

# Estructuras Básicas de Algoritmos y de Datos III. Vectores

José Otero

<sup>1</sup>Departamento de informática  
Universidad de Oviedo

6 de noviembre de 2008

- 1 Concepto de vector
- 2 Declaración de vectores
- 3 Uso de vectores
- 4 Recorrido de un vector
- 5 Funciones útiles.
- 6 Matrices
- 7 Recorrido de una matriz

- 1 Concepto de vector
- 2 Declaración de vectores
- 3 Uso de vectores
- 4 Recorrido de un vector
- 5 Funciones útiles.
- 6 Matrices
- 7 Recorrido de una matriz

- Los los tipos vistos hasta el momento son *escalares*.
  - Cada variable de esos tipos alberga un único valor.
- Un vector es una agrupación de valores del mismo tipo.
  - Cada vector contiene varios valores de ese tipo.
  - La agrupación completa tiene un nombre.
  - Cada elemento del vector es accesible mediante ese nombre y un índice.
  - Es la representación de los vectores habituales en matemáticas.
- El tipo de los elementos de un vector se denomina tipo base de ese vector.
- En C++ el tamaño de los vectores definidos en la librería STL (Standard Template Library) puede cambiar en tiempo de ejecución.
  - Pueden quitarse/añadirse elementos.

- 1 Concepto de vector
- 2 Declaración de vectores**
- 3 Uso de vectores
- 4 Recorrido de un vector
- 5 Funciones útiles.
- 6 Matrices
- 7 Recorrido de una matriz

## Declaración de vectores.

- es necesario incluir el archivo de cabeceras adecuado, al principio del programa.

```
#include<vector>
```

- sintaxis:

```
vector<t_base> nombre(n_elementos,v_inicial);
```

- `t_base` es el tipo de cada elemento del vector.
- `nombre` es el nombre de la variable de tipo vector que se declara.
- `n_elementos` es el número de elementos que alberga el vector, tiene que ser mayor que cero.
- `v_inicial` es el valor inicial de todos los elementos del vector.
  - tiene que ser del tipo `t_base`.

## Sintaxis mínima:

- `vector<t_base> nombre;`

## Ejemplos:

- Un vector de enteros sin tamaño inicial.  
`vector<int> a;`
- Un vector de caracteres de tamaño inicial 7.  
`vector<char> x(7);`
- Un vector de reales de tamaño inicial 10, cada uno de ellos vale 0.5.  
`vector<float> z(10,0.5);`
- Varias declaraciones juntas.  
`vector<int> tmp,datos(3),x(20,0.0);`

- Para saber el tamaño de un vector se usa la función `size()`

```
vector<int> a(3),b;  
//muestra:3 0  
cout<<a.size()<<' '  
      <<b.size()<<endl;  
b=a;  
//muestra:3 3  
cout<<a.size()<<' '  
      <<b.size()<<endl;
```

- Se puede declarar un vector y darle tamaño más tarde asignándole un vector del tamaño deseado.

```
vector<int> a;  
int n;  
//suponer que se  
//introduce 5  
cin>>n;  
a=vector<int>(n);  
//muestra 5  
cout<<a.size();
```



Explicación a la segunda columna:

`vector<t_base>(tamano, valor_inicial)`

es una expresión de tipo vector:

- Su tipo base es `t_base`
- Su tamaño es `tamano`
- El valor inicial de cada elemento es `valor_inicial`

y se puede asignar a una variable de tipo vector del mismo tipo base.

Ejemplos:

`vector<int>(4)` es un vector de tamaño 4.

`vector<float>(7, 3.0)` es un vector de tamaño 3, cada elemento vale inicialmente 3.0.

