CS193E: Assignment 1-B
WhatATool

Due Date

This assignment is due by **5:00 PM, January 18**.

Assignment

In this assignment you will be getting your feet wet with Objective-C by writing a small command line tool. You will create and use various common framework classes, define and use a new class, and practice good Obj-C memory management habits.

Note that Handout #3 includes a good number of Xcode tips, tricks and shortcuts which should prove handy.

The Objective-C runtime provides a great deal of functionality, and the gcc compiler understands and compiles Objective-C syntax. The Objective-C language itself is a small set of powerful syntax additions to the standard C environment.

To that end, for this assignment, you'll be working in the most basic C environment available – the main() function.

This assignment is divided up into five small sections. Each section is a mini-exploration of a number of Objective-C classes and language features. Please put each section's code in a separate C function. The main() function should call each of the section functions in order.

IMPORTANT: The assignment walkthrough begins on page 3 of this document. The assignment walkthrough contains all of the section-specific details of what is expected.

The basic layout of your program should look something like this:

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

// sample function for one section, use a similar function per section
void PrintPathInfo() {
      // Code from path info section here
}

int main (int argc, const char * argv[]) {
    NSAutoreleasepool * pool = [[NSAutoreleasePool alloc] init];

    PrintPathInfo(); // Section 1
    PrintProcessInfo(); // Section 2
    PrintBookmarkInfo(); // Section 3
    PrintPersonInfo(); // Section 5 (No function for section 4)
    PrintCategoryInfo(); // Section 6
    PrintIntrospectionInfo(); // Section 7

    [pool release];
    return 0;
}
```

Testing

In most assignments testing of the resulting application is the primary objective. In this case, testing/grading will be done both on the output of the tool, but also on the code of each section.

We will be looking at the following:

1. Your project should build without errors or warnings.
2. Your project should run without crashing.
3. Each section of the assignment describes a number of log messages that should be printed. These should print.
4. Each section of the assignment describes certain classes and methodology to be used to generate those log messages – the code generating those messages should follow the described methodology, and not be simply hard-coded log messages.
5. All code should use proper memory management techniques, and neither leak nor over-release objects. This is especially important in the final two sections where you implement and use your own Obj-C class. We will check your tool for leaks.

A note about the NSLog function. It will generate a string that is prefixed with a timestamp, the name of the process, and the pid. It will also automatically append a newline to the log statement.

```
2007-04-05 13:49:42.275 WhatATool[360] Your message here.
```

This is expected. You are only being graded on the text not generated automatically by NSLog. You do not need to attempt to suppress what NSLog prints by default.

To help make the output of your program more readable, it would be helpful to put some kind of header or separator between each of the sections.

Hints

Hints are distributed section by section but generally, the lecture notes from lecture 2 and 3 provide some very good clues with regards to Objective-C syntax, commonly used methods, and common Obj-C memory management idioms.

Troubleshooting

Remember that an Objective-C NSString constant is prefixed with an @ sign. The compiler will warn, and your program will crash if you use a C string where an NSString is required.

Extra Credit

Exploration and Q&A: We didn't seem to worry much about memory management in sections 1-3. Why is that the case? Take your program and comment out the lines of code that deal with the NSAutoreleasePool, then compile and run. Describe what happens – does it still run? Does anything new show up in the console? Why do you think you see the results that you do?

Extending the Person class:
Add methods for a Person to get and set a reference to another Person who is their bestFriend.

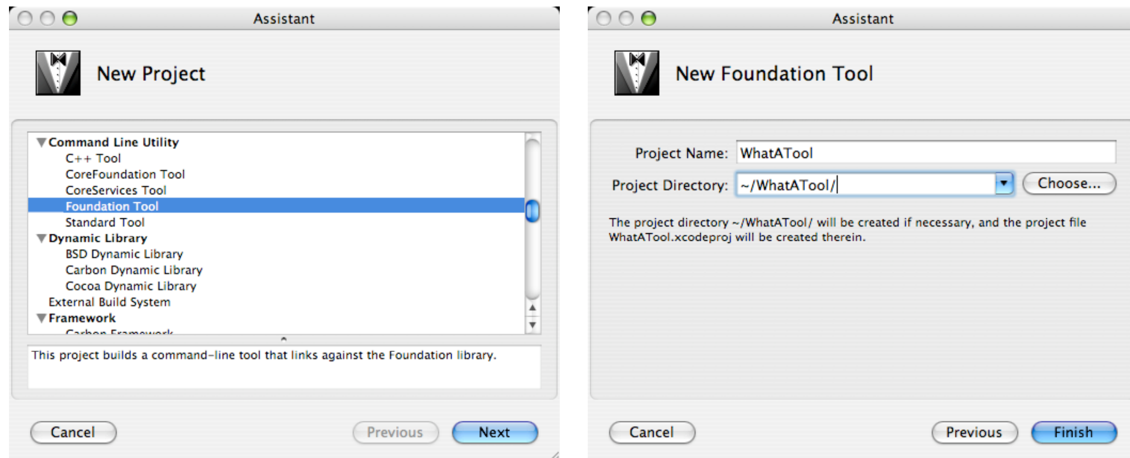
Now add a method –(float)loanValueForPerson:(Person *)person; that will return the amount of money the person receiving the message is willing to loan the person passed in as an argument.

A Person should loan their best friend 100.00, and everyone else 0.00.

Make one person (Person A) the best friend of another Person (Person B). Have Person A ask all other people for a loan, and log the loan values.

Assignment Walkthrough

In Xcode, create a new Foundation Tool project. Name it WhatATool.



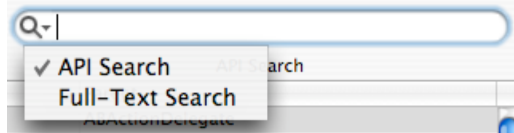
You will be doing the bulk of your work in the WhatATool.m file.
You should build and test at the end of each code-writing section.

Finding Answers / Getting to documentation

Some of the information required for this exercise exists in the class documentation, not in the course materials. You can use the Xcode documentation window to search for class documentation as well as conceptual articles.

Open the Xcode documentation window by choosing **Help** > **Documentation**.

The magnifying glass is a pop up menu that lets you choose between searching the API, or searching all documentation. For now, use the API Search.



Section 1: Strings as file system paths

NSString has a set of methods that allow you to manipulate file system paths. You can find them in the NSString documentation grouped under the heading "Working with paths".

In this section, begin with an NSString constant that is a tilde – the symbolic representation for your home directory in Unix.

```
NSString *path = @"~";
```

Starting with that string, find a path method that will expand the tilde in the path to the full path to your home directory. Use that method to make a new string, and log the full path in a format like:

```
My home folder is at '/Users/dempsey'
```

NSString has a method that will return an array of path components. Each element in the returned array is a single component in the original path.

Use this method to get an array of path components for the path you just logged.

Use an NSEnumerator to enumerate through each item in the returned array, and log each path component. The result should look something like:

```
/
Users
dempsey
```

Section Hints:
With string convenience methods, a common pattern is to reuse the same variable:

```
NSString *aString = @"Bob";
aString = [aString stringWithSomeModification];
```

The use of an NSEnumerator is a common Cocoa programming idiom. The lecture slides and NSEnumerator documentation both contain an example.

Section 2: Finding out a bit about our own process

Look up the class NSProcessInfo in the documentation.

You will find a class method that will return an NSProcessInfo object. (In fact, you should find a very handy code sample there as well).

From this object, you can access the name and process identifier (pid) of the process.

Log these pieces of information to the console using NSLog in the format:

```
Process Name: 'WhatATool' Process ID: '4556'
```

Section Hints:
We didn't discuss NSProcessInfo in lecture at all, so you aren't missing any slides or notes. All of the information you need is in the NSProcessInfo class documentation.

Section 3: A little bookmark dictionary

In this section, you will build a small URL bookmark repository using a mutable dictionary. Each key is an NSString that serves as the description of the URL, the value is an NSURL.

Create a mutable dictionary that contains the following key/value pairs:

Key	**URL value**
Stanford University	http://www.stanford.edu
Apple	http://www.apple.com
CS193E	http://cs193e.stanford.edu
Stanford on iTunes U	http://itunes.stanford.edu
Stanford Mall	http://stanfordshop.com

Get a key enumerator for the dictionary. Enumerate through the keys of the dictionary.

While enumerating through the keys, check each key to see if it starts with @"Stanford". If it does, retrieve the NSURL object for that key from the dictionary, and log both the key string and the NSURL object in the following format below. If the key does not start with @"Stanford", no output should be printed.

