XcodeによるiPhone/Mac アプリの制作 (1)

Objective-C programming

正整数電卓を制作する
create View-based Application

株式会社あいはら 山田 泰司 <taiji@aihara.co.jp>

iPhone(iPod touch)/iPad アプリ制作に必要なもの

- Intel Mac … PowerPC Mac は不可
- Mac OS X 10.6(Snow Leopard) 以上
  … 10.5 Leopard 以前は不可
- Xcode 3.2.5 & iOS SDK 4.2 以上
- iOS Dev Center へのメンバー登録
  … iOS SDKのダウンロードに必要。無償

iOS Dev Center への
メンバー登録に必要なもの

- メールアドレス … Apple ID となる
- iOS Dev Center へのログインパスワード
- 氏名（姓名）… 英語での登録を推奨
  日本語だと後に一手間かかる、けど、大丈夫
- 住所、国籍、電話番号、郵便番号、所属など
- 詳細は ADC(Apple Developer Connection)
  http://developer.apple.com/jp/ から

Xcode 利用に必要なもの

iOS SDK は使わない、
Xcode による Mac アプリや Unix/X11 プログラムの開発のみの場合

- Intel Mac or PowerPC Mac マシン …
  Mac mini (Intel) ¥64,800〜
- Mac OS X … マシンに付属のOS
- Xcode … 付属インストールDVD/CD
  の追加インストールとして同梱
- つまり Mac があれば特に他は不要

iPhone/iPad 実機への自作アプリの動作には
iOS Developer Program への入会が必要

- 年間参加費 ¥10,800
- クレジットカード、メールアドレス
- アップルストアのアカウント
- 購入手続きが完了するとメールで届く
  アクティベーションコード
- iPhone(iPod touch)/iPad … ¥20,900〜

iPhone/iPad 実機への自作アプリの動作には
iOS Provisioning Portal による App ID が必要

1. iOS PP(Provisioning Portal) のアクティベーションが正常に完了

2. iOS Development 証明書の取得と登録

   (1) CSR(Certificate Signing Request)の作成：
   キーチェーンアクセス→証明書アシスタント→認証局に証明書を要求→ファイルに保存

   (2) CSR を iOS PP へ提出し、承認

   (3) Development 証明書をダウンロードし、キーチェーンアクセスへ登録する

3. iOS PP へ実機のデバイスIDの登録

4. iOS PP で App ID (例: **com.mac.taiji-yamada.uint64Calculator**) の作成

5. iOS PP で App ID の Provisioning Profile の取得と Xcode のオーガナイザへの登録

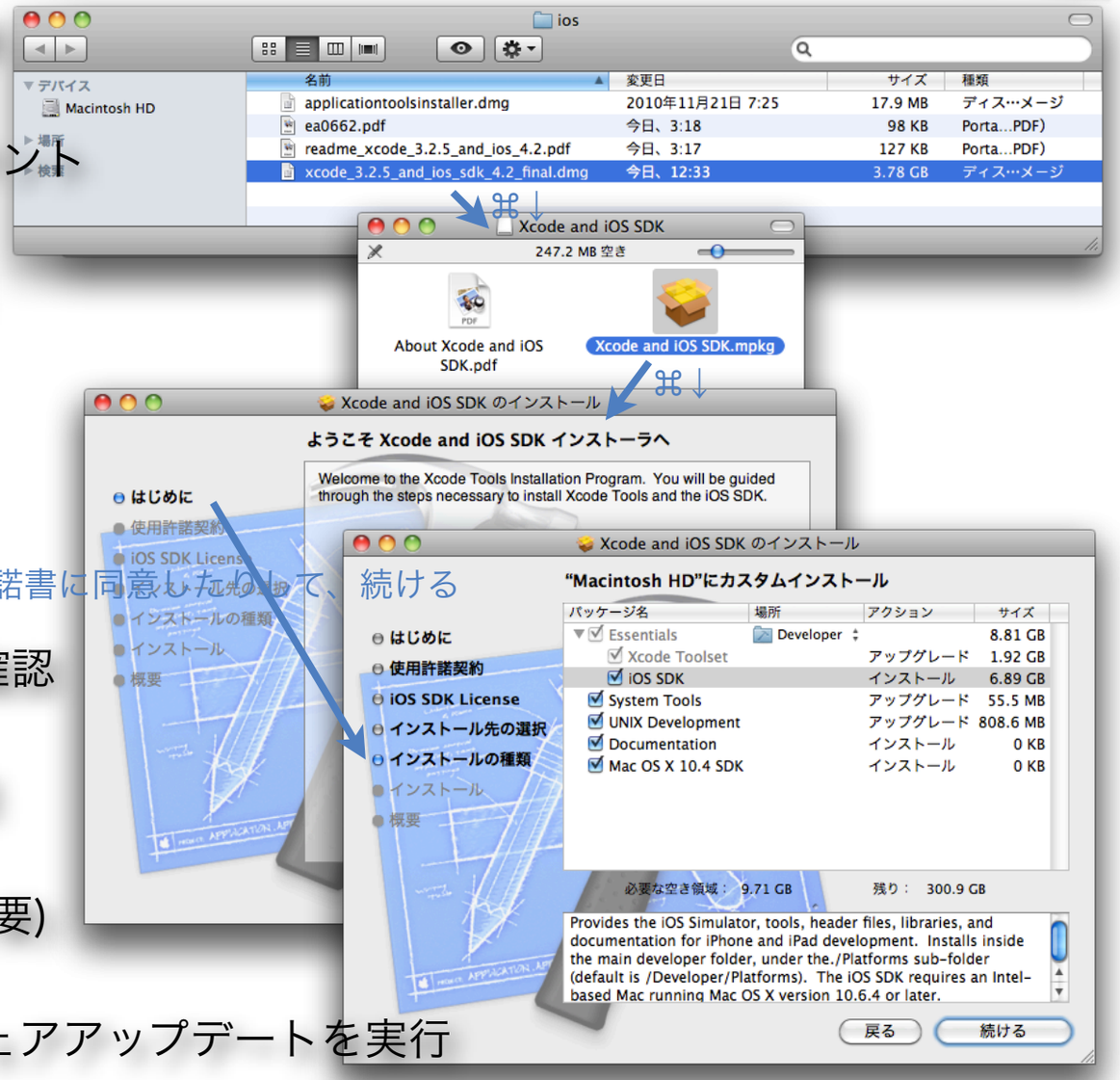
✴ 詳しくは iOS PP ホームの文書や動画による解説などの一次情報を必ず参考にすること

Xcode & iOS SDK のインストール

1. Mac での通常のインストーラと同様

   (1) ～.dmg ディスクイメージをマウント

   (2) ～.mpkg メタパッケージを開く



使用許諾書に同意したりして、続ける

2. インストールの種類で iOS SDK を確認

3. 古い Mac OS X 用 SDK も選択可能

4. インストール(但し、管理者権限が必要)

5. インストール後、念の為、ソフトウェアアップデートを実行

Xcode の起動および Dock への追加

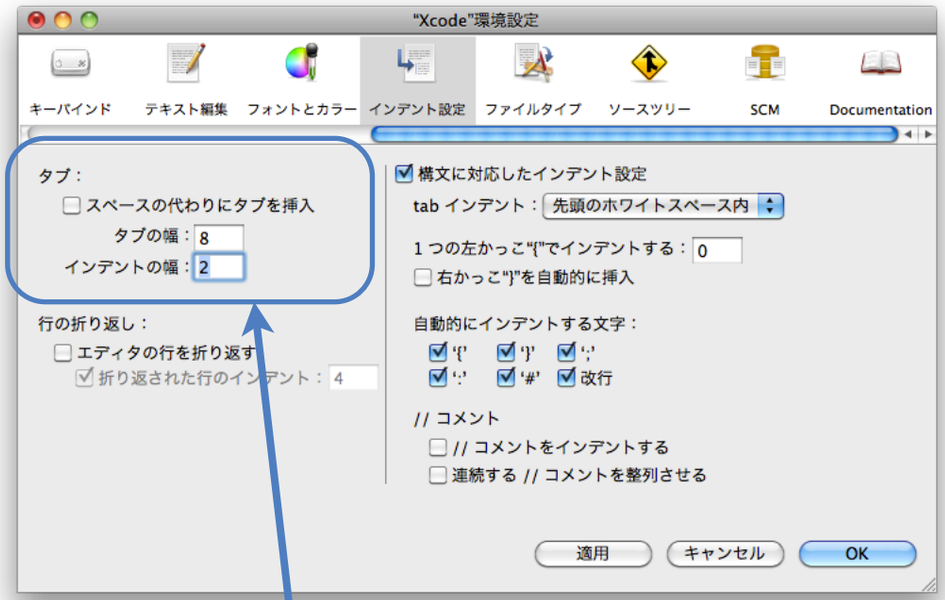
1. /Developer/Application/Xcode.app まで辿っていって開く

