Full Name:_____

Midterm Exam, Fall 2015 Date: October 29th, 2015

Instructions:

- This midterm exam takes 70 minutes. Read through all the problems and complete the easy ones first.
- This exam is OPEN BOOK. You may use any books or notes you like. However, the use of any electronic devices including laptops, ipads, phones etc. is forbidden.

1 (xx/25)	2 (xx/25)	3 (xx/20)	4 (xx/30)	Total (xx/100)

1 Multiple choice questions (25 points):

Answer the following multiple-choice questions. Circle *all* answers that apply. Each problem is worth 5 points.

A. The 8-bit register %al has a hex value of `0xff`, what could be its corresponding value in decimal?

1. `128`
2. `255`
3. `-1`
4. `256`
5. None of the above

B. Consider the assembly instruction, `movq (%rax), %rcx`. If %rax represents a C variable, what are its possible data types?

1. `char **`
2. `char *`
3. `int *`
4. `long *`
5. `int`
6. `long`
7. None of the above

C. Suppose `int c = 0xdeadbeef`, which of the following statements clears the rightmost two bits in `c` and leaves the rest of the bits unchanged.

1. `c = 0xdeadbeec`
2. `c &= 0xfffffffc`
3. `c = (c >> 2) << 2`
4. `c |= 0xfffffffc`
5. `c |= 0x00000003`
6. None of the above

D. What is the content of array `A` after executing the following code snippet?

```
long A[3] = {1, 2, 3}
long *p;
long **q;
p = A;
p++;
q = &p;
p++;
(*p) = (**q)*2;
```

1. 1 2 3
2. 1 2 4
3. 1 2 6
4. 1 4 3
5. 1 4 6
6. None of the above

E. Consider the following code snippet,

```
float f = 1.0;
int b = *(int *)&f;
int c = 1;
```

Which of the following logical statements are true?

1. b != c
2. b > c
3. b < c
4. b == c
5. None of the above

2 C basics (25 points):

Ben Bitddle wants to write a program that sorts a set of strings using bubble sort. Below is the main body of Ben's program. It should print out three sorted strings, "apple orange pear".

```
1: void
2: main() {
3:    char strs[3][20] = {"orange", "apple", "pear"};
4:    for (int i = 0; i < 3; i++) {
5:       for (int j = i; j < 3; j++) {
6:     //strcmp returns positive if str[i] is greater than str[j] in alphabetical order
7:         if (strcmp(strs[i], strs[j]) > 0) {
8:            swap1(strs[i], strs[j]);
9:         }
10:      }
11:   }
12:   for (int i = 0; i < 3; i++)
13:      printf("%s ", strs[i]);
14: }
```

(a) (10 points) Please implement the `swap1` function to enable Ben's program to sort correctly. You may want to use the standard library function such as `strlen`, `strcpy` (see Appendix I) and others.

```
void
swap1(char *s1, char *s2) {



}
```

(b) (5 points) Ben has changed his code at *line 3* to be `char *strs[3] = {"orange", "apple", "pear"};`. Does his program that invokes `swap1` still work correctly? Please explain.

(c) (10 points) Ben has changed his code at *line 8* to be `swap2(&strs[i], &strs[j])`. Please implement the `swap2` function so that Ben's program will sort correctly when *line 3* is changed to be `char *strs[3] = {"orange", "apple", "pear"};`.

3 Assembly (20 points):

Ben Bitdiddle has encountered a `mystery` function. Its machine code is given below.

```
0000000000000000 <mystery>:
   0: 48 89 f8              mov    %rdi,%rax
   3: eb 09                 jmp    e <ex5+0xe>
   5: 48 39 30              cmp    %rsi,(%rax)
   8: 74 09                 je     13 <ex5+0x13>
   a: 48 8b 40 08           mov    0x8(%rax),%rax
   e: 48 85 c0              test   %rax,%rax
  11: 75 f2                 jne    5 <ex5+0x5>
  13: f3 c3                 repz retq
```

Ben is given the hint that the first argument of the mystery function is of the type `node *`, where `node` is defined as follows:

```
typedef struct node {
   long value;
   struct node *next;
}node;
```

(a) (5 points) How many arguments does function `mystery` take? If there are multiple arguments, what are their types?

