VORTEX

EMBEDDED MAKERS

SASE 2017 SIMPOSIO ARGENTINO DE SISTEMAS EMBEBIDOS

Software más flexible con interfaces en C

Ing. Leandro Francucci (lf@vortexmakes.com)

10 de Agosto 2017

Objetivo

Comprender los principios fundamentales del diseño orientado a objetos (OOD) aplicados al desarrollo de software en lenguaje C. Y por medio de estos:

- Fomentar la producción de **software flexible**, **fácil de mantener** y **reutilizable**.
- Lograr un diseño de software adaptable a los cambios, hoy conocido como **diseño ágil**.
- Disminuir el tiempo, complejidad y costo del desarrollo.

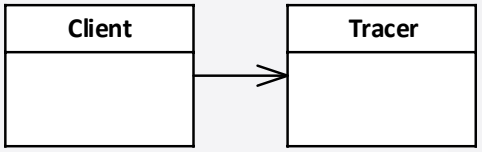
Agenda

- Utilizar abstracciones, mediante la herencia y el polimorfismo en lenguaje C
  - Abstracciones. ¿Porqué y cuándo aplicarlas?. Definición y aplicación de interfaces y operaciones polimórficas.
  - Selección de implementación de interfaces en tiempo de compilación y en tiempo de ejecución.
  - Diferentes estrategias para implementarlas en C

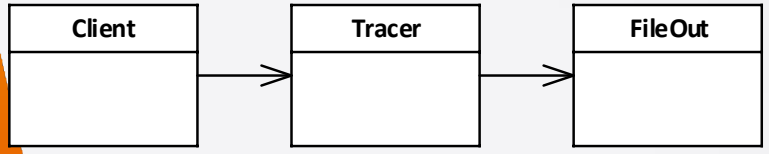
Abstracción, interfaces y polimorfismo

Abstracciones

- Supongamos que disponemos de un sistema que registra y emite la trazabilidad de una aplicación.

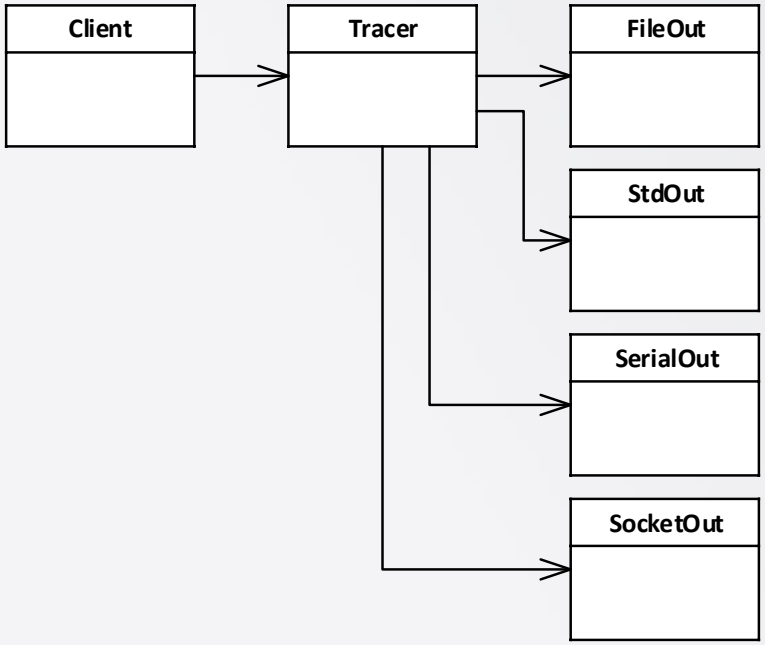


- El módulo `Tracer` almacena los registros de la trazabilidad en un archivo determinado, como una entidad monolítica.



- Ahora, delegamos la emisión de los registros de `Tracer` al módulo `FileOut` dedicado exclusivamente a la manipulación de estos registros sobre archivos.

