Unidad 1
Introducción a las Estructuras de Datos

1.1 Tipos de datos abstractos (TDA).
1.2 Modularidad.
1.3 Ejemplos de uso de TDAs.
1.4 Manejo de memoria Estática.

1.1 Tipos de Datos Abstractos

Aunque la programación de computadoras es realizada por personas con conocimientos extensos en las ciencias computacionales, **esconder la complejidad natural de los datos** a los diseñadores y programadores de computadoras ha permitido alcanzar el grado de desarrollo que tiene en la actualidad la industria informática.

De eso se trata la **abstracción de datos.**

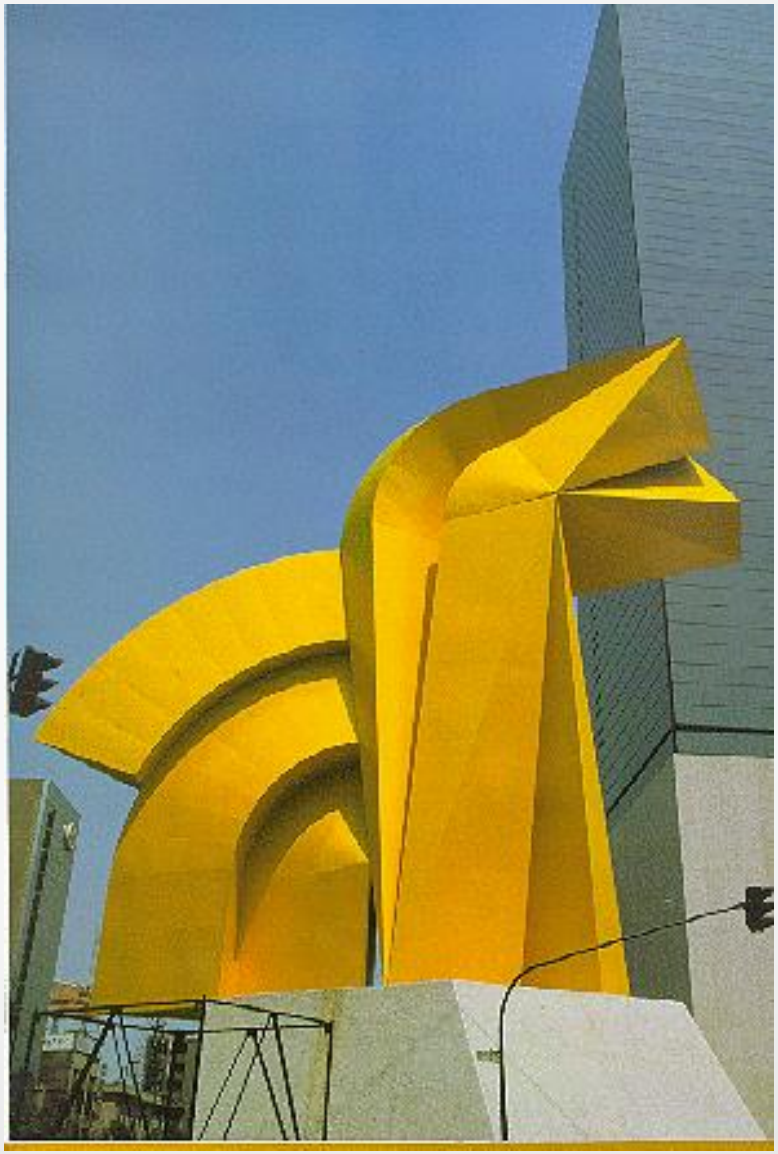
1.2 Tipos de Datos Abstractos

¿Qué es la **abstracción**?

Análisis de una cosa prescindiendo de los detalles de sus componentes.

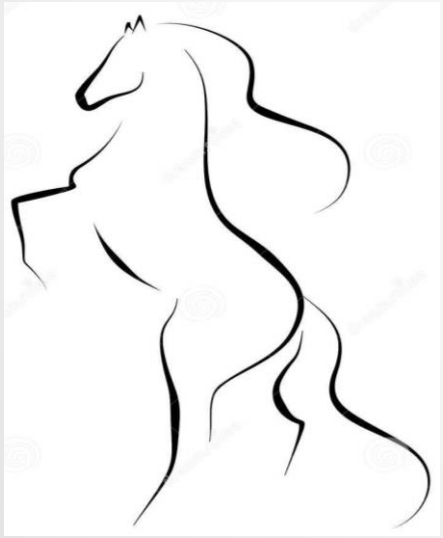
Ejemplo:

El caballito en la Ciudad de México



Sebastián

1.2 Tipos de Datos Abstractos

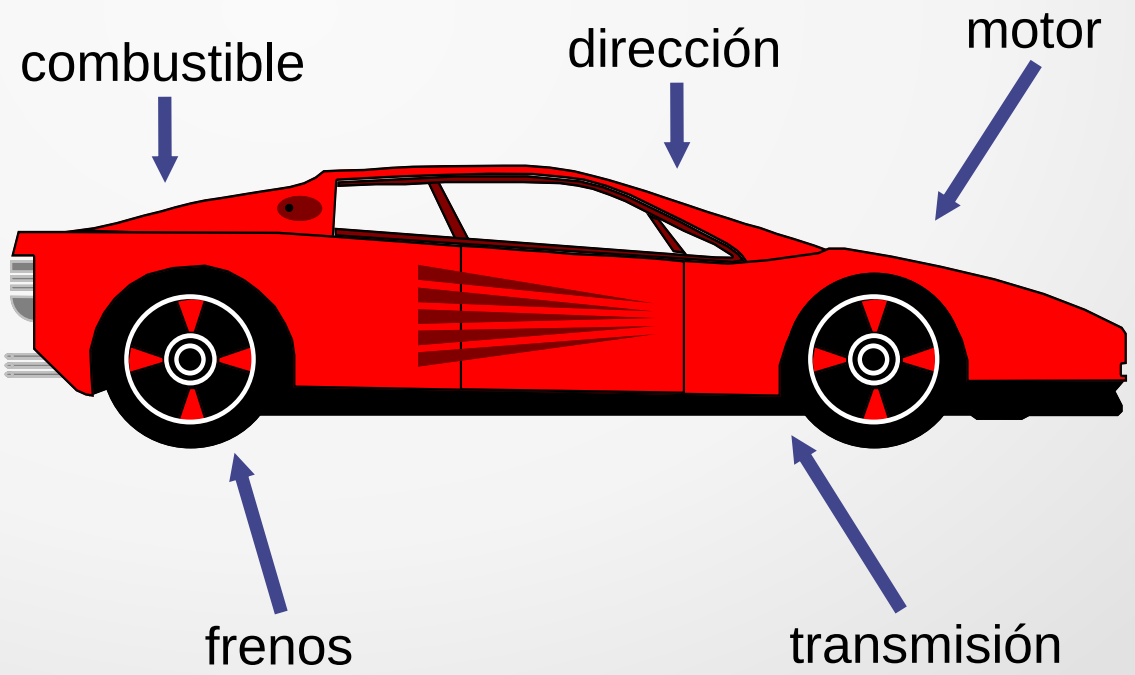


Arte
Abstracto

Arte Figurativo (**Realista**)

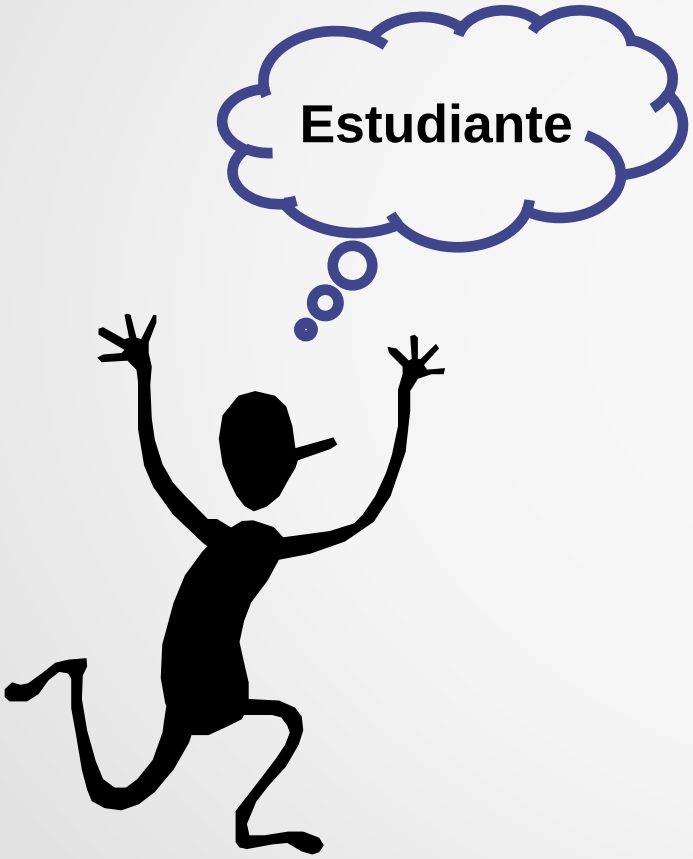
1.2 Tipos de Datos Abstractos

abstracción



1.2 Tipos de Datos Abstractos

abstracción



Número de Matrícula	Nombre	Carrera	Semestre que cursa

1.1 Tipos de Datos Abstractos

Tipo de dato abstracto es un concepto en el que el tipo de dato está definido por su comportamiento desde el punto de vista del usuario de los datos y no del programador (el usuario es con mucha frecuencia el mismo programador que usará el TDA).

Si el programador utiliza sus propios TDA, al usarlos, ignorará la composición interna del TDA y solo se concentrará en su comportamiento (de esta forma **se simplifica la programación**).

1.2 Modularidad

La modularidad es un concepto de **programación** que permite, para efectos de claridad y eficiencia **dividir** la solución de un problema grande **en módulos más pequeños**.

Para escribir un TDA es indispensable descomponer el problema en procesos más simples para lograr un nivel de abstracción adecuado.

1.2 Modularidad

La Programación Orientada a Objetos (POO), usa los conceptos de TDA y modularidad mediante el uso de métodos (procedimientos o funciones pertenecientes a una clase).

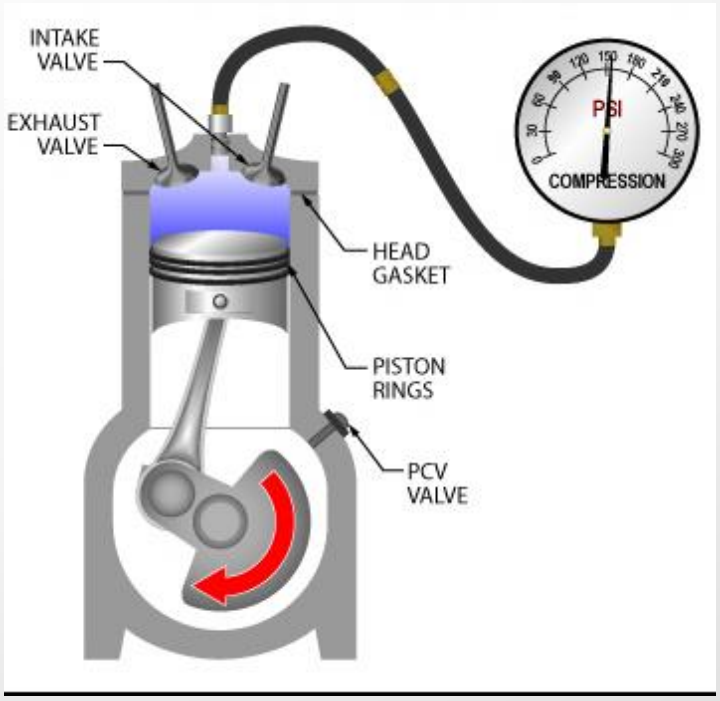
Una ventaja importante de la modularidad es la reutilización de código. Se busca que un procedimiento o función pueda ser invocado en múltiples ocasiones desde diferentes procesos para evitar redundancia de código que puede causar inconsistencias.

1.3 Ejemplo de Tipos de Datos Abstractos

Ejercicio 1

Escriba un programa, en C++ preferentemente, que resuelva el siguiente problema:

Un mecánico de automóviles necesita realizar varias pruebas de compresión a uno de los cilindros de un automóvil.

