OBJECTIVE-C RECORDATORIO Y DETALLES ADICIONALES

Desarrollo de Aplicaciones Avanzadas 2015-2016

Universidad de Salamanca
Grado en Ingeniería Informática
Dr. J.R. García-Bermejo Giner

OBJECTIVE-C

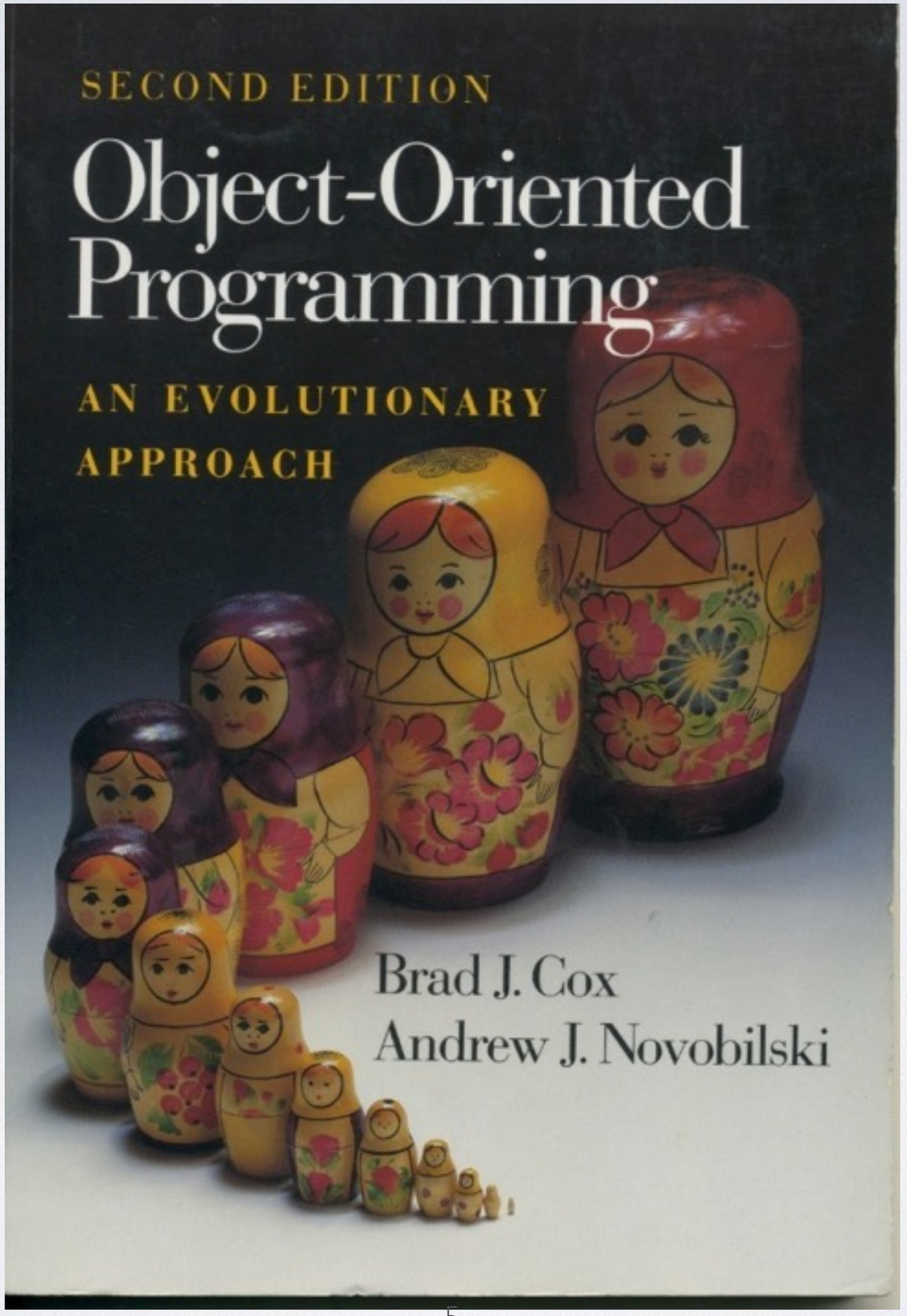
- Primera versión curso 2012-2013
- Segunda versión curso 2015-2016

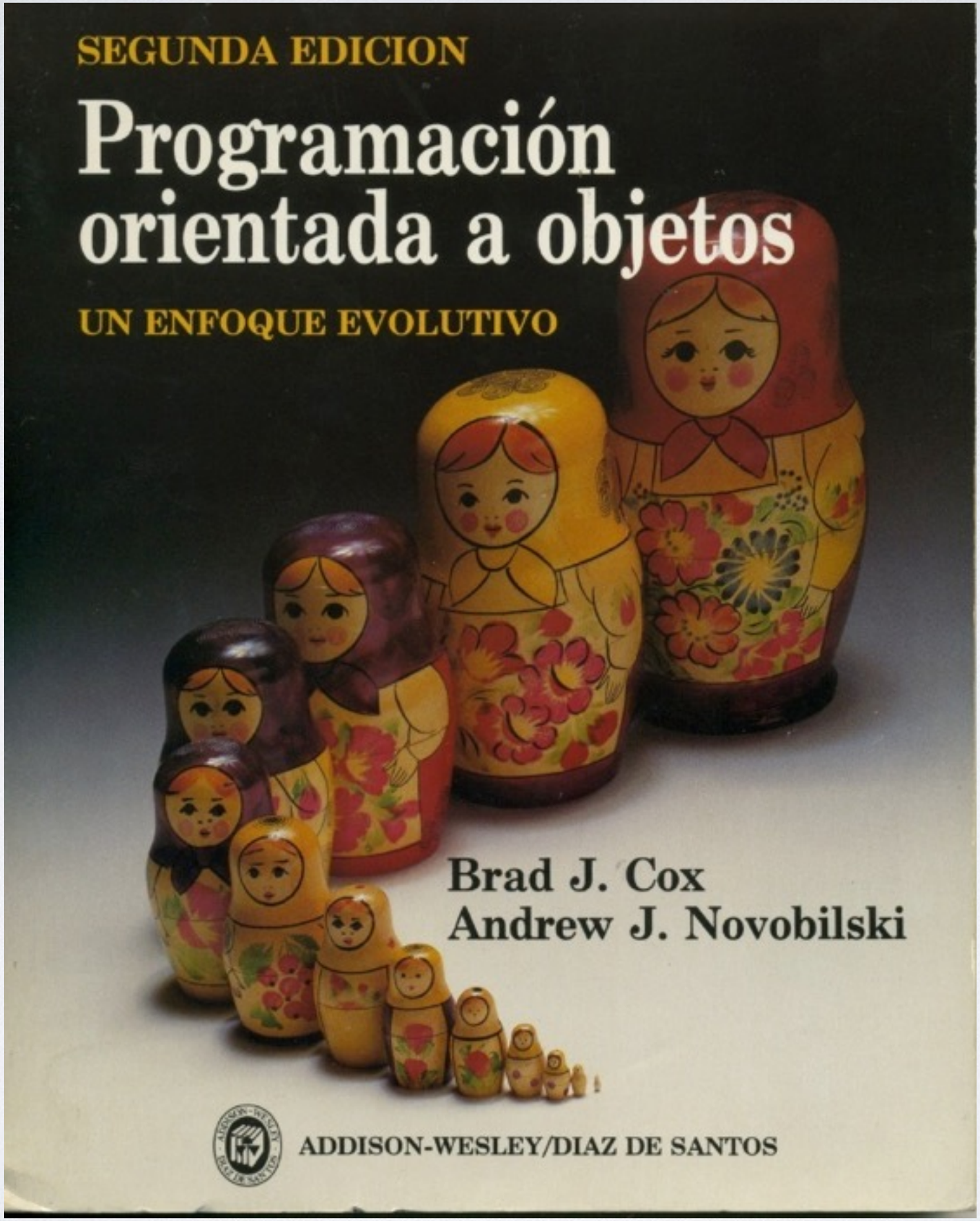
ORIGEN DE OBJECTIVE-C

OBJECTIVE-C

- Es una extensión de C que forma parte de los compiladores de GNU.

- Creador: Bradford Cox, ca. 1980. Quizá el libro más conocido sea el de 1991, con A. Novobilski (página siguiente)


SECOND EDITION
Object-Oriented Programming
AN EVOLUTIONARY APPROACH
Brad J. Cox
Andrew J. Novobilski


SEGUNDA EDICION
Programación orientada a objetos
UN ENFOQUE EVOLUTIVO
Brad J. Cox
Andrew J. Novobilski
ADDISON-WESLEY/DIAZ DE SANTOS

Programación orientada a objetos

Un enfoque evolutivo

Brad J. Cox
Andrew J. Novobilski
The Stepstone Corporation

Versión en español de:
Rafael García-Bermejo Giner
Universidad de Salamanca
Salamanca, España

Con la colaboración de:
Luis Joyanes Aguilar
Universidad Pontificia de Salamanca
Campus de Madrid, España

ADDISON-WESLEY / DIAZ DE SANTOS

Argentina · Brasil · Chile · Colombia · Ecuador · España
Estados Unidos · México · Perú · Puerto Rico · Venezuela

OBJECTIVE-C

El libro de Cox describe pormenorizadamente el funcionamiento interno del lenguaje, y no resulta muy agradable a efectos prácticos. Es parecido a "**`The Objective-C Programming Language`**", con respecto a C, con un énfasis que está más en las bondades del lenguaje que en su utilización.

El libro describe la versión de Objective-C existente en 1996; en la actualidad se ha llegado a la versión 2.0 de Objc, que posee numerosas mejoras y adiciones.

REFERENCIA

OBJECTIVE-C

La referencia definitiva en lo tocante al lenguaje Objective-C 2.0 se encontrará aquí:

(https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html)

The Objective-C Programming Language

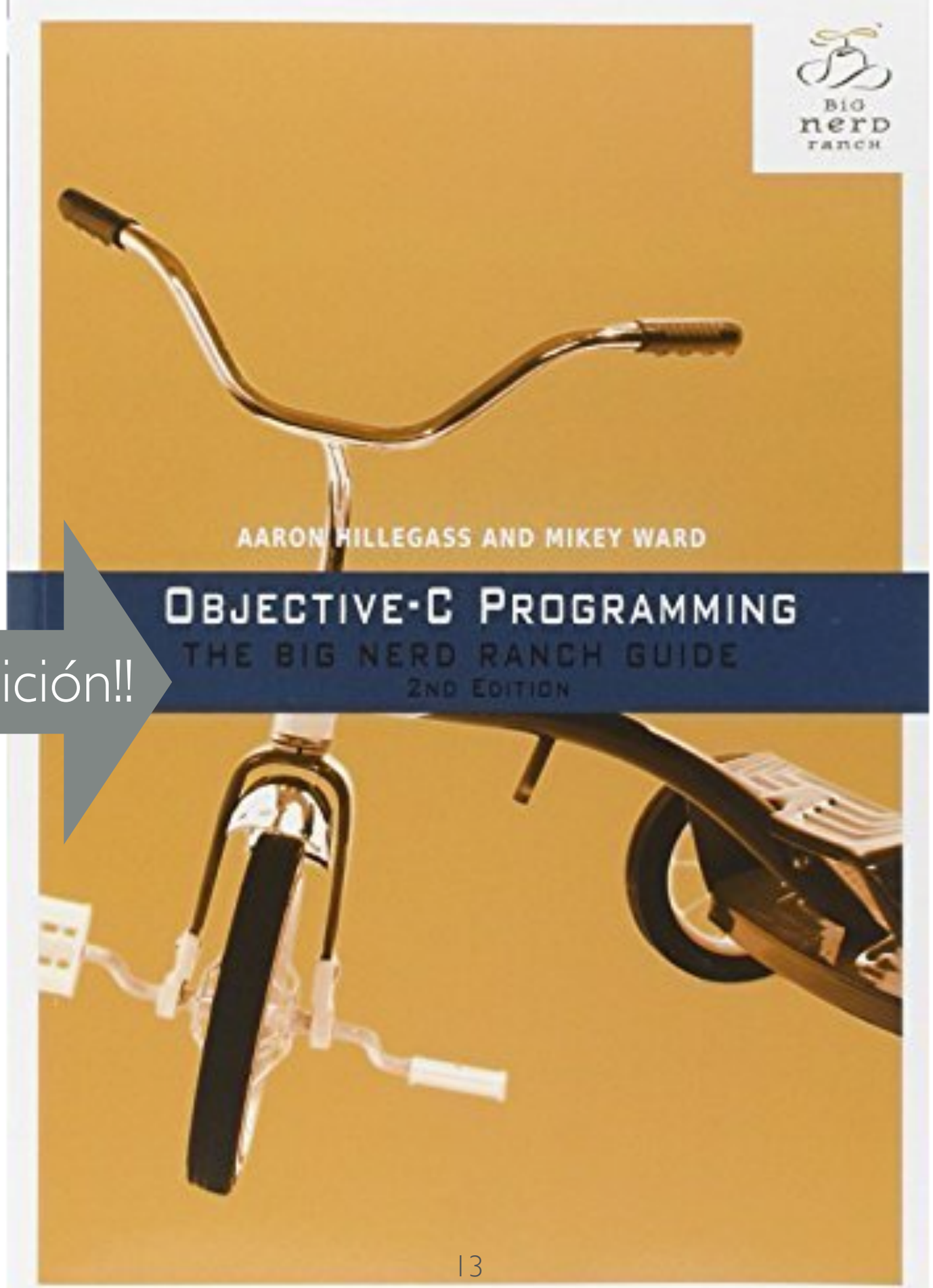
BIBLIOGRAFÍA

OBJECTIVE-C

En la actualidad, un buen libro para estudiar el lenguaje es:

`Objective-C Programming: The Big Nerd Ranch Guide`.

Es recomendable porque expone las características del lenguaje más utilizadas en Cocoa, con muchos ejemplos y con un enfoque más práctico que formal. (Ver página siguiente)



Segunda Edición!!

OBJECTIVE-C

Ojo: hay que pedir la segunda edición, no la primera que es de 2011. La segunda es de 2013, y posiblemente no haya más porque Swift ha llegado hace ya un año.

SWIFT

SWIFT

Ya están disponibles libros sobre Swift, aunque la versión inglesa tardará algunos meses, y la española algunos mas.

El libro de Holger Hinzberg es agradable y tiene ejemplos comprensibles.

Holger Hinzberg

mitp

Einführung in Swift

Mit Referenzkarte

- Se hallarán muchas posibilidades en www.amazon.com, incluyendo una de Hillegass para el 10 de enero

CREAR EJEMPLARES EN OBJECTIVE-C

CREACIÓN DE EJEMPLARES

La sintaxis que se utiliza para crear un ejemplar de una clase en Objective-C es la siguiente:

```
NSMutableArray * ma;
ma = [NSMutableArray alloc];
```

(Como se recordará, `NSMutableArray` es una clase de Cocoa cuya funcionalidad es similar a la de un `Vector` de Java. La colección contiene punteros de los objetos añadidos; al añadir un objeto a una colección, aumenta en 1 su contador de retención).

CREACIÓN DE EJEMPLARES

El método **`alloc`** solamente reserva memoria para el objeto, pero no da valores iniciales a sus campos. De eso se encarga el método **`init`**. Por tanto, para poder utilizar este ejemplar, sería preciso escribir:

```
NSMutableArray * ma;
ma = [NSMutableArray alloc];
[ma init];
```

Como **`alloc`** proporciona un puntero del objeto que ha creado, se pueden combinar las dos llamadas en una sola:

```
NSMutableArray * ma = [[NSMutableArray alloc] init];
```

que es la forma que se emplea normalmente. Existe también la posibilidad de emplear la macro **`new`**, que efectúa las dos llamadas en un solo paso:

```
NSMutableArray * ma = [NSMutableArray new];
```

METODOS EN OBJECTIVE-C

USO DE MÉTODOS

La clase `NSMutableArray` ofrece una amplia gama de métodos (véase la documentación). Para añadir un objeto a `ma`, se utiliza una expresión como la siguiente:

```
NSString * pensamiento = @“María tiene una oveja”;
NSString * otro = “tan blanca como la nieve”;
[ma addObject:pensamiento];
```

`NSMutableArray` permite insertar objetos en una posición determinada, empleando el método `insertObject:atIndex:`. Se utiliza en la forma siguiente:

```
[ma insertObject:otro atIndex:5];
```

USO DE MÉTODOS

Java	Objective-C
`String cadena = “Hola”;` `int n = cadena.length();`	`NSString *cadena=@”hola”;` `unsigned long n = [cadena length];`

La conocida expresión de java

```
cadena.length()
```

se transforma en su equivalente en Objective-C cambiando el punto por un espacio, y los paréntesis al final de `length()` por unos corchetes que engloban toda la expresión:

```
[cadena length];
```

USO DE MÉTODOS

El cambio de notación anterior, pasando del operador "." al operador "[ ]" es la diferencia sintáctica básica entre Objective-C y Java.

Al faltar los paréntesis que denotan llamada a un método, y que contienen los argumentos, Objective-C utiliza el signo ":" antes de cada argumento. Normalmente, antes del signo ":" se escribe una etiqueta opcional, que permite saber la misión del argumento.

USO DE MÉTODOS

Java	Objective-C

`String cadena = “Hola”;`
`Vector v = new Vector;`
`v.add(cadena);`
	`NSString *cadena=@”hola”;`
	`NSMutableArray * ma = [NSMutableArray new];`
	`[ma addObject:cadena];`

Este ejemplo muestra la manera de efectuar una llamada a un objeto. Se escribe en primer lugar el puntero de la clase, **`ma`**, y después el argumento, **`cadena`**, a la derecha de la etiqueta de argumento **`addObject`**, que va seguida por **`:`**. Toda la expresión se encierra entre corchetes.

Las expresiones con varios puntos de Java se expresan en Objective-C por anidamiento de corchetes.

USO DE MÉTODOS

Java	`// La clase Ficha contiene el atributo nombre de tipo String` `int longitud = miFicha.getNombre().length();`
Objective-C	`// La clase RGBFicha contiene el atributo nombre de tipo NSString` `int longitud = [[miFicha nombre] length];`

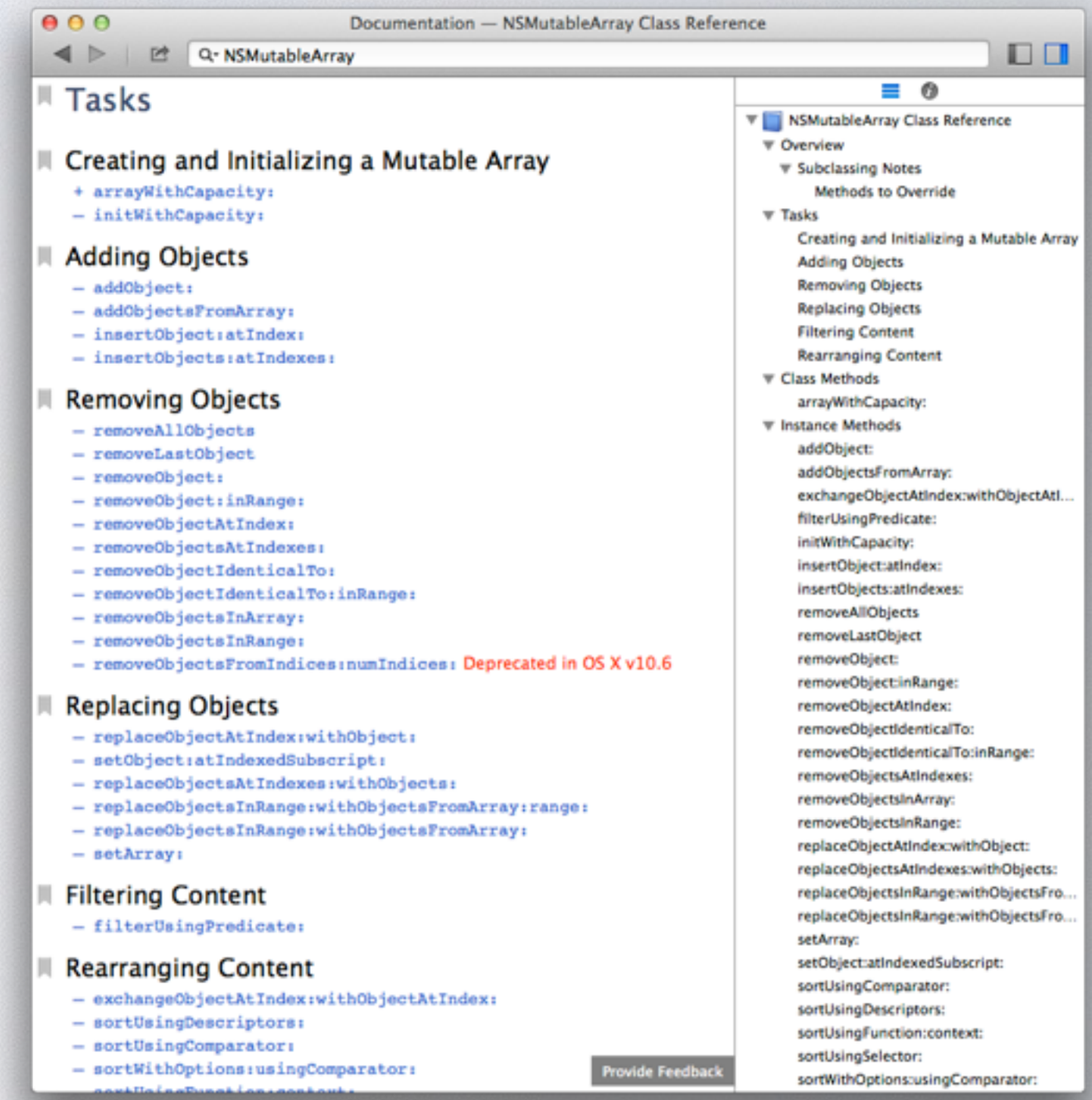
Este ejemplo muestra la manera de anidar llamadas a métodos. En efecto, se hace la llamada al método nombre de la clase **`miFicha`** para obtener su valor, y después se hace la llamada al método length para hallar su longitud.

Obsérvese, por cierto, que esta longitud es el número de caracteres Unicode que hay en la cadena. Se puede determinar también el número de bytes que tiene la cadena empleando una determinada codificación mediante **`lengthOfBytesUsingEncoding:`**. Como se recordará, los dos puntos al final del método indican que hay un argumento.

COLECCIONES

NSMUTABLEARRAY

`NSMutableArray` en la documentación (Xcode). Obsérvese que más abajo se agrupan los métodos de clase (**`static`** en Java) y los métodos de ejemplar.



NSMUTABLEARRAY

Creating and Initializing a Mutable Array

+ arrayWithCapacity:
– initWithCapacity:

Adding Objects

– addObject:
– addObjectsFromArray:
– insertObject:atIndex:
– insertObjects:atIndexes:

Removing Objects

– removeAllObjects
– removeLastObject
– removeObject:
– removeObject:inRange:
– removeObjectAtIndex:
– removeObjectsAtIndexes:
– removeObjectIdenticalTo:
– removeObjectIdenticalTo:inRange:
– removeObjectsInArray:
– removeObjectsInRange:
– removeObjectsFromIndices:numIndices: Deprecated in OS X v10.6

Replacing Objects

– replaceObjectAtIndex:withObject:
– setObject:atIndexedSubscript:
– replaceObjectsAtIndexes:withObjects:
– replaceObjectsInRange:withObjectsFromArray:range:
– replaceObjectsInRange:withObjectsFromArray:
– setArray:

Filtering Content

– filterUsingPredicate:

Rearranging Content

– exchangeObjectAtIndex:withObjectAtIndex:
– sortUsingDescriptors:
– sortUsingComparator:
– sortWithOptions:usingComparator:
– sortUsingFunction:context:
– sortUsingSelector:

CREACIÓN DE UNA HERRAMIENTA

Para probar pequeños programas (con interfaz de texto) escritos en Objective-C, hay que crear un proyecto de tipo `Command Line Tool`, Una vez especificado el nombre del proyecto, se indica la carpeta en que debe residir y todo está preparado para insertar el código fuente, como puede ser el siguiente.

EJEMPLO

Construir un programa que lea una cadena de C, del tipo con un carácter **`NULL`** como marcador de fin de cadena. Suponer que contiene un máximo de 63 bytes útiles. Crear después un ejemplar de **`NSString`** equivalente a la cadena de C, y escribir ese ejemplar en pantalla junto con su longitud.

No se utiliza **`scanf()`** porque se detiene en los espacios.

Vamos a emplear **`fgets()`** para leer toda la línea, puesto que la longitud está restringida a 64 bytes totales.

Proyecto: **`longitud_cadena`**

EJEMPLO

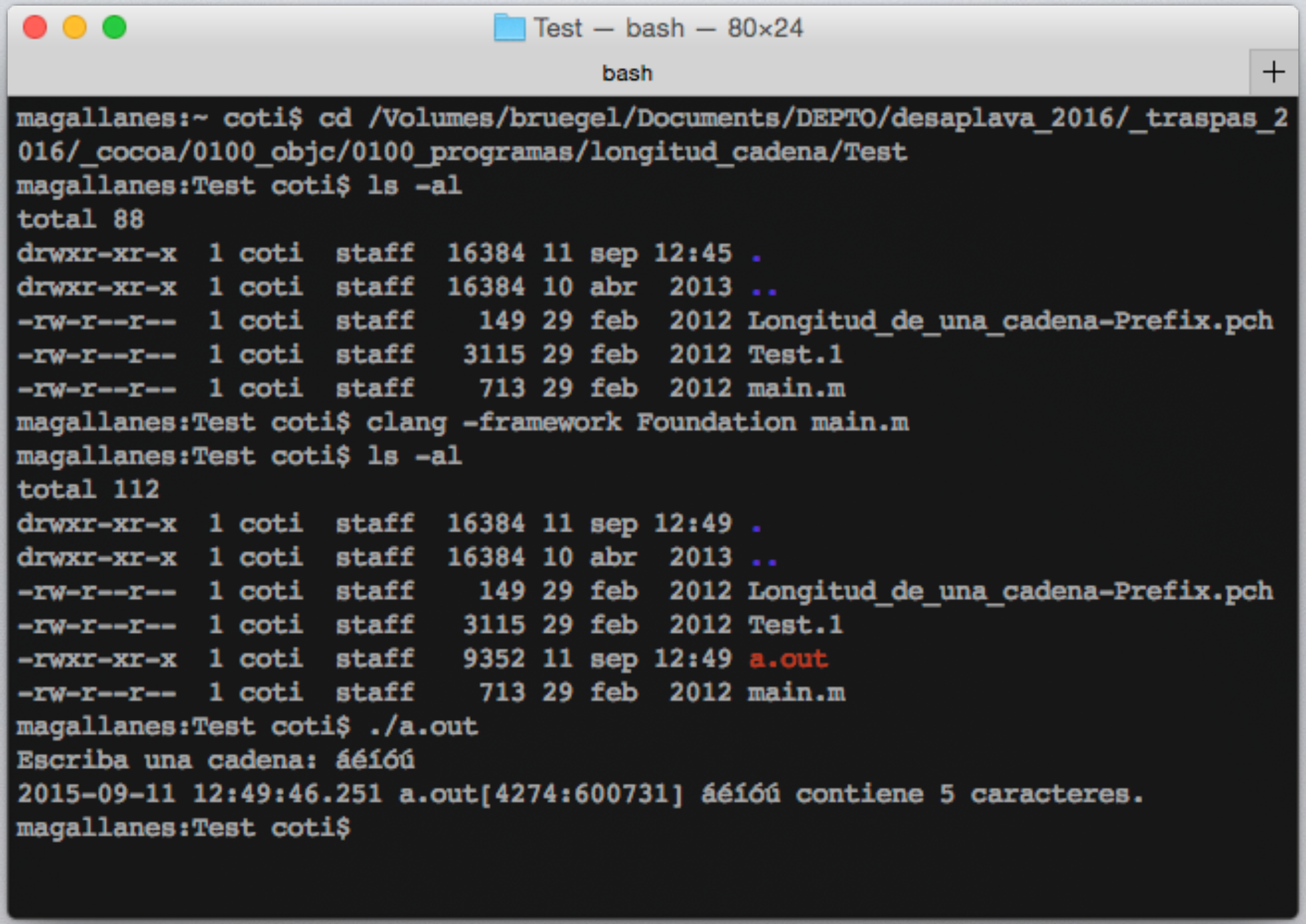
```
1. // Proyecto longitud_cadena
2. #import <Foundation/Foundation.h>

4. int main(int argc, const char * argv[])
5. {

7.     @autoreleasepool {
8.
9.         NSString * cadena;
10.         char buffer[64];
11.         printf("Escriba una cadena: ");
12.         fgets(buffer, 64, stdin);
13.         if ('\n' == buffer[strlen(buffer)-1]) {
14.             buffer[strlen(buffer)-1] = '\0';
15.         }
16.
17.         cadena = [NSString stringWithCString:buffer encoding:NSUTF8StringEncoding];
18.         unsigned long num_caracteres = [cadena length];
19.         NSLog(@"%@ contiene %lu caracteres.", cadena, num_caracteres);
20.
21.     }
22.     return 0;
23. }
```

Demo

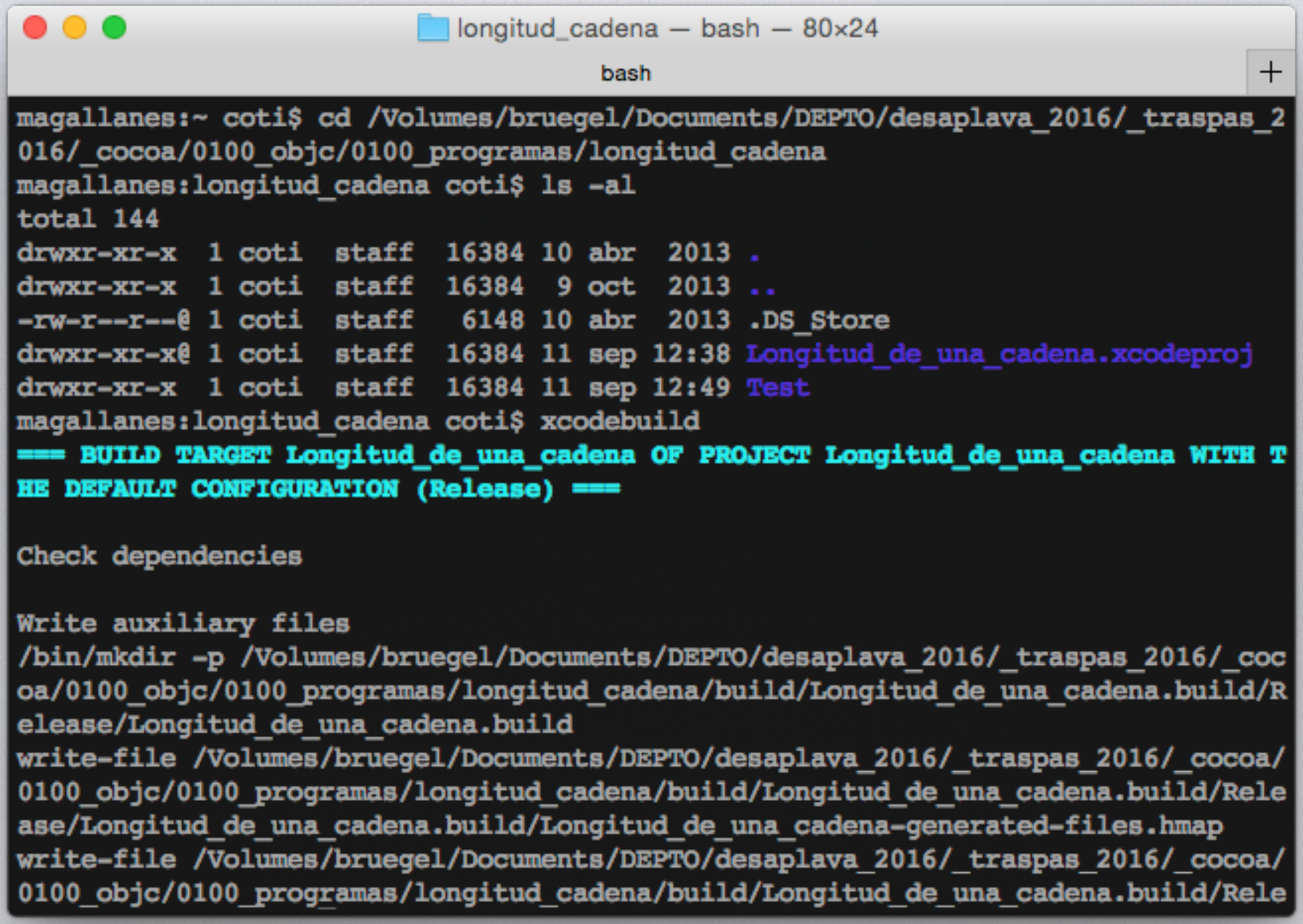
Proyecto: `longitud_cadena`

COMPILAR DESDE LA LÍNEA DE ÓRDENES

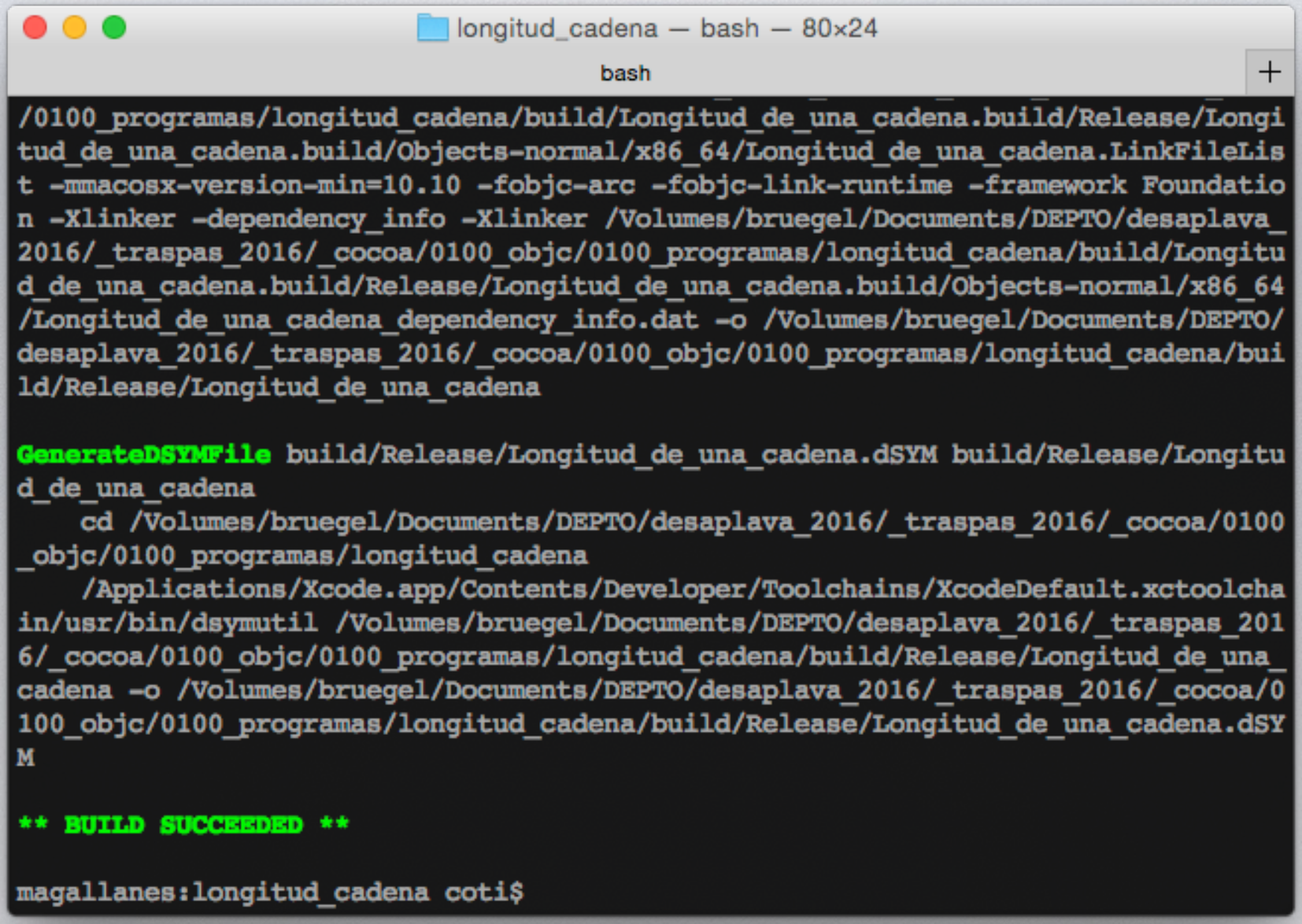
VERSIÓN UNO


Test — bash — 80×24
bash
magallanes:~ coti$ cd /Volumes/bruegel/Documents/DEPTO/desaplava_2016/_traspas_2016/_cocoa/0100_objc/0100_programas/longitud_cadena/Test
magallanes:Test coti$ ls -al
total 88
drwxr-xr-x 1 coti staff 16384 11 sep 12:45 .
drwxr-xr-x 1 coti staff 16384 10 abr 2013 ..
-rw-r--r-- 1 coti staff   149 29 feb 2012 Longitud_de_una_cadena-Prefix.pch
-rw-r--r-- 1 coti staff   3115 29 feb 2012 Test.1
-rw-r--r-- 1 coti staff   713 29 feb 2012 main.m
magallanes:Test coti$ clang -framework Foundation main.m
magallanes:Test coti$ ls -al
total 112
drwxr-xr-x 1 coti staff 16384 11 sep 12:49 .
drwxr-xr-x 1 coti staff 16384 10 abr 2013 ..
-rw-r--r-- 1 coti staff   149 29 feb 2012 Longitud_de_una_cadena-Prefix.pch
-rw-r--r-- 1 coti staff   3115 29 feb 2012 Test.1
-rwxr-xr-x 1 coti staff   9352 11 sep 12:49 a.out
-rw-r--r-- 1 coti staff   713 29 feb 2012 main.m
magallanes:Test coti$ ./a.out
Escriba una cadena: áéíóú
2015-09-11 12:49:46.251 a.out[4274:600731] áéíóú contiene 5 caracteres.
magallanes:Test coti$

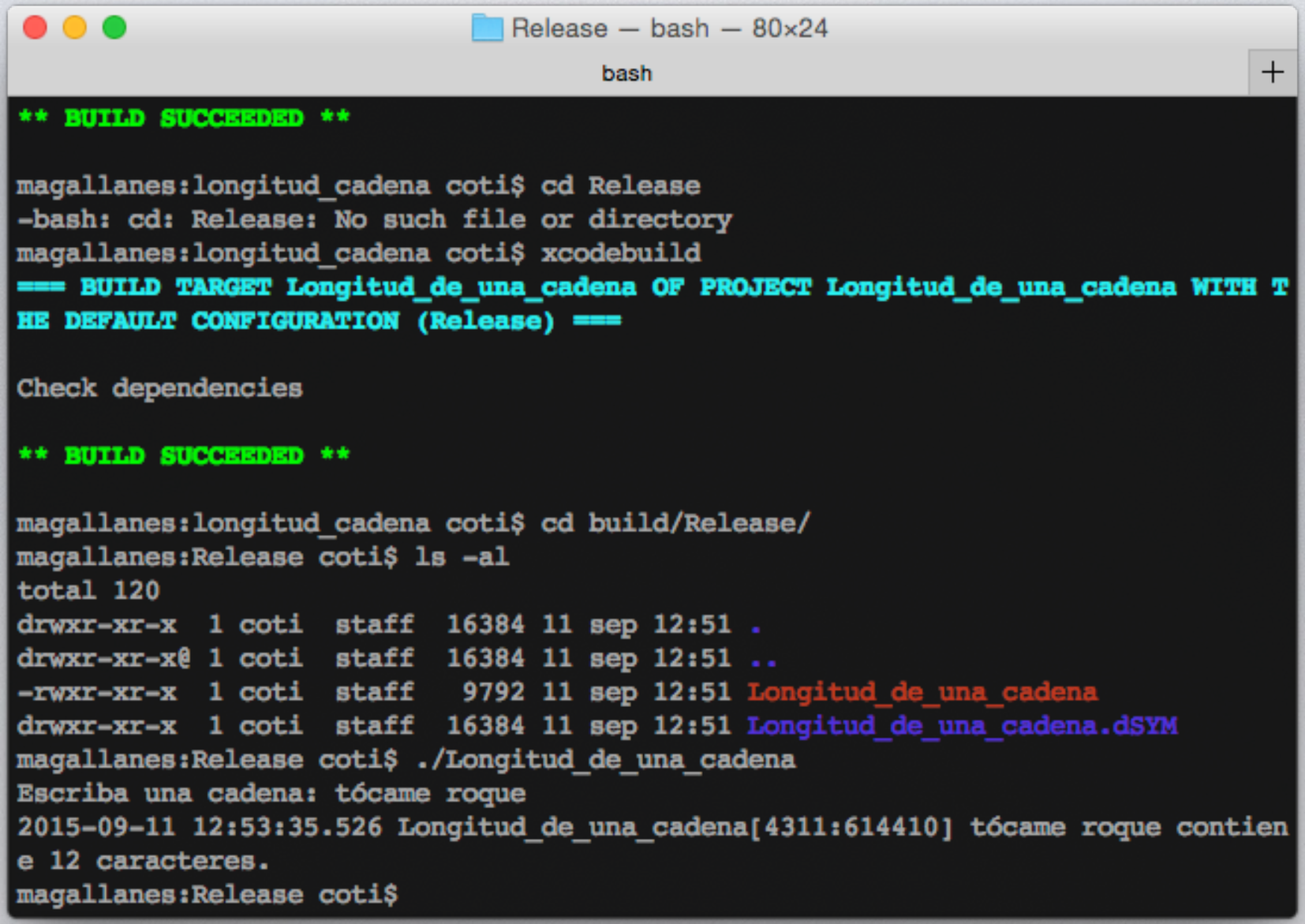
VERSION DOS


longitud_cadena — bash — 80×24
bash
magallanes:~ coti$ cd /Volumes/bruegel/Documents/DEPTO/desaplava_2016/_traspas_2016/_cocoa/0100_objc/0100_programas/longitud_cadena
magallanes:longitud_cadena coti$ ls -al
total 144
drwxr-xr-x 1 coti staff 16384 10 abr 2013 .
drwxr-xr-x 1 coti staff 16384 9 oct 2013 ..
-rw-r--r--@ 1 coti staff   6148 10 abr 2013 .DS_Store
drwxr-xr-x@ 1 coti staff 16384 11 sep 12:38 Longitud_de_una_cadena.xcodeproj
drwxr-xr-x 1 coti staff 16384 11 sep 12:49 Test
magallanes:longitud_cadena coti$ xcodebuild
=== BUILD TARGET Longitud_de_una_cadena OF PROJECT Longitud_de_una_cadena WITH THE DEFAULT CONFIGURATION (Release) ===
Check dependencies
Write auxiliary files
/bin/mkdir -p /Volumes/bruegel/Documents/DEPTO/desaplava_2016/_traspas_2016/_cocoa/0100_objc/0100_programas/longitud_cadena/build/Longitud_de_una_cadena.build/Release/Longitud_de_una_cadena.build
write-file /Volumes/bruegel/Documents/DEPTO/desaplava_2016/_traspas_2016/_cocoa/0100_objc/0100_programas/longitud_cadena/build/Longitud_de_una_cadena.build/Release/Longitud_de_una_cadena.build/Longitud_de_una_cadena-generated-files.hmap
write-file /Volumes/bruegel/Documents/DEPTO/desaplava_2016/_traspas_2016/_cocoa/0100_objc/0100_programas/longitud_cadena/build/Longitud_de_una_cadena.build/Rele

- Cuando acaba la compilación, si no hay errores, se llega a esta pantalla


longitud_cadena — bash — 80×24
bash
/0100_programas/longitud_cadena/build/Longitud_de_una_cadena.build/Release/Longitud_de_una_cadena.build/Objects-normal/x86_64/Longitud_de_una_cadena.LinkFileList -mmacosx-version-min=10.10 -fobjc-arc -fobjc-link-runtime -framework Foundation -Xlinker -dependency_info -Xlinker /Volumes/bruegel/Documents/DEPTO/desaplava_2016/_traspas_2016/_cocoa/0100_objc/0100_programas/longitud_cadena/build/Longitud_de_una_cadena.build/Release/Longitud_de_una_cadena.build/Objects-normal/x86_64/Longitud_de_una_cadena_dependency_info.dat -o /Volumes/bruegel/Documents/DEPTO/desaplava_2016/_traspas_2016/_cocoa/0100_objc/0100_programas/longitud_cadena/build/Release/Longitud_de_una_cadena
GenerateDSYMFile build/Release/Longitud_de_una_cadena.dSYM build/Release/Longitud_de_una_cadena
cd /Volumes/bruegel/Documents/DEPTO/desaplava_2016/_traspas_2016/_cocoa/0100_objc/0100_programas/longitud_cadena
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/dsymutil /Volumes/bruegel/Documents/DEPTO/desaplava_2016/_traspas_2016/_cocoa/0100_objc/0100_programas/longitud_cadena/build/Release/Longitud_de_una_cadena -o /Volumes/bruegel/Documents/DEPTO/desaplava_2016/_traspas_2016/_cocoa/0100_objc/0100_programas/longitud_cadena/build/Release/Longitud_de_una_cadena.dSYM
** BUILD SUCCEEDED **
magallanes:longitud_cadena coti$

- Ahora se accede al directorio en q se ha creado la herramienta (build/Release) y se ejecuta, con idéntico resultado al obtenido anteriormente


Release — bash — 80×24
bash
** BUILD SUCCEEDED **
magallanes:longitud_cadena coti$ cd Release
-bash: cd: Release: No such file or directory
magallanes:longitud_cadena coti$ xcodebuild
=== BUILD TARGET Longitud_de_una_cadena OF PROJECT Longitud_de_una_cadena WITH THE DEFAULT CONFIGURATION (Release) ===
Check dependencies
** BUILD SUCCEEDED **
magallanes:longitud_cadena coti$ cd build/Release/
magallanes:Release coti$ ls -al
total 120
drwxr-xr-x 1 coti staff 16384 11 sep 12:51 .
drwxr-xr-x@ 1 coti staff 16384 11 sep 12:51 ..
-rwxr-xr-x 1 coti staff   9792 11 sep 12:51 Longitud_de_una_cadena
drwxr-xr-x 1 coti staff 16384 11 sep 12:51 Longitud_de_una_cadena.dSYM
magallanes:Release coti$ ./Longitud_de_una_cadena
Escriba una cadena: tócame roque
2015-09-11 12:53:35.526 Longitud_de_una_cadena[4311:614410] tócame roque contiene 12 caracteres.
magallanes:Release coti$

- Pregunta

- ¿Cuantos bytes ocupa a.out?

- ¿Cuantos bytes ocupa Longitud_de_una_cadena?

OTRO EJEMPLO

Crear un ejemplar de **`NSMutableArray`** y añadir al ejemplar una cadena y un número de coma flotante (**`NSNumber`**). A continuación, mostrar en pantalla el valor de todos los objetos almacenados en la colección.

Proyecto: **`una_coleccion`**

USO DE MÉTODOS

```
// Proyecto una_coleccion
    #import “Foundation/Foundation.h”
    int main(int argc, const char * argv[])
    {

        @autoreleasepool {

            NSMutableArray * ma = [[NSMutableArray alloc] init ];
            NSString * cadena = @"María tiene una oveja";
            NSNumber * numero = [NSNumber numberWithFloat:3.14];

            [ma addObject:cadena];
            [ma addObject:numero];

            for (NSObject * objeto in ma) {
                NSLog(@"%@",objeto);
            }

        }
        return 0;
}
```

Este programa completo muestra el uso de un bucle forin y una coleccion (`NSMutableArray`).

Demo

Proyecto: `una_coleccion`

`NSLOG`: ESPECIFICADORES DE FORMATO

`%@`	Válido para cualquier tipo no primitivo, llama a `description`
`%d, %i`	enteros con signo
`%u`	enteros sin signo
`%f`	`float/double`
`%x, %X`	enteros en hexadecimal
`%o`	enteros en octal
`%zu`	`size_t`
`%p`	valor de un puntero
`%e`	`float/double` en notación científica
`%g`	`float/double` (equivale a `%f` o a `%e`, según el valor)
`%s`	Cadenas de C con formato de 1 byte por char
`%S`	Cadenas de C con formato `unichar`
`%.*s`	Cadenas de Pascal (requiere 2 argumentos)
`%c`	caracteres
`%C`	caracteres de formato unichar
`%lld`	`long long`
`%llu`	`unsigned long long`
`%Lf`	`long double`

EJEMPLO

Construir un programa que muestre el uso de algunos los especificadores de formato que admite NSLog(). El programa deberá crear las clases y variables necesarias para verificar el funcionamiento de los especificadores

Proyecto: **`formato_nslog`**

EJEMPLO

```
        // Proyecto: formato_nslog
#import <Foundation/Foundation.h>
#import "RGBTestClass.h"

int main(int argc, const char * argv[])
{

    @autoreleasepool {
        RGBTestClass * testClass = [RGBTestClass new];
        NSLog(@"Using \"%%@\" %@", testClass);
        int numberOfPages = 127;
        NSLog(@"Using %d, that is %i", numberOfPages, numberOfPages);
        float height = 1.68f;
        double weight = 123478901234567.56;
        NSLog(@"Using %%f: the float is %f and the double is %f", height, weight);
        NSLog(@"%d in hex is %x or %X", numberOfPages, numberOfPages, numberOfPages);
        NSLog(@"%d in octal is %o", numberOfPages, numberOfPages);
        NSLog(@"The pointer testClass is %p", testClass);
        NSLog(@"sizeof(height) = %zu bytes", sizeof(height)); // Sizeof does not take classes
        NSLog(@"Using %%e: the float is %e and the double is %e", height, weight);
        NSLog(@"Using %%g: the float is %g and the double is %g", height, weight);
        // Terminal (in Xcode at least) expects NSMacOSRomanStringEncoding.
        const char * cadenaDeChar = [@"María tiene una oveja" cStringUsingEncoding:NSMacOSRomanStringEncoding];
        NSLog(@"Using %%s, cadenaDechar es %s", cadenaDeChar);
        char letras[] = "abcde";
        NSLog(@"Using %%c, letras[1] = %c", letras[1]);
        long long veryLong = -1234123412341234123;
        NSLog(@"Using %%lld, Rather long long number:       %lld", veryLong);
        unsigned long long unsignedVeryLong = 8234123412341234123;
        NSLog(@"Using %%llu, Even longer unsigned long long  %llu", unsignedVeryLong);
        long double big = 33333333333333333.333333333333333333;
        NSLog(@"Using %%Lf, rather big floating-point number   %Lf", big);
    }
    return 0;
}
```

Demo

Proyecto: `formato_nslog`

UNICHAR

El tipo `unichar` es un `char`, pero con 2 bytes de longitud en formato UTF-16. Sirve para codificar todos aquellos símbolos que no pertenecen al código ASCII, y también para escribir en idiomas que no se pueden expresar mediante `char`s de 1 solo byte.

Es posible insertar caracteres Unicode en un ejemplar de `NSString`, véase el ejercicio siguiente.

EJEMPLO

Construir un programa que genere 50000 caracteres UNICHAR, añadiéndolos a una cadena y mostrándolos después en pantalla.

Proyecto: **`muchos_unichar`**

EJEMPLO UNICHAR

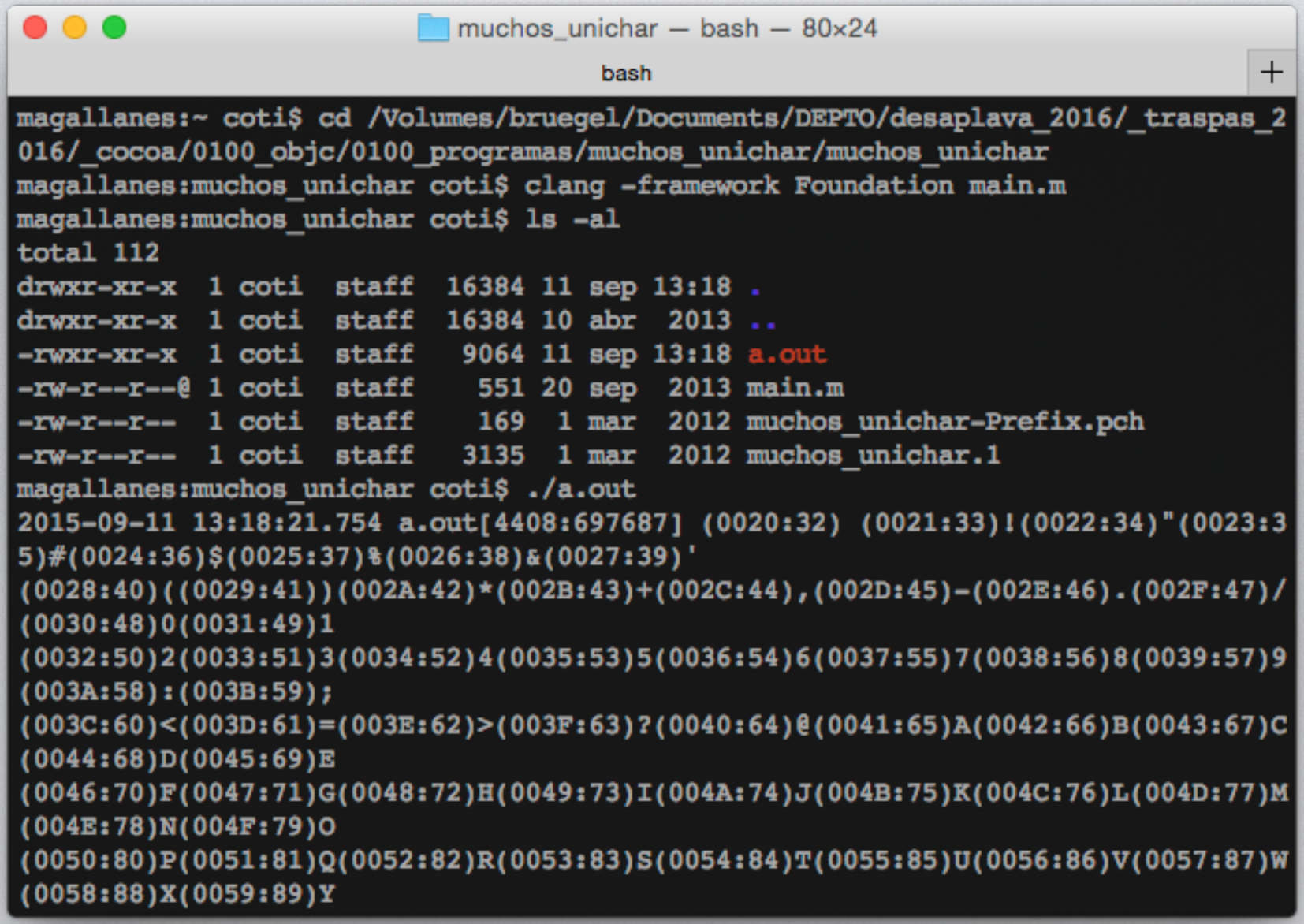
```
//  Proyecto muchos_unichar. clang -framework Foundation main.m
#import <Foundation/Foundation.h>

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        int i;
        NSMutableString *string = [NSMutableString string];
        printf("Un listado de unichar (hex:dec) caracter\n\n");
        for (i = 32; i < 55296; i++) {
            if (!(i % 10)) [string appendString:@"\n"];
            [string appendFormat:@"(%04X:%i)%C\n", i, i, i];
        }
        NSLog(@"\n\n%@",string);

    }
    return 0;
}
```

Demo

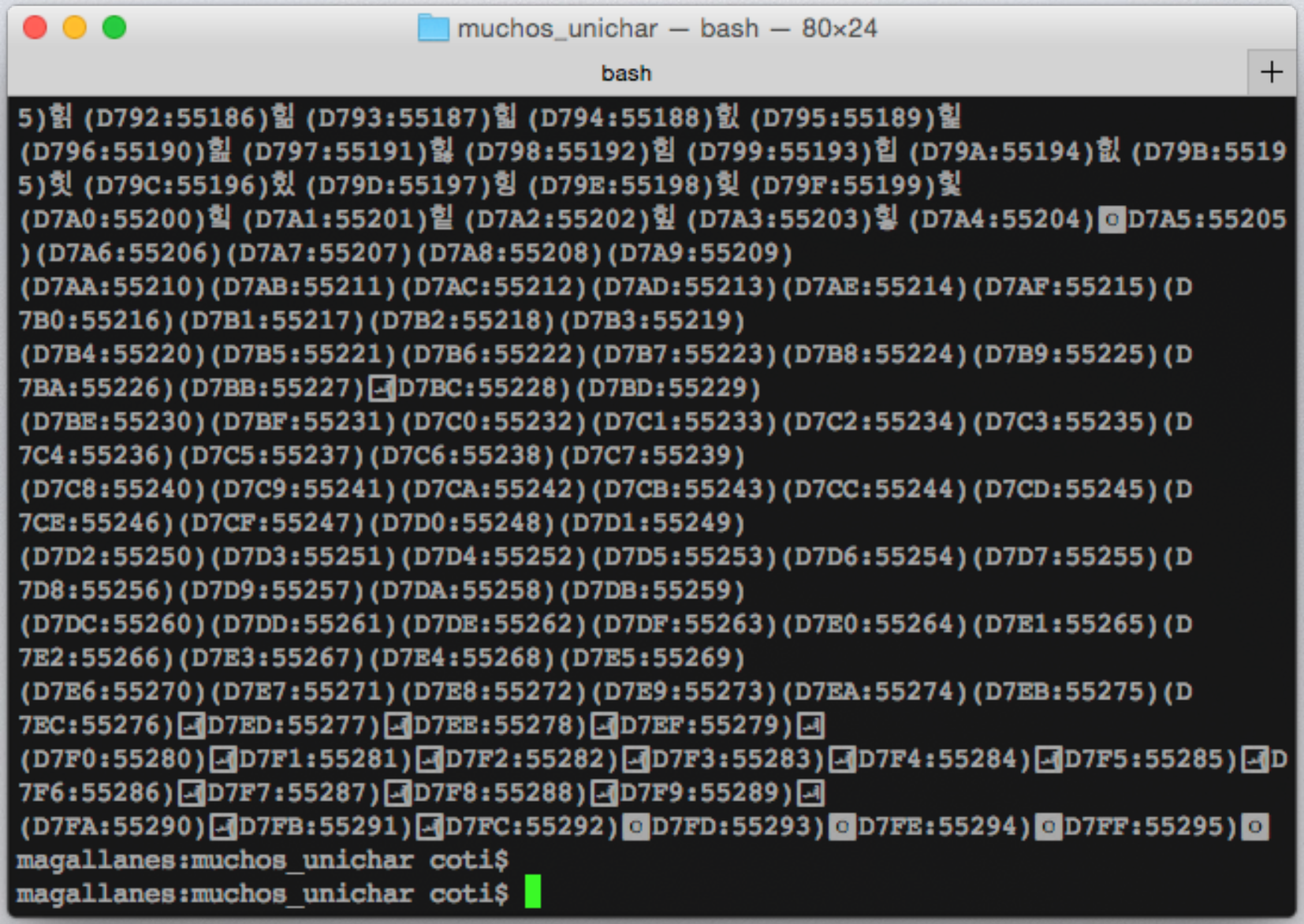
Proyecto: `muchos_unichar`

DESDE LA LÍNEA DE ÓRDENES

VERSION UNO


muchos_unichar — bash — 80×24
bash
magallanes:~ coti$ cd /Volumes/bruegel/Documents/DEPTO/desaplava_2016/_traspas_2016/_cocoa/0100_objc/0100_programas/muchos_unichar/muchos_unichar
magallanes:muchos_unichar coti$ clang -framework Foundation main.m
magallanes:muchos_unichar coti$ ls -al
total 112
drwxr-xr-x 1 coti staff 16384 11 sep 13:18 .
drwxr-xr-x 1 coti staff 16384 10 abr 2013 ..
-rwxr-xr-x 1 coti staff   9064 11 sep 13:18 a.out
-rw-r--r--@ 1 coti staff   551 20 sep 2013 main.m
-rw-r--r-- 1 coti staff   169 1 mar 2012 muchos_unichar-Prefix.pch
-rw-r--r-- 1 coti staff   3135 1 mar 2012 muchos_unichar.1
magallanes:muchos_unichar coti$ ./a.out
2015-09-11 13:18:21.754 a.out[4408:697687] (0020:32) (0021:33)!(0022:34)"(0023:35)#(0024:36)$(0025:37)%(0026:38)&(0027:39)'
(0028:40)((0029:41))(002A:42)*(002B:43)+(002C:44),(002D:45)-(002E:46).(002F:47)/(0030:48)0(0031:49)1
(0032:50)2(0033:51)3(0034:52)4(0035:53)5(0036:54)6(0037:55)7(0038:56)8(0039:57)9(003A:58):(003B:59);
(003C:60)<(003D:61)=(003E:62)>(003F:63)?(0040:64)@(0041:65)A(0042:66)B(0043:67)C(0044:68)D(0045:69)E
(0046:70)F(0047:71)G(0048:72)H(0049:73)I(004A:74)J(004B:75)K(004C:76)L(004D:77)M(004E:78)N(004F:79)O
(0050:80)P(0051:81)Q(0052:82)R(0053:83)S(0054:84)T(0055:85)U(0056:86)V(0057:87)W(0058:88)X(0059:89)Y

- y después de muchas pantallas acabamos aquí


muchos_unichar — bash — 80×24
bash
5)힑 (D792:55186)힒 (D793:55187)힓 (D794:55188)힔 (D795:55189)힕
(D796:55190)힖 (D797:55191)힗 (D798:55192)힘 (D799:55193)힙 (D79A:55194)힚 (D79B:5519
5)힛 (D79C:55196)힜 (D79D:55197)힝 (D79E:55198)힞 (D79F:55199)힟
(D7A0:55200)힠 (D7A1:55201)힡 (D7A2:55202)힢 (D7A3:55203)힣 (D7A4:55204)D7A5:55205
)(D7A6:55206)(D7A7:55207)(D7A8:55208)(D7A9:55209)
(D7AA:55210)(D7AB:55211)(D7AC:55212)(D7AD:55213)(D7AE:55214)(D7AF:55215)(D
7B0:55216)(D7B1:55217)(D7B2:55218)(D7B3:55219)
(D7B4:55220)(D7B5:55221)(D7B6:55222)(D7B7:55223)(D7B8:55224)(D7B9:55225)(D
7BA:55226)(D7BB:55227)D7BC:55228)(D7BD:55229)
(D7BE:55230)(D7BF:55231)(D7C0:55232)(D7C1:55233)(D7C2:55234)(D7C3:55235)(D
7C4:55236)(D7C5:55237)(D7C6:55238)(D7C7:55239)
(D7C8:55240)(D7C9:55241)(D7CA:55242)(D7CB:55243)(D7CC:55244)(D7CD:55245)(D
7CE:55246)(D7CF:55247)(D7D0:55248)(D7D1:55249)
(D7D2:55250)(D7D3:55251)(D7D4:55252)(D7D5:55253)(D7D6:55254)(D7D7:55255)(D
7D8:55256)(D7D9:55257)(D7DA:55258)(D7DB:55259)
(D7DC:55260)(D7DD:55261)(D7DE:55262)(D7DF:55263)(D7E0:55264)(D7E1:55265)(D
7E2:55266)(D7E3:55267)(D7E4:55268)(D7E5:55269)
(D7E6:55270)(D7E7:55271)(D7E8:55272)(D7E9:55273)(D7EA:55274)(D7EB:55275)(D
7EC:55276)D7ED:55277)D7EE:55278)D7EF:55279)
(D7F0:55280)D7F1:55281)D7F2:55282)D7F3:55283)D7F4:55284)D7F5:55285)D
7F6:55286)D7F7:55287)D7F8:55288)D7F9:55289)
(D7FA:55290)D7FB:55291)D7FC:55292)D7FD:55293)D7FE:55294)D7FF:55295)
magallanes:muchos_unichar coti$
magallanes:muchos_unichar coti$

VERSION DOS

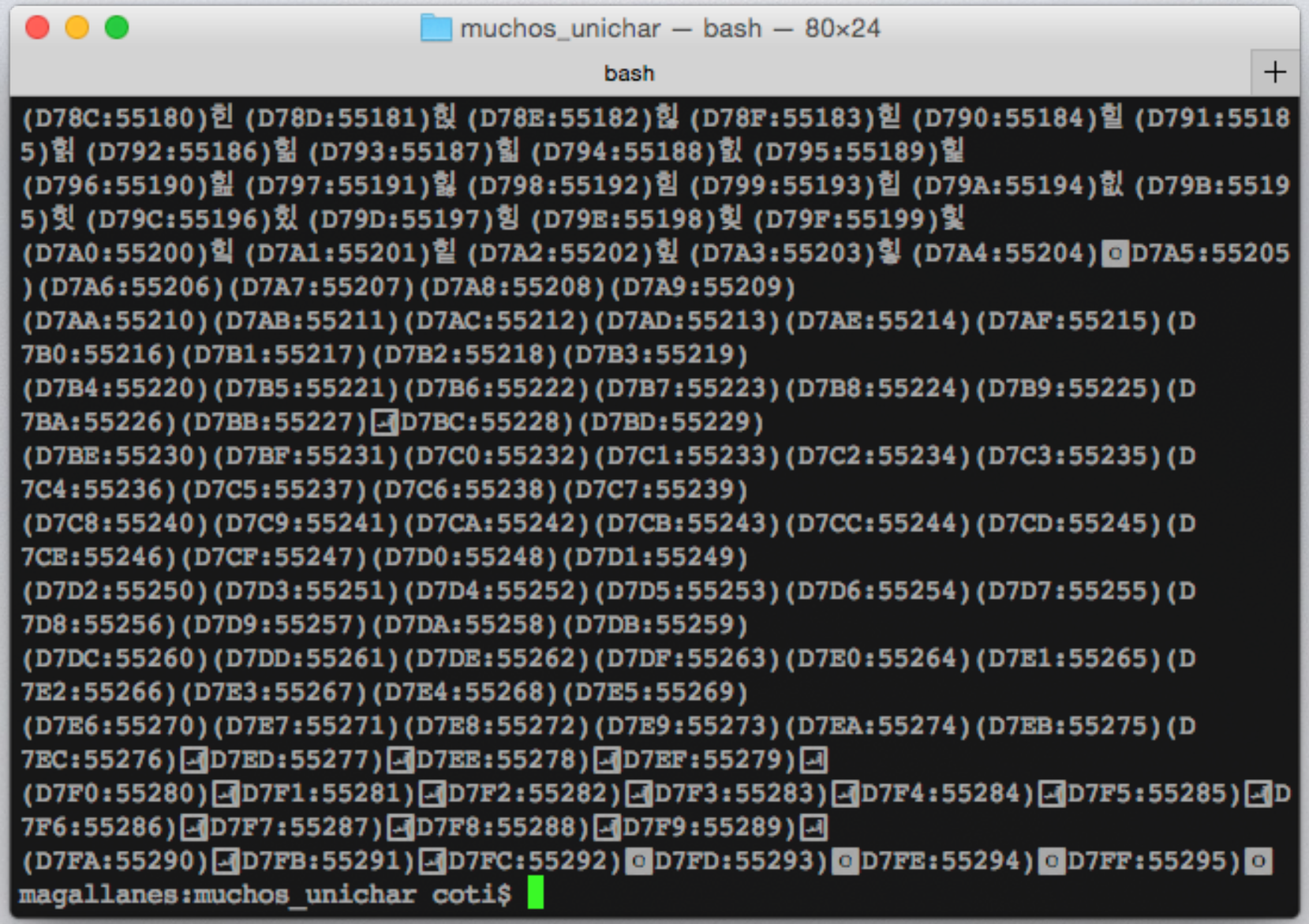
Después de compilar desde Xcode

muchos_unichar — bash — 80×24

bash

```
magallanes:muchos_unichar coti$ ls -al
total 3952
drwxr-xr-x  1 coti  staff    16384 11 sep 13:22 .
drwxr-xr-x  1 coti  staff    16384  9 oct  2013 ..
-rw-r--r--@ 1 coti  staff     6148 10 abr  2013 .DS_Store
drwxr-xr-x@ 1 coti  staff    16384 11 sep 13:22 build
drwxr-xr-x  1 coti  staff    16384 11 sep 13:18 muchos_unichar
drwxr-xr-x@ 1 coti  staff    16384 20 sep  2013 muchos_unichar.xcodeproj
-rw-r--r--@ 1 coti  staff  1931379 25 sep  2012 resultado.rtf
magallanes:muchos_unichar coti$ xcodebuild
=== BUILD TARGET muchos_unichar OF PROJECT muchos_unichar WITH THE DEFAULT CONFI
GURATION (Release) ===

Check dependencies

** BUILD SUCCEEDED **

magallanes:muchos_unichar coti$ build/Release/muchos_unichar
```

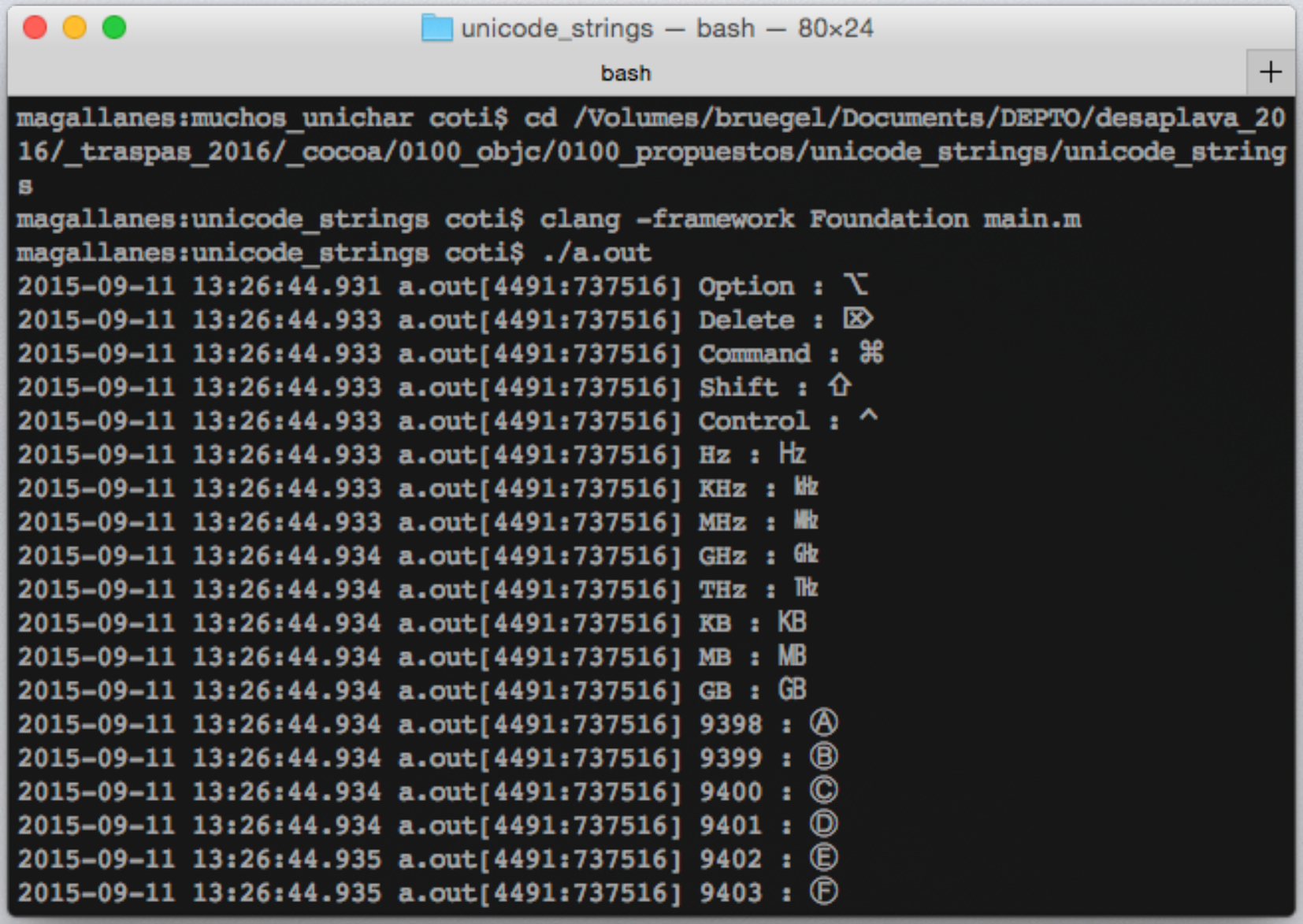
- Después de muchas pantallas. El archivo de resultados ocupa 1.9 MB!

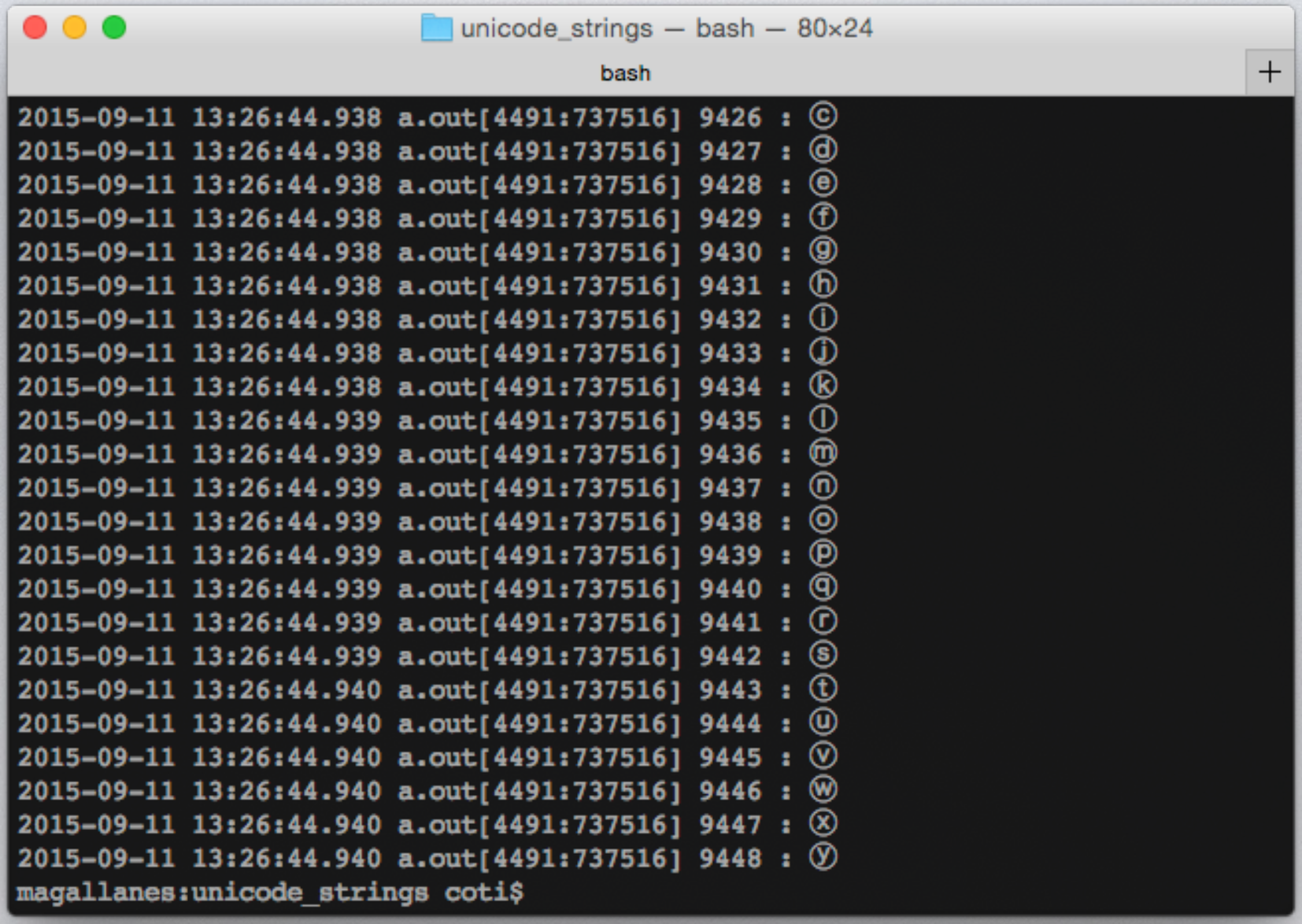

muchos_unichar — bash — 80×24
bash
(D78C:55180)힌 (D78D:55181)힍 (D78E:55182)힎 (D78F:55183)힏 (D790:55184)힐 (D791:55185)힑 (D792:55186)힒 (D793:55187)힓 (D794:55188)힔 (D795:55189)힕
(D796:55190)힖 (D797:55191)힗 (D798:55192)힘 (D799:55193)힙 (D79A:55194)힚 (D79B:55195)힛 (D79C:55196)힜 (D79D:55197)힝 (D79E:55198)힞 (D79F:55199)힟
(D7A0:55200)힠 (D7A1:55201)힡 (D7A2:55202)힢 (D7A3:55203)힣 (D7A4:55204)ㅇD7A5:55205)(D7A6:55206)(D7A7:55207)(D7A8:55208)(D7A9:55209)
(D7AA:55210)(D7AB:55211)(D7AC:55212)(D7AD:55213)(D7AE:55214)(D7AF:55215)(D7B0:55216)(D7B1:55217)(D7B2:55218)(D7B3:55219)
(D7B4:55220)(D7B5:55221)(D7B6:55222)(D7B7:55223)(D7B8:55224)(D7B9:55225)(D7BA:55226)(D7BB:55227)D7BC:55228)(D7BD:55229)
(D7BE:55230)(D7BF:55231)(D7C0:55232)(D7C1:55233)(D7C2:55234)(D7C3:55235)(D7C4:55236)(D7C5:55237)(D7C6:55238)(D7C7:55239)
(D7C8:55240)(D7C9:55241)(D7CA:55242)(D7CB:55243)(D7CC:55244)(D7CD:55245)(D7CE:55246)(D7CF:55247)(D7D0:55248)(D7D1:55249)
(D7D2:55250)(D7D3:55251)(D7D4:55252)(D7D5:55253)(D7D6:55254)(D7D7:55255)(D7D8:55256)(D7D9:55257)(D7DA:55258)(D7DB:55259)
(D7DC:55260)(D7DD:55261)(D7DE:55262)(D7DF:55263)(D7E0:55264)(D7E1:55265)(D7E2:55266)(D7E3:55267)(D7E4:55268)(D7E5:55269)
(D7E6:55270)(D7E7:55271)(D7E8:55272)(D7E9:55273)(D7EA:55274)(D7EB:55275)(D7EC:55276)D7ED:55277)D7EE:55278)D7EF:55279)
(D7F0:55280)D7F1:55281)D7F2:55282)D7F3:55283)D7F4:55284)D7F5:55285)D7F6:55286)D7F7:55287)D7F8:55288)D7F9:55289)
(D7FA:55290)D7FB:55291)D7FC:55292)ㅇD7FD:55293)ㅇD7FE:55294)ㅇD7FF:55295)ㅇ
magallanes:muchos_unichar coti$

PROPUESTO

Crear un programa que, haciendo uso de caracteres unichar, muestre en pantalla distintos signos del teclado que no corresponden a letras. Por ejemplo, mostrar en pantalla los signos de Opción, Borrar, Comando,Mayúsculas y Control. Adicionalmente, mostrar los signos de Hz, KHz, MHz, GHz, THz, KB, MB, GB. Los signos deben aparecer en la terminal (bien sea de Xcode o en Terminal). Crear un guión para compilar el programa. Se puede compilar un proyecto de Xcode desde la terminal?

Proyecto: **`unicode_strings`**


unicode_strings — bash — 80×24
bash
magallanes:muchos_unichar coti$ cd /Volumes/bruegel/Documents/DEPTO/desaplava_2016/_traspas_2016/_cocoa/0100_objc/0100_propuestos/unicode_strings/unicode_strings
magallanes:unicode_strings coti$ clang -framework Foundation main.m
magallanes:unicode_strings coti$ ./a.out
2015-09-11 13:26:44.931 a.out[4491:737516] Option : ⌥
2015-09-11 13:26:44.933 a.out[4491:737516] Delete : ⌦
2015-09-11 13:26:44.933 a.out[4491:737516] Command : ⌘
2015-09-11 13:26:44.933 a.out[4491:737516] Shift : ⇧
2015-09-11 13:26:44.933 a.out[4491:737516] Control : ^
2015-09-11 13:26:44.933 a.out[4491:737516] Hz : ㎐
2015-09-11 13:26:44.933 a.out[4491:737516] KHz : ㎑
2015-09-11 13:26:44.933 a.out[4491:737516] MHz : ㎒
2015-09-11 13:26:44.934 a.out[4491:737516] GHz : ㎓
2015-09-11 13:26:44.934 a.out[4491:737516] THz : ㎔
2015-09-11 13:26:44.934 a.out[4491:737516] KB : KB
2015-09-11 13:26:44.934 a.out[4491:737516] MB : MB
2015-09-11 13:26:44.934 a.out[4491:737516] GB : GB
2015-09-11 13:26:44.934 a.out[4491:737516] 9398 : Ⓐ
2015-09-11 13:26:44.934 a.out[4491:737516] 9399 : Ⓑ
2015-09-11 13:26:44.934 a.out[4491:737516] 9400 : Ⓒ
2015-09-11 13:26:44.934 a.out[4491:737516] 9401 : Ⓓ
2015-09-11 13:26:44.935 a.out[4491:737516] 9402 : Ⓔ
2015-09-11 13:26:44.935 a.out[4491:737516] 9403 : Ⓕ


unicode_strings — bash — 80×24
bash
2015-09-11 13:26:44.938 a.out[4491:737516] 9426 : ⓒ
2015-09-11 13:26:44.938 a.out[4491:737516] 9427 : ⓓ
2015-09-11 13:26:44.938 a.out[4491:737516] 9428 : ⓔ
2015-09-11 13:26:44.938 a.out[4491:737516] 9429 : ⓕ
2015-09-11 13:26:44.938 a.out[4491:737516] 9430 : ⓖ
2015-09-11 13:26:44.938 a.out[4491:737516] 9431 : ⓗ
2015-09-11 13:26:44.938 a.out[4491:737516] 9432 : ⓘ
2015-09-11 13:26:44.938 a.out[4491:737516] 9433 : ⓙ
2015-09-11 13:26:44.938 a.out[4491:737516] 9434 : ⓚ
2015-09-11 13:26:44.939 a.out[4491:737516] 9435 : ⓛ
2015-09-11 13:26:44.939 a.out[4491:737516] 9436 : ⓜ
2015-09-11 13:26:44.939 a.out[4491:737516] 9437 : ⓝ
2015-09-11 13:26:44.939 a.out[4491:737516] 9438 : ⓞ
2015-09-11 13:26:44.939 a.out[4491:737516] 9439 : ⓟ
2015-09-11 13:26:44.939 a.out[4491:737516] 9440 : ⓠ
2015-09-11 13:26:44.939 a.out[4491:737516] 9441 : ⓡ
2015-09-11 13:26:44.939 a.out[4491:737516] 9442 : ⓢ
2015-09-11 13:26:44.940 a.out[4491:737516] 9443 : ⓣ
2015-09-11 13:26:44.940 a.out[4491:737516] 9444 : ⓤ
2015-09-11 13:26:44.940 a.out[4491:737516] 9445 : ⓥ
2015-09-11 13:26:44.940 a.out[4491:737516] 9446 : ⓦ
2015-09-11 13:26:44.940 a.out[4491:737516] 9447 : ⓧ
2015-09-11 13:26:44.940 a.out[4491:737516] 9448 : ⓨ
magallanes:unicode_strings coti$

CSTRING Y NSSTRING

Las cadenas de C se pueden convertir en `NSString`, y viceversa. Téngase en cuenta que `NSString` puede contener cadenas codificadas en Unicode, y la traducción entre cadenas de C y `unichar` no es trivial. Lo normal en Cocoa emplear `NSString`, puesto que todos los métodos están diseñados para emplear este tipo de datos.

EJEMPLO

Construir un programa que genere un ejemplar de NSString y lo transforme en cadena de caracteres, y viceversa. El programa debe mostrar tanto la cadena de caracteres como el NSString en pantalla. Hay que asegurarse de que los acentos y signos diacríticos se codifiquen y muestren correctamente (código fuente en UTF-8, NSUTF8StringEncoding).

CSTRING Y NSSTRING

CONVERSIÓN CSTRING-NSSTRING

```
// proyecto nsstring__cstring
#import <Foundation/Foundation.h>

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        const char * cadena_c = "María tiene una oveja";
        NSString * nsstring = [NSString stringWithCString:cadena_c
                                                  encoding:NSUTF8StringEncoding];
        NSString * otra_nsstring = @"como la nieve de blánca";
        const char * otra_cadena_c = [otra_nsstring
                                      cStringUsingEncoding:NSUTF8StringEncoding];

        printf("Versos 1\n\n");
        NSLog(@"%@",nsstring);
        NSLog(@"%@\n",otra_nsstring);

        printf("Versos 2\n\n");
        printf("%s\n", cadena_c);
        printf("%s\n", otra_cadena_c);

    }
    return 0;
}
```

Este proyecto muestra la conversión bidireccional entre cadenas de C y NSString.
Obsérvese la conveniencia de codificar en UTF8, que garantiza la traducción.

Proyecto: `nsstring__cstring`

PENSAMIENTOS

Un mensaje enviado a `nil` no hace nada, y no produce un error.

```
UnObjeto * puntero = nil;
int numero = [puntero count];
// numero recibe el valor 0,  o nil si fuese un puntero.
```

CLASES IMPORTANTES EN OBJECTIVE-C

CLASES IMPORTANTES

Un enlace interesante:

https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/ObjC_classic/Intro/IntroFoundation.html

CLASES IMPORTANTES

El *framework* `Foundation` está formado por una jerarquía de clases relacionada por herencia. En su totalidad, son más de 130 clases! Vamos a estudiar brevemente cuatro clases importantes: `NSObject`, `NSArray`, `NSMutableArray` y `NSString`.

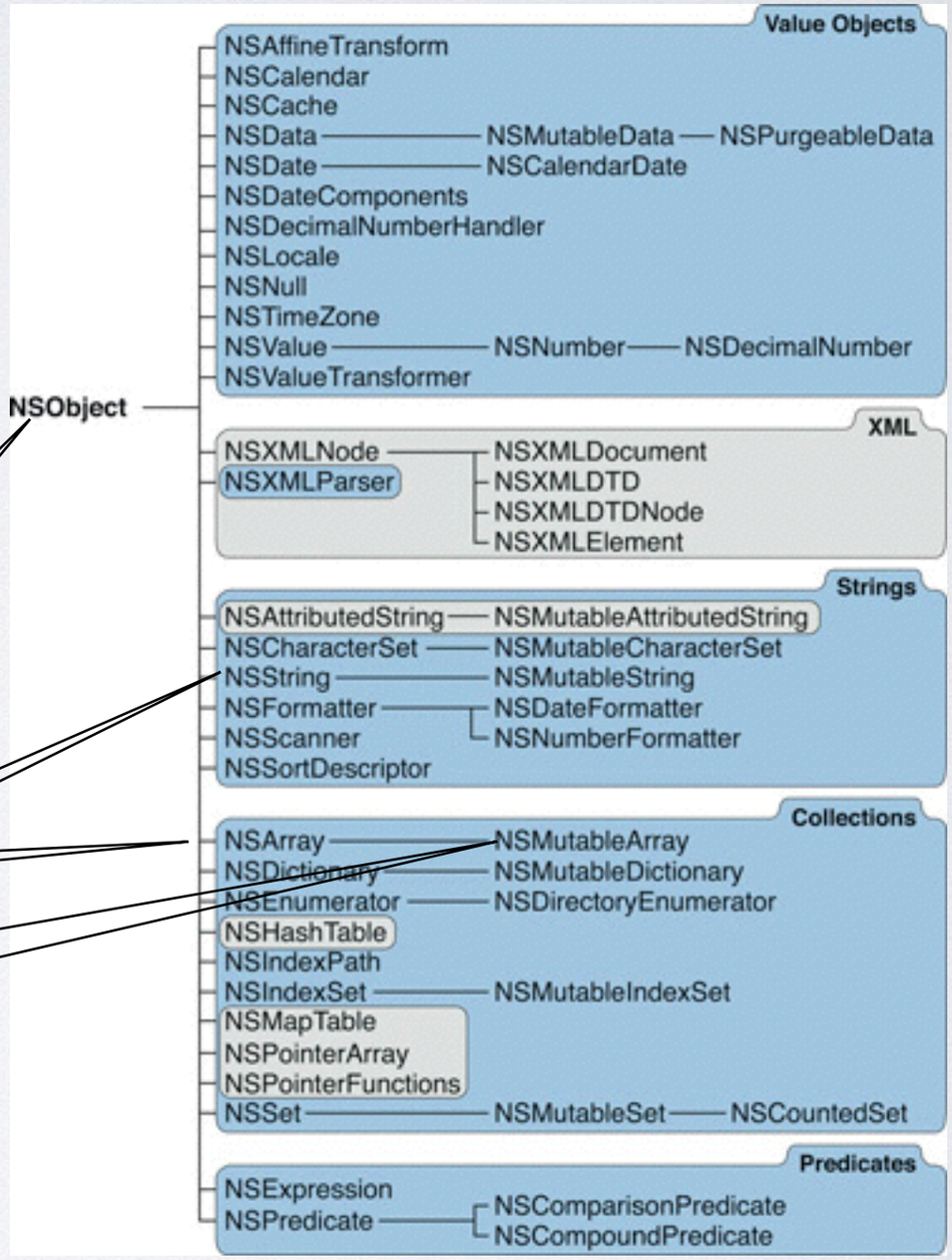
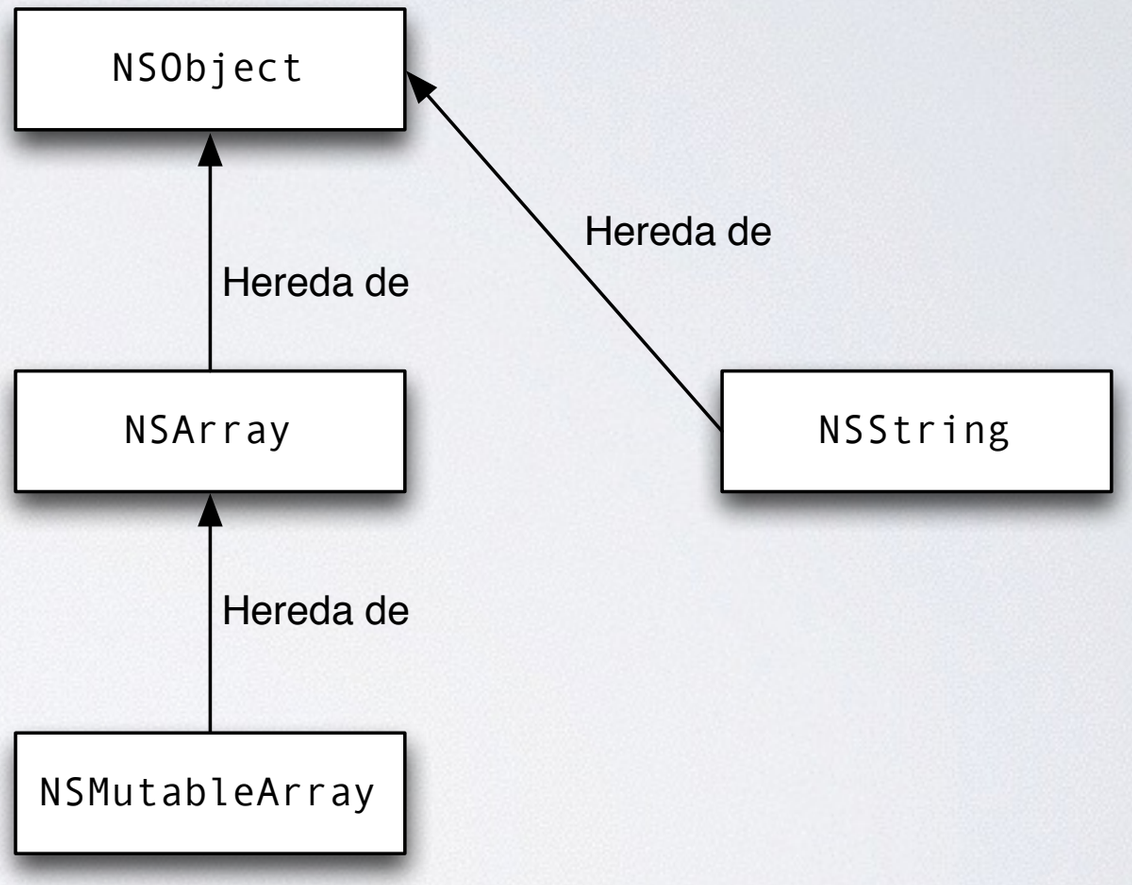


Diagrama PARCIAL de clases de Foundation

CLASES IMPORTANTES

De forma resumida, la jerarquía de herencia que existe entre estas cuatro clases puede verse en el diagrama. `NSObject` es la raíz del árbol de herencia, para todas las clases de Cocoa.



NSOBJECT

NSOBJECT

Es la raíz del árbol de jerarquía de Cocoa. Cabe destacar los métodos siguientes:

`-(id)init`

Este método da valores iniciales a todos los campos del objeto una vez que se ha reservado memoria para éste mediante `alloc`. Normalmente `init` se ejecuta junto con `alloc`:

```
UnaClase * unObjeto = [[UnaClase alloc] init];
```

NSOBJECT - MÉTODOS

`-(NSString*)description`

Proporciona una cadena que muestra el estado del objeto. Este método se invoca de forma automática en `NSLog()`. Las dos últimas líneas del siguiente fragmento de código, por tanto, producen el mismo resultado

```
NSNumber * n = [NSNumber numberWithInt:7];
NSLog(@"%@", n);
NSLog(@"%@", [n description]);
```

Las clases propias deben redefinir `description`. Normalmente, todas las clases de Cocoa poseen su propia versión; de no ser así, se ejecuta la de `NSObject`.

NSOBJECT - MÉTODOS

`-(BOOL)isEqual:(id)otroObjeto`

Proporciona `YES` si el objeto que recibe el mensaje es de la misma clase que el objeto que se proporciona como argumento.

Muchas clases redefinen este método para darle el significado que espera el usuario. Por ejemplo, `NSString` proporciona `YES` si todos los caracteres de las dos cadenas son iguales.

EJEMPLO

Crear un programa que compare dos cadenas de igual valor, y dos cadenas de caracteres de igual valor. Estudiar el resultado de la comparación efectuada con el operador `==` y con el método **`isEqual:`**.

proyecto: **`iguales_o_no`**

IGUALES_O_NO

```
// proyecto: iguales_o_no
#import <Foundation/Foundation.h>

int main(int argc, const char * argv[])
{

  @autoreleasepool {
    NSString * str1 = @"pepe";
    NSString * str2 = @"juan";
    const char * cad1 = "pepe";
    const char * cad2 = "pepe";

    NSLog(@"Según ==, %@", (str1 == str2) ? @"str1 == str2" : @"str1 != str2");

    NSLog(@"Según ==, %@", ((char*)cad1 == (char*)cad2) ? @"cad1 == cad2" : @"cad1 != cad2");

    NSLog(@"[str1 equals:str2 = %@", [str1 isEqual:str2] ? @"TRUE" : @"FALSE");

  }
  return 0;
}
```

Demo

Proyecto: `iguales_o_no`

NSARRAY

NSARRAY

Es una lista de punteros de objetos de cualquier tipo (pero no admite valores de tipos primitivos). La lista se recorre mediante índices que van desde `0` hasta `N-1`, siendo `N` el número total de objetos que estén almacenados en ese ejemplar de `NSArray`. No se puede almacenar el valor `nil`.

Los ejemplares de `NSArray` son inmutables, esto es, una vez creados su contenido es constante (no se puede añadir, eliminar ni modificar elementos).

Existe una subclase mutable de `NSArray`, denominada `NSMutableArray`.

NSARRAY - MÉTODOS

`-(unsigned)count`

Proporciona el número de elementos que contiene ese ejemplar de `NSArray`.

`-(id)objectAtIndex:(unsigned)i`

Proporciona el objeto situado en la i-ésima posición. Hay comprobación de alcances; si se sobrepasa `[miArray count] -1`, se producirá un error de ejecución.

NSARRAY - MÉTODOS

`-(id)lastObject`

Proporciona el último elemento de la lista, o `nil` si está vacía. de elementos que contiene ese ejemplar de `NSArray`.

`-(BOOL)containsObject:(id)objeto`

Proporciona `YES` si alguno de los elementos de la lista produce `YES` al invocar al método `equals:` con ese objeto como argumento. Esto es, si `YES == [[miArray objectAtIndex:i] equals:objeto]` para algún `i`, entonces este `containsObject:` proporciona `YES`, o bien `NO` si ningún objeto de la lista satisface la condición anterior.

MATRICES INMUTABLES

EJEMPLO

Crear una aplicación que muestre el uso de los principales métodos de **`NSArray`**, creando uno o más ejemplares y mostrando la aplicación de los métodos indicados anteriormente. Téngase en cuenta que **`NSArray`** es el predecesor de **`NSMutableArray`**, esto es, que **`NSMutableArray`** es la clase derivada, y no al revés.

proyecto: matrices_inmutables

IMPORTANTE

Para utilizar RGBTextInput hay que indicar a Xcode
la ubicación del .h y del .a

RGBTextInput reside en 5000_utiles

Resource Tags | Build Settings | Build Phases | Build Rules

PROJECT

matrices_inmutabl...

TARGETS

matrices_inmutabl...

Basic | All | Combined | Levels | + | header

Base SDK — Latest OS X (OS X 10.11)

▶ Build Locations

▶ Build Options

▶ Headers

▶ Linking

▶ Packaging

▼ Search Paths

Setting	matrices_inmutables_2
▶ Always Search User Paths	No
Framework Search Paths	
Header Search Paths	/usr/local/include/
Use Header Maps	Yes
User Header Search Paths	

matrices_inmutables_2

Resource Tags | Build Settings | Build Phases | Build Rules

PROJECT

matrices_inmutabl...

TARGETS

matrices_inmutabl...

Basic | All | Combined | Levels | +

linker

▼ Architectures

Setting	matrices_inmutables_2
Base SDK	Latest OS X (OS X 10.11)

▶ Build Options

▼ Linking

Setting	matrices_inmutables_2
Display Mangled Names	No
Link With Standard Libraries	Yes
Other Linker Flags	**-lutiles**
▼ Path to Link Map File	<Multiple values>
Debug	build/matrices_inmutables_2.build/Deb
Release	build/matrices_inmutables_2.build/Rele
Perform Single-Object Prelink	No
Quote Linker Arguments	Yes
Warning Linker Flags	
Write Link Map File	No

▼ Search Paths

Setting	matrices_inmutables_2
Framework Search Paths	
Library Search Paths	**/usr/local/lib/**

EJEMPLO

```
// proyecto: matrices_inmutables
// archivo: DieFestplatte.h
#import <Foundation/Foundation.h>
#import "RGBTextInput.h"

@interface DieFestplatte : NSObject

@property NSString * fabricante, *modelo, *tecnologia;
@property float megabytes;
@property int velocidadDeRotacion, tasaDeTransmision;
@property NSString * tipoDeInterface, *numeroDeSerie;

-(void)readFromKeyboard;

@end
```

Ojo falta (readwrite, copy)

EJEMPLO

```
#import "DieFestplatte.h"

@implementation DieFestplatte

-(void)readFromKeyboard
{
  self.fabricante = [RGBTextInput readStringWithPrompt:@"Escriba el fabricante"];
  self.modelo     = [RGBTextInput readStringWithPrompt:@"Escriba el modelo    "];
  self.tecnologia = [RGBTextInput readStringWithPrompt:@"Escriba la tecnología"];
  /* Peligro... el locale nos puede dar un susto */
  self.megabytes           = [[RGBTextInput readStringWithPrompt:@"Escriba la capacidad en megabytes"]
floatValue];
  self.velocidadDeRotacion = [[RGBTextInput readStringWithPrompt:@"Escriba la velocidad de rotación "] intValue];
  self.tasaDeTransmision   = [[RGBTextInput readStringWithPrompt:@"Escriba la tasa de transmisión   "] intValue];
  self.tipoDeInterface     =  [RGBTextInput readStringWithPrompt:@"Escriba el tipo de interface     "];
  self.numeroDeSerie       =  [RGBTextInput readStringWithPrompt:@"Escriba el número de serie       "];
}

-(NSString*)description
{
  return [NSString stringWithFormat:@"{%@,%@,%@,%f,%d,%d,%@,%@}",
                        _fabricante,
                        _modelo,
                        _tecnologia,
                        _megabytes,
                        _velocidadDeRotacion,
                        _tasaDeTransmision,
                        _tipoDeInterface,
                        _numeroDeSerie
                        ];
}

@end
```

EJEMPLO

```
// proyecto: matrices_inmutables
// archivo: DieFestplatte.m
#import <Foundation/Foundation.h>
#import "DieFestplatte.h"

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        DieFestplatte * f = [DieFestplatte new];
        [f readFromKeyboard];
        NSLog(@"Características: %@", f);
        NSArray * array = @[@"Hallo, Welt!",
                            [NSNumber numberWithBool:YES],
                            [NSNumber numberWithFloat:3.14f],
                            f];
        NSLog(@"La matriz contiene: %@", array);

    }
    return 0;
}
```

DESDE LA LÍNEA DE ÓRDENES

Forma de compilar el ejercicio de matrices inmutables dos. Lo primero es compilar la clase independiente:

```
clang -c DieFestplatte.m
```

Esto se hace desde en la carpeta del proyecto, en la cual se ve directamente esa clase. El paso siguiente consiste en combinar la clase principal, que reside en un subdirectorio. Para compilar el trabajo principal, se utiliza la idea de órdenes siguiente:

```
clang -c -I . matrices_inmutables_2/main.m
```

Cuando ya se dispone de los dos archivos de código objeto, se da una tercera orden que compila esos dos archivos y además añade el framework Foundation.

```
clang -framework Foundation -lutiles
DieFestplatte.o main.o -o m2
```

Nota: parece que no se pueden utilizar "_" en el nombre de la aplicación(herramientas) producida.
Nota: es fácil crear un guión q compile todo :)
Nota: xcodebuild también crea la aplicación

Demo

Proyecto: `matrices_inmutables`

NSMUTABLEARRAY

NSMUTABLEARRAY

Esta clase desciende de **`NSArray`** y le proporciona la capacidad de añadir o eliminar elementos. Para crear una lista mutable a partir de una inmutable, se utiliza el método `mutableCopy` de `NSArray`:

```
NSMutableArray * copiaMutable = [listaInmutable mutableCopy];
```

NSMUTABLEARRAY - METODOS

`-(void)addObject:(id)objeto`

Inserta `objeto` al final de la lista. No se permite añadir el valor `nil`.

`-(void)addObjectsFromArray:(NSArray*)otraLista`

Añade todos los elementos de `otraLista` al final de la lista.

NSMUTABLEARRAY - METODOS

`-(void)insertObjet:(id)objeto atIndex:(unsigned)i`

Inserta `objeto` en la posición dada por i, que no puede ser mayor que el número de objetos, o se producirá un error. Si la posición ya está ocupada, se desplaza su contenido (y los siguientes) para alojar el nuevo elemento en esa posición.

`-(void)removeAllObjects`

Vacía la lista.

NSMUTABLEARRAY - METODOS

`-(void)removeObjectAtIndex:(unsigned)i`

Descarta el objeto situado en la posición dada por `i`, y desplaza el contenido de las posiciones siguientes, si las hay, para cerrar el hueco. `i` no puede ser posterior al último elemento.

Para "dejar un hueco" se puede añadir como elemento el valor proporcionado por la expresión `[NSNull null]`, que proporciona un puntero de la clase `NSNull`, una clase nula que tiene un único ejemplar.

EJEMPLO

Crear un ejemplar de **`NSMutableArray`** e insertar en él dos listas de diez números enteros (NSNumber).

La primera lista consta de los números del 0 al 9.

La segunda lista consta de los mismos números que la primera multiplicados por dos, y debe insertarse antes que la primera.

Finalmente, mostrar en pantalla el contenido total de la colección.

proyecto: **`matrices_mutables`**

EJEMPLO

```
// Proyecto otra_coleccion
#import <Foundation/Foundation.h>

int main(int argc, const char * argv[])
{

    @autoreleasepool {
        NSMutableArray * ma = [[NSMutableArray alloc] init ];
        for (int i=0; i<10; ++i) {
            [ma insertObject:[NSNumber numberWithInt:i] atIndex:i];
        }
        for (int i=0; i<10; ++i) {
            [ma insertObject:[NSNumber numberWithInt:i*2] atIndex:i];
        }
    NSLog(@"La lista contiene %lu elementos:",[ma count]);
        for (int i=0; i<[ma count]; i++) {
            NSLog(@"ma[%d] = %@",i, [ma objectAtIndex:i]);
        }
}
return 0;
}
Este programa completo muestra el uso de métodos con 1 y 2 argumentos.
```

Demo

Proyecto: `otra_coleccion`

NSSTRING

NSSTRING

`NSString` es una lista de caracteres Unicode. Cocoa hace uso de ejemplares de esta clase para todas las manipulaciones de cadenas.

Para crear un ejemplar de `NSString`, Objective-C ofrece la expresión `@"..."`, que genera un ejemplar de `NSString` cuyo contenido es el indicado por los caracteres dados, como se ha visto en los ejemplos anteriores.

```
NSString * cadena = @"María tiene una oveja";
```

`NSString` es inmutable, pero `NSMutableString`, que hereda de `NSString`, es mutable.

NSSTRING - METODOS

```
+(id)stringWithFormat:
(NSString*)formato,...
```

Este método es similar a `sprintf()`, siendo `formato` la cadena que contiene los especificadores (que son los de `NSLog`, ya vistos). Los puntos suspensivos denotan una colección de variables, separadas mediante comas, cuyos valores se traducen con los especificadores, devolviéndose esa cadena, en un ejemplar de `NSString`, como resultado.

NSSTRING - METODOS

```
-(NSUInteger)length
```

Este método proporciona el número de caracteres que hay en la cadena.

```
- (NSString*)stringByAppendingString:(NSString*)cadena
```

Este método toma el argumento cadena, concatena su cadena al de la cadena receptora del mensaje y proporciona el resultado en una nueva cadena.

NSSTRING - METODOS

`-(NSComparisonResult)compare:otraCadena`

Este método proporciona `NSOrderedAscending`, `NSOrderedSame` o `NSOrderedDescending` según la cadena receptora sea alfabéticamente anterior, igual o posterior a `otraCadena`.

`-(NSComparisonResult)caseInsensitiveCompare:otraCadena`

Este método es análogo al anterior, pero no distingue entre mayúsculas y minúsculas.

EJEMPLO

Se dispone de una colección de nombres, otra de tallas y otra de ciudades. Todas las listas tienen el mismo número de elementos. Se pide escribir un programa que construya una sola cadena (un ejemplar de `NSMutableString`) que contenga todos los datos, ordenados en forma de registros {nombre, talla, ciudad} con formato delimitado por asteriscos. Las llaves son solo indicativas; el objetivo es solamente guardar con formato delimitado por asteriscos (podría ser por tabuladores o cualquier otro).

EJEMPLO

```
// Proyecto: metodos_nsstring
#import <Foundation/Foundation.h>

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        NSArray * nombres = [[NSArray alloc] initWithObjects:@"Juan Pérez",
                                                             @"Ana Sánchez",
                                                             @"Manuel López",
                                                             nil];
NSArray * alt= [[NSArray alloc] initWithObjects:[NSNumber numberWithFloat:1.75],
                                                [NSNumber numberWithFloat:1.6],
                                                [NSNumber numberWithFloat:1.68],
                                               ,nil];
NSArray * direcciones = [[NSArray alloc] initWithObjects:@"Salamanca",
                                                         @"Toledo",
                                                         @"Villar de Peralonso",
                                                        nil];
NSMutableString * deposito = [[NSMutableString alloc] init ];
```

EJEMPLO

```
for (int i=0; i<[nombres count]; i++) {
    NSString * temp = [NSString stringWithFormat:@"%-20s%4.2f%30s\n",
                                        [[nombres objectAtIndex:i] cString],
                                        [[alt objectAtIndex:i] floatValue],
                                        [[direcciones objectAtIndex:i] cString]];
    [deposito appendString:temp];
}
        NSLog(@"\n\n%@",deposito);
    }
    return 0;
}
```

Este programa muestra una aplicación de los métodos `stringWithFormat:`, `appendString:` y `length` de `NSString`. *Esta clase posee muchos otros métodos, que conviene conocer para no reinventar la rueda*.

Demo

Proyecto: `metodos_nsstring`

CLASES PROPIAS

PENSAMIENTOS

En Objective-C se utiliza mucho más la composición que la herencia. Esto es, para formar nuevas clases se hace uso de otras ya existentes; es muy infrecuente crear clases nuevas por herencia de otras.

CREACIÓN DE CLASES PROPIAS

El concepto de clase, sobradamente conocido, implica declarar atributos y métodos. La sintaxis de Objective-C se parece a la de C++, porque las clases se escriben en dos archivos (uno de encabezado, con la extensión `.h`, y otro de implementación, con la extensión `.m`).

