Programación de Ordenadores

Ingeniería Química

Curso 2007 - 2008

David Pelta

Depto de Ciencias de la Computación e I.A.

Universidad de Granada

6

VECTORES Y MATRICES

- Motivación.
- Operaciones Básicas.
- Ejemplos
- Paso de vectores como parámetros.
- Matrices
- Ejemplos Avanzados

Motivación

- En casi todos los problemas, es necesario mantener una relación entre variables diferentes o almacenar y referenciar variables como un grupo. Para ello, los lenguajes ofrecen tipos más complejos que los vistos hasta ahora.
- Un tipo de dato compuesto es una composición de tipos de datos simples (o incluso compuestos) caracterizado por la organización de sus datos y por las operaciones que se definen sobre él.
- Una matriz es un tipo de dato, compuesto de un número fijo de componentes del mismo tipo y donde cada una de ellas es directamente accesible mediante un índice. Un vector es una matriz de una dimensión.

Un Ejemplo

```
/* Programa para realizar la media de TRES notas, y
   calcular cuántos alumnos han superado la media */
#include <iostream.h>
int main(){
int cuantos=0;
double nota1, nota2, nota3, media;
leer nota1, nota2, leer nota3;

media = (nota1+nota2+nota3)/3.0;
if (nota1 > media) cuantos++;
if (nota2 > media) cuantos++;
if (nota3 > media) cuantos++;

cout << "Media Aritmetica = " << media << endl;
cout << cuantos << " alumnos han superado la media"
   << endl; }
```

¿Qué sucede si queremos almacenar las notas de 150 alumnos?

Número de variables imposible de sostener y recordar

Solución: Introducir un *tipo de dato nuevo* que permita representar dichas variables en una única estructura de datos

	Notas[1]	Notas[2]	Notas[3]
NOTAS	3.8	5.7	6.5

El tipo nuevo será una matriz o vector.

Se declara una variable ***notas*** de tipo vector y cada componente se accede en la forma:

notas[indice]

Declaración

<tipo><identificador>[<CantValores>];

donde

<tipo> indica el tipo de dato común a todas las componentes del vector.

<CantValores> determina el número de componentes del vector. Puede ser un literal entero o una constante entera.

Ejemplos

```
double notas[3];
bool casados[40];
int temp[100];
```

Operaciones Básicas

Acceso:

<identificador>[indice]

0 ≤ Indice ≤ CantValores-1

Dada la declaración:

double v[3];

Cada componente del vector es una variable de tipo double.

v[2] , v[0] son variables de tipo double

v['3'] , v[9] no son accesos correctos.

Operaciones Básicas

Asignación de Valores:

<identificador>[indice] = <Expresion>

Donde *<Expresión>* ha de ser del mismo tipo que el definido en la declaración del vector (o al menos compatible).

V[0] = 3.45 *v[1] = sqrt(256);*

No se permiten asignaciones globales sobre todos los elementos del vector, salvo en el momento de la definición de la variable, la inicialización. Las asignaciones se deben realizar componente a componente.

Mecanismos de Inicialización

C++ permite inicializar una variable de tipo vector, en la declaración.

int v[3]={4,5,6};
inicializa v[0]=4, v[1]=5 y v[2]=6

int v[7]={3,5};
inicializa v[0]=3 v[1]=5 y el resto se inicializan a cero.

int v[7]={0};
inicializa todas las componentes a cero.

int v[]={1,3,9};
automáticamente el compilador asume *int v[3]*

Recorriendo Vectores

La lectura y/o escritura de los valores de un vector, se realiza componente a componente (salvo en la inicialización).

Para recorrer las componentes, utilizaremos el siguiente esquema:

```
const int TAM = 20;
int main()
{int vec[TAM];
  int i;
  for(i=0; i < TAM; i=i+1)
   { manipulo vec[i];
    }
}
```

Búsqueda de un Elemento (I)

```
int main(){
const int TAM = 100;
int V[TAM];
int i, posicion, elemento;
bool Encontrado;

// supongamos que V ya tiene valores
cout << "Ingrese el elemento a buscar:";
cin >> elemento;
Encontrado=false;
for(i=0; i < TAM; i=i+1)
 {if (V[i] == elemento)
    {posicion=i; Encontrado=true;
    }
  }
if (Encontrado)
   cout << "Encontrado en la posición " << posicion << endl;
else
   cout << "No Encontrado" << endl;
}
```

Búsqueda de un Elemento II

```
int main(){
const int TAM = 100;
int V[TAM];
int i, posicion, elemento;
bool Encontrado;

// supongamos que V ya tiene valores
cout << "Ingrese el elemento a buscar:";
cin >> elemento;
Encontrado=false; i=0;
while ((i<TAM) && !Encontrado)
   {if (V[i] == elemento)
       {posicion=i; Encontrado=true;
       }
    else
       i++;
    }
if (Encontrado)
   cout << "Encontrado en la posición " << posicion << endl;
else
   cout << "No Encontrado" << endl;
}
```

