ECE264 Fall 2014
Exam 1, February 11, 2014

In signing this statement, I hereby certify that the work on this exam is my own and that I have not copied the work of any other student while completing it. I understand that, if I fail to honor this agreement, I will receive a score of ZERO for this exam and will be subject to possible disciplinary action.

Signature:

You must sign here. Otherwise you will receive a **2-point** *penalty.*

Read the questions carefully.
Some questions have conditions and restrictions.

This is an *open-book, open-note* exam. You may use any book, notes, or program printouts. No personal electronic device is allowed. You may not borrow books from other students.

No learning objectives are tested in this exam.

Contents

1 Strings (5 points)

Strings in C are defined as arrays of characters with a terminating character ('\0'). Please fill in the missing code in the string functions below. You may **not** use existing string functions such as `strcpy()`, `strlen()`, `tolower()`, `toupper()`, or similar. (You may use `malloc()`, however.)
Please see the ASCII table attached to the exam if you need to look up ASCII encodings.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Count the number of characters in a string.
5   *
6   * This function should return the number of characters in a string,
7   * NOT including the terminating null character ('\0'). An empty
8   * string is represented by 'char *s = ""', and has length 0.
9   */
10 size_t my_strlen(const char *str)
11 {
12     /* FILL IN CODE HERE */
13
14
15
16
17
18
19
20
21
22
23
24
25
26 }
27
28 /* Perform a string duplication.
29  * This function should allocate the necessary space (remember the
30  * null character!), copy the contents of the source string into the
31  * new memory, and return a pointer to the new buffer.
32  *
33  * NOTE: You may use my_strlen() here, but no external string
34  * functions in string.h.
35  */
```

```
36 char *my_strdup(const char *str)
37 {
38     /* FILL IN CODE HERE */
39
40
41
42
43
44
45
46
47
48
49
50 }
51
52 /* Convert this string to lowercase. All uppercase characters should
53  * be changed to their lowercase equivalent. Does not apply to numbers
54  * or symbols (including whitespace).
55  */
56 void my_strtolower(char *str)
57 {
58     /* FILL IN CODE HERE */
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74 }
```

2 Pointers (5 points)

Please see the code below and fill in the missing numbers in the planned output after the source code listing. Note that `%p` in `printf()` prints a pointer address; please use the addresses given after the `---` divider when determining how pointer addresses are swapped.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 void swap1(int a, int b)
 5 {
 6     int tmp = a;
 7     a = b;
 8     b = tmp;
 9     printf("a = %d, b = %d\n", a, b);
10 }
11
12 void swap2(int *a, int *b)
13 {
14     int tmp = *a;
15     *a = *b;
16     *b = tmp;
17     printf("*a = %d, *b = %d\n", *a, *b);
18 }
19
20 void swap3(int **a, int **b)
21 {
22     int *tmp = *a;
23     *a = *b;
24     *b = tmp;
25     printf("**a = %d, **b = %d\n", **a, **b);
26 }
27
28 int main(int argc, char **argv)
29 {
30     int x = 42, y = 9;
31     printf("x = %d, y = %d\n", x, y);
32     swap1(x, y);
33     printf("x = %d, y = %d\n", x, y);
34     x = 42, y = 9;
35     swap2(&x, &y);
36     printf("x = %d, y = %d\n", x, y);
37
```

```
38      printf("---\n");
39      int z = 77, w = 33;
40      int *p = &z;
41      int *q = &w;
42
43      printf("*p = %d, *q = %d, p = %p, q = %p\n", *p, *q, p, q);
44      swap2(p, q);
45
46      z = 77, w = 33;
47      printf("*p = %d, *q = %d, p = %p, q = %p\n", *p, *q, p, q);
48      swap3(&p, &q);
49      printf("*p = %d, *q = %d, p = %p, q = %p\n", *p, *q, p, q);
50
51      printf("z = %d, w = %d\n", z, w);
52
53      return EXIT_SUCCESS;
54  }
```

Fill in the missing numbers in the program's output below:

```
x = 42, y = 9
a = 9, b = 42

x = ___, y = ___

*a = ___, *b = ___

x = ___, y = ___

---

*p = 77, *q = 33, p = 0x22aac4, q = 0x22aac0
*a = 33, *b = 77

*p = ___, *q = ___, p = 0x____________, q = 0x____________

**a = ___, **b = ___

*p = ___, *q = ___, p = 0x____________, q = 0x____________

z = ___, w = ___
```