Todas las clases de Objective-C descienden de **`NSObject`**, y esto implica que la declaración de las clases debe dar cuenta de este hecho.

CREACIÓN DE CLASES PROPIAS

La declaración de una clase en Objective-C consta de dos archivos. El primero de ellos, que es la interfaz o archivo de encabezado, tiene el aspecto siguiente:

```
@interface NombreDeLaClase : NombreDeLaSuperclase
    // Declaraciones de métodos
    // Declaraciones de propiedades
@end
```

El archivo de implementación, a su vez, tiene este aspecto:

```
@implementation NombreDeLaClasae
  // Síntesis de propiedades (opcional)
  // Implementaciones de métodos
@end
```

ELEMENTOS QUE SIGUEN A @INTERFACE

La palabra reservada `@interface` va seguida obligatoriamente por el nombre de la clase, que es cualquier identificador válido en Objective-C. Opcionalmente, se pueden escribir dos elementos más.

- Una (y una sola) superclase, de la cual hereda la clase creada. El nombre de la clase va separado del nombre de la superclase mediante ":", en la forma siguiente:

```
@interface NombreDeLaClase : NombreDeLaSuperclase
```

Si una clase no desciende por herencia de ninguna otra clase, entonces desciende directamente de `NSObject`. Si la clase no se deriva de otra, y no implemente ningún protocolo, la declaración va a tener este aspecto:

```
@interface NombreDeLaClase : NSObject
```

ELEMENTOS QUE SIGUEN A @INTERFACE

- Opcionalmente, la clase puede adoptar uno o más protocolos (como diríamos en Java, una clase puede implementar una o más interfaces). Esto se indica escribiendo la lista de protocolos a la derecha del nombre de la superclase. La lista de protocolos va encerrada entre corchetes angulares, y los nombres de protocolos van separados mediante comas. El aspecto de una clase que implementa uno o más protocolos es el siguiente:

```
@interface Clase : Superclase <Prot_1,Prot_2,...,Prot_N>
```

Si `Persona` adopta el protocolo `NSCoding`, se escribe

```
@interface Persona : NSObject <NSCoding>
```

que en Java sería

```
Persona extends Object implements Coding
```

DECLARACIÓN DE MÉTODOS

En Objective-C existen dos clases de métodos:

- **Métodos de clase**, que se pueden ejecutar empleando el nombre de la clase como destinatario de mensaje. Los métodos de clase llevan el signo + delante del tipo proporcionado.(Son los métodos `static` de Java)

```
@interface Clase : NSObject
+(id)nameOfClassMethod;
@end
```

- **Métodos de ejemplar**, que se pueden ejecutar empleando como destinatario el puntero de un ejemplar de la clase. Los métodos de ejemplar llevan el signo - delante del tipo proporcionado.

```
@interface Clase : NSObject
-(unsigned)nameOfInstanceMethod
@end
```

Obsérvese que el tipo proporcionado se escribe entre paréntesis.

DECLARACIÓN DE MÉTODOS

Las declaraciones de los métodos de clase suelen situarse en primer lugar, pero no es un requisito del compilador.

```
@interface Clase : NSObject
+(id)nameOfClassMethod;
-(unsigned)nameOfInstanceMethod
@end
```

Los métodos anteriores no reciben parámetros. Para indicar que un método recibe un solo parámetro, se añaden al nombre del método dos puntos, el tipo del parámetro entre paréntesis y el nombre del parámetro:

```
@interface Lista : NSObject
-(void)anadirElemento:(float)elemento;
@end
```

EJEMPLO

Crear una clase dotada de un método de clase y otro de ejemplar. Ejercitar la clase, construyendo un ejemplar de la clase y mostrando la forma de utilizar ambos métodos.

proyecto: **`TiposDeMetodos`**

EJEMPLO

```
// proyecto: TiposDeMetodos
// archivo: ClassWithMethods.h
#import <Foundation/Foundation.h>

@interface ClassWithMethods : NSObject
@property NSString * attribute1;
@property float attribute2;
+(void)classMethod;
-(void)instanceMethod;
@end
```

EJEMPLO

```
// proyecto: TiposDeMetodos
// archivo: ClassWithMethods.m
#import "ClassWithMethods.h"

@implementation ClassWithMethods

- (id)init
{
    self = [super init];
    if (self) {
        _attribute1 = @"*Value of attribute1*";
        _attribute2 = 33;
    }
    return self;
}

+(void)classMethod
{
    NSLog(@"This is the class method\n\n");
    // Cannot access instance attributes
    //NSLog(@"attribute_1 = %@ and attribute_2 =
%f",
    // _attribute1, _attribute2);

    // Can access class methods through self
(recursive!)
    //[self classMethod];

    // Cannot access instance methods through
self
    //[self instanceMethod];

}

-(void)instanceMethod
{
    NSLog(@"Results of the instance method:\n
\n");
    NSLog(@"attribute_1 = %@ and attribute_2 = %f
\n\n",
          _attribute1, _attribute2);

    // One can call class methods from instance
methods
    //[ClassWithMethods classMethod];
}

@end
```

EJEMPLO

```
// proyecto: TiposDeMetodos
// archivo: ClassWithMethods.m
#import <Foundation/Foundation.h>
#import "ClassWithMethods.h"

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        // Para llamar a un método de clase
        // se usa la clase como receptor del mensaje; esto es lo normal

        [ClassWithMethods classMethod];

        // La clase no puede llamar a métodos de ejemplar
        // [ClassWithMethods instanceMethod];

        ClassWithMethods * cwm = [ClassWithMethods new];

        // Un ejemplar de la clase puede, naturalmente, llamar a métodos de ejemplar
        [cwm instanceMethod];

        // También se puede llamar a un método de clase mediante el puntero de un ejemplar
        //[cwm classMethod];

    }
    return 0;
}
```

Demo

Proyecto: `TiposDeMetodos`

DECLARACIÓN DE MÉTODOS

DECLARACIÓN DE MÉTODOS

Para declarar un método con dos parámetros, se añade al nombre del método el primer parámetro, como se ha hecho en el caso anterior, y después:

- Una etiqueta de parámetro (un identificador que comience por una letra)
- dos puntos
- el tipo del parámetro entre paréntesis
- el nombre del parámetro

DECLARACIÓN DE MÉTODOS

```
@interface Lista : NSObject
-(id)init;
-(void)anadirElemento:(float)elemento
-(void)insertarElemento:(float)elemento enPosicion:(unsigned)i;
@end;
```

El primer método recibe un solo argumento, `elemento`, de tipo `float`. El segundo método recibe un argumento de tipo `float` y otro de tipo `unsigned int`. La llamada a estos métodos sería como sigue:

```
Lista * l = [[Lista alloc] init];
[l anadirElemento:3.14];
[l insertarElemento:2.71 enPosicion:0];
```

Obsérvese que la etiqueta del segundo argumento, `enPosicion`, facilita aportar los argumentos por orden correcto.

NOMBRES DE LOS MÉTODOS

Los nombres de los métodos de la clase `Circulo`, a efectos de Objective-C, son como sigue:

```
init
anadirElemento:
insertarElemento:enPosicion:
```

De hecho, estas expresiones se denominan selectores. La importancia de los selectores, es grande, porque se pueden seleccionar durante la ejecución del programa.

IMPLEMENTACIÓN DE LOS MÉTODOS

Una vez efectuada la declaración, la implementación de los métodos se efectúa en otro archivo, cuyo nombre es el de la clase, y cuya extensión es `.m`. Así, la clase `List` se declara en `Lista.h` y se implementa en `Lista.m`.

IMPLEMENTACIÓN DE LOS MÉTODOS

```
@implementation
-(id)init
{
  // Contenido del método
}
-(void)anadirElemento:(float)elemento
{
  // Contenido del método
}
-(void)insertarElemento:(float)elemento enPosicion:(int)i
{
  // Contenido del método
}
@end
```

EJEMPLO

Crear una clase adecuada para representar listas de **`float`**. La clase dispondrá de los métodos de clase y de ejemplar necesarios para realizar su función.

Este tipo de clases es interesante cuando se dispone de un cierto número de variable de tipo **`float`**. Si la lista formada por esas variables sufre modificaciones continuas, tales eliminaciones e inserciones, necesitamos un tipo de datos que sea un intermedio entre una corrección genérica y una lista de números de coma flotante.

EJEMPLO

La clase que proponemos se adapta bastante bien a esta funcionalidad, aunque sin duda podrían añadírsele otros métodos.

proyecto: **`ListaDeFloats`**

EJEMPLO

```
// proyecto:ListaDeFloats
// archivo:Lista.h
#import <Foundation/Foundation.h>

@interface Lista : NSObject

@property NSMutableArray * datos;


- (id) init;
- (void)anadirElemento:(float)elemento;
- (void)insertarElemento:(float)elemento enPosicion:(unsigned)i;
- (float)obtenerElementoEnPosicion:(unsigned)i;
- (NSUInteger)numElementosEnLista;

@end
```

EJEMPLO

```
// proyecto:ListaDeFloats
// archivo:Lista.m (I)
#import <Cocoa/Cocoa.h>
#import "Lista.h"

@implementation Lista

- (id)init
{
    self = [super init];
    if (self) {
        _datos = [[NSMutableArray alloc] init];
    }
    return self;
}

- (void)anadirElemento:(float)elemento
{
    NSNumber * numero = [NSNumber numberWithFloat:elemento];
    [_datos addObject:numero];
}
...
```

EJEMPLO

```
// proyecto:ListaDeFloats
// archivo:Lista.m (II)
- (void)insertarElemento:(float)elemento enPosicion :(unsigned int)i
{
    if (i<[_datos count]) {
        NSNumber * numero = [NSNumber numberWithFloat:elemento];
        [_datos insertObject:numero atIndex:i];
    }
    else {
        NSLog(@"Error al insertar, posición incorrecta");        NSBeep();
    }
}

- (float)obtenerElementoEnPosicion:(unsigned int)i
{
    if(i<[_datos count]){
        return [[_datos objectAtIndex:i] floatValue];
    }
    else {
        NSLog(@"Error al obtener un elemento, posición incorrecta");        NSBeep();
        return 0;
    }
}
...
```

EJEMPLO

```
// proyecto:ListaDeFloats
// archivo:Lista.m (y III)

- (NSUInteger)numElementosEnLista
{
    return [_datos count];
}
@end
```

EJEMPLO

```
// proyecto:ListaDeFloats
// archivo:main.m

- (NSUInteger)numElementosEnLista
{
    return [_datos count];
}
@end
```

Demo

Proyecto: `ListaDeFloats`

@PROPERTY (PROPIEDADES)

PROPIEDADES

En programas antiguos es frecuente encontrar una sintaxis de Objective-C parecida a la primera que tuvo C. Esto es, se encuentran frecuentemente declaraciones como esta:

```
#import <Foundation/Foundation.h>

@interface Lista : NSObject {

    NSMutableArray * datos;
}

- (id) init;
- (void)anadirElemento:(float)elemento;
- (void)insertarElemento:(float)elemento enPosicion:(unsigned)i;
- (float)obtenerElementoEnPosicion:(unsigned)i;
- (unsigned)numElementosEnLista;

@end
```

PROPIEDADES

En la actualidad, los atributos o propiedades (campos de datos) de una clase suelen manejarse a través de métodos de acceso, con objeto de respetar el principio de encapsulamiento. De este modo se aísla a los clientes de la clase de posibles cambios en la implementación.

La creación de métodos de acceso (`get` y `set` en Java) resulta tediosa, especialmente cuando son muchos los atributos. Además, conviene dar a conocer si los métodos de acceso son atómicos, y, en el caso de los métodos `set`, conviene saber si efectúan una copia del argumento proporcionado.

PROPIEDADES

La solución que ofrece Objective-C consiste en crear métodos de acceso en dos etapas, una de declaración y otra de implementación.

La fase de declaración de propiedades tiene la forma siguiente:

```
@property (atributos opcionales) tipoDeDatos nombreDeLaPropiedad;
```

La fase de implementación de métodos de acceso tiene la forma siguiente:

```
@synthesize nombre;// Opcional
```

PROPIEDADES

La declaración de una propiedad mediante @property da lugar a que se creen internamente (no son visibles) dos métodos de acceso para el atributo considerado, de aspecto análogo al siguiente:

```
- (tipo)atributo
{
  return atributo;
}

- (void)setAtributo:(tipo)valorAtributo
{
  atributo = valorAtributo;
}
```

Se han creado los métodos de acceso (los *getters* y *setters* de Java y C++). Obsérvese que el método de consulta, el getter, NO lleva get delante! Esta convención se sigue en todas las clases de Cocoa.

PROPIEDADES

La referencia oficial del mecanismo de propiedades se hallará en este documento

https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/ProgrammingWithObjectiveC.pdf

que describe detalladamente las técnicas habituales (y menos habituales) de programación propias de Objective-C.

NSDATE

NSDATEFORMATTER

EL PASO DEL TIEMPO

Cocoa ofrece varias clases que permiten un sofisticado manejo de fechas y tiempos transcurridos entre fechas. Las clases correspondientes son `NSDate` y `NSCalendar`, junto con otras accesorias como `NSDateFormatter` y `NSDateComponents`.

NSDATE

La clase `NSDate` representa un instante de tiempo inmutable. Posee muchos métodos, de los cuales mencionaremos solo algunos.

```
+(id)date
```

Obsérvese que este es un método de clase (lleva delante el signo +). Por tanto, no es preciso crear un ejemplar de `NSDate` para invocar al método date, que produce como resultado un ejemplar de `NSDate` cuyo estado describe la fecha y hora en que se invoca:

```
NSDate * ahora = [NSDate date];
```

NSDATE

```
-(id)dateByAddingTimeInterval:(NSTimeInterval)intervaloDeTiempo;
```

Proporciona una fecha que se obtiene *sumando* a la fecha y hora representada por el receptor el intervalo de tiempo indicado como argumento. `NSTimeInterval` es un número de segundos (de tipo `double`).

```
- (NSTimeInterval)timeIntervalSinceDate:(NSDate*)otraFecha
```

Proporciona el intervalo de tiempo transcurrido *desde* la fecha dada como argumento *hasta* la fecha representada por el receptor del mensaje, medido en segundos. Si la fecha denotada por el receptor es anterior a `otraFecha`, el resultado es negativo.

NSDATE

```
+(NSTimeInterval)timeIntervalSinceReferenceDate;
```

Proporciona el tiempo transcurrido desde el 1 de Enero de 2001 (Greenwich Mean Time) hasta el momento en que se invoca.

```
- (NSComparisonResult)compare:(NSDate*)otraFecha
```

Proporciona `NSOrderedAscending` si el receptor es anterior a otraFecha, `NSOrderedSame` si son iguales y `NSOrderedDescending` si el receptor es posterior a `otraFecha`. Por así decir, compara de izquierda (el receptor) a derecha (`otraFecha`).

NSDATE

El proyecto Loto, que se muestra próximamente (tras el estudio de los iniciadores designados) muestra la forma de operar con fechas.

"CONSTRUCTORES"

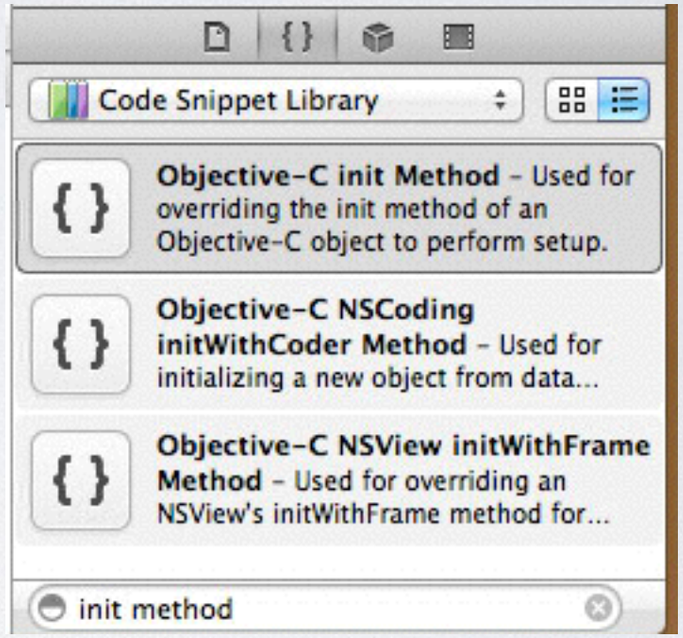
En Objective-C no existe el mecanismo de constructores que tienen Java o C++, pero por convención las clases disponen de métodos denominados "iniciadores" que efectúan precisamente la misión de los constructores. Estos métodos reciben el nombre "`init`" u otros parecidos, en función de los argumentos que reciban.

El papel de los dtores está a cargo de otro método, llamado `dealloc`, cuya misión es liberar todos los objetos que, no siendo ya útiles, hayan sido creados por nuestra clase.

Tanto `init` como `dealloc` están íntimamente relacionados con el manejo de memoria en Objective-C. En la actualidad, dealloc está cayendo en desuso porque ARC suprime la utilidad de este método.

"CONSTRUCTORES"

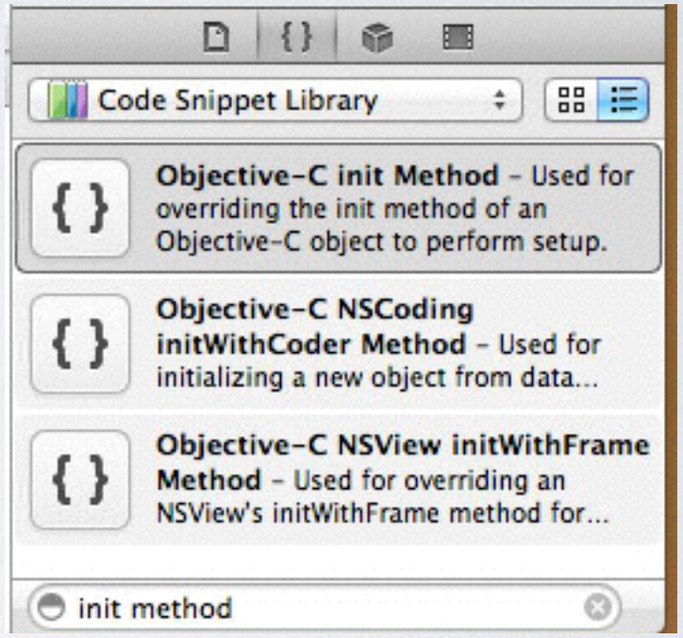
Para crear el método `init` resulta cómodo usar un de los fragmentos de código que ofrece Xcode.
Si se escribe `init method` en la parte inferior de la biblioteca de fragmentos de código, que se encuentra en la esquina inferior derecha de Xcode, el primer fragmento seleccionado es precisamente el deseado, que se puede arrastrar hasta el archivo de implementación de la clase que se está construyendo.



"CONSTRUCTORES"

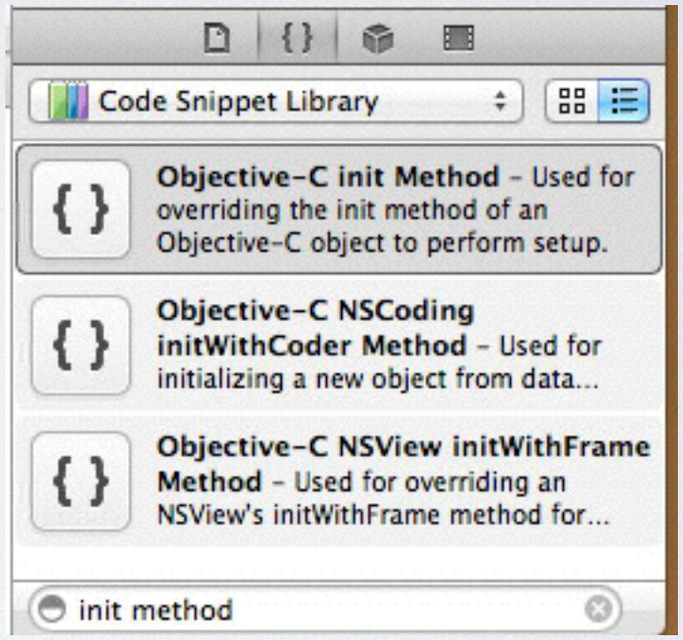
El aspecto que tiene el fragmento de código generado es el siguiente:

```
- (id)init
{
    self = [super init];
    if (self) {
        // Aquí se inserta el código necesario para
        // poner todos los atributos de la clase en
        // condiciones iniciales.

    }
    return self;
}
```



"CONSTRUCTORES"

Evidentemente, hay ocasiones en que resulta conveniente aportar datos al constructor. Dicho de otro modo, es conveniente crear un iniciador con argumentos. Veamos un iniciador que recibe como argumento una fecha:




```
- (id)initWithEntryDate:(NSDate*)laFecha
{
    self = [super init];
    if (self) {
        // Aquí se inserta el código necesario para
        // poner todos los atributos de la clase en
        // condiciones iniciales.

    }
    return self;
}
```

“CONSTRUCTORES”

Si la clase que tiene este iniciador fuera utilizada por alguien que no se diera cuenta de la existencia del iniciador con argumentos, podría crear un ejemplar en la forma habitual:

```
MiClase * p = [[MiClase alloc] init];
```

Esto no es un error de compilación... pero van a producir problemas, porque el método `init` ejecutado será el de `NSObject` (o el de la clase padre, si la hay), y los atributos de nuestra clase quedarán simplemente puestos a cero (no habrá un valor para la fecha).

"CONSTRUCTORES"

La solución consiste en crear un método `init` que llame a `initWithEntryDate:`, dando a la fecha un valor predeterminado (y un valor razonable a todos los demás atributos). Así evitamos el problema.

```
- (id) init
{
  return [self initWithEntryDate:[Date date]];
}
```

El **iniciador designado** (*designated initializer*) es normalmente el que recibe más argumentos, y tiene una propiedad especial:

¡Todos los iniciadores deben llamar obligatoriamente al iniciador designado!

"CONSTRUCTORES"

Las reglas de creación (o no) de iniciadores son como sigue:

- Si el iniciador de la predecesora basta, no es preciso crear un iniciador en la clase derivada (porque al hacer `[[xxx alloc] init]` ya se ejecuta el iniciador de la clase predecesora.

- Si se crea un iniciador en una clase derivada por herencia, éste debe redefinir el iniciador designado de la clase base .

- Si se crean varios iniciadores, solo un de ellos efectúa realmente las tareas de iniciación (el iniciador designado). Todos los demás iniciadores llaman al iniciador designado.

- Por último, el iniciado designado invoca directamente al iniciador designado de la clase predecesora.

"CONSTRUCTORES"

Los iniciadores `init` e `initWithEntryDate:` del ejemplo anterior satisfacen estas reglas:

- Se crea un iniciador porque hay que dar valores iniciales a los atributos de la clase (véase el proyecto `Loto`; cada `Apuesta` contiene dos números y una fecha).
- Se redefine el método `init`, que es el iniciador designado de `NSObject`,porque `Apuesta` se deriva de `NSObject`.
- El iniciador designado, `initWithEntryDate:`, es el que efectúa todas las tareas de iniciación. Además, `init` llama a `initWithEntryDate:`
- Por último, `initWithEntryDate:` llama al iniciador designado de `NSObject`, porque la primera línea de código es `[super init]`.

“CONSTRUCTORES”

¿Qué sucede si una clase necesita imprescindiblemente recibir un parámetro en su iniciación? Podemos hacer forzoso el uso directo del iniciador designado empleando un `init` como el siguiente:

```
-(id)init
{
@throw [NSException exceptionWithName:@”Iniciación incorrecta!!!”
                                reason:@”Toda fracción TIENE dividendo y divisor”
                              userInfo:nil];
return nil;
}
```

EJEMPLO

Crear un programa que muestre una lista de apuestas, creando una clase llamada `Apuesta` que debe estar dotada de los métodos de iniciación correspondiente. El programa debe mostrar 2 números por cada apuesta, para 10 semanas, comenzando en la semana actual.
Proyecto: `Loto`

EJEMPLO

```
// proyecto: Loto
archivo: Apuesta.h
#import <Foundation/Foundation.h>

@interface Apuesta : NSObject

@property  NSDate * fecha;
@property (readonly) int primerNumero,segundoNumero;

- (id)initWithEntryDate:(NSDate*)laFecha;
@end
```

EJEMPLO

```
// proyecto: loto
// archivo: Apuesta.m
#import "Apuesta.h"

@implementation Apuesta
@synthesize fecha, primerNumero,segundoNumero;


- (id)initWithEntryDate:(NSDate*)laFecha
{
    //
    // Este es el iniciador designado, que resulta necesario
    // porque hay q dar valor inicial a primerNumero, segundoNumero
    // y fecha.
    // Obsérvese que llama al iniciador designado de la clase predecesora
    //
    self = [super init];
    if (self) {
        fecha = laFecha;
        primerNumero = ((int)random() % 100) +1;
        segundoNumero = ((int)random() % 100) +1;

    }
    return self;
}
```

EJEMPLO

```
// proyecto: loto
// archivo: Apuesta.m (y II)

- (id) init
{
    return [self initWithEntryDate:[NSDate date]];
}

- (NSString*)description
{
    NSDateFormatter * formato_de_fecha = [NSDateFormatter new];
    [formato_de_fecha setDateStyle:NSDateFormatterMediumStyle];
    [formato_de_fecha setTimeStyle:NSDateFormatterNoStyle];

    NSString * fecha_con_formato = [formato_de_fecha stringFromDate:fecha];

    NSString * formato_descripcion = @"%@; un número es:%3d y el otro:%3d";

    NSString * resultado = [[NSString alloc] initWithFormat:formato_descripcion,
                                             fecha_con_formato,
                                             primerNumero,
                                             segundoNumero];

    return resultado;
}
@end
```

EJEMPLO

```
// proyecto: loto
// archivo: main.m

#import <Foundation/Foundation.h>
#define  NUMERO_DE_APUESTAS 10
#import "Apuesta.h"
```

```
int main(int argc, const char * argv[])
{
     @autoreleasepool {
        // Se preparan las variables
        NSDate * hoy = [NSDate new ];
        NSCalendar * calendario_actual = [NSCalendar currentCalendar];
        NSDateComponents * componentes_semana = [NSDateComponents new];
        // Se siembra el generador de números aleatorios
        srandom((unsigned)time(NULL));

        NSMutableArray * lista = [NSMutableArray new];
```

EJEMPLO

```
for(int i=0;i<NUMERO_DE_APUESTAS;i++) {

    [componentes_semana setWeek:i];

     NSDate * i_semanas_a_partir_de_hoy;
     i_semanas_a_partir_de_hoy = [calendario_actual dateByAddingComponents:componentes_semana
                           toDate:hoy
                           options:0];
          Apuesta * nueva_apuesta = [[Apuesta alloc] initWithEntryDate:i_semanas_a_partir_de_hoy ];
          [lista addObject:nueva_apuesta];
        }
        // Se muestra el contenido de la lista
        for (Apuesta * ap in lista) {
          NSLog(@"%@", ap);
        }
        // Curiosidad
        //NSLog(@"%@", lista);
    }
    return 0;
}
```

Demo

Proyecto: `Loto`

DEBUGGER

DEPURADOR - INTRO

Xcode permite utilizar fácilmente el depurador, esto es, posibilita crear puntos de ruptura, visualizar el contenido de las variables, ejecutar líneas individuales de código, etc.

Cuando hay un punto de ruptura en el programa, el IDE ejecuta del depurador en el momento en que arranca el programa.

