**Making an Objective-C Statement**

Programming iPhone and Mac apps in Objective-C is about making a statement. You can recognize a statement in Objective-C immediately by noting the semicolon at the end:

statement;

You will see other lines of code, but unless the line ends with a semicolon, it is not an Objective-C statement.

**Objective-C Built-in Data Types and New Data Types**

The variables you declare in Objective-C, Objective-C data types, must be a type that the compiler can recognize. Objective-C comes with a number of built-in data types, as well as mechanisms to create new ones, for programming your iPhone or Mac OS X applications.

|  |  |  |
| --- | --- | --- |
| Built-In Types | | |
| **Type** | **Description** | **Size** |
| char | A character | 1 byte |
| int | An integer — a whole number | 4 bytes |
| float | Single precision floating point number | 4 bytes |
| Double | Double precision floating point number | 8 bytes |
| short | A short integer | 2 bytes |
| long | A double short | 4 bytes |
| long long | A double long | 8 bytes |
| BOOL | Boolean (signed char) | 1 byte |

**Enumeration types**

enum typeName { identifier1, ... identifiern};

Identifiers are of constants of type int.

**typedef**

typedef typeName identifier;

Associates an identifier with a specific type.

**Constants**

const type identifier = value;

#define identifier value

Allows you to define names for constants.

**Objective-C Operators**

Objective-C operators, like those in other programming languages, let you perform operations on variables (hence the name). Objective-C provides many operators, and keeping track of all of them can be difficult as you program your iPhone or Mac OS X apps. Use the following tables to jog your memory as to which operator accomplishes what task.

|  |  |
| --- | --- |
| Arithmetic Operators | |
| **Operator** | **What It Does** |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulo |

|  |  |
| --- | --- |
| Relational and Equality Operators | |
| **Operator** | **What It Does** |
| == | Equal to |
| != | Not equal to |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |

|  |  |
| --- | --- |
| Logical Operators | |
| **Operator** | **What It Does** |
| ! | NOT |
| && | Logical AND |
| || | Logical OR |

|  |  |
| --- | --- |
| Compound Assignment Operators | |
| **Operator** | **What It Does** |
| += | Addition |
| -= | Subtraction |
| \*= | Multiplication |
| \/= | Division |
| \%= | Modulo |
| &= | Bitwise AND |
| |= | Bitwise Inclusive OR |
| ^= | Exclusive OR |
| <<= | Shift Left |
| >>= | Shift Right |

|  |  |
| --- | --- |
| Increment and Decrement Operators | |
| **Operator** | **What It Does** |
| ++ | Addition |
| -- | Subtraction |
| \*= | Multiplication |
| /= | Division |
| %= | Modulo |
| &= | Bitwise AND |
| |= | Bitwise Inclusive OR |
| ^= | Exclusive OR |
| <<= | Shift Left |
| >>= | Shift Right |

|  |  |
| --- | --- |
| Bitwise Operators | |
| **Operator** | **What It Does** |
| & | Bitwise AND |
| | | Bitwise Inclusive OR |
| ^ | Exclusive OR |
| ~ | Unary complement (bit inversion) |
| << | Shift Left |
| >> | Shift Right |

|  |  |
| --- | --- |
| Other operators | |
| **Operator** | **What It Does** |
| () | Cast |
| , | Comma |
| Sizeof() | Size of |
| ? : | Conditional |
| & | Address |
| \* | Indirection |

**Control Statements and Loops in Objective-C**

In programming, as in life, you have to make decisions and act on them. Objective-C provides control statements and loops to help your program take action. You may want to repeat a set of instructions based on some condition or state, for example, or even change the program execution sequence. Here is the basic syntax for Objective-C control statements and loops.

**if else**

**if** (condition) {

statement(s) if the condition is true;

}

**else** {

statement(s) if the condition is not true;

}

**for**

**for** (counter; condition; update counter) {

statement(s) to execute while the condition is true;

}

**for in**

**for** (Type newVariable **in** expression ) {

statement(s);

}

or

Type existingVariable ;

for (existingVariable in expression) {

statement(s);

}

Expression is an object that conforms to the NSFastEnumeration protocol.

* An NSArray and NSSet enumeration is over content.
* An NSDictionary enumeration is over keys.
* An NSManagedObjectModel enumeration is over entities.

**while**

while (condition) {

statement(s) to execute while the condition is true

}

**do while**

do {

statement(s) to execute while the condition is true

} while (condition);

**Jump statements**

return ;

Stop execution and returns to the calling function.

break;

Leave a loop.

continue;

Skip the rest of the loop and start the next iteration.

goto labelName;

...

labelName:

An absolute jump to another point in the program (don’t use it).

exit();

Terminates your program with an exit code.

**Declaring Classes and Sending Messages in Objective-C**

Object-oriented programming languages enable you to declare classes, create derived classes (subclass), and send messages to the objects instantiated from a class. This is the essence of object-oriented programming and part of the object-oriented extensions that Objective-C adds to C. To ensure that everything operates smoothly, compiler directives are available that enable you to inform the compiler of your classes by using @class and #import.

**Interface**

#import "Superclass.h"

@interface ClassName : Superclass {

instance variable declarations;

}

method declarations

@property(attributes) instance variable declaration;

–d

**Implementation**

#import "ClassName.h"

@implementation ClassName

@synthesize instance variable ;

method definitions

–d

**Message Syntax**

[receiver message]

**#import**

#import “filename.h”

Guarantees that a header file will be included only once.

**@class**

@class ClassName;

Clues the compiler into user defined types.