Alfredo Vellido : www.lsi.upc.edu/~avellido

Fonaments d'Informàtica

Semana 5, p2. Tipos estructurados // tuplas+tablas

Tipos estructurados. Tablas, matrices

Tablas bidimensionales. Matrices / Recorrido de matrices

- Las iteraciones típicas de las tablas unidimensionales suelen hacer uso de un solo bucle, p.ej.:

```
int T[10]; // declaramos tabla de 10 enteros
    // Llenamos la tabla con ceros
for (int k = 0; k < 10; k++) { T[k] = 0;}
```

Propuesta de ejercicio 4

Supongamos la declaración de la variable tabla:

```
int M[10][20];
```

Escribir código para:

- Llenar la primera fila de la matriz con el valor -1.
- Llenar la última columna de la matriz con el valor 5.

Tablas bidimensionales. Matrices: Recorrido de matrices

- Las iteraciones típicas de las tablas unidimensionales suelen hacer uso de un solo bucle, p.ej.:

```
int T[10]; // declaramos tabla de 10 enteros
    // Llenamos la tabla con ceros
for (int k = 0; k < 10; k++) { T[k] = 0;}
```

- **Propuesta de ejercicio 5**

  Escribir una **acción** que reciba **3 parámetros**: una **matriz** de 8 x 12 enteros; un entero ***k*** (que será una fila); y un valor **val** (un entero). La acción ha de llenar la fila ***k*** de la matriz con el valor **val**.

Tablas bidimensionales. Matrices: Recorrido de matrices

- ... Sin embargo, si hacemos uso de una sola iteración con las matrices no podremos recorrer todas sus casillas. El siguiente código:

```
int M[10][10];
for (int k = 0; k < 10; k++) {M[k][k] = 0;}
```

… **no llena toda la matriz!!!.**

- **Propuesta de ejercicio 6:** ¿Qué parte de la matriz llena, realmente?

Tablas bidimensionales. Matrices: Recorrido de matrices

- La existencia de filas y columnas hace necesaria la definición de **una iteración dentro de otra** para poder recorrer todas las casillas de una matriz. Por ejemplo, el siguiente código llena la matriz `M` de ceros:

```
for (int i=0; i < 10; i++) // itera las filas
{ for (int j=0; j < 10; j++) // para cada fila, itera columnas
    M[i][j] = 0;
}
```

- Este tipo de "doble bucle" es **MUY** común en matrices.

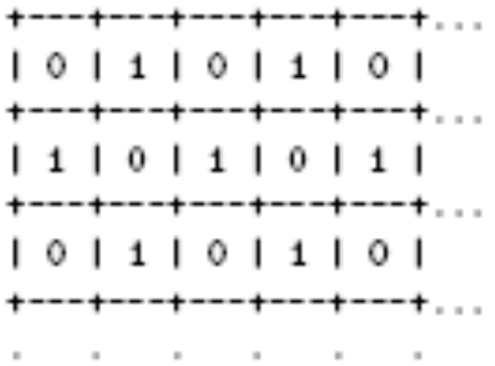
Tipos estructurados. Tablas, matrices

Tablas bidimensionales. Matrices:
Recorrido de matrices

Propuesta de ejercicio 7

Declara una matriz de valores Booleanos de 10 filas y 10 columnas y escribe una acción que la llene como si fuera un tablero de damas, como ilustra el dibujo de la derecha. En él, el 0 simboliza *false* y el 1 *true*:

```
typedef bool matBool[10][10];
...
matBool damas;
rellena_tablero(damas);
...
```




```
void rellena_tablero (matBool dam){
 for (int i=0; i<10; i++)
 { for (int j=0; j<10; j++)
     { if ((10*i+j)%2 = = 0)
       dam[i][j] = 0;
       else dam[i][j] = 1;
     }
 }
```

incorrecto

```
void rellena_tablero (matBool dam){
 for (int i=0; i<10; i++)
 { for (int j=0; j<10; j++)
     { if (i%2!=0 && j%2!=0)
        dam[i][j] = 1;
       else dam[i][j] = 0;
     }
 }
```

correcto

```
void rellena_tablero (matBool dam){
 for (int i=0; i<10; i++)
 { for (int j=0; j<10; j++)
    { if ((i+j)%2 = = 0)
      dam[i][j] = 0;
      else dam[i][j] = 1;
    }
 }
```

Tablas bidimensionales. Matrices /

Estructuras de datos para almacenamiento de datos bidimensionales

- Estructura de datos para almacenar información de ordenadores (máximo 20/aula) en un grupo de aulas : la **declaración** de la estructura de datos del problema podría ser:

