# Programando con Inform: vectores, uso práctico

## Introducción

El manejo de vectores por parte de *Inform* es bastante poco intuitivo, es decir, utiliza una sintaxis muy extraña, tanto para el programador habitual en otros lenguajes (que suelen utilizar el operador [], como C, C++, Pascal ... etc.), como para el creador de aventuras novel en programación. En este pequeño artículo veremos qué es un vector, la sintaxis utilizada en *Inform*, y un ejemplo práctico del uso de vectores en *InformATE*!.

Un vector es básicamente una colección de valores. El vector tiene unas determinadas posiciones donde podemos "colocar" ciertos datos de cualquier tipo. Es decir, podemos tener un vector de cadenas, un vector numérico, un vector de objetos ...

```
v = {1, 46, 5, 78, 90}
```

Lo anterior es un vector de longitud 5. Tiene por tanto cinco posiciones: desde la 0 hasta la 4. En cada posición, podemos almacenar un valor, como si de una variable se tratara, y posteriormente recuperarlo.

Además de los vectores, tenemos los arrays, que son vectores de vectores, o colecciones bidimensionales (o más dimensiones, aunque normalmente, en programación de aventuras, será difícil encontrarle aplicación a un array de más de dos dimensiones), en lugar de las unidimensionales como los vectores. Un ejemplo de un array podría ser un array *a* de 10x10, diez filas y diez columnas, o lo que es lo mismo, 10 vectores de 10 posiciones cada uno. Una posible aplicación, llevar el control de los tiros del enemigo en una batalla de barcos (no_acertado [0], agua [1], barco [2], barco_tocado [3]).

```
      | 1 0 0 0 0 0 0 1 0 0|
      | 1 0 0 0 2 2 0 0 1 0|
      | 1 0 0 0 0 0 0 0 0 0|
      | 1 0 0 0 0 0 0 0 0 1|
      | 0 0 0 0 0 0 0 0 0 0|
a =   | 0 2 2 2 0 0 0 0 0 3|
      | 0 0 0 0 0 0 0 0 0 3|
      | 3 3 0 0 0 0 0 0 0 3|
      | 0 0 0 0 0 0 0 0 2 0|
      | 0 2 2 2 2 2 0 0 0 0|
```

### *Algo más práctico*

Así, por ejemplo, en el lenguaje de programación C, vemos un ejemplo de utilización de un vector de 5 posiciones:

```
#include <stdio.h>
int main(void) {
      int v[5],
            i;

      v[0] = 1;
      v[1] = 46;
      v[2] = 5;
      v[3] = 78;
      v[4] = 90;

      printf("\nv = {");
      for(i=0; i<5;++i)
      {
            printf("%d", v[i]);
            if (i<4)
                  printf(",");
      }
```

```
        printf(“}\n”);
}
```

Definimos el vector, de cinco posiciones, en la tercera línea. A partir de la 6ª línea, y en las cinco líneas siguientes, damos valores a cada una de las posiciones del mismo. En las seis últimas líneas, recorremos el vector y lo visualizamos, provocando la siguiente salida en pantalla:

```
v = {1, 46, 5, 78, 90}
```

Es bastante intuitiva la sintaxis utilizada: *v[x]* permite acceder o modificar la posición, siendo *x* la posición deseada, de entre *0* y el tamaño del vector.

## Vectores y Arrays en Inform

La sintaxis de *Inform* es bastante distinta. Es la siguiente, que se utiliza tanto para asignar como para acceder a los valores:

```
self.&elementos-->i
```

En lugar de utilizar los corchetes, se utiliza una combinación entre el símbolo ‘&’ y la flecha ‘→’ para indicar el acceso a una determinada posición. El programa anterior se realizaría de la siguiente manera en Inform.

```
object v
    with elementos 1 46 5 78 90
;

[main i;
    print "^v = {";
    for (i=0:i<5:i++)
    {
      print v.&elementos-->i;
      if (i<4)
            print ",";
    }
    print "}^";
];
```

El programa es básicamente el mismo, si bien el vector se declara como un objeto fuera de la función *main*, no como una variable más dentro de ella, como en el caso de C.

Para obtener la longitud de un vector, se utiliza el mismo método que en C, dividiendo el tamaño total en bytes del vector por el tamaño de cada uno de los elementos. Aunque el caso de *Inform* es .ligeramente diferente: todos los elementos ocupan siempre lo mismo; una palabra de memoria. El tamaño de la palabra de memoria viene definida para el compilador por la máquina virtual de destino, según se genere código *Z* o *Glulx*. El tamaño de palabra de la máquina de destino está siempre disponible en la constante del compilador WORDSIZE. En el caso de la máquina Z es 16 bits, 2 bytes, WORDSIZE == 2, y en el caso de Glulx es de 32 bits, 4 bytes, WORDSIZE == 4.

