Question 1:

The following code transposes the elements of an `M` × `M` array, where `M` is a constant defined by `#define`:

```
1  void transpose(long A[M][M]) {
2      long i, j;
3      for (i = 0; i < M; i++)
4          for (j = 0; j < i; j++) {
5              long t = A[i][j];
6              A[i][j] = A[j][i];
7              A[j][i] = t;
8          }
9  }
```

When compiled with optimization level -O1, gcc generates the following code for the inner loop of the function:

```
1  .L6:
2    movq    (%rdx), %rcx
3    movq    (%rax), %rsi
4    movq    %rsi, (%rdx)
5    movq    %rcx, (%rax)
6    addq    $8, %rdx
7    addq    $120, %rax
8    cmpq    %rdi, %rax
9    jne     .L6
```

We can see that gcc has converted the array indexing to pointer code.

A. Which register holds a pointer to array element `A[i][j]`?

(%rdx)

B. Which register holds a pointer to array element `A[j][i]`?

(%rax)

C. What is the value of `M`?

`M = 8`

Question 2:

Consider the following union declaration:

```
1  union ele {
2      struct {
3          long *p;
4          long y;
5      } e1;
6      struct {
7          long x;
8          union ele *next;
9      } e2;
10 };
```

This declaration illustrates that structures can be embedded within unions.

The following function (with some expressions omitted) operates on a linked list having these unions as list elements:

```
1  void proc (union ele *up) {
2      up->_________ = *(_________) - _________;
3  }
```

A. What are the offsets (in bytes) of the following fields:

`e1.p`: 0

`e1.y`: 8

`e2.x`: 0

`e2.next`: 8

B. How many total bytes does the structure require?

16 bytes

C. The compiler generates the following assembly code for proc:

```
    void proc (union ele *up)
    up in %rdi
1   proc:
2     movq    8(%rdi), %rax
3     movq    (%rax), %rdx
4     movq    (%rdx), %rdx
5     subq    8(%rax), %rdx
6     movq    %rdx, (%rdi)
7     ret
```

On the basis of this information, fill in the missing expressions in the code for proc. Hint: Some union references can have ambiguous interpretations. These ambiguities get resolved as you see where the references lead. There is only one answer that does not perform any casting and does not violate any type constraints.

```
void proc (union ele *up) {
        up->y = *(up->p) - up.y;
}
```

Question 3:

The C code below shows a (low-quality) implementation of a function that reads a line from standard input, copies the string to newly allocated storage, and returns a pointer to the result.

a) C Code

```
char *get_line(){{
   char buf[8];
   char *result;
   gets(buf);
   result = malloc(strlen(buf+1));
   if(result)
          strcpy(result, buf);
   return result;
}
```

b) Assembly code up through call to gets

```
   get_line:
          400720: 53                  push   %rbx
          400721: 48 83 ec 10         subq   $0x10, %rsp
          400725: 48 89 e7            movq   %rsp, %rdi
          400728: e8 73 ff ff ff      callq  4006a0  <gets>
```

Consider the following scenario. Procedure **`get_line`** is called with the return address equal to **`0x400776`** and register **`%rbx`** equal to **`0x0123456789012345`**. You type in the string

`ABCDEFGHIJKLMNOPQRSTUVWXYZ`

The program terminates with a segmentation fault. You run GDB and determine that the error occurs during the execution of the **`ret`** instruction in **`get_line`**.

A. Show the stack layout, on the diagram below, as much as you can after executing the instruction at the address **`0x400721`** in the assembly code above. Label the quantities stored on the stack (eg. “Return address”) on the right, and their hexadecimal values (if known) within the box. Each box represents 8 bytes. Indicate the position of **`%rsp`**. Recall that ASCII codes for characters A-Z are **`0x41-0x5A`**.

`00 00 00 00 00 40 07 76`	Return address
`41 42 43 44 45 46 47 48`	40077E
`49 4a 4b 4c 4d 4e 4f 50`	400787
`51 52 53 54 55 56 57 58`	40078F

`59 5a`	400798
`00 00 00 00 00 00 00 00`	40079A=> sub 0x10
`00`	4007A3 => rsp

B. Modify the stack diagram to show the effect of the call to **`gets`** after executing the instruction at the address **`0x400728`**.

`00 00 00 00 00 40 07 76`	Return address
`41 42 43 44 45 46 47 48`	40077E
`49 4a 4b 4c 4d 4e 4f 50`	400787
`51 52 53 54 55 56 57 58`	40078F
`59 5a`	400798
`41 42 43 44 45 46 47 48`	40079A => gets(buf)

C. To what address does the program attempt to return?

0x400776

D. What register(s) have corrupted value(s) when **`get_line`** returns?

%rbx