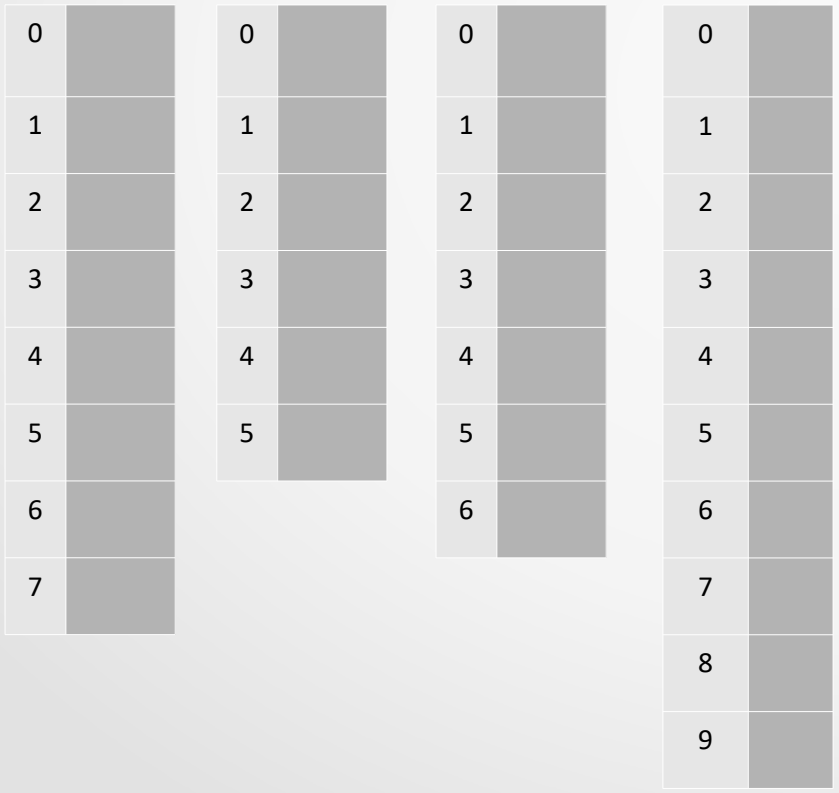


1.3 Ejemplo de Tipos de Datos Abstractos

- A intervalos de 10 segundos aproximadamente va a realizar varias tomas de compresión pero no se sabe de antemano cuantas va a realizar.
- El mecánico va a registrar una a una las diferentes lecturas del manómetro.
- La presión se registrará en PSI (libras por pulgada cuadrada; valor numérico entero).
- El mecánico indicará a la aplicación que las lecturas terminaron, escribiendo como valor un cero.
- El programa deberá reportar el primer valor y el último registrados, el número de lecturas realizadas, el valor promedio y el número de lecturas mayores al promedio.

1.3 Ejemplo de Tipos de Datos Abstractos

Para resolver ciertas aplicaciones de manera más simple, es conveniente tener vectores dinámicos (la memoria reservada por los vectores en los lenguajes es estática).



Memoria estática: Espacio reservado en la memoria durante la compilación; el espacio se libera al terminar la ejecución del programa. Si se necesitara añadir más datos a un vector, tendría que modificarse el programa y declarar un vector de mayor tamaño.

Memoria dinámica: Espacio reservado o liberado durante la ejecución del programa.

1.3 Ejemplo de Tipos de Datos Abstractos

Se deberá crear una clase llamada VectorDinamico que se pueda usar, en un programa, de la manera que se indica enseguida (se ilustra el uso con datos tipo **unsigned char**).

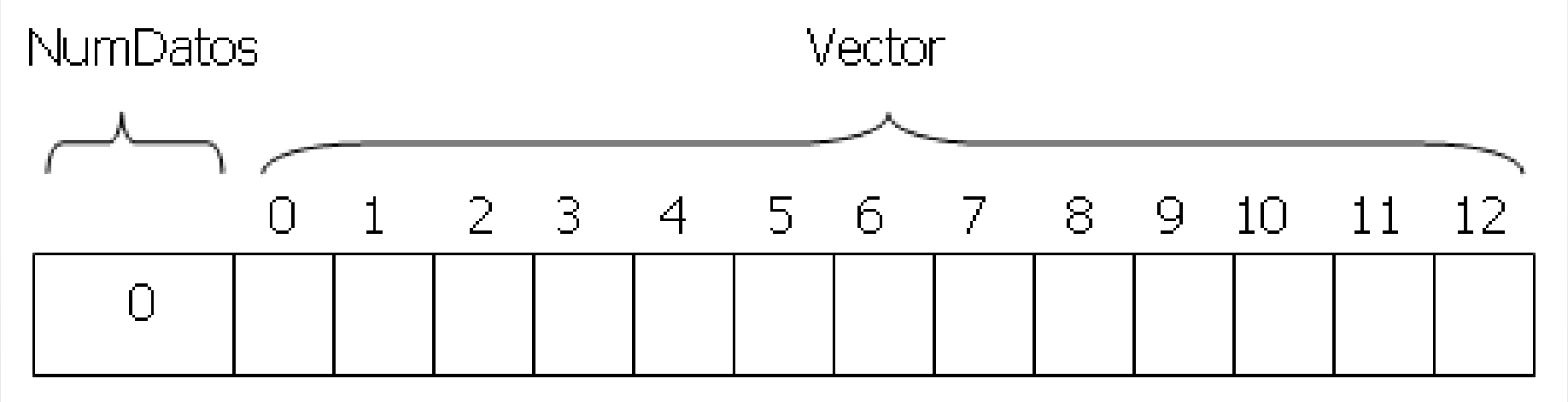
VectorDinamico v; // Se declara una variable de tipo VectorDinamico

Ejemplo de uso	**Significado**
v.Anadir('*');	
v.Anadir(42)	// añadir un dato al final
v.Insertar(´#´,3);	
v.Insertar(34,3)	// insertar un dato en la posición 3
v.Eliminar(5)	// Eliminar el dato de la posición 5
v.ModifTamanio(11);	// Modificar tamaño del vector a 11
cout << v.Get(5)	// Consultar el dato de la posición 5
cout << v.Tamanio()	// Número total de datos guardados

1.3 Ejemplo de Tipos de Datos Abstractos

Si los arreglos son estáticos como lograr este comportamiento?

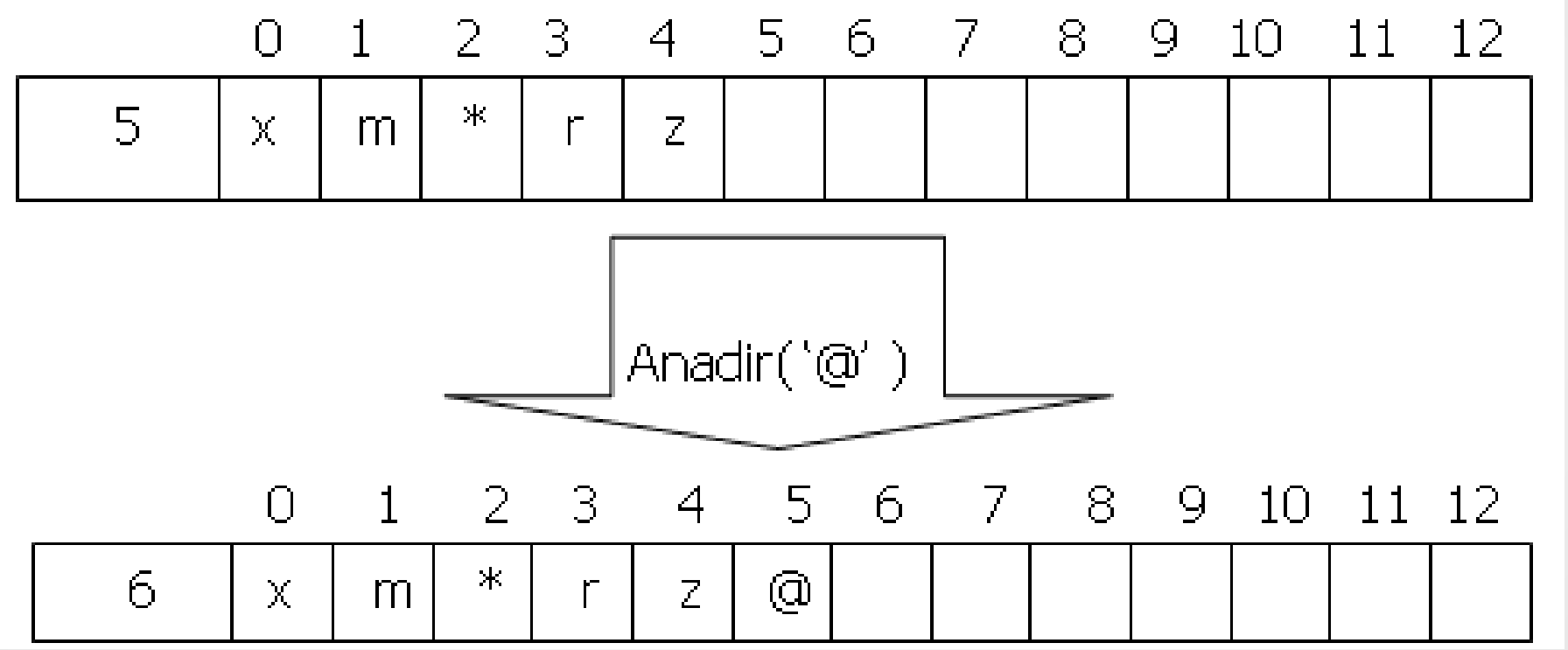
Con un TDA que asuma a un vectoor y aun entero (que contenga el número de posiciones usadas del vector) como un ***vector dinámico***



VectorDinamico::Constructor()

Numdatos ← 0

1.3 Ejemplo de Tipos de Datos Abstractos

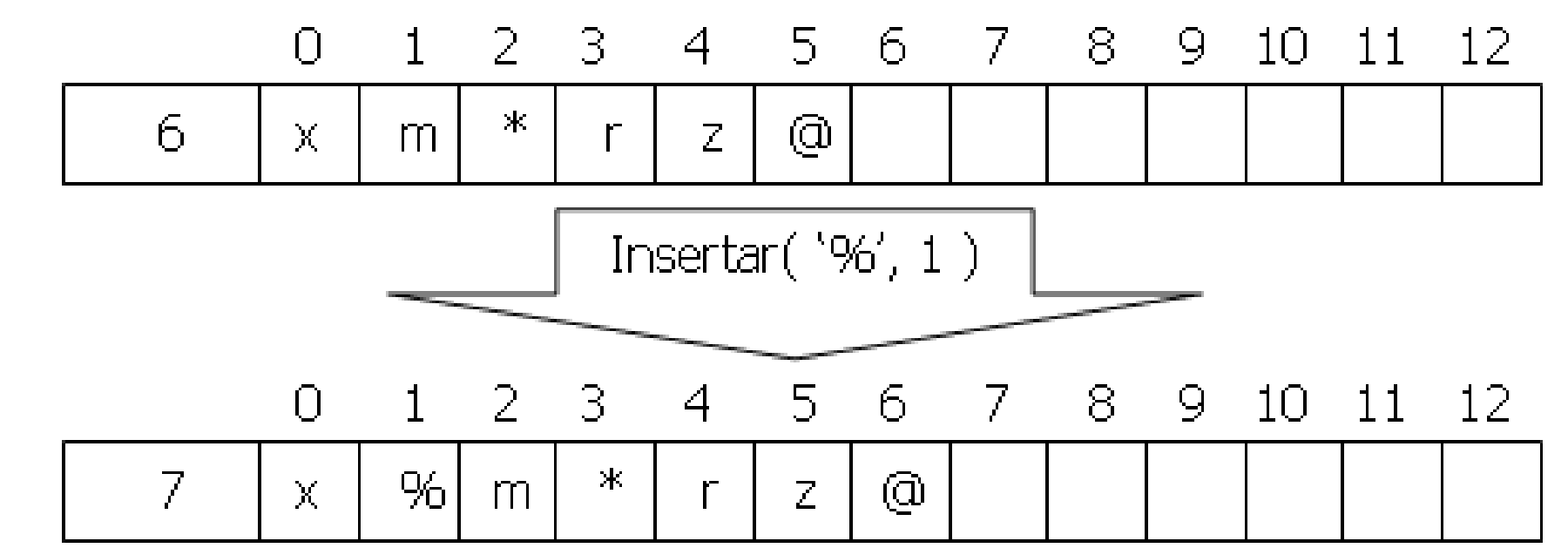


VectorDinamico::Anadir(dato)