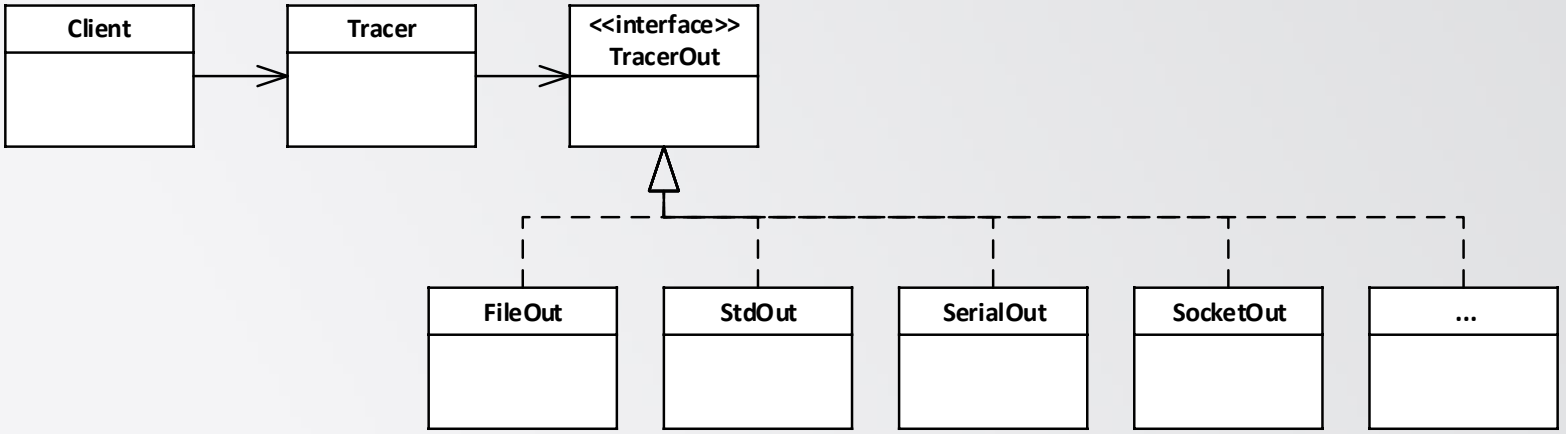
Abstracciones



- El sistema evoluciona y ahora requiere emitir los registros a diferentes medios (archivo, serial, pantalla, socket, otros).
- Por lo tanto, modificamos `Tracer` para lograr el soporte necesario.
- De ahora en más, un cambio en los módulos de salida puede implicar cambios en el módulo `Tracer`.

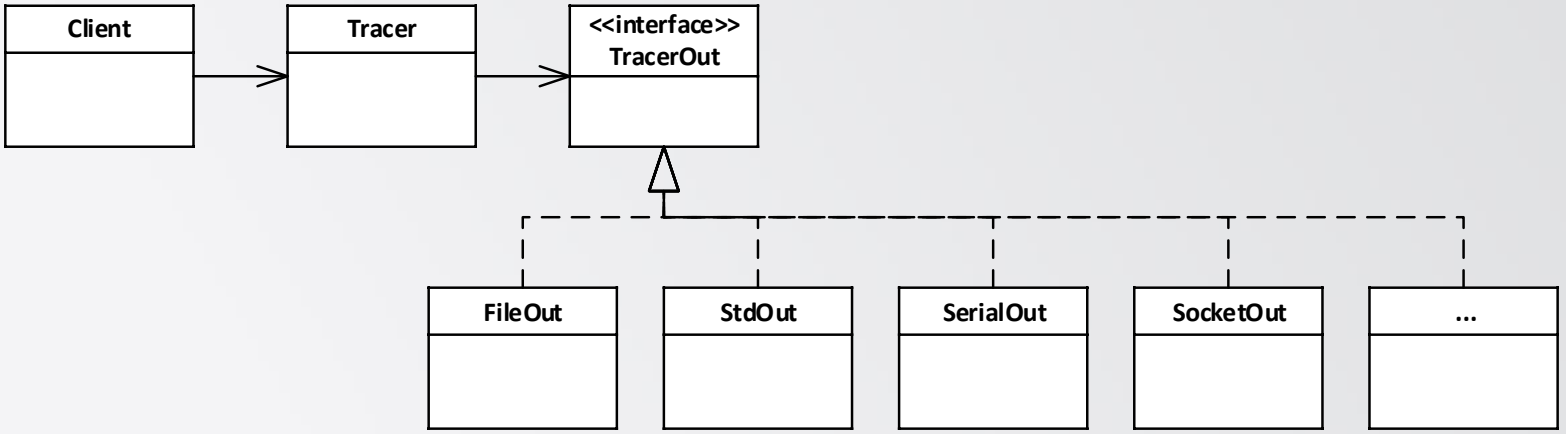
¿Cómo podríamos evitar esto último?

Abstracciones



- La mejor manera de aislar o desacoplar dos o más módulos es mediante una abstracción.
- Si bien las abstracciones están fijas, representan un grupo ilimitado de posibles comportamientos.
- Ahora, `Tracer` accede a la abstracción, y por lo tanto permanece "cerrado" para su modificación, ya que depende de una abstracción que está fija. Sin embargo, su comportamiento puede extenderse mediante la creación de nuevos módulos derivados de la abstracción.
- Esto último conforma el **Principio de Abierto/Cerrado** (OCP).

Abstracciones



- El módulo de alto nivel `Tracer` no depende de los módulos de bajo nivel, `FileOut`, `StdOut`, etc. Estos dependen de la abstracción `TracerOut`.
- La abstracción no depende de los detalles de la implementación. Por el contrario, estas dependen de la abstracción.
- Esto conforma el **Principio de Inversión de Dependencias** (DIP)

Abstracciones

Definición:

“la amplificación de lo esencial y la eliminación de lo irrelevante”

- Es la mejor manera de eliminar el código redundante
- Tendiendo a lograr sistemas más fáciles de entender y mantener.

¿Porqué estas prácticas?

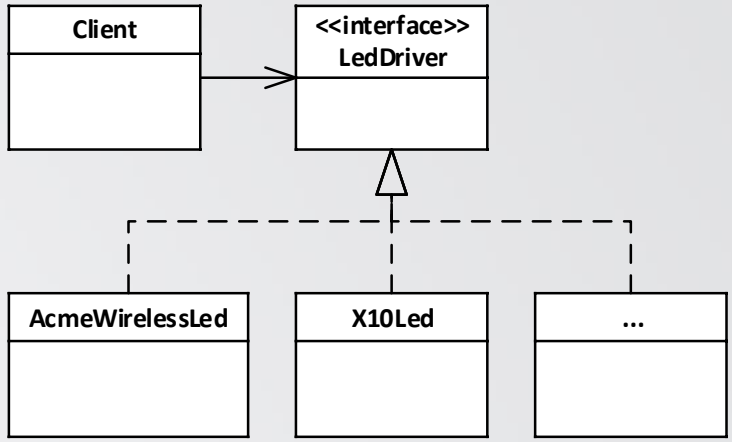
- Construir **software** que posea una buena estructura, de modo tal que sea **flexible**, **fácil de mantener** y **reutilizable**.
- La aplicación de estos principios, prácticas y patrones para mejorar la estructura y legibilidad del software, *es un proceso*, no un evento, es decir se aplican contínuamente, manteniendo en todo momento el **diseño** del sistema tan **simple**, **claro** y **expresivo** como sea posible.
- A su vez, si el desarrollo se realiza en pequeños incrementos, aplicando prácticas que provean la disciplina y la realimentación necesaria, el software se construye rápidamente, aún ante cambios vertiginosos de requerimientos.