(b) (5 points) Does the function have a return value? If so, what is the type of its return value?

(c) (10 points) Complete the puzzle by filling in the blanks for the corresponding C implementation of `mystery`.

```
_____
mystery(node *head, __________ )
{
   node *n = head;

   while (____________________) {

      if (____________________) {

         break;
      }

      ___________________;

   }

   return ________________;
}
```

4 C and Buffer Overflow (30 points):

(a) (10 points) Implement a C function, called `replace()`, that replaces the *first* occurance of a given substring `s1` in `str` with another given substring `s2`. The function `replace` returns 1 if replacement has happened, otherwise it returns zero. For example, given the following code snippet, `char str[20] = "hello"; int r = replace(str, "ll", "LLL");` The content of `str` should be "heLLLo" afterwards. And `r` should be 1.

For your implementation, you should just assume the argument `str` corresponds to a buffer large enough to hold the new string after replacement.

You may use standard library functions such as `strstr` or `strcpy` (the Appendix I contains the man pages for these two functions), but you do not have to.

```
//str, s1, s2 are null-terminated strings
int
replace(char *str, char *s1, char *s2) {
   //your code here




}
```

Consider the following code snippet that invokes `replace()`.

```
int
dangerous() {
   char buf[8] = "hello x";
   int r = replace(buf, "x", "friends and families");
   return r;
}
void
main() {
   dangerous();
   printf("Nothing bad happened\n");
}
```

Its corresponding object code is shown below. Note the unfamiliar `movabs` instruction moves an 8 byte constant value into the destination register.

```
000000000040065d <replace>:
  40065d: 41 57                 push   %r15
  ...
  400726: c3                    retq

0000000000400727 <dangerous>:
  400727: 48 83 ec 10           sub    $0x10,%rsp
  40072b: 48 b8 68 65 6c 6c 6f  movabs $0x78206f6c6c6568,%rax
  400732: 20 78 00
  400735: 48 89 04 24           mov    %rax,(%rsp)
  400739: ba f4 07 40 00        mov    $0x4007f4,%edx
  40073e: be fc 07 40 00        mov    $0x4007fc,%esi
  400743: 48 89 e7              mov    %rsp,%rdi
  400746: e8 12 ff ff ff        callq  40065d <replace>
  40074b: 48 83 c4 10           add    $0x10,%rsp
  40074f: c3                    retq

0000000000400750 <main>:
  400750: 48 83 ec 08           sub    $0x8,%rsp
  400754: b8 00 00 00 00        mov    $0x0,%eax
  400759: e8 c9 ff ff ff        callq  400727 <dangerous>
  40075e: bf fe 07 40 00        mov    $0x4007fe,%edi
  400763: e8 b8 fd ff ff        callq  400520 <puts@plt>
  400768: 48 83 c4 08           add    $0x8,%rsp
  40076c: c3                    retq
```

(b) (5 points) Read the machine code for function `dangerous`. If one is to print out the 8-byte constant `0x78206f6c6c6568` as an array of 8 ASCII characters, what are its contents? (If you need to consult an ASCII table, you may refer to Appendix II)

(c) (5 points) What is string stored at memory address `0x4007fc`? What is the string stored at memory address `0x4007f4`?

(d) (5 points) What is the value of the 8-bytes stored at the memory location `(%rsp)` just before executing the first instruction in `dangerous` (i.e. `400727: sub $0x10,%rsp`)?

(e) (5 points) Running this code gives a segmentation fault. What is the last instruction executed before the segmentation fault occurs? Please circle one.