3 Quicksort (5 points)

Quicksort is an efficient sorting algorithm. The C standard library contains an implementation of quicksort in `stdlib.h` called `qsort()`. Please implement the missing code below for a program that sorts integers given on the command line.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Comparison function. See the manual page on qsort for details. */
5  int intComp(const void *a, const void *b)
6  {
7      /* FILL IN CODE BELOW */
8
9
10
11
12
13 }
14
15 /* Sort an array of integers of length len in ascending order (smaller
16  * numbers first).  You MUST use the built-in qsort() function from
17  * the C standard library (stdlib.h).  You will have to define a
18  * comparison function; he manual page for qsort will tell you how.
19  */
20 void sortNumbers(int *numbers, int len)
21 {
22     /* FILL IN CODE BELOW */
23
24
25
26
27
28
29
30 }
31
32 int main(int argc, char **argv)
33 {
34     int i;
35     if (argc == 1) return EXIT_FAILURE;
36     int *numbers = malloc((argc - 1) * sizeof(int));
37     for (i = 1; i < argc; i++) {
```

```
38          numbers[i - 1] = atoi(argv[i]);
39      }
40      sortNumbers(numbers, argc - 1);
41      for (i = 0; i < argc - 1; i++) {
42          printf("%d ", numbers[i]);
43      }
44      printf("\n");
45      free(numbers);
46      return EXIT_SUCCESS;
47  }
```

Here is an example usage:

```
$ ./q3 3 42 6 2 1 19 -20
-20 1 2 3 6 19 42
```

Here is the manual page for `qsort()`:

```
NAME
  qsort - sort an array

SYNOPSIS
  #include <stdlib.h>

  void qsort(void *base, size_t nmemb, size_t size, int
            (*compar)(const void *, const void *));

DESCRIPTION

  The qsort() function sorts an array with nmemb elements of size
  size. The base argument points to the start of the array.

  The contents of the array are sorted in ascending order according to
  a comparison function pointed to by compar, which is called with two
  arguments that point to the objects being compared.

  The comparison function must return an integer less than, equal to,
  or greater than zero if the first argument is considered to be
  respectively less than, equal to, or greater than the second. If two
  members compare as equal, their order in the sorted array is
  undefined.

RETURN VALUE

  The qsort() function returns no value.
```

4 Fibonacci Numbers (5 points)

Below is a source code listing with two different implementations of the Fibonacci sequence. Recall that the Fibonacci number F_n is defined for positive integers n as $F_n = F_{n-1} + F_{n-2}$, where $F_0 = 0$ and $F_1 = 1$. The first implementation is the classic recursive definition; the second does the same, but without recursion. Unfortunately, there are **two** bugs in the `fib_loop()` function. Please answer the following two questions about this listing:

(a) Find at least two bugs in the `fib_loop()` function and fix them so that the function is correct (i.e., mark the bugs and write the new code.) (3 points)

(b) The `fib_loop()` function, without recursion, is significantly faster than `fib_recursion()`. Why? (Note: It is **not** sufficient to say that loops are faster than recursion.) (2 points)

```
 1 unsigned int fib_recursion(unsigned int n) { /* NO BUGS here */
 2     if (n == 0) return 0;
 3     if (n == 1) return 1;
 4     return fib_recursion(n - 1) + fib_recursion(n - 2);
 5 }
 6
 7 unsigned int fib_loop(unsigned int n) { /* BUGS in this function */
 8
 9     int i;
10     int *buffer = malloc((n + 1) * sizeof(int));
11
12     buffer[0] = 0;
13     buffer[1] = 1;
14
15     for (i = 0; i <= n; i++) {
16         buffer[i] = buffer[i - 1] + buffer[i - 2];
17     }
18
19     free(buffer);
20     return buffer[n];
21
22 }
23
24 int main(int argc, char **argv) { /* NO BUGS here */
25     if (argc != 2) return EXIT_FAILURE;
26     int value = atoi(argv[1]);
27     printf("fib_recursion(%d) = %d\n", value, fib_recursion(value));
28     printf("fib_loop(%d) = %d\n", value, fib_loop(value));
29     return EXIT_SUCCESS;
30 }
```

The ASCII Table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
00	00	NUL	32	20	SP	64	40	@	96	60	‘
01	01	SOH	33	21	!	65	41	A	97	61	a
02	02	STX	34	22	"	66	42	B	98	62	b
03	03	ETX	35	23	#	67	43	C	99	63	c
04	04	EOT	36	24	$	68	44	D	100	64	d
05	05	ENQ	37	25	%	69	45	E	101	65	e
06	06	ACK	38	26	&	70	46	F	102	66	f
07	07	BEL	39	27	’	71	47	G	103	67	g
08	08	BS	40	28	(	72	48	H	104	68	h
09	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL