Object Representation in Objective-C

CS 274 iPhone App Development
Robert Muller

Today

• The Objective-C Compilation and Execution Models
• Pointers
• Objective-C Data Representation

What is in an iPhone 3GS?

• Samsung S5PC100 System on a Chip
  — CPU: ARM Cortex A8, 32-bit
  — GPU: PowerVR SGX
  — 32KB L1, 256KB L2 Caches
  — 600MHz Clock
  — Very Low Power
• 256MB RAM
• 16 or 32GB Persistent Storage, Solid-state Flash Drive, NO DISK
Xcode Objective-C Compiler
Clang/LLVM

@interface
Point {
...
@property
--
}@interface

Clang/LLVM
OBJECTIVE-C
COMPILER

Move:
LDA $0, X
ADD $1,$2,$3
BEC Done
MOV $1,$0
MUL $1,$0,$2
JSR
Print
MOV $0,$0
--

Xcode Compiler

ARM object code
transferred to the
iPhone’s flash drive.
Pointers

Random Access Memory

Addresses

Contents
Example

- UILabel *myLabel = [[UILabel alloc] init];
When an App is Loaded into RAM from the Flash Drive
Values with **indefinite extent**. I.e., values whose lifetimes aren’t bounded by the invocation of a procedure or method.

After such values are created, they may live until program exit.

For example, objects are almost always allocated in the heap.

**Object Representation**
Representations of Objects in Java and Obj-C

UILabel, E.g.,

```objective-c
@interface UILabel : UIView {
    NSString *text;
    NSTextAlignment textAlignment;  // NB: No *!
    ...}
@property(nonatomic, copy) NSString *text;
@property(nonatomic) NSTextAlignment textAlignment;
...-
-(void)drawTextInRect:(CGRect)rect;
-(CGRect)textRectForBounds:(CGRect)bounds
    limitedToNumberOfLines:(NSInteger)numberOfLines;
...-
@end;
```

Run-Time Representation of the UILabel Class

```objective-c
superclass:

Code for drawTextInRect

textRectForBounds:limited:

Code for textRectFor...
```
In many languages, (e.g., LISP, CAML, Java, Python, ...) memory in the heap is allocated and reclaimed automatically using garbage collection.

The iPhone engineers at Apple decided that garbage collection consumed too much battery power.

So on the iPhone, the programmer is required to manage heap allocation and deallocation using reference counting.

Values with dynamic extent. I.e., values whose lifetimes are bounded by the invocation of a procedure or method.
When a procedure or method is called, a record of the activation of the procedure, i.e., an activation record, is pushed on the stack.

Among other things, the activation record has storage space for all parameters, local variables and temps.

When the procedure or method exits, the activation record is popped from the stack and the storage is thereby automatically reclaimed and can be reused.

Activation Records for Functions

```c
void doSomething(int x, int y) {
    double z;
    ...
}
```

<table>
<thead>
<tr>
<th>y:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x:</td>
<td></td>
</tr>
<tr>
<td>other bookkeeping</td>
<td></td>
</tr>
<tr>
<td>z:</td>
<td></td>
</tr>
</tbody>
</table>
Activation Records for **Methods**

```java
- (void) doSomething(int x, int y)
  {
    double z;
    ...
  }
```

Repr. of host object (in heap).

<table>
<thead>
<tr>
<th>sel</th>
<th>y</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>sel</td>
<td>self:</td>
<td>other bookkeeping</td>
</tr>
<tr>
<td></td>
<td>z:</td>
<td></td>
</tr>
</tbody>
</table>

1/29/10