Full Name:______________________________________

Quiz II, Spring 2018 Date: 3/27

Instructions:

- Quiz II takes 60 minutes. Read through all the problems and complete the easy ones first.
- This exam is **closed book**, except that you may bring a single doube-sided page of prepared note.

1 (xx/24)	**2 (xx/24)**	**3 (xx/30**	**4 (xx/28)**	**Bonus (xx/10)**	**Total (xxx/100+10)**

1 Machine representation, bitwise operation (24 points):

Answer the following multiple-choice questions. Circle *all* answers that apply. Each problem is worth 8 points.

A. Suppose register %eax corresponds to the C variable `x` of some integer type. If the value of %eax is `0xffffffff`, what potentially could be the corresponding value of `x`?

1. -1
2. -2^{32}
3. -2^{64}
4. 2^{32}
5. $2^{32} - 1$
6. 2^{64}
7. $2^{64} - 1$
8. none of the above

B. Suppose `x` is of type `unsigned int`. Which of the following statement computes 0 if and only if the i-th bit of x (starting from the left) is zero? (The 0-th bit corresponds to the most significant bit).

1. `x & 0x80000000`
2. `(x << i) >> i`
3. `x & (1 << (31-i))`
4. `x | (0x80000000 >> i)`
5. `x & (0x7fffffff >> i)`
6. None of the above

C. Suppose `x` is of type `unsigned int`. Which of the following statement sets the i-th bit of x (starting from the left) to be one?

1. `x &= (1 << i)`
2. `x |= (1 << i)`
3. `x |= (1 << (31-i))`
4. `x &= (1 << (31-i))`
5. `x &= ~(1 << (31-i))`
6. None of the above

2 Basic C (24 points)

Answer the following multiple-choice questions. Circle *all* answers that apply. Each problem is worth 8 points.

A. In the following code snippet, what's the most likely outcome of the third line `*p = a`?

```
int a = 5;
int *p;
*p = a;
printf("%d\n", *p);
```

1. It will produce a compilation error.
2. It will cause a segmentation fault.
3. It will print 5.
4. It will print some memory address.
5. None of the above

B. Given the following code snippet, which assignment statement causes the program to print the character 'c'?

```
char *s = "abcdef";
char v;
v = ______;
printf("%c\n", v);
```

1. `v = s+2;`
2. `v = *(s+2);`
3. `v = s[2];`
4. `v = *(++s);`
5. `v = ++s;`
6. None of the above

C. What is the output of the following code snippet?

```
int a[5] = {1, 2, 3, 4, 5};
char *p;
p = (char *)a;
printf("%d\n", (int)p[4]);
```

1. 1
2. 2
3. 3
4. 4
5. 5
6. 0
7. none of the above

3 Hex to integer conversion(30 points):

Ben Bitdiddle is trying to implement a function `decodeHexToInteger` that decodes a string of ASCII hex digits in Little Endian byte-order into its corresponding integer format. Furthermore, the function should be able to decode into integer formats of various length. Ben will implement this function in three modular steps.

As a first step, Ben will implement a helper function, called `hex2Bin`, that outputs the corresponding integer value of a single ASCII hex digit. The following code snippet tests the correctness of `hex2Bin` using two examples.

```
unsigned char n = hex2Bin('f');
assert(n == 15);
n = hex2Bin('5');
assert(n == 5);
```

(a) (10 points) Please write the `hex2Bin` helper function below. You may assume that the input only contains lowercase hex digits. (Hint: Appendix I shows the ASCII table).

```
unsigned char
hex2Bin(char c)
{



}
```

In the second step, Ben will implement another helper function called `decodeHexByte` using the previous helper function `hex2Bin`. The `decodeHexByte` function that outputs the integer value of a byte represented as two ASCII hex digits. The following code snippet tests the correctness of `decodeHexByte` using two examples.

```
unsigned char n = decodeHexByte('5', '5');
assert(n == 85);
n = decodeHexByte('1', 'f');
assert(n == 31);
```

(b) (10 points) Please help Ben complete the `decodeHexByte` function below.

```
// decodeHexByte returns the integer value of a hex byte represented
// as two ASCII hex digits. "hd" is the higher order ASCII hex digit
// "ld" is the lower order ASCII hex digit
// Note: you must use the helper function hex2Bin
unsigned char decodeHexByte(char hd, char ld)
{













}
```