Vector[NumDatos] ← dato

Numdatos ← NumDatos + 1

1.3 Ejemplo de Tipos de Datos Abstractos



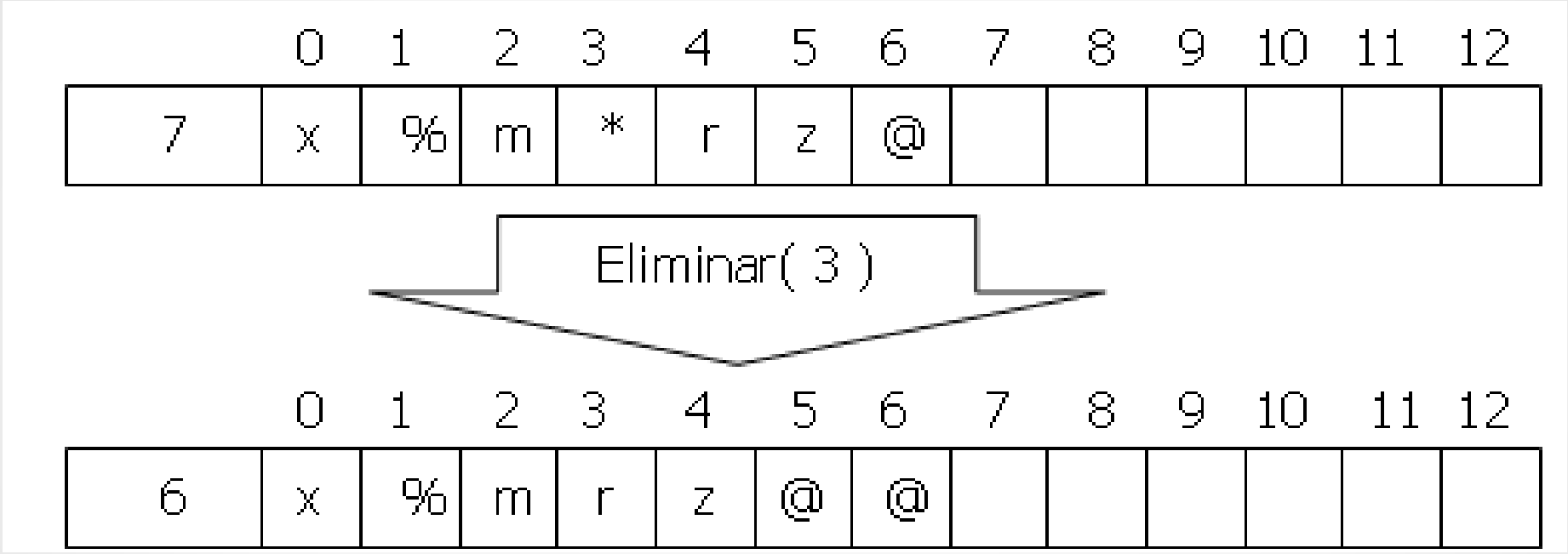
VectorDinamico::Insertar(dato,pos)

Copiar cada valor, entre las posiciones **NumDatos-1** y **pos**, a la siguiente dirección. A este proceso se le llama *corrimiento inverso*.

Vector[pos] ← dato

NumDatos ← NumDatos + 1

1.3 Ejemplo de Tipos de Datos Abstractos

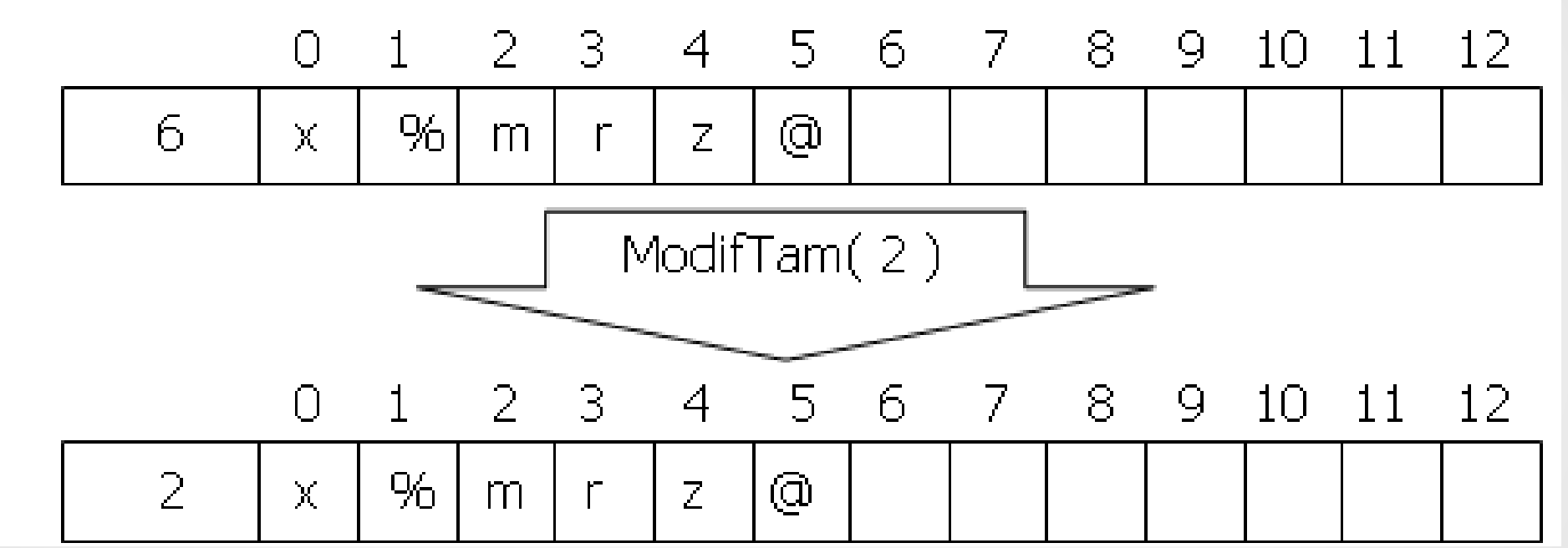


VectorDinamico::Eliminar(pos)

Copiar cada valor, entre las posiciones **pos+1** y **NumDatos-1,** a la dirección anterior.

NumDatos ← NumDatos - 1

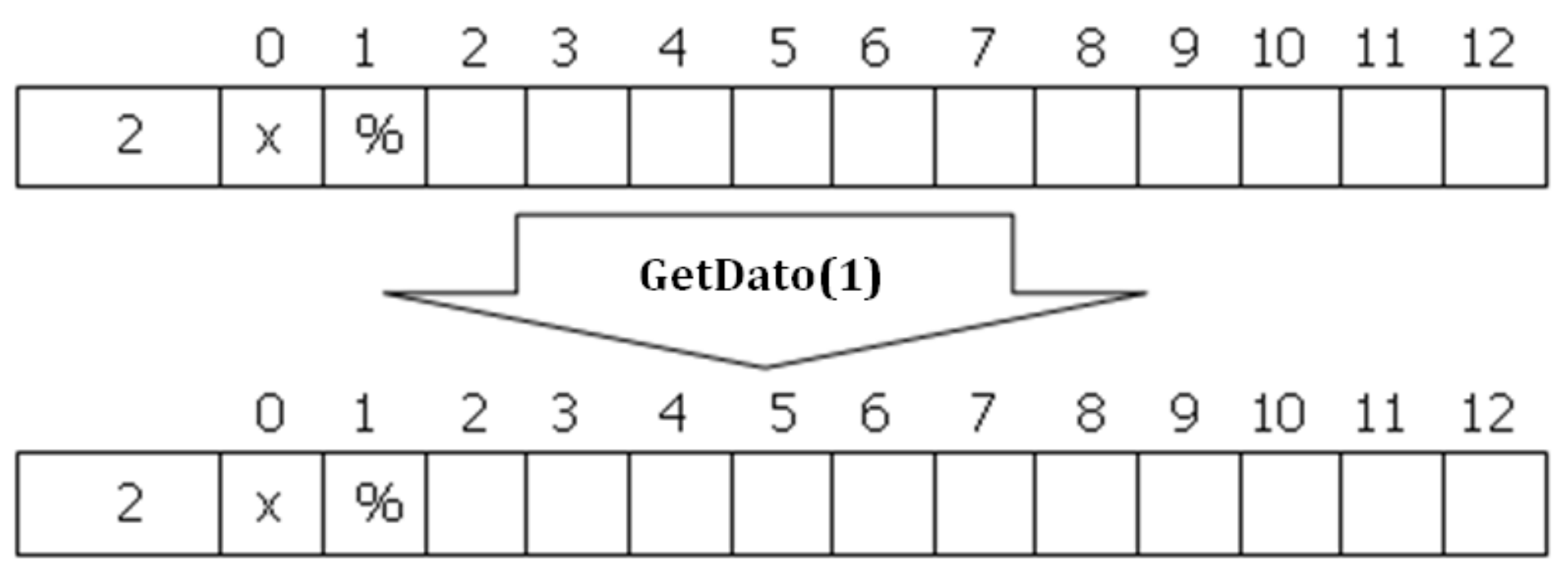
1.3 Ejemplo de Tipos de Datos Abstractos



VectorDinamico::ModifTam(NuevoTamanio)

NumDatos ← NuevoTamanio

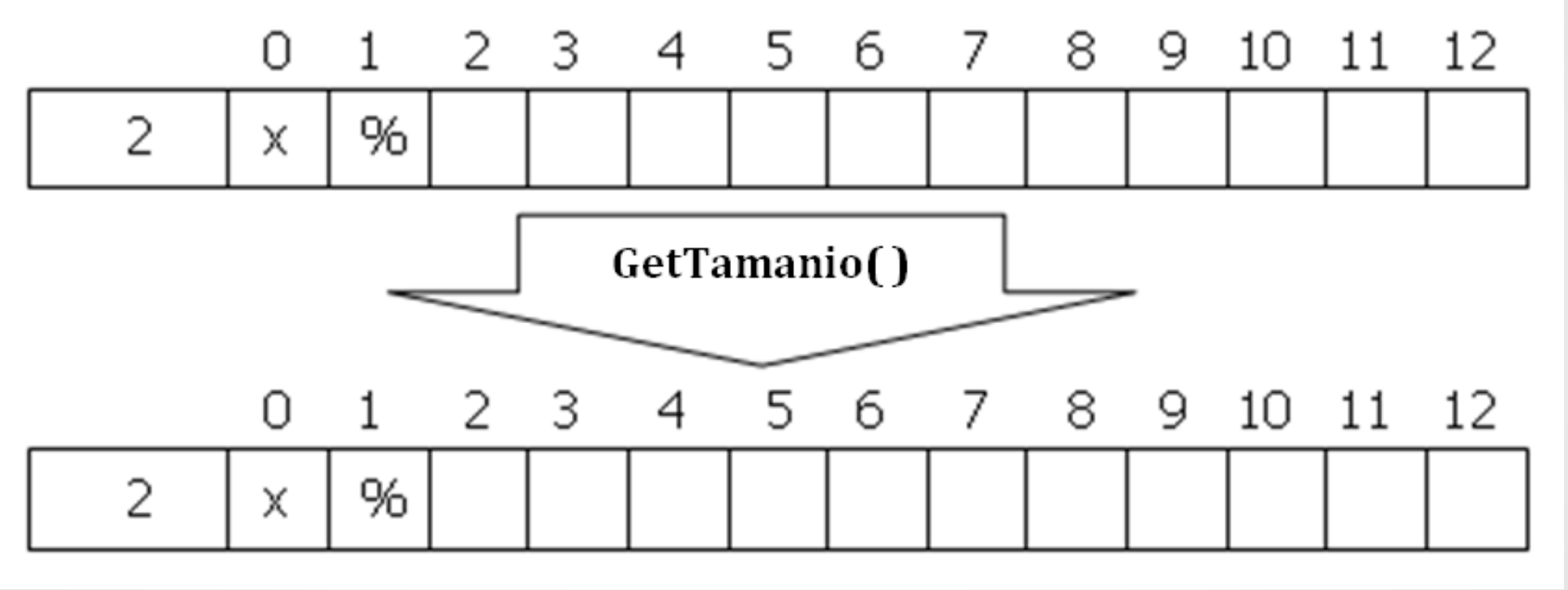
1.3 Ejemplo de Tipos de Datos Abstractos



VectorDinamico::Get(pos)

Regresa como resultado Vector[pos]

1.3 Ejemplo de Tipos de Datos Abstractos



VectorDinamico::Tamanio()

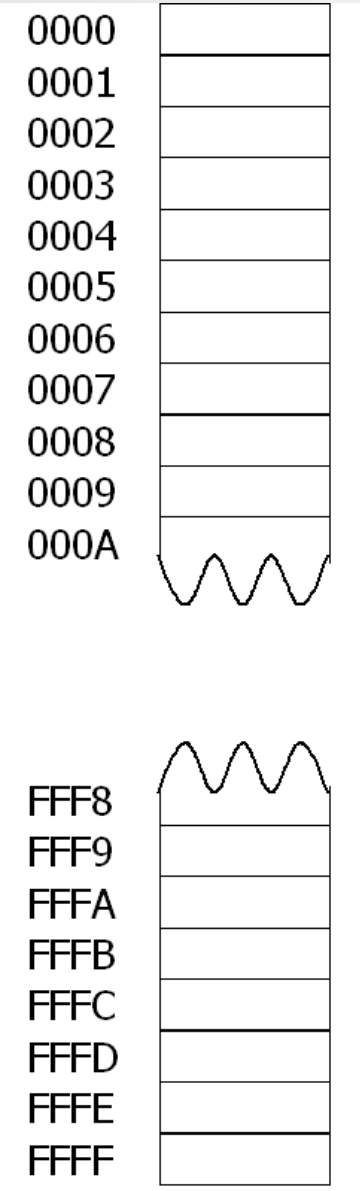
Regresa como resultado NumDatos

1.3 Ejemplo de Tipos de Datos Abstractos

Ejercicio 2 Tema 1.3

Usar la clase *VectorDinamico* para resolver el problema planteado anteriormente para registrar las lecturas de compresión de un motor.

