Homework 1: Azure, git, and linking

Accessing your virtual machine

For this class, you will be using a virtual machine hosted by Microsoft Azure. You will receive an access code and instructions for using it via email. Week one's lab section is the best way to get an introduction to using Azure. I encourage you to seek help on Piazza or during office hours ASAP if you can't get things running properly. When you've received your code, please enter it at the following link to get your account started up: Microsoft Azure Pass (http://www.microsoftazurepass.com/).

Once you've got an account and redeemed your code, you can go to the Azure portal, available here: Azure Portal Page (https://portal.azure.com/). In the portal, you should select **New** –> **Compute** –> **Ubuntu 14.04 LTS**. Then choose **classic** at the bottom of page for deployment model, click on **create** and choose your hostname, username and password. The VM will be instantiated and deployed.

For Mac/linux users, you can access the VM by `ssh` :

```
ssh [your chosen username]@[your chosen hostname]
```

For Windows users, you can access the VM with `puTTY` . For instructions on installing putty, please visit: putty home page (http://www.putty.org/).

git, personal and public repositories

The first objective of this homework is to get you familiarized with the versioning system we will be using for homework turn-in, called git. Git is a decentralized revision control system.

If you've never used git before, I'd suggest getting up to speed on it: you can learn some of the basics interactively here (https://try.github.io/levels/1/challenges/1).

Using your private key, check out your personal repository:

```
$ git clone git@git.uicbits.net:cs361-s16/YOURNETID.git
```

In this example, my username is `ckanich-student` . When you first clone the repository, it will be empty. Your first task is to add a file named `README.md` to your repository. You can put whatever text you would like in it, but you need to commit and push this commit to the course git server.

After you've put some text in `README.md` , to commit and push the file, you can run commands like so:

```
ubuntu@ip-10-143-165-210:~/ckanich-student$ git add README.md
ubuntu@ip-10-143-165-210:~/ckanich-student$ git commit -a -m'my first commit!'
[master e66c87f] added hw1 skeleton
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
ubuntu@ip-10-143-165-210:~/ckanich-student$ git push
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 1.13 KiB, done.
Total 6 (delta 0), reused 0 (delta 0)
To git@git.uicbits.net:cs361-s16/ckanich-student.git
 * [new branch]      master -> master
ubuntu@ip-10-143-165-210:~/ckanich-student$
```

NOTE: If you didn't follow these instructions fully, the next part won't work correctly!

Beyond the very basics of git, we will be using an "upstream" repository to distribute skeleton files for homeworks. This will make it easy for you to merge updates to skeleton files into your code. We'll push updates to the upstream for each new homework, as well as if there are any bug fixes to outstanding homework skeletons. If we do that, we'll announce it on Piazza.

To add the course public repository as an upstream repository, run:

```
git remote add upstream git@git.uicbits.net:cs361-s16/public.git
```

When new changes are available from the public repository, you can run this command to incorporate them into your local branch:

```
$ git fetch upstream
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From git.uicbits.net:cs361-s16/public
   868d644..b85fce8  master     -> upstream/master
$ git merge upstream/master
```

Now that you have the skeleton code, you can start coding. You should commit early and often, and push to your remote repository whenever is convenient to back up your work!

When you have finished your assignment, commit your changes, and then add a **git tag** called `hw1-submission` to the commit which you want graded, and then push both your code and the tag to `origin`. If you change your mind and want a different commit graded, delete the tag, add it to the new commit, and push that new tag.

Remember, if you don't push, we don't see it! We will be checking for your most recent tagged commit at each deadline (full credit, 10% off, etc), so do not leave this for the last minute. If you submit it one second too late, you'll be in the next lateness bracket, no exceptions.

Part of this assignment is to learn git; part of this class is to learn how to solve problems you encounter while programming. We have **not** included instructions on how to add a **git tag** to a **commit**. Google, however, is your friend, and can help you learn how to tag your commits.