1. The `retq` instruction in `main` function, i.e. `40076c: retq`.
2. The `retq` instruction in `dangerous` function, i.e. `40074f: retq`.
3. The `retq` instruction in `replace` function, i.e. `400726: retq`.
4. The instruction to deallocate from stack in `dangerous`, i.e. `400727: sub $0x10,%rsp`.

Appendix I: strstr, strcpy, strncpy

```
STRSTR(3)                                    Linux Programmer's Manual

NAME
       strstr - locate a substring

SYNOPSIS
       #include <string.h>

       char *strstr(const char *haystack, const char *needle);

DESCRIPTION
       The  strstr()  function finds the first occurrence of the substring needle
in the string haystack.  The terminating null bytes ('\0') are not compared.

RETURN VALUE
       The strstr() function returns a pointer to the beginning of the substring
in haystack, or NULL if the substring is not found.
```

```
STRCPY(3)                                    Linux Programmer's Manual

NAME
       strcpy, strncpy - copy a string

SYNOPSIS
       #include <string.h>

       char *strcpy(char *dest, const char *src);
       char *strncpy(char *dest, const char *src, size\_t n);

DESCRIPTION
       The  strcpy() function copies the string pointed to by src, including the
terminating null byte ('\0'), to the buffer pointed to by dest.  The strings may
not overlap, and the destination string dest must be large  enough to receive the
copy.

       The strncpy() function is similar, except that at most n bytes of src
are copied.  Warning: If there is no null byte among the first n bytes of src,
the string placed in dest  will  not  be null-terminated.

       If  the length of src is less than n, strncpy() writes additional null
bytes to dest to ensure that a total of n bytes are written.

RETURN VALUE
       The strcpy() and strncpy() functions return a pointer to the destination
string dest.
```

Appendix II: ASCII

NAME

ascii - ASCII character set encoded in octal, decimal, and hexadecimal

DESCRIPTION

ASCII is the American Standard Code for Information Interchange. It is a 7-bit code. Many 8-bit codes (such as ISO 8859-1, the Linux default character set) contain ASCII as their lower half. The international counterpart of ASCII is known as ISO 646.

The following table contains the 128 ASCII characters.

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	{

5 Solution

```
1. Multiple choice questions
A. 2, 3
B. 1, 4
C. 1, 2, 3
D. 3
E. 1, 2

2. C Basics
(a)
void
swap1(char *s1, char *s2) {
   char *tmp = (char *)malloc(len);
   strcpy(tmp, s1);
   strcpy(s1, s2);
   strcpy(s2, tmp);
}

(b) Ben's program would have a segmentation fault. This is because
the string buffers are stored in a read-only memory segment and
cannot be overwritten in swap1.

(c)
void
swap2(char **s1, char **s2) {
   char *tmp;
   tmp = *s1;
   *s1 = *s2;
   *s2 = tmp;
}

3. Assembly
(a) mystery function takes two arguments. We know this because mystery function uses
%rsi and %rdi without initialization (and %rsi and %rdi contain argument values by
the C calling convention) The first argument has type node *, second argument has type long.
(b) mystery function has one return value and the type is node *.
(c)
node *
mystery(node *head, long v) {
   node *n = head;
   while (n!= NULL) {
      if (n->value == v) {
         break;
      }
      n = n->next;
   }
   return n;
}

4. C and Buffer Overflow
(a)
int
replace(char *str, char *s1, char *s2) {
   char *s = strstr(str, s1);
   if (!s) return 0;
   char *tmp = (char *)malloc(strlen(str) + strlen(s2));
   strcpy(tmp, str);

   char *tmps = strstr(tmp, s1);
   strcpy(tmps, s2);
   strcpy(tmps+strlen(s2), s+strlen(s1));
   strcpy(str, tmp);
   free(tmp);
   return 1;
}

(b) "hello x" with a terminating null character
(c) 0x4007fc "x"
0x4007f4 "friends and families"
(d) 40075e
(e) 2.
```