In the third and last step, Ben is ready to implement the function `decodeHexToInteger` that turns a string of hex digits into an integer with a given length.

```
// decodeHexToInteger parses 2*n hex ASCII digits
// into a n-byte integer value and stores the value in *result.
void decodeHexToInteger(char *buf, int n, char *result)
{
   for (int i = 0; i < n; i++) {
     result[i] = decodeHexByte(buf[2*i], buf[2*i+1]);
   }
}
```

For example, one could use the `decodeHexToInteger` to get a one-byte integer value as follows.

```
char y;
char *s = "1f";
decodeHexToInteger(s, sizeof(char), &y);
assert(y == 31);
```

(c) (10 points) Given Ben's implementation of `decodeHexToInteger` above, please fill in the appropriate spaces so that the code snippet prints out 10 when running on his laptop (a Little Endian machine).

```
int x;
char *s = _______________________;
decodeHexToInteger(s, sizeof(int), __________);
printf("%d\n", x); // should print out 10
```

4 Buffer Overflow (28 points):

Ben Bitdiddle writes a program that reads a hex string from the input, converts the string to an integer, increments it by one, and prints the result.

The following code is Ben's program (*(Hint: Ben's program potentially contains a bug.)*

```
int readAndIncrement() {
   int n;
   char buf[20];
   fgets(buf, 20, stdin); // read input from terminal. fgets' man page is in Appendix II
   decodeHexToInteger(buf, 8, &n); // decodeHexToInteger has the same impl. from Sec 3.
   n++;
   return n;
}

int main() {
   int x = readAndIncrement();
   printf("result is %d\n", x);
}
```

The corresponding disassembled code of Ben's program is shown below. The part marked as `...` indicates omitted lines for brevity.

```
000000000040063f <decodeHexByte>:
  ...
  400647:       e8 da ff ff ff          callq  400626 <hex2Bin>
  ...
  400652:       e8 cf ff ff ff          callq  400626 <hex2Bin>
  ...
  40065e:       c3                      retq

00000000004006a6 <readAndIncrement>:
  4006a6:       48 83 ec 18             sub    $0x18,%rsp
  ...
  4006b1:       be 0a 00 00 00          mov    $0x14,%esi //%esi stores 2nd arg. of fgets
  4006b6:       48 89 e7                mov    %rsp,%rdi //%rdi stores 1st arg. of fgets
  4006b9:       e8 42 fe ff ff          callq  400500 <fgets@plt>
  4006be:       48 8d 54 24 0c          lea    0x14(%rsp),%rdx   //%rdx stores 3rd arg. of decodeHexToInteger
  4006c3:       be 08 00 00 00          mov    $0x8,%esi //%esi stores 2nd arg. of decodeHexToInteger
  4006c8:       48 89 e7                mov    %rsp,%rdi //%rdi stores 1st arg. of decodeHexToInteger
  4006cb:       e8 8f ff ff ff          callq  40065f <decodeHexToInteger>
  4006d0:       8b 44 24 0c             mov    0x14(%rsp),%eax
  4006d4:       83 c0 01                add    $0x1,%eax
  4006d7:       48 83 c4 18             add    $0x18,%rsp
  4006db:       c3                      retq


00000000004006dc <main>:
   ...
  4007b5:       b8 00 00 00 00          mov    $0x0,%eax
  4007ba:       e8 e7 fe ff ff          callq  4006a6 <readAndIncrement>
  4007bf:       89 44 24 0c             mov    %eax,0xc(%rsp)
   ...
  4007ef:       c3                      retq
```

(a) (9 points) In Ben's program, which "buffer" might get overflowed?

1. The 20-byte `buf` in `readAndIncrement` can get overrun when `fgets` reads too many characters from .
2. The 4-byte storage for the local variable `x` in `main` might get overrun by `readAndIncrement`.
3. The 4-byte storage for the local variable `n` in `readAndIncrement` might get overrun by `decodeHexToInteg`
4. None of the above.

(b) (9 points) When the buffer overflow occurs, which return address among the following addresses gets overwritten?

1. `0x00000000004006be`
2. `0x00000000004006d0`
3. `0x00000000004006cb`
4. `0x00000000004007bf`
5. None of the above.

(c) (10 points) Alyssa P. Hacker crafts a malicious input string to exploit Ben's buffer overflow bug to hijack the control flow of the normal execution. When processing the malicious input, which is the last "normal" instruction executed by Ben's program before its control flow gets hijacked to execute code intended by Alyssa P. Hacker?

1. The `retq` instruction at address `0x00000000004006db`.
2. The `retq` instruction at address `0x00000000004007ef`.
3. The `retq` instruction at address `0x000000000040065e`.
4. None of the above.

