p1: A Fortune to Get You Started

Due Feb 7 by 10pm **Points** 60 **Submitting** a file upload
Available until Feb 7 at 10:33pm

This assignment was locked Feb 7 at 10:33pm.

OVERVIEW LINUX EDITING BUILDING EXECUTING SUBMITTING

- This project must be done individually.
- All work for this project is to be done on the CS Department's instructional Linux machines (https://csl.cs.wisc.edu/services/instructional-facilities) . You're encouraged to remotely login using the ssh command in the terminal app on Macs, or on Windows using an ssh client such as MobaXterm.
- All projects in this course are graded on CS Department's instructional Linux machines (https://csl.cs.wisc.edu/services/instructional-facilities) . To receive credit, make sure your code runs as you expect on these machines.

OVERVIEW

The Caesar cipher is a simple way to make secret messages. To encode a message, each character is shifted forward or backward in the alphabet. For the message (aka plaintext):

```
attack at dawn!
```

a forward shift of 3 for each character results in the secret message (aka ciphertext):

```
dwwdfn dw gdzq!
```

To decode this secret message, we simply shift back every character by 3. Note that punctuation is not encoded for this simple approach and we've restricted the characters to lower case.

For this assignment, you'll use a C program we've written to decode a fortune that was encoded specifically for you. The exact shift we've used is based on a calculation using your CS login username. You'll need to get your fortune, get a copy of our source file, add a file header comment to that source file, determine your fortune, and submit it along with some files you'll produced during the build process.

Goals:

- Learn some basic Linux commands.
- Practice using a text editor.
- Get files from the CS servers.
- Become familiar with the build process for C code.

- Learn how to submit your assignments to Canvas.

LINUX Commands

We'll start by using some Linux commands to set up your workspace on the CS Linux machines and get the source file for this project (for more info see **Linux Command Reference**).

1. Remotely connect to a CS Linux machine (or if you're sitting at a CS lab computer then open the terminal: `Ctrl+Alt+T` in Ubuntu).
2. Make sure you're in your home directory where your files are located. Change to your home directory using `cd ~` (change directory). Recall that you can find where you're at with `pwd` (print working directory).
3. List all the files and directories under your home directory. See if you can remember the command to do this (if not, look in the reference linked above). Find a directory named `private` among the files and directories listed. Notice there is a directory named `Public` and another named `public`. These are two different directories. Unlike in Windows, filenames are case sensitive.
4. Change to your `private` directory.
5. Make a new directory named `cs354` inside your `private` directory. Find the appropriate Linux command in the reference linked above.
6. Change to `cs354` as your working directory (aka current directory).
7. Make a new directory named `p1` inside `cs354`, which you'll use for this project.
8. Change your working directory to `p1`.
9. Copy the file `decode.c` from the location:
   ```
   /p/course/cs354-skrentny/public/code/p1/decode.c
   ```
   to your `p1` directory. Find the appropriate Linux command to do this in the reference linked above. This is the source file you'll be using to build and run your first program, which we'll explain below.

EDITING C Source Files

We strongly encourage you to learn vim, which is a popular text editor used in the Linux OS environment. Run `vimtutor` on a CS Linux machine to learn the basics. Here's a link to a useful **cheatsheet (https://www.maketecheasier.com/vim-keyboard-shortcuts-cheatsheet/)** of vim keyboard shortcuts. Other common text editors are: gedit, emacs and nano. If you're transitioning from Windows to Linux gedit is the easiest editor to use when you're working at a CS lab computer. Using gedit remotely requires additional configuration of your machine that you'll need to figure out on your own (hint: x forwarding).

We've provided the code for this assignment, but you must edit that source file and add your file header comment as specified in this **Program Commenting Guide**.

BUILDING C Executables

Going from a C source file to an executable we typically call compiling. Actually, compiling is just one phase of the process we call "building", which is described below.

1. Preprocessing Phase

Preprocessing is the first phase of the build process, which prepares a C source file for compiling. We can just preprocess the **`decode.c`** source file and store its result in a file named **`decode.i`** using the command:

```
gcc -E decode.c -Wall -m32 -std=gnu99 -o decode.i
```

1. Learn more about gcc's "-E" option by looking at the manual page for gcc. Type **`man gcc`** at the Linux prompt.
2. Recall the **`-Wall`** option is recommended to be used so that all of the warnings are displayed during the build process.
3. Recall the **`-m32`** option is used to generate code for a 32-bit environment, which we'll be using to study assembly language.

