Open Source Software — CSCI-4966-01 — Spring 2019
Test 1
February 22, 2018

SOLUTIONS

1. The Open Source Initiative defines 10 characteristics of open source software. List 5 along with a **brief** explanation of what they mean: (15 pts)

   Need 5 of the 10 below. Each correct characteristic is worth 3 points with the **bolded** phrases (or close) worth 1 point. Description is worth 2 points.

   (a) **Free Redistribution** The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

   (b) **Source Code** The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost, preferably downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

   (c) **Derived Works** The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

   (d) **Integrity of The Author's Source Code** The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

   (e) **No Discrimination Against Persons or Groups** The license must not discriminate against any person or group of persons.

   (f) **No Discrimination Against Fields of Endeavor** The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

   (g) **Distribution of License** The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

   (h) **License Must Not Be Specific to a Product** The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

   (i) **License Must Not Restrict Other Software** The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

   (j) **License Must Be Technology-Neutral** No provision of the license may be predicated on any individual technology or style of interface.

2. Consider the scenario where a company is using open source software to produce applications that they then sell.

(a) What is the impact (license, ability to distribute, etc.) on their product if the open source software they used is licensed under: (9 pts)

i. GPL: Any code that is distrbuted (given away or sold) and that directly uses or links to GPL code must be licensed under the GPL or a compatible license.

ii. LGPL: Any code that is distrbuted (given away or sold) and that directly uses or **staticly** links to the LGPL code must be licensed under the LGPL or a compatible license. The idea here is that if the LGPL code can be modified and those modifications then used in the distributed product, the terms of the LGPL are met. That is why dynamic linking is allowed but not static linking.

iii. MIT: Any code that is distrbuted (given away or sold) and that directly uses or links to the MIT licensed code must carry copyright and license notice for the original work.

(b) Which license (in the open source software they are using) serves the company best if their business plan is to (4 pts):

i. Sell unique software that they developed which uses the Open Source Software

MIT -
Under the MIT license, they are free to license any changes they make according to whatever business plan they choose. They may even make the resulting software product proprietary so long as they include the original copyright and license notices when distributing the software.

ii. Provide support services around the Open Source Software they are using without creating new applications or code

GPL -
The company is not basing their business plan and ability to compete on any software differentiators. Instead they are competing based on service. The GPL ensures that any competitor cannot "outflank" them by making proprietary changes to the software that makes the competing version "better." Instead, the GPL requires that all such changes be disclosed keeping the competition for customers solely on the services provided.

Make sure you support your answer.

3. For each question below, circle the best answer (12 pts)

(a) Which command changes you repository to a new (existing) branch?

i. git checkout newbranch

(b) Literate programming:

i. Mixes code and comments in an easily human readable format

(c) Open or Free software:

i. Can be redistributed either for free or for a fee

(d) It is a good idea when selecting an open source license to:

i. Go to an authority such as the OSI and pick an approved license

4. Give a sequence of git commands to accomplish the following (you can assume that you are always working on the "master" branch") (15 pts):

   (a) Create a local copy of your repoository by cloning from "https://www.mypublicrepository/public.git". This will be your origin.

   (b) Create an "upstream" remote to point to "https://www.everyonespublicrepository/public.git"

   (c) Assume you have a new file "foo.txt" in your local directory. Add this file to your repository.

   (d) Send your changes to the public repository

   (e) Assume someone else makes changes in the "upstream". Add the changes in the public repository into your local version.