- 1 Concepto de vector
- 2 Declaración de vectores
- 3 Uso de vectores**
- 4 Recorrido de un vector
- 5 Funciones útiles.
- 6 Matrices
- 7 Recorrido de una matriz

## Uso de vectores.

- El nombre de un vector seguido de una pareja de [ ] encerrando una *expresión* de tipo entero  $\in [0, \text{tamaño}-1]$  es el elemento que ocupa ese lugar dentro del vector.
  - Tanto como para *asignarle* valor
  - como para *usar* su valor.
  - Como si fuese una variable del tipo base del vector.
- Ejemplos:

```
vector<int> x(10);  
int a=3;  
x[0]=7;  
x[9]=a*x[0];  
//x[10]=33; ERROR  
//a=x[10]; ERROR
```

- Los vectores se pueden asignar, como cualquier variable. La variable de tipo vector a la izquierda del operador de asignación:
  - Tiene que ser construida a partir del mismo tipo que el de la derecha.
  - El tamaño pasa a ser el mismo que el del de la derecha.
  - Cada elemento será igual al que ocupe el mismo lugar en el de la derecha.

- Ejemplo:

```
vector<float> x,y(3);  
vector<int> a;  
y[0]=0.5;  
y[1]=-1.1;  
y[2]=33.0;  
x=y;  
a=x; // ERROR
```

Dos vectores *del mismo tipo base* se pueden comparar a igualdad utilizando ==

- Dos vectores son iguales si:
  - El tamaño es el mismo.
  - Contienen los mismos elementos en las mismas posiciones.

Ejemplo:

```
vector<int> a(3),b(3);
vector<float> x(x,0.0);
a[0]=1;
a[1]=3;
a[2]=-3;
//después de esta
//asignación
b=a;
//evidentemente
//son iguales
if(a==b)//es true
    //se ejecuta:
    cout<<"son iguales";
//si modificamos uno
//de los elementos
//de uno ya no son iguales
b[0]=7;
if (a==b)//es false
    //no se ejecuta
    cout<<"son iguales";
```

**NOTA:** tiene que estar definido == para el tipo base

Dos vectores *del mismo tipo base* se pueden comparar utilizando el operador `<`

- La comparación es *lexicográfica*, en la expresión `a<b`:
- Se comparan `a[0]` y `b[0]`
- Si `a[0]<b[0]` es `true` entonces `a<b` es `true`
- Si `a[0]>b[0]` es `true` entonces `a<b` es `false`
- Si `a[0]==b[0]` es `true` se comparan `a[1]` y `b[1]` y así sucesivamente
- Si la longitud de `a` es menor que la de `b` y todos los elementos de `a` están al principio de `b` en el mismo orden, `a<b` es `true`, en caso contrario `false`

**NOTA:** tiene que estar definido `<` para el tipo base

```

vector<int> a(3),b(3),      c[0]=1;
                        c[1]=2;
                        c[2]=3;
                        c[3]=3;
a[0]=1;
a[1]=2;
a[2]=3;
b[0]=1;
b[1]=3;
b[2]=3;
if(a<b)//true
    //se ejecuta
    cout<<"a < b";

```

**NOTA:** el comportamiento del resto de operadores se deduce del de estos dos, por ejemplo  $a \leq b$  es equivalente a  $a < b \mid \mid a = b$ ,  $a \neq b$  es equivalente a  $!(a = b)$ , etc.

- 1 Concepto de vector
- 2 Declaración de vectores
- 3 Uso de vectores
- 4 Recorrido de un vector**
- 5 Funciones útiles.
- 6 Matrices
- 7 Recorrido de una matriz



- Salvo la asignación y las comparaciones, no existen operaciones definidas para vectores *completos*.
- Cualquier otra operación sobre un vector hay que descomponerla en acciones individuales *sobre cada elemento*.
- Un algoritmo realiza el *recorrido* de un vector cuando opera con todos los elementos del vector, uno tras otro y en orden

- Una forma cómoda de realizar el recorrido de un vector es usar un bucle `for`

```
vector<tipo> nombre;  
//dar tamaño...asignar valores  
...  
//si i no se usa FUERA del for  
//se puede declarar ahí  
for(int i=0;i<nombre.size();i++)  
{  
    //hacer lo que sea con el elemento i  
    ...nombre[i]...  
}
```

## ■ Lo mismo del último al primero

```
vector<tipo> nombre;  
//dar tamaño...asignar valores  
...  
for(int i=nombre.size()-1;i>=0;i--)  
    {  
    //hacer lo que sea con el elemento i  
    ...nombre[i]...  
    }
```

