

## 15-122: Principles of Imperative Computation

---

### Lab D: passwordLab

James Wu, Rob Simmons

**Collaboration:** In lab, we encourage collaboration and discussion as you work through the problems. These activities, like recitation, are meant to get you to review what we've learned, look at problems from a different perspective and allow you to ask questions about topics you don't understand. We encourage discussing problems with your neighbors as you work through this lab!

**Setup:** Copy the lab code from our public directory to your private directory:

```
% cd private/15122
% cp -R /afs/andrew/course/15/122/misc/lab-dre .
% cd lab-dre
```

**Grading:** Finish tasks (1.a), (1.b), and (1.c) for 2 points, and additionally complete (1.d) and (1.e) for 3 points.

### Dr. Evil's passwords

Genius supervillian Dr. Evil is on the loose! Known for a series devilishly tricky yet completely vulnerable assembly bombs, Dr. Evil has left a trail of destruction across Carnegie Mellon's undergraduate computer science curriculum. Authorities have been unable to track the whereabouts of this mastermind, but we have new intelligence on Dr. Evil's Super Secret Evil Plan™ to investigate.

You have been hired as an agent to crack the code of Dr. Evil's Super Secret Evil Plan. It seems that she left her secret plans in a password protected c0 binary file, accessible to you on the cluster computers by typing `evilplan`. She also accidentally left her COVM bytecode in a public folder! She seems to have deleted most of the helpful comments, though, so we'll need help figuring out the passwords by hand. We were also able to acquire the main function's source code in `password-main.c0`, but it relies on functions that only appear in the bytecode file `password.bc0`.

You'll need to read through `password.bc0` to figure out some of the function calls. (Namely, the function calls `password1()`, `password2()`, `password3()`, etc).

Each of the password functions either takes in a password as input, and returns a boolean, or simply returns the password as an integer. Some passwords are numbers while others are strings. For the first four passwords, the user's input is passed to `parse_int`, but for the last password, it the string is passed directly to the function. We've filled in the bytecode file with all the intelligence we have, so you'll have to figure out the rest.

We highly suggest you keep the COVM writeup handy as you complete this lab.

To check if you're correct, just run the password binary file, and type in the passwords you think are correct:

```
% evilplan
Welcome to Dr. Evil's Super Secret Evil Plan Terminal
This terminal should only be run by Dr. Evil to read the
Super Secret Evil Plan.
If you are anyone else, get OUT.
Password1:
```

- (1.a) Dr. Evil's first password function seems pretty simple. It seems to return an integer. What is it?
- (1.b) Dr. Evil's second password is a bit more complicated. It uses `vload` and `vstore` to store some local variables. Figure out what integer `password2` returns!
- (1.c) Dr. Evil's third password is definitely more complicated. It uses `ildc` to load integers from the integer pool. What's going on there?
- (1.d) Dr. Evil's fourth password has a loop! The function jumps around, doing something to an integer input. What's the password?
- (1.e) Dr. Evil's fifth and final password calls a helper function, `func5`. Figure out what it's doing, and crack the last password! The ASCII table to the right, which includes both integer and hex values, may come in handy.
- (1.f) For the most clever of agents, Dr. Evil seems to have left a hidden 6th password. She didn't activate it in the source code file, which means it must have been so complicated even she didn't want to deal with it! Figure it out through the bytecode, and tell your TA if you think you got it.

32	20	̀	64	40	@	96	60	‘
33	21	!	65	41	A	97	61	a
34	22	"	66	42	B	98	62	b
35	23	#	67	43	C	99	63	c
36	24	\$	68	44	D	100	64	d
37	25	%	69	45	E	101	65	e
38	26	&	70	46	F	102	66	f
39	27	'	71	47	G	103	67	g
40	28	(	72	48	H	104	68	h
41	29	)	73	49	I	105	69	i
42	2A	*	74	4A	J	106	6A	j
43	2B	+	75	4B	K	107	6B	k
44	2C	,	76	4C	L	108	6C	l
45	2D	-	77	4D	M	109	6D	m
46	2E	.	78	4E	N	110	6E	n
47	2F	/	79	4F	O	111	6F	o
48	30	0	80	50	P	112	70	p
49	31	1	81	51	Q	113	71	q
50	32	2	82	52	R	114	72	r
51	33	3	83	53	S	115	73	s
52	34	4	84	54	T	116	74	t
53	35	5	85	55	U	117	75	u
54	36	6	86	56	V	118	76	v
55	37	7	87	57	W	119	77	w
56	38	8	88	58	X	120	78	x
57	39	9	89	59	Y	121	79	y
58	3A	:	90	5A	Z	122	7A	z
59	3B	;	91	5B	[	123	7B	{
60	3C	<	92	5C	\	124	7C	
61	3D	=	93	5D	]	125	7D	}
62	3E	>	94	5E	^	126	7E	~
63	3F	?	95	5F	_			