Todo esto constituyen el **Desarrollo Ágil**

Vinculación de las implementaciones

Vinculando la implementación con la abstracción

- Establecida la abstracción, resta vincularla con las implementaciones. En lenguaje C, podemos utilizar varios métodos para lograrlo, básicamente:
  - Vinculación **estática**
    - Se produce al momento de compilar (mediante preprocesador) o linkear (mediante el linker)
  - Vinculación **dinámica**
    - Se efectúa en momento de ejecución mediante punteros a función (puede aplicarse a las instancias de manera grupal o individual)
- La elección de uno u otro depende de la relación de compromiso entre sus ventajas y desventajas para una aplicación determinada.



Vinculación preprocesador

```
 3 #define TracerOut_open() \
 4     if ((file = fopen("../tracer_out", "w+b")) == NULL) \
 5     { \
 6         perror("TraceOut: can't open \"trace_out\" file\n"); \
 7         exit(EXIT_FAILURE); \
 8     }
 9
10 #define TracerOut_close()    ...
11 #define TracerOut_flush()    ...
12 #define TracerOut_getTimeStamp()    ...
```

- La solución más básica consiste en una macro que provea la abstracción.
- La vinculación es en momento de compilación.
- Entre otras cuestiones, todo cambio en la implementación de la abstracción provoca cambios en el código que utiliza la abstracción.

Vinculación preprocesador

```
15 /* in TraceOut.h */
16 #if TRACEOUT_FILE_EN == ENABLED
17 #include "FileOut.h"
18 #define TracerOut_open()            FileOut_open()
19 #define TracerOut_close()           FileOut_close()
20 #define TracerOut_flush()           FileOut_flush()
21 #define TracerOut_getTimeStamp()    FileOut_getTimeStamp()
22 #elif TRACEOUT_STD_EN == ENABLED
23 #include "StdOut.h"
24 #define TracerOut_open()            StdOut_open()
25 #define TracerOut_close()           StdOut_close()
26 #define TracerOut_flush()           StdOut_flush()
27 #define TracerOut_getTimeStamp()    StdOut_getTimeStamp()
28 #elif TRACEOUT_SERIALOUT_EN == ENABLED
29 #include "SerialOut.h"
30 #define TracerOut_open()            SerialOut_open()
31 #define TracerOut_close()           SerialOut_close()
32 #define TracerOut_flush()           SerialOut_flush()
33 #define TracerOut_getTimeStamp()    SerialOut_getTimeStamp()
34 #elif TRACEOUT_SOCKETOUT_EN == ENABLED
35 #include "SocketOut.h"
36 #define TracerOut_open()            SocketOut_open()
37 #define TracerOut_close()           SocketOut_close()
38 #define TracerOut_flush()           SocketOut_flush()
39 #define TracerOut_getTimeStamp()    SocketOut_getTimeStamp()
40 #else
41 #include "StdOut.h"
42 #define TracerOut_open()            StdOut_open()
43 #define TracerOut_close()           StdOut_close()
44 #define TracerOut_flush()           StdOut_flush()
45 #define TracerOut_getTimeStamp()    StdOut_getTimeStamp()
46 #endif
```

- Si afrontamos un problema de incompatibilidad entre la interfaz que necesita el cliente y la provista por el servidor, necesitamos definir un adaptador que interceda entre el cliente y el servidor.
- Difícil determinar que código se compila realmente para diferentes situaciones (compilación condicional).

Vinculación por linker

```
49 /* in TraceOut.h */
50 void TracerOut_open(void);
51 void TracerOut_close(void);
52 void TracerOut_flush(void);
53 TimeStamp TracerOut_getTimeStamp(void);
54
55 /* in FileOut.c (StdOut.c, SerialOut.c and SocketOut.c) */
56 void
57 TracerOut_open(void)
58 {
59     if ((file = fopen("../tracer_out", "w+b")) == NULL)
60     {
61         perror("TraceOut: can't open \"trace_out\" file\n");
62         exit(EXIT_FAILURE);
63     }
64 }
65
66 void
67 TracerOut_close(void)
68 {
69     /*...*/
70 }
71
72 void
73 TracerOut_flush(void)
74 {
75     /*...*/
76 }
77
78 TimeStamp
79 TracerOut_getTimeStamp(void)
80 {
81     /*...*/
82 }
```

- `TraceOut.h` define la abstracción
- Cada implementación se define en su propio archivo, donde se respeta la firma de la operación definida en la abstracción.
- Se elije para linkear el archivo de la implementación requerida.
- Un cambio en la implementación no provoca un cambio en el código que utiliza la abstracción.

