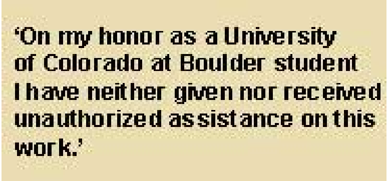
Full Name ______________________________________



'On my honor as a University of Colorado at Boulder student I have neither given nor received unauthorized assistance on this work.'

CSCI 2400, Fall 2017

Practice Second Midterm Exam

Instructions:

- Check that your exam has all N pages, and write your full name clearly on the front.
- Write your answers in the space provided for each problem. Feel free to use the back of each page to help you determine the answer, but make sure your answer is entered in the space provided on the front of the page.
- This exam is CLOSED BOOK and no electronics are allowed, except in the last 15 minutes and only to upload your answers to the moodle. You can use one page of personal notes and the printed midterm packet of tables. Good luck!

Problem	Possible	Score
1	**	
2	**	
3	**	
Total	**	

1. **[30 Points]**

Look at the assembly code below for the functions: main, p, q, and r. The code below assumes 64-bit stack conventions, NOT 32-bit. Use it to answer the following questions. All answers must be provided in decimal (base 10) system.

a) Which instruction in the main() function sets up the argument for the call to function p()? Please use the line number and NOT the hex address.

For instance if the answer is the line

0x400539 ⟨+0⟩: push %rbp

Then you should type **0** as the answer.

b) How many input arguments does the function r() have?

c) What is the stack frame size for function p() ? Please provide the answer including the old %rbp storage as part of the frame for the p() function.

d) What is the furthest offset/distance in bytes reached by the stack top pointer throughout the program from its original position at the beginning of main()?

e) What is the value in the register %eax at line 27 in the main() function?

```
main:
0x400539  <+0>: push   %rbp
0x40053a  <+1>: mov    %rsp,%rbp
0x40053d  <+4>: sub    $0x10,%rsp
0x400541  <+8>: movq   $0x20,-0x8(%rbp)
0x400549 <+16>: mov    -0x8(%rbp),%rax
0x40054d <+20>: mov    %eax,%edi
0x40054f <+22>: callq  0x40050a <p>
0x400554 <+27>: mov    %eax,-0xc(%rbp)
0x400557 <+30>: nop
0x400558 <+31>: leaveq
0x400559 <+32>: retq

q:
0x4004ea  <+0>: push   %rbp
0x4004eb  <+1>: mov    %rsp,%rbp
0x4004ee  <+4>: sub    $0x18,%rsp
0x4004f2  <+8>: mov    %edi,-0x14(%rbp)
0x4004f5 <+11>: mov    -0x14(%rbp),%eax
0x4004f8 <+14>: mov    %eax,%edi
0x4004fa <+16>: callq  0x4004d6 <r>
0x4004ff <+21>: mov    %eax,-0x4(%rbp)
0x400502 <+24>: mov    -0x4(%rbp),%eax
0x400505 <+27>: sar    $0x2,%eax
0x400508 <+30>: leaveq
0x400509 <+31>: retq
```

```
p:
0x40050a  <+0>: push   %rbp
0x40050b  <+1>: mov    %rsp,%rbp
0x40050e  <+4>: sub    $0x18,%rsp
0x400512  <+8>: mov    %edi,-0x14(%rbp)
0x400515 <+11>: mov    -0x14(%rbp),%eax
0x400518 <+14>: mov    %eax,%edi
0x40051a <+16>: callq  0x4004ea <q>
0x40051f <+21>: mov    %eax,-0x4(%rbp)
0x400522 <+24>: mov    -0x14(%rbp),%eax
0x400525 <+27>: mov    %eax,%edi
0x400527 <+29>: callq  0x4004d6 <r>
0x40052c <+34>: mov    %eax,-0x8(%rbp)
0x40052f <+37>: mov    -0x4(%rbp),%edx
0x400532 <+40>: mov    -0x8(%rbp),%eax
0x400535 <+43>: add    %edx,%eax
0x400537 <+45>: leaveq
0x400538 <+46>: retq

r:
0x4004d6  <+0>: push   %rbp
0x4004d7  <+1>: mov    %rsp,%rbp
0x4004da  <+4>: mov    %edi,-0x34(%rbp)
0x4004dd  <+7>: mov    -0x34(%rbp),%eax
0x4004e0 <+10>: add    %eax,%eax
0x4004e2 <+12>: mov    %eax,-0x4(%rbp)
0x4004e5 <+15>: mov    -0x4(%rbp),%eax
0x4004e8 <+18>: pop    %rbp
0x4004e9 <+19>: retq
```