2. 毎回これは面倒なので、Dock に Xcode を追加しよう
   Dock の Xcode を長押し、または、^+ボタン

3. 新規 Xcode プロジェクトの作成
   クリック

Xcode の環境設定

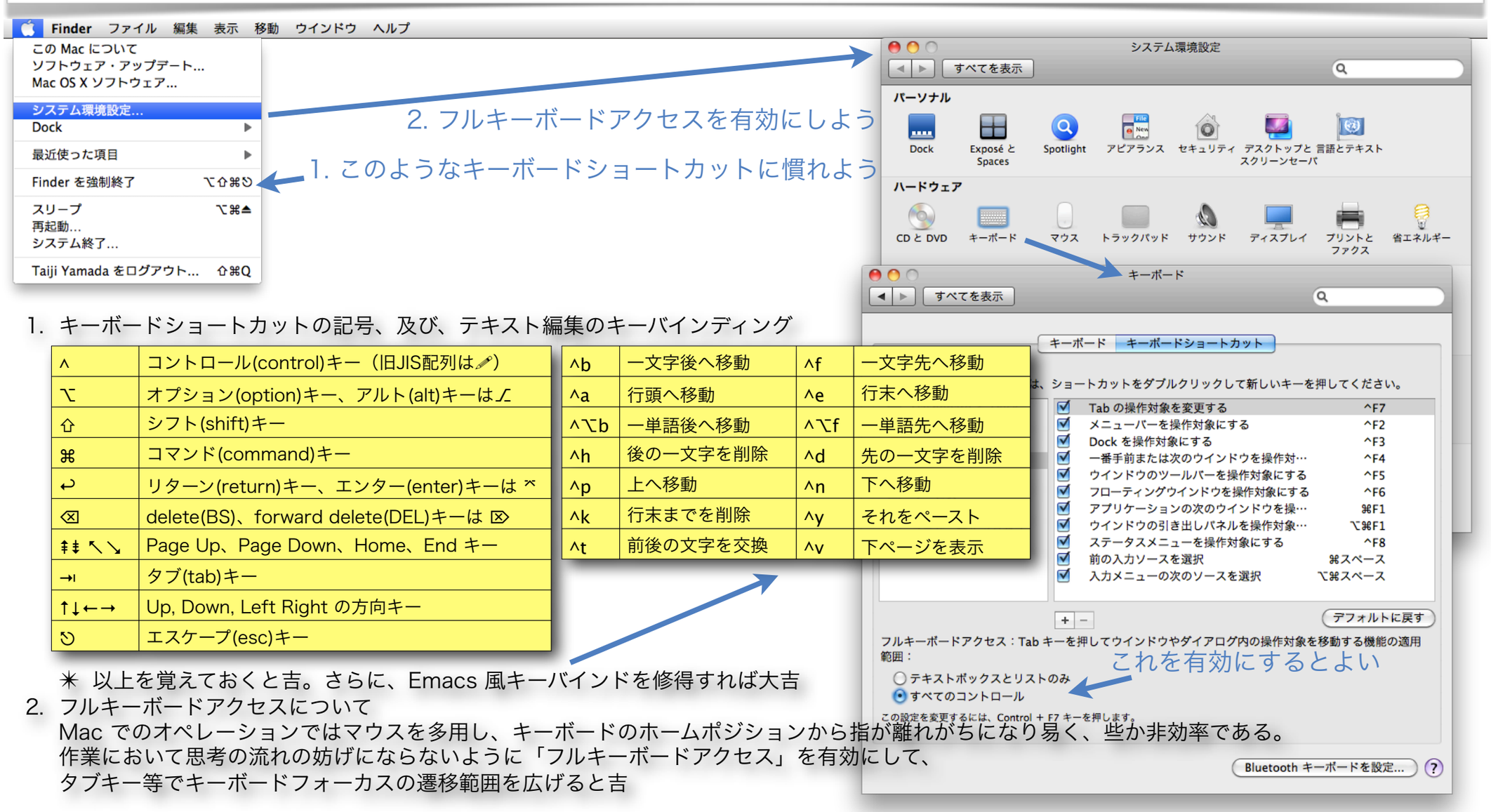
Xcode メニュー「環境設定」(⌘,)



Xcode の環境設定で少なくともこれだけは設定しておこう！

デフォルトのタブの設定が Emacs 等のエディタと親和性が低いので、Xcode メニューの「環境設定」の「インデント設定」で上記のようにカスタマイズ！

キーボードショートカットの記号、テキスト編集のキーバインディング及びフルキーボードアクセス



1. キーボードショートカットの記号、及び、テキスト編集のキーバインディング

^	コントロール(control)キー（旧JIS配列は）	^b	一文字後へ移動	^f	一文字先へ移動
⌥	オプション(option)キー、アルト(alt)キーは	^a	行頭へ移動	^e	行末へ移動
⇧	シフト(shift)キー	^⌥b	一単語後へ移動	^⌥f	一単語先へ移動
⌘	コマンド(command)キー	^h	後の一文字を削除	^d	先の一文字を削除
↩	リターン(return)キー、エンター(enter)キーは	^p	上へ移動	^n	下へ移動
⌫	delete(BS)、forward delete(DEL)キーは ⌦	^k	行末までを削除	^y	それをペースト
⇞ ↖ ↘	Page Up、Page Down、Home、End キー	^t	前後の文字を交換	^v	下ページを表示
⇥	タブ(tab)キー				
↑↓←→	Up, Down, Left Right の方向キー				
⎋	エスケープ(esc)キー				

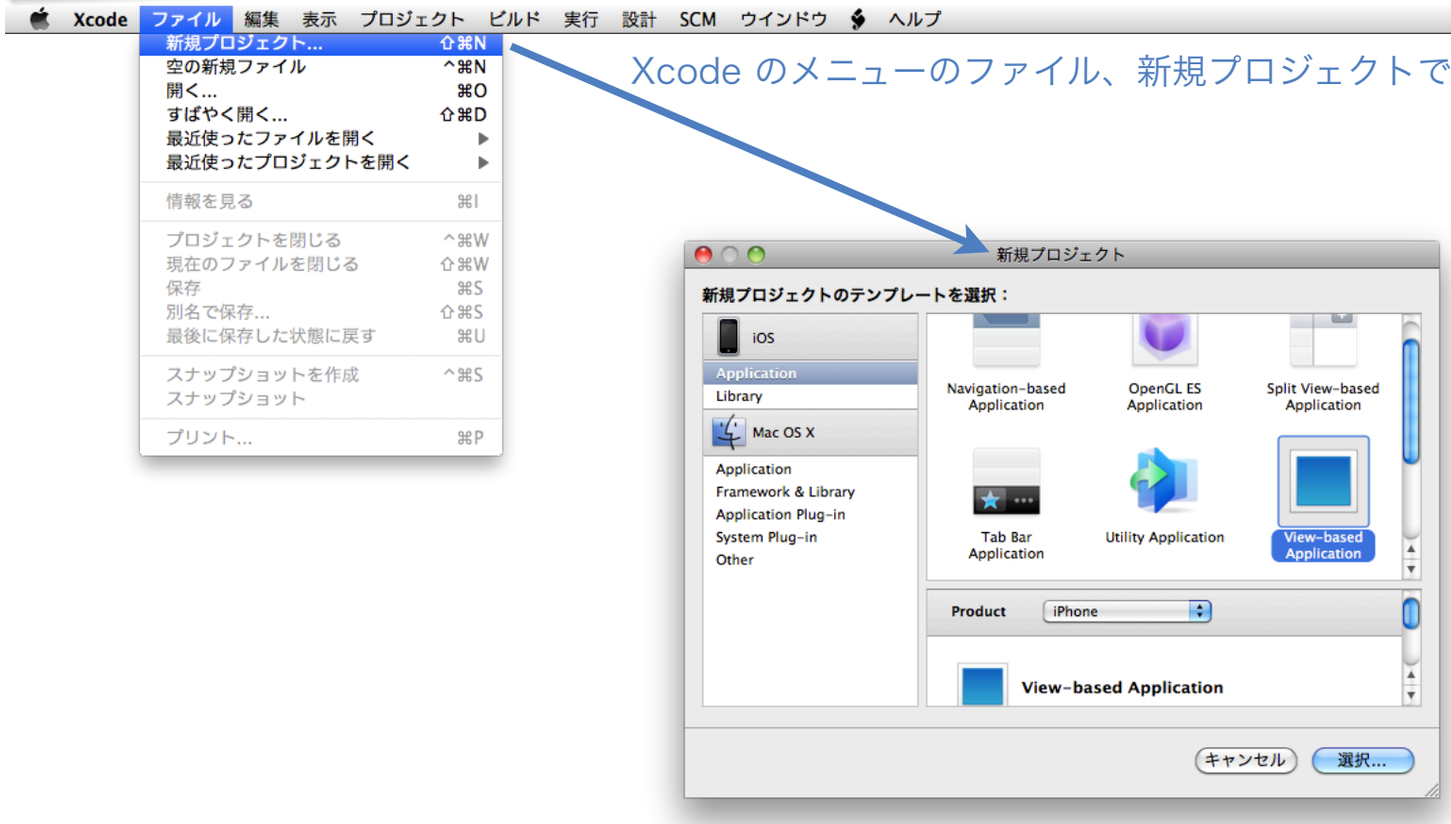
✳ 以上を覚えておくと吉。さらに、Emacs 風キーバインドを修得すれば大吉

2. フルキーボードアクセスについて

Mac でのオペレーションではマウスを多用し、キーボードのホームポジションから指が離れがちになり易く、些か非効率である。
作業において思考の流れの妨げにならないように「フルキーボードアクセス」を有効にして、
タブキー等でキーボードフォーカスの遷移範囲を広げると吉