Para crear o eliminar un punto de ruptura se ubica el cursor en el lugar deseado y se pulsa ⌘+⌥+\. Este equivalente de teclado activa *o desactiva* un punto de ruptura en la línea en que esté el cursor; se puede observar una flecha azul en la línea gris que aparece en el lado izquierdo del editor.

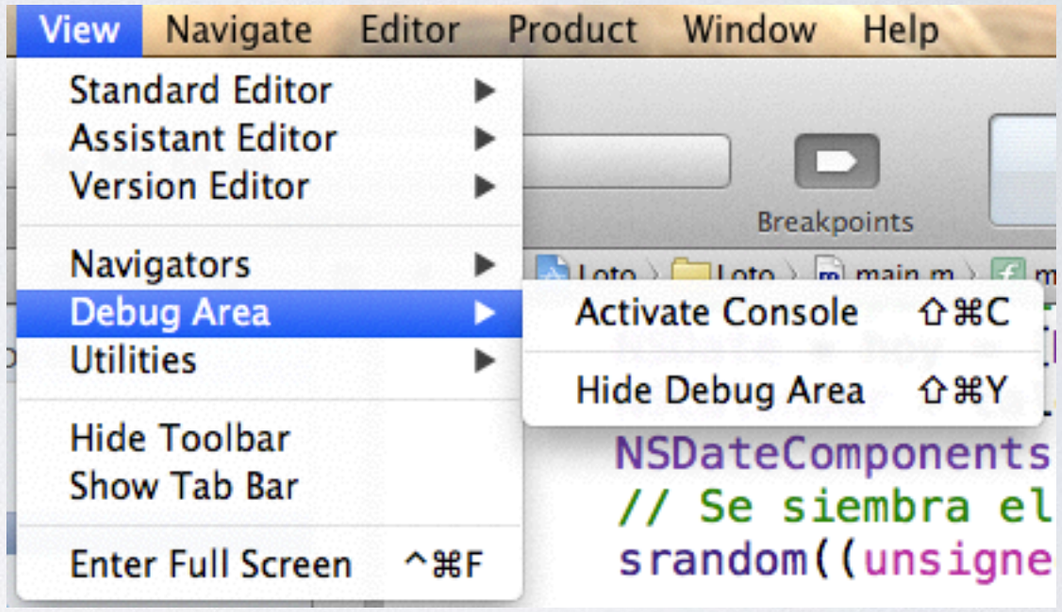
DEPURADOR - INTRO

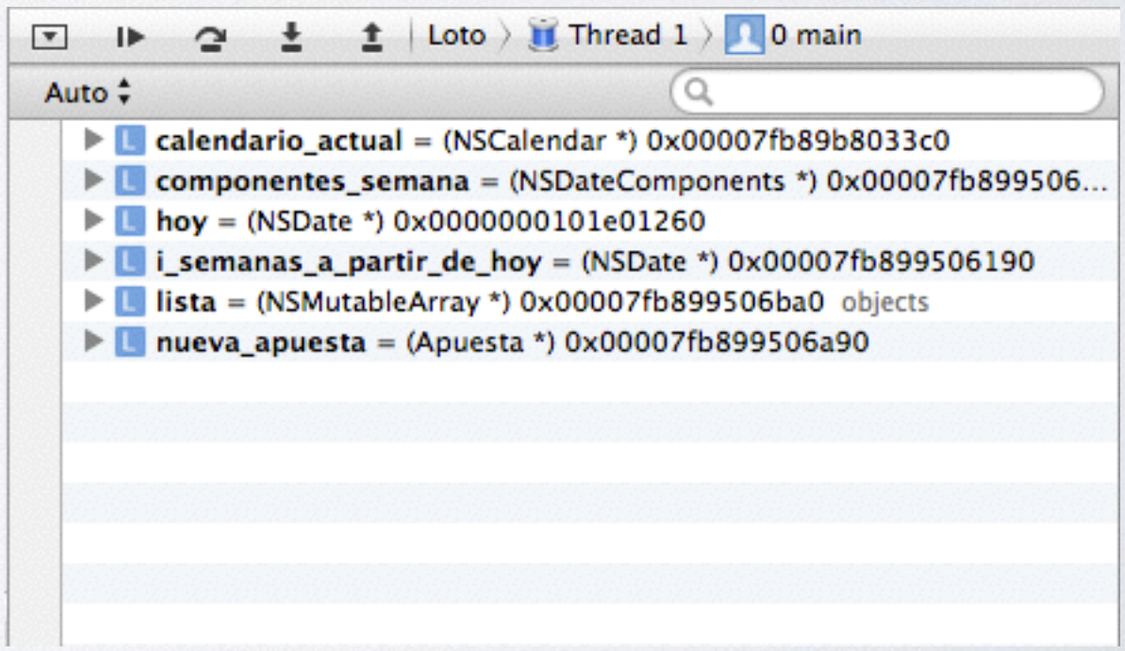
Al poner en marcha el programa, la ejecución se detiene automáticamente en el punto de ruptura, y es posible observar los valores de las variables en la parte inferior de la pantalla. Además, en el costado izquierdo se muestran los hilos que se estén ejecutando. Cocoa usa hilos de forma habitual, mejorando así el rendimiento del programa (especialmente en máquinas que dispongan de múltiples núcleos).

DEPURADOR - INTRO

Para mostrar u ocultar la zona de depuración se selecciona la opción `View->Debug Area->Show Debug Area` o se pulsa ⌘+⇧+Y.

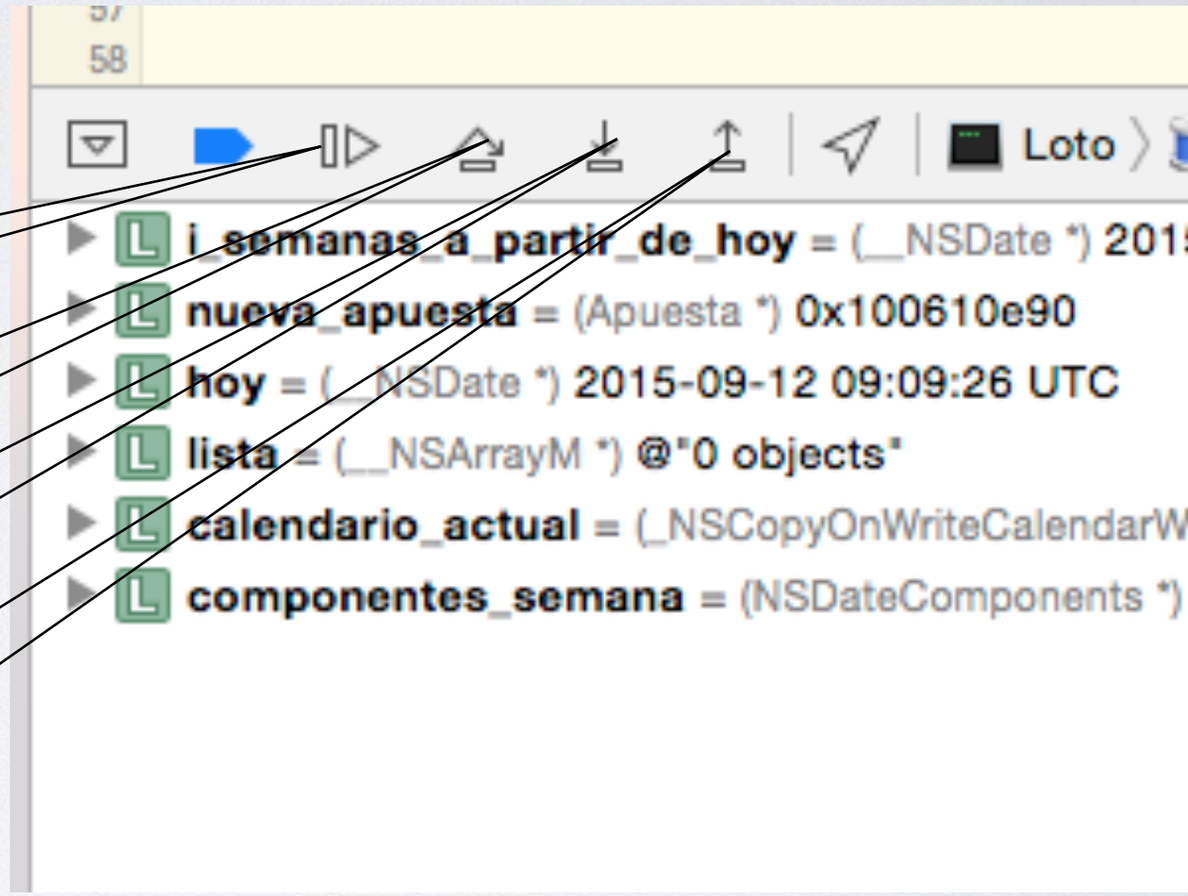
El depurador, una vez creado un punto de ruptura, se activa automáticamente cuando se ejecuta el programa. El resultado es una visión de las variables similar a la que se muestra a la derecha.





DEPURADOR - INTRO

Los iconos que aparecen en la parte superior de la ventana de variables permiten gobernar la ejecución del programa.



Proseguir la ejecución

Saltar instrucción (step over)

Examinar instrucción (step into)

Abandonar instrucción (step out)

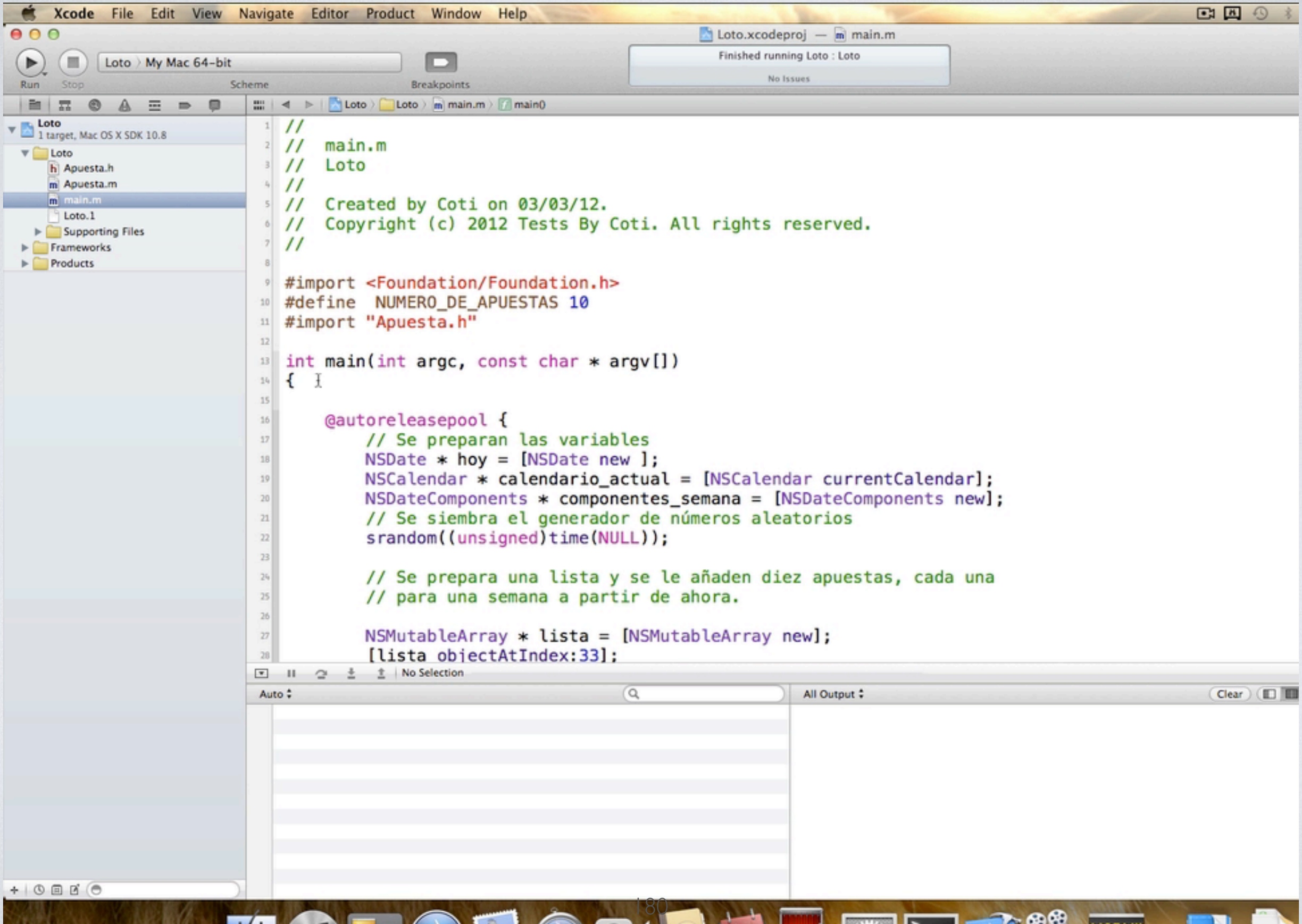
`lldb` y `gdb` pueden manejarse también a través de la línea de órdenes. Por ejemplo, `po`

DEPURADOR

El depurador también se activa cuando se produce una excepción, si previamente se ha creado un punto de ruptura adecuado.

El navegador de puntos de ruptura se activa seleccionando la opción `View->Navigators->Show Breakpoint Navigator`, o pulsando `⌘+6`. Una vez activado el navegador, se hace clic en el botón "+" situado en la esquina inferior izquierda del navegador, y se selecciona `Add Exception Breakpoint`, para añadir un punto de ruptura asociado a excepciones. El depurador señala el punto que ha dado lugar a la excepción.

DEPURADOR


Xcode File Edit View Navigate Editor Product Window Help
Loto.xcodeproj — main.m
Loto › My Mac 64-bit
Run Stop Scheme Breakpoints
Finished running Loto : Loto
No Issues
Loto › Loto › main.m › main()
Loto
1 target, Mac OS X SDK 10.8
Loto
Apuesta.h
Apuesta.m
main.m
Loto.1
Supporting Files
Frameworks
Products
//
// main.m
// Loto
//
// Created by Coti on 03/03/12.
// Copyright (c) 2012 Tests By Coti. All rights reserved.
//
#import <Foundation/Foundation.h>
#define NUMERO_DE_APUESTAS 10
#import "Apuesta.h"
int main(int argc, const char * argv[])
{
@autoreleasepool {
// Se preparan las variables
NSDate * hoy = [NSDate new];
NSCalendar * calendario_actual = [NSCalendar currentCalendar];
NSDateComponents * componentes_semana = [NSDateComponents new];
// Se siembra el generador de números aleatorios
srandom((unsigned)time(NULL));
// Se prepara una lista y se le añaden diez apuestas, cada una
// para una semana a partir de ahora.
NSMutableArray * lista = [NSMutableArray new];
[lista objectAtIndex:33];
No Selection
Auto
All Output
Clear

DEPURADOR

Para quienes estén habituados a utilizar asertos, la macro `NSAssert()` permite lanzar una excepción si falla una cierta condición establecida dentro de un método de una clase de Objective-C. La sintaxis es como sigue:

```
NSAssert(condicion, @"Mensaje de error");
```

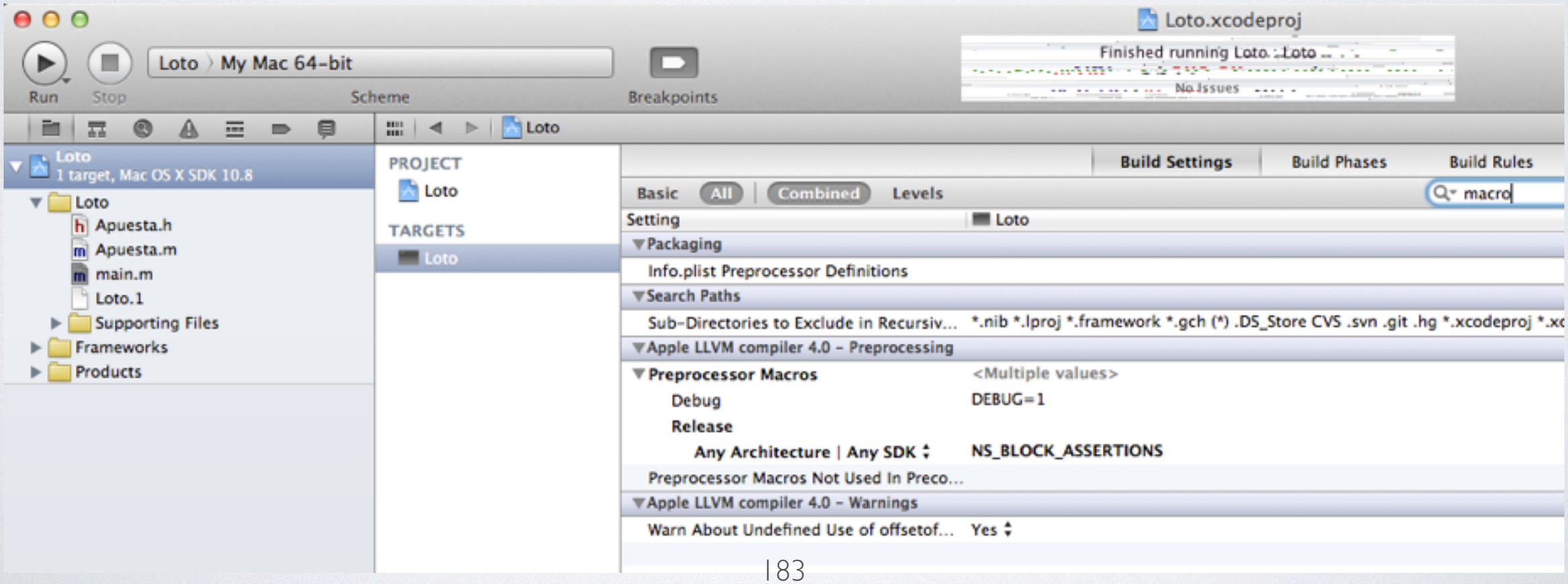
Si la condición toma el valor `NO`, entonces se lanza una excepción (`NSInternalInconsistencyException`) que muestra el mensaje dado como segundo argumento. La excepción se puede capturar estableciendo previamente un punto de ruptura, como se ha visto. Normalmente no se hace, puesto que un aserto es un mecanismo de desarrollo, y debe ser posible eliminar ese error.

DEPURADOR

También es posible escribir asertos dentro de funciones de Objective-C, pero en tal caso es preciso utilizar la macro `NSCAssert()`, de funcionamiento análogo a la anterior. Si se produce un error de compilación del tipo "`Use of undeclared identifier _cmd`", hay que asegurarse de que se está empleando la versión correcta: `NSAssert()` dentro de métodos, y `NSCAssert()` dentro de funciones.

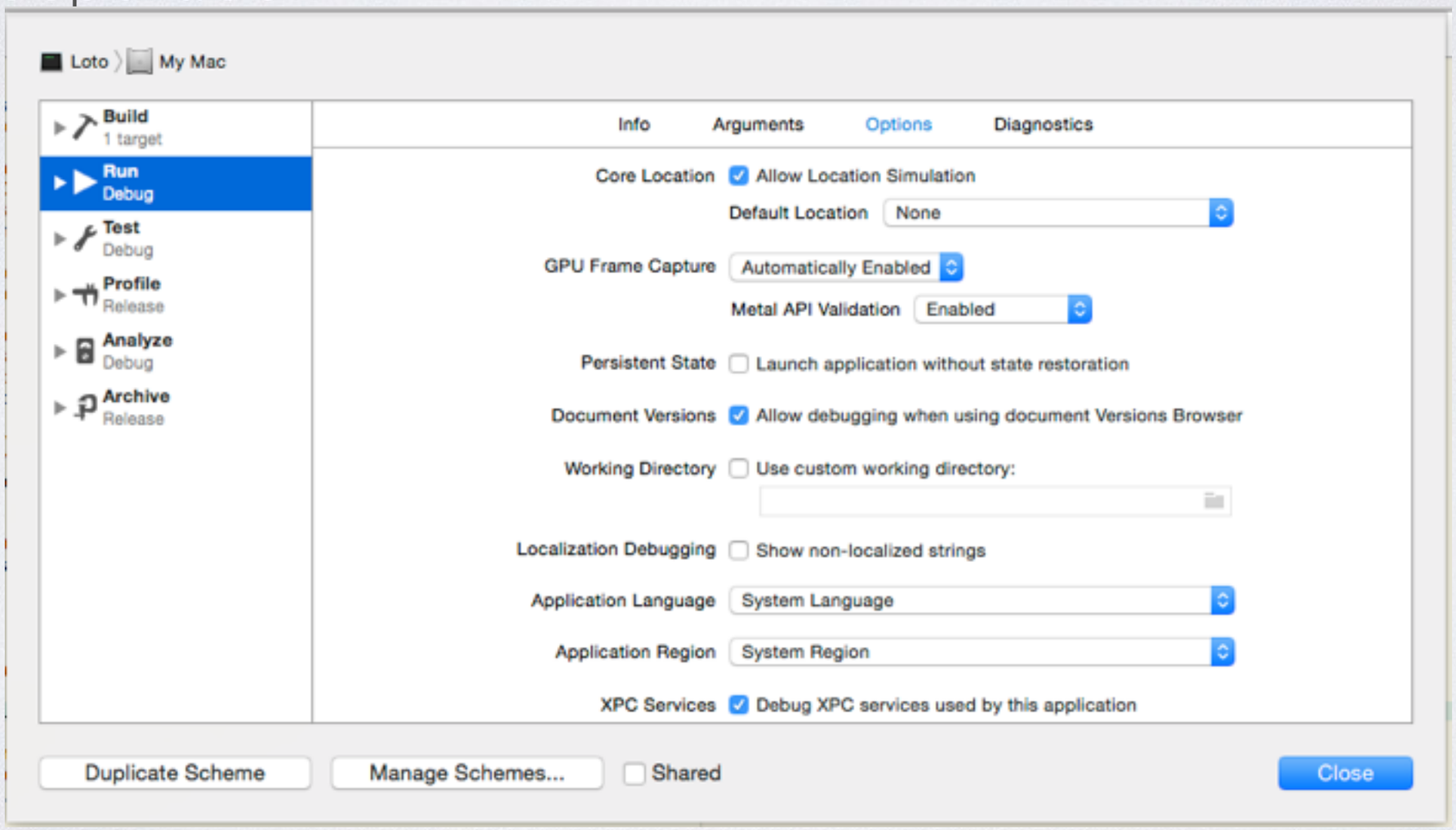
DEPURADOR

Finalmente, es posible compilar el código de tal modo que no se lancen errores debidos a asertos. Para hacer esto, se define `NS_BLOCK_ASSERTIONS` en las opciones de construcción del proyecto. Obsérvese que lo normal es ignorar los asertos solo en la versión final (`release`) y no en la de depuración



VERSIONES DEBUG Y RELEASE

El control de versiones construidas y ejecutadas (debug o release) se lleva a cabo mediante el cuadro de diálogo que aparce al pulsar ⌘-<. En el combo superior se selecciona Debug o Release, en el inferior se selecciona GDB o LLDB como depurador.



STATIC ANALYZER

El analizador estático detecta fugas de memoria. Hasta el momento no hemos tenido problema porque estaba activado el Automatic Reference Counting. Sin embargo, el problema aparecería si se desactiva, salvo que se tomen medidas (llamar a autorelease para algunas variables del programa Loto).

EL Analizador Estático (`Product->Analyze` o ⌘+⇧+B) detecta y señala las fugas de memoria, siempre que el ARC esté desactivado. Es buen ejercicio, por tanto, desactivar ARC (en `Build Options`) y analizar el código. El capítulo siguiente muestra la forma de eliminar estas fugas de memoria; de hecho explica los distintos mecanismos de administración de memoria dinámica que ofrece Objective-C. El futuro, a fecha de hoy, es ARC y no el recolector automático de basura.

MENSAJES EN OBJ-C

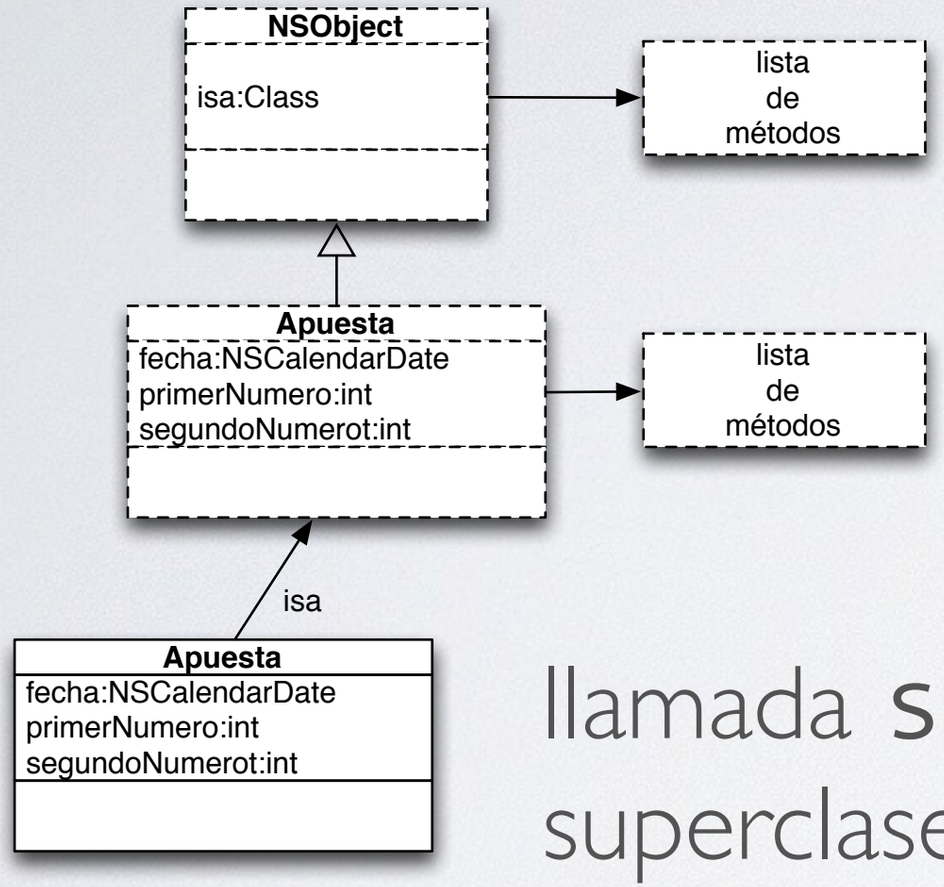
MENSAJES EN OBJECTIVE-C

Cuando se envía un mensaje a un objeto, se ejecuta el método correspondiente. Para llevar a cabo esta llamada, internamente se hace uso de una función denominada `objc_msgSend()` cuyo prototipo es el siguiente:

```
id objc_msgSend(id receptor, SEL selector, ...)
```

donde `receptor` es el puntero del objeto que contiene el método invocado, y `selector` es el índice de ese método dentro de una tabla que contiene todos los métodos del objeto señalado por `receptor`. Esta función recibe además una lista variable formada por los argumentos de los distintos métodos.