## Ejemplo:

- Pedir un vector de reales por el teclado:

```
//tamano tiene que tener
//valor definido
vector<float> a(tamano);
for(int i=0;i<a.size();i++)
    cin>>a[i];
```

## Ejemplo:

- Mostrar un vector de reales por la pantalla:

```
//previamente los valores  
//de los elementos se han  
//asignado  
for(int i=0;i<a.size();i++)  
    cout<<a[i]<<' ';
```

## Ejemplo:

- Sumar los elementos de un vector de reales:

```
//previamente los valores
//de los elementos se han
//asignado
float suma=0;
for(int i=0;i<a.size();i++)
    suma+=a[i]; //suma=suma+a[i];
```

## Ejemplo:

- Sumar vectorialmente dos vectores de reales:

```
//previamente los valores
//de los elementos se han
//asignado
//se suponen de == tamaño
for(int i=0;i<a.size();i++)
    c[i]=a[i]+b[i];
```

## Ejemplo:

- Copiar los elementos de un vector en otro en orden inverso:

```
//previamente los valores
//de los elementos se han
//asignado
b=vector<float>(a.size());
for(int i=0;i<a.size();i++)
    b[a.size()-1-i]=a[i];
```



- 1 Concepto de vector
- 2 Declaración de vectores
- 3 Uso de vectores
- 4 Recorrido de un vector
- 5 Funciones útiles.**
- 6 Matrices
- 7 Recorrido de una matriz

En esta sección veremos funciones útiles para:

- Saber si un vector está vacío (saber si el tamaño es cero).
- Vaciar un vector (hacer que tenga tamaño cero).
- Insertar un elemento en el vector.
  - Por el final.
  - En cualquier otro lugar.
- Eliminar un elemento.
  - Por el final.
  - En cualquier otro lugar.
- Cambiar el tamaño de un vector.

## Saber si un vector está vacío:

- Para saber si un vector está vacío se puede usar `size`:  
`nombre.size()==0` es `true` cuando está vacío.
- Se puede utilizar la función `empty()`:  
`nombre.empty()` es `true` cuando está vacío y es más eficiente.

## Vaciar un vector:

- Se puede asignar un vector vacío (poco ortodoxo):  
`nombre=vector<tipo_base>()`;
- Se puede usar `clear`:  
`nombre.clear()`; hace que `nombre` tenga tamaño cero.

Añadir un elemento por el final:

- La función `push_back` añade un elemento a un vector, incrementando en uno el tamaño.

```
nombre.push_back(expresion_tipo_base);
```

Esta operación es *relativamente eficiente*.

Insertar un elemento en otra posición:

- Se usa la función `insert`

```
nombre.insert(nombre.begin()+indice,  
expresion_tipo_base);
```

Inserta el valor de `expresion_tipo_base` en la posición `indice` incrementando en uno el tamaño de `nombre`

Los elementos siguientes se desplazan hacia el final: esta operación es *ineficiente*.

**NOTA:** `nombre.begin()` es el principio del vector en memoria.

## Aplicación:

- Pedir un vector por el teclado sin definir el tamaño ni pedirlo por el teclado.

```
vector<tipo_base> x;  
tipo_base valor;  
char c;  
do{  
    cout<<"Introduce elemento:";  
    cin>>valor;  
    x.push_back(valor);  
    cout<<"Otro?";  
    cin>>c;  
}while(c!='n' && c!='N');
```

El proceso termina cuando después del valor se pulsa una tecla distinta de n o N.

Eliminar un elemento por el final:

- La función `pop_back` elimina un elemento de un vector, decrementando en uno el tamaño.

```
nombre.pop_back();
```

Esta operación es *eficiente*.

Eliminar un elemento en otra posición:

- Se usa la función `erase`

```
nombre.erase(nombre.begin()+indice);
```

Elimina el elemento que está en la posición `indice` decrementando en uno el tamaño de `nombre`

Los elementos siguientes se desplazan hacia el principio:  
esta operación es *ineficiente*.

