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

Fonaments d'Informàtica

Semana 10. Tipos estructurados // tuplas

RECAP: 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;
};
```

RECAP: 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;`

RECAP: Tipos estructurados, Tuplas

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

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.

Un inciso: tuplas

Técnicas de programación con tuplas

Propuesta de ejercicio

Hacer una función que calcule y retorne la distancia entre 2 puntos bi-dimensionales (**tPunt2D**), pasados como parámetros.

Tipos estructurados, Tuplas

Un inciso: tuplas. Copia de tuplas y tablas

- Una de las **diferencias** importantes entre tuplas y tablas es la posibilidad de copiarlas haciendo una asignación. Es decir, si tenemos:

```
struct tTupla {
 int i;
 char c;
 string s;
};
typedef bool tVector[100];
```

y tenemos también dos variables de cada tipo:

```
tTupla A, B; tVector X, Y;
```

nos interesan las asignaciones:

`A = B;` // **SE PUEDE HACER**, igual que: `A.i=B.i; A.c=B.c; A.s=B.s;`

`X = Y;` // **NO SE PUEDE HACER**

Un inciso: tuplas

Composición de tipos: tuplas de tablas

```
struct tArticulo {
 string titulo; // título de un artículo
 string palabras[5000]; // texto de un artículo
 int npar; // número de palabras
};
```

- En este caso, la tupla `tArticulo` tiene un título y una "secuencia" de palabras que es el texto del artículo. Para saber cuántas palabras contiene realmente (5,000 es tan sólo el límite superior), usamos `npar`, que indica la primera casilla de la tabla `palabras` que está **vacía**.

Un inciso: tuplas

Composición de tipos: tuplas de tablas (DEFINICIÓN DE TIPO ALTERNATIVA)

```
typedef string texto[5000];
struct tArticulo {
 string titulo; // título de un artículo
 texto palabras; // texto de un artículo
 int npar; // número de palabras
};
```

- En este caso, la tupla `tArticulo` tiene un título y una "secuencia" de palabras almacenadas en una tabla de tipo **`texto`**. Para saber cuántas palabras contiene realmente (5,000 es tan sólo el límite superior), usamos **`npar`**, que indica la primera casilla de la tabla `palabras` que está **vacía**.

Tipos estructurados. Tablas+Tuplas

Un inciso: tuplas Composición de tipos: tablas de tuplas

- **Las tablas pueden tener tuplas como contenido de las casillas**. Por ejemplo:

```
struct tPunto2D { float x, y; };
typedef tPunto2D tTablaPuntos[100];
```

- En este ejemplo, hemos creado una tabla, cada casilla de la cual es un punto bidimensional. Para acceder a sus coordenadas, hemos de usar la notación de tablas y tuplas en el orden correcto. Esto es:

```
tTablaPuntos P,Q;
P[5].x = -3.4;
Q[10].y = 2.7;
```

- En este caso, ya que `P` es una tabla, hemos de acceder a la casilla 5 con `P[4]`. Pero la casilla `P[4]` es una tupla y, por tanto, para acceder a la coordenada `x` pondremos `P[4].x`.