15-122: Principles of Imperative Computation   Spring 2020

Lab 12: Once you C1 you C them All   Tuesday April 7th

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/lab12 .
% cd lab12
```

Grading: Attendance for full credit. Complete task 4 for extra credit. NOTE: This lab is NOT autograded. Submit your code for task 4 in Autolab to collect the points.

Storing and using strings in C

Load the file `ex1.c` into a text editor. Read through the file and write down what you think the output will be before you run the program:

`word` **`string`**`:` ______________________________

`word ASCII values:` ________ ________ ________ ________

Once you have done this, compile with the following command (all on one line):

```
% gcc -Wall -Wextra -Werror -Wshadow -std=c99
      -pedantic -g ex1.c
```

(1.a) Which parts differed from what you expected?

(1.b) Change the '`\0`' character in the array to something else, like '`d`'. Predict how this will change the answer, and then compile and see if you're right.

(1.c) Run the modified code under `valgrind`, and read through its output to see which lines in `ex1.c` are given as part of the output.

Partial ASCII Table

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	_			

Arrays of strings

Load the file `ex2.c` into a text editor. Read through the file and write down what you think the output will be *before* you run the program.

Once you have done this, compile and run the program:

```
% gcc -Wall -Wextra -Werror -Wshadow -std=c99 -pedantic -g ex2.c
% ./a.out
% valgrind ./a.out
```

(2.a) We never free any memory in this program, yet valgrind reports no memory leaks. Why? Where are the strings stored? Where is the memory for the array stored?

(2.b) What do you think will happen if we change `num_states` to 7 without changing any other part of the program? Make this change, and explain the output you see in Valgrind.

Discuss the answer to (2.a) with a TA, and explain whether you would be able to use the output from `valgrind` to identify the bug you introduced in (2.b), to get checked in for this lab.

C string libraries

The header file **`string.h`** outlines a number of string functions that can be used (often incorrectly) in C programs. They include:

```
char *strcpy(char *dest, const char *src)
char *strncpy(char *dest, const char *src, size_t n)
size_t strlen(const char *str)
```

Read about how these functions work here:

`http://en.wikipedia.org/wiki/C_string_handling#Functions`

These functions assume that the pointers point to a `NUL`-terminated string (i.e., a string that ends with the character '`\0`', which has ASCII value 0).

(3.a) Load the file `ex3.c` into a text editor. Read through the file and decide what you think the output will be before you run the program.

```
% gcc -Wall -Wextra -Werror -Wshadow -std=c99 -pedantic -g ex3.c
% ./a.out
```

(3.b) Did the results surprise you? Can you explain the difference in behavior of the two functions?

Programming with C strings

(4.a) Write a C function in a new file `ex4.c` that reverses a string and returns a pointer a new string with the result. The function should have the following prototype:

```
char *reverse(char *s);
```

(4.b) Write a main function to test your function on a number of strings. Include only those header files that are necessary to compile your code. If you allocate memory, use `calloc` and be sure to free what you allocate.

Compile and run your code with these commands:

```
% gcc -Wall -Wextra -Werror -Wshadow -std=c99 -pedantic -g lib/*.c ex4.c
% ./a.out
```

4pt