The Programming Part!

This part will give you a quick introduction to using `readelf` and `dumpobj` to better understand the linking process.

In this assignment, you must fill hw1.c with code which will:

1. cause your username (and nothing else) to be printed on the first line of output when the program is run.
2. cause `gcc -Wall hw1.c` to issue zero warnings (and zero errors, duh).
3. cause the correct code to be in the current working directory when the grader runs `git checkout hw1`.
4. cause the output of `readelf -s hw1.o` to have identical values in the bolded sections of the output below:

Symbol table '.symtab' contains 18 entries:

Num:	**Value**	**Size**	**Type**	**Bind**	**Vis**	**Ndx**	**Name**
0:	0000000000000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	0000000000000000	0	FILE	LOCAL	DEFAULT	ABS	hw1.c
2:	0000000000000000	0	SECTION	LOCAL	DEFAULT	1	
3:	0000000000000000	0	SECTION	LOCAL	DEFAULT	3	
4:	0000000000000000	0	SECTION	LOCAL	DEFAULT	4	
5:	0000000000000000	0	SECTION	LOCAL	DEFAULT	5	
6:	0000000000000000	0	SECTION	LOCAL	DEFAULT	7	
7:	0000000000000000	0	SECTION	LOCAL	DEFAULT	8	
8:	0000000000000000	0	SECTION	LOCAL	DEFAULT	6	
9:	0000000000000000	81	**FUNC**	**GLOBAL**	**DEFAULT**	**1**	**main**
10:	0000000000000000	**0**	**NOTYPE**	**GLOBAL**	**DEFAULT**	**UND**	**printf**
11:	0000000000000051	38	**FUNC**	**GLOBAL**	**DEFAULT**	**1**	**all**
12:	0000000000000004	**4**	**OBJECT**	**GLOBAL**	**DEFAULT**	**COM**	**your**

13:	0000000000000077	38	**FUNC**	**GLOBAL**	**DEFAULT**	**1**	**cs361**
14:	0000000000000004	**1**	**OBJECT**	**GLOBAL**	**DEFAULT**	**3**	**are**
15:	0000000000000004	**4**	**OBJECT**	**GLOBAL**	**DEFAULT**	**COM**	**belong**
16:	0000000000000000	**4**	**OBJECT**	**GLOBAL**	**DEFAULT**	**4**	**to**
17:	0000000000000008	**37**	**OBJECT**	**GLOBAL**	**DEFAULT**	**COM**	**us**

Hints:

- Are you seeing `puts` instead of `printf`? Check the man pages for what the difference is, and make sure that the compiler can't optimize away your call to printf. Remember, requirement #1 **only** mentions the **first** line of output.
- Function lengths are very difficult to reproduce - note that for every FUNC, you do not have to duplicate the length (the length is the number of bytes of assembly code chosen by the compiler to execute the body of each function).

Template

There is no skeleton code for this assignment, only a Makefile.

Turn-in instructions

When you have completed your assignment and wish to have it graded, you must tag the commit you want graded with the tag `hw1-submission`.

To make sure your submission is complete, try the following **in a temporary directory**, i.e. create and change into a temporary directory under the `/tmp` filesystem:

```
mkdir /tmp/hw1-temp
cd /tmp/hw1-temp
git clone git@git.uicbits.net:cs361-s16/MYNETID.git
cd MYNETID
git checkout hw1-submission
cd hw1
make test
```

Grading

Grading will be done automatically using a script, so make sure that your `readelf` output is identical where it matters. The grading script will only consider the bolded parts.

Due Date

This assignment is due Friday, January 22, at 8 AM. See the syllabus (syllabus.html) for the late turnin policy. This assignment is worth just as much as every other homework, so getting as much credit on it as possible is important (don't turn it in late!).

helpful documents

Think Like (a) Git (http://think-like-a-git.net/)
A successful Git branching model (http://nvie.com/posts/a-successful-git-branching-model/)