In preprocessor stage the lines in **`decode.c`** beginning with a **`#`**, called preprocessor directives, which are included header files and defined macros, are expanded and merged within the source file to produce an updated source file. Open the file **`decode.i`** and you'll see that those lines have been replaced with intermediate code. You don't need to understand the intermediate code, just know that the preprocessing step substitutes preprocessor directives like **`#include <stdio.h>`** so that the compiler knows the definitions of library functions like printf that are defined elsewhere.

2. Compilation Phase

The next phase of the build process is the compilation of the preprocessed source code. Compiling translates this source to assembly language for a specific processor. Let's stop after compilation to see the generated the assembly file. The option to let gcc know it should stop the build process after compilation can be discovered in the man page under "compilation proper".

Run one of the following commands at the command prompt:

```
gcc <option> decode.c -Wall -m32 -std=gnu99
```

OR

```
gcc <option> decode.i -Wall -m32 -std=gnu99
```

Find the correct <option> to stop the build process after compilation by taking a look at gcc's man page.

Next, open and inspect the generated **`decode.s`** file in a text editor.

Don't worry about understanding the contents of this file right now; we'll learn more about it after the midterm. For now, just get a feel of how assembly language code looks. By the end of this semester, you'll be able to understand much more of this file.

3. Assembling Phase

Computers can only understand machine-level code (in binary), which requires an assembler to convert the assembly code into machine code that the computer can execute.

Let's now stop the build process after the assembling phase by entering one of the following commands to create the object file **`decode.o`**:

```
gcc -c decode.c -Wall -m32 -std=gnu99
```

OR

```
gcc -c decode.s -Wall -m32 -std=gnu99
```

Note that the input to gcc can either be the C source file (**`decode.c`**) or the Assembly Code file (**`decode.s`**) that was generated from the previous step. If you use the source file then all the prior phases will be repeated.

Try opening the **`decode.o`** file in your text editor and see what happens.

You can view the contents of an object file (**`decode.o`**) using a tool named objdump (object dump) as shown below:

```
objdump -d decode.o
```

objdump is a disassembler that converts the machine code to assembly code, which is the inverse operation of the assembler. Understand the use of the command **`objdump`** and the meaning of the option "-d" by looking at its man page or by typing **`objdump --help`** at the Linux prompt.

Next, save the disassembled output of the object file **`decode.o`** in a file named **`objfile_contents.txt`**. An easy way to do this is to redirect the output of the command to a file as follows:

```
objdump -d decode.o > objfile_contents.txt
```

You could also do this by copying what objdump displays on the terminal and paste it in a text file, but that is more error prone.

4. Linking Phase

The last phase of the build process combines your object file with other object files such as those in the standard C library to create the executable file. Execute one of the following commands to create the executable file.

```
gcc decode.c -Wall -m32 -std=gnu99 -o decode
```

OR

```
gcc decode.o -Wall -m32 -std=gnu99 -o decode
```

Use objdump to view the disassembled contents of the executable file, which is also a binary file, as we did for the object file **`decode.o`**.

Redirect the disassembled output that you got to a file named **`exefile_contents.txt`**. This file should be much larger than the disassembled output of the **`decode.o`** file since it's an executable file, which has information combined from **`decode.o`** and library functions like printf.

What's Typically Used

We've now seen the steps of the build process and generated intermediate files for each. You'll find that the two files that you'll most often use are:

1. C Source File (**`decode.c`**)
2. Executable File (**`decode`**)

In most cases, you would compile the source file directly to the executable file using the command in this form:

```
gcc <source-file-name> -Wall -m32 -std=gnu99 -o <executable-name>
```

EXECUTING C Programs

Next we'll run the executable file to decode your encoded fortune.

First, copy the file **`ciphertext.txt`** from the location:

```
/p/course/cs354-skrentny/public/students/<your-cs-login>/p1/
```

to your **`p1`** directory. This file contains the fortune encoded using your CS login. The **`ciphertext.txt`** and the executable **`decode`** must both be present in the same directory at this point.

