Problem 2-1: Reverse Engineer

This is CS50. Harvard University. Fall 2014.

Table of Contents

Questions? Feel free to head to CS50 on Reddit[1], CS50 on StackExchange[2], or the CS50 Facebook group[3].

Objectives

- Dig deeper into the compilation process.
- Understand how a machine converts source code into object code.
- Get as close as humans today typically get to reading object code.
- Appreciate how few things a computer can actually do (and how we can only combine them in interesting ways).
- Reverse engineer a simple piece of software.

Recommended Reading

- Pages 11 – 14 and 39 of http://www.howstuffworks.com/c.htm.

1 https://www.reddit.com/r/cs50
2 http://cs50.stackexchange.com
3 https://www.facebook.com/groups/cs50

- Chapters 6, 7, 10, 17, 19, 21, 22, 30, and 32 of *Absolute Beginner's Guide to C*.
- Chapters 7, 8, and 10 of *Programming in C*.

Academic Honesty

This course's philosophy on academic honesty is best stated as "be reasonable." The course recognizes that interactions with classmates and others can facilitate mastery of the course's material. However, there remains a line between enlisting the help of another and submitting the work of another. This policy characterizes both sides of that line.

The essence of all work that you submit to this course must be your own. Collaboration on problems is not permitted (unless explicitly stated otherwise) except to the extent that you may ask classmates and others for help so long as that help does not reduce to another doing your work for you. Generally speaking, when asking for help, you may show your code or writing to others, but you may not view theirs, so long as you and they respect this policy's other constraints. Collaboration on quizzes and tests is not permitted at all. Collaboration on the final project is permitted to the extent prescribed by its specification.

Below are rules of thumb that (inexhaustively) characterize acts that the course considers reasonable and not reasonable. If in doubt as to whether some act is reasonable, do not commit it until you solicit and receive approval in writing from your instructor. If a violation of this policy is suspected and confirmed, your instructor reserves the right to impose local sanctions on top of any disciplinary outcome that may include an unsatisfactory or failing grade for work submitted or for the course itself.

Reasonable

- Communicating with classmates about problems in English (or some other spoken language).
- Discussing the course's material with others in order to understand it better.
- Helping a classmate identify a bug in his or her code, such as by viewing, compiling, or running his or her code, even on your own computer.
- Incorporating snippets of code that you find online or elsewhere into your own code, provided that those snippets are not themselves solutions to assigned problems and that you cite the snippets' origins.

- Reviewing past years' quizzes, tests, and solutions thereto.
- Sending or showing code that you've written to someone, possibly a classmate, so that he or she might help you identify and fix a bug.
- Sharing snippets of your own solutions to problems online so that others might help you identify and fix a bug or other issue.
- Turning to the web or elsewhere for instruction beyond the course's own, for references, and for solutions to technical difficulties, but not for outright solutions to problems or your own final project.
- Whiteboarding solutions to problems with others using diagrams or pseudocode but not actual code.
- Working with (and even paying) a tutor to help you with the course, provided the tutor does not do your work for you.

Not Reasonable

- Accessing a solution to some problem prior to (re-)submitting your own.
- Asking a classmate to see his or her solution to a problem before (re-)submitting your own.
- Decompiling, deobfuscating, or disassembling the staff's solutions to problems.
- Failing to cite (as with comments) the origins of code, writing, or techniques that you discover outside of the course's own lessons and integrate into your own work, even while respecting this policy's other constraints.
- Giving or showing to a classmate a solution to a problem when it is he or she, and not you, who is struggling to solve it.
- Looking at another individual's work during a quiz or test.
- Paying or offering to pay an individual for work that you may submit as (part of) your own.
- Providing or making available solutions to problems to individuals who might take this course in the future.
- Searching for, soliciting, or viewing a quiz's questions or answers prior to taking the quiz.

- Searching for or soliciting outright solutions to problems online or elsewhere.
- Splitting a problem's workload with another individual and combining your work (unless explicitly authorized by the problem itself).
- Submitting (after possibly modifying) the work of another individual beyond allowed snippets.
- Submitting the same or similar work to this course that you have submitted or will submit to another.
- Using resources during a quiz beyond those explicitly allowed in the quiz's instructions.
- Viewing another's solution to a problem and basing your own solution on it.

Assessment

Your work on this problem set will be evaluated along just one axis.

Scope

To what extent does your implementation attempt to replicate the features of our mystery program?

To obtain a passing grade in this course, all students must ordinarily submit all assigned problems unless granted an exception in writing by the instructor.

