CSCI 2021, Spring 2017
Homework Assignment II

Problem 1:

Clearly label your assignment with the time of your recitation section. This will help us turn back your graded assignments more efficiently.

Problem 2:

Textbook problem 3.20 (p. 219).

Problem 3:

Textbook problem 3.31 (p. 237).

Problem 4:

Consider the source code below, where `M` and `N` are constants declared with `#define`.

```
int mat1[M][N];
int mat2[N][M];

int sum_element(int i, int j)
{
  return mat1[i][j] +  mat2[i][j];
}
```

A. Suppose the above code generates the following assembly code:

```
sum_element:
movslq %esi, %rsi
movslq %edi, %rdi
leaq (%rsi,%rdi,8), %rdx
subq %rdi, %rdx
leaq (%rdi,%rdi,4), %rax
addq %rax, %rsi
movl mat2(,%rsi,4), %eax
addl mat1(,%rdx,4), %eax
ret
```

What are the values of `M` and `N`?

M =

N =

Problem 5:

Condider the following assembly code for a C `for` loop:

```
loop:
movslq %esi, %rsi
leaq -1(%rdi,%rsi), %rdx
jmp .L2
.L3:
movzbl (%rdi), %eax
xorb (%rdx), %al
movb %al, (%rdi)
xorb (%rdx), %al
movb %al, (%rdx)
xorb %al, (%rdi)
addq $1, %rdi
subq $1, %rdx
.L2:
cmpq %rdx, %rdi
jb .L3
ret
```

Based on the assembly code above, fill in the blanks below in its corresponding C source code. (Note: you may only use the symbolic variables `h`, `t` and `len` in your expressions below — *do not use register names.*)

```
void loop(char *h, int len)
{
   char *t;

   for (_____________; ___________; h++,t--) {

      ________________;

      ________________;

      ________________;
   }

   return;
}
```

The following problem concerns the following, low-quality code:

```
void foo(int x)
{
  int a[3];
  char buf[4];
  a[0] = 0xF0F1F2F3;
  a[1] = x;
  gets(buf);
  printf("a[0] = 0x%x, a[1] = 0x%x, buf = %s\n", a[0], a[1], buf);
}
```

In a program containing this code, procedure `foo` has the following disassembled form:

```
.LC0:
        .string "a[0] = 0x%x, a[1] = 0x%x, buf = %s\n"

foo:
        pushq   %rbx
        subq    $16, %rsp
        movl    %edi, %ebx
        movq    %rsp, %rdi
        call    gets
        movq    %rsp, %rcx
        movl    %ebx, %edx
        movl    $-252579085, %esi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        addq    $16, %rsp
        popq    %rbx
        ret
```

For the following questions, recall that:

- `gets` is a standard C library routine.
- x86-64 machines are little-endian.
- C strings are null-terminated (i.e., terminated by a character with value 0x00).
- Characters '`0`' through '`9`' have ASCII codes `0x30` through `0x39`.

Problem 6:

Fill in the following table indicating where on the stack the following program values are located. Express these as decimal offsets (positive or negative) relative to register `%rsp`:

Program Value	Decimal Offset
`a`	
`a[2]`	
`x`	
`buf`	
`buf[3]`	
Saved value of register `%rbx`	

Consider the case where procedure `foo` is called with argument `x` equal to `0xE3E2E1E0`, and we type "`12345678901234567890`" in response to `gets`.

A. Fill in the following table indicating which program values are/are not corrupted by the response from `gets`, i.e., their values were altered by some action within the call to `gets`.

Program Value	Corrupted? (Y/N)
`a[0]`	
`a[1]`	
`a[2]`	
`x`	
Saved value of register `%ebx`	

B. What will the `printf` function print for the following:

- `a[0]` (hexadecimal): ______________________________
- `a[1]` (hexadecimal): ______________________________
- `buf` (ASCII): ______________________________

C. Array `a` is not allocated on the stack in this program. What additional code can force the array to be allocated on the stack? If the array is allocated on the stack, how will the results for part B of this question change?

Problem 7:

Consider the following incomplete definition of a C struct along with the incomplete code for a function `func` given below.

```
typedef struct node {

  _______________ x;

  _______________ y;

  struct node *next;

  struct node *prev;

} node_t;
```

```
node_t n;

void func() {

  node_t *m;

  m = ______________________;

  m->y /= 16;

  return;
}
```

When this C code was compiled, the following assembly code was generated for function `func`.

```
func:
        movq    n+16(%rip), %rax
        movq    24(%rax), %rax
        movw    $16, 8(%rax)
        ret
```

Given these code fragments, fill in the blanks in the C code given above. Note that there is a unique answer.

The types must be chosen from the following table, assuming the sizes and alignment given.

Type	Size (bytes)	Alignment (bytes)
char	1	1
short	2	2
unsigned short	2	2
int	4	4
unsigned int	4	4
double	8	8

The next problem concerns the following C code. This program reads a string on standard input and prints an integer in hexadecimal format based on the input string it read.

```
#include <stdio.h>

/* Read a string from stdin into buf */
int evil_read_string()
{
    int buf[2];

    scanf("%s",buf);
    return buf[1];
}

int main()
{
    printf("0x%x", evil_read_string());
}
```

Here is the corresponding machine code on a x86-64 machine:

```
evil_read_string:
        subq    $24, %rsp
        movq    %rsp, %rsi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    __isoc99_scanf
        movl    4(%rsp), %eax
        addq    $24, %rsp
        ret

.LC1:
        .string "0x%x"

main:
        subq    $8, %rsp
        movl    $0, %eax
        call    evil_read_string
        movl    %eax, %esi
        movl    $.LC1, %edi
        movl    $0, %eax
        call    printf
        addq    $8, %rsp
        ret
```

Problem 8:

This problem tests your understanding of the stack discipline and byte ordering. Here are some notes to help you work the problem:

- `scanf("%s", buf)` reads an input string from the standard input stream (stdin) and stores it at address `buf` (including the terminating `'\0'` character). It does **not** check the size of the destination buffer.
- `printf("0x%x", i)` prints the integer i in hexadecimal format preceded by “`0x`”.
- Recall that x86-64 machines are Little Endian.

Suppose we run this program on a x86-64 machine, and give it the string “`abcdefghijk`” as input on stdin.

Here is a template for the stack, showing the locations of `buf[0]` and `buf[1]`. Fill in the value of `buf[1]` (in hexadecimal).

```
               |<- buf[0]->|<-buf[1] ->|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

What is the 4-byte integer (in hex) printed by the `printf` inside main?

0x__________________________

How many bytes of input will corrupt the return address?

____

Problems 1, 4 and 6 should be submitted for grading.