Metodología de la Programación

Tema 0. Structs

Departamento de Ciencias de la Computación e I.A.

ETSIIT Universidad de Granada

Curso 2012-13

Contenido del tema

Contenido del tema

Definición y sintaxis

Definición

Las estructuras o registros son tipos de dato compuestos que se definen a partir de elementos de otros tipos.

Sintaxis

```
struct <NombreEstructura> {
  <tipo1> <miembro1>;
  <tipo2> <miembro2>;
  ...     ...
  <tipon> <miembron>;
};
```

La estructura tiene un nombre `<NombreEstructura>` que es el nombre del tipo de dato. Cada miembro tiene un nombre asociado `<miembroX>` que nos permitirá referenciarlo.

Ejemplos

- Definición de un punto en el plano (con coordenadas x, y).

```
struct Punto{
  double x;
  double y;
};
```

- Información sobre un alumno (NIF, nombre, curso, grupo, calificaciones parciales).

```
struct Alumno{
  string NIF;
  string nombre;
  int curso;
  char grupo;
  double notas[3];
};
```


Structs y class

- Los `struct` son herramientas muy similares a las clases. Pueden contener:
  - Especificadores de acceso
  - Métodos miembro
  - Constructores y destructores
- Diferencia:
  - Los miembros de una estructura son por defecto `public` (`private` en `class`).
- Habitualmente usaremos `struct` en lugar de `class` cuando la clase es muy simple y no necesita métodos (comportamiento): serviría como forma de agrupar los datos que contiene.
- Los `struct` suelen usarse a menudo para ayudar a definir *estructuras de datos*.

Contenido del tema

Declaración de variables

```
struct Alumno{
  string NIF;
  string nombre;
  int curso;
  char grupo;
  double notas[3];
};
```

Ejemplo

```
Alumno ahora, arrayAlumnos[10], matrizAlumnos[5][7];
```

Inicialización

Una estructura se puede inicializar en la declaración usando la misma notación que para inicializar arrays.

Ejemplo

```
1 struct Punto{
2   double x;
3   double y;
4 };
5 struct Alumno{
6   string NIF;
7   string nombre;
8   int curso;
9   char grupo;
10  double notas[3];
11 };
12 Punto origen = {0.0, 0.0};
13 Alumno estudiante = {"12345678Z", "Juan Sevilla", 1, 'B',
14                      {0.0,0.0,0.0} };
```


Operadores de acceso a valores miembros de la estructura

Los miembros de una estructura se acceden mediante:

- El operador punto (.)
  Accede a un miembro a través del nombre de la variable del tipo de la estructura.

```
Punto punto={7.5, 2.3};

cout<<punto.x<<", "<<punto.y;
```

Importante:

En general, `<variable>.<miembro>` es una variable y se comporta como cualquier variable.

- El operador flecha (->)
  El operador flecha accede a un miembro a través de la dirección de memoria de una variable del tipo de la estructura.
  Lo estudiaremos más adelante.

Asignación campo a campo

- La asignación se puede realizar de forma individual sobre cada uno de los miembros de la estructura, combinando las operaciones de acceso con el operador de asignación
  `<OperacionAcceso> = <expresion>;`

Ejemplo

```
struct Alumno{
  string NIF;
  string nombre;
  int curso;
  char grupo;
  double notas[3];
};
Alumno alumno;
alumno.NIF="26262727T";
...
alumno.notas[0]=7.2;
...
```


Asignación completa

- La asignación se puede realizar de forma completa, asignando a una estructura otra estructura del mismo tipo.

Ejemplo

```
struct Alumno{
  string NIF;
  string nombre;
  int curso;
  char grupo;
  double notas[3];
};
Alumno estudiante1 = {"12345678Z", "Juan Sevilla", 1, 'B',
                      {7.2,5.3,3.7} };
Alumno estudiante2;
estudiante2=estudiante1;
```

Operaciones de entrada y salida

En principio, se deben realizar individualmente sobre cada valor miembro de la estructura, y consiste en combinar las operaciones de entrada y salida con las operaciones de acceso a valores miembro.

```
Punto punto;
...
cout<< "Introduce coordenada x: ";
cin>>punto.x;
cout<<Introduce coordenada y: ";
cin>>punto.y;

cout<<"El punto introducido es: "<<punto.x<<
                              ", "<<punto.y<<endl;
```


Contenido del tema

Paso de estructuras a funciones

Las estructuras se comportan en C++ como si fueran tipos de datos básicos cuando se utilizan como argumento de las funciones:

- Se pueden pasar **por valor**:

```
double calculaDistancia(Punto punto1,Punto punto2);
```

- Se pueden pasar **por referencia**:

```
void leerPunto(Punto &punto);
```

- Se pueden pasar **por referencia constante**:

```
double calculaDistancia(const Punto &punto1,const Punto
&punto2);
```

- Las funciones pueden **devolver estructuras**:

```
Punto puntoMedio(const Punto &punto1,const Punto &punto2);
```


Diferencia entre paso por referencia constante y paso por valor

- Paso por valor:

```
void imprimePunto(Punto punto){
   ....
   punto.x=5.2; //Permitido: se modifica punto que es una copia
```

- Paso por referencia constante:

```
void imprimePunto(const Punto &punto){
   ....
   punto.x=5.2; //NO Permitido
```

El paso por referencia constante, pasa una referencia sobre el dato con el que se quiere trabajar (con lo que evitamos una copia que puede ocupar mucha memoria) pero lo protege para que no se pueda modificar el dato original.

Contenido del tema

Peculiaridad de arrays o matrices miembros de una estructura

- No es válida la asignación directa de arrays:

```
int v1[50], v2[50];
v2 = v1;
```

Se produce un error ya que las copias de arrays se deben hacer componente a componente.

- Sin embargo, sí que es válido lo siguiente:

```
struct array50int{
   int util; // num de elementos usados
   int miarray[50];
};
array50int v1, v2;
...
v2 = v1;
```


Contenido del tema

Arrays y matrices de estructuras

El acceso a los diferentes miembros de las estructuras, se realiza combinando el acceso a los elementos de la matriz con las operaciones de acceso a los miembros de la estructura.

```
struct Alumno{
  string NIF;
  string nombre;
  int curso;
  char grupo;
  double notas[3];
};
int main(){
   Alumno listaAlumnos[100];

   listaAlumnos[0].NIF="26262727T";
   ...
   cin >> listaAlumnos[3].notas[0];
   ...
   listaAlumnos[1]=listaAlumnos[0];
```

Contenido del tema

Estructuras de estructuras

- Como caso particular, un valor miembro de una estructura puede ser a su vez otra estructura.

```
struct Punto{
  double x;
  double y;
};
struct Circulo{
  Punto centro;
  double radio;
};
Circulo circulo1={{5.0,4.0},10.0};
Circulo circulo2;
circulo2.centro.x=7.2;
circulo2.centro.y=5.2;
circulo2.radio=3.0;
```