1.4 Manejo de Memoria Estática



- La memoria se compone de lugares consecutivos.
- Para fines didácticos asumiremos que cada lugar es de un byte de capacidad.
- Cada byte tiene una dirección.
- Se acostumbra escribir la dirección en hexadecimal.

1.4 Manejo de Memoria Estática

Variables y arreglos

¿Cómo se distribuye el espacio en la memoria con una declaración como la siguiente?

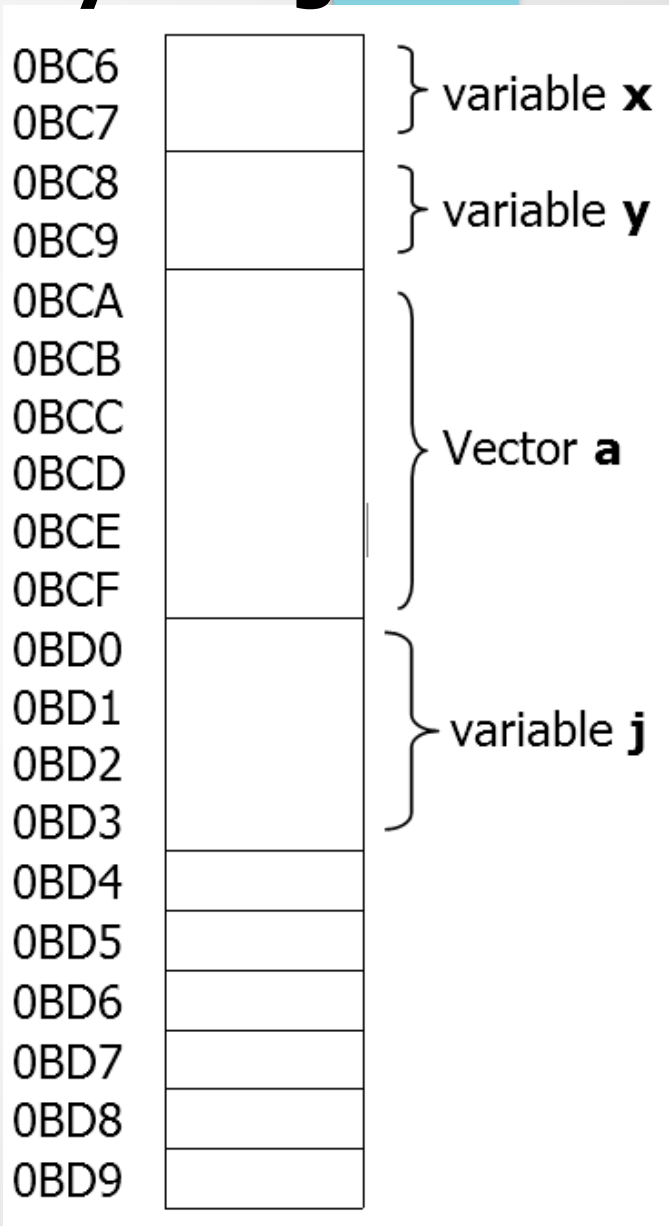
```
short x,y;
char a[6];
int j;
```

Respuesta:

Durante la compilación se reserva el espacio necesario para cada variable de acuerdo a su tipo.

Se asigna el espacio de acuerdo al orden de las variables dentro de la declaración.

Observe que *char*, aunque ocupa 2 bytes (UNICODE), en nuestro ejemplo para simplificar, lo consideramos como 1 byte (ASCII).



1.4 Manejo de Memoria Estática

```
// Pruebe con este programa C++
#include <iostream>
using namespace std;
int main()
{
    short x,y;
    char a[6];
    int j;
    cout << "Direcciones\n";
    cout << "de Memoria\n";
    cout << "x " << &x << endl;
    cout << "y " << &y << endl;
    cout << "a " << &a << endl;
    cout << "j " << &j << endl;
    system("pause");
    return 0;
}
```

1.4 Manejo de Memoria Estática

Durante el proceso de producción del código máquina, el compilador crea una tabla, de uso generalmente temporal para sustituir los nombres de variables y otros elementos por direcciones de memoria. Esta tabla se llama ***tabla de símbolos***.

Nombre de la variable	Tipo	Dirección inicial
x	short	0BC6
y	short	0BC8
a	char[0..5]	0BCA
j	int	0BD0

1.4 Manejo de Memoria Estática

Nombre de la variable	Tipo	Dirección inicial
x	short	0BC6
y	short	0BC8
a	char[0..5]	0BCA
j	int	0BD0

De acuerdo a la tabla de símbolos, ¿cómo quedaría en código objeto (al reemplazarse las variables por las direcciones de memoria) la instrucción **y = 4;**?

1.4 Manejo de Memoria Estática

Nombre de la variable	Tipo	Dirección inicial
x	short	0BC6
y	short	0BC8
a	char[0..5]	0BCA
j	int	0BD0

¿Cómo quedará el código compilado si se asigna un "*" en la posición 2 del vector ***a***?

1.4 Manejo de Memoria Estática

Nombre de la variable	Tipo	Dirección inicial
x	short	0BC6
y	short	0BC8
a	char[0..5]	0BCA
j	int	0BD0

¿Cómo quedará si no se conoce en que posición del vector irá un dato sino hasta que se ejecute el programa?

```
cin >> j
a[ j ] = '@'
```

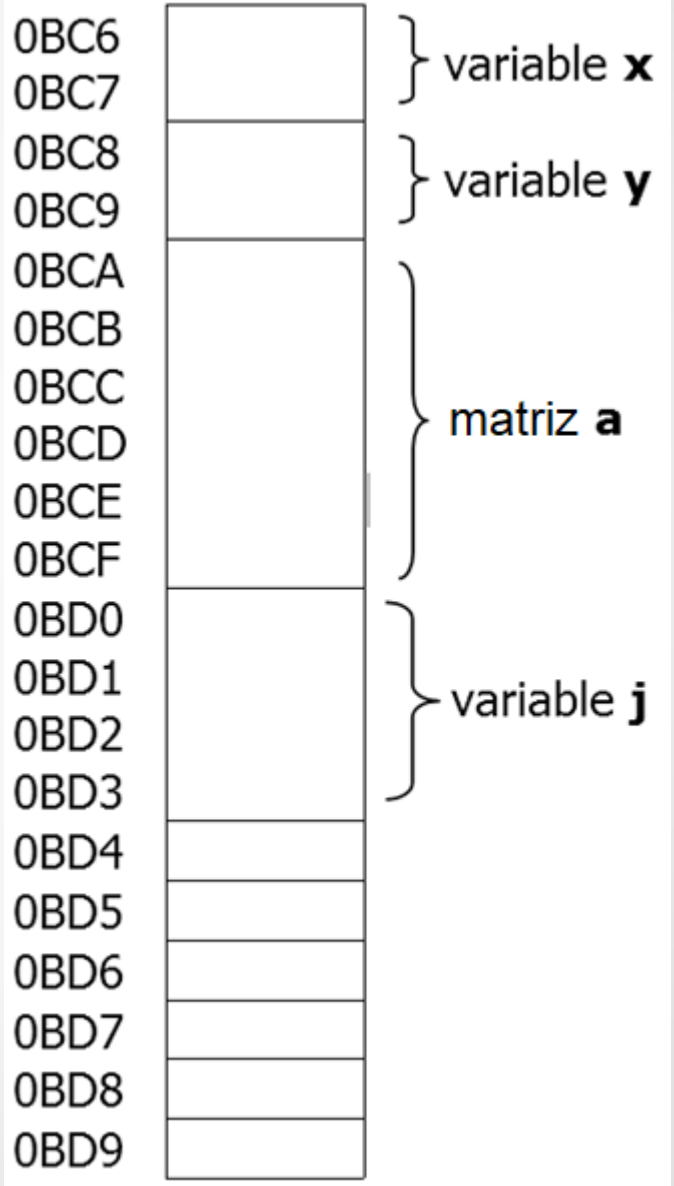

```
cin >> *0BD0
*(0BCA + *0BD0) = 0100 0000
```

1.4 Manejo de Memoria Estática

Considerar ahora que la declaración de variables es como la siguiente:

short x,y;
char a[3][2];
int j;

¿Como se guardan los elementos de la matriz a en la memoria?



1.4 Manejo de Memoria Estática

Antes de responder a la pregunta anterior, hay que escribir un programa que use una **matriz de 4x5**. El programa debe contener un menú como el siguiente:

1.- Guardar un valor en la matriz.
2.- Consultar cierta posición de la matriz.
0.- Terminar.

Guardar preguntará al usuario el valor a guardar y el renglón y columna donde se guardará.
Consultar preguntará el renglón y columna que se desea consultar e informará el valor contenido en ese lugar.

1.4 Manejo de Memoria Estática

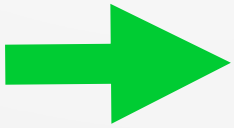
El objetivo del programa anterior es iniciar el razonamiento sobre la forma en la que el compilador esconde la complejidad del proceso al calcular la dirección equivalente unidimensional a partir de una bidimensional.

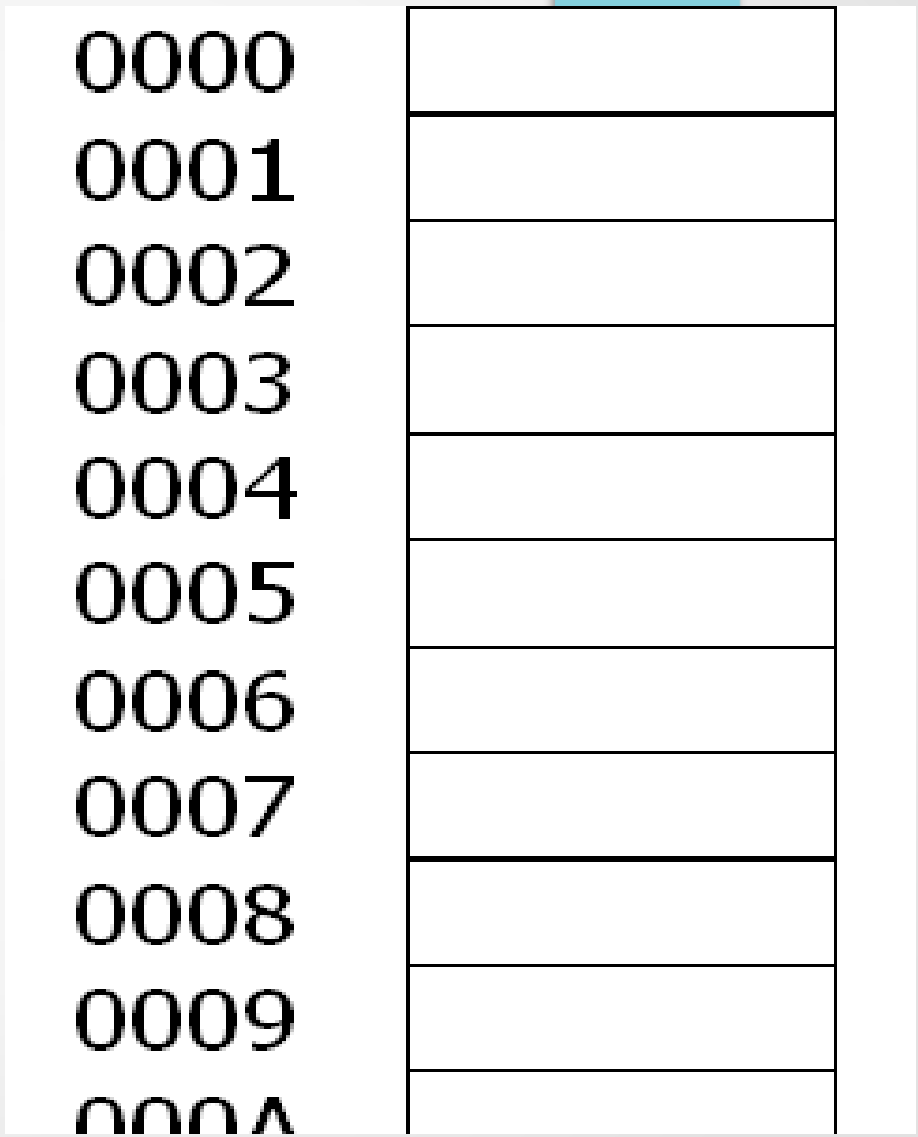
El compilador lo hace por los programadores, ahora nos toca simular ese proceso.