Run the **`decode`** executable file and you will be prompted for your CS login. Correctly enter your CS login to get your decoded fortune, which should be a valid phrase in English. See the sample run shown below.

```
[skrentny@jimbo] (11)$ ./decode
Ciphertext:
c eqpenwukqp ku ukorna vjg rnceg yjgtg aqw iqv vktgf qh vjkpmkpi.
Enter your CS login: skrentny
Plaintext:
a conclusion is simply the place where you got tired of thinking.
```

Create a new file named **`fortune.txt`** and save your decoded fortune output string as the first and only line of this file.

Make sure that the contents of **`fortune.txt`** are exactly the same as the decoded string and nothing else. We'll use a script to automatically match this string with our answer key to grade your submission. Include all the punctuation marks that are present in the plaintext output (including the trailing period, exclamation or question mark). Copy the output from the terminal instead of retyping, to avoid trivial spelling mistakes. In the above sample run, the **`fortune.txt`** file should have only one line that is "a conclusion is simply the place where you got tired of thinking." without the quotes.

Before you finish, take a look at the code in **`decode.c`** and make sense of the major steps this decode program. Understanding this can help you with your C programming and upcoming assignments.

SUBMITTING Your Work

Submit the following files under Project p1 in Assignments on Canvas before the deadline:

1. **`decode.c`** (the source file with your file header comment added)
2. **`decode.i`** (the intermediate file after preprocessing)
3. **`decode.s`** (the assembly file after compilation proper)
4. **`decode.o`** (the object file after assembling)
5. **`decode`** (the executable file after linking with standard libraries)
6. **`objfile_contents.txt`** (disassembled output of decode.o object file)
7. **`exefile_contents.txt`** (disassembled output of decode executable file)
8. **`fortune.txt`** (decoded plaintext)

It is your responsibility to ensure your submission is complete with the correct file names having the correct contents. The following points will seem obvious to most, but we've found we must explicitly state them otherwise some students will request special treatment for their carelessness:

- **You will only receive credit for the files that you submit.** You will not receive credit for files that you do not submit. Forgetting to submit, not submitting all the listed files, or submitting executable files or other wrong files will result in you losing credit for the assignment.
- **Do not zip, compress, submit your files in a folder, or submit each file individually.** Submit only the files as listed as a single Canvas submission.
- **Make sure your file names exactly match those listed.** If you resubmit your work, Canvas will modify the file names by appending a hyphen and a number (e.g., fortune-1.txt) and these Canvas modified names are accepted for grading.

Repeated Submission: You may resubmit your work repeatedly until the deadline has passed. **We strongly encourage you to use Canvas as a place to store a backup copy of your work.** If you resubmit, you must resubmit all of your work rather than updating just some of the files.

Project p1

Criteria	Ratings				Pts
Submission contains 'decode.i'; 'decode.i' contains 'extern int printf'	**5.0 pts Full Marks**	**3.0 pts Any of these two is incorrect.**	**0.0 pts No Marks**		5.0 pts
Submission contains 'decode.s'; 'decode.s' contains 'main'	**5.0 pts Full Marks**	**3.0 pts Any of these two is incorrect.**	**0.0 pts No Marks**		5.0 pts
Submission contains 'decode.o'; 'decode.o' contains 'relocatable'	**5.0 pts Full Marks**	**3.0 pts Any of these two is incorrect.**	**0.0 pts No Marks**		5.0 pts
Submission contains 'decode'; 'decode' contains 'executable'	**5.0 pts Full Marks**	**3.0 pts Any of these two is incorrect.**	**0.0 pts No Marks**		5.0 pts
Submission contains 'objfile_contents.txt'; 'objfile_contents' contains 'main>:' and doesn't contain '_start>:'	**5.0 pts Full Marks**	**3.0 pts Any of these two is incorrect.**	**0.0 pts No Marks**		5.0 pts
Submission contains 'exefile_contents.txt'; 'exefile_contents.txt' contains '_start>:'	**5.0 pts Full Marks**	**3.0 pts Any of these two is incorrect.**	**0.0 pts No Marks**		5.0 pts
Submission contains 'fortune.txt'; Get correct contents of 'fortune.txt'	**25.0 pts Full Marks**	**22.0 pts a few characters incorrect**	**14.0 pts many characters incorrect**	**0.0 pts Wrong or no submission of 'fortune.txt'**	25.0 pts
Submission contains decode.c, decode.c has header comment.	**5.0 pts Full Marks**	**3.0 pts Any of these two is incorrect**	**0.0 pts No Marks**		5.0 pts

Criteria	Ratings	Pts
		Total Points: 60.0