Projet 7

CountOnMe

Parcours DA IOS
Daniel BENDEMAGH

Objectif

Pour une calculatrice qui effectue les opérations de base + et -, il m'a été demandé de compléter les opérations élémentaires et d'ajouter des bonus supplémentaires.

Bonus

Bonus 1 – Multiplication et division

Les opérateurs * et / sont prioritaires et doivent être calculés avant les opérateurs + et -. Pour cela, j'ai créé la fonction **priorCalculation** qui effectue les calculs avant les opérateurs + et -.

La fonction est appelée au début de la fonction **calculateTotal** :

Calculation.swift

```swift
func calculateTotal() -> Double {
    if !isExpressionCorrect {
        return 0
    }

    priorCalculation()
```

Calculation.swift

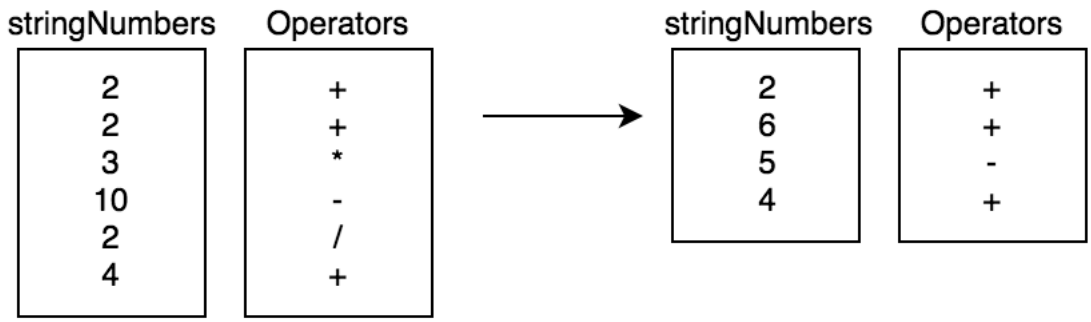
```swift
// Calculate * and / operators in priority
// The calculation order must be from beginning to the end
private func priorCalculation() {
    var result: Double = 0
    let priorOperators = "*/"
    var stringNumber: String

    var i = 0

    while i < stringNumbers.maxIndex() {
        // Find * or / operators
        var currentNumber = stringNumbers[i].double()
        while priorOperators.contains(operators[i.nextIndex()]) {
            // Loop on each * or / operators
            stringNumber = stringNumbers[i.nextIndex()]
            let nextNumber = stringNumber.double()
            result = calculate(currentNumber, nextNumber, operators[i.nextIndex()])

            // Put result in first number
            stringNumbers[i] = String(result)
            currentNumber = result
            // Remove next number and operator
            stringNumbers.remove(at: i.nextIndex())
            operators.remove(at: i.nextIndex())

            if i == stringNumbers.maxIndex() {
                break
            }
        }
        i += 1
    }
}
```

Le principe est de rechercher dans le tableau **Operators** les opérateurs * ou /, une fois trouvé faire le calcul et ne laisser que le résultat, de manière à ce que toutes les opérations de multiplication et de division soit traitées.

Exemple :

Les étapes sont :

1 – Rechercher les opérateur * ou /
2 – Faire le calcul entre le nombre courant, l'opérateur suivant et le nombre suivant
3 - Remplacer le premier nombre par le résultat
4 - Supprimer le nombre suivant.
5 – Tant que l'opérateur suivant est * ou / revenir à l'étape 2



Why While

L'utilisation d'**array.remove** dans une boucle classique **for** n'est pas possible, le nombre d'index est modifié et il y a un risque de plantage.

La solution aurait été de faire une boucle avec un ordre inversé :

```
for (i, stringNumber) in stringNumbers.enumerated().reversed() {
    //
    // Remove next number and operator
    stringNumbers.remove(at: i)
    operators.remove(at: i)
    //
    }
}
```

Mais dans le cas d'opérateurs * et / consécutifs, les calculs qui sont fait à l'envers faussent les résultats.

Exemple avec 10/20*30.

Dans une boucle du début à la fin :

10/20 = 0,5
0,5*30 = 15

Dans une boucle à l'ordre inversé :

20*30 = 600
10/600 = 0,016666667

l'ordre de calcul doit obligatoirement être du début à la fin.

Voilà pourquoi j'utilise while qui prends les calculs dans l'ordre et qui me permet de contrôler le dernier index qui évolue avec array.remove.