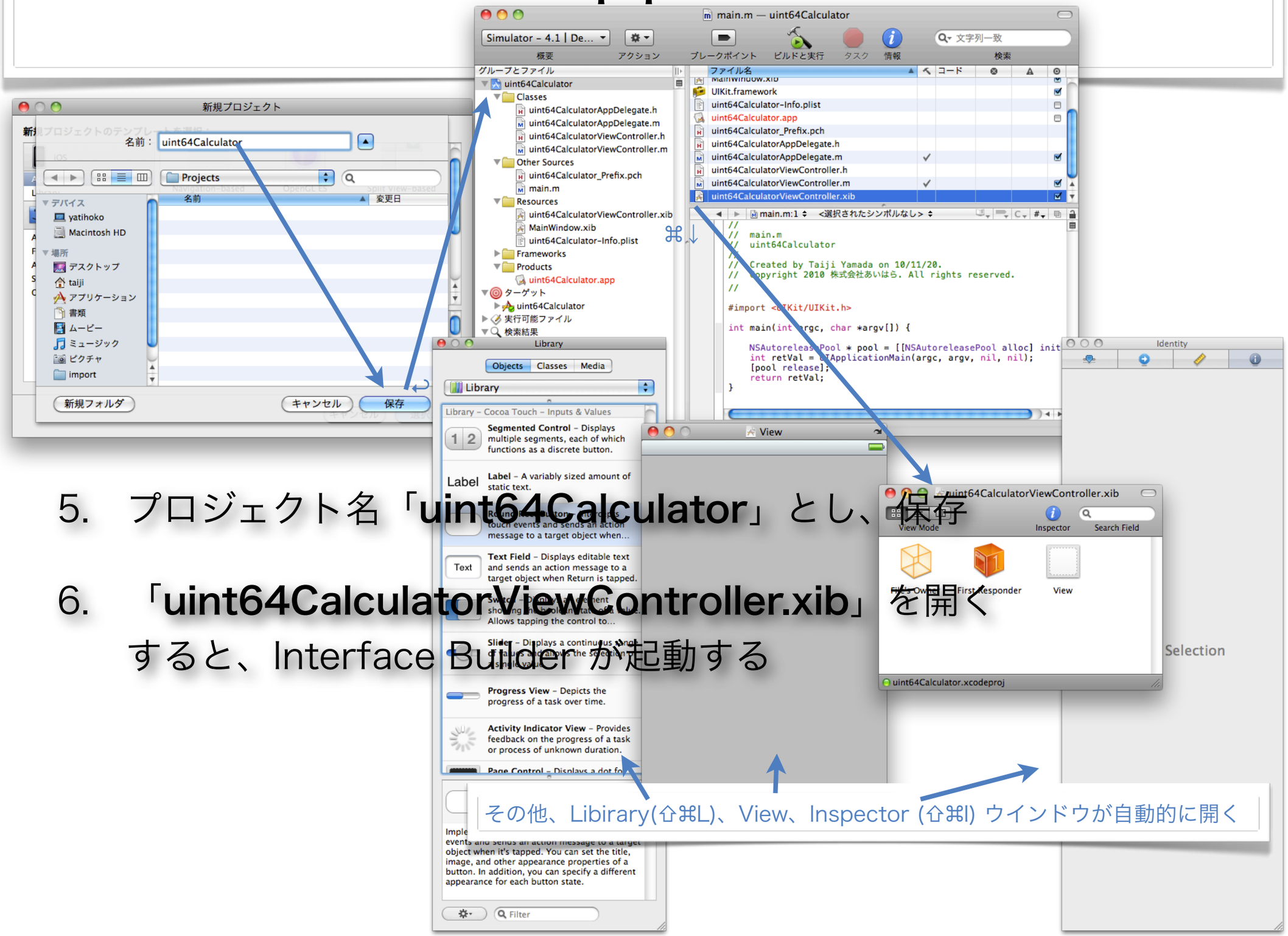
新規プロジェクトの作成

View-based Application テンプレートの選択



4. iOS の Application の
   View-based Application を選択

View-based Application の制作



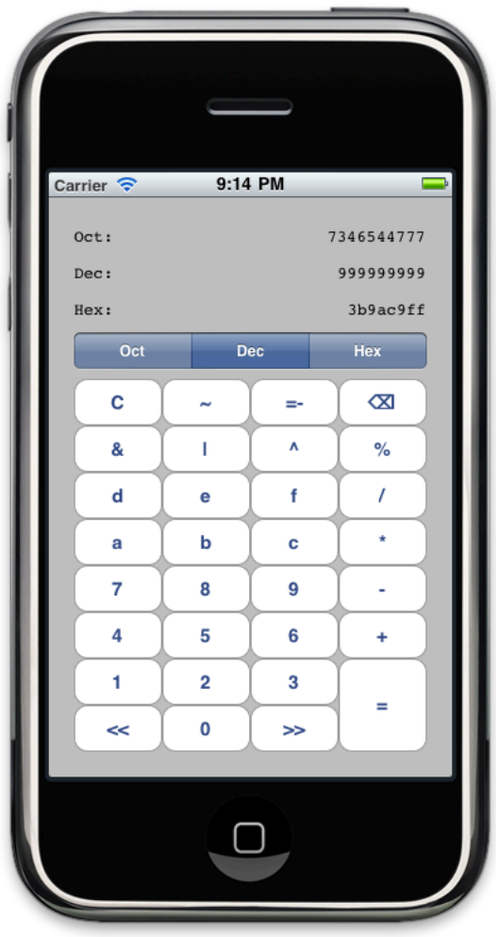
5. プロジェクト名「**uint64Calculator**」とし、保存

6. 「**uint64CalculatorViewController.xib**」を開く
   すると、Interface Builder が起動する

正整数電卓の設計

- プログラマ向け電卓
- 符号無し整数の四則演算、論理演算、ビット演算が可能
- 8進数、10進数、16進数が同時に表示可能
- とりあえずは、６４ビット符号無し整数のみ

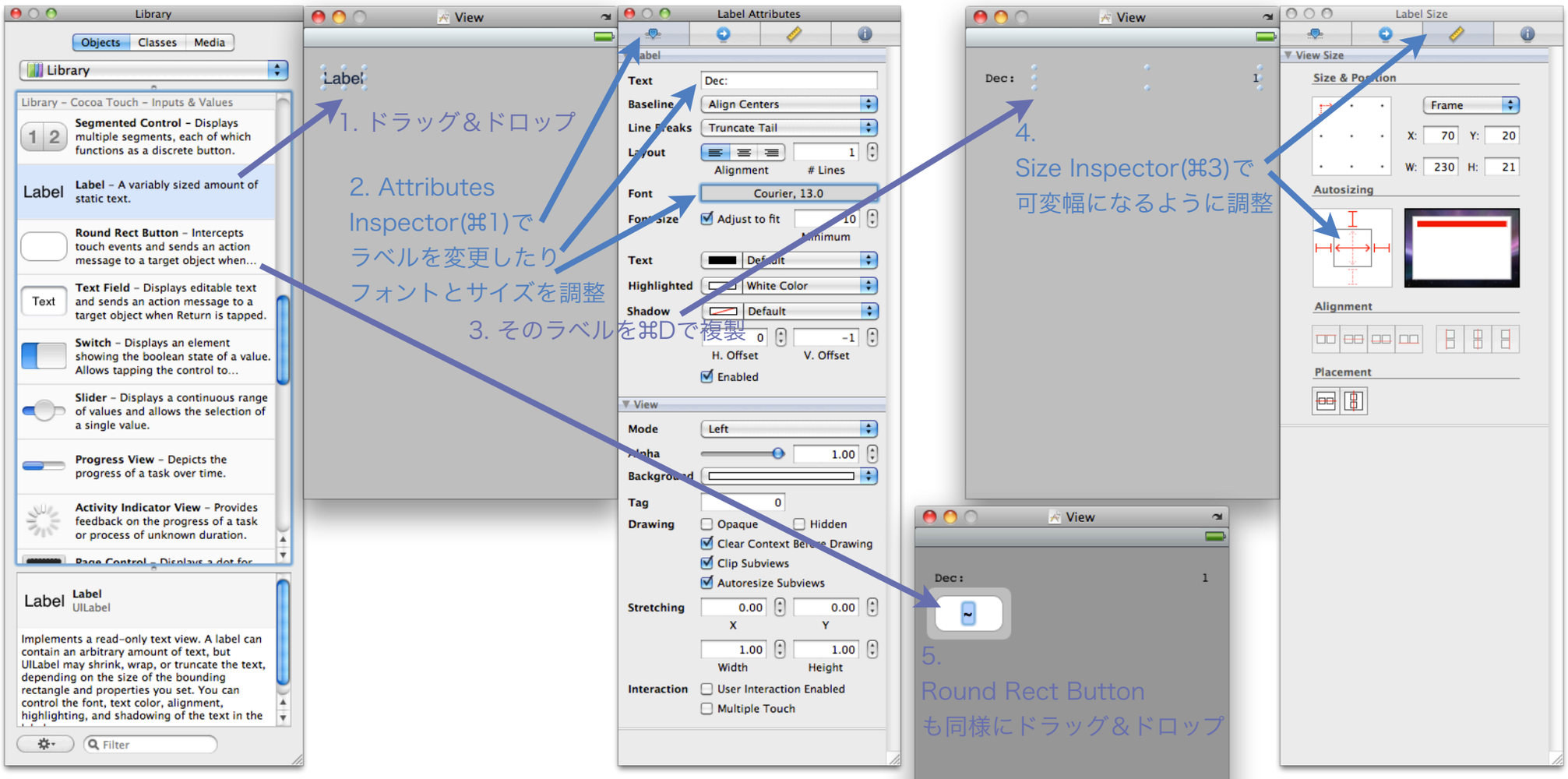
とにかくこんなのを作りたい ☞
簡単なところから手を着けよう



1. 「~」１の補数(ビット否定)などの単項演算子のボタンの処理
2. Clear「C」、Delete「⌫」、数値ボタンの処理
3. 「+-*/%«»&|^」などの二項演算子と「＝」ボタンの処理
4. 電卓操作の様式に沿った数値、演算子入力の処理
5. 10進数に加えて、8進数、16進数の入出力の処理

正整数電卓の制作(1)

単項演算子「~」1の補数の処理

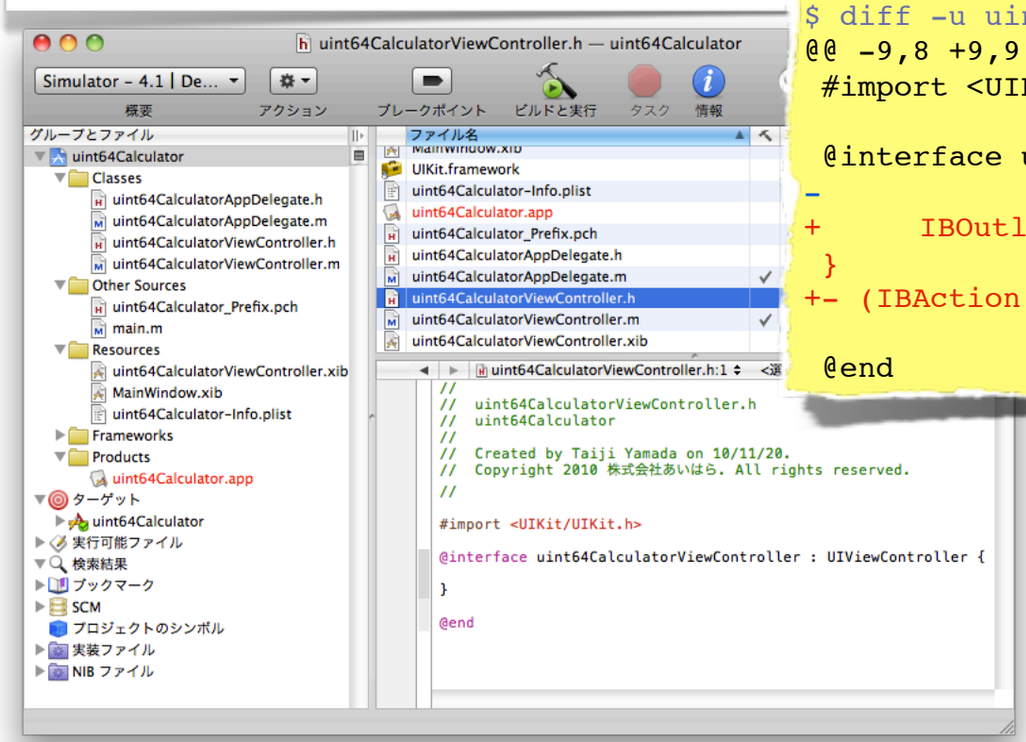


1. ラベルとボタンの配置が完了したら、⌘Sで保存

2. ⌘タブで Interface Builder から Xcode へ戻る

正整数電卓の制作(2)