Por tanto, la forma de obtener el tamaño total de un vector es:

```
v.#elementos
```

Y la forma de obtener el número de elementos es:

```
(v.#elementos / WORDSIZE)
```

La razón de que una posición del vector ocupe siempre lo mismo, viene de cómo maneja *Inform* los tipos de datos. Un entero ocupa siempre el total de una palabra de memoria. Los objetos no se guardan directamente en el vector, sino sólo un número que permite saber dónde está el objeto en memoria. Por ejemplo, en *Inform*, un vector puede ser de cadenas. Sin embargo, la propia cadena no es la que se almacena en el vector, sino un número que permitirá identificar la cadena en una posición de memoria. Éstos

números son más o menos equivalentes a los punteros de C/C++, o las referencias en Java.

```
object v
    with elementos
            "La casa se sitúa al este, con la puerta bloqueada."
            "La casa al este tiene la puerta tapiada."
            "Hay una casa al este. Tiene la puerta tapiada."
;
```

Así, si queremos imprimir la cadena, tendremos que decirle a Inform que imprima el número (puntero) que le damos (guardado en el vector) como la cadena a la que apunta, no como el número en sí mismo que es interno a *Inform* y no nos interesa.

```
print (string) v.&elementos-->i;
```

### *Los arrays*

*Inform* no los soporta, pero es fácil simular un array. Supongamos que deseamos un array de 2x2 valores. Es decir v[0][0], v[0][1], v[1][0], v[1][1]; supongamos que esas cuatro posiciones guardan los valores 1, 2, 3, y 4, respectivamente. Cada una de las dos filas es un pequeño vector. Lo único que se debe hacer es crear un vector con el tamaño total deseado (cuatro, en este caso), y calcular la posición de entre el total, asumiendo que cada fila es almacenada una a continuación de la otra. Es decir, el *array* sería en realidad un vector normal, de cuatro posiciones:

v = {1, 2, 3 4}

Pero calculamos la posición según las coordenadas f,c que nos pidan:

posición_real = (f*total_columnas) + c;

De esta forma, si es necesario crear un array de 2x2, tan sólo es necesario crear un vector de [2x2=] 4 posiciones, y codificar una función de acceso a los elementos como la siguiente:

```
class array2x2
private
      elementos 0 0 0 0,
with
      elemento [f c;
            return self.&elementos-->((f*2) + c);
      ],
      pon_elemento [f c val;
            self.&elementos- ->((f*2) + c) = val;
      ],
      longitud [;
            return (self.#elementos / WORDSIZE);
      ]
;
```

Y por supuesto, es posible crear una clase para el caso general de una matriz *n*x*m* [3].

Pueden encontrarse clases de arrays y vectores en el módulo array.h [2].

## Un caso práctico en InformATE!

Un caso típico para el cual es necesario el manejo de vectores es el de acumular mensajes, para algún propósito específico, como las descriciones de una serie de objetos. Otra aplicación es la de colocar los mensajes de un PSI, para indicar cómo se va a seguir una conversación; o responder con diferentes frases a una misma acción del jugador, por ejemplo al examinar un objeto. Un ejemplo podría ser el siguiente.

```
class responde
class vector
private
      num 0
```

```
with
      reset [; self.num = 0;],
      dev_msg [
            if (n < (self.longitud() – 1))
                  self.num++;
            return self.elemento(n);
      ]
;
```

La clase desciende de la clase vector, por lo que para utilizar *responde*, debemos crearla con un número de mensajes dentro de la propiedad *elementos*. Lo que hace es que, automáticamente, responde con diferentes mensajes, los guardados en *elementos*, según un contador que guarda en su interior. Al llegar al último mensaje del vector que guarda en su interior, se detiene. Por supuesto, no es necesario el desarrollar esta clase para conseguir esa funcionalidad, pero de esta forma se automatiza totalmente el proceso. Un ejemplo de su uso se muestra a continuación.

```
object exInscrip
class responde
private
      elementos
            “Con un vistazo inicial, descubres que son inscripciones egipcias. Pero necesitarías investigar en más detalle.”
            “Poco a poco descifras las inscripciones. El dialecto egipcio en el que están escritas te es familiar.”
            “Las inscripciones ahora se revelan a tus ojos: La puerta al este guarda el tesoro de éste”
;

object inscripciones HabitacionSecreta
with
      nombre ‘inscripciones’,
      descripcion [; print (string) exInscrip.dev_msg(); print “^”;]
has nombre_plural;
```

Obsérvese el pequeño cambio necesario en el objeto inscripciones. En lugar de poner el código necesario para mostrar uno tras otro los tres mensajes, en el método descripción, tan sólo llamamos al método `dev_msg()` del objeto adecuado, encargándose `exInscrip`, de la clase responde, de efectuar todo el trabajo. Así, el escritor de la aventura sólo debe preocuparse de utilizar una serie de mensajes adecuados para la aventura: el trabajo de mostrarlos ya lo hace la clase desarrollada. Incluso puede decidir en cualquier momento añadir o quitar mensajes: no tiene que preocuparse de la funcionalidad. Clases cómo ésta pueden encontrarse en el módulo responde.h [2].

Las aplicaciones son muy variadas: es sencillo hacer, por ejemplo, una clase que devuelva un mensaje aleatorio del total de mensajes en el vector, y utilizar este mensaje para dar cada vez a una localidad una descripción distinta. También se puede crear una clase para dar una serie de mensajes, hasta llegar al último, y volver a comenzar desde el principio, o hacerlo aleatoriamente.

## Conclusiones

Se ha visto como automatizar el uso de vectores de Inform con una sencilla aunque potente aplicación práctica en InformATE!. Los vectores de mensajes suelen ser un recurso muy utilizado en programación de aventuras, sobre todo cuando no tenemos un parser y el lenguaje de programación utilizado es Pascal o C o C++. InformATE! elimina la necesidad inmediata de tener esas estructuras, pero en cuanto es necesario mantener una colección de valores para algún propósito, como mostrar una serie de mensajes cuando un objeto es examinado como se ha visto, los vectores se revelan como muy útiles, y sus aplicaciones son infinitas.

## Referencias

[1] baltasarq@yahoo.es
[2] Lo encontrarás en http://usuarios.lycos.es/elarquero/
[3] Que queda como ejercicio para el amable lector (aunque está resuelto en el módulo array.h).