Vinculación dinámica
Singleton

```
85 /* in TraceOut.h */
86 typedef struct TraceOutVtbl TraceOutVtbl;
87 struct TraceOutVtbl
88 {
89     void (*open)(void);
90     void (*close)(void);
91     void (*flush)(void);
92     TimeStamp (*getTimeStamp)(void);
93 };
94
95 void TracerOut_open(void);
96 void TracerOut_close(void);
97 void TracerOut_flush(void);
98 TimeStamp TracerOut_getTimeStamp(void);
99 void TracerOut_setInterface(TraceOutVtbl *interface);
100
101 /* in TraceOut.c */
102 static TraceOutVtbl *vtbl = (TraceOutVtbl *)0;
103
104 void
105 TraceOut_open(void)
106 {
107     (*vtbl->open)();
108 }
109
110 /* ... */
111
112 void
113 TraceOut_setInterface(TraceOutVtbl *interface)
114 {
115     vtbl = interface;
116 }
```

- La implementación se vincula en tiempo de ejecución.
- Esto permite elegir el comportamiento en tiempo de ejecución.
- Obteniendo un único binario ejecutable para todas las implementaciones.
- En esta caso, el módulo `TraceOut` administra una única instancia (singleton).
- Para lograrlo se utiliza una tabla de punteros a función, `vtbl` (conocida como tabla de funciones virtuales)

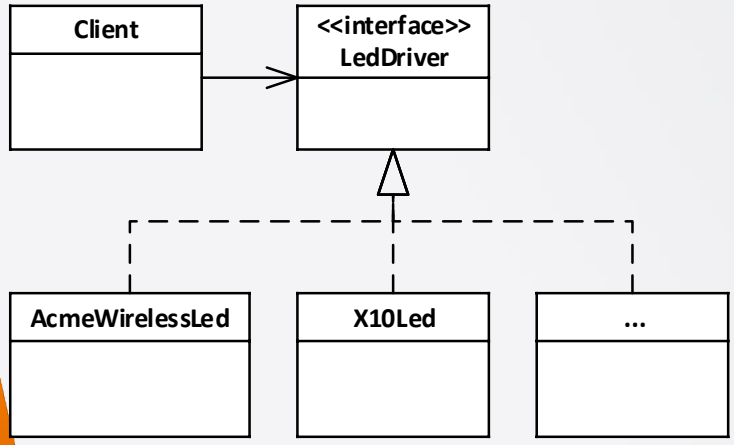
Vinculación dinámica
Singleton

```
118 /* in FileOut.h */
119 void FileOut_init(char *path);
120
121 /* in FileOut.c */
122 static void
123 open(void)
124 {
125     if ((file = fopen(filePath, "w+b")) == NULL)
126     {
127         perror("TraceOut: can't open the output file\n");
128         exit(EXIT_FAILURE);
129     }
130 }
131
132 static TraceOutVtbl vtbl = {open, close, flush, getTimeStamp};
133
134 void
135 FileOut_init(char *path)
136 {
137     strcpy(filePath, path);
138     TraceOut_setInterface(&vtbl);
139 }
```

- Cada implementación se define en un archivo único.
- La tabla de funciones virtuales `vtbl` provee las realizaciones de las operaciones.

Vinculación dinámica
Instancias múltiples

- Supongamos que disponemos de un sistema que controla, mediante un módulo, un conjunto de LEDs. Teniendo como requerimientos:



- Posibilitarnos, en tiempo de ejecución, configurar el tipo de LED que se controla.
- Tipos AcmeWireless, X10, otros
- De manera grupal (para todos los LEDs por igual) o de manera individual (a cada LED se le asigna un tipo particular)

https://github.com/vortextech/ood.git

Vinculación dinámica
Instancias múltiples – Versión 1

```c
/*
 *  main.c
 */

#include <stdio.h>
#include "X10Led.h"
#include "AcmeWirelessLed.h"

static X10Led x10Led;
static AcmeWirelessLed acmeWirelessLed;

void
main(void)
{
    LedDriver *ledA, *ledB;

    X10Led_init(&x10Led, X10_A, 8);
    AcmeWirelessLed_init(&acmeWirelessLed, "Home", "123abc", 4);

    ledA = &x10Led.base;          /* ledA = (X10Led *)&x10Led; */
    ledB = &acmeWirelessLed.base; /* ledB = */
                                  /* (AcmeWirelessLed *)&acmeWirelessLed; */

    LedDriver_turnOn(ledA);
    LedDriver_turnOff(ledA);

    LedDriver_turnOn(ledB);
    LedDriver_turnOff(ledB);

    ledA = (LedDriver *)&acmeWirelessLed;
    LedDriver_turnOn(ledA);
    LedDriver_turnOff(ledA);

    getchar();
}
```

- La versión 1 resuelve la vinculación mediante, una tabla de funciones virtuales, y el miembro offset de `LedDriver`.
- La Inicialización de las instancias suele suceder en boot-time o en configuración.
- Las funciones de `LedDriver` se denominan polimórficas (también funciones virtuales u operaciones abstractas).

Vinculación dinámica
Instancias múltiples – Versión 1

```
10 typedef struct LedDriverVtbl LedDriverVtbl;
11 typedef struct LedDriver LedDriver;
12
13 struct LedDriver
14 {
15     size_t offset;
16     const LedDriverVtbl *vptr;
17     const char *type;
18     int id;
19 };
20
21 struct LedDriverVtbl
22 {
23     void (*turnOn)(LedDriver *const me);
24     void (*turnOff)(LedDriver *const me);
25 };
26
27 void LedDriver_turnOn(LedDriver *const me);
28 void LedDriver_turnOff(LedDriver *const me);
29 const char *LedDriver_getType(LedDriver *const me);
30 int LedDriver_getId(LedDriver *const me);
```

Especificación de LedDriver

- `LedDriver` es la abstracción que desacopla a `Client` de la implementación concreta.

```
 5 #include "LedDriver.h"
 6
 7 void
 8 LedDriver_turnOn(LedDriver *const me)
 9 {
10     if (me)
11     {
12         const LedDriverVtbl *vptr = me->vptr;
13         size_t addr = (size_t)me;
14         void *realMe = (void *)(addr - me->offset);
15         (*vptr->turnOn)(realMe);
16     }
17 }
```

Implementación de LedDriver

- El módulo `LedDrive` administra múltiples instancias de diferentes tipos (clases derivadas), identificadas por el parámetro `me`.