単項演算子「~」1の補数の処理




```
$ diff -u uint64CalculatorViewController.h~ uint64CalculatorViewController.h
@@ -9,8 +9,9 @@
 #import <UIKit/UIKit.h>

 @interface uint64CalculatorViewController : UIViewController {
-
+       IBOutlet UILabel *labelDec;
 }
+- (IBAction)buttonPressed:(UIButton *)sender;

 @end
```

3. 「**uint64CalculatorViewController.h**」を編集、⌘Sで保存

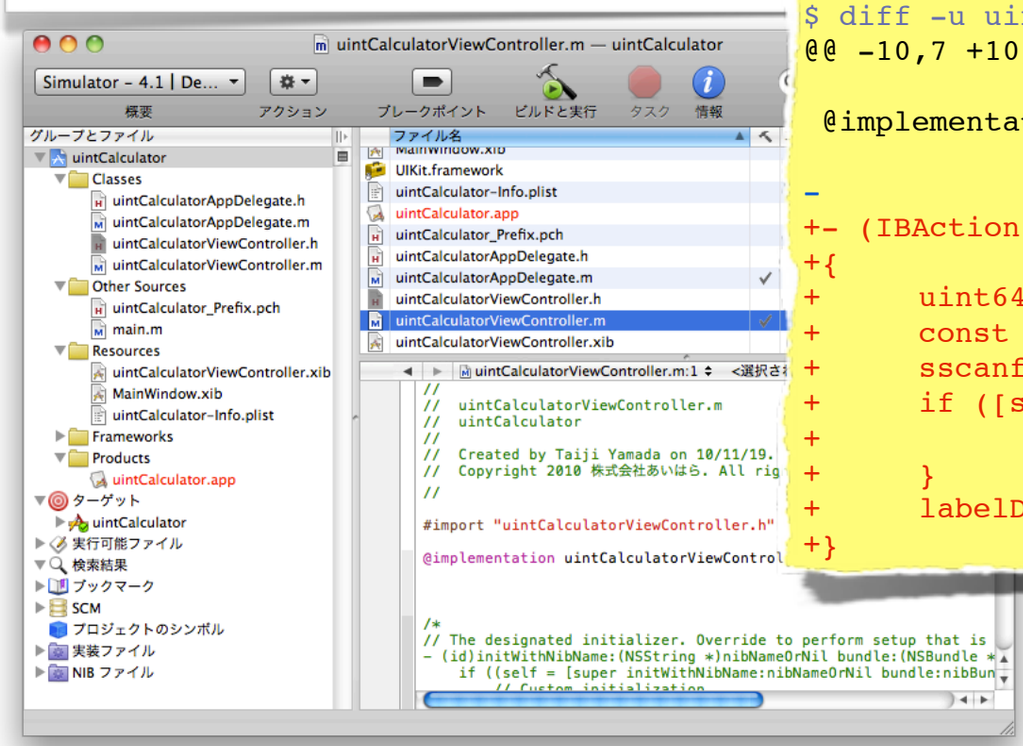
   * 上記、編集内容は diff -u (Unified diff. 形式)「行頭"-"が削除行、"+"が追加行」で表記

   (1) UILabel クラスの Outlet 先となるポインタ **labelDec** を宣言

   (2) UIButton クラスからの IBAction メソッド **buttonPressed:** を宣言

正整数電卓の制作(3)

単項演算子「~」1の補数の処理




```
$ diff -u uint64CalculatorViewController.m~ uint64CalculatorViewController.m
@@ -10,7 +10,16 @@

 @implementation uint64CalculatorViewController


-
+- (IBAction)buttonPressed:(UIButton *)sender
+{
+      uint64_t number;
+      const char *p = [labelDec.text cStringUsingEncoding:NSUTF8StringEncoding];
+      sscanf(p, "%llu", &number);
+      if ([sender.titleLabel.text isEqual:@"~"]) {
+            number = ~number;
+      }
+      labelDec.text = [NSString stringWithFormat:@"%llu", number];
+}
```

4. 「**uint64CalculatorViewController.m**」を編集、⌘Sで保存

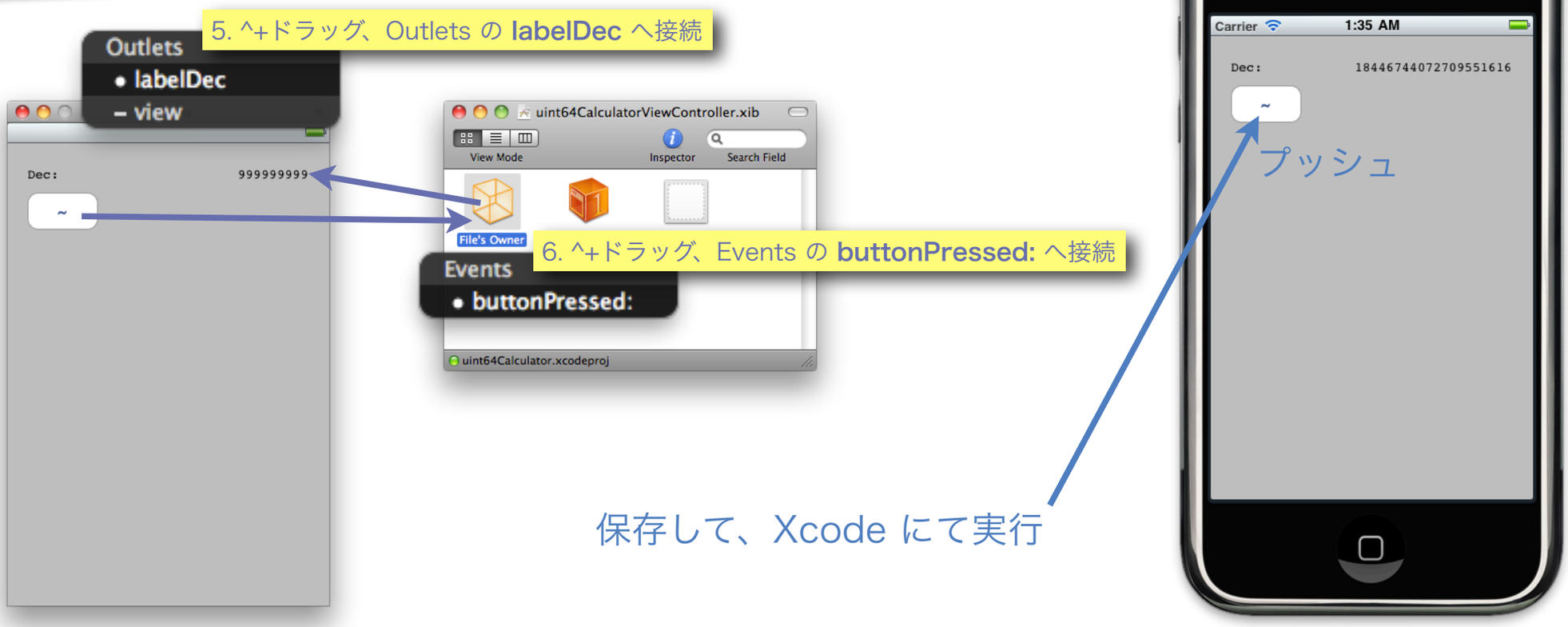
   * 上記、編集内容は diff -u (Unified diff. 形式) で明記

(1) UIButton クラスからの IBAction メソッド **buttonPressed:** を定義

(2) ここでは、**labelDec** の text をプログラミング言語C形式の文字列に変換し、
sscanf によって64ビット符号無し整数 number へ10進数として入力している。
そして、**labelDec**.text に結果となる number の10進数表記をセットしている。

正整数電卓の制作(4)

単項演算子「~」1の補数の処理



5. File's Owner から UILabel へコントロールキー+ドラッグで **labelDec** に Outlet
   (File's Owner: クラス **uint64CalculatorViewController**、⌘4で確認可能)

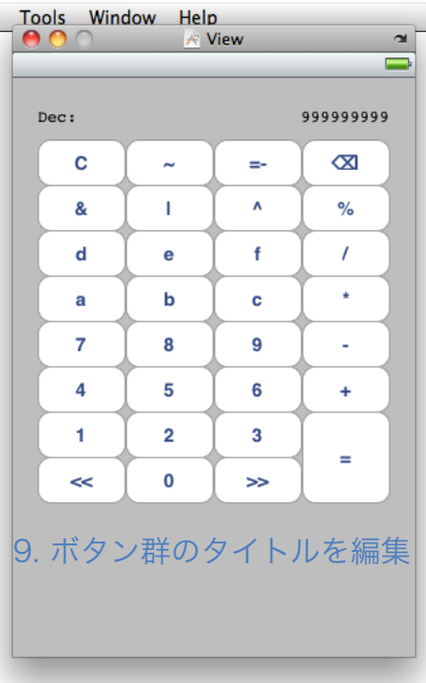
6. UIButton から File's Owner へコントロールキー+ドラッグで **buttonPressed:** に Action

