CSC209H Worksheet: Inspecting Executables

You know already that programs are simply binary files whose contents can be interpreted as a set of machine instructions. Now that you know how to read the contents of binary files, you can start writing programs to actually inspect these executable files (and even modify them). One such Unix program `objdump`; When used on a compiled "Hello World" program called `hello` on teach.cs it gave the following output:

```
$ objdump -s -j .rodata hello

hello:     file format elf64-x86-64

Contents of section .rodata:
 2000 01000200 48656c6c 6f20576f 726c6421  ....Hello World!
 2010 00                                   .
```

Passing these flags to `objdump` enables us to see the contents of the read-only section of the executable, in which the string literals are stored (among other things). For this worksheet, we will write our own programs to inspect executable files.

1. Your first task is to write a program `literals.c` which takes three command-line arguments:

   - The address of the first byte in the .rodata section, in hexadecimal (with a leading `0x`).
   - The size (number of bytes) of the .rodata section, in decimal.
   - The name of an executable file.

   This program should print all of the string literals stored in the .rodata section, one per line. For simplicity, it can also print out other data in the .rodata section. You will need to use *objdump* first to learn the correct values to pass to your program as the command-line arguments. Here is an example of calling it on the same hello executable and the corresponding output.

   ```
   $ ./literals 0x2000 17 hello


   Hello, world!
   ```

   Or better, you can pipe the output to another useful tool `od` which allows you to to inspect the characters:

   ```
   $ ./literals 0x2000 17 hello | od --format xCc
   0000000  01  0a  02  0a  48  65  6c  6c  6f  20  57  6f  72  6c  64  21
           001  \n 002  \n   H   e   l   l   o       W   o   r   l   d   !
   0000020  0a
            \n
   0000021
   ```

   - Use `strtol` with the base as `16` to parse hexademical numbers, including the leading `0x`.
   - Read in the entire .rodata section first, and then think about how to print out individual strings on new lines. Remember that there could be many null-terminated strings in the .rodata section.
   - If you have time (because you are doing this after class) write the complete program on your own. If you are in lecture, download starter code from Quercus.

CSC209H Worksheet: Inspecting Executables

2. Of course, `literals` by itself is not very useful. The real question is how to determine the location and size of the rodata section for a given executable. It should not be surprising that this data is encoded in the executable as well. Here is the relevant information **for basic programs on teach.cs (64-bit machines) compiled by gcc**:

   - The eight bytes starting at address `0x28` contain the starting address of all the section headers. You will read this in as a `long` (instead of an `int`, which is only four bytes).
   - Each section header is 64 bytes in length. The section header for the .rodata section is the 19th header (so add an offset for 18 headers). Use this to compute the starting address of the .rodata section header.
   - Each section header has integers stored in addresses 24-31 and 32-39 bytes after its starting address. These two integers store the address and size of the section data, respectively. You'll store these as `long` variables as well.

   Using this information and the starter code, complete the program `rodata.c` which takes an executable filename as its only command-line argument, and prints the address of the .rodata section (in hex) and the size of the section (in decimal).

   Bonus: modify your program from Part 1 so that you can connect the two programs with a pipe.

```
$ ./rodata hello
0x2000 17
```