Dadas las siguientes declaraciones en Pascal:

```
Const
      LimInf = 10;
      LimSup = 255;
Var
      X,Y: real;
      N,M: integer;
      Cadena: string;
      Car: char;
      Mayus, Cumple, Ok: boolean;
```

Analizar si son sintáctica y semánticamente correctas, redundantes, las siguientes sentencias:

a) `Mayus := upcase(car) = car;`

Es correcto. Mayus va a ser true si car es mayúscula. Car es una variable que contiene un carácter, por ej. 'P', en este caso upcase('P') = 'P'. Si car = 'p', upcase('p') = 'p' es falso porque 'P' <> 'p

b) `Cumple := (X <= LimInf) and (X >= LimSup);`

Error semántico: un valor no puede ser menor a 10 y mayor a 255 a la vez.

c) `Mayus := 'A' = car or car = ´B' or TRUE;`

Error sintáctico: faltan paréntesis: ('A' = car) or (car = ´B') or TRUE.
Error semántico y redundancia: al estar TRUE al final de una expresión con OR siempre retorna true más allá de lo que se evalúe en las 2 primeras expresiones, los que las hace redundantes.

d) `Cumple := length(cadena) > LimSup;`

Error semántico: la función length devuelve el largo de una cadena. El límite en Pascal es 255 caracteres. LimSup vale 255. No puede haber una cadena de más de 255 caracteres.

e) `M := N / LimInf;`

Error sintáctico. Se pierden los decimales de la división al asignarlo a una variable entera. No coinciden los tipos de la variable y la expresión que participan de la asignación.

f) `Ok := LimInf < M < LimSup`

Error sintáctico: la expresión es correcta semánticamente, pero debe reescribirse como
(M > LimInf) and (M < LimSup).

g) `Ok := Odd(n *(n - 1));`

Error semántico: el argumento siempre es par, sea n par o impar. Multiplicar 2 valores consecutivos siempre es par. Ok siempre es falso, porque no es impar.