7. ⌘S で保存して、Xcode に戻って、⌘R でシミュレータを実行
   「~」ボタンを押して正しく動作していることを確認

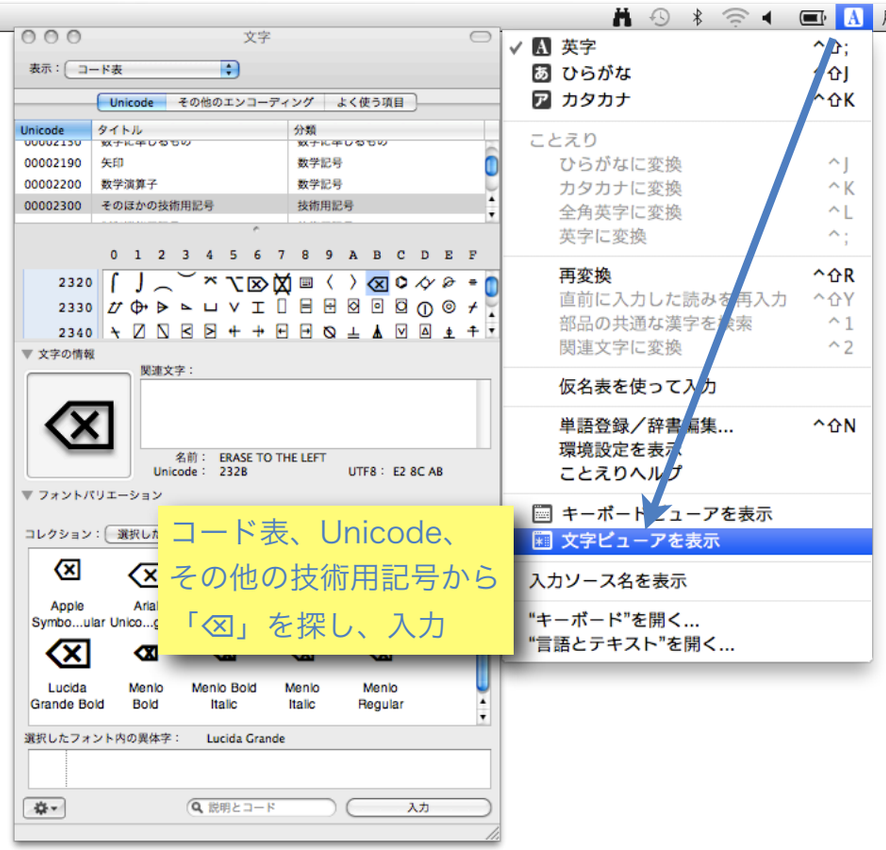
正整数電卓の制作(5)

Clear や Delete、数値、二項演算子、「=」ボタンの処理

8.「~」ボタン群を複製する



9. ボタン群のタイトルを編集



8.　「~」ボタンを⌘Dで複製、ボタン群をさらに⌘Dで複製して、総数２８個のボタンを作成

9.　ボタン群のタイトルをすべて編集する。但し、
「=」ボタンのみはサイズを大きくして、余りのボタン一つは削除する。
ボタンのタイトルの「⌫」は、メニューバーの入力メニューの文字ビューアから入力。
ついでに、**labelDec** のテキストを「0」にしておく。(⌘Sで保存を忘れずに。)

正整数電卓の制作(6)

Clear や Delete、数値、二項演算子、「=」ボタンの処理

```
$ diff -u uint64CalculatorViewController.h~ uint64CalculatorViewController.h
 @interface uint64CalculatorViewController : UIViewController {
       IBOutlet UILabel *labelDec;
+      uint64_t lhsNumber;
+      NSString *stringOp;
 }
 - (IBAction)buttonPressed:(UIButton *)sender;
```

```
$ diff -u uint64CalculatorViewController.m~ uint64CalculatorViewController.m
       if ([sender.titleLabel.text isEqual:@"~"]) {
              number = ~number;
       }
+      else if ([sender.titleLabel.text isEqual:@"=-"]) {
+             number = -number;
+      }
+      else if ([sender.titleLabel.text isEqual:@"C"]) {
+             number = 0;
+      }
+      else if ([sender.titleLabel.text isEqual:@"⌫"]) {
+             const char *p = [[labelDec.text substringToIndex:labelDec.text.length-1] cStringUsingEncoding:NSUTF8StringEncoding];
+             if (p[0] != '\0') sscanf(p, "%llu", &number); else number = 0;
+      }
+      else if ([sender.titleLabel.text isEqual:@"+"]) {
+             lhsNumber = number;
+             number = 0;
+             stringOp = @"+";
+      } …（中略：「-」「*」「/」「%」「<<」「>>」「&」「|」「^」についても同様に処理する）
+      else if ([sender.titleLabel.text isEqual:@"="]) {
+             if ([stringOp isEqual:@"+"]) {
+                    number = lhsNumber + number;
+             } …（中略：「-」「*」「/」「%」「<<」「>>」「&」「|」「^」についても同様に処理する）
+      }
+      else {
+             uint64_t num;
+             const char *p = [sender.titleLabel.text cStringUsingEncoding:NSUTF8StringEncoding];
+             sscanf(p, "%llu", &num);
+             const char *q = [[labelDec.text stringByAppendingFormat:@"%llu", num] cStringUsingEncoding:NSUTF8StringEncoding];
+             sscanf(q, "%llu", &number);
+      }
       labelDec.text = [NSString stringWithFormat:@"%llu", number];
 }
```

まだ『1<<1=========』が「256」というような「=」の連続押しには対応していない

10. 「uint64CalculatorViewController.{h,m}」を編集、⌘Sで保存。⌘Rで実行、確認。

正整数電卓の制作(7)

「=」ボタンの連続押しに対応

```
$ diff -u uint64CalculatorViewController.h~ uint64CalculatorViewController.h
 @interface uint64CalculatorViewController : UIViewController {
       IBOutlet UILabel *labelDec;
-      uint64_t lhsNumber;
+      uint64_t lhsNumber, rhsNumber;
       NSString *stringOp;
 }
 - (IBAction)buttonPressed:(UIButton *)sender;
```

```
$ diff -u uint64CalculatorViewController.m-20101122175050 uint64CalculatorViewController.m
              number = -number;
       }
       else if ([sender.titleLabel.text isEqual:@"C"]) {
+             if (!number) {
+                   rhsNumber = 0;
+                   stringOp = nil;
+             }
              number = 0;
       }
       else if ([sender.titleLabel.text isEqual:@"⌫"]) {
@@ -79,35 +83,66 @@
              stringOp = @"^";
       }
       else if ([sender.titleLabel.text isEqual:@"="]) {
+             if ([stringOp isEqual:@"+="])
+                   number += rhsNumber;
+             : (中略:「-=」「*=」「/=」「%=」「<<=」「>>=」「&=」「|=」「^=」についても同様に処理する)
+             else
              if ([stringOp isEqual:@"+"]) {
-                   number = lhsNumber + number;
+                   number = lhsNumber + (rhsNumber = number);
+                   stringOp = @"+=";
              }
              : (中略:「-」「*」「/」「%」「<<」「>>」「&」「|」「^」についても同様に処理する)

       }
       else {
```

11. 「uint64CalculatorViewController.{h,m}」を編集、⌘Sで保存。⌘Rで実行、確認。

正整数電卓の制作(8)

8進数、16進数の入出力に対応

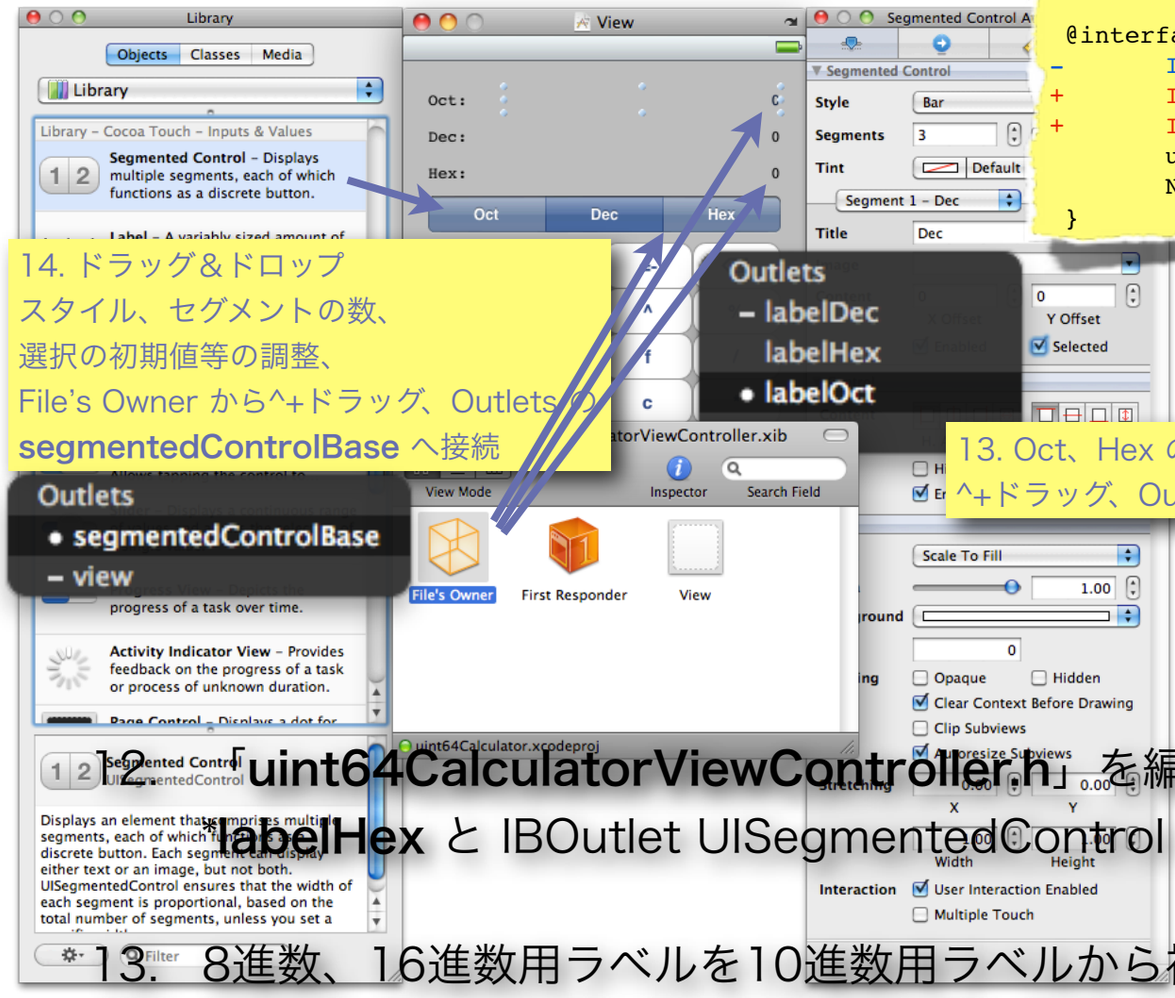
```
$ diff -u uint64CalculatorViewController.h~ uint64CalculatorViewController.h
 #import <UIKit/UIKit.h>

 @interface uint64CalculatorViewController : UIViewController {
-       IBOutlet UILabel *labelDec;
+       IBOutlet UILabel *labelOct, *labelDec, *labelHex;
+       IBOutlet UISegmentedControl *segmentedControlBase;
        uint64_t lhsNumber, rhsNumber;
        NSString *stringOp;
 }
```