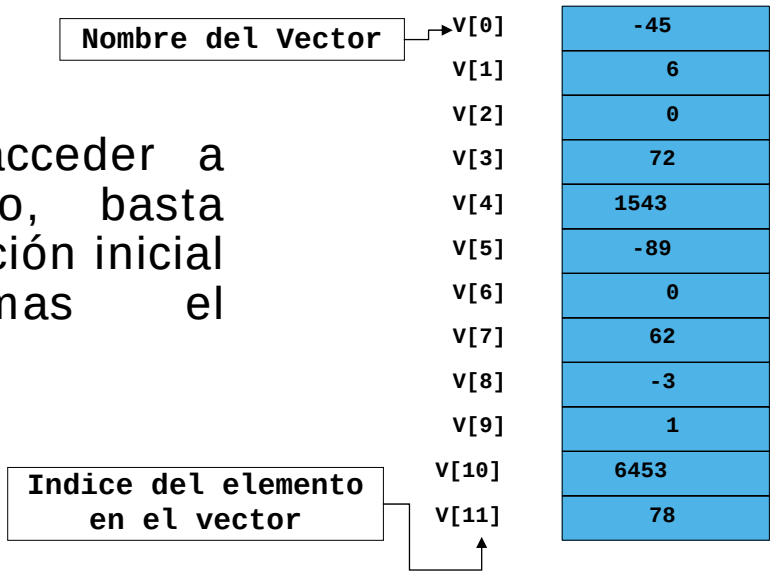
Frecuencia de un Elemento

```
int main(){
const int TAM = 100;
int V[TAM];
int i, contador, elemento;
bool Encontrado;
// supongamos que V ya tiene valores
cout << "Ingrese el elemento:";
cin >> elemento;
Contador = 0;
for(i=0; i<TAM; i=i+1)
  {if (V[i] == elemento)
      contador = contador + 1;
  }

cout << “El elemento “<< elemento
     << “ aparece “ << contador << “ veces.” << endl;
}
```

Vectores como Parámetros

La declaración de un vector produce la reserva de un conjunto consecutivo de posiciones de memoria (tantas como componentes tenga el vector).

Notar que para acceder a cualquier elemento, basta con conocer la posición inicial del vector mas el desplazamiento.



Vectores como Parámetros

En el parámetro actual, indicaremos el nombre del vector sin los corchetes.

El paso de un vector

int Medidas[24]

a una función denominada *Calculo*, será similar a

Calculo(Medidas, 24);

Por lo general, siempre se pasa el tamaño del vector como parámetro.

Vectores como Parámetros

Los vectores se pasan por referencia (aunque no se indique &)

- El nombre del vector es la dirección del primer elemento
- La función “conoce” donde está almacenado el vector
- La función modifica las posiciones de memoria originales

Por defecto, los valores individuales del vector, son pasados por valor

Vectores como Parámetros

El prototipo de la función será:

void Calculo(int V[], int nroElems)

Como antes, los nombres son opcionales en el prototipo

int V[] puede ser int []

int nroElems puede ser int

Ejemplo

```
/*   multiplica   por   2   cada
componente del vector*/
#include <iostream>
void doble(int[], int);
void mostrarVector(int[], int);
int main()
{const int TAM = 12;
 int vector[TAM];
 int i;
 for(i=0; i < TAM; i=i+1)
    vector[i] = i;
 mostrarVector(vector, TAM);
 doble(vector, TAM);
 mostrarVector(vector, TAM);
 system("PAUSE");
 return(1);
}
```

```
void doble(int T[], int N)
    {int i;
     for(i=0; i < N; i=i+1)
       T[i] = T[i] * 2;
    }
```

```
void mostrarVector(int T[], int N)
    {int i;
     for(i=0; i < N; i=i+1)
       cout << T[i] << ",";
     cout << endl;
    }
```

Ejercicios

- Dado un vector V de enteros, con elementos *V[i]*, *i* $\in$ *[0,N]*, construir un vector *A* de forma tal que $A[i]=\sum_{j=0}^{i} V[j]$
- Dado un vector C de tamaño N, donde cada componente C[i] representa el i-esimo coeficiente de un polinomio de grado N, construir una función que permita evaluarlo para un valor X dado.

