Trabalho I: Calculadora

Observação

Este material foi traduzido e adaptado do curso CS 193P - *iPhone Application Development* oferecido pela Universidade de Stanford através do iTunesU. Ele está protegido pela licença *Creative Commons Attribution-Noncommercial-Share*[1].

Objetivo

O objetivo deste trabalho é recriar o programa elaborado no laboratório e então realizar algumas pequenas modificações. É importante que você entenda o que está fazendo em cada passo da recriação para que você esteja preparado para as modificações.

Um objetivo adicional é adquirir experiência na criação de projetos com o XCode e em digitar código do zero. Procure não copiar/colar qualquer código de lugar algum. Digite cada linha e veja o que o XCode faz enquanto você digita.

Lista de Tarefas Requeridas

1. Implemente a calculadora vista nas aulas 1 e 2. A parte de *autolayout* do final das aulas vale uma nota extra. Mesmo assim sugiro que tente, pois conseguir um bom *autolayout* requer experiência. Se você não fizer o *autolayout*, certifique-se de que tudo será posicionado de forma que fique visível em todos os iPhones pelo menos.

2. A calculadora já funciona com números de ponto flutuante (por exemplo, se você digitar 3 ↵ 4 ÷, o resultado será 0.75). Contudo, não há um meio de digitar um número de ponto flutuante (cuidado porque 19.2.3.4 não é um número de ponto flutuante válido!). Para que esta tarefa seja completada, um botão com o ponto deve ser adicionado à *interface*.

3. Adicione as operações:
   a. **sin**: calcula o seno do operando no topo da pilha;

1 http://creativecommons.org/licenses/by-nc-sa/3.0/us/

b. **cos**: calcula o cosseno do operando no topo da pilha;
c. **pi**: calcula o valor de pi. Por exemplo, 3 ↲ pi x deve apresentar três vezes o valor de pi no visor da calculadora.

4. Adicione um *UILabel* à *interface*. Ele deve mostrar um histórico de cada entrada do usuário, seja operando ou operador. Coloque-o em uma localização adequada da *interface*.

5. Adicione um botão C que limpa tudo (o visor, o *UILabel* do histórico, etc.). A calculadora deve voltar ao mesmo estado de quando foi iniciada quando este botão for pressionado.

6. Evite os problemas listados na seção de avaliação deste documento. Esta lista crescerá a cada novo trabalho. Certifique-se de verificá-la novamente a cada trabalho.

Dicas

1. O método *rangeOfString (substring : String)* da classe *String* pode ser útil para a tarefa de aceitar a digitação de números de ponto flutuante deste trabalho. Ele retorna um *Optional*. Se o argumento não pode ser encontrado no receptor, ele retorna *nil* (caso contrário não se preocupe com o que ele retorna por enquanto).

2. A tarefa de aceitar a digitação de números de ponto flutuante pode ser implementada utilizando apenas uma linha de código. Veja que isso é uma dica, não uma tarefa requisitada. Mesmo assim, veja se você consegue descobrir como implementar desta forma (uma chave em uma só linha não é considerada uma linha de código).

3. Cuidado com o caso em que o usuário inicia a digitar um número pelo ponto decimal, por exemplo ".5", sem as aspas. Pode ser que a sua solução já funcione para este caso, mas é bom testar.

4. *sin ()* e *cos ()* são funções disponíveis em Swift.

5. O valor de pi pode ser obtido com a expressão *M_PI*. Por exemplo: *let pi = M_PI*

6. Você pode implementar pi como um operando ou como uma operação (i.e. um novo tipo de operação que não retira operandos da pilha e retorna um valor). A decisão é sua. Mas, de qualquer forma, seria interessante ser capaz de adicionar outras constantes na calculadora com o

mínimo de código.

7. Você pode querer inserir o botão C fora da grade com os botões dos dígitos e operações (já que não há espaço para ele nesta grade). Isso é um bom desafio às suas capacidades de trabalhar com *autolayout*.

8. Economia de linhas de código é algo que tem valor em programação. A maneira mais fácil de garantir que não há erros em uma linha de código é não ter a linha de código de forma alguma. Este trabalho requer apenas umas poucas linhas de código. Assim sendo, se você se encontrar digitando código demais, provavelmente está no caminho errado.

Aprendizado

Eis uma lista parcial de conceitos que este trabalho pretende fazer com que você ganhe prática ou demonstre o seu conhecimento:

1. XCode
2. Swift
3. *Target*/*Action*
4. *Outlets*
5. *UILabel*
6. *UIViewController*
7. Classes
8. Funções e propriedades (variáveis de instância)
9. *let* e *var*
10. *Optional*
11. Propriedades computadas e armazenadas
12. *String* e *Array*
13. *Autolayout* (nota extra)

Avaliação

Em todos os trabalhos realizados com Swift, escrever código com qualidade que compila sem avisos ou erros, e então testar as aplicações resultantes até que elas funcionem corretamente é o objetivo.