Vinculación dinámica
Instancias múltiples – Versión 1

```
1  /*
2   *  AcmeWirelessLed.h
3   */
4
5  #ifndef __ACMEWIRELESSLED_H__
6  #define __ACMEWIRELESSLED_H__
7
8  #include "LedDriver.h"
9
10 typedef struct AcmeWirelessLed AcmeWirelessLed;
11 struct AcmeWirelessLed
12 {
13     LedDriver base;
14     const char *ssid;
15     const char *key;
16     int channel;
17 };
18
19 void AcmeWirelessLed_init(AcmeWirelessLed *const me, const char *ssid,
20                                                      const char *key,
21                                                      int channel);
22
23 #endif
```

- A través de `me`, cada operación identifica la instancia sobre la cual opera.
- Cada módulo se crea a partir de `LedDriver` mediante la herencia simple, incluyendo la clase padre en la clase derivada.
- Es pública la definición de `AcmeWirelessLed`.

Vinculación dinámica
Instancias múltiples – Versión 1

```
29 static void
30 turnOff(LedDriver *const me)
31 {
32     AcmeWirelessLed *realMe = (AcmeWirelessLed *)me;
33
34     sendMessage(realMe, "TurnOff");
35 }
36
37 void
38 AcmeWirelessLed_init(AcmeWirelessLed *const me, const char *ssid,
39                                                  const char *key, in
40 {
41     static const LedDriverVtbl vtbl = {turnOn, turnOff};
42
43     LedDriver *base = &me->base;
44
45     base->id = 1;
46     base->type = "Acme wireless";
47     base->offset = offsetof(AcmeWirelessLed, base);
48     base->vptr = &vtbl;
49     me->ssid = ssid;
50     me->key = key;
51     me->channel = channel;
52 }
```

Implementación de AcmeWirelessLed

- La operaciones se definen en la tabla de funciones virtuales `vtbl`
- Dentro de cada operación se realiza un *downcast* para acceder a los datos de la clase derivada.
- La macro `offset()` estable la ubicación de la estructura derivada

Vinculación dinámica
Instancias múltiples – Versión 2

```
1  /*
2   *  AcmeWirelessLed.h
3   */
4
5  #ifndef __ACMEWIRELESSLED_H__
6  #define __ACMEWIRELESSLED_H__
7
8  #include "LedDriver.h"
9
10 typedef struct AcmeWirelessLed AcmeWirelessLed;
11 struct AcmeWirelessLed
12 {
13     const char *ssid;
14     const char *key;
15     int channel;
16     LedDriver base;
17 };
18
19 void AcmeWirelessLed_init(AcmeWirelessLed *const me, const char *ssid,
20                                                      const char *key,
21                                                      int channel);
22
23 #endif
```

- Igual que la versión 1 pero cambia la ubicación del miembro `base` de las clases derivadas, demostrado que este método es independiente al *layout* de las clases derivadas, gracias a `offset` de `LedDriver`.

Vinculación dinámica
Instancias múltiples – Versión 2

```
 5 #include <stdio.h>
 6 #include "X10Led.h"
 7 #include "AcmeWirelessLed.h"
 8
 9 static X10Led x10Led;
10 static AcmeWirelessLed acmeWirelessLed;
11
12 void
13 main(void)
14 {
15     LedDriver *ledA, *ledB;
16
17     X10Led_init(&x10Led, X10_A, 8);
18     AcmeWirelessLed_init(&acmeWirelessLed, "Home", "123abc", 4);
19
20     ledA = &x10Led.base;
21     ledB = &acmeWirelessLed.base;
22
23     LedDriver_turnOn(ledA);
24     LedDriver_turnOff(ledA);
25
26     LedDriver_turnOn(ledB);
27     LedDriver_turnOff(ledB);
28
29     ledA = &acmeWirelessLed.base;
30     LedDriver_turnOn(ledA);
31     LedDriver_turnOff(ledA);
32
33     getchar();
34 }
```

Inicialización y uso para versión 2

- Uso e inicialización igual que la versión 1.
- Observar como cambia el valor de `ledA` en tiempo de ejecución.

Vinculación dinámica
Instancias múltiples – Versión 3

```
1  /*
2   *  AcmeWirelessLed.h
3   */
4
5  #ifndef __ACMEWIRELESSLED_H__
6  #define __ACMEWIRELESSLED_H__
7
8  #include "LedDriver.h"
9  #include "DoorLockDriver.h"
10
11 typedef struct AcmeWirelessLed AcmeWirelessLed;
12 struct AcmeWirelessLed
13 {
14     LedDriver ledDriver;
15     DoorLockDriver lockDriver;
16     const char *ssid;
17     const char *key;
18     int channel;
19 };
20
21 void AcmeWirelessLed_init(AcmeWirelessLed *const me, const char *ssid,
22                                                      const char *key,
23                                                      int channel);
24
25 #endif
```

- Igual que la versión 1 pero ahora `AcmeWirelessLed` deriva de las clases `LedDriver` y `DoorLockDriver`

