Clases en C++ - Struct & Class

Di Paola Martín
martinp.dipaola <at> gmail.com

Facultad de Ingeniería
Universidad de Buenos Aires

structs y clases en C++

Bundle

TDAs - Clases en C

```
1  struct Vector {
2    int *_data; /*private*/
3    int _size; /*private*/
4  };
```

```
15 void f() {
16   struct Vector v;
17   vector_create(&v, 5);
18   vector_get(&v, 0);
19   vector_destroy(&v);
20 }
```

```
5  void vector_create(struct Vector *v, int size) {
6    v->_data = malloc(size*sizeof(int));
7    v->_size = size;
8  }
9  int vector_get(struct Vector *v, int pos) {
10   return v->_data[pos];
11 }
12 void vector_destroy(struct Vector *v) {
13   free(v->_data);
14 }
```

- El estándar de C++ garantiza que si un `struct` no tiene ningun feature de C++ (o sea, se parece a un struct de C) se lo llama "plain struct" y puede ser usado por libs de C desde C++
- Por convención los nombres de las funciones del TDA deben tener como prefijo de su nombre el nombre del TDA: esto es por que en C todas las funciones terminan en el mismo espacio global y deben tener nombres únicos. El conflicto de nombres es un problema común en proyectos grandes en C. En C++ tenemos mejores formas de resolverlos....
- Asi también es convención pasar como primer argumento un puntero al `struct`. Veremos que en C++ hay una forma más conveniente de hacer esto...
- Y otra convención mas: los atributos que no deberían ser ni leídos ni modificados por el usuario son marcados como privados. Dependiendo de la convención hay gente que le pone un guion bajo al principio de la variable, otros al final y otros ponen solamente un comentario. C++ nos dara herramientas para forzar esto en tiempo de compilación.

Keyword struct implícita

```
1  struct Vector {
2    int *_data; /*private*/
3    int _size; /*private*/
4  };
```

```
15 void f() {
16   Vector v;
17   vector_create(&v, 5);
18   vector_get(&v, 0);
19   vector_destroy(&v);
20 }
```

```
5  void vector_create(Vector *v, int size) {
6    v->_data = malloc(size*sizeof(int));
7    v->_size = size;
8  }
9  int vector_get(Vector *v, int pos) {
10   return v->_data[pos];
11 }
12 void vector_destroy(Vector *v) {
13   free(v->_data);
14 }
```

- En C++ no es necesario usar la keyword `struct` en todos lados.

Bundle: atributos + métodos

```
1  struct Vector {
2    int *_data; /*private*/
3    int _size; /*private*/
4
5    void vector_create(Vector *v, int size) {
6      v->_data = malloc(size*sizeof(int));
7      v->_size = size;
8    }
9
10   int vector_get(Vector *v, int pos) {
11     return v->_data[pos];
12   }
13
14   void vector_destroy(Vector *v) {
15     free(v->_data);
16   }
17 };
```


- Se integran las funciones y los datos del TDA en una sola unidad.
- Los datos del TDA se lo llaman atributos y las funciones métodos.

this: un puntero a la instancia

```
1  struct Vector {
2    int *_data; /*private*/
3    int _size; /*private*/
4
5    void vector_create(int size) {
6      this->_data = malloc(size*sizeof(int));
7      this->_size = size;
8    }
9
10   int vector_get(int pos) {
11     return this->_data[pos];
12   }
13
14   void vector_destroy() {
15     free(this->_data);
16   }
17 };
```


- Las funciones del TDA pasan a ser métodos del TDA y reciben como parámetro implícito un puntero a la instancia.
- El puntero es un puntero constante a la instancia (`Vector *const`) y se lo nombra con la keyword `this`. En otras palabras `this` es un puntero constante que apunta al objeto sobre el cual se esta invocando el método.

Invocación de métodos

```
14 // En C
15 void f() {
16   struct Vector v;
17   vector_create(&v, 5);
18   vector_get(&v, 0);
19
20   v._data;
21
22   vector_destroy(&v);
23 }
```

```
14 // En C++
15 void f() {
16   Vector v;
17   v.vector_create(5);
18   v.vector_get(0);
19
20   v._data;
21
22   v.vector_destroy();
23 }
```


- Se accede a los atributos y/o métodos como en C
- En C, la instancia sobre la que se quiere invocar un método es pasada como parámetro de forma explícita mientras que en C++ es implícita e invisible.

Reducción de colisiones de nombres

```
1  struct Vector {
2    int *_data; /*private*/
3    int _size; /*private*/
4
5    void create(int size) { // Vector::create
6      this->_data = malloc(size*sizeof(int));
7      this->_size = size;
8    }
9
10   int get(int pos) { // Vector::get
11     return this->_data[pos];
12   }
13
14   void destroy() { // Vector::destroy
15     free(this->_data);
16   }
17 };
```


- Los métodos de un TDA no entran en conflicto con otros aunque se llamen iguales. El método `get` de `Vector` no entra en conflicto con el método `get` de `Matrix`, por ejemplo
- En rigor un método de un TDA se lo llama `NombreTDA::NombreMetodo`, por eso `Vector::get` es distinto de `Matrix::get`.
- Veremos con mas detalle el concepto de namespace en las próximas clases.

structs y clases en C++

Permisos de acceso

Permisos de acceso

```
1  struct Vector {
2    private:
3    int *data;
4    int size;
5
6    public:
7    void create(int size) {
8      this->data = malloc(size*sizeof(int));
9      this->size = size;
10   }
11
12   int get(int pos) {
13     return this->data[pos];
14   }
15
16   void destroy() {
17     free(this->data);
18   }
19 };
```

- Por default, un `struct` tiene sus atributos y métodos públicos. Esto significa que pueden accederse desde cualquier lado.
- Se puede cambiar el default forzando distintos permisos.
- `private` hace que sólo los métodos internos puedan acceder a los métodos y atributos privados.
- Más sobre los permisos `public`/`protected`/`private` y su relación con la herencia en las próximas clases.

Permisos de acceso

```
14 // En C
15 void f() {
16   struct Vector v;
17   vector_create(&v, 5);
18   vector_get(&v, 0);
19
20   v._data;
21
22   vector_destroy(&v);
23 }
```

```
14 // En C++
15 void f() {
16   Vector v;
17   v.create(5);
18   v.get(0);
19
20   v.data;
21
22   v.destroy();
23 }
```

structs y clases en C++

Clases

Clases en C++

```
1  struct Vector {
2    int *data;  // public by default
3    int size;   // public by default
4  };
5
6  class Vector {
7    int *data;  // private by default
8    int size;   // private by default
9  };
```


- Absolutamente todo lo visto con structs en C++ aplica a las clases de C++. La única diferencia es que las clases tienen sus atributos y métodos privados por default.

Unidades de compilación

```
1  class Vector {
2    private:
3    int *data;
4    int size;
5
6    public:
7    void create(int size);
8    int get(int pos);
9    void destroy();
10
11 }; // en el archivo vector.h
```

```
1  #include "vector.h"
2  void Vector::create(int size) {
3    this->data = malloc(
4    this->size = size;
5  }
6
7  int Vector::get(int pos) {
8    return this->data[pos];
9  }
10
11 void Vector::destroy() {
12   free(this->data);
13 } // en el archivo vector.cpp
```

- Hasta ahora se integró en un solo lugar el código de cada método. Es más simple pero trae problemas de performance del proceso de compilación.
- Para evitar recompilar una y otra vez el código de los métodos se le define en un archivo .cpp separado de las declaraciones del .h