## Cambiar el tamaño de un vector:

- Se puede usar la función `resize`  
`nombre.resize(tamaño, valor_defecto);`
  - `tamaño` es el tamaño que tendrá el vector después de ejecutarse `resize`
  - `valor_defecto` es el valor que tendrán los nuevos elementos del vector (cuando `tamaño` es mayor que el tamaño actual). Si no se especifica, para los valores numéricos será 0.
- Si `tamaño` es mayor que el tamaño inicial, los elementos se añaden por el final.
- Si `tamaño` es menor que el tamaño inicial, los elementos sobrantes se eliminan comenzando por el final.

Esta operación puede ser *ineficiente* cuando se usa para *aumentar* el tamaño.

- 1 Concepto de vector
- 2 Declaración de vectores
- 3 Uso de vectores
- 4 Recorrido de un vector
- 5 Funciones útiles.
- 6 Matrices**
- 7 Recorrido de una matriz



- No se ha establecido ninguna restricción para el tipo base de un `vector`.
- El tipo base puede ser también `vector`.
- Un `vector<vector<tipo_base> >` es una matriz.
- Reparar en el espacio en blanco entre los dos signos `>`. Es obligatorio ponerlo.

- Declaración (puede ir en una misma línea, no cabía):  

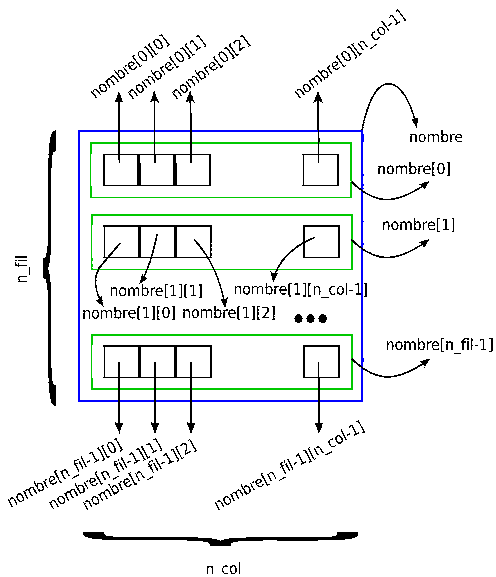
```
vector<vector<tipo_base> >  
nombre(n_fil, vector<t_base>(n_col, v_ini));
```

  - `tipo_base` es el tipo de cada elemento de la matriz.
  - `n_fil` es el número de filas.
  - `n_col` es el número de columnas.
  - `v_ini` es el valor inicial de cada elemento.
- Sintaxis mínima:  

```
vector<vector<tipo_base> > nombre;
```

Si nombre es un `vector<vector<tipo_base> >`

- `nombre.size()` es el número de filas.
- `nombre[exp_entera]` con `exp_entera` en `[0,nombre.size()-1]` es una fila completa y es de tipo `vector<tipo_base>`.
- `nombre[exp_entera].size()` con `exp_entera` en `[0,nombre.size()-1]` es el tamaño de esa fila (número de columnas).
- `nombre[exp_entera1][exp_entera2]` con `exp_entera1` en `[0,nombre.size()-1]` y `exp_entera2` en `[0,nombre[exp_entera1].size()-1]` es el elemento de la matriz que está en la posición `[exp_entera1, exp_entera2]`.



- Las filas no tienen por que tener todas el mismo tamaño.
  - 1 Declarar la matriz y definir *sólo* el número de filas.
  - 2 Para cada fila, asignar el tamaño.
    - Pidiéndolo por el teclado.
    - O de cualquier otra forma.

```
vector<vector<tipo_base> > nombre(nfil);  
int tam;  
for (int i=0;i<nombre.size();i++)  
{  
    //dar valor a tam  
    ...tam...  
    nombre[i]=vector<tipo_base>(tam);  
}
```

**NOTA:** También se puede usar `push_back` dentro de un bucle para insertar elementos en `nombre[i]`

Ejemplo: matriz triangular, sólo tiene elementos por debajo de la diagonal principal.

```
vector<vector<t_base> > a(n_fil);  
for (int i=0;i<a.size();i++)  
    a[i]=vector<t_base>(i+1);
```

a[0] tiene tamaño 1.

a[1] tiene tamaño 2.

a[2] tiene tamaño 3.