The Pre-Process

Before we begin, know that this problem involves relatively little coding. It's not really a writing problem, and there is coding involved, but it's mostly incidental to the primary goal. That goal is to understand what happens at a lower level on your system. How does a computer understand and translate the C code you write into the zeroes and ones of binary? In this problem, you're going to try to find out, by reading exactly those zeroes and ones[4] and trying to recreate the C program that generated those zeroes and ones.

To get started, have a look at Rob's short on compilers.

https://www.youtube.com/watch?v=CSZLNYF4Klo

[4] Well, almost. Rather, their closest human equivalent.

Notice how in that video Rob not only compiles some simple programs, but actually interrupts the compilation process to show what is happening at the various steps. To compile a program, a compiler goes through a set of four steps.

- Pre-processing
- Compiling
- Assembling
- Linking

Rob also talks about `clang`, which is the compiler that is used by default in CS50 IDE, but other compilers for C exist. One other popularly-used compiler is called `gcc`, and indeed for reasons that don't bear going into right now, the file that you will be reverse engineering in this problem was compiled by `gcc` and interrupted at the second step above, compiling. That is, we typed:

```
gcc -S file.c
```

to obtain the output you'll see at the bottom of this specification, which came to live in a file called `file.s`.

Assemble Some Knowledge

Though not produced by us, there's actually a really fantastic YouTube video of someone going through the process of comparing C code to assembly code (sometimes called "machine code") that will likely be quite helpful as you start to think about this problem.

https://www.youtube.com/watch?v=yOyaJXpAYZQ
Along the same lines is this webpage[5] which also shows a little bit of translating between source code and machine code.

The other thing you'll need to do is read up on what the various assembly instructions mean. The `gcc` compiler takes C code and translates it to machine code using the IA-32 instruction set[6] (of which the linked document shows a majority of the useful instructions,

[5] http://cs.lmu.edu/~ray/notes/c2asmexamples/
[6] http://web.stanford.edu/class/cs107/IA32_Cheat_Sheet.pdf

but not the full set). As it turns out, there's actually very few things a computer can realistically do! They can perform some extremely basic math, jump to other points in memory, and flip bits around. That's… pretty much it. All the amazing things we can do in C (and we've only **just** scratched the surface of that!) eventually boil down to just those.

Link it Together

Here's the assembly code you'll be trying to replicate.

```
 .LC0:
  .string "%i\n"
  .text
  .globl main
  .type main, @function
 main:
 .LFB0:
  .cfi_startproc
  pushq %rbp
  .cfi_def_cfa_offset 16
  .cfi_offset 6, -16
  movq %rsp, %rbp
  .cfi_def_cfa_register 6
  subq $16, %rsp
  movl $1, -4(%rbp)
  jmp .L2
 .L3:
  movl -4(%rbp), %eax
  movl %eax, %esi
  movl $.LC0, %edi
  movl $0, %eax
  call printf
  addl $1, -4(%rbp)
 .L2:
  cmpl $50, -4(%rbp)
  jle .L3
  leave
  .cfi_def_cfa 7, 8
  ret
  .cfi_endproc
```

Again, your goal here is to create C code that, when partially compiled with:

```
gcc -S file.c
```

results in assembly code that is effectively identical. (Because every machine is slightly different, the numbers and names and labels may differ, but for the most part the behavior should be quite similar.) We've also stripped out a few bits of information from our actual assembly code (in particular some stuff at the top and bottom of what was actually output by `gcc`) because it would much more obviously give away what the program does.

The program is not terribly complex, by the way. Including curly braces, a `#include`, and a completely blank line, it is possible to write this program in C in just ten lines or less. It doesn't do anything particularly amazing.

In the interest of full disclosure, you should know that it **is** possible to transform the above assembly code into a typical executable which you could run. We're not going to share the steps for how to do that here, but if you know the right questions to ask of Google, it won't take you too long to figure out the answer. If you do so, what this program actually does will become incredibly obvious and you'll likely be able to replicate it very quickly.

But this is a Hacker edition, and we expect you to hack. Sometimes that means being clever and finding a workaround, but since we've disclosed that such a path exists and we've intimated how to walk that path, you should try to solve this problem the hard way. This is an opportunity to learn about assembly code in a way that no other assignment in this course will permit, and we hope you'll take advantage of it. But that choice is yours. This problem is really not so much about getting the right answer, but about the process you use to arrive at your answer.

There's no `check50` or staff solution for this problem. After all, where would be the fun in that?!

This was Problem 2-1.