

Bloque de actividades avanzadas I

1. Cómo separar el videojuego en módulos

1.1 Estructuras

Como ya sabes, es posible declarar variables de diferentes tipos (int, string, float...) en la zona GLOBAL del programa o bien en las zonas PRIVATE y LOCAL de los procesos.

```
string nombre;  
int damage;  
float velocidad;
```

Hay datos, como por ejemplo las estadísticas de un arma, que pueden requerir multitud de variables. En esos casos conviene crear estructuras o TYPES con varios campos, por ejemplo:

```
TYPE tp_arma  
    string nombre;  
    int damage;  
    float velocidad;  
END
```

Con esto ya podríamos declarar variables del tipo tp_arma igual que lo hacemos al declarar variables de los tipos por defecto:

```
tp_arma arma;
```

Y para dar valores a cada uno de los campos se utiliza el operador '.' así:

```
arma.nombre="espada";  
arma.damage=10;  
arma.velocidad=1.0;
```

Una de las muchas ventajas de los TYPES es que resulta muy sencillo trabajar con listas. Podemos declarar una lista de 100 armas:

```
tp_ama lista_armas[100];
```

1. Cómo separar el videojuego en módulos

1.2 Módulos o TADs

Siempre que definimos un nuevo TYPE conviene crear unas operaciones asociadas a él (FUNCTIONS) que en primer lugar sirven para trabajar cómodamente con el TYPE y en segundo lugar para separar su código a otro archivo PRG distinto.

Las FUNCTIONS se diferencian de los PROCESS en que no tienen gráfico, simplemente realizan unas operaciones. Has solamente 3 tipos de FUNCTIONS.

1.2.1 Las que dan valores iniciales al dato son las de creación.

```
FUNCTION nueva_arma(tp_arma *arma, string nombre, int damage, float velocidad)
BEGIN
    (*arma).nombre=nombre;
    (*arma).damage=damage;
    (*arma).velocidad=velocidad;
END
```

1.2.2 Las que modifican el dato son las de modificación:

```
FUNCTION aumentar_damage(tp_arma *arma, int damage_extra)
BEGIN
    (*arma).damage=(*arma).damage+1;
END
```

1.2.3 Y por último las que consultan un dato son las de observación.

```
FUNCTION damage_por_segundo(tp_arma arma)
BEGIN
    RETURN arma.damage * arma.velocidad;
END
```

Atención porque el operador * se utiliza para indicar que algún parámetro de la FUNCTION va a ser modificado. Hasta ahora los parámetros que pasábamos eran simples copias, y al modificarlos NO modificábamos el parámetro original.

Por ejemplo al invocar desde el protagonista un disparo:

```
disparo(301,x,y);
```

Le pasábamos la x,y del protagonista, pero el disparo NO modificaba esa x,y, simplemente la consultaba para determinar su propia x,y.

NO debe confundirse el operador infijo * que sirve para multiplicar dos operandos con el operador prefijo * que sirve para "apuntar". El nombre formal del operador prefijo * es puntero, y tiene una infinidad de aplicaciones que trabajaremos poco a poco.