# CH09. Review Question

## Short Answer

1. DF = 1
2. 2byte ( DF = 0이면 EDI += 2, DF = 1이면 EDI -= 2)
3. 피연산자가 모두 암시적이라 내부적으로 esi와 edi 중 어떤 쪽에서 어떤 쪽을 빼는지 명확하게 보이지 않는다
4. SCASB는 edi와 al을 비교후 edi를 1증가시킴. 반복이 끝난 후에는 찾은 문자 바로 다음 위치를 가리킴
5. REPNE / REPNZ
6. STD를 사용해 역방향 진행
7. 현재 문자가 구분자와 같지 않을 때(JNE) 트리밍 위치를 찾았다 판단하고 null ㅇㄹ 그 뒤에 넣기 위함
8. 아무변화 없이 그대로 유지된다
9. null까지 스캔 해야되기 때문에 REPNE가 가장 적절하다
10. .
    EDI ← 문자열 시작 주소, AL ← 0, CLD, ECX는 충분히 큰 값으로 설정.
    repne scasb 로 널(0) 문자를 찾을 때까지 스캔.
    반복 종료 후:
        EDI 는 널 다음 위치를 가리킴.
        문자열 길이 = (EDI - 1) - 문자열 시작 주소.
    이 길이를 EAX에 넣어서 반환한다.
11. N = 1024 -> log2(1024) = 10.x~번  => 11번 비교
12. .

FillArray는 배열을 낮은->높은 주소 순으로 채운다. 하지만 DF는 이전 코드에 의해 역방향(1)일수도 있으므로 CLD를 항상 DF = 0으로 초기화 해야 올바른 방향으로 증가가 가능하기 때문이다

13.
L1에 이미 cmp edx, edi를 수행해서 플래그들이 세팅 되어있고, L2는 같은거를 한번 더 하는 중복코드라서 CMP를 제거해도 동작 결과가 동일하다

14.
L4에 있는 코드 바로아래 L1라벨을 옮기거나, 분기 구조를 바꿔 l1으로 떨어지게 하면 L4의 jmp 문장을 제거할 수 있다.

## Algorithm Workbench

1.
```
mov eax, [ebx + esi]
```

2.
```
mov eax, myArray[ebx + esi]
```

3.
```
myArray[ESI + EDI]
```

4.
```
mov esi, OFFSET sourcew
mov edi, OFFSET targetw
mov ecx, Count
cld
repe cmpsw
```

5.
```
mov ax, 0100h
mov edi, OFFSET wordArray
mov ecx, LENGTHOF wordArray
cld
repne scasw

jnz  NotFound
lea eax, [edi - 2]
jmp  Done

NotFound:
mov eax, 0

Done:
```

6.
```
INVOKE Str_compare, ADDR string1, ADDR string2


.IF ZERO?
    mov edx, OFFSET string1
.ELSEIF CARRY?
    mov edx, OFFSET string2
.ELSE
    mov edx, OFFSET string1
.ENDIF

call WriteString
call Crlf
```

7.
```
INVOKE Str_trim, ADDR myString, '@'
```

8.
```
Str_lcase PROC
    push esi
    mov esi, [esp+4]

L1: mov al, [esi]
    cmp al, 0
    je  L2
    cmp al, 'A'
    jb  NextChar
    cmp al, 'Z'
    ja  NextChar
    add al, 20h
   mov [esi], al
NextChar:
    inc esi
    jmp L1

L2: pop esi
    ret
Str_lcase ENDP
```

9.

```
Str_trim64 PROC
    mov rdi, rcx
FindEnd:
    mov al, [rdi]
    cmp al, 0
    je  BackScanStart
    inc rdi
    jmp FindEnd

BackScanStart:
    dec rdi
    std
BackScan:
    cmp rdi, rcx
    jb  Done
    mov al, [rdi]
    cmp al, dl
    jne NotDelim
    mov byte ptr [rdi], 0
    dec rdi
    jmp BackScan
NotDelim:
    cld
Done:
    ret
Str_trim64 ENDP
```

10.

```
mov rax, [rbx + rsi]
```

11.

```
mov eax, myArray[ebx*COLS*4 + edi*4]
```

12.

```
mov rax, myArray[rbx*COLS*8 + rdi*8]
```