2. **[24 Points]**

```
char *attack="vulnerable!aaaa";
int get_buffer(){
char buf[5];
read_input(buf);
return 0;
}

void read_input(char *buf){
int i;
for (i=0; attack[i] != 0; i++){
buf[i] = attack[i];
}
buf[i]=0;
}
```

```
0x00000000004004d6 <get_buffer>:
4004d6  push   %rbp
4004d7  mov    %rsp,%rbp
4004da  sub    $0x10,%rsp
4004de  lea    -0x5(%rbp),%rax
4004e2  mov    %rax,%rdi
4004e5  mov    $0x0,%rax
4004ea  callq  0x4004f6 <read_input>
400r4ef mov    $0x0,%rax
4004f4  mov    %rbp,%rsp
4004f6  pop    %rbp
4004f7  retq

0x0000000000400555  <main>
400555  push   %rbp
400556  mov    %rsp,%rbp
400559  mov    $0x0,%eax
40055e  callq  0x4004d6 <get_buffer>
400563  mov    $0x0,%eax
400568  pop    %rbp
400569  retq
```

Similarly to `Gets` from your Attack lab, `read_input` writes data in to the array, buf, that is its only argument. To simplify things `read_input`' input comes from a null-terminated string stored in memory, called `attack`. (Remember that C stores strings as null-terminated character arrays).

The diagram below represents a section of the stack, with memory addresses and `buf` labeled. The following table of characters' hex values may also be helpful.

a	0x61
v	0x76
u	0x75
!	0x21

q	0x71
b	0x6f
r	0x72
e	0x65

n	0x6e
z	0x7a
l	0x62
null	0x00

Using the code above answer the following questions (**Assuming 64-bit System and a program compiled with frame pointer enabled**)

(a) Suppose we have just executed the pop instruction (at 0x4004f6) within get buffer. Fill in the following diagram of the stack with the correct hex values at that point. The value of buf and a few memory addresses have been filled in to get you started. (Assuming buf[0] is at 0x30). (12 points)

0x30	0x31	0x32	0x33	0x34	0x35	0x36	0x37
76	75	62	6e				

0x38	0x39	0x3a	0x3b	0x3c	0x3d	0x3e	0x3f

(b) What is the distance in terms of number of bytes between main's %rsp (at 0x400556) and get buffer's %rbp (after being set at 0x4004d7)? (4 points)

(c) What is the location of %rsp after the execution of the pop instruction at 0x4004f6? (4 points)

(d) To what value is %rbp set after the execution of the pop instruction at 0x4004f6 (Hex value)? (4 points)

3. **[22 Points]**

Consider the following two 8-bit floating-point representations based on the IEEE floating point format. Neither has a sign bit–they can only represent nonnegative numbers.

(a) Format A

i. There are k = 4 exponent bits. The exponent bias is 7.

ii. There are n = 4 fraction bits.

(b) Format B

i. There are k = 5 exponent bits. The exponent bias is 15.

ii. There are n = 3 fraction bits.

Below, you are given some bit patterns in Format A, and your task is to convert them to the closest value in Format B. **If necessary, you should apply the round-to-even rounding rule**. In addition, give the values of numbers given by the Format A and Format B bit patterns. Specify values as whole numbers or decimals.

Finally to summarize the types expected for blanks—-Please enter a **numeric value** for the **Exponent, Mantissa and Final Value** fields and enter a **bit pattern for the Bits** field.

Some useful constants:

- $1/2 = 0.5$
- $1/4 = 0.25$
- $1/8 = 0.125$
- $1/16 = 0.0625$
- $1/32 = 0.03125$

Format A Bits	Exponent (E)	Mantissa (M)	Final Value ($M \times 2^E$)	**Format B** Bits	Final Value
0111 0000	0	1.0	1	01111 000	1
1001 1001					