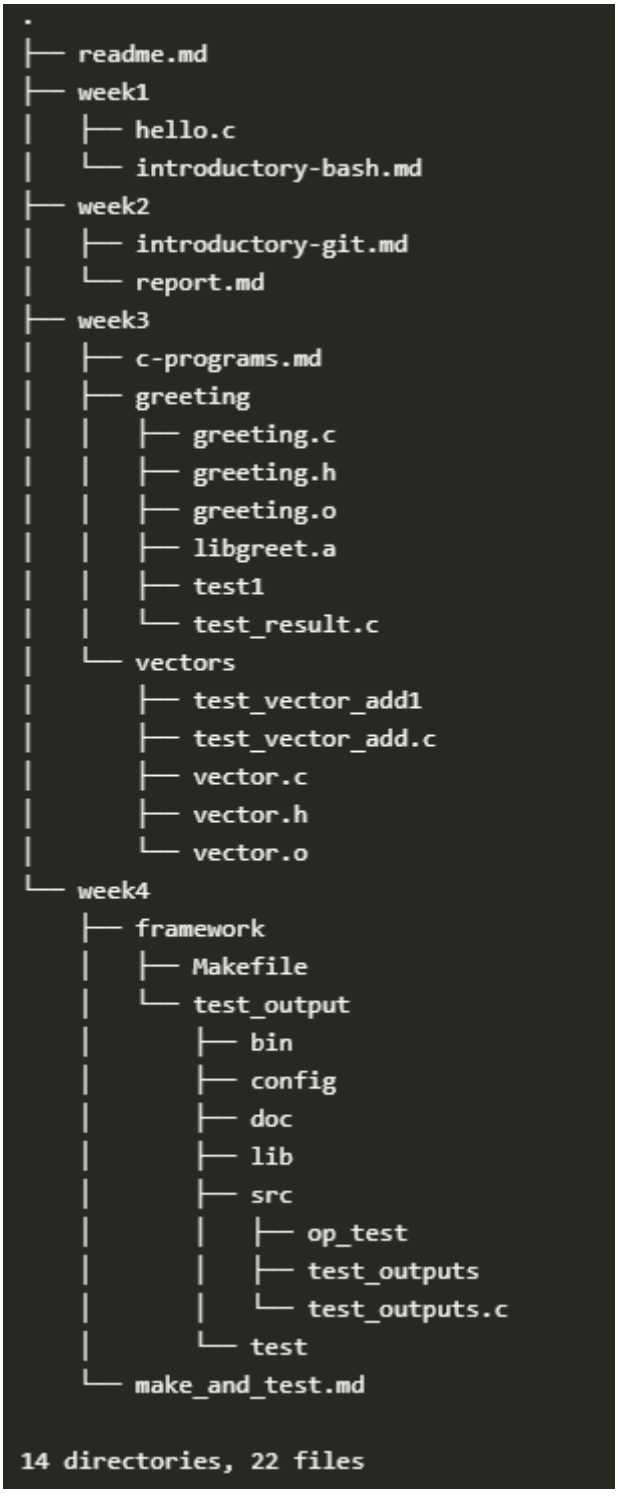
should therefore tell us we have
framework
greeting
master
vectors
week2

as our branches. If yours does not say this, we recommend you ask for help sooner rather than later!

We have already merged week2 (we did this during week2). When executing these merge commands **DO PAY ATTENTION** to the information the command returns on the screen. If you have 'stray' folders or files which you have added, then you may find that git reports that it cannot merge without your help. And that will probably mean you will want our help too!

3. `$ git merge greeting`
4. `$ git merge vectors`
5. `$ git merge framework`

At this point, run the command
tree
it should show you something like this:



If yours is significantly different to this – for example, missing things, or with extra directories, clean up your working tree!

6. `$ git branch baseconversion`
7. `$ git switch baseconversion`

Let's use the Makefile we created last week to set up a folder structure, but we will create a week5 folder first.

8. `$ mkdir week5 ; cd week5`
9. `$ make -f ../week4/framework/Makefile feature NAME=dec2bin`
10. `$ cd dec2bin`

Create a file called "dec2bin_tests" in the "test" folder. You can either cd in to test, create the file, and cd back up to dec2bin, or edit it from where you are, e.g.
nano test/dec2bin_tests

The file contents should be:

```
bin/dec2bin
0
                   0
bin/dec2bin
1
                   1
bin/dec2bin
8
                1000
bin/dec2bin
10
                1010
```

NOTE: the leading spaces on the expected output lines. The field should be 19 characters in total, so the 0 and 1 should have 18 spaces in front of them and the 1000 and 1010 should have 15 spaces in front of them. You can ask Linux to tell you how many spaces there are on each line…
cat test/dec2bin_tests | tr -d -c ' \n' | awk '{print length;}'
but explaining all of that might take a little while… and we have work to do!

Create a file called "conv.h" in the "src" folder. You can either cd in to src, create the file, and cd back up to dec2bin, or edit it from where you are, e.g.
nano src/conv.h

The file contents should be:

```
#define STRLEN 20
void dec2r(char in[], int r, char out[]);
```

Create a file called "conv.c" in the "src" folder. You can either cd in to src, create the file, and cd back up to dec2bin, or edit it from where you are, e.g.
nano src/conv.c

The file contents should be:

```
#include "conv.h"
#include <stdio.h>
/** convert a string from base 10 to another base <=10 and >1 (!)
* limit inputs to non-negative integers
* also assume (never a good idea!) that the input string is a valid number
* can consider other values later...!
*/
void dec2r(char in[], int r, char out[]) {
  int decval;
  sscanf(in, "%d",&decval);
  int pos=STRLEN-1;
  out[pos-1]='0';
  while (decval > 0) {
    out[--pos]=(decval % r) + '0';
    decval /= r;
  }
  return;

}
```

Don't worry – we will explain all of this later.