This is a bonus question. You only need to do this for extra points.

Bonus question: (10 points) If Alyssa P. Hacker wants to hijack the program's control flow to execute the instruction at address `0000000000406789`, what is the input hex string that she should give as the input to Ben's program?

—END of Quiz II—-

Appendix: ASCII

ASCII(7) Linux Programmer’s Manual ASCII(7)

NAME

The following table contains the 128 ASCII characters encoded in octal, decimal, and hexadecimal

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	0	00	NUL ’\0’	100	64	40	@
001	1	01	SOH (start of heading)	101	65	41	A
002	2	02	STX (start of text)	102	66	42	B
003	3	03	ETX (end of text)	103	67	43	C
004	4	04	EOT (end of transmission)	104	68	44	D
005	5	05	ENQ (enquiry)	105	69	45	E
006	6	06	ACK (acknowledge)	106	70	46	F
007	7	07	BEL ’\a’ (bell)	107	71	47	G
010	8	08	BS ’\b’ (backspace)	110	72	48	H
011	9	09	HT ’\t’ (horizontal tab)	111	73	49	I
012	10	0A	LF ’\n’ (new line)	112	74	4A	J
013	11	0B	VT ’\v’ (vertical tab)	113	75	4B	K
014	12	0C	FF ’\f’ (form feed)	114	76	4C	L
015	13	0D	CR ’\r’ (carriage ret)	115	77	4D	M
016	14	0E	SO (shift out)	116	78	4E	N
017	15	0F	SI (shift in)	117	79	4F	O
020	16	10	DLE (data link escape)	120	80	50	P
021	17	11	DC1 (device control 1)	121	81	51	Q
022	18	12	DC2 (device control 2)	122	82	52	R
023	19	13	DC3 (device control 3)	123	83	53	S
024	20	14	DC4 (device control 4)	124	84	54	T
025	21	15	NAK (negative ack.)	125	85	55	U
026	22	16	SYN (synchronous idle)	126	86	56	V
027	23	17	ETB (end of trans. blk)	127	87	57	W
030	24	18	CAN (cancel)	130	88	58	X
031	25	19	EM (end of medium)	131	89	59	Y
032	26	1A	SUB (substitute)	132	90	5A	Z
033	27	1B	ESC (escape)	133	91	5B	[
034	28	1C	FS (file separator)	134	92	5C	\ ’\\’
035	29	1D	GS (group separator)	135	93	5D	]
036	30	1E	RS (record separator)	136	94	5E	^
037	31	1F	US (unit separator)	137	95	5F	_
040	32	20	SPACE	140	96	60	`
041	33	21	!	141	97	61	a
042	34	22	"	142	98	62	b
043	35	23	#	143	99	63	c
044	36	24	$	144	100	64	d
045	37	25	%	145	101	65	e
046	38	26	&	146	102	66	f
047	39	27		147	103	67	g
050	40	28	(	150	104	68	h
051	41	29	)	151	105	69	i
052	42	2A	*	152	106	6A	j
053	43	2B	+	153	107	6B	k
054	44	2C	,	154	108	6C	l
055	45	2D	−	155	109	6D	m
056	46	2E	.	156	110	6E	n
057	47	2F	/	157	111	6F	o
060	48	30	0	160	112	70	p
061	49	31	1	161	113	71	q
062	50	32	2	162	114	72	r
063	51	33	3	163	115	73	s
064	52	34	4	164	116	74	t
065	53	35	5	165	117	75	u
066	54	36	6	166	118	76	v
067	55	37	7	167	119	77	w
070	56	38	8	170	120	78	x
071	57	39	9	171	121	79	y
072	58	3A	:	172	122	7A	z
073	59	3B	;	173	123	7B	{
074	60	3C	<	174	124	7C	|
075	61	3D	=	175	125	7D	}
076	62	3E	>	176	126	7E	˜
077	63	3F	?	177	127	7F	DEL

Appendix: `fgets`

```
NAME
     fgets -- get a line from a stream

SYNOPSIS
     #include <stdio.h>

     char *
     fgets(char * str, int size, FILE * stream);

DESCRIPTION
     The fgets() function reads at most one less than the number of characters
     specified by size from the given stream and stores them in the string str.
     Reading stops when a newline character is found, at end-of-file or error.
     The newline, if any, is retained.  If any characters are read
     and there is no error, a ‘\0’ character is appended to end the string.
```