MENSAJES EN OBJECTIVE-C

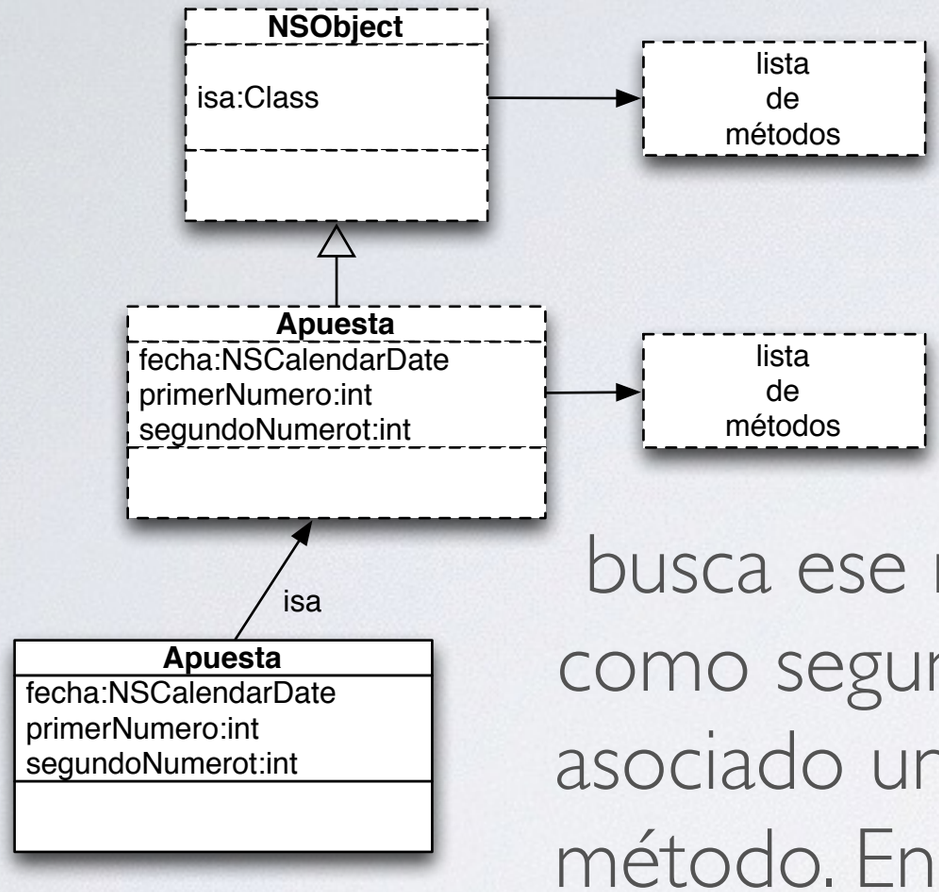


La clase `NSObject` posee una variable de ejemplar llamada `isa`, que apunta a un objeto de tipo `Class`. y otra variable de ejemplar llamada `superclass`, que apunta a la superclase (y vale `nil` en `NSObject`).

MENSAJES EN OBJECTIVE-C

Los objetos de tipo `Class` contienen como atributos un nombre de clase y un listado accesible formado por todos los atributos y métodos de la clase a la que pertenecen. Como todas las clases de Objective-C heredan de `NSObject`, en cada una de ellas hay un puntero de `Class`, que en cada caso apunta a una descripción de la clase en que se encuentra ese atributo `isa`. Esto se llama mecanismo de introspección.

MENSAJES EN OBJECTIVE-C



Cuando se hace una llamada a un método de `Apuesta`, la función `objc_msgSend()` hace uso de la variable `isa` para acceder a la lista de métodos de `Apuesta`, en la cual busca ese método empleando el selector dado como segundo argumento. Si el selector tiene asociado un método en `Apuesta`, se ejecuta ese método. En caso contrario, `objc_msgSend()` hace uso de la variable `superclass` para acceder a la superclase de `Apuesta`. Si el selector tiene asociado un método en la lista de métodos de la superclase, se ejecuta ese método. Si no hay un método asociado, se asciende a la superclase mediante `superclass`. Eventualmente, se ejecuta un método o se llega a `NSObject`, cuya `superclass` vale `nil`, y en tal caso se lanza una excepción.

PROTOCOLOS

Los protocolos son el análogos de las interfaces en Java. Se trata de clases que solo contienen métodos abstractos (sin cuerpo), y que no se pueden instanciar. Los métodos de un protocolo pueden ser obligatorios u opcionales, lo cual se indica mediante las correspondientes directrices.

Los protocolos son adoptados (implementados) por otras clases, que se comprometen a implementar todos los métodos obligatorios del protocolo.

PROTOCOLOS

PROTOCOLOS

La sintaxis de un protocolo es como sigue:

```
@protocol NombreDelProtocolo <ProtocoloAntecesor> // Herencia!!
[@required]
-(tipo)metodoRequerido1:(tipo2)argumento...;
-(tipo3)metodoRequeridoN:(tipo3)argumento...;
[@optional]
-(tipo)metodoOpcionalM:(tipo4)argumento...;
-(tipo3)metodoOpcionalP:(tipo5)argumento...;
@end
```

Los protocolos se almacenan en archivos de tipo **`.h`**, cuyo nombre debe coincidir con el nombre del protocolo.

PROTOCOLOS

Para indicar que una clase adopta un protocolo se emplea la sintaxis:

```
@class NombreDeClase <NombreDeProtocolo>
```

Normalmente, la expresión tendrá la forma

```
@class Clase : Predecesora <Prot_1, Prot_2,...>
```

Una clase puede adoptar uno o más protocolos; en tal caso está obligada a implementar todos los métodos **`@required`**, aunque no los **`@optional`**. No hay diferencia, por tanto, entre un protocolo informal (una categoría) y un protocolo formal en que todos los métodos sean **`@optional`**.

PROTOCOLOS

Un detalle más: la clase **`NSObject`** proporciona el método **`conformsToProtocol:`**, que recibe como argumento una sentencia de la forma

```
@protocol(NombreDelProtocolo)
```

El método proporciona el valor **`TRUE`** si la clase en la cual se invoca adopta el protocolo indicado como argumento. Esto es más rápido que llamar a **`respondsToSelector:`** para todos los métodos del protocolo.

EJEMPLO

Crear un protocolo dotado de métodos requeridos (**`@required`**) y opcionales (**`@optional`**). Construir una clase que adopte el protocolo. Verificar que la clase tiene que implementar todos los métodos requeridos, pero puede no implementar los métodos opcionales.

EJEMPLO

```
// Projekt: ProtokollTest
// Datei:   DasProtokoll.h
#import <Foundation/Foundation.h>

@protocol DasProtokoll <NSObject>

@required

-(void)zwingendeMethode:(NSString *)parameter;

@optional

-(void)beliebigeMethode:(NSString*)parameter;

@end
```

EJEMPLO

```
// Projekt: ProtokollTest
// Datei:   DieKlasse.h
#import <Foundation/Foundation.h>
#import "DasProtokoll.h"

@interface DieKlasse : NSObject <DasProtokoll>

@end
```

EJEMPLO

```
// Projekt: ProtokollTest
// Datei:   DieKlasse.m
#import "DieKlasse.h"

@implementation DieKlasse

-(void)zwingendeMethode:(NSString*)parameter
{
  NSLog(@"Der Parameter der zwingenden Methode ist: \"%@\"",
parameter);
}

-(void)beliebigeMethode:(NSString *)parameter
{
  NSLog(@"Der Parameter der beliebigen Methode ist \"%@\"",
parameter);
}
```

EJEMPLO

```
// Projekt: ProtokollTest
// Datei:   main.m
#import <Foundation/Foundation.h>
#import "DieKlasse.h"

int main(int argc, const char * argv[])
{
  @autoreleasepool {

    DieKlasse * k = [[DieKlasse alloc] init];
    [k zwingendeMethode:@"Required method - this is the parameter"];
    [k beliebigeMethode:@"Optional method - this is the parameter"];

  }
    return 0;
}
```

CATEGORÍAS O EXTENSIONES

CATEGORÍAS

Las categorías son un mecanismo que permite añadir métodos a una clase sin necesidad de conocer su código fuente. Los métodos añadidos mediante categorías son métodos de pleno derecho, y tienen el mismo acceso que los métodos normales a los atributos de la clase.

Las categorías son un método frecuente de extensión de clases, que se usa con frecuencia en Cocoa, a menudo sin que el usuario sea consciente de que emplea métodos de una categoría. Por ejemplo, la clase NSString tiene varias categorías adicionales

EJEMPLO

Crear una clase, añadirle una categoría y verificar su correcto funcionamiento. La clase tiene un atributo privado; comprobar que un método añadido en la categoría tiene acceso directo a ese atributo.

Proyecto: **`TestCategories`**

EJEMPLO

```
// proyecto TestCategories
// archivo Main-Class.h
#import <Foundation/Foundation.h>

@interface MainClass : NSObject
{
    @private NSString * test;
}
@end
```

EJEMPLO

```
// proyecto TestCategories
// archivo Main-Class.m
@implementation MainClass

-(id)init
{
     self = [super init];
     if (self) {
          test = @"Hallo, Welt";

     }
     return self;
}
@end
```

EJEMPLO

```
// proyecto TestCategories
// archivo MainClass-Methods.h

#import <Foundation/Foundation.h>
#import "MainClass.h"

@interface MainClass (Methods)

-(void)showTest;

@end
```

EJEMPLO

```
// proyecto TestCategories
// archivo MainClass-Methods.m

#import "MainClass-Methods.h"

@implementation MainClass ( Methods )

-(void)showTest
{
    NSLog(@"%@", test);
}
@end
```

EJEMPLO

```
// proyecto TestCategories
// archivo main.m
#import <Foundation/Foundation.h>
#import "MainClass.h"
#import "MainClass-Methods.h"

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        MainClass * m = [[MainClass alloc]  init];

        [m showTest];
    }
    return 0;
}
```

Demo

Proyecto: `TestCategories`

PROPUESTO

Se dispone de una clase, llamada ListaDeAlimentos, que implementa el protocolo BasicXML, definido en la forma que puede verse a continuación:

```
#import <Cocoa/Cocoa.h>


@protocol BasicXML

@required
-(BOOL)readFromXML:(NSString *)theXMLfile;
-(BOOL)writeToXML:(NSString *)theXMLfile;
@optional
-(void)makeAllKeysUppercase:(BOOL)yesorno;
@end
```

PROPUESTO

Se pide construir una categoría de la forma siguiente:

```
// Categoría solicitada
#import <Cocoa/Cocoa.h>
#import "ListaDeAlimentos.h"

@interface ListaDeAlimentos(ES)

 -(BOOL)readListFrom:(NSString*)theDataFile;
 -(BOOL)writeListTo:(NSString*)theDataFile;
  -(NSString*)desktopPath; // Esto viene bien

@end
```

PROPUESTO

El objetivo de los métodos de esta categoría es añadir a los ejemplares de la clase **`ListaDeAlimentos`** la capacidad de almacenarse en disco con formato delimitado por asteriscos. Una vez construida la categoría, se pide mostrar su correcto funcionamiento mediante un programa principal dotado de las siguientes capacidades:

- Crear un ejemplar de **`ListaDeAlimentos`** con varios registros.
- Guardar la lista en disco con formato delimitado por asteriscos
- Leer la lista de alimentos (a partir del archivo creado).

BLOQUES

BLOQUES

Un bloque es un objeto que contiene código ejecutable. Como tal objeto, pertenece a un tipo, y se trata de un tipo definido por el usuario.
Los bloques pueden ser anónimos, esto es, se puede crear un bloque sin nombre, que se pasará como argumento a un métodos.
También es posible crear (mediante typedef) un tipo de bloque. Entonces se pueden declarar variables de ese tipo, que igualmente se pueden pasar a un bloque, o ejecutar directamente como si fueran punteros de función.

BLOQUES

Los bloques pueden tener o no argumentos, y devolver o no un resultado. La declaración de argumentos de bloque es análoga a la declaración de argumentos de función, y lo mismo ocurre con la especificación del tipo devuelto por el bloque.

BLOQUES

Cuando se declara un bloque, el bloque captura las variables locales de su ámbito en forma de constantes. Por tanto, se pueden usar esas variables en el cuerpo del bloque, aunque no se puedan modificar.
Si se declara un bloque globalmente, puede acceder (como constantes en principio) a las variables globales.

BLOQUES

Si se declara un bloque en un método, ese bloque puede hacer uso de las variables locales del método. Esas variables locales se tratan como constantes.
Es posible utilizar la directriz **`__block`** para modificar el comportamiento ciertas variables; de hacerse así, el código del bloque puede modificar los elementos marcados con **`__block`**, que dejan de comportarse como constantes.

Crear un bloque que no reciba argumentos y ni produzca resultados. Ejecutar el bloque creado.

```
#import <Foundation/Foundation.h>

int main(int argc, const char * argv[])
{

  @autoreleasepool {

    int numPitufos = 7;
    __block int numPitufas = 0;

    void (^bloque)(void);

    bloque = ^{
        NSLog(@"Mensaje hallado en un bloque");
      };

    bloque();

    bloque = ^{
      NSLog(@"El número de pitufos es de %d", numPitufos);
        //numPitufos = 14; // No se admite
      numPitufas = 1; // Se admite porque la variables está marcada como __block
    };

    NSLog(@"Número inicial de pitufas: %d", numPitufas);
      // El bloque ve las variables locales
    bloque();
    NSLog(@"Número final de pitufas : %d", numPitufas);

    NSLog(@"End of program");
  }
    return 0;
}
```

EJEMPLO

Crear un bloque que reciba argumentos y produzca resultados. El bloque debe mostrar los argumentos recibidos.

```
// proyecto: bloques_102
#import <Foundation/Foundation.h>

int main(int argc, const char * argv[])
{
  @autoreleasepool {

    int (^bloque)(int, float, NSString*);

    bloque = ^(int num1, float num2, NSString * cadena){
      NSLog(@"Ejecutando desde un bloque");
      NSLog(@"Argumento num1:   %d", num1);
      NSLog(@"Argumento num2:   %f", num2);
      NSLog(@"Argumento cadena: %@", cadena);
    return 0;
    };

    bloque(1, 2.0, @"Ejemplo");

    NSLog(@"End of program");
  }
  return 0;
}
```

EJEMPLO

Crear una clase dotada de métodos que admitan bloques como argumentos. Uno de los métodos debe admitir bloques sin argumentos ni resultados. Otro de los métodos debe admitir bloques con argumentos y resultados. Comprobar el correcto funcionamiento de la clase, creando un ejemplar de la misma y efectuando llamadas a los métodos que admiten bloques.

EJEMPLO

```
// proyecto: MetodoConBloque
// archivo: ClaseConMetodoConBloque.h
#import <Foundation/Foundation.h>

typedef int (^TipoDeBloque)(int, float, NSString *);

@interface ClaseConMetodoConBloque : NSObject

@property int integerAttribute;
@property float floatAttribute;
@property NSString * stringAttribute;

-(void)metodoConBloque:(TipoDeBloque)unBloque;
-(void)metodoConOtroBloque:(  float (^)(int, float, NSString
*) )otroBloque;

@end
```

EJEMPLO

```
// proyecto: MetodoConBloque
// archivo: ClaseConMetodoConBloque
#import "ClaseConMetodoConBloque.h"

@implementation ClaseConMetodoConBloque

- (id)init
{
    self = [super init];
    if (self) {
        _integerAttribute = 333;
        _floatAttribute = 444.44;
        _stringAttribute = @"Mary hat ein kleines Lamm";
    }
    return self;
}
-(void)metodoConBloque:(TipoDeBloque)unBloque {
    float m = unBloque(_integerAttribute, _floatAttribute, _stringAttribute);
    NSLog(@"Resultado de unBloque  : %f", m);
}

-(void)metodoConOtroBloque:( float (^)(int a, float b, NSString *c) )otroBloque
{
    float m = otroBloque(_integerAttribute, _floatAttribute, _stringAttribute);
    NSLog(@"Resultado de otroBloque : %f = %d*%f*%ld",
          m,
          _integerAttribute,
          _floatAttribute,
          [_stringAttribute length]);
}
@end
```

EJEMPLO

```
// proyecto: MetodoConBloque
// archivo: ClaseConMetodoConBloque
#import <Foundation/Foundation.h>
#import "ClaseConMetodoConBloque.h"

int main(int argc, const char * argv[])
{
    @autoreleasepool {

        ClaseConMetodoConBloque * ccmb = [ClaseConMetodoConBloque new];
        [ccmb metodoConBloque:^(int num1, float num2, NSString * cadena){
            NSLog(@"Ejecución de unBloque");
            NSLog(@"Argumento num1:   %d", num1);
            NSLog(@"Argumento num2:   %f", num2);
            NSLog(@"Argumento cadena: %@", cadena);
            return 0;
        }];
        [ccmb metodoConOtroBloque:^(int n1, float n2, NSString * cad){
            NSLog(@"Ejecución de otroBloque");
            return n1*n2*[cad length];
        } ];
    }
    return 0;
}
```

Demo

Proyecto: `MetodoConBloque`

EJEMPLO

Crear una clase dotada de un método que admite como argumentos uno de tipo estándar y un bloque

Comprobar el correcto funcionamiento de la clase, creando un ejemplar de la misma y efectuando llamadas al método en cuestión.

EJEMPLO

```
// proyecto: MetodoConBloque2
// archivo: ClaseConBloque.h
#import <Foundation/Foundation.h>

@interface ClaseConBloque2 : NSObject

-(void)metodo:(int)num andBlock:(NSString * (^)(int, BOOL *,
float))aBlock;

@end
```

EJEMPLO

```
// proyecto: MetodoConBloque2
// archivo: ClaseConBloque2.m
#import "ClaseConBloque2.h"

@implementation ClaseConBloque2

-(void)metodo:(int)num andBlock:(NSString * (^)(int, BOOL *, float))aBlock
{
    BOOL argument_1 = NO;
    float argument_2 = 325;
    NSString * s = aBlock(num, &argument_1,argument_2);
    if(argument_1)
        NSLog(@"Result of block: %@",s);
    else
        NSLog(@"Nothing to say");
}

@end
```

EJEMPLO

```
// proyecto: MetodoConBloque2
// archivo: main.m
#import <Foundation/Foundation.h>
#import "ClaseConBloque2.h"

int main(int argc, const char * argv[])
{

    @autoreleasepool {
        ClaseConBloque2 * cb2 = [ClaseConBloque2 new];
        [cb2 metodo:33 andBlock:^(int a, BOOL* b, float c){
            NSString * res = [NSString stringWithFormat:
                              @"arg_1:%d, arg_2:%@, arg_3:%f",
                              a,
                              b ? @"YES" : @"NO",
                              c];
            return res;
        }];

    }
    return 0;
}
```

Demo

Proyecto: `MetodoConBloque2`

EJEMPLO

Crear una clase dotada de un método que admita un bloque argumentos y proporcione una cadena como resultado. El método debe declarar variables **`__block`** y variables estándar, con objeto de mostrar que el que es posible modificar los valores de las variables **`__block`**, pero no los valores de las variables estándar.

EJEMPLO

```
//proyecto: MetodoConBloque3
//archivo: ClaseConBloque3.h
#import <Foundation/Foundation.h>

@interface ClaseConBloque3 : NSObject
@property int num1;
@property float num2;

-(void)metodoConBloque3:(NSString * (^)())bloque;

@end
```

EJEMPLO

```
//proyecto: MetodoConBloque3
//archivo: ClaseConBloque3.m
#import "ClaseConBloque3.h"

@implementation ClaseConBloque3

- (id)init
{
    self = [super init];
    if (self) {
      _num1 = 33;
      _num2 = 44;
    }
    return self;
}
-(void)metodoConBloque3:(NSString * (^)())bloque
{

    NSString * res = bloque();

    NSLog(@"Resultado del bloque: %@", res);
}

@end
```

EJEMPLO

```
//proyecto: MetodoConBloque3
//archivo: main.m
#import <Foundation/Foundation.h>
#import "ClaseConBloque3.h"

int main(int argc, const char * argv[])
{
    @autoreleasepool {
    ClaseConBloque3 * cb3 = [ClaseConBloque3 new];

    int arg1 = 33;
    __block NSString * arg2 = @"Mary hat ein kleines Lamm";
    __block float arg3 = 123.456;

    [cb3 metodoConBloque3:^(){
      NSString * res = [NSString stringWithFormat:@"arg1=%d, arg2=%@, arg3=%f",
                        arg1, arg2, arg3];
        // arg1 = 44;           // No se puede; como no tiene __block, es una constante
      arg2 = @"Tócame roque"; // Sin problemas, tiene __block
      arg3 = 567.890;         // Sin problemas, tiene __block
      cb3.num1 = 324;         // Se puede acceder desde el bloque a los atributos de la clase
      cb3.num2 = 8988.77f;    // porque cb3 está en el ámbito de declaración del bloque
      return res;
    }];
    NSLog(@"Después de llamar al bloque, arg1=%d, arg2=%@, arg3=%f",
          arg1, arg2, arg3);
    NSLog(@"Después de la llamada al bloque, num1 = %d, num2 = %f",
          cb3.num1, cb3.num2);
  }
  return 0;
}
```

BIBLIOTECAS

BIBLIOTECAS

- Es posible compilar por anticipado una clase, produciendo un archivo de código objeto, con la intención de disponer de esta clase en formato **.o**.

- Una biblioteca es una colección optimizada de archivos **.o**.

- Para utilizar la clase compilada, sin necesidad de disponer de su código fuente, bastará incluir el **.h** de la clase en el archivo de encabezado de la clase cliente.

- Además, es preciso indicar al entorno que se desea hacer uso de una biblioteca.

EJEMPLO

- Vamos a construir una clase que contiene algunos métodos adecuados para leer de teclado con comodidad en programas escritos en Objetive-C.

- Vamos a escribir el siguiente código fuente en un archivo de encabezado y otro de implementación.

- Este ejemplo se propuso hace algún tiempo como ejercicio

EJEMPLO

```
//  RGBTextInput.h
//  RGBTextInput
//
//  Created by bruegel on 22/09/13.
//  Copyright (c) 2013 bruegel. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface RGBTextInput : NSObject

+(NSString *)readStringWithPrompt:(NSString*)prompt;
+(int)readIntWithPrompt:(NSString*)prompt;
+(float)readFloatWithPrompt:(NSString*)prompt;
+(double)readDoubleWithPrompt:(NSString*)prompt;

@end
```

EJEMPLO

```
//  RGBTextInput.m
//  RGBTextInput
//
//  Created by bruegel on 22/09/13.
//  Copyright (c) 2013 bruegel. All rights reserved.
//
//22 de septiembre de 2013 13:06:18 GMT+02:00

#import "RGBTextInput.h"

@implementation RGBTextInput

+(NSString *)readStringWithPrompt:(NSString*)prompt
{
    size_t availableBytes = 0;
    printf("%s : ", [prompt cStringUsingEncoding:NSUTF8StringEncoding]);
    char * pointerToText = fgetln(stdin, &availableBytes);/* No leftovers */
    char * cString = malloc(availableBytes+2);
    memcpy(cString,pointerToText,availableBytes);
    cString[availableBytes-1] = '\0';
    NSString * string =[NSString stringWithCString:cString
                                          encoding:NSUTF8StringEncoding];
    free(cString);
    return string;
}
```

EJEMPLO

```
+(int)readIntWithPrompt:(NSString*)prompt
{
    NSString * temp = [self readStringWithPrompt:prompt];
    return [temp intValue];

}

+(float)readFloatWithPrompt:(NSString*)prompt
{
    NSString * temp = [self readStringWithPrompt:prompt];
    return [temp floatValue];

}

+(double)readDoubleWithPrompt:(NSString*)prompt
{
    NSString * temp = [self readStringWithPrompt:prompt];
    return [temp doubleValue];

}

@end
```

EJEMPLO

- Una vez escritos los dos archivos anteriores, vamos a compilarlos. La ubicación es importante, porque después se invitará a Xcode donde se encuentra el archivo de código objeto, una vez compilado este.

- Para compilar este archivo, se utiliza la expresión siguiente:

- `clang -c RGBTextInput.m`

EJEMPLO

- También se puede utilizar esta otra expresión, que da lugar a un código objeto optimizado.

- `clang -O3 -c RGBTextInput.m`

EJEMPLO

- El resultado de utilizar estas dos formas de compilación puede verse en el listado siguiente. Como se observará, la compilación optimizada da lugar a un código objeto más pequeño, y de hecho más rápido.

EJEMPLO

```
ursaminor:~ bruegel$ cd /Users/bruegel/com/coti/objc
ursaminor:objc bruegel$ ls -al
total 32
drwxr-xr-x  5 bruegel  staff   170 22 sep 13:06 .
drwxr-xr-x  5 bruegel  staff   170 22 sep 12:51 ..
-rw-r--r--@ 1 bruegel  staff   404 22 sep 12:57 RGBTextInput.h
-rw-r--r--@ 1 bruegel  staff  1211 22 sep 13:06 RGBTextInput.m
-rw-r--r--  1 bruegel  staff  5108 22 sep 13:06 RGBTextInput.o
ursaminor:objc bruegel$ clang -c RGBTextInput.m
ursaminor:objc bruegel$ clang -O3 -c RGBTextInput.m
ursaminor:objc bruegel$ ls -al
total 32
drwxr-xr-x  5 bruegel  staff   170 22 sep 18:38 .
drwxr-xr-x  5 bruegel  staff   170 22 sep 12:51 ..
-rw-r--r--@ 1 bruegel  staff   404 22 sep 12:57 RGBTextInput.h
-rw-r--r--@ 1 bruegel  staff  1211 22 sep 13:06 RGBTextInput.m
-rw-r--r--  1 bruegel  staff  4640 22 sep 18:38 RGBTextInput.o
ursaminor:objc bruegel$
```

EJEMPLO

A continuación, vamos a crear una pequeña herramienta que hace uso de los métodos de esta clase con objeto de simplificar la entrada de texto a través de la consola en un programa escrito en Objective-C

EJEMPLO

En primer lugar, se crea una clase que reproduce los atributos de un disco duro. Éstos atributos son propiedades y tienen distintos tipos, pertenecientes a los ya conocidos en Objective-C. Consiguientemente, es preciso leer de teclado variables de todos estos tipos. El ejercicio se resolvió anteriormente de forma relativamente incómoda. Ahora será más breve y sencillo.