Aqui estão as razões mais comuns para que trabalhos sejam negativados:

- o projeto não compila;
- o projeto não compila sem avisos;
- um ou mais itens da lista de tarefas não foi cumprido;
- um conceito fundamental não foi entendido;
- o código é visivelmente desorganizado e difícil de ler (por exemplo, a indentação não é consistente, etc.);
- a solução é difícil (ou impossível) para alguém que está lendo o código entender devido à falta de comentários, nomes ruins de variáveis ou métodos, estrutura de solução pobre, métodos longos, etc.

Com frequência os alunos perguntam "quanto comentário deve ser adicionado ao código?". A resposta é "o suficiente para que o seu programa seja compreensível por qualquer pessoa que o leia". Você pode assumir que o leitor conhece o SDK, mas não pode assumir que ele sabe a (ou uma) solução para o problema.

Lista de Tarefas com Nota Extra

A lista de tarefas que vale nota extra é uma oportunidade de expandir o que foi aprendido. É altamente recomendado que você tente fazer as tarefas desta lista para que possa aprender o máximo possível.

1. Implemente um botão de apagar para que o usuário possa corrigir o que ele digitou antes de inserir na pilha. Isso não é para ser um botão de desfazer, ou seja, se o usuário pressionar a operação errada, sinto muito! Cabe a você decidir o que fazer se o usuário pressionar o botão de apagar e todo o número for apagado do visor. Deixar o visor em branco não é uma boa saída.
As funções globais *countElements* e *dropLast* serão de grande ajuda nesta tarefa. Ambas podem pegar uma *String* como seus únicos argumentos. A primeira é o que você geralmente pensaria para o comprimento e a outra remove o último caractere da *String* (e retorna o resultado após a remoção). Você pode achar estranho se clicar nesta função com a tecla Alt selecionada. Os tipos dos argumentos e valores de retorno necessitam de explicação adicional. Aqui não temos espaço para isso. Resumindo, *String* é uma coleção de caracteres que podem ser indexados e partidos em sub-coleções de caracteres. Por isso *countElements* (que recebe uma coleção como parâmetro) e *dropLast* (que recebe uma coisa divisível) funcionam com *String*.

2. Quando o usuário pressionar um botão de operação, coloque um = no final do *UILabel* que foi pedido para exibir o histórico na lista de tarefas requeridas. Assim, o usuário poderá saber se o número no visor da calculadora é o resultado de uma operação ou um número que o usuário acabou de digitar. Cuidado para não exibir múltiplas ocorrências de = no *UILabel*.

3. Adicione uma operação +/- que muda o sinal do número no visor. Muito cuidado com esta operação. Se o usuário estiver no meio da digitação de um número, você provavelmente desejará que ele mude o sinal do número e possa continuar a digitar. Não force um enter como outras operações o fazem. Por outro lado, se o usuário não estiver no meio da digitação de um número, então esta operação funciona exatamente como qualquer outra operação unária (por exemplo, cos).

4. Mude a variável de instância computada *displayValue* para ser um *Optional Double* ao invés de apenas um *Double*. Seu valor dever ser *nil* se o conteúdo de *display.text* não puder ser interpretado como um *Double* (você precisará usar a documentação para entender o uso de *NSNumberFormatter*). Uma atribuição de seu valor com *nil* deve resultar em limpar o visor da calculadora.

5. Use autolayout para fazer com que sua calculadora fique boa em todos os tipos diferentes de iPhones em qualquer orientação de tela (não se preocupe com os iPads por enquanto). Da mesma forma que arrastamos bordas visor na *interface* gráfica com a tecla Ctrl pressionada para os cantos da tela para posicioná-lo, você pode fazer o mesmo entre os *UILabel* (e/ou o botão C) para ajustar o espaçamento vertical/horizontal relativo entre eles. Use as linhas de grade azuis! Uma boa ideia é reiniciar todo o seu *autolayout* (isso pode ser feito pressionando um botão pequeno no canto inferior direito do *interface builder*). Depois, então, arraste os objetos com a tecla Ctrl pressionada para adicionar restrições nos objetos que não são parte da grade do teclado e os botões de operação, então use os botões no canto inferior direito para ajustá-los (após eles terem sido movidos a locais relativos aos *UILabel*, etc., usando as linhas de grade azuis, é claro!).