Vinculación dinámica
Instancias múltiples – Versión 3

```
 9 static X10Led x10Led;
10 static AcmeWirelessLed acmeWirelessLed;
11
12 void
13 main(void)
14 {
15     LedDriver *ledA, *ledB;
16     DoorLockDriver *foo;
17
18     ledA = (LedDriver *)&x10Led;
19     ledB = &acmeWirelessLed.ledDriver;
20     foo = &acmeWirelessLed.lockDriver;
21
22     X10Led_init(&x10Led, X10_A, 8);
23     AcmeWirelessLed_init(&acmeWirelessLed, "Home", "123abc", 4);
24
25     LedDriver_turnOn(ledA);
26     LedDriver_turnOff(ledA);
27
28     LedDriver_turnOn(ledB);
29     LedDriver_turnOff(ledB);
30
31     DoorLockDriver_lock(foo);
32     DoorLockDriver_unlock(foo);
33
34     getchar();
35 }
```

Inicialización y uso para versión 3

Vinculación dinámica
Instancias múltiples – Versión 4

```c
/*
 *  main.c
 */

#include <stdio.h>
#include "X10Led.h"
#include "AcmeWirelessLed.h"

static X10Led x10Led;
static AcmeWirelessLed acmeWirelessLed;

void
main(void)
{
    LedDriver *ledA, *ledB;

    X10Led_init(&x10Led, X10_A, 8);
    AcmeWirelessLed_init(&acmeWirelessLed, "Home", "123abc", 4);

    ledA = (LedDriver *)&x10Led;
    ledB = (LedDriver *)&acmeWirelessLed;

    LedDriver_turnOn(ledA);
    LedDriver_turnOff(ledA);

    LedDriver_turnOn(ledB);
    LedDriver_turnOff(ledB);

    getchar();
}
```

- Igual que la versión 1 pero utilizando `void*` en lugar de `LedDriver*` en las operaciones de `LedDriver`.
- No es totalmente compatible con las versiones 1 y 2, ya que es dependiente del *layout* de la estructura `LedDriver`.

Vinculación dinámica
Instancias múltiples – Versión 4

```c
/*
 *  LedDriver.h
 */

#ifndef __LEDDRIVER_H__
#define __LEDDRIVER_H__

#include <stddef.h>

typedef struct LedDriverVtbl LedDriverVtbl;
typedef struct LedDriver LedDriver;

struct LedDriver
{
    size_t offset;
    const LedDriverVtbl *vptr;
    const char *type;
    int id;
};

struct LedDriverVtbl
{
    void (*turnOn)(void *const me);
    void (*turnOff)(void *const me);
};

void LedDriver_turnOn(void *const me);
void LedDriver_turnOff(void *const me);
const char *LedDriver_getType(LedDriver *const me);
int LedDriver_getId(LedDriver *const me);

#endif
```

```c
/*
 *  LedDriver.c
 */

#include "LedDriver.h"

void
LedDriver_turnOn(void *const me)
{
    if (me)
    {
        const LedDriverVtbl *vptr = ((LedDriver *)me)->vptr;
        size_t addr = (size_t)me;
        void *realMe = (void *)(addr - ((LedDriver *)me)->offset);
        (*vptr->turnOn)(realMe);
    }
}
```

```c
#include "LedDriver.h"

typedef struct AcmeWirelessLed AcmeWirelessLed;
struct AcmeWirelessLed
{
    LedDriver base;
    const char *ssid;
    const char *key;
    int channel;
};
```

Especificación AcmeWirelessLed

Vinculación dinámica
Instancias múltiples – Versión 5

```
1  /*
2   *  LedDriver.h
3   */
4
5  #ifndef __LEDDRIVER_H__
6  #define __LEDDRIVER_H__
7
8  #include <stddef.h>
9
10 typedef struct LedDriverVtbl LedDriverVtbl;
11 typedef struct LedDriver LedDriver;
12
13 struct LedDriver
14 {
15     const LedDriverVtbl *vptr;
16     const char *type;
17     int id;
18 };
19
20 struct LedDriverVtbl
21 {
22     void (*turnOn)(LedDriver *const me);
23     void (*turnOff)(LedDriver *const me);
24 };
25
26 void LedDriver_turnOn(LedDriver *const me);
27 void LedDriver_turnOff(LedDriver *const me);
28 const char *LedDriver_getType(LedDriver *const me);
29 int LedDriver_getId(LedDriver *const me);
30
31 #endif
```

- Igual que la versión 1, pero NO utiliza la macro `offset()` en `LedDriver`
- El método se basa en el *layout* de la estructura derivada, donde su primer miembro siempre es del tipo de su clase padre.

Vinculación dinámica
Instancias múltiples – Versión 5

```c
/*
 *  main.c
 */

#include <stdio.h>
#include "X10Led.h"
#include "AcmeWirelessLed.h"

static X10Led x10Led;
static AcmeWirelessLed acmeWirelessLed;

void
main(void)
{
    LedDriver *ledA, *ledB;

    X10Led_init(&x10Led, X10_A, 8);
    AcmeWirelessLed_init(&acmeWirelessLed, "Home", "123abc", 4);

    ledA = (LedDriver *)&x10Led;
    ledB = (LedDriver *)&acmeWirelessLed;

    LedDriver_turnOn(ledA);
    LedDriver_turnOff(ledA);

    LedDriver_turnOn(ledB);
    LedDriver_turnOff(ledB);

    getchar();
}
```

```c
/*
 *  LedDriver.c
 */

#include "LedDriver.h"

void
LedDriver_turnOn(LedDriver *const me)
{
    if (me)
    {
        (*me->vptr->turnOn)(me);
    }
}

void
LedDriver_turnOff(LedDriver *const me)
{
    if (me)
    {
        (*me->vptr->turnOff)(me);
    }
}
```