1.4 Manejo de Memoria Estática

La memoria es unidimensional (como un vector). Si en un lenguaje de programación se declara una matriz, el compilador contribuye aparentando que los datos se guardan en una matriz cuando en realidad se guardan en un vector (la memoria).

	0	1	2	3
0	A_{00}	A_{01}	A_{02}	A_{03}
1	A_{10}	A_{11}	A_{12}	A_{13}
2	A_{20}	A_{21}	A_{22}	A_{23}





1.4 Manejo de Memoria Estática

Ya que la memoria es unidimensional, se reducirá el problema para efectos de aprendizaje.

Se definirá un procedimiento que sirva de base para escribir un programa en C++ para **guardar los datos de lo que aparentemente es una matriz en un vector**.

1.4 Manejo de Memoria Estática

	0	1	2	3
0	A_{00}	A_{01}	A_{02}	A_{03}
1	A_{10}	A_{11}	A_{12}	A_{13}
2	A_{20}	A_{21}	A_{22}	A_{23}

0	1	2	3	4	5	6	7	8	9	10	11
A_{00}	A_{01}	A_{02}	A_{03}	A_{10}	A_{11}	A_{12}	A_{13}	A_{20}	A_{21}	A_{22}	A_{23}

1er renglón 2º renglón 3er renglón

1.4 Manejo de Memoria Estática

Nomenclatura:

i, j Dirección bidimensional de cualquiera de los elementos de la matriz.

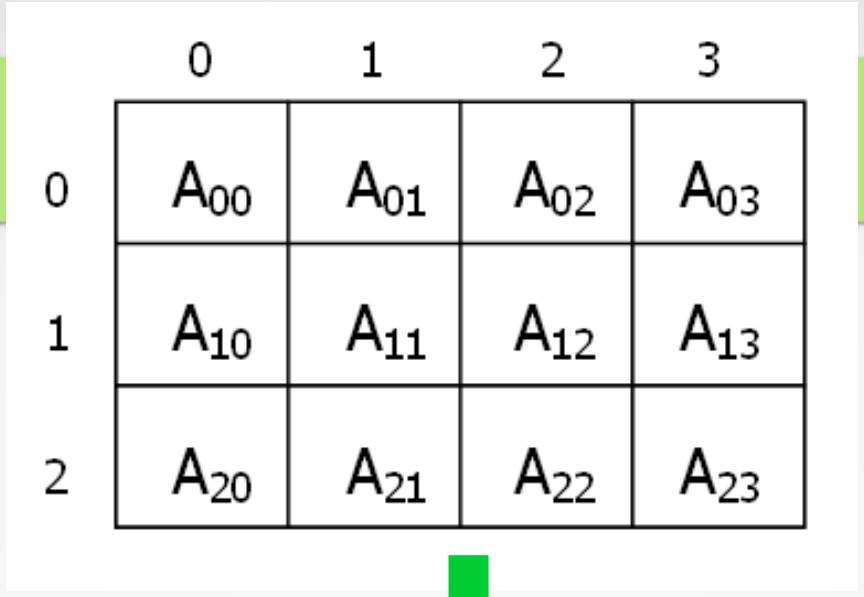
m Número total de renglones de la matriz.

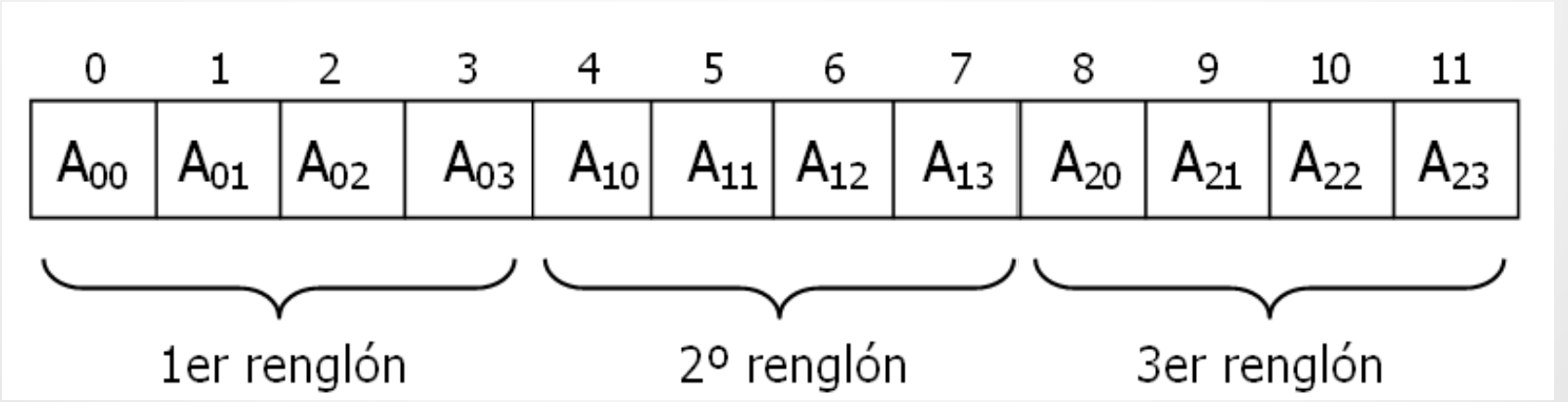
n Número total de columnas de la matriz.

p Dirección en el vector equivalente que le corresponderá a uno de los elementos de la matriz.

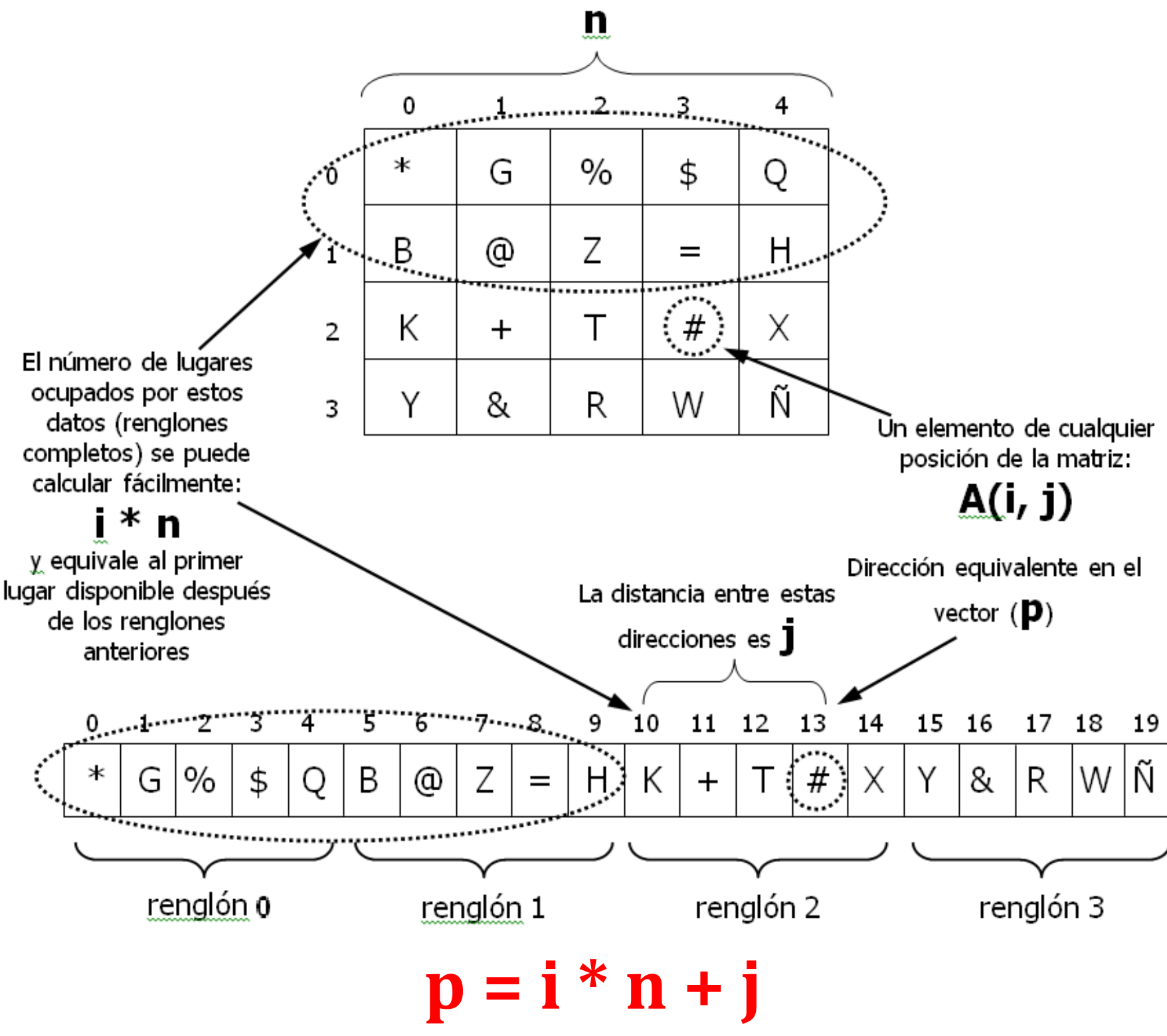
Se debe encontrar una fórmula para **p**, en función del estado de cualquier elemento de la matriz

$$\mathbf{P} = f(\mathbf{i, j, m, n})$$


0 1 2 3
0 A00 A01 A02 A03
1 A10 A11 A12 A13
2 A20 A21 A22 A23


0 1 2 3 4 5 6 7 8 9 10 11
A00 A01 A02 A03 A10 A11 A12 A13 A20 A21 A22 A23
1er renglón
2º renglón
3er renglón

$$p = i * n + j$$


n
0 1 2 3 4
0 * G % $ Q
1 B @ Z = H
2 K + T # X
3 Y & R W Ñ
El número de lugares ocupados por estos datos (renglones completos) se puede calcular fácilmente:
i * n
y equivale al primer lugar disponible después de los renglones anteriores
Un elemento de cualquier posición de la matriz:
A(i, j)
La distancia entre estas direcciones es j
Dirección equivalente en el vector (p)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
* G % $ Q B @ Z = H K + T # X Y & R W Ñ
renglón 0
renglón 1
renglón 2
renglón 3
p = i * n + j

1.4 Manejo de Memoria Estática

Ahora se debe escribir un programa que declare **un vector de 20 posiciones en vez de una matriz de 4x5** … **pero el funcionamiento del programa debe ser idéntico al anterior**:

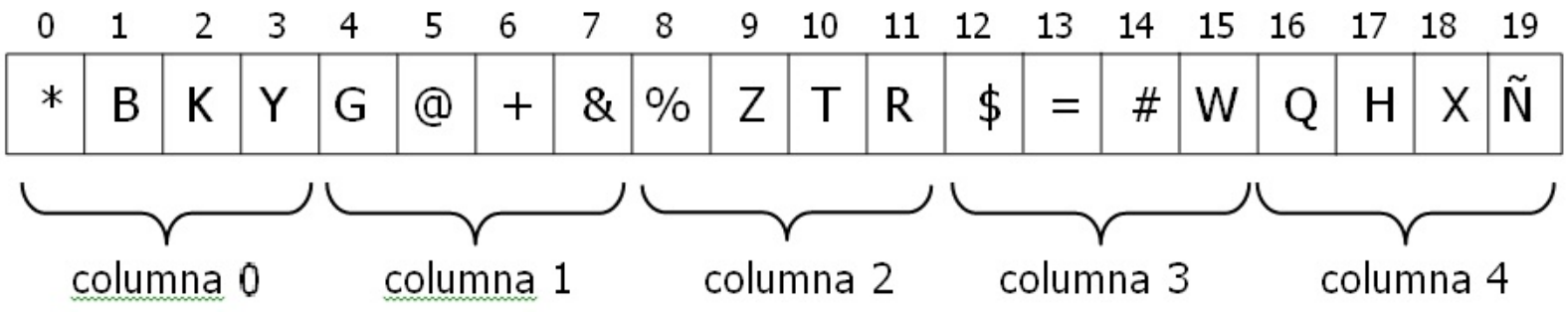
1.- Guardar un valor en la matriz.
2.- Consultar cierta posición de la matriz.
0.- Terminar.

Guardar debe preguntar la posición (renglón y columna) del lugar donde se va a guardar el valor y el valor mismo. *Consultar debe* preguntar el renglón y columna que se desea consultar e informa el valor que está guardado en ese lugar.