12. IBOutlet UILabel ***labelOct***, ***labelHex*** と IBOutlet UISegmentedControl ***segmentedControlBase*** を追加

13. Oct、Hex のためのラベルを複製し、File's Owner から ^+ドラッグ、Outlets の **labelOct、labelHex** へ接続

14. ドラッグ＆ドロップ
スタイル、セグメントの数、
選択の初期値等の調整、
File's Owner から^+ドラッグ、Outlets の **segmentedControlBase** へ接続



12. 「**uint64CalculatorViewController.h**」を編集して、IBOutlet UILabel ***labelOct**, ***labelHex** と IBOutlet UISegmentedControl ***segmentedControlBase** を追加

13. 8進数、16進数用ラベルを10進数用ラベルから複製し、File's Owner から UILabel へコントロールキー+ドラッグで **labelOct, labelHex** に Outlet

14. Segmented Control の配置、⌘1でスタイルやセグメントの数、セグメント1の選択状態を調整、 File's Owner から UISegmentedControl へコントロールキー+ドラッグで **labelOct, labelHex** に Outlet

正整数電卓の制作(9)

8進数、16進数の入出力に対応

```
$ diff -u uint64CalculatorViewController.m~ uint64CalculatorViewController.m
                number = 0;
        }
        else if ([sender.titleLabel.text isEqual:@"⌫"]) {
-               const char *p = [[labelDec.text substringToIndex:labelDec.text.length-1] cStringUsingEncoding:NSUTF8StringEncoding];
-               if (p[0] != '\0') sscanf(p, "%llu", &number); else number = 0;
+               const char *p;
+               switch (segmentedControlBase.selectedSegmentIndex) {
+               case 0:
+                       p = [[labelOct.text substringToIndex:labelOct.text.length-1] cStringUsingEncoding:NSUTF8StringEncoding];
+                       if (p[0] != '\0') sscanf(p, "%llo", &number); else number = 0;
+                       break;
+               case 1:
+                       p = [[labelDec.text substringToIndex:labelDec.text.length-1] cStringUsingEncoding:NSUTF8StringEncoding];
+                       if (p[0] != '\0') sscanf(p, "%llu", &number); else number = 0;
+                       break;
+               default:
+                       p = [[labelHex.text substringToIndex:labelHex.text.length-1] cStringUsingEncoding:NSUTF8StringEncoding];
+                       if (p[0] != '\0') sscanf(p, "%llx", &number); else number = 0;
+                       break;
+               }
        }
        else if ([sender.titleLabel.text isEqual:@"+"]) {
                lhsNumber = number;
@@ -147,12 +160,28 @@
        }
        else {
                uint64_t num;
-               const char *p = [sender.titleLabel.text cStringUsingEncoding:NSUTF8StringEncoding];
-               sscanf(p, "%llu", &num);
-               const char *q = [[labelDec.text stringByAppendingFormat:@"%llu", num] cStringUsingEncoding:NSUTF8StringEncoding];
-               sscanf(q, "%llu", &number);
+               const char *p = [sender.titleLabel.text cStringUsingEncoding:NSUTF8StringEncoding], *q;
+               switch (segmentedControlBase.selectedSegmentIndex) {
+               case 0:
+                       sscanf(p, "%llo", &num);
+                       q = [[labelOct.text stringByAppendingFormat:@"%llo", num] cStringUsingEncoding:NSUTF8StringEncoding];
+                       sscanf(q, "%llo", &number);
+                       break;
+               case 1:
+                       sscanf(p, "%llu", &num);
+                       q = [[labelDec.text stringByAppendingFormat:@"%llu", num] cStringUsingEncoding:NSUTF8StringEncoding];
+                       sscanf(q, "%llu", &number);
+                       break;
+               default:
+                       sscanf(p, "%llx", &num);
+                       q = [[labelHex.text stringByAppendingFormat:@"%llx", num] cStringUsingEncoding:NSUTF8StringEncoding];
+                       sscanf(q, "%llx", &number);
+                       break;
+               }
        }
+       labelOct.text = [NSString stringWithFormat:@"%llo", number];
        labelDec.text = [NSString stringWithFormat:@"%llu", number];
+       labelHex.text = [NSString stringWithFormat:@"%llx", number];
 }
 /* 15. 「uint64CalculatorViewController.m」を編集、⌘Sで保存。⌘Rで実行、確認。
```

さらに、
押しても無効なボタン、8進数入力のときは「8」～「f」、
10進数入力のときは「a」～「f」は無効化したい

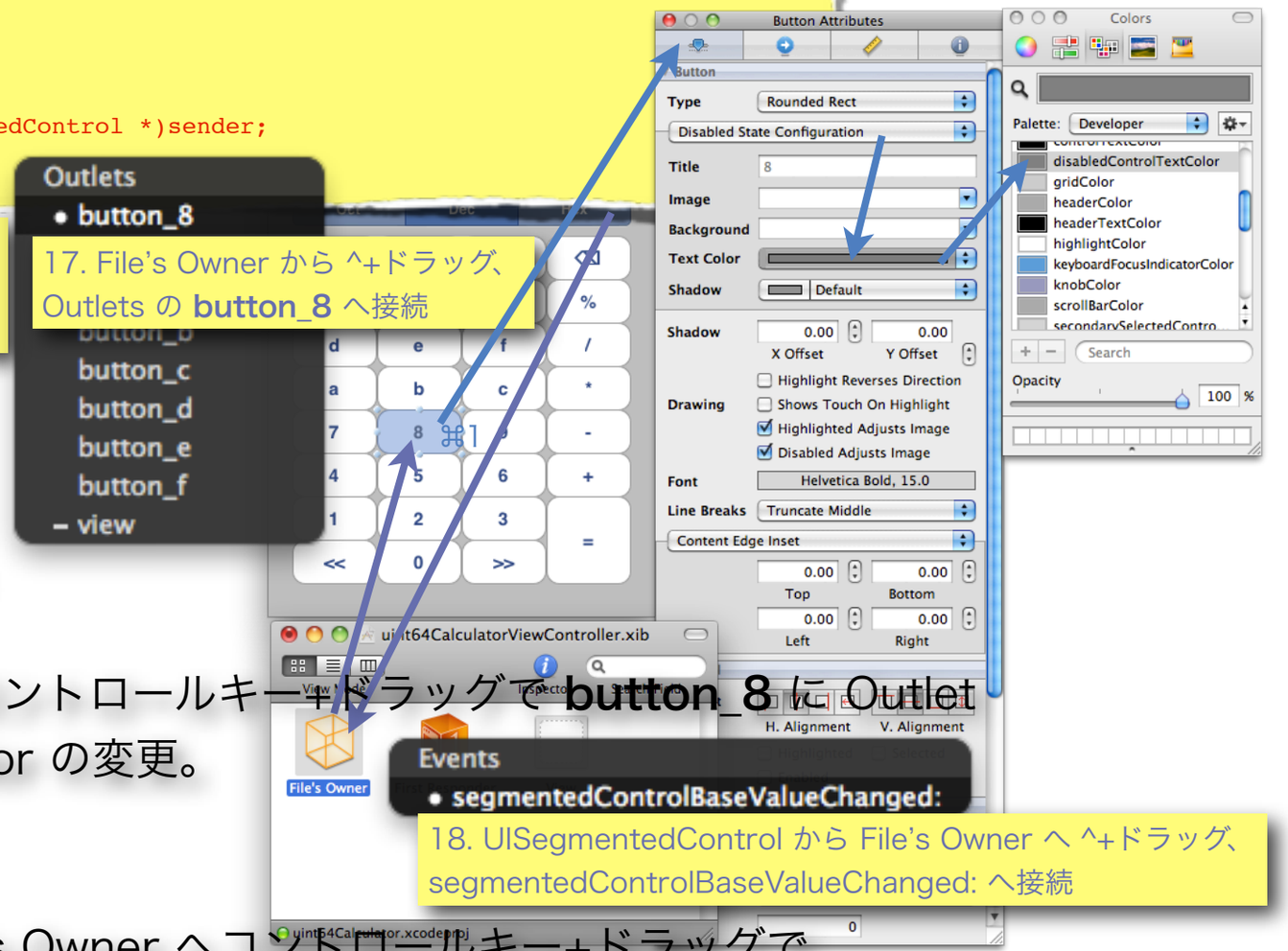
正整数電卓の制作(10)

8進数、16進数の入出力に対応 … 入力モードに応じて数字ボタンを無効に

```
$ diff -u uint64CalculatorViewController.h~ uint64CalculatorViewController.h
@@ -11,10 +11,12 @@
 @interface uint64CalculatorViewController : UIViewController {
        IBOutlet UILabel *labelOct, *labelDec, *labelHex;
        IBOutlet UISegmentedControl *segmentedControlBase;
+       IBOutlet UIButton *button_8, *button_9, *button_a, *button_b, *button_c, *button_d, *button_e, *button_f;
        uint64_t lhsNumber, rhsNumber;
        NSString *stringOp;
 }
 - (IBAction)buttonPressed:(UIButton *)sender;
+- (IBAction)segmentedControlBaseValueChanged:(UISegmentedControl *)sender;

 @end
```

16. IBOutlet UIButton ***button_8～button_f** を追加
(IBAction)**segmentedControlBaseValueChanged:** を追加



16. 「**～ViewController.h**」を編集

17. File's Owner から UIButton へコントロールキー+ドラッグで **button_8** に Outlet
⌘1で Disable State の Text Color の変更。
button_f まで同様に行う。

18. UISegmentedControl から File's Owner へコントロールキー+ドラッグで
segmentedControlBaseValueChanged: に Action

正整数電卓の制作(11)