Vinculación dinámica
Instancias múltiples – Versión 6

```c
/*
 *  main.c
 */

#include <stdio.h>
#include "X10Led.h"
#include "AcmeWirelessLed.h"

void
main(void)
{
    LedDriver *ledA, *ledB;

    X10Led_init(x10LedA, X10_A, 8);
    X10Led_init(x10LedB, X10_B, 6);
    AcmeWirelessLed_init(acmeWirelessLed, "Home", "123abc", 4);

    ledA = (LedDriver *)x10LedA;
    ledB = (LedDriver *)acmeWirelessLed;

    LedDriver_turnOn(ledA);
    LedDriver_turnOff(ledA);

    LedDriver_turnOn(ledB);
    LedDriver_turnOff(ledB);

    ledA = (LedDriver *)x10LedB;
    LedDriver_turnOn(ledA);
    LedDriver_turnOff(ledA);

    getchar();
}
```

- Igual que la versión 1, pero oculta la definición de las clases derivadas.
- Para lo cual utiliza punteros opacos.
- Las asignación de memoria para las instancias se realiza de manera estática dentro de la implementación.
- Siempre es necesario instanciar dentro de la implementación.

Vinculación dinámica
Instancias múltiples – Versión 6

```
1  /*
2   *  AcmeWirelessLed.h
3   */
4
5  #ifndef __ACMEWIRELESSLED_H__
6  #define __ACMEWIRELESSLED_H__
7
8  #include "LedDriver.h"
9
10 typedef struct AcmeWirelessLed AcmeWirelessLed;
11 extern AcmeWirelessLed *acmeWirelessLed;
12
13 void AcmeWirelessLed_init(AcmeWirelessLed *const me, const
14                                                       const
15                                                       int ch
16
17 #endif
```

```
1  /*
2   *  AcmeWirelessLed.c
3   */
4
5  #include <stdio.h>
6  #include "AcmeWirelessLed.h"
7
8  struct AcmeWirelessLed
9  {
10     LedDriver base;
11     const char *ssid;
12     const char *key;
13     int channel;
14 };
15
16 static AcmeWirelessLed acmeWirelessLedObj;
17 AcmeWirelessLed *acmeWirelessLed = &acmeWirelessLedObj;
18
```

- La especificación de las clases derivadas utilizan un tipo incompleto para especificar su clase.
- En la implementación se define el tipo completo de su clase.

Vinculación dinámica
Instancias múltiples – Versión 7

```
 8 #include "X10Led.h"
 9 #include "AcmeWirelessLed.h"
10
11 static X10Led *ledA;
12 static AcmeWirelessLed *ledB;
13
14 int
15 main(void)
16 {
17     ledA = X10Led_ctor(X10_A, 8);
18
19     ledB = AcmeWirelessLed_ctor("Home", "123abc", 4);
20     if (ledB == (AcmeWirelessLed *)0)
21     {
22         printf("Can't create a AcmeWirelessLed object\r\n");
23         exit(EXIT_FAILURE);
24     }
25
26     LedDriver_turnOn((LedDriver *)ledA);
27     LedDriver_turnOff((LedDriver *)ledA);
28
29     LedDriver_turnOn((LedDriver *)ledB);
30     LedDriver_turnOff((LedDriver *)ledB);
31
32     AcmeWirelessLed_dtor(ledB);
33     X10Led_dtor(ledA);
34
35     getchar();
36     return 0;
37 }
```

Inicialización y uso para versión 7

- Igual que la versión 4, pero instancia estáticamente mediante un pool dentro de la implementación.
- Es necesario verifica de alguna manera el éxito de la asignación de memoria de las instancias.

Vinculación dinámica
Instancias múltiples – Versión 7

```
 8 struct AcmeWirelessLed
 9 {
10     LedDriver base;
11     const char *ssid;
12     const char *key;
13     int channel;
14 };
15
16 static AcmeWirelessLed pool[ACME_MAX_NUM_LEDS];
```

- `pool[]` mantiene una cantidad acotada de posibles instancias.
- La operación especial `ctor()` tiene por objetivo la asignación de memoria mediante el *pool*.

```
53 AcmeWirelessLed *
54 AcmeWirelessLed_ctor(const char *ssid, const char *key, int channel)
55 {
56     static const LedDriverVtbl vtbl = {turnOn, turnOff};
57     LedDriver *base;
58     AcmeWirelessLed *slot;
59
60     for (slot = pool; slot < pool + ACME_MAX_NUM_LEDS; ++slot)
61     {
62         base = &slot->base;
63         if (slot->base.vptr == (LedDriverVtbl *)0)
64         {
65             base->id = 1;
66             base->type = "Acme wireless";
67             base->vptr = &vtbl;
68             slot->ssid = ssid;
69             slot->key = key;
70             slot->channel = channel;
71             return slot;
72         }
73     }
74     return (AcmeWirelessLed *)0;
75 }
```