1.4 Manejo de Memoria Estática

Como ejercicio debe usted encontrar una fórmula para **p** siguiendo un razonamiento diferente (almacenando los elementos de la matriz **por columnas**)

	0	1	2	3	4
0	*	G	%	$	Q
1	B	@	Z	=	H
2	K	+	T	#	X
3	Y	&	R	W	Ñ



1.4 Manejo de Memoria Estática

Encontrar una fórmula para **p** con base en este otro razonamiento: **por renglones *versión 2***)

	0	1	2	3	4
0	*	G	%	$	Q
1	B	@	Z	=	H
2	K	+	T	#	X
3	Y	&	R	W	Ñ

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Q	$	%	G	*	H	=	Z	@	B	X	#	T	+	K	Ñ	W	R	&	Y

renglón 0 renglón 1 renglón 2 renglón 3

1.4 Manejo de Memoria Estática

Encontrar una fórmula para **p** con base en este otro razonamiento: **por renglones *version 3***)

	0	1	2	3	4
0	*	G	%	$	Q
1	B	@	Z	=	H
2	K	+	T	#	X
3	Y	&	R	W	Ñ

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Y	&	R	W	Ñ	K	+	T	#	X	B	@	Z	=	H	*	G	%	$	Q

renglón 3 | renglón 2 | renglón 1 | renglón 0

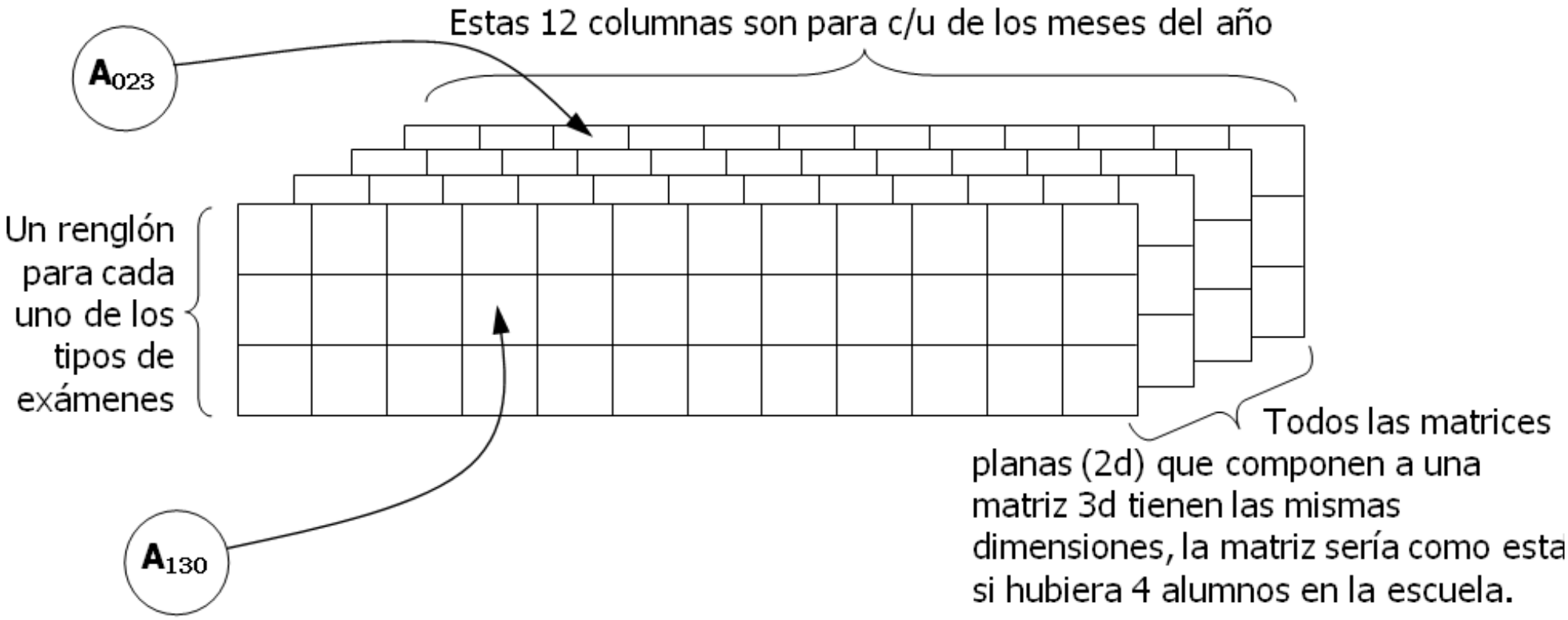
1.4 Manejo de Memoria Estática

Arreglos de tres o más dimensiones

Algunos problemas se resuelven más fácilmente usando arreglos tridimensionales, por ejemplo:

Se requiere guardar en una matriz los resultados **mensuales** de **todos los alumnos** de una escuela secundaria, obtenidos en **tres** **diferentes exámenes** (conocimientos, sicométrico y cultura general).

1.4 Manejo de Memoria Estática



La declaración sería:

char examenes[3][12][4];

1.4 Manejo de Memoria Estática

Fórmula para el Indice Equivalente

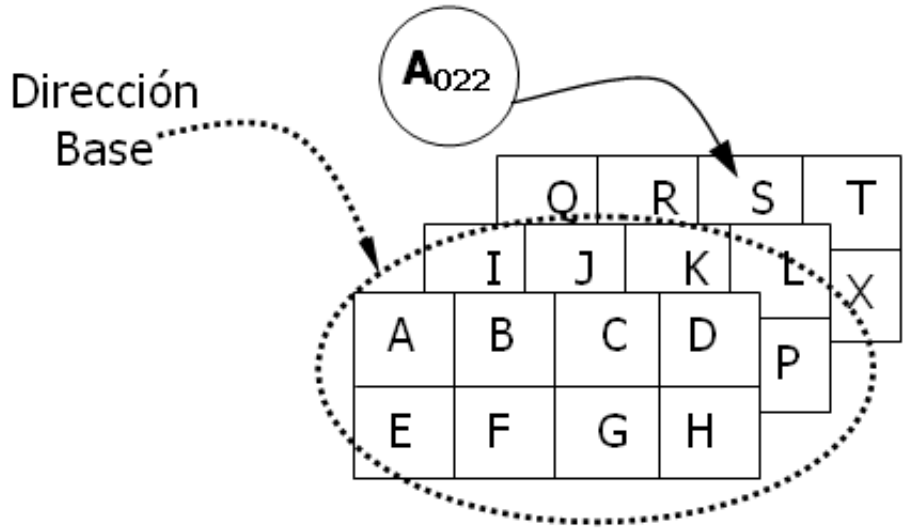
Se puede ver a los **arreglos tridimensionales** como una **proyección** de arreglos de dos dimensiones para facilitar la deducción de la fórmula.

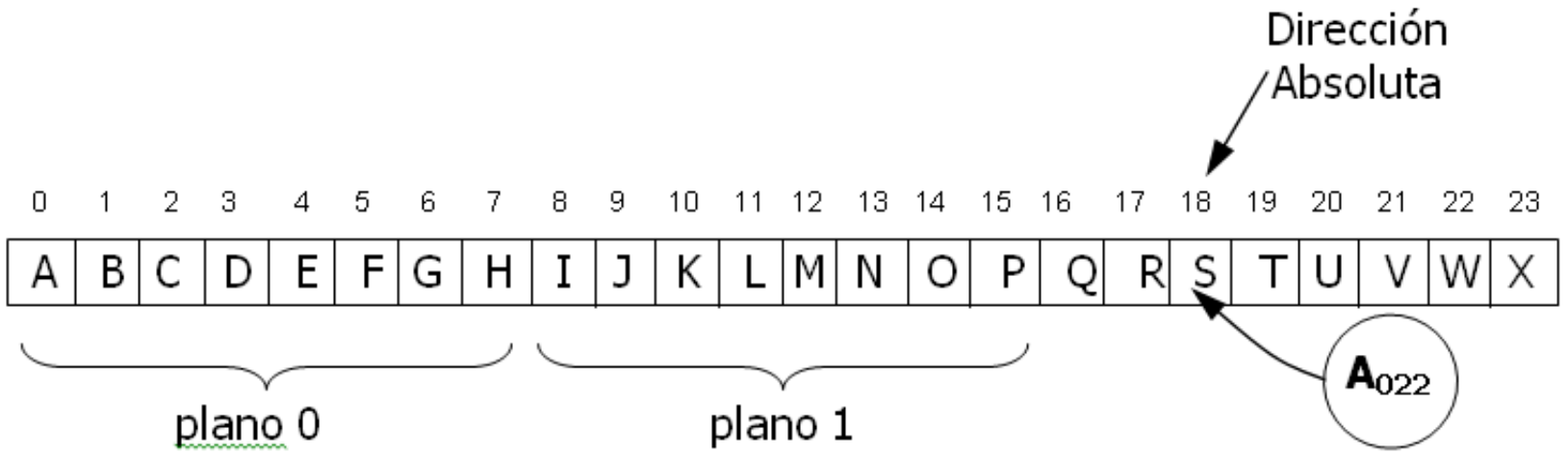
Cada elemento se identifica con una expresión de subíndices con tres componentes: $\mathbf{A}_{\mathbf{i\,j\,k}}$

i renglón donde se encuentra un elemento cualquiera.
j columna donde se encuentra un elemento cualquiera.
k plano (profundidad) donde se encuentra un elemento cualquiera.
m Número de renglones en total de cada plano.
n Número de columnas en total de cada plano.
o Número de planos del arreglo.

1.4

Apoyémonos usando un **arreglo 3d** más pequeño

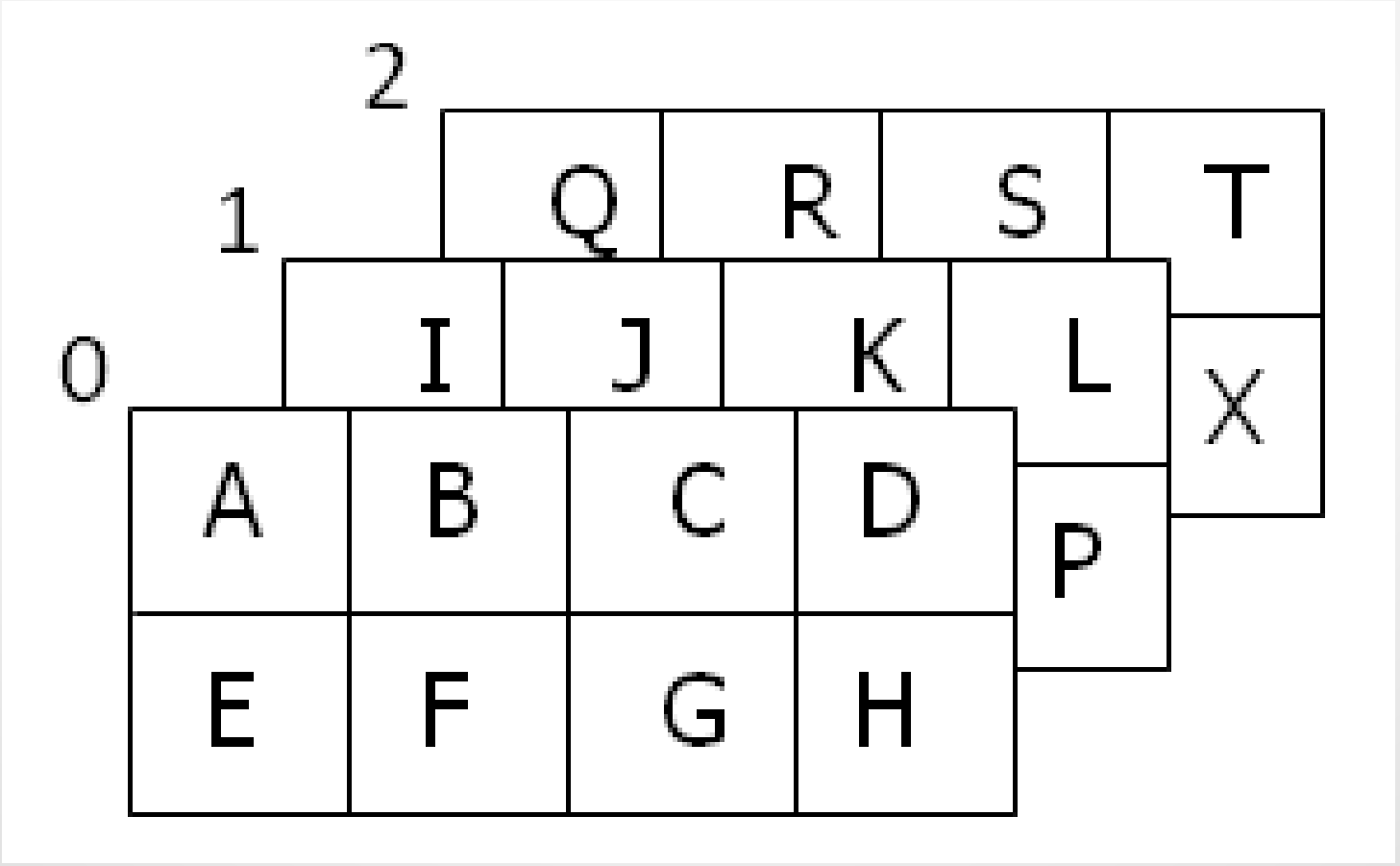




$$p = k\ m\ n + i\ n + j$$

1.4 Manejo de Memoria Estática

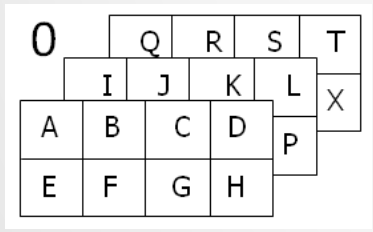
Matriz de tres dimensiones = Vector de matrices de dos dimensiones.