8進数、16進数の入出力に対応 … 入力モードに応じて数字ボタンを無効に

```
$ diff -u uint64CalculatorViewController.m~ uint64CalculatorViewController.m
@@ -183,6 +183,26 @@
        labelDec.text = [NSString stringWithFormat:@"%llu", number];
        labelHex.text = [NSString stringWithFormat:@"%llx", number];
 }
+- (IBAction)segmentedControlBaseValueChanged:(UISegmentedControl *)sender
+{
+       switch ([segmentedControlBase selectedSegmentIndex]) {
+       case 0:
+               button_8.enabled = button_9.enabled =
+               button_a.enabled = button_b.enabled = button_c.enabled =
+               button_d.enabled = button_e.enabled = button_f.enabled = NO;
+               break;
+       case 1:
+               button_8.enabled = button_9.enabled = YES;
+               button_a.enabled = button_b.enabled = button_c.enabled =
+               button_d.enabled = button_e.enabled = button_f.enabled = NO;
+               break;
+       default:
+               button_8.enabled = button_9.enabled =
+               button_a.enabled = button_b.enabled = button_c.enabled =
+               button_d.enabled = button_e.enabled = button_f.enabled = YES;
+               break;
+       }
+}

 /*
 // The designated initializer. Override to perform setup that is required before the view is loaded.
@@ -201,12 +221,11 @@
 */


-/*
 // Implement viewDidLoad to do additional setup after loading the view, typically from a nib.
 - (void)viewDidLoad {
        [super viewDidLoad];
+       [self segmentedControlBaseValueChanged:segmentedControlBase];
 }
-*/
```

8進数のときは8〜fは無効に

10進数のときは8〜9は有効に
a〜fは無効に

16進数のときはa〜fは有効に

UI の状態で初期化

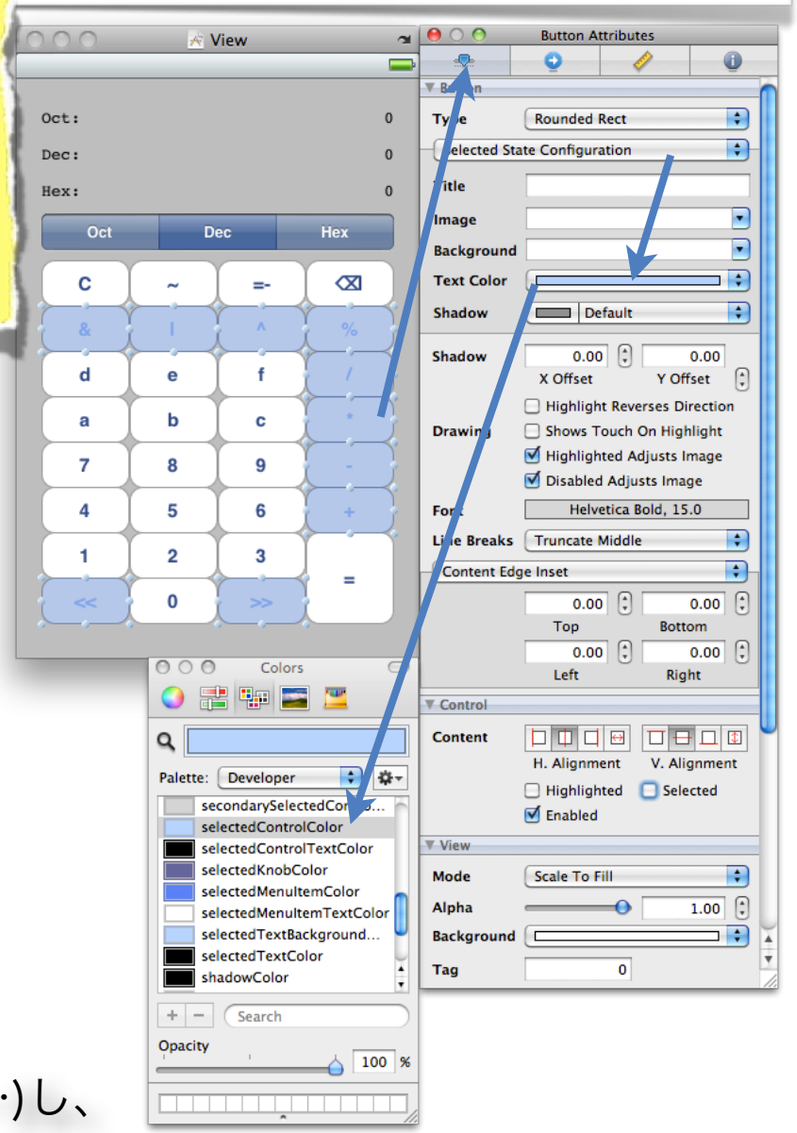
19. 「**uint64CalculatorViewController.m**」を編集、⌘Sで保存。⌘Rで実行、確認。

さらに、
押されて二項演算として選択されたボタンを目立たせたい

正整数電卓の制作(12)

選択されている二項演算子を目立たせる

```
$ diff -u uint64CalculatorViewController.h~ uint64CalculatorViewController.h
@@ -14,6 +14,7 @@
        IBOutlet UIButton *button_8, *button_9,
           *button_a, *button_b, *button_c, *button_d, *button_e, *button_f;
        uint64_t lhsNumber, rhsNumber;
        NSString *stringOp;
+       UIButton *selectedButton;
 }
 - (IBAction)buttonPressed:(UIButton *)sender;
 - (IBAction)segmentedControlBaseValueChanged:(UISegmentedControl *)sender;
```



20. 「uint64CalculatorViewController.h」を編集
    UIButton 型のポインタ **selectedButton** を追加
    ＊ IBOutlet 識別子は付いていないことに注意！

21. 二項演算子のボタン群を選択(クリック、⌘+クリック…)し、
    Selected State Configuration の Text Color を変更。
    この変更を見たい場合は、Control の Selected を**一時的に**無効にする。

正整数電卓の制作(13)

選択されている二項演算子を目立たせる

```
$ diff -u uint64CalculatorViewController.m~ uint64CalculatorViewController.m
@@ -10,6 +10,13 @@

 @implementation uint64CalculatorViewController

+- (void)selectButton:(UIButton *)sender
+{
+      if (selectedButton) selectedButton.selected = NO;
+      if (sender) sender.selected = YES;
+      selectedButton = sender;
+}
+
 - (IBAction)buttonPressed:(UIButton *)sender
 {
       uint64_t number;
@@ -25,6 +32,7 @@
               if (!number) {
                     rhsNumber = 0;
                     stringOp = nil;
+                    [self selectButton:nil];
               }
               number = 0;
       }
@@ -49,51 +57,61 @@
               lhsNumber = number;
               number = 0;
               stringOp = @"+";
+              [self selectButton:sender];
       }
       : (中略:「-」「*」「/」「%」「<<」「>>」「&」「|」「^」についても同様に処理する)
```

selectedButton の選択状態を NO にし、新たな selectedButton の選択状態を YES にするインスタンスメソッド

「C」ボタンを二度押した時は selectedButton は nil

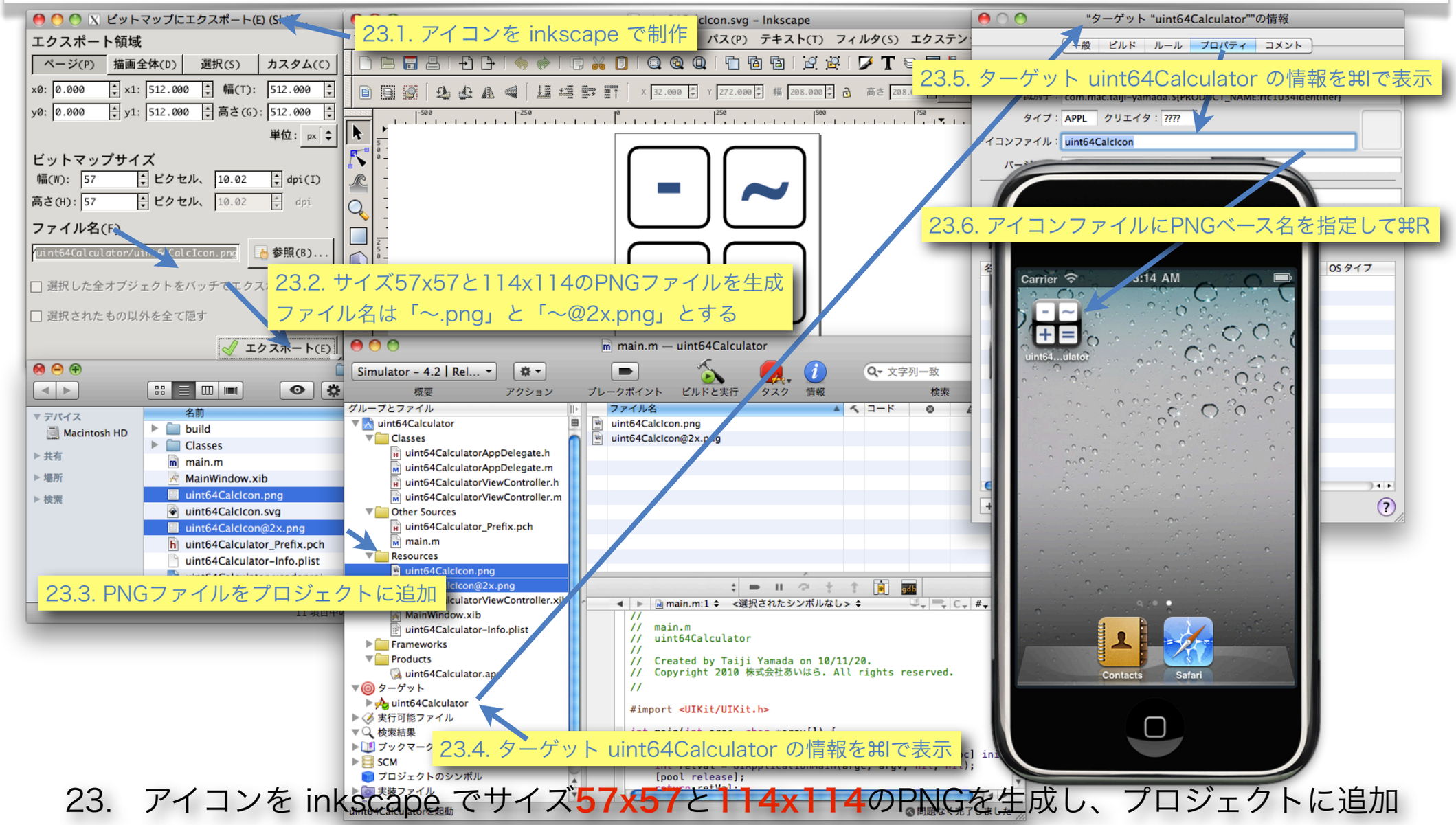
「+」ボタンを押した時には selectedButton は sender

22. 「**uint64CalculatorViewController.m**」を編集、⌘Sで保存。⌘Rで実行、確認。

完成。
しかし、アプリケーションのアイコンが無いので悲しい

正整数電卓の制作(14)