EJEMPLO

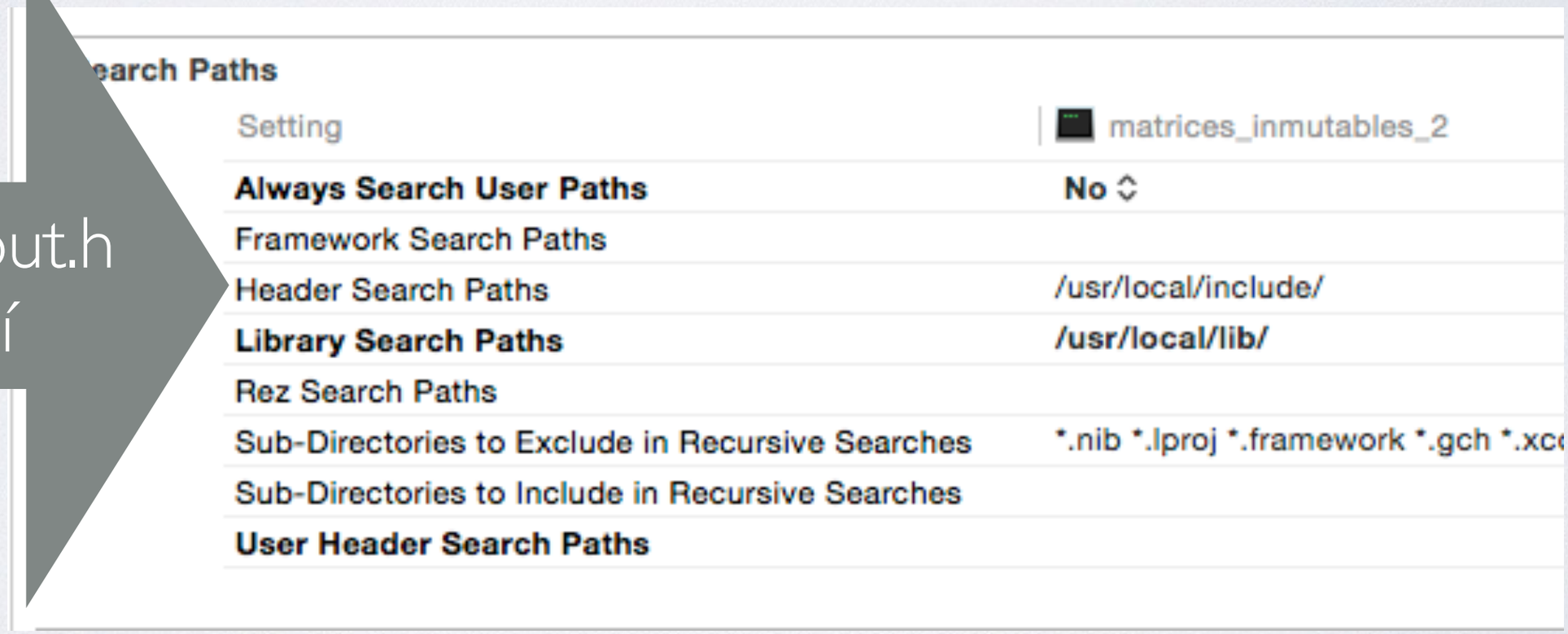
```
// Proyecto: matrices_inmutables2
// Archivo: DieFestplatte.h
#import <Foundation/Foundation.h>
   #import "RGBTextInput.h"

#import <Foundation/Foundation.h>
#import "RGBTextInput.h"

@interface DieFestplatte : NSObject

@property (readwrite, copy) NSString * fabricante, *modelo, *tecnologia;
@property float megabytes;
@property int velocidadDeRotacion, tasaDeTransmision;
@property (readwrite, copy) NSString * tipoDeInterface, *numeroDeSerie;

-(void)readFromKeyboard;

@end
```

Como puede observarse, se lee el archivo de encabezado de la clase RGBTextInput empleando simplemente el nombre del archivo. Esto es posible porque se ha indicado a Xcode que busque en un determinado directorio.

EJEMPLO

Se lee el archivo de encabezado de la clase `RGBTextInput` empleando simplemente el nombre del archivo. Esto es posible porque se ha indicado a Xcode que busque en los directorios estándar de Unix. Esta es la forma correcta de hacerlo.

RGBTextInput.h está aquí



EJEMPLO

Véase a continuación el archivo de implementación de la clase que representa un disco duro. Como se apreciará, se hace uso de forma intensiva de uno de los métodos almacenados en la clase de cuyo código objeto se dispone.

EJEMPLO

```
// Proyecto: matrices_inmutables2
// Archivo: DieFestplatte.m
#import "DieFestplatte.h"
@implementation DieFestplatte

-(void)readFromKeyboard
{
    self.fabricante = [RGBTextInput readStringWithPrompt:@"Escriba el fabricante"];
    self.modelo     = [RGBTextInput readStringWithPrompt:@"Escriba el modelo    "];
    self.tecnologia = [RGBTextInput readStringWithPrompt:@"Escriba la tecnología"];
    /* Peligro... el locale nos puede dar un susto */
    self.megabytes           = [RGBTextInput readFloatWithPrompt: @"Escriba la capacidad en megabytes"];
    self.velocidadDeRotacion = [RGBTextInput readIntWithPrompt:   @"Escriba la velocidad de rotación "];
    self.tasaDeTransmision   = [RGBTextInput readIntWithPrompt:   @"Escriba la tasa de transmisión   "];
    self.tipoDeInterface     = [RGBTextInput readStringWithPrompt:@"Escriba el tipo de interface     "];
    self.numeroDeSerie       = [RGBTextInput readStringWithPrompt:@"Escriba el número de serie       "];
}

-(NSString*)description
{
    return [NSString stringWithFormat:@"{%@,%@,%@,%f,%d,%d,%@,%@}",
            _fabricante,
            _modelo,
            _tecnologia,
            _megabytes,
            _velocidadDeRotacion,
            _tasaDeTransmision,
            _tipoDeInterface,
            _numeroDeSerie
            ];
}

@end
```

EJEMPLO

```
-(NSString*)description
{
    return [NSString stringWithFormat:@"{%@,%@,%@,%f,%d,%d,%@,%@}",
                _fabricante,
                _modelo,
                _tecnologia,
                _megabytes,
                _velocidadDeRotacion,
                _tasaDeTransmision,
                _tipoDeInterface,
                _numeroDeSerie
                ];
}

@end
```

EJEMPLO

Por último, vamos a estudiar el programa principal que hace uso de un ejemplar de la clase creada. Como puede observarse es muy breve debido al uso de múltiples métodos creados en las clases anteriores.

EJEMPLO

```
#import <Foundation/Foundation.h>

    #import "DieFestplatte.h"

    int main(int argc, const char * argv[])
    {

        @autoreleasepool {
            DieFestplatte * f = [DieFestplatte new];
            [f readFromKeyboard]; // Hace uso de RGBTextInput
            NSLog(@"Características: %@", f);
            NSArray * array = @[@"Hallo, Welt!",
                                [NSNumber numberWithBool:YES],
                                [NSNumber numberWithFloat:3.14f],
                                f];
            NSLog(@"La matriz contiene: %@", array);

        }
        return 0;
}
```

EJEMPLO

En cuanto al resultado obtenido al ejecutar este programa, puede verse también a continuación.

EJEMPLO

```
Escriba el fabricante
OWC
Escriba el modelo
Electra
Escriba la tecnología
SSD
Escriba la capacidad en megabytes
512
Escriba la velocidad de rotación
0
Escriba la tasa de transmisión
550
Escriba el tipo de interface
Sata 6
Escriba el número de serie
341241234
Características: {OWC,Electra,SSD,512.000000,0,550,Sata 6,341241234}
La matriz contiene: (
    "Hallo, Welt!",
    1,
    "3.14",
    "{OWC,Electra,SSD,512.000000,0,550,Sata 6,341241234}"
)
```

EJEMPLO

- Ha sido preciso indicar a Xcode la situación del archivo de encabezado.

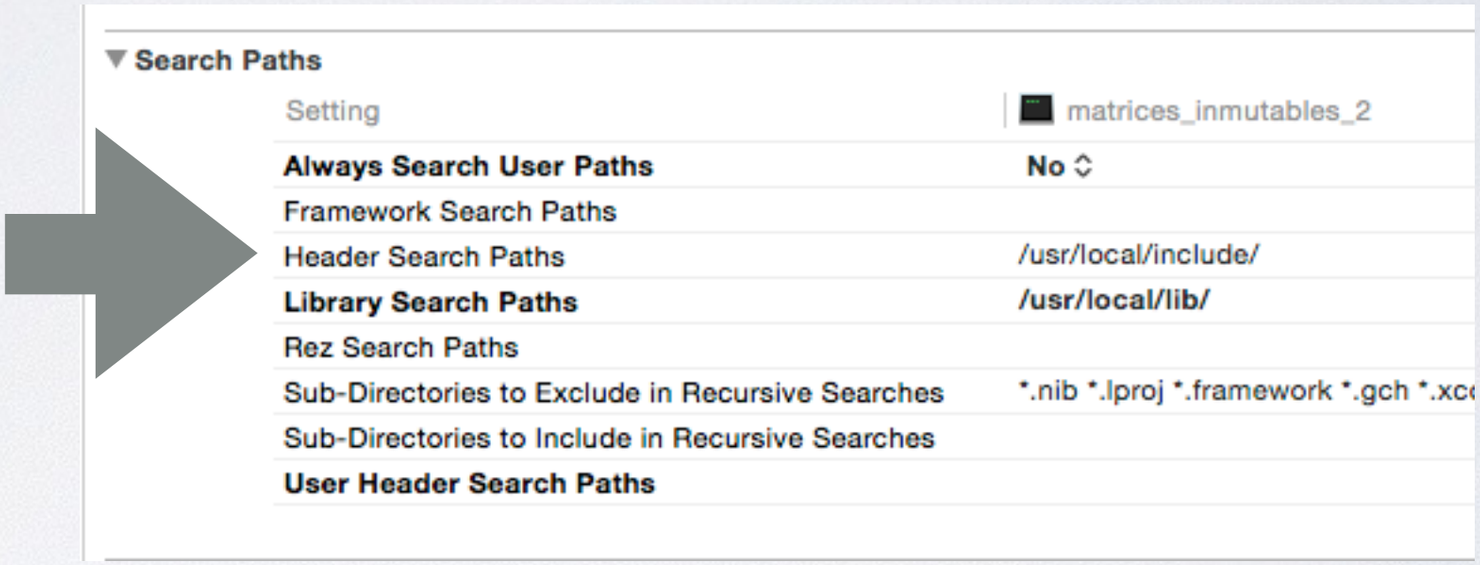
- También ha sido preciso indicar a Xcode que debe enlazar este programa con una biblioteca.

- La transparencia siguiente muestra el lugar del entorno en que se indica esta información a Xcode.

MECANISMO

El archivo `RGBTextInput.h` debe estar situado en el directorio `/usr/local/include/`

¿DONDE ESTÁ RGBTEXTINPUT.H?

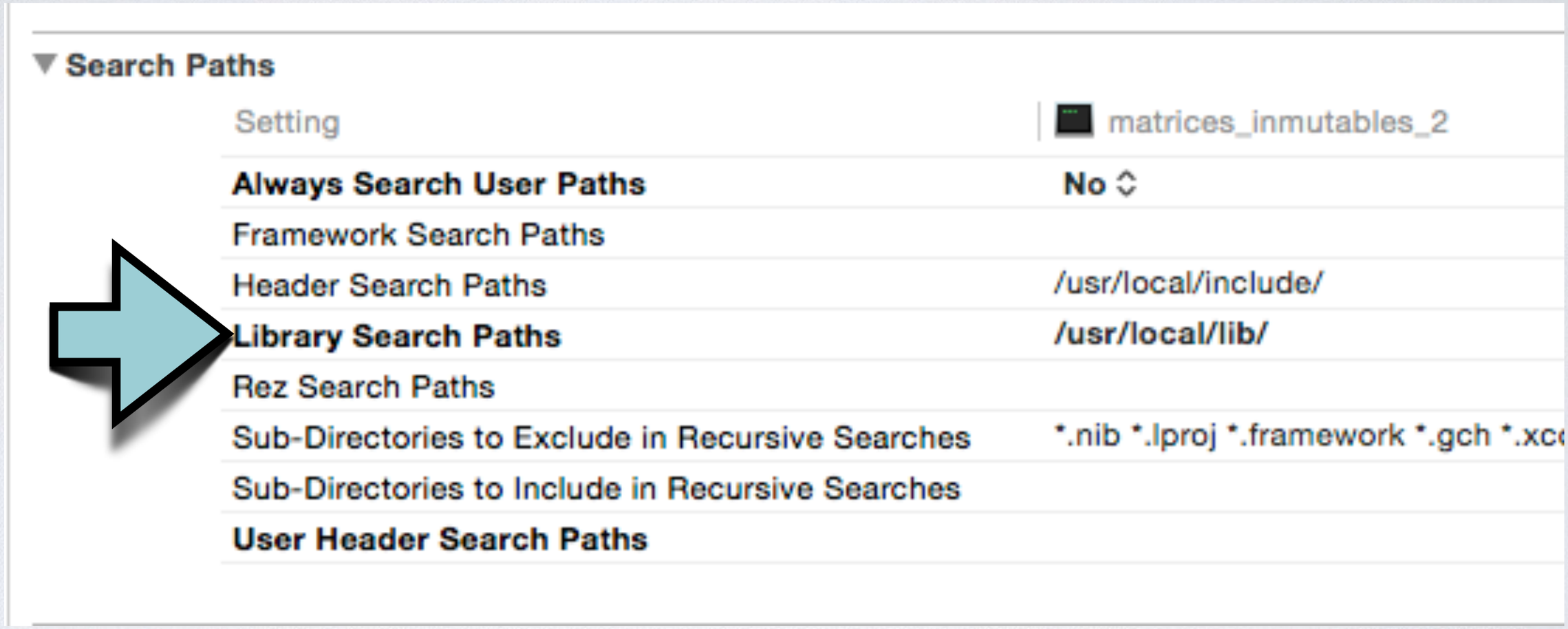


¿DONDE ESTÁ LIBUTILES.A?

El archivo **`libutiles.a`**, que se genera mediante un guión, debe estar situado en el directorio **`/usr/local/lib/`**.

Para indicar a Xcode su ubicación, se escribe esa ruta como puede ver en la transparencia siguiente.

¿DONDE ESTÁ LIBUTILES.A?



BIBLIOTECAS

- Los archivos **.a** pueden contener varios archivos de código objeto (**.o**) en su interior.

- Es es preciso un paso previo para transformar varios archivos de código objeto en un único archivo .a.

- Tanto los archivos de encabezado de los distintos archivos de código objeto como la biblioteca construida a partir de ellos deben situarse en directorios determinados, que después se harán conocidos para Xcode. Véase el guión **compile_and_move.sh**.

EJEMPLO

Se dispone de dos clases que aportan diferentes métodos. La primera clase es la ya conocida **`RGBTextInput`**. La segunda es una nueva clase destinada a la lectura y escritura de matrices, denominada **`RGBArrayIO`**. Su interfaz se muestra en la página siguiente. Evidentemente, se trata tan sólo de un ejemplo, que servirá en un futuro para desarrollar una clase más real.

Estas dos clases se van a ejercitar en un proyecto llamado **`TestUtils`**, que lee y escribe matrices en pantalla empleando las dos clases anteriores.

EJEMPLO

```
// proyecto:TestUtils
// archivo: RGBArrayIO.h
#import <Foundation/Foundation.h>

@interface RGBArrayIO : NSObject

+(float *)inputArrayWithRows:(int)rows
                  andColumns:(int)columns;

+(void)outputArray:(float*)array
          withRows:(int)rows
        andColumns:(int)columns;

@end
```

TESTUTILS

La implementación de esta clase consta únicamente de los dos métodos indicados, pero en un futuro se añadirán otros que permitan utilizar matrices con más flexibilidad.

Como puede verse como estos métodos están preparados para funcionar con cualquier número de filas y columnas.

Como curiosidad, este método muestra la razón por la cual se insistía tanto en los primeros cursos de programación en el uso de la aritmética de punteros.

EJEMPLO

```
// proyecto:TestUtils
// archivo: RGBArrayIO.m
#import "RGBArrayIO.h"
#import "RGBTextInput.h"

@implementation RGBArrayIO

+(float *)inputArrayWithRows:(int)rows andColumns:(int)columns
{
   float * array = malloc(rows*columns*sizeof(float));
   for(int i=0; i < rows; i++)
    for(int j=0; j < columns; j++)
        array[i+ j*columns] = [RGBTextInput readFloatWithPrompt:[NSString
stringWithFormat:@"elemento[%d][%d] = ", i, j]];
   return array;
} //inputArrayWithRows:andColumns:

+(void)outputArray:(float*)array withRows:(int)rows andColumns:(int)columns
{
   for(int i=0;i<rows; i++)
   {
    printf("| ");
    for(int j=0; j<columns; j++)
        printf("%8.2f", array[i + j*columns]);
    printf(" |\n");
   }
}// outputArray:withRows:andColumns:

@end
```

EJEMPLO

El programa principal comienza por leer un número entero empleando uno de los métodos creados en la clase anterior.

Sería sencillo leer dos números y utilizar estos números como números de filas y columnas de la matriz que se crea a continuación. Véase **`TestUtils2`**.

Para crear una matriz, se emplea un método de la segunda clase. Una vez creada la matriz, se escribe en pantalla utilizando un segundo método.

Finalmente, se libera la memoria reservada para la matriz. De este modo, la matriz se destruye una vez utilizada, y no hay fugas de memoria

EJEMPLO

```
//proyecto: TestUtils
//archivo: main.m
#import <Foundation/Foundation.h>

#import "RGBTextInput.h"
#import "RGBArrayIO.h"

int main(int argc, char * argv[]) {

  @autoreleasepool {

        int n = [RGBTextInput readIntWithPrompt:@"Escriba un número"];
        printf("El número es %d\n", n);

        float * p = [RGBArrayIO inputArrayWithRows:2 andColumns:2];
        [RGBArrayIO outputArray:p withRows:2 andColumns:2];
        free(p);// No hay fugas

  }

  return 0;
}
```

EJEMPLO

Una segunda aplicación de estas dos clases es TestUtils2, que puede verse la transparencia siguiente.

Nos hemos limitado a leer dos números enteros, de los cuales uno representa el número de filas, y el otro el número de columnas.

De esta manera, se puede crear una matriz con cualquier número de filas y columnas. Por ejemplo, se puede probar introduciendo cuatro filas y cuatro columnas. Es fácil construir de este modo la matriz identidad.

EJEMPLO

```
//proyecto: TestUtils2
//archivo: main.m
#import <Foundation/Foundation.h>

#import "RGBTextInput.h"
#import "RGBArrayIO.h"

int main(int argc, char * argv[]) {

    @autoreleasepool {

        int numFilas = [RGBTextInput readIntWithPrompt:@"Escriba el número de filas"];
        int numCols = [RGBTextInput readIntWithPrompt:@"Escriba el número de columnas"];

        float * p = [RGBArrayIO inputArrayWithRows:numFilas andColumns:numCols];
        [RGBArrayIO outputArray:p withRows:numFilas andColumns:numCols];
        free(p);

    }

    return 0;
}
```

- El resultado de la ejecución se ve en la página siguiente:

```
Escriba el número de filas3
Escriba el número de columnas4
elemento[0][0] = 1
elemento[0][1] = 2
elemento[0][2] = 3
elemento[0][3] = 4
elemento[1][0] = 1
elemento[1][1] = 2
elemento[1][2] = 3
elemento[1][3] = 4
elemento[2][0] = 1
elemento[2][1] = 2
elemento[2][2] = 3
elemento[2][3] = 4
|     1.00    2.00    3.00    4.00 |
|     1.00    2.00    3.00    4.00 |
|     1.00    2.00    3.00    4.00 |
```

MECANISMO

En los dos últimos ejemplos se ha empleado el método ya escrito para utilizar clases de las cuales solamente se conoce el encabezado, y se dispone de un archivo de tipo **.a**.

Para indicar a Xcode que debe buscar la biblioteca en la ruta

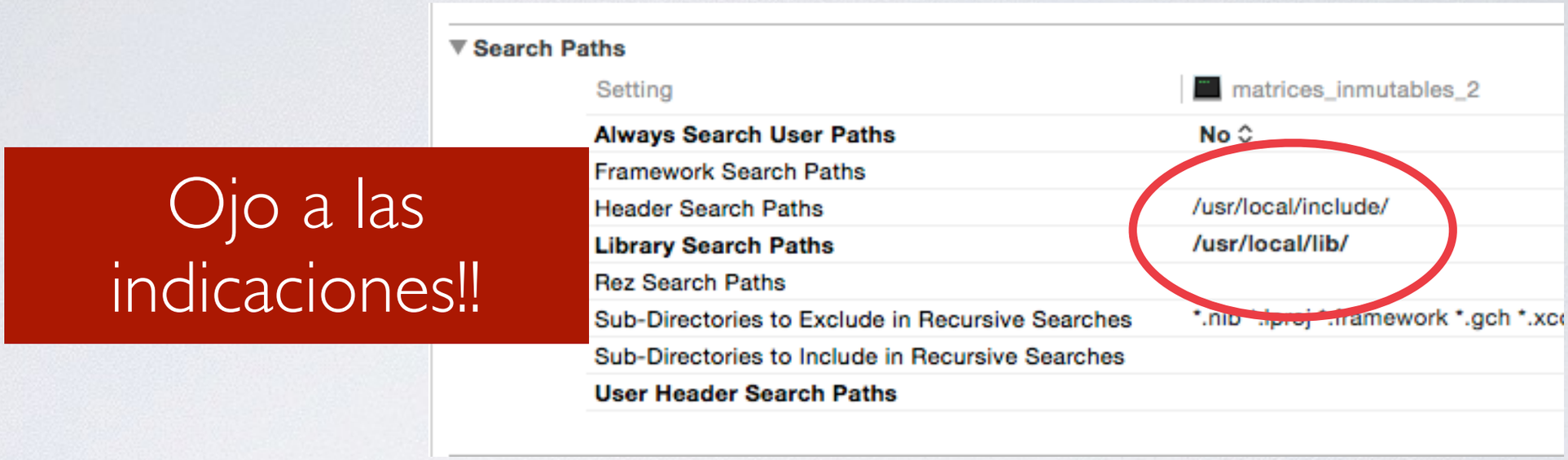
`/usr/local/lib/`

y los encabezados en la ruta

`/usr/local/include/`

se han especificado ambas rutas en la información propia del proyecto. Véase la página siguiente.

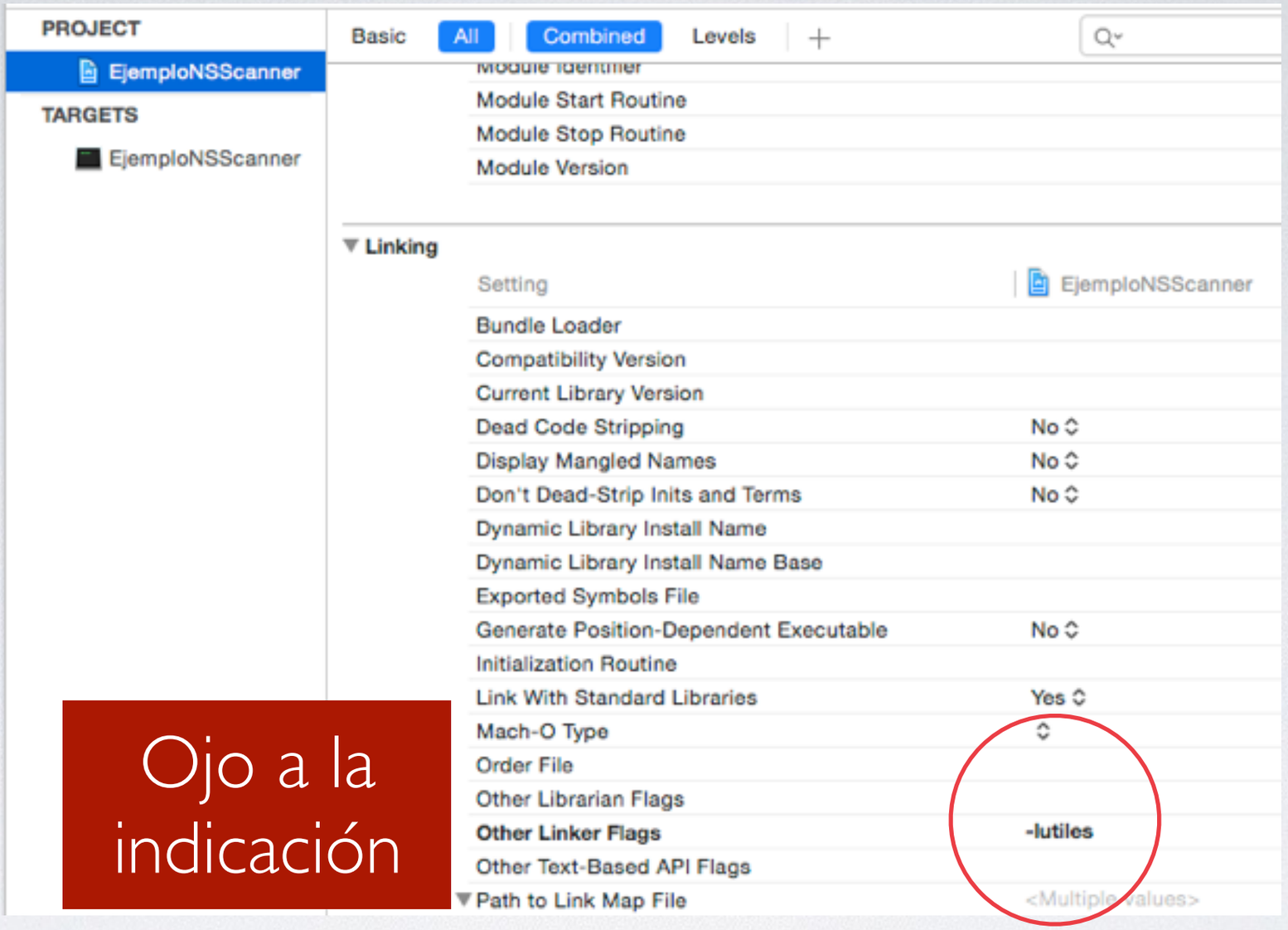
EJEMPLO



Ojo a las indicaciones!!

Evidentemente, es preciso indicar en el código fuente el nombre del archivo de encabezado, incluyendo el nombre del directorio en que se encuentra dentro del subdirectorio de inclusión. Éstas son precisamente las dos primeras líneas del programa principal.

```
#import "RGBTextInput.h"
#import "RGBArrayIO.h"
```



Por último, para indicar al compilador que debe enlazar junto con el código objeto la biblioteca **utiles**, se emplea la línea que puede verse en negrita en la imagen, dentro del epígrafe.

Como se recordará, esta biblioteca se llama **libutiles.a**, y ha sido creada previamente, empleando la instrucción `ar -rcs libutiles.a *.o`

COMENTARIOS FINALES

Puede resultar útil crear un pequeño shellscript que efectúe todas las compilaciones y traslados oportunos. Véase **`compile_and_move.sh`**.

NSSCANNER

NSSCANNER

```
// proyecto: EjemploNSSCanner

#import <Foundation/Foundation.h>
#import "RGBTextInput.h"

int main(int argc, const char * argv[])
{
@autoreleasepool {

    float numero;
    bool goodResult = true;
    NSLocale * actualLocale = [NSLocale autoupdatingCurrentLocale];
    NSString * decimalSeparator = [actualLocale
                                   objectForKey:NSLocaleDecimalSeparator];
    printf("\nEl separador de decimales es %s\n\n",
           [decimalSeparator cStringUsingEncoding:NSUTF8StringEncoding]);
    NSScanner * sc = nil;
    NSString * cadena = nil;
```

NSSCANNER

```
do {
    cadena = [RGBTextInput readStringWithPrompt:@"Escriba un número "];
    sc = [NSScanner localizedScannerWithString:cadena];
    goodResult = [sc scanFloat:&numero];
    if (!goodResult) {
        printf("\nPerdón, \"%s\" no es un número.\n\n",
               [cadena cStringUsingEncoding:NSUTF8StringEncoding]);
        continue;
    }
    if([decimalSeparator isEqualToString:@","])
    {
        NSRange r = [cadena rangeOfString:@"."];
        if (r.location != NSNotFound) {
            printf("\n***Número incompleto: se ha detectado"
                   "un separador decimal (\".\") incorrecto.\n\n");
        }
    }
    else
    {
        NSRange r = [cadena rangeOfString:@","];
        if (r.location != NSNotFound) {
            printf("\n***Número incompleto: se ha detectado"
                   "un separador decimal (\",\") incorrecto.\n\n");
        }
    }
} while (!goodResult);
printf("El número leído es: %f\n", numero);

}
return 0;
}
```

Demo

Proyecto: `EjemploNSScanner`

OBJECTIVE-C RECORDATORIO Y DETALLES ADICIONALES

Desarrollo de Aplicaciones Avanzadas 2015-2016

Universidad de Salamanca
Grado en Ingeniería Informática
Dr. J.R. García-Bermejo Giner