J'ai ajouté un **test unitaire** pour m'assurer que, en cas d'évolution du code, le calcul consécutif s'effectuera toujours dans le bon ordre :

CalculationTestCase.swift

```
175     // Ensure that consecutive * and / operators are calculated in order from beginning to the end
        func testGivenNumber10_WhenDivideBy20ANDMultiplyBy30_ThenResultIs15() {
177         calculation.addNumber(1)
178         calculation.addNumber(0)
179         calculation.addOperator(stringOperator: "/")
180         calculation.addNumber(2)
181         calculation.addNumber(0)
182         calculation.addOperator(stringOperator: "*")
183         calculation.addNumber(3)
184         calculation.addNumber(0)
185         XCTAssertEqual(calculation.calculateTotal(), 15)
186     }
```

Bonus 2 - Calcul en nombre flottant

Conversion de stringNumber en Double

Dans String.Swift, j'ai créé une fonction **double** qui remplace la virgule par un point et qui converti le nombre en Double.

"123,45" -> 123.45

String.swift

```
extension String {
    func double() -> Double {
        if let value = Double(self.split(separator: ",").joined(separator: ".")) {
            return value
        }

        return 0
    }
}
```

Exemple d'utilisation dans la fonction **calculateTotal** :

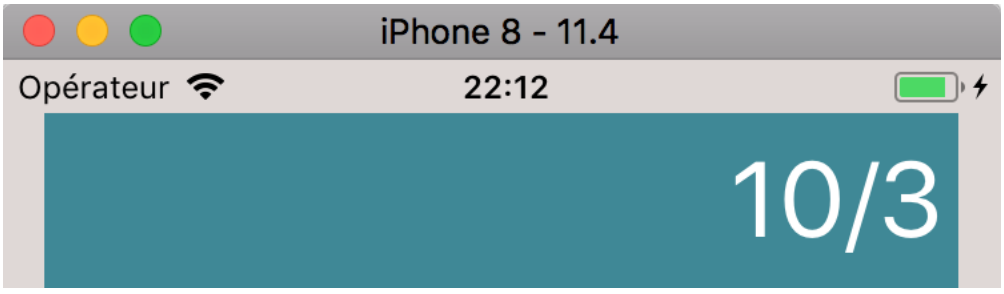
Calculation.swift

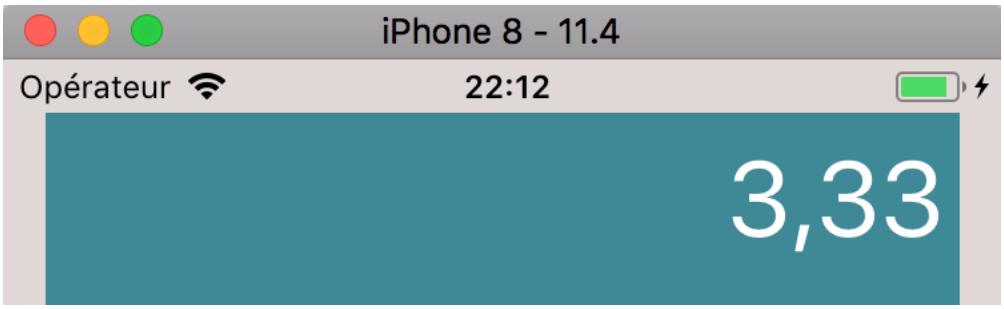
```
for (i, stringNumber) in stringNumbers.enumerated() {
    let number = stringNumber.double()
    total = calculate(total, number, operators[i])
}
```

Affichage avec une précision de 2 décimales

Le résultat du calcul est affiché avec une précision de 2 décimales après la virgule.

```
extension Double {
    func fraction2() -> String {
        let formatter = NumberFormatter()
        formatter.numberStyle = .decimal
        formatter.roundingMode = NumberFormatter.RoundingMode.halfUp
        formatter.usesGroupingSeparator = false
        formatter.maximumFractionDigits = 2

        let value: Double = self
        let nsnumberValue: NSNumber = NSNumber(value: value)

        if let roundedValue = formatter.string(from: nsnumberValue) {
            return roundedValue
        }

        return ""
    }
}
```





Bonus 3 - Réutilisation du total

A la fin de la fonction **calculateTotal**, le tableau **stringNumbers** est effacé. Le total est ensuite ajouté au tableau. Pour indiquer que le total est actuellement affiché, **textIsLastTotal** est passé à True :

Calculation.swift

```
// Reuse the last result
stringNumbers[0] = total.fraction2()
textIsLastTotal = true
```

En fonction de ce qui est tapé ensuite, le total est réutilisé ou non.