$$ax^0 + bx^1 + ... + px^n$$

- Dados dos vectores X e Y de enteros, ambos de tamaño N, implemente una función que permita calcular el producto escalar

$$\sum_{i=0}^{N-1} x[i] * y[i]$$

Ordenación de un Vector I

Inicial	97	64	74	65	7	43	66	6	28	94
Paso 0	6	64	74	65	7	43	66	97	28	94
Paso 1	6	7	74	65	64	43	66	97	28	94
Paso 2	6	7	28	65	64	43	66	97	74	94
Paso 3	6	7	28	43	64	65	66	97	74	94
Paso 4	6	7	28	43	64	65	66	97	74	94
Paso 5	6	7	28	43	64	65	66	97	74	94
Paso 6	6	7	28	43	64	65	66	97	74	94
Paso 7	6	7	28	43	64	65	66	74	97	94
Paso 8	6	7	28	43	64	65	66	74	94	97
Final	6	7	28	43	64	65	66	74	94	97

Ordenación de un Vector II

```
#include <iostream>
int buscaMenor(int v[], int posI, int posF);
int swap(int & a, int & b);
int main()
{const int TAM = 20;
 int v[TAM];
 int i,posMenor;

 //asumimos que el vector tiene valores
 for(i=0;i<TAM-1;i++)
  {posMenor = buscaMenor(v,i,TAM);
      swap(v[i],v[posMenor]);
   }

for(i=0;i<TAM;i++)
  cout << v[i] << " ";

return(0);
}
```

Ordenación de un Vector III

```
int swap(int & a, int & b)
{int tmp;
 tmp = a;
 a = b;
 b = tmp;
}

int buscaMenor(int v[], int posI, int posF)
{int min,j;

 min = posI;
 for(j=posI+1;j<posF;j++)
   {if(v[j] < v[min])
      min = j;
   }
return(min);
}
```

Matrices: definición y uso

La definición de una estructura multidimensional también es posible.

<tipo><identificador>[d1] [d2]…[dn];

Solo debemos indicar los valores de cada dimensión. Por ejemplo

int a[3][4]

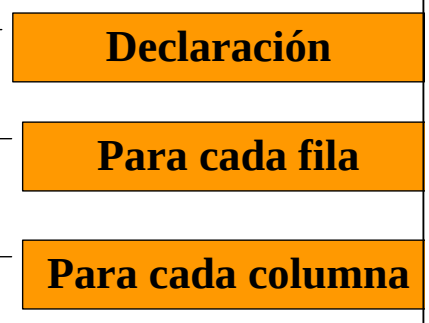
define una estructura *"a"* de 3 filas y 4 columnas.

	Columna 0	Columna 1	Columna 2	Columna 3
Fila 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Fila 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Fila 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Matrices: definición y uso

El acceso y manipulación de componentes sigue el mismo formato que en los vectores.

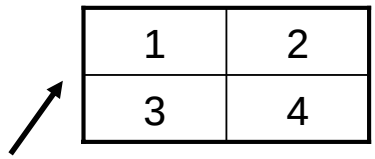
```
const int FILA = 10;
const int COL = 20;

int main()
{int mat[FILA][COL];
  int i,j;
  for(i=0; i < FILA; i=i+1)
   {for(j=0; j < COL; j=j+1)
          manipulo mat[i][j];
    }
}
```



Matrices: definición y uso

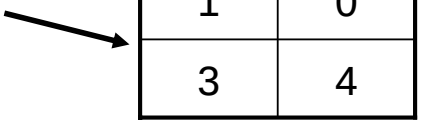
Inicialización

```
int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };
```



1	2
3	4

```
int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };
```



1	0
3	4

Paso de Matrices a Funciones

En el parámetro actual, indicaremos el nombre de la matriz sin los corchetes.

El paso de una matriz

int tabla[24][12]

a una función denominada *Calculo*, será similar a

Calculo(tabla, 24,12);

Por lo general, siempre se pasan todos los valores de las dimensiones

Paso de Matrices a Funciones

El prototipo de la función será:

void Calculo(int V[][12], int nroFil, int nroCol)

Si la matriz tiene mas de dos dimensiones, entonces en el prototipo deben indicarse los valores máximos para todas excepto la primera.

void tresD(int v[][10][15], int d1, int d2, int d3)

Ejemplo

```
#include <iostream>
const int FILA = 4;
const int COL = 8;
void cargaMatriz(int M[][COL],
                 int f, int c);
int main()
{ int mat[FILA][COL];
  int i,j;
  cargaMatriz(mat, FILA, COL);
  for(i=0; i < FILA; i=i+1)
   {cout << endl;
    for(j=0; j < COL; j=j+1)
     cout << mat[i][j] << "\t";
    }
}
```

```
void cargaMatriz(int M[][COL],
       int f, int c)
{int i,j;
  for(i=0; i < f; i=i+1)
   {cout << endl;
    for(j=0; j < c; j=j+1)
                M[i][j] = i*COL
+ j;
    }
}
```

Ejercicios

1. Diseñe un procedimiento que permita, dada una matriz y sus dimensiones, mostrar los elementos de la triangular superior

2. Utilizando una matriz cuyos elementos son de tipo *char* sugiera una manera de implementar graficos de barras

3. Diseñe un esquema para calcular el histograma de un conjunto de valores