11. $ gcc src/conv.c -o lib/conv.o -c
12. $ ar rv lib/libconv.a lib/conv.o

Note: we are putting the files in the right folders to keep things tidy!

Create a file called "dec2bin.c" in the "src" folder. You can either cd in to src, create the file, and cd back up to dec2bin, or edit it from where you are, e.g.
nano src/dec2bin.c

The file contents should be:

```
#include "conv.h"
#include <stdio.h>

int main(int argc, char * argv[]) {

  /** requires a decimal value as the single command line argument
  */
  int num;
  char output[STRLEN]={[0 ... 18] = ' ', [19]='\0'};
  dec2r(argv[1],2,output);
  printf("%s\n", output);
  return 0;
}
```

13. $ gcc src/dec2bin.c -o bin/dec2bin -Isrc -lconv -Llib

Now, we will use the test_outputs program from week4. Unfortunately we were less than tidy and compiled it in to the src folder there, but nevermind, we can still use it!

14. $ ~/portfolio/week4/framework/test_output/src/test_outputs test/dec2bin_tests

And you should see something that looks like this:

```
Expected                    0
  and got                    0
OK
Expected                    1
  and got                    1
OK
Expected                 1000
  and got                 1000
OK
Expected                 1010
  and got                 1010
OK
```

If that has worked, time to merge our branch and create a report. If it hasn't – try and figure out where you went wrong and *ask for help if you need it!*

15. $ cd ~
16. $ `git switch master`
17. $ `git merge baseconversion`
18. $ `mkdir docs`

Creating the report using doxygen

First we create a doxygen configuration file:

19. $ `doxygen -g`

This produces a file called Doxyfile. We want to make some small changes to it.

Edit the file and change the line which says "PROJECT_NAME" so it says

PROJECT_NAME = "<student id>"

(substitute your actual id, e.g. xy345987 for <student id> so it looks like "xy345987"

OUTPUT_DIRECTORY = docs

OPTIMIZE_OUTPUT_FOR_C = YES

EXTRACT_ALL = YES

RECURSIVE = YES

20. $ `git add Doxyfile`
21. $ `Answer the questions on the coursework specification, writing your answers in to a file called "submission_answers.md"`

22. Create, or edit, a file in the main portfolio folder, called "_FrontPage.md" with the following contents (filled in as appropriate)

> Module Code: CS1PC20
>
> Assignment report Title: Portfolio
>
> Student Number (e.g. 25098635):
>
> Date (when the work completed):
>
> Actual hrs spent for the assignment:
>
> Assignment evaluation (3 key points):

23. `$ git add submission_answers.md`
24. `$ git add _FrontPage.md`
25. `$ git commit -m "added configured Doxyfile, answers and frontpage"`
26. `$ git push`
27. `$ doxygen`

That will create two folders inside your docs folder, html and latex

28. `$ cd docs/latex`
29. `$ make`
30. `$ git add refman.pdf`
31. `$ git commit -m "adding documentation"`
32. `$ git push`

Now, go and check your repository on CSGitLab. You should find the file "refman.pdf" in the docs/latex folder on the main branch.

The order of the weeks' exercises won't be in the week number order – but that does not matter at the moment. If, on reading your report, you want to make changes to any of your observations files that you took during the first 5 weeks, you can – but you *do not need to*. If you do, just run doxygen again from the portfolio folder, and then make in the docs/latex folder.

Submit your work on Blackboard

All you need to submit on Blackboard for this assignment is the URL of your CSGitLab repository, e.g. https://csgitlab.reading.ac.uk/xy345987/cs1pc20_portfolio

IF YOU HAVE NOT MANAGED TO COMPLETE THE EXERCISES, CREATE THE REPORT AND SUBMIT THE URL ANYWAY.

IF YOU CANNOT MANAGE TO CREATE THE REPORT, SUBMIT THE URL ANYWAY.

IF YOU ARE USING A MAC, YOU MAY NEED TO RUN THE FINAL 'make' STEP ON A DIFFERENT MACHINE, e.g. a PC in the lab (G56).

IF YOU DO NOT HAVE ACCESS TO THE LAB, LET ME KNOW IF YOU ARE HAVING TROUBLE WITH THE 'make' STAGE TO CREATE THE PDF. I CAN (for a small number of you!) DOWNLOAD YOUR CSGITLAB PROJECT AND RUN THAT FINAL STEP FOR YOU.

Hints

- We have used fixed size arrays – we will look at more flexible options later
- We have used very minimal sets of tests – think about what other values we should test
- We have not handled what happens if the input file is badly formatted (e.g. not enough lines, a spare blank line on the end…). Why might that be a problem for us?
- You can use

```
$ history > my_work_log.md
```

To produce a file with what you have done so that you can edit it with comments.

Portfolio (directory: portfolio/week5/bases_and_reports.md)

portfolio/week5/bases_and_reports.md

An enumerated list that provides for each command a prose sentence what the commands did. Where your expectation differs from observed behaviour, include a sentence describing what surprised you. See if you can explain what the commands did (briefly!)
You definitely do not need to reproduce the code in your .md file!