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

# Fonaments d'Informàtica

## Semana 10. Tipos estructurados // tuplas

# RECAP: Tipos estructurados, Tablas

## Declaración de tipos tabla

- Por ejemplo, para definir un tipos de tabla de 10 enteros, escribiríamos la declaración:

```
typedef int diezenteros[10];
```

- La declaración incluye: la palabra reservada **`typedef`**, el tipo de casilla (aquí, **`int`**), el nombre del tipo y, entre corchetes, el tamaño de la tabla declarada. Es importante ver que **`diezenteros`** es ahora el nombre de un tipo, no de una variable.

**NOTA:** la declaración ha de estar al principio del fichero de código, fuera del **`main`**:

```
#include <iostream>
using namespace std;
typedef int diezenteros[10];
int main()
 {...}
```

# Técnicas de programación con tablas

## Tablas para almacenar secuencias

- Imaginemos un problema consistente en que queremos ver si en una secuencia de palabras (acabada en "**.**") hay palabras repetidas. Para poder plantearse esto, habríamos de almacenar las palabras de la secuencia según llegan. Para cada nueva palabra, podemos buscar en las palabras anteriores a ver si ya aparecía. **El siguiente programa hace eso precisamente: identifica si hay alguna repetición …**

# Tipos estructurados, Tablas

```
bool busca_pal (string pal, tSecPal S, int desde)
 {bool trobat = false;
  while (desde >= 0 && !trobat)
   { if (S[desde] = = pal) trobat = true;
     else desde--;
   }
  return trobat;
 }

typedef string tSecPal[1000];

int main() {
 tSecPal sec; // La tabla
 int k = 0; // índ. "casilla actual"
 string p; // Palabra leída
 bool repes = false;

 cin >> p;
 while (p != "." && !repes)
  { if (busca_pal(p, sec, k-1)) repes = true;
    else {sec[k] = p; // guard pal.act.
          k++; cin >> p;}
  }
 if (repes) cout << "Hay "; else cout << "No hay ";
 cout << " palabras repetidas" << endl;[...]
```

# Técnicas de programación con tablas

## Tablas para almacenar secuencias (3)

- **Propuesta de ejercicio 7**

  Escribir un programa que lea una secuencia de caracteres (acabada en '.') y que una vez leída la muestre por pantalla al revés.

# Un inciso: tuplas

- **Concepto:** Agrupación de variables de diferentes tipos. Las tuplas también se denominan “estructuras” (del inglés *structure*).
- Las tuplas sirven para tratar como una entidad única a un conjunto de datos posiblemente heterogéneos pero con entidad individual propia.
- Un **campo** es cualquiera de las variables individuales que conforman una tupla.

# Tipos estructurados, Tuplas

## Un inciso: tuplas (Declaración de tipo)

- Las tuplas requieren una **declaración de tipo** (al principio del programa, antes del `main`). Esto implica que primero hemos de declarar cuál es el **contenido de una tupla** (los tipos de variables que contiene y sus nombres).
- Ejemplo:

```
struct tData
{
 int dia;
 int mes;
 int any;
};
```

# Tipos estructurados, Tuplas

## Un inciso: tuplas (Declaración)

La declaración incluye:

- La palabra reservada **struct**.
- El nombre del tipo (en el ejemplo, **tData**).
- Un bloque con las declaraciones (como si fuesen variables, y sin inicializar) de los campos de la tupla.

```
struct tData
{
 int dia;
 int mes;
 int any;
};
```

- **Una vez declarado el tipo tupla, podemos declarar variables de este tipo**, p.ej.: `tData aniversario; tData sant_joan;`

## Un inciso: tuplas (Declaración)

- **Propuesta de ejercicio 1**

  Declara las siguientes tuplas:

  - Una tupla **`tHora`** que incluya dos enteros, uno para horas y otro para minutos.
  - Una tupla **`tPunto2D`** que incluya dos reales, uno para la coordenada de abscisas y otro para la de ordenadas.
  - Una tupla **`tPersona`** que incluya lo siguiente: nombre, dos apellidos, estado civil, edad y DNI o NIE.

## Un inciso: tuplas (Acceso)

- Por ejemplo, utilizando la declaración de la variable **aniversario** anterior, de tipo **tData**, llenamos los **campos de esta tupla**:

```
aniversario.dia = 12;
aniversario.mes = 2;
aniversario.any = 1969;
```

- En contraposición a las tablas, las tuplas no tienen índices, de manera que para especificar a cuál de las "variables internas" queremos acceder, usaremos el nombre del campo y no un índice.

