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

Fonaments d'Informàtica

Semana 9. Tipos estructurados // tablas

Tipos estructurados. Tablas.

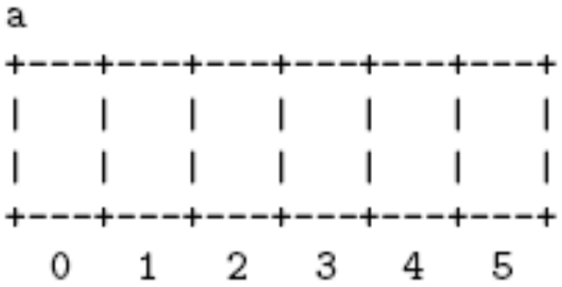
Qué son los TE y por qué los necesitamos

- Los tipos simples de datos (`int, float, char` ...) nos permiten describir información de un **nivel limitado de complejidad.**
- Los tipos simples de datos nos permiten tan sólo describir **información homogénea** en sus características.
- Necesitamos tipos que nos permitan describir entidades más **complejas y heterogéneas.**
- Tablas uni- y bi-dimensionales (matrices), tuplas, y sus combinaciones.

Tipos estructurados. Tablas.

Tablas

- Una tabla simple (1-D) es una **agrupación de variables del mismo tipo**, accesibles mediante un **índice** (entero). Estas variables individuales se denominan **casillas de una tabla**. El diagrama de la derecha representa una tabla:
- A esta tabla ejemplo se le da un nombre (digamos, **a**) y agrupa 6 enteros, indexados del 0 al 5. **NOTA: fijaos en que los índices de una tabla comienzan en el 0 y no en el 1**, como cabría imaginarse.
- **ATENCIÓN:** esto es a menudo fuente de errores ... =:^(



Tipos estructurados. Tablas.

Tablas (declaración de variable)

- Para **declarar una tabla** simple en un programa de C++, podemos, por ejemplo, hacerlo de la siguiente manera:

```
int mitabla[10];
```

- Esta declaración incluye un tipo (`int`), el nombre de la variable tabla (`mitabla`) y su tamaño "reservado", entre corchetes (aquí, espacio reservado a 10 enteros).

Propuesta de ejercicio 1:

Escribir la declaración de:

- Una variable tabla de 50 chars.
- Una variable tabla de 100 reales.
- Una variable tabla de 2 strings.

Tipos estructurados. Tablas.

Tablas (acceso)

- Para **acceder** a una casilla de una tabla, invocamos el nombre de la tabla y, después, entre corchetes, el índice de la casilla a la que buscamos acceder. Por ejemplo, las siguientes instrucciones declaran una tabla de 2 caracteres y la llenan con 2 valores:

```
char grupocar[2];
grupocar[0] = 'U';
grupocar[1] = '2';
```

- Cada una de las casillas de una tabla simple se puede ver como una variable con un "nombre especial" referenciado por un índice.

Propuesta de ejercicio 2

Escribir un programa que declare una tabla de 5 enteros y la llene con los valores 1, 5, 3, 2, 4.

Tipos estructurados. Tablas.

Tablas (declaración+acceso=inicialización)

- Podemos inicializar directamente una variable tabla al declararla. Por ejemplo, podríamos asignar directamente:

```
double mitabla[5] = {0.1, 2.5, -0.3, 4.1, 111.8};
```

- Esto tiene utilidad directa sólo para tablas pequeñas.
- Inicialización de longitud indeterminada … ¿? …

Tipos estructurados. Tablas.

Tablas (acceso a un índice calculado)

- **El cálculo de los índices permite automatizar el acceso a las casillas**. Por ejemplo, si declaramos una tabla de 100 números reales, podemos implementar una "tarea iterativa" que llene todas las casillas con, por ejemplo, el valor 1.5:

```
float tablareales[100];
int k;
for (k = 0; k < 100; k++) {tablareales[k] = 1.5;}
```

Propuesta de ejercicio 3

Escribir un programa que declare una tabla de 500 enteros y la llene con los valores del 500 al 1, en orden descendente.

Tablas (acceso a un índice calculado)

- Una característica interesante del índice de las tablas es que no tiene porqué ser un número fijo: **puede provenir de variables enteras**. Por ejemplo, el siguiente código hace lo mismo que el anterior, pero hace servir una variable entera para indicar las casillas a las que se quiere acceder:

```
char grupo[2];
int a = 0;
grupo[a] = 'U';
grupo[a+1] = '2';
```

Tipos estructurados. Tablas.

¡los *strings* son tablas!

- Las variables de tipo ***string*** son, de hecho, tablas de caracteres, y de longitud variable. Cuando se declara un **string** y no se inicializa, e.g.

```
string x;
```

la tabla está de hecho vacía (no “tiene casillas”). Si hacemos una asignación como …

```
x = "ex ovum omnia";
```

la tabla que representa `x` pasa a hacerse de 13 casillas (con índices del 0 al 12), y se llena con los 13 caracteres que hemos puesto entre comillas dobles.

Tipos estructurados. Tablas.

¡los *strings* son tablas!

- Se puede acceder a las casillas de un **`string`** como a las de cualquier tabla. Por ejemplo, el siguiente código

```
string g = "radiohear";
g[8] = 'd';
cout << g << endl;
```

… mostraría por pantalla: **`radiohead`**

Hemos, por tanto, cambiado la novena casilla de la tabla (índice 8), de ser una **`'r'`** a ser una **`'d'`**.

Propuesta de ejercicio 4

Escribir un programa que lea una secuencia de palabras acabadas en **"."**, pase la primera letra de cada palabra a mayúsculas y la muestre por pantalla.

Tipos estructurados. Tablas.

¡los *strings* son tablas!

Propuesta de ejercicio 4

Escribir un programa que lea una secuencia de palabras acabadas en ".", pase la primera letra de cada palabra a mayúsculas y la muestre por pantalla.

```
int main()
{
    string pal;
    cin >> pal;
    while (pal!=".")
    {pal[0] = toupper(pal[0]);
     cout << pal << endl;
     cin >> pal;}
  […]
}
```

Tipos estructurados. Tablas

¡los *strings* son tablas!: cálculo de longitud de un *string*

Propuesta de ejercicio 5 (`s.size()`)

Escribir un programa que reciba una secuencia de palabras acabada en "." y diga la "longitud" de la palabra más corta y de la más larga.

```
int main( ) {
 string pal;
 int long_corta, long_larga;
 cout << "Introduce palabra" << endl;
 cin >> pal;
 long_corta = pal.size(); long_larga = pal.size();
 while (pal!=".")
 {
  if (pal.size() > long_larga) long_larga=pal.size();
  else if (pal.size() < long_corta) long_corta=pal.size();
  cin >> pal;
 }
 cout << "l. max es: "<< long_larga << "y l. min es: "<< long_corta <<endl;
[…]
}
```