Implementación de la clase derivada AcmeWirelessLed

Vinculación dinámica
Instancias múltiples – Versión 8

```
 9 static X10Led x10LedA, x10LedB;
10 static AcmeWirelessLed acmeWirelessLedA, acmeWirelessLedB;
11
12 void
13 main(void)
14 {
15     LedDriver *ledA, *ledB;
16
17     X10Led_init(&x10LedA, X10_A, 8);
18     X10Led_init(&x10LedB, X10_B, 4);
19
20     ledA = (LedDriver *)&x10LedA;
21     ledB = (LedDriver *)&x10LedB;
22
23     LedDriver_turnOn(ledA);
24     LedDriver_turnOn(ledB);
25
26     AcmeWirelessLed_init(&acmeWirelessLedA, "Home", "123abc", 4);
27     AcmeWirelessLed_init(&acmeWirelessLedB, "Home", "5231aa", 2);
28
29     ledA = (LedDriver *)&acmeWirelessLedA;
30     ledB = (LedDriver *)&acmeWirelessLedB;
31
32     LedDriver_turnOn(ledA);
33     LedDriver_turnOn(ledB);
34
35     getchar();
36 }
```

Inicialización y uso para versión 8

- Igual que la versión 3, pero soporta una única implementación de la abstracción al mismo tiempo.
- El binario ejecutable permite elegir únicamente la implementación para todos los LEDs en uso.

Vinculación dinámica
Instancias múltiples – Versión 8

```c
/*
 *  LedDriver.h
 */

#ifndef __LEDDRIVER_H__
#define __LEDDRIVER_H__

#include <stddef.h>

typedef struct LedDriverVtbl LedDriverVtbl;
typedef struct LedDriver LedDriver;
struct LedDriver
{
    const char *type;
    int id;
};

struct LedDriverVtbl
{
    void (*turnOn)(LedDriver *const me);
    void (*turnOff)(LedDriver *const me);
};

void LedDriver_turnOn(LedDriver *const me);
void LedDriver_turnOff(LedDriver *const me);
const char *LedDriver_getType(LedDriver *const me);
int LedDriver_getId(LedDriver *const me);
void LedDriver_setInterface(const LedDriverVtbl *interface);

#endif
```

```c
/*
 *  LedDriver.c
 */

#include "LedDriver.h"

static const LedDriverVtbl *vptr = (LedDriverVtbl *)0;

void
LedDriver_setInterface(const LedDriverVtbl *interface)
{
    vptr = interface;
}

void
LedDriver_turnOn(LedDriver *const me)
{
    if (me)
    {
        (*vptr->turnOn)(me);
    }
}
```

- `LedDriver` no utiliza el puntero a la tabla de funciones virtuales

Vinculación dinámica
Instancias múltiples – Versión 8

```
37 void
38 AcmeWirelessLed_init(AcmeWirelessLed *const me, const char *ssid,
39                                                 const char *key, int channel)
40 {
41     static const LedDriverVtbl vtbl = {turnOn, turnOff};
42     LedDriver *base = &me->base;
43
44     base->id = 1;
45     base->type = "Acme wireless";
46     me->ssid = ssid;
47     me->key = key;
48     me->channel = channel;
49     LedDriver_setInterface(&vtbl);
50 }
```

- La operación especial `init()` fija la implementación para todas las instancias de LEDs, por lo tanto no es posible elegir el tipo de manera individual.

Principios S.O.L.I.D

- Cada uno de los criterios y métodos presentados se basan fundamentalmente en estos principios.
- Estos forman parte de la estrategia del desarrollo ágil y adaptativo de software.

Inicial	Acrónimo	Concepto
S	SRP	Principio de responsabilidad única (*Single Responsibility Principle*). Una clase debería tener una única razón para cambiar.
O	OCP	Principio de abierto/cerrado (*Open/Closed Principle*). Las entidades de software (clases, módulos, funciones, etc) deben estar abiertas para su extensión, pero cerradas para su modificación.
L	LSP	Principio de sustitución de Liskov (*Liskov Substitution Principle*). Los subtipos deben ser sustituibles por sus tipos bases.
I	ISP	Principio de segregación de la interfaz (*Interface Segregation Principle*). la noción de que "muchas interfaces cliente específicas son mejores que una interfaz de propósito general"
D	DIP	Principio de inversión de la dependencia (*Dependency Inversion Principle*). Las abstracciones no deberían depender de los detalles. Los detalles deberían depender de las abstracciones.

Conclusiones

- La aplicación de los principios del OOD, como el encapsulamiento, la abstracción, la herencia, y el polimorfismo, como fundamentos del diseño de embedded software en C, no es una imitación caprichosa de lenguajes orientados a objetos, sino más bien es la aplicación de ideas y conceptos maduros y formalizados, que bien empleados y en su “justa medida” permiten no sólo **aumentar** la **productividad** sino también la **calidad** de nuestros desarrollos.
- Y así lograr software **más flexible**, **fácil de mantener** y **reutilizable**.

Preguntas

Referencias

[1] Leandro Francucci, “Principios de OOP aplicados en C para Embedded Systems”, Febrary, 2014, embedded-exploited.com.ar

[2] Leandro Francucci, “Modularidad, abstracción y múltiples instancias en C para Embedded Software”, April, 2014, embedded-exploited.com.ar

[3] Dan Saks, "Alternatives Idioms for Inheritance in C", Embedded.com, April 2, 2013.

[4] Robert C. Martin, “Clean Code”, Prentice Hall, 2009

[5] Robert C. Martin, “Agile Principles Patterns and Practices in C#”, Pearson Education, 20 jul. 2006

[6] James W. Grenning, “Test Driven Development for Embedded C”, Pragmatic Bookshelf, 2011

[7] Kernighan & Ritchie, "C Programming Language (2nd Edition)", April 1, 1988.

[8] Repositorio Git con los ejemplos: https://github.com/vortextech/ood.git