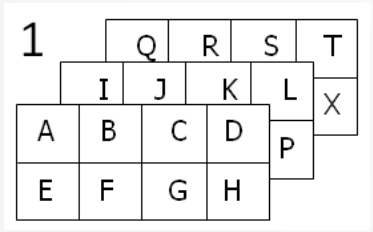


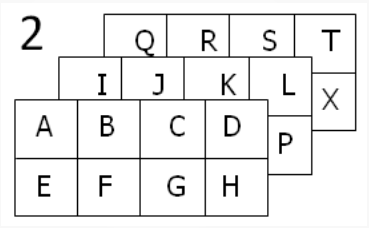
1.4 Manejo de Memoria Estática

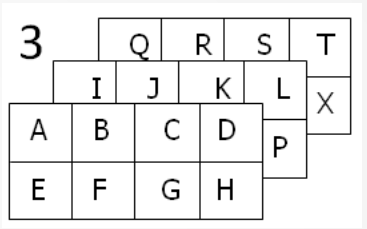
Arreglos de más de Tres Dimensiones

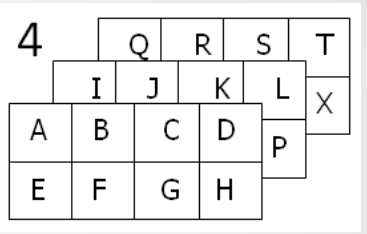
Matriz de **4** dimensiones = Vector de matrices de **3** dimensiones.











$$p = \ell m n o + \quad k m n + \acute{\imath} n + j$$

Siendo ele la posición de cualquier elemento en la 4ª dimensión.

Las fórmulas para los arreglos de más dimensiones son obtenidas por la serie correspondiente.

1.4 Manejo de Memoria Estática

Matrices poco densas regulares

10 personas participan en un juego de lanzamiento de dados en el cual se asigna un **número del 1 al 10** a cada una de ellas.

Cada jugador lanzará un **par de dados** tantas veces como indique el número que le haya tocado (el jugador **10 lanzará diez veces** y el **1 solo una vez**). Los resultados de cada lanzamiento deberán conservarse, para fines estadísticos, en una matriz en la cual los renglones representan a los jugadores y las columnas a cada lanzamiento particular.

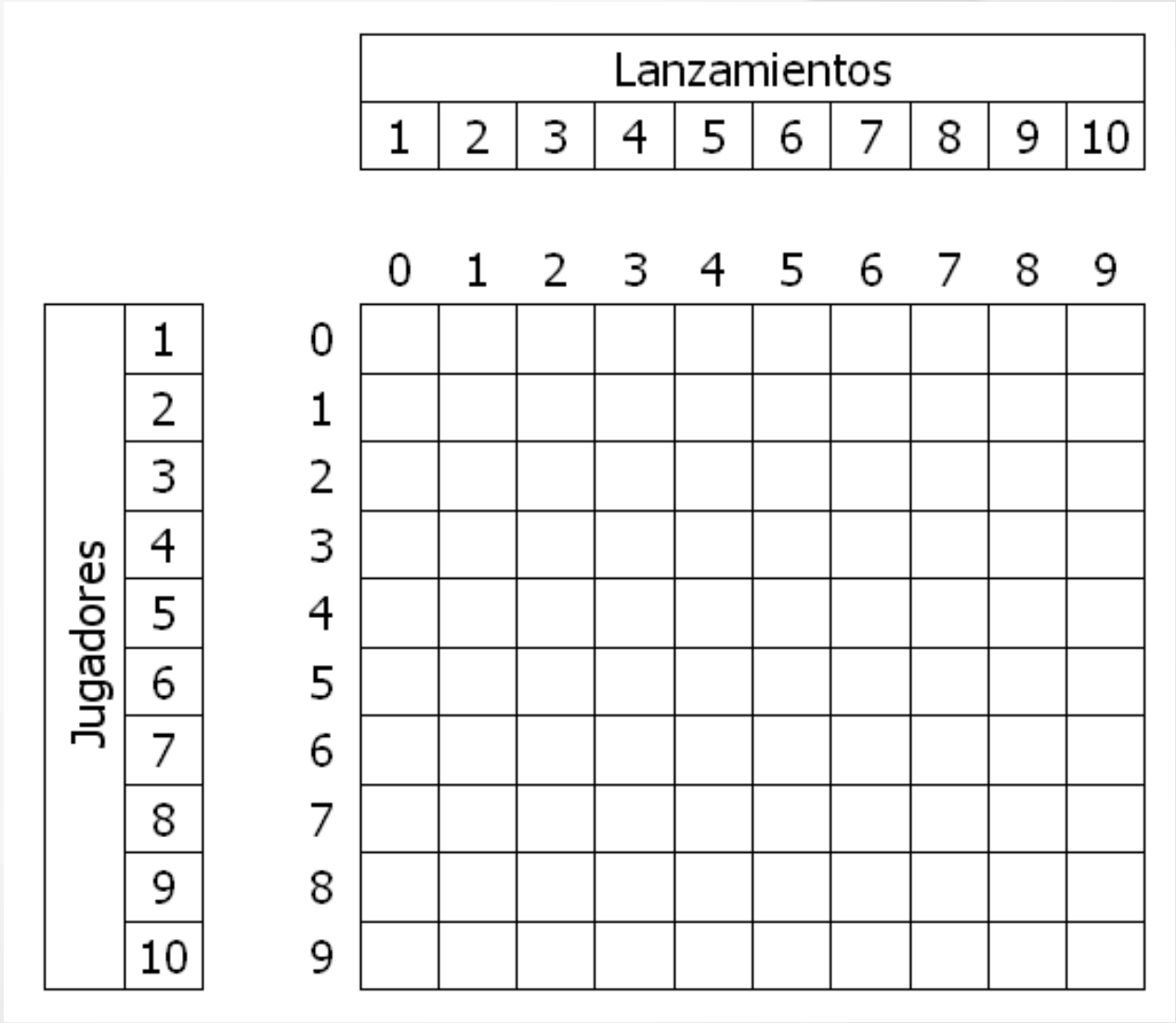
El jugador que obtenga el promedio más alto es el que ganará.

1.4 Manejo de Memoria Estática

Ejercicio 1

Escriba un programa que haga una simulación de éste juego imprimiendo, al menos, el resultado promedio de cada jugador.

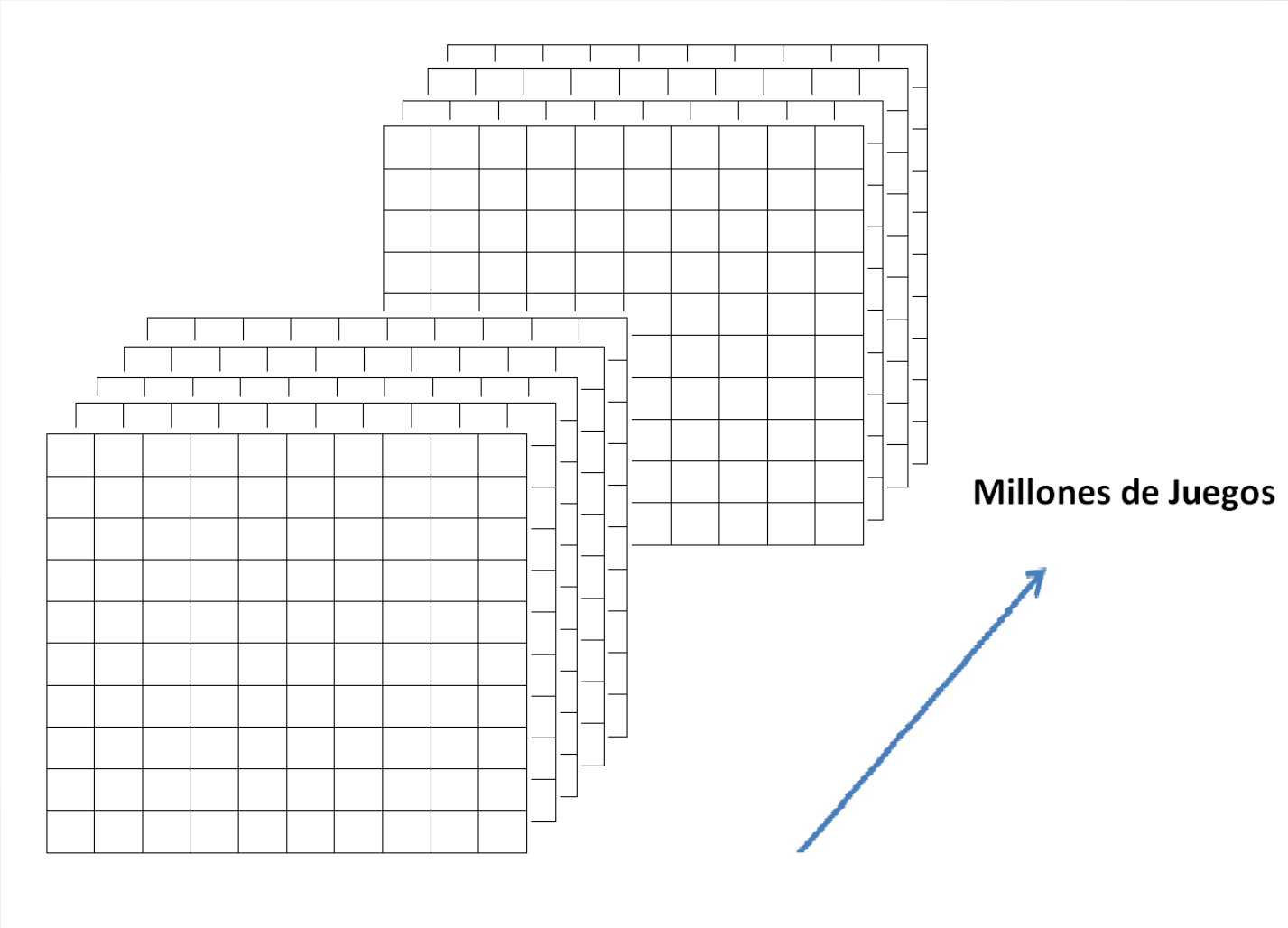
Utilice una matriz completa en la que los renglones representen a los jugadores y las columnas a los lanzamientos de los dados.



1.4 Manejo de Memoria Estática

Ejercicio 2

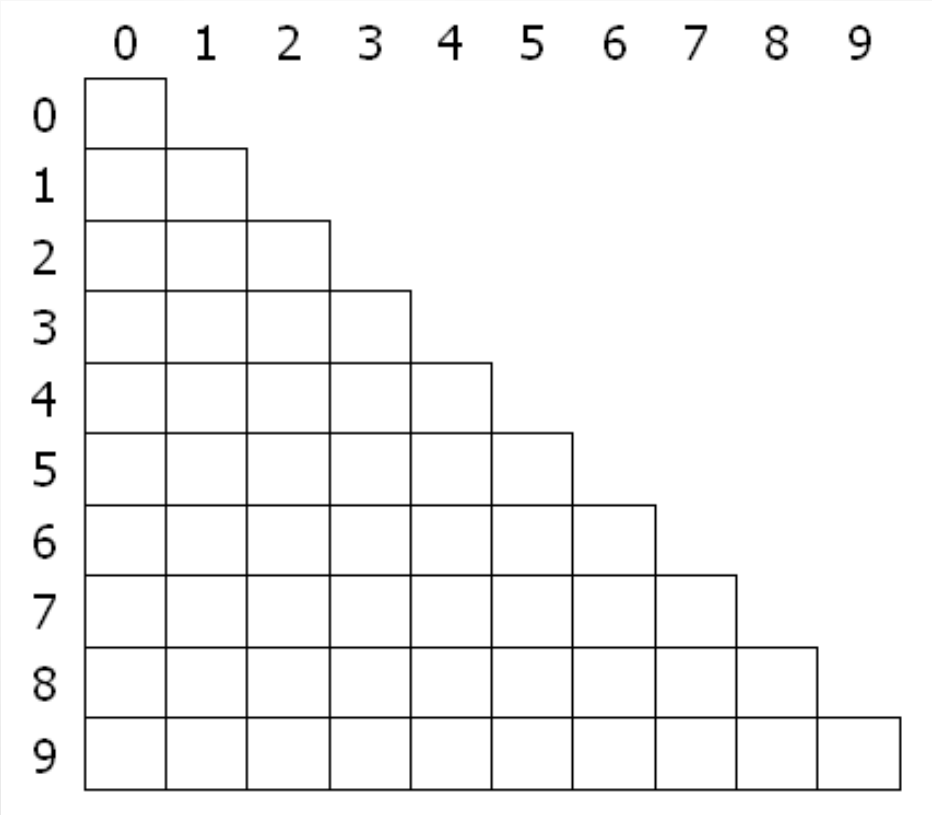
Modifique el programa para que se haga una simulación de cualquier número de juegos (hasta millones de ellos) y se conserven los resultados de todos en una matriz tridimensional.



