Errores más comunes en el manejo de punteros

1. Errores de compilación más frecuentes

   a) Las variables puntero sólo pueden apuntar a datos de un tipo particular. Por lo tanto, para que los punteros puedan compararse o asignarse entre sí tienen que ser del mismo tipo.

   b) Confundir el puntero (P) con la variable a la que apunta(P^)

2. Errores de ejecución más frecuentes

   a) La variable referenciada por un puntero sólo existe cuando se inicia el apuntador mediante la asignación a una variable ya existente o mediante *new()*. Un error muy frecuente es intentar acceder a la variable referenciada cuando no existe. En este caso, estaremos intentando acceder a una dirección de memoria no válida y causará un error de ejecución –normalmente un *Segmentation Fault*.

   - Incorrecto:

```
type tPos:^integer;
var p:tPos;
begin
    p^:=...
```

   - Correcto.

```
type tPos:^integer;
var p, q: tPos;
begin
    new(p); p^:=... ó
    new(q);p:=q; p^:=...
```

   Hay que tener en cuenta que los punteros acceden directamente a la memoria del ordenador y, por lo tanto, al acceder a posiciones de memoria no reservadas y escribir en ellas pueden ocurrir errores inesperados como escribir en el propio código del programa

   b) Para evitar este problema haremos que el puntero contenga el valor nil siempre que no apunte a una variable. Así, podremos reconocer cuándo el puntero apunta o no a una variable con sólo preguntar por su valor

   c) Cuidaremos las expresiones compuestas de los bucles. La mayoría de los compiladores de Pascal no evalúan en cortocircuito, por lo que la sentencia:

```
while (Ptr<>nil)  and  (ptr^ > valor) do
```

   producirá un error cuando Ptr sea nil.

   d) Hay que tener cuidado cuando tenemos varios punteros que apuntan a la misma variable, ya que la modificación de la variable por parte de uno de ellos implicará que también cambiará el contenido para los demás

   e) Además, si uno de ellos libera la variable los demás quedarán desreferenciados (referencias perdidas)

f) La memoria de un ordenador es grande pero no ilimitada, y puede acabarse si constantemente creamos nuevas variables sin liberar el espacio de las que ya no necesitemos. En este sentido, otro tipo de errores que no causan error de ejecución están relacionados con el dispose:

1) Dejar variables a las que ya no apuntan ningún puntero, sin haber hecho un dispose. Supone una pérdida de capacidad de memoria para ese programa
2) Hacer `Dispose(P)` y no preocuparnos de que P apunte a alguna dirección válida o a *nil*.
3) Hacer un `Dispose(P)` y acceder posteriormente a P.