   Write git commands below:

```
git clone https://www.mypublicrepository/public.git
git remote add upstream https://www.everyonespublicrepository/public.git
git add foo.txt
git commit -m "adding foo"
git push origin master
git pull upstream master
```

5. Write markdown to duplicate the document below. You can assume the photo name is "photo.jpg" (10 pts):

Test File - Biggest Header

Next Smallest Header

1. Enumerated list
2. Of multiple lines
3. Just something else to type

A clickable link to Google (http://google.com) here: Google link

A picture from file photo.jpg in the current directory:

Write Markdown commands below:

```
# Test File - Biggest Header

## Next Smallest Header

1. Enumerated list
2. Of multiple lines
3. Just something else to type

A clickable link to Google (http://google.com) here: [Google link](http://google.com)

A picture from file photo.jpg in the current directory:
![](./photo.jpg)
```

6. Assume you have 3 source files: a main file "prog.c" and two additional files"f1.c", and "f2.c" containing code that "prog" depends upon. Write a Makefile that explicitly creates object files for all 3 sources, create a static library from the "f1.c" and "f2.c" object files, and then create an executable named "prog.exe". Make sure your Makefile contains appropriate "all" and "clean" targets. (15 pts):

   Write your Makefile below:

```
all: prog.exe

clean:
   rm *.o prog libf.so

prog.exe: prog.o libf.a
   cc -o prog.exe prog.o libf.a

prog.o: prog.c
   cc -c prog.c -o prog.o

libf.a: f1.o f2.o
   ar -qc libf.a f1.o f2.o

f1.o: f1.c
   cc -c f1.c -o f1.o

f2.o: f2.c
   cc -c f2.c -o f2.o
```

   or

```
all: prog.exe

clean:
   rm *.o prog.exe libf.a

prog.exe: prog.o libf.a
   cc -o prog.exe prog.o libf.a

prog.o: prog.c

libf.a: f1.o f2.o
   ar -qc libf.a f1.o f2.o

f1.o: f1.c

f2.o: f1.c
```

7. Repeat the previous exercise for CMake. (6 pts): Write your CMakeLists.txt file below:

```
cmake_minimum_required(VERSION 3.0)
project(Hello C)

add_library(Lib STATIC f1.c f2.c)

add_executable(prog.exe prog.c)

target_link_libraries(prog.exe Lib)
```

8. Consider the following code binary search code in file binary.py:

```
def binary_search( x, L):
    low = 0
    high = len(L)
    while low != high:
        mid = (low+high)//2
        if x >= L[mid]:
            low = mid+1
        else:
            high = mid
    return low
```

The code contains an error.

(a) Find and correct the error. Write your correction alongside the code above. You may use any tools on your computer to help you find the error. (2 pts)

```
def binary_search( x, L):
    low = 0
    high = len(L)
    while low != high:
        mid = (low+high)//2
        if x > L[mid]:
            low = mid+1
        else:
            high = mid
    return low
```

(b) Now, on the next page, write a sequence of tests in file binary_unit_test.py to ensure that the algorithm is correct. For full credit you will need to cover at least 4 different use cases. We give you the start and the end of the test file (12)

```
'''
Test binary.py with unittest
To run tests:
python binary_unit_test.py
'''
import unittest
from binary import binary_search

class TestBinaryPy(unittest.TestCase):

    def setUp(self):
        pass
```

```
    #
    # Pick 4 test cases from the following:
    #
    def test_null(self):
        print("Test the empty list")
        self.assertEqual(binary_search(-5, []), 0)

    def test_before(self):
        print("Test before front")
        self.assertEqual(binary_search(-5, range(0, 15)), 0)

    def test_first(self):
        print("Test at front")
        self.assertEqual(binary_search(0, range(0, 15)), 0)

    def test_after(self):
        print("Test beyond end")
        self.assertEqual(binary_search(15, range(0, 15)), 15)

    def test_last(self):
        print("Test at end")
        self.assertEqual(binary_search(14, range(0, 15)), 14)

    def test_duplicates(self):
        print("Test duplicates")
        self.assertEqual(binary_search(12, 10*[12]), 0)

    def test_middle(self):
        print("Test middle")
        self.assertEqual(binary_search(11, range(0, 15)), 11)
```

```
if __name__ == '__main__':
    unittest.main()
```