

Gestión de memoria dinámica

Programación II

Alicia Garrido Alenda

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante

- Utilidad: poder almacenar una cantidad de datos variable a lo largo de la ejecución de un programa.
- La cantidad de datos que se puede almacenar se calcula en el momento en que es necesario durante la ejecución del programa, ya que el tamaño de una estructura dinámica puede cambiar durante la ejecución de un programa.

Diferencias entre vectores/matrices estáticos y dinámicos

- Las matrices y vectores *estáticos* son aquellos que se declaran con un tamaño constante, por ejemplo:

```
const int NUMFILAS = 1000, NUMCOLUMNS = 100;
int vector[NUMCOLUMNS], matriz[NUMFILAS][NUMCOLUMNS];
char tablero[7][10];
```

y dicho tamaño no puede cambiar en el curso de la ejecución.

- En las estructuras dinámicas de datos:
 - ▶ El tamaño del conjunto de datos puede cambiar cuando es necesario.
 - ▶ No tienen memoria reservada previamente: es necesario reservar memoria para almacenar datos en este tipo de estructuras.

¿Qué es un puntero?

- Es un tipo de dato que se utiliza (principalmente) para construir estructuras dinámicas de datos.
- Cada variable de tipo puntero contiene un número entero largo que representa una dirección de memoria. Esta dirección de memoria se tiene que:
 - ▶ O bien reservar para el tipo de dato que queremos almacenar en ella.
 - ▶ O bien asignarle una dirección de memoria que ya está reservada.
- En la declaración de un puntero se debe especificar el tipo de dato que contiene la dirección de memoria.
- Ejemplos de declaraciones de variables de tipo puntero en C/C++:

```
int *punteroAEntero;
char *punteroACharacter;
double **punteroApunteroAReal;

int *vectorPunterosAEntero[100];
```

- Se pueden utilizar para crear nuevas variables:

```
//declaracion que no reserva memoria
char *pc;
...
// al crear hay que reservar memoria si no ...
pc = new char[1];
// esto produciria un error de ejecucion
pc[0] = 'B';
cout << pc[0] <<endl; // escribe en pantalla B
delete []pc;          // libera la memoria
pc = NULL;            // valor de inicializacion
```

- Se pueden utilizar para acceder a otra variable ya existente:

```
...
int *ejemplo = new int[n];
...
int *pi;
pi = ejemplo; // 'pi' tiene la direccion de 'ejemplo'
// dos variables apuntando a la misma zona de memoria
```

- No existen los vectores/matrices estáticos.

```
int vEntero[10]; // Error de compilacion
```

- En Java no se pueden utilizar punteros, pero siempre es necesario reservar memoria para arrays/matrices, porque son estructuras de memoria dinámica:

```
int *vEntero; // error de compilacion
// declara la variable, no reserva memoria
int []vEntero; // Ok
// al no reservar memoria ...
vEntero[0]=1; // produce error de ejecucion
```

```
// declara la variable sin reserva de memoria
int []vEntero;
vEntero = new int[10]; // Ok
vEntero[0]=1; //Ok
```

Vectores/matrices en Java

- Las matrices en Java se pueden crear de forma incremental:
 - Se reserva memoria para el número de filas que tendrá la matriz.
 - Para cada fila se puede reservar memoria para un número distinto de columnas.
 - El número de elementos para los que tiene reservada memoria un array nos lo proporciona el atributo `length`.

```
// declara la variable, no reserva memoria
int [][]mEntero;
int j=7;
// reserva memoria para 3 filas, sin indicar columnas
mEntero=new int[3][];
// mEntero.length: numero de filas de la matriz
for(int i=0;i<mEntero.length;i++){
    //reserva memoria para j columnas en la fila i
    mEntero[i]=new int[j];
    //mEntero[i].length: numero de columnas de la fila i
    j++;
}
```

Paso de parámetros en Java


- El paso de parámetros en Java es siempre por valor.
- En el caso de variables de memoria dinámica, se pasa su dirección de memoria para poder acceder/modificar el contenido apuntado por esa dirección de memoria.
- Las modificaciones realizadas permanecerán una vez haya finalizado la ejecución del método (función).
- Las modificaciones que se hayan podido realizar en la propia dirección de memoria serán locales al método donde se realizan; una vez finalizada la ejecución del método dichas modificaciones no permanecen.

Paso de parámetros en Java

- Ejemplo de paso por parámetro de un array en Java:

```
public static void M(int []v){
    if(v!=null)
        for(int i=0;i<v.length;i++) v[i]=i;
    v=new int[5];
    for(int i=0;i<v.length;i++) v[i]=i*10;
    for (int i=0;i<v.length;i++)
        System.out.println(v[i]);
}

public static void main(String[] args){
    int vector[];
    vector = new int[2];
    vector[0] = 100;
    vector[1] = 150;
    M(vector);
    for (int i=0;i<vector.length;i++)
        System.out.println(vector[i]);
}
```

- Por tanto con variables de memoria dinámica se puede modificar su contenido, pero no la dirección a la que apunta la variable. 

Errores comunes en el uso de variables de memoria dinámica

- Utilizar la variable sin comprobar si tiene memoria reservada:

```
int pi[];
pi[0] = 76; // Error!!!
```

- No tener en cuenta los rangos al acceder a alguna posición:

```
char modifica(char[][] cs,int f,int c){
    ...
    cs[f][c]=neo; //error si f o c se salen del rango
    ...
}
...
char cadenas[][];
cadenas = new[filas][columnas];
...
bak=modifica(cadenas,i,j);
```

Errores comunes en el uso de variables de memoria dinámica

- **Recomendación:** comprobar si una variable de memoria dinámica tiene memoria reservada antes de acceder a ella comparando que es distinta de null:

```
char modifica(char[][] cs,int f,int c){
    char d=' ';
    if(cs!=null){
        ...
        if ((f>=0 && f<cs.length)&&
            (c>=0 && c<cs[f].length)){
            d=cs[f][c];
            cs[f][c]=neo;
        }
        ...
    }
    return d;
}
```

Ejercicio 1

- Implementa un método al que se le pasan por parámetro dos números enteros. Dicho método debe crear una matriz de enteros que tenga el número de filas indicado por el primer parámetro si éste es mayor que 0, en otro caso tendrá 2 filas por defecto, y el número de columnas indicado por el segundo si es mayor que 0, en otro caso tendrá 3 columnas por defecto. Una vez creada, debe rellenar dicha matriz por filas con valores consecutivos, empezando por el 1, y la devuelve.

Ejercicio 2

- Implementa un método al que se le pasa por parámetro un array de enteros. Dicho método debe devolver el número de números negativos contenidos en el array.

Ejercicio 3

- Implementa un método que recibe por parámetro un array de enteros. El método devuelve la media de los valores positivos que contiene el array.
La media se calcula como la suma de los valores dividido por el número de valores.

Ejercicio 4

- Implementa un método que recibe por parámetro un array de `String`. El método devuelve el número de cadenas del array que son iguales (ignorando diferencias entre mayúsculas y minúsculas) a la que ocupa la primera posición del array.