**Propuesta de ejercicio 2**

Haciendo uso de las declaraciones del ejercicio anterior:

- Declara una variable de tipo **tHora** y "ponla a las 10:22".
- Declara una var. de tipo **tPunto2D** y sitúala en la posición (5.0, 3.0).
- Declara una var. de tipo **tPersona** y asignale los valores: **nombre** "Groucho", **apellidos** "Marx Neumann", **estado civil** soltero, **edad** 40 años, **DNI** 63086708P.

## Un inciso: tuplas (Inicialización)

- Las variables de tipo tupla se pueden inicializar al declararse. Si tuviésemos una tupla como, per ejemplo:

```
struct tEjemplo {
 int a;
 char b;
 string c;
};
```

podríamos crear una variable de este tipo e inicializarla como:

```
tEjemplo X = { 2, 'z', "prax" };
```

# Un inciso: tuplas (Inicialización)

- La sintaxis se asemeja a la de inicialización de variables de tipos básicos, pero el valor utilizado es un bloque, entre llaves, en el mismo orden que aparecen en la declaración. Por tanto, esta inicialización es equivalente a:

```
tEjemplo X;
X.a = 2;
X.b = 'z';
X.c = "prax";
```

# Un inciso: tuplas Tuplas como parámetro de entrada

- Para pasar **tuplas como parámetros a subprogramas** se ha de tener en cuenta que cuando pasamos un parámetro por valor, el mismo se copia íntegramente en la variable reservada para el parámetro de la función. Esta copia, para los tipos básicos, es rápida, ya que se ocupa poco espacio de memoria.
- El problema es que la **copia de una tupla puede ser mucho más costosa**. La ineficiencia puede incluso implicar que la copia lleve más tiempo que la ejecución de la función/acción.
- Por ello a menudo se usa el **paso por referencia**, unido a una cláusula **const**.

# Tipos estructurados, Tuplas

## Un inciso: tuplas Tuplas como parámetro de entrada

- Por ejemplo, supongamos la declaración de tipo:

```
struct tAsign {
 string titulo;
 int codigo;
 bool fase_selectiva;
 int cuatrimestre;
 string profesor1, profesor2, profesor3;
};
```

- Queremos una acción que reciba una asignatura. La tupla **tAsign** contiene 7 campos de datos y es bastante más grande que una variable básica … Así que, en vez de :

```
void mostra_assign(tAsign A) { // ...}
```

... usamos:

```
void mostra_assign (const tAsign& A) {// ... }
```

# Un inciso: tuplas

**Técnicas de programación con tuplas: Agrupar fechas relacionadas**

- Si un programa manipula datos y queremos, por ejemplo, hacer **una función que compare dos fechas**, en teoría esta función recibirá 6 parámetros (día, mes y año de las dos fechas). Podemos declarar un tipo tupla y hacer que la función reciba 2 parámetros en vez de 6. Suponiendo la declaración previa de **tData**, veamos el siguiente ejemplo:
- // función `posterior`: calcula si d1 es posterior a d2

```
bool posterior (const tData& d1, const tData& d2) {
 bool post = false;
 if (d1.any > d2.any) post = true;
 else if (d1.any = = d2.any)
  {if (d1.mes > d2.mes) post = true;
   else if (d1.mes = = d2.mes)
    {if (d1.dia > d2.dia)
      post = true;
    }
  }
 return post;
}
```

## Un inciso: tuplas

### Técnicas de programación con tuplas: Agrupar fechas relacionadas

- **Variante del problema anterior:** Escribir **una acción que compare dos fechas**, d1 y d2, de manera que saque por pantalla cuál es la más reciente.

```
void posterior (const tData& d1, const tData& d2)
{tData reciente = d2;
 if (d1.any > d2.any) reciente = d1;
 else if (d1.any = = d2.any)
  {if (d1.mes > d2.mes) reciente = d1;
   else if (d1.mes = = d2.mes)
    { if (d1.dia > d2.dia) reciente = d1;
      else if (d1.dia = = d2.dia) cout << “son la misma” << endl;
    }
  }
 cout << reciente;
}
```

# Un inciso: tuplas

## Técnicas de programación con tuplas

**Propuesta de ejercicio**: Escribir una acción **leer_data** que lea una fecha (de tipo **tData**) entrada por teclado.

**Propuesta de ejercicio**: Hacer un programa entero (utilizando los resultados previos) que lea dos fechas y y diga cuál es la más reciente.