```
Key: 'Stanford University' URL: 'http://www.stanford.edu'
```

Section Hints:
Use +URLWithString: to create the URL instances.

NSString has methods to determine if a string has a particular prefix or suffix. *Remember to log only those keys that start with 'Stanford'.*

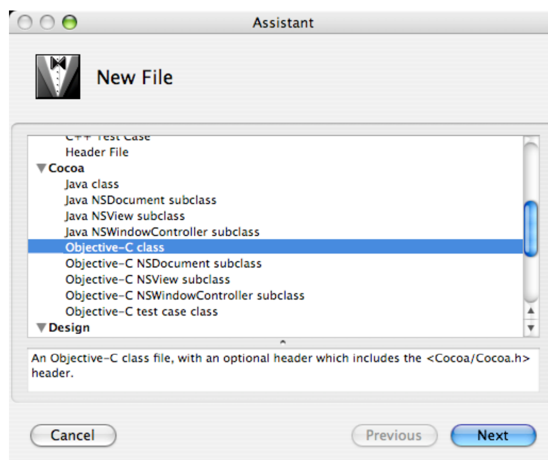
The methods for creating and adding items to a mutable dictionary are located in the Lecture #2 slides. Remember that an NSMutableDictionary is a subclass of NSDictionary and so inherits all of its methods.

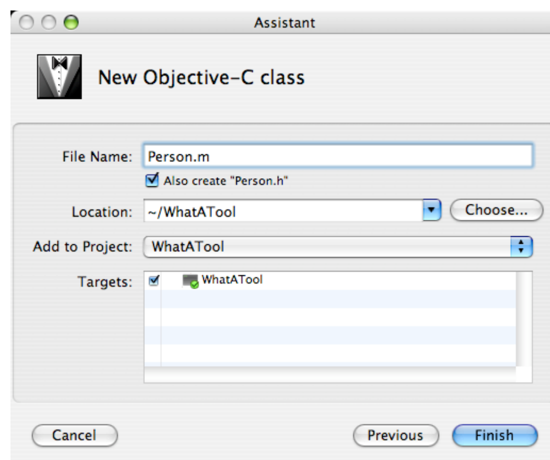
The lecture notes describe an object enumerator. Check the NSDictionary documentation for a method that will return an NSEnumerator that will enumerate keys instead.

Section 4: Creating a new class

Create a Person class. To create the files for the new class in Xcode:

1. Choose File > New Files…
2. Select 'Objective-C class' from the list of file choices
3. Name the file Person.m – Be certain the checkbox to create a header file is also checked





Please implement the Person class as follows:

1. It should inherit from NSObject.

2. The Person class will have methods to access and set the following:

    - A first name – the first name will be an NSString.
    - A last name – also an NSString
    - A favorite number - for purposes of this assignment, just an int
    - Likes cheese? – a BOOL value. Do they like cheese?

3. The Person class will also have a method to return their full name. This is the first and last name concatenated, with a space between them.

4. The person will have a -description method. Example output from this method:

```
Hello my name is James Dempsey.  My favorite number is 109.  I like cheese.
```

5. The Person class will use good Objective-C memory management techniques when created and deallocated, as well as when attributes are accessed or set.

Section Hints:
The lecture slides from Lecture #2 / #3 contain a number of hints, including accessor method naming conventions, and the general syntax for memory management.

Section 5: Using the Person class

Write a C function called by main() that uses your newly created Person class.

You will have to add an import statement to the WhatATool.m file:

```
#import "Person.h"
```

IMPORTANT: In this section, you are expected to use +alloc and –init methods to create objects.

1. Create a mutable array.

2. Create instances of the following people with the following values

2A. Have each person log their description method

2B. Then add each person to the array.

First Name: Stan
Last Name: Jirman
Favorite Number: 200
Likes Cheese? No

First Name: Paul
Last Name: Marcos
Favorite Number: 566
Likes Cheese? Yes

First Name: Derek
Last Name: Clegg

Favorite Number: -1
Likes Cheese? Yes

First Name: James
Last Name: Dempsey
Favorite Number: 109
Likes Cheese? Yes

There should be four 'introductions' (Person descriptions) logged.

3. We are throwing a wine and cheese party for people who like cheese and positive numbers.

Generate the guest list for the array of people by logging a header string, and then enumerating the array, and logging the full name of each person who likes cheese and has a positive favorite number.

The result should look something like:

```
Cheese Party Guest List:
Paul Marcos
James Dempsey
```

Section 6: Creating a Category

In this section you will add a category to the NSString class. The category adds a convenience method for retrieving a particular system path location, which may come in handy for future assignments (wink, wink, nudge, nudge).

Using a category, you can declare and implement additional methods for a class. This can be done even with classes you don't have the source code to, e.g. NSString or NSArray. This is often used to add utility or convenience methods to existing framework classes. You can add both instance methods and class methods.

Create a header file and an implementation file for your category.

Xcode does not provide a separate "New File" type for categories. It is usually easiest to create a new Objective-C class file and header, and edit those files to make them into a category.

It is a common naming convention of category files to include the name of the class being extended and the name of the category separated by a plus sign.

Name the file `NSString+SystemPathUtilities.m`. Be certain the checkbox to create a header file is also checked.

Note that you will need to edit the existing file to turn it from a class definition to a category definition.

An aside regarding paths in Mac OS X:

> Mac OS X has a flexible structure of well known file locations. For instance, the "Library" folder contains things such as preference files, application support files, plug-ins, caches, etc.
>
> There are multiple domains in which the "Library" folder can live.

> The **system domain** library folder is located in `/System/Library`. This is used by Mac OS X itself and is typically not writable by non-administrators (i.e. keep your hands off it).
>
> The **local domain** library folder is located at `/Library`. This is where applications put information that pertains to all users of a given machine.
>
> The **user domain** library folder is located at `~username/Library`. This is where settings and information specific to a particular user are kept.
>
> Another well known file location is the `Application Support` folder. It is located inside of the Library directory. It can be in either the user domain or the local domain. For example, the `Application Support` directory in the local domain would be located at `/Library/Application Support`.
>
> It is not wise to hardcode path names when it can be avoided. Cocoa provides a function, `NSSearchPathForDirectoriesInDomains()` which provides a very flexible mechanism for returning an array of paths that can be searched in different domains.
>
> The `NSSearchPathForDirectoriesInDomains` function takes three arguments. The first indicates what directory you are looking for. There are a wide variety of directories you can look for, see the definition for NSSearchPathDirectory to determine which enum value to use. The second argument is a bitmask of the domains you are interested in. You can, for example, indicate that you are interested in the system and local domains (NSSystemDomainMask | NSLocalDomainMask). Or you could say you want all domains (NSAllDomainsMask). Or you could indicate a single specific domain. The final argument indicates whether you want the user's home directory in any of the returned paths to be abbreviated with a "~" instead of being fully specified.
>
> The returned array will contain a list of directories meeting the criteria set forth by the arguments. They are listed in the order an application should search.

Use the NSSearchPathForDirectoriesInDomains function to obtain the Application Support directory in the user domain. It should be the first object in the returned array. You should take care to validate the count of objects in the returned array before accessing the path(s) it contains.

In your NSString category, add a **class method** named '`userApplicationSupportDirectoryPath`' which returns a string.

Add a second method to your NSString category that is an **instance method**. This method should return a string which has the original string value duplicated, separated by a space. For example, if you have a string @"wakka", your category method would return the string @"wakka wakka".

Call your category's class method and log the result.

Create a few strings and call your category's instance method on each of the strings and log both the original string and the result of the category method. For example:

```
Original string: 'wakka'  category result: 'wakka wakka'
```

Section Hints:
The NSString class defines a number of utility methods for working with paths. In fact, these utilities methods are defined in a category on NSString, found in NSPathUtilities.h. You should never write Cocoa code that assumes a path separator is "/". The NSString utilities provide sufficient abstraction for working with paths and path components.

Remember that inside of a method the "self" object is the receiver of the method. So in the context of your category's instance method, "self" is the string object that received the message.

Section 7: Selectors, Classes and Introspection

Objective-C has a number of facilities that add to its dynamic object-oriented capabilities. Many of these facilities deal with determining and using an object's capabilities at runtime.

Create a mutable array and add objects of various types to it. Create instance of the classes we've used elsewhere in this assignment to populate the array: NSString, NSURL, NSProcessInfo, Person, etc. Create some NSMutableString instances and put them in the array as well. Feel free to create other kinds of objects also.

Iterate through the objects in the array and do the following:

1. Print the class name of the object.
2. Log if the object is member of class NSString.
3. Log if the object is kind of class NSString.
4. Log if the object responds to the selector "lowercaseString".
5. If the object does respond to the lowercaseString selector, log the result of asking the object to perform that selector (using performSelector:)

For example, if an array contained an NSString, an NSURL and a Person the output might look something like this:

```
2008-01-10 20:56:03 WhatATool[360] Class name:  NSCFString
2008-01-10 20:56:03 WhatATool[360] Is Member of NSString: NO
2008-01-10 20:56:03 WhatATool[360] Is Kind of NSString: YES
2008-01-10 20:56:03 WhatATool[360] Responds to lowercaseString: YES
2008-01-10 20:56:03 WhatATool[360] lowercaseString is:  hello world!
2008-01-10 20:56:03 WhatATool[360] ======================================
2008-01-10 20:56:03 WhatATool[360] Class name:  NSURL
2008-01-10 20:56:03 WhatATool[360] Is Member of NSString: NO
2008-01-10 20:56:03 WhatATool[360] Is Kind of NSString: NO
2008-01-10 20:56:03 WhatATool[360] Responds to lowercaseString: NO
2008-01-10 20:56:03 WhatATool[360] ======================================
2008-01-10 20:56:03 WhatATool[360] Class name:  Person
2008-01-10 20:56:03 WhatATool[360] Is Member of NSString: NO
2008-01-10 20:56:03 WhatATool[360] Is Kind of NSString: NO
2008-01-10 20:56:03 WhatATool[360] Responds to lowercaseString: NO
2008-01-10 20:56:03 WhatATool[360] ======================================
```

Section Hints:
When you ask various classes, such as NSString, for its class name, you may not get the result you expect. Your logs should report what the runtime reports back to you - don't worry if some of the class names may seem strange, these are implementation details of the NSString class. Encapsulation at work!