Homework 7

Problem 1

Consider the following declaration:

```
short     S[5];
short     *T[4];
int       **U[6];
double    V[8];
double    *W[7];
```

Fill in the table below:

Array	Size of Element	Size of Array	Start Address	Address of i-th Element
S			x_S	
T			x_T	
U			x_U	
V			x_V	
W			x_W	

Problem 2

Consider the following codes:

```
1   int mat1[M][N];
2   int mat2[N][M];
3
4   int sum_element(int i, int j)
5   {
6       return mat1[i][j] + mat2[j][i];
7   }
```

GCC generates the following assembly codes:

```
1       movl    8(%ebp), %ecx
2       movl    12(%ebp), %eax
3       leal    0(,%eax,4), %ebx
4       leal    0(,%ecx,8), %edx
5       subl    %ecx, %edx
6       addl    %ebx, %eax
7       sall    $2, %eax
8       movl    mat2(%eax,%ecx,4), %eax
9       addl    mat1(%ebx,%edx,4), %eax
```

Use reverse engineering techniques to determine the values of M and N according to assembly codes above.

Problem 3

Suppose you are responsible for maintaining a C program and encounter the following codes:

```
1   typedef   struct {
2       int left;
3       a_struct a[CNT];
4       int right;
5   } b_struct;
6
7   void test(int i, b_struct *bp)
8   {
9       int n = bp->left + bp->right;
10      a_struct *ap = &bp->a[i];
11      ap->x[ap->idx] = n;
12  }
```

Unfortunately, you are not permitted to access ".h" file which defines constant CNT and struct a_struct. But you can access ".o" file and use *objdump* to disassemble the object file. Disassembled codes are displayed as follows:

```
1  00000000    <test>:
2    0:     55                    pushl     %ebp
3    1:     89 e5                 movl      %esp, %ebp
4    3:     53                    pushl     %ebx
5    4:     8b 45 08              movl      0x8(%ebp), %eax
6    7:     8b 4d 0c              movl      0xc(%ebp), %ecx
7    a:     8d 04 80              leal      (%eax, %eax, 4), %eax
8    d:     8d 44 81 04           leal      0x4(%ecx, %eax, 4), %eax
9    11:    8b 10                 movl      (%eax), %edx
10   13:    c1 e2 02              shll      $0x2, %edx
11   16:    8b 99 b8 00 00 00     movl      0xb8(%ecx), %ebx
12   1c:    03 19                 addl      (%ecx), %ebx
13   1e:    89 5c 02 04           movl      %ebx, 0x4(%edx, %eax, 1)
14   22:    5b                    popl      %ebx

15   23:    89 ec                  movl      %ebp, %esp
16   25:    5d                     popl      %ebp
17   26:    c3                     ret
```

Use reverse engineering techniques to get following values:

A. Value of CNT;

B. Declaration of struct a_struct, assuming that a_struct contains only idx and x;

Problem 4

Suppose we have the following function 'login' to perform login process.

```
int login()
{
	char username[8];
	char password[8];

	gets(username);
	gets(password);

	return check_match_in_database(username, password);
}
```

Here is a part of the function's assembly.

```
	pushl	%ebp
	movl	%esp, %ebp
	subl	$40, %esp
	leal	-16(%ebp), %eax
	movl	%eax, (%esp)
	call	_gets
	leal	-24(%ebp), %eax
	movl	%eax, (%esp)
	call	_gets
	……
```

In the normal process, if the username and the password are both ok, the function 'login_ok' will be called to indicate login success. We've already known that the address of 'login_ok' is 0x004013da. Can you construct an input to make the function 'login_ok' be called after 'login' returns? You just need to specify **the key bytes** and **their positions** rather than the complete input.