...

a[a.size()-1] tiene tamaño a.size()

- 1 Concepto de vector
- 2 Declaración de vectores
- 3 Uso de vectores
- 4 Recorrido de un vector
- 5 Funciones útiles.
- 6 Matrices
- 7 Recorrido de una matriz**

Para recorrer una matriz se pueden utilizar dos bucles `for` anidados.

```
vector<vector<tipo_base> >  
nombre(nfil, vector<tipo_base>(ncol));  
for (int i=0;i<nombre.size();i++)  
    for (int j=0;j<nombre[i].size();j++)  
        {  
            //procesar el elemento i j  
            ...nombre[i][j]...  
        }
```

En este caso se recorre de izquierda a derecha y de arriba a abajo.

**NOTA:** válido para cualquier matriz.



En este caso se recorre de arriba a abajo y de izquierda a derecha

```
vector<vector<tipo_base> >
nombre(nfil, vector<tipo_base>(ncol));
for (int j=0;j<nombre[0].size();j++)
    for (int i=0;i<nombre.size();i++)
        {
            //procesar el elemento i j
            ...nombre[i][j]...
        }
```

**NOTA:** sólo válido para matrices rectangulares.

## Pedir una matriz por el teclado conocidas sus dimensiones

```
int nfil, ncol;
cin>>nfil>>ncol;
vector<t_base> v(nfil,vector<t_base>(ncol));
for (int i=0;i<v.size();i++)
    for (int j=0;j<v[i].size();j++)
        {
            cout<<"elemento ["<<i<<"]["<<j<<"]=";
            cin>>v[i][j];
        }
```

**NOTA:** el uso de `push_back` como se vio en vectores unidimensionales para la lectura de matrices es trivial.

## Mostrar una matriz por la pantalla

```
for (int i=0;i<v.size();i++)
    {
    for (int j=0;j<v[i].size();j++)
        cout<<v[i][j]<<' ';
    cout<<endl;
    }
```

Sumar dos matrices. Tiene que estar definida la suma para `t_base`.

Se suponen matrices del mismo tamaño.

```
//se declaran a, b ,c
//se da valor a a y b
...
for (int i=0;i<a.size();i++)
    for (int j=0;j<a[i].size();j++)
        c[i][j]=a[i][j]+b[i][j];
```

## Sumar todos los elementos de una matriz.

```
//se declara a
//se da valor a a
...
//t_base es int, float...
t_base suma=0;
for (int i=0;i<a.size();i++)
    for (int j=0;j<a[i].size();j++)
        suma=suma+a[i][j];
```

Calcular medias por filas almacenando el resultado en el elemento correspondiente de un vector.

```
//declarar a
//dar tamaño
//dar valor
...
//declarar el vector resultado
vector<t_base> media(a.size());
t_base suma;
for (int i=0;i<a.size();i++)
    {
    suma=0;
    for (int j=0;j<a[i].size();j++)
        suma=suma+a[i][j];
    //esto está fuera del for j
    media[i]=suma/a[i].size();
    }
```

## Calcular medias por columnas, sólo válido para matrices rectangulares

```
//declarar a
//dar tamaño
//dar valor
...
//declarar el vector resultado
vector<t_base> media(a[0].size());
t_base suma;
for (int j=0;j<a[0].size();j++)
    {
    suma=0;
    for (int i=0;i<a.size();i++)
        suma=suma+a[i][j];
    //esto está fuera del for j
    media[i]=suma/a.size();
    }
```

## Trasponer una matriz cuadrada.

```
//declarar a
//dar tamaño
//dar valor
...
t_base tmp;
for (int i=0;i<a.size()-1;i++)
    for (int j=i+1;j<a[i].size();j++)
        {
            tmp=a[i][j];
            a[i][j]=a[j][i];
            a[j][i]=tmp;
        }
```



## Trasponer una matriz rectangular.

```
//declarar a
//dar tamaño
//dar valor
...
vector<vector<t_base> >
b(a[0].size(),vector<t_base>(a.size()));
for (int i=0;i<a.size();i++)
    for (int j=0;j<a[i].size();j++)
        b[j][i]=a[i][j];
//ahora podríamos hacer
a=b;
```