- ejemplos de cpu: pentium, centrino, AMD, core_duo, core_quad …

```
struct tOrdenador
{ string cpu;      // Tipo de CPU
  int ram;         // tamaño de la memoria RAM, en Mb
  bool linux;      // sí o no
};
const int NAULAS = 12;
const int NORDENADORES = 20;
typedef tOrdenador tInfoOrds[NAULAS][NORDENADORES];
```

Tablas bidimensionales. Matrices /

Estructuras de datos para almacenamiento de datos bidimensionales

```
struct tOrdenador
{ string cpu;      // Tipo de CPU
  int ram;         // tamaño de la memoria RAM, en Mb
  bool linux;      // sí o no
};
const int NAULAS = 12;
const int NORDENADORES = 20;
typedef tOrdenador tInfoOrds [NAULAS][NORDENADORES];
```

- Con la tupla **tOrdenador** almacenamos la info de cada ordenador, y con la matriz **tInfoOrds,** la información para cada ordenador de cada aula.

Tablas bidimensionales. Matrices /

Estructuras de datos para almacenamiento de datos bidimensionales

Imaginemos que tenemos un programa que manipula datos de tipo **`tInfoOrds`**, así como que alguna parte del mismo se encarga de llenar la estructura de datos con datos. **Podemos crear una función que calcule qué porcentaje de aulas** usan mayoritariamente **(>50%)** el tipo de CPU "**`core_duo`**". La función recibirá la información completa de los ordenadores en una matriz tipo **`tInfoOrds`**.

```
double percent_may_core_duo (const tInfoOrds& IO) {
 int n_may_core_duo=0, int n_core_duo, i, j;
 for (i=0; i<NAULAS; i++)
 {
    n_core_duo = 0;
    for (j=0; j<NORDENADORES; j++)
     {if (IO[i][j].cpu = = "core_duo") n_core_duo++;}
    if (n_core_duo > NORDINADORS/2)
    n_may_core_duo++;
 }
 return (double(n_may_core_duo)/double(NAULAS)*100);
}
```

Tablas bidimensionales. Matrices /

Estructuras de datos para almacenamiento de datos bidimensionales

Propuesta: Dividir la función **percent_may_core_duo** en 2 subprogramas: uno que calcule si una aula concreta tiene mayoría de ***core duo*** (con un nombre como ***mayoria_core_duo***) y otro que se llame ***percent_may_core_duo*** pero que haga servir ***mayoria_core_duo***.

```
bool mayoria_core_duo (const tInfoOrds& InfOrd, int i) {
 int n_core_duo = 0; bool core=false;
 for (j=0; j<NORDENADORES; j++)
     {if (InfOrd[i][j].cpu = = “core_duo”) n_core_duo++;}
 if (n_core_duo > NORDINADORS/2) core=true;
return core;
}
double percent_may_core_duo (const tInfoOrds& IO) {
 int n_may_core_duo=0;
 for (int i=0; i<NAULAS; i++)
 {if (mayoria_core_duo (IO,i))
  n_may_core_duo++;}
 return (double(n_may_core_duo)/double(NORDINADORS)*100);
}
```

Tablas bidimensionales. Matrices /

Estructuras de datos para almacenamiento de datos bidimensionales

- **Propuesta de ejercicio**

  Diseñar un tipo de datos **`tInfoParcs`** para almacenar datos sobre listas de especies de árboles en un número de parques naturales. En concreto queremos poder almacenar, para cada especie en cierto parque natural:
  - el número estimado de ejemplares de la especie en este parque (entero).
  - si la especie está amenazada (en el parque concreto).
  - en qué estación del año se reproduce (**`pri, ver, oto, inv`**) en este parque.

  Hay que declarar los tipos de datos necesarios para poder guardar ordenadamente esta información, pero no hay que escribir un programa como tal.

```
const int NParcs=10;
const int NEsp=100;
struct EspParc {
 int nEjempl;
 bool amenazada;
 string estReprod;};
typedef EspParc tInfoParcs[NParcs][NEsp];
```

Tablas bidimensionales. Matrices /

Estructuras de datos para almacenamiento de datos bidimensionales

- **Propuesta de ejercicio**

  Hacer una función que reciba la información de los parques **tInfoParcs** y también de una especie concreta (un número/índice), y que retorne el número de ejemplares total que hay en todos los parques.

```
[...]
struct EspParc {
 int nEjempl;
 bool amenazada;
 string estReprod;};
typedef EspParc tInfoParcs[NParcs][NEsp];

int num_ejemp (const tInfoParcs& tIP, int Esp) {
 int sumaEjemp = 0;
 for (int i=0; i<Nparcs; i++) sumaEjemp+=tIP[i][Esp].nEjempl;
 return sumaEjemp;
}
```