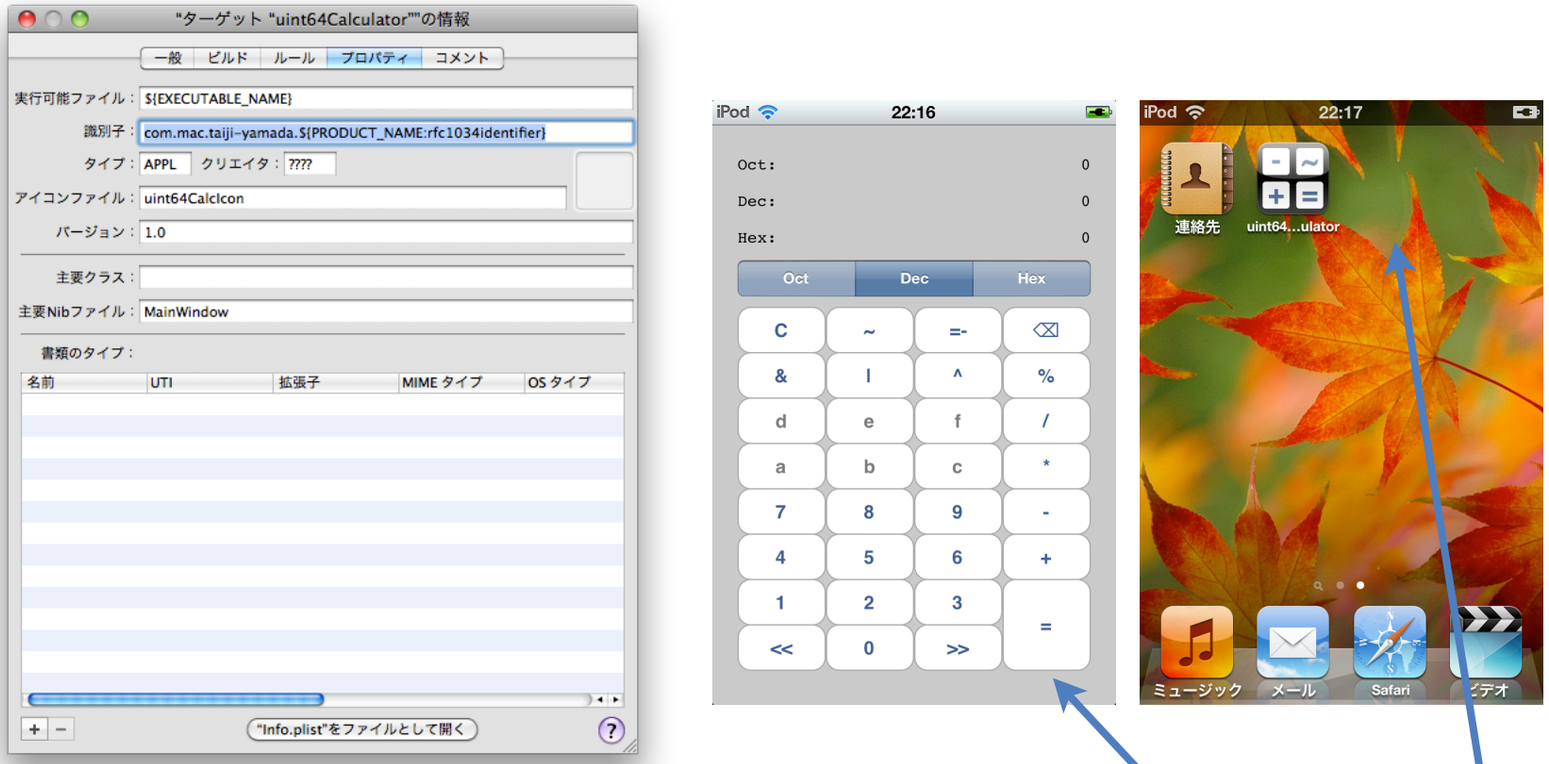
アプリケーションアイコンの追加



23. アイコンを inkscape でサイズ**57x57**と**114x114**のPNGを生成し、プロジェクトに追加
それをターゲット「**uint64Calculator**」の情報のプロパティのアイコンファイルに指定。

正整数電卓の実機での動作

App ID の作成とオーガナイザへの登録



1. 冒頭で示した方法で App ID の作成とオーガナイザへの登録を行う。但し、当然のことながら Apple の一次情報が本稿よりも最新で信頼できる情報源であるので、これに関しては本稿の参考文献や Apple のサイトを参照することを強くお勧めする。

2. 正しく手順を踏めば、Xcode の⌘Rで実機へのインストールと実行が成される。Xcode のオーガナイザでは実機での動作中のスクリーンショットも撮影できる。

プログラミング言語 Objective-C

主な各種ディレクティブ、型、記法(1)

`@interface` *ClassName* `:` *SuperClass* `{`
`  double` *value*`;`
`}`
`+` *classWithValue*`:(double)`*aValue*`;`
`-` *initWithValue*`:(double)`*aValue*`;`
`- (double)`*value*`;`
`- (void)`*setValue*`:(double)`*aValue*`;`
`@property double` *value*`;`
`@end`
	クラスのインターフェースの宣言開始のディレクティブ
	インスタンス変数の宣言
	
	クラスメソッドの宣言（簡易コンストラクタ）
	インスタンスメソッドの宣言（指定イニシャライザ）
	インスタンスメソッドの宣言（ゲッターメソッド）
	インスタンスメソッドの宣言（セッターメソッド）
	アクセサ（インスタンス）メソッドの宣言、上記２行と等価
	宣言または定義終了のディレクティブ

`@implementation` *ClassName*
`+` *classWithValue*`:(double)`*aValue*
`{`
`  return [[[self alloc]` *initWithValue*`:`*aValue*`]`
`  autorelease];`
`}`
`-` *initWithValue*`:(double)`*aValue* `{`
`  if ((self = [super init]))`
`  `*value* `=` *aValue*`;`
`  return self;`
`}`
`- (double)`*value*
`{`
`  return` *value*`;`
`}`
`- (void)`*setValue*`:(double)`*aValue*
`{`
`  `*value* `=` *aValue*`;`
`}`
`@synthesize` *value*`;`
`@end`
	クラスのインターフェースの定義開始のディレクティブ
	クラスメソッドの定義（簡易コンストラクタ）
	
	インスタンスメソッドの定義（指定イニシャライザ）
	
	インスタンスメソッドの定義（ゲッターメソッド）
	
	インスタンスメソッドの定義（セッターメソッド）
	
	アクセサ（インスタンス）メソッドの定義、上記２ブロックと同等
	宣言または定義終了のディレクティブ

プログラミング言語 Objective-C

主な各種ディレクティブ、型、記法(2)

id	オブジェクトへのポインタ型
nil	NULLオブジェクトポインタ、(id)NULL
BOOL	真偽型、char型
NO	偽値、(BOOL)0
YES	真値、(BOOL)1
#import	二度読み防止済み#includeディレクティブ
self	受信側オブジェクトへのポインタ変数
super	継承されている受信側オブジェクトへのポインタ変数
@"文字列"	NSString型オブジェクト定数ディレクティブ
@"文字列*1*" @"文字列*2*" … @"文字列*n*"	NSString型オブジェクト定数ディレクティブ、*n*個のディレクティブで指定された文字列を結合
[*receiver message*]	メッセージ式、セレクタ＝メソッド名は *message*
[*string* isEqual:*@"文字列"*]	単一引数をもつメッセージ式、セレクタは isEqual:
[NSString stringWithCstring:*buffer* encoding:*stringEncoding*]	複数引数をもつメッセージ式、セレクタは stringWithCstring:encoding:
[string withFormat:*@"%@=%g\n", @"文字列", 100.0*]	可変引数をもつメッセージ式、セレクタは withFormat:
receiver.value	ドット記法、アクセサのゲッターメソッドの呼び出し、[*receiver value*]と等価
receiver.value = 100.0	ドット記法、アクセサのセッターメソッドの呼び出し、[*receiver* set*Value*:*100.0*]と等価

まとめ

Objective-C programming
create View-based Application

1. 簡単な正整数電卓を制作した。
2. UI の配置と Outlet, Action が解れば簡単なアプリは作れてしまう。
3. Objective-C とは C の拡張である。C の機能は存分に使用可能
4. View-based Application はアプリの基本となる簡単な構成要素
5. 一次情報源が重要である。Apple のサイトマップから辿りましょう！
6. ドキュメントも同様。ウェブで検索する前に Xcode のヘルプから！

参考文献

Objective-C programming
create View-based Application

1. Apple Inc., “iOSアプリケーションチュートリアル,” 2010. 簡単なiPhoneアプリ作成の紹介。本稿の方が多少シンプル

2. Apple Inc., “iPhone開発ガイド,” 2010. 簡単なiPhoneアプリ作成だけでなく、実機で動作させる為のiOS Developer Programの入会からApp IDの登録、さらには、App Storeでの公開までを紹介

3. Apple Inc., “iPhoneヒューマンインターフェイスガイドライン,” 2010. アイコンのサイズやUIの配色ほか、遵守すべき項目の紹介

4. Apple Inc., “Objective-Cプログラミング言語,” 2009. Objective-C プログラミングの基礎をわかりやすく紹介