1.4 Manejo de Memoria Estática

Matrices Triangulares Inferiores

- Son matrices cuadradas.
- Solo los datos de las direcciones **i >= j** son significativos.
- La matriz que se requiere para el ejercicio 1 es una **MTI** como la siguiente:



1.4 Manejo de Memoria Estática

Matrices Triangulares Inferiores

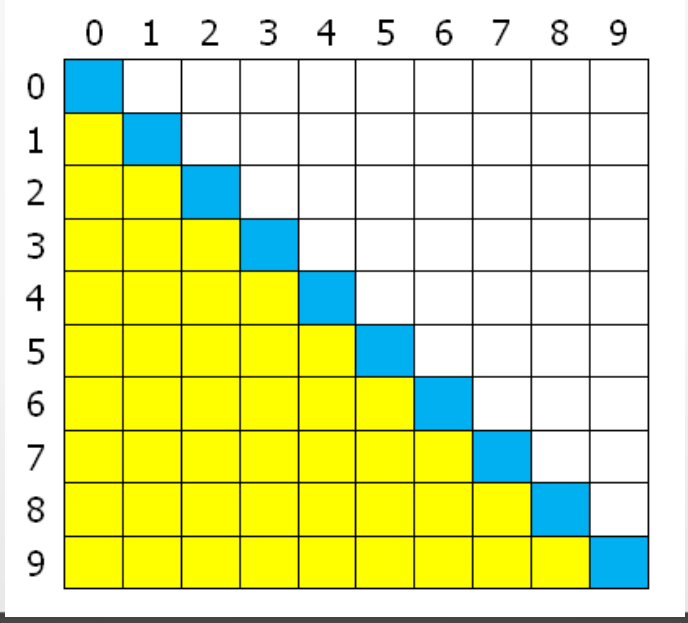
- Una **MTI** es **matriz poco densa** porque el número de elementos que contiene es pequeño con respecto a la matriz completa.
-
- Una **MTI** es una **matriz regular** porque se puede predecir su forma sin importar el tamaño.
-
- Si se requiere una **MTI de gran tamaño**, conviene guardar los datos en un **vector** para optimizar el uso de la memoria.

1.4 Manejo de Memoria Estática

Determinación del Tamaño del Arreglo Unidimensional
(para declarar un vector del tamaño justo)

Número de elementos en la diagonal principal es: **n**
Total de elementos en una matriz completa de orden n es: **n^2**

Número de lugares bajo la diagonal principal es: $\frac{n^2 - n}{2}$

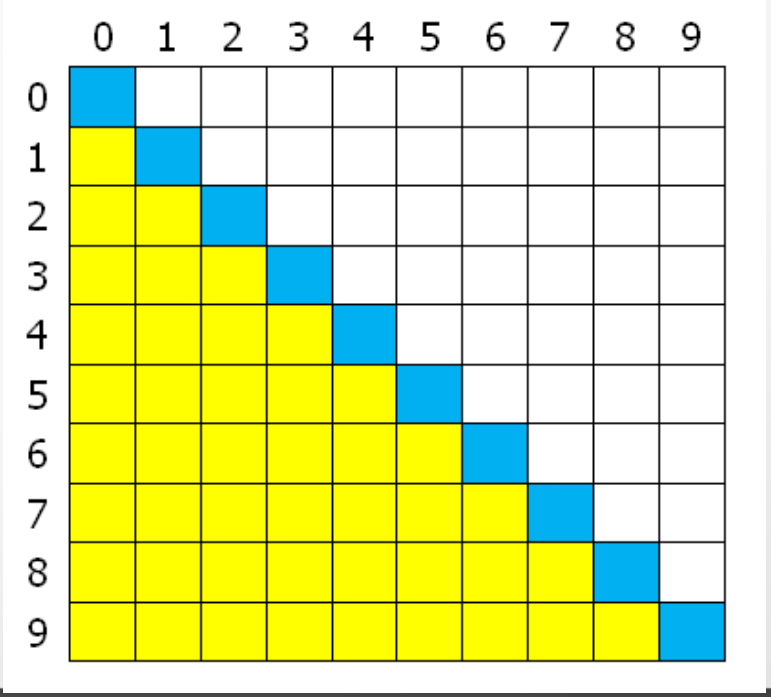


1.4 Manejo de Memoria Estática

Determinación del Tamaño del Arreglo Unidimensional

Número total de elementos en una **MT** es:

$$NTE = \frac{n^2 - n}{2} + n = \frac{n(n+1)}{2}$$



1.4 Manejo de Memoria Estática

Encontrar fórmula para el Indice Equivalente de una Matriz Triangular Inferior en un vector

	0	1	2	3	4
0	Z				
1	X	A			
2	C	S	F		
3	V	D	G	Q	
4	B	E	H	W	E

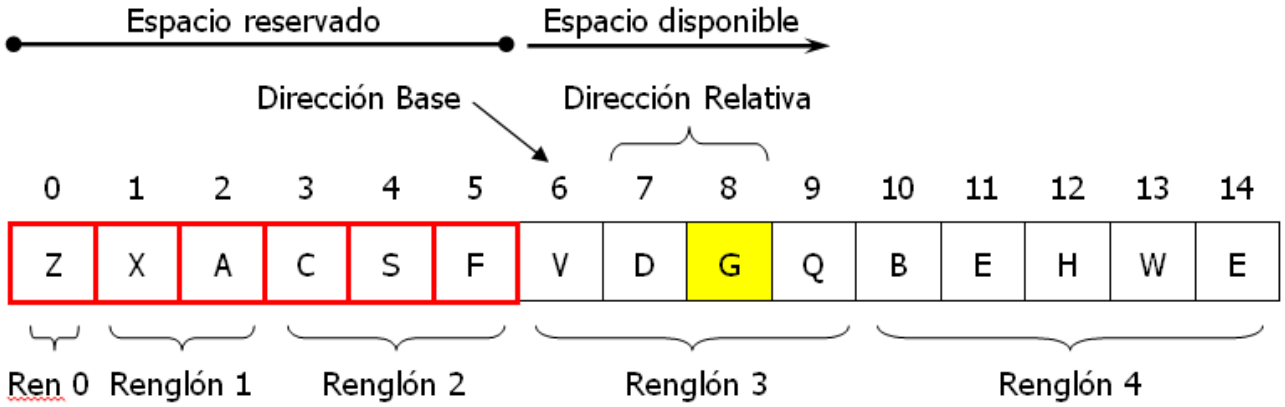
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Z	X	A	C	S	F	V	D	G	Q	B	E	H	W	E

Ren 0 Renglón 1 Renglón 2 Renglón 3 Renglón 4

1.4 Manejo de M

El elemento marcado con amarillo está en el renglón 3, y como se puede ver, encima de ese renglón hay una MTI de tamaño 3, por lo tanto, para cada elemento de la posición **i**, arriba hay una MTI de tamaño **i**, y como el número de elementos en una MTI es n(n+1)/2:

	0	1	2	3	4
0	Z				
1	X	A			
2	C	S	F		
3	V	D	G	Q	
4	B	E	H	W	E



Dirección Base, primer lugar disponible para usarse

$$DB = \frac{i\,(i+1)}{2}$$

La *Dirección Relativa* (distancia del lugar asignado desde la dirección base)

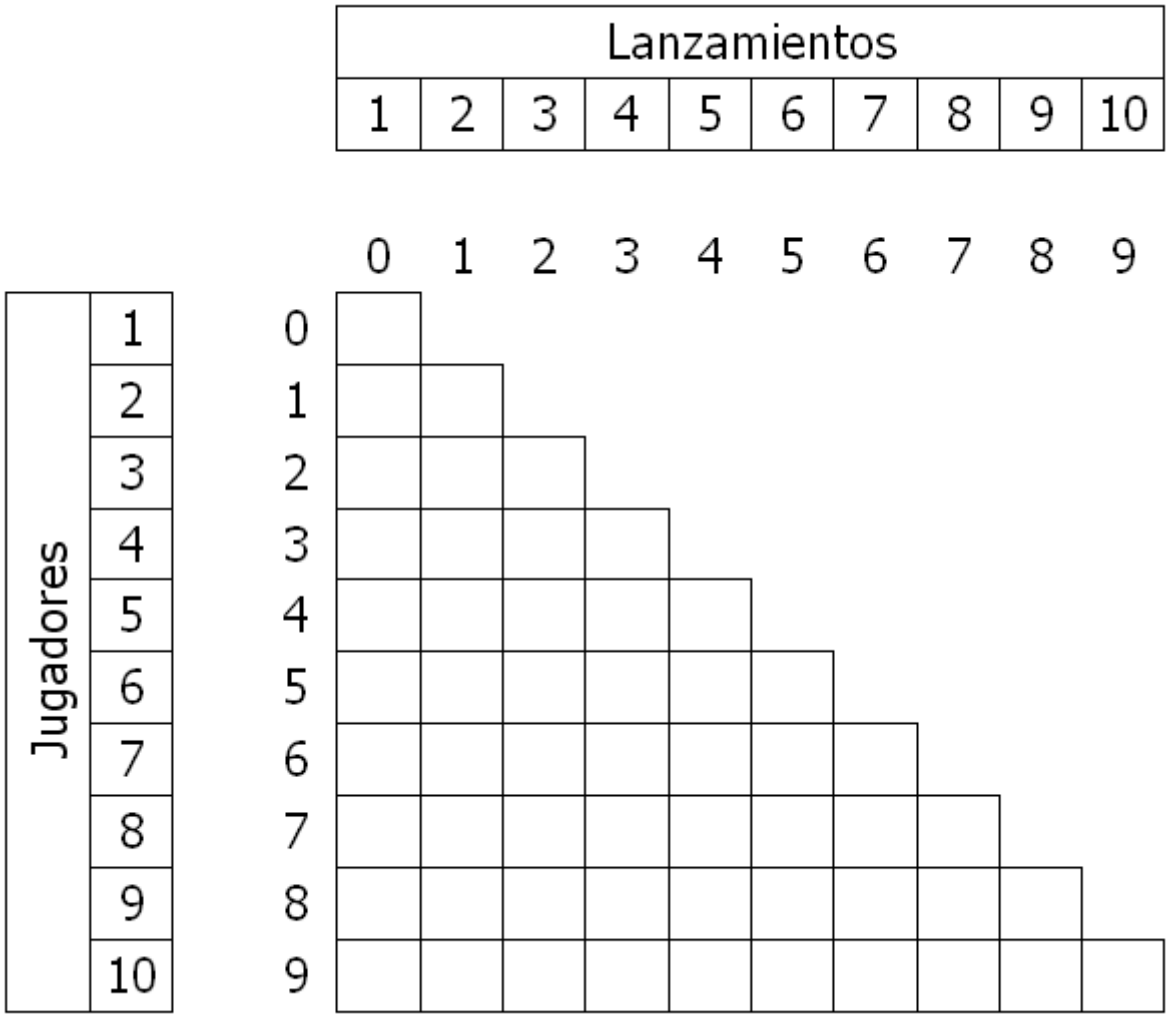
$$DR = j$$

$$p = \frac{i\,(i+1)}{2} + j$$

1.4 Manejo de Memoria Estática

Ejercicio 3

MODIFIQUE el programa que escribió para el Ejercicio 1 para la simulación del juego pero debe crear un TDA y emplear la formula que acabamos de deducir.



1.4 Manejo de Memoria Estática

Algunas aplicaciones requieren el uso de Matrices Triangulares Superiores.

Ejercicio 4

Encuentre dos fórmulas para el índice equivalente en el vector para una Matriz Triangular Superior.