Un numéro est tapé, le total est effacé :

123 (appuie sur 4) -> 4

Un opérateur est tapé, le total est conservé :

123 (appuie sur +) -> 123+

C'est dans la fonction **addNumber** que le contrôle est effectué.

Si **textIsLastTotal** est à True, stringNumber est effacé.
Sinon, le numéro est ajouté à stringNumber

```
if textIsLastTotal {
    // Remove last total
    stringNumberMutable = ""
} else {
    stringNumberMutable = stringNumber
}
```

Un opérateur qui est tapé :

Le total est conservé.

Si c'est un nombre, le total est effacé.

Bonus 4 - Effacement des entrées

J'ai ajouté 3 boutons qui permettent de gérer les effacements :

AC : All Clear (tout effacer)
CE : Clear Entry (Effacer le dernier nombre ou le dernier opérateur)
Backspace (Flèche gauche) : Effacer le dernier caractère entré

Pour le bouton **Backspace**, j'ai ajouté dans **Assets** une image de flèche gauche :

Et j'ai attribué l'image au bouton.

Bouton AC

Il fait appel à la fonction **Clear** :

Calculation.swift

```
func clear() {
    stringNumbers = [String()]
    operators = ["+"]
}
```

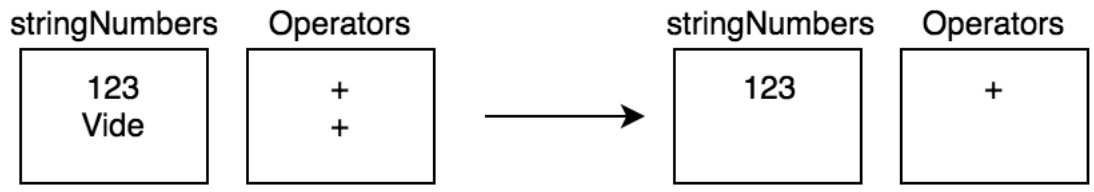
Les boutons **CE** et **Backspace** utilisent la fonction **eraseNumber** avec en paramètre une énumération pour indiquer le type d'effacement à effectuer :

Calculation.swift

```
enum EraseType {
    case Backspace
    case ClearEntry
}
```

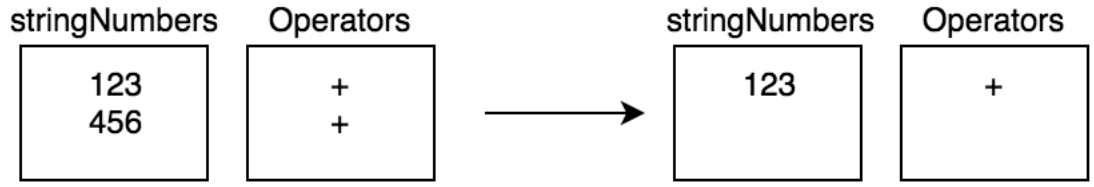
Logique

Si Le dernier **stringNumbers** est vide, retirer le dernier **stringNumbers** et le dernier **Operators.**

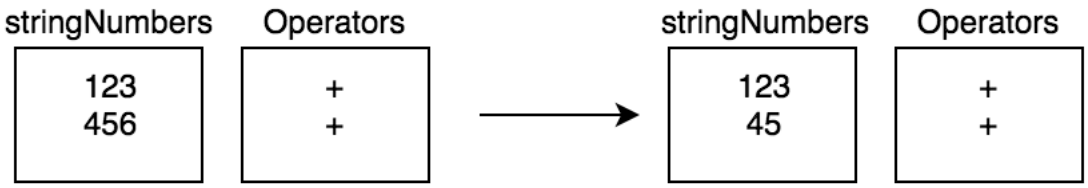


Sinon, agir en fonction du type d'effacement.

Clear Entry : Effacer le dernier nombre



Backspace : Effacer le dernier caractère du dernier nombre



Calculation.swift

```swift
// Backspace or Clear Entry
func eraseNumber(eraseType: EraseType) {
    if var stringNumber = stringNumbers.last {
        if stringNumber.isEmpty {
            if stringNumbers.count > 1 {
                stringNumbers.removeLast()
                operators.removeLast()
            }
        } else {
            switch eraseType {
            case .Backspace:
                stringNumber.removeLast()
                stringNumbers[stringNumbers.maxIndex()] = stringNumber
            case .ClearEntry:
                stringNumbers[stringNumbers.maxIndex()] = ""
            }
        }
    }
}
```

Bonus 5 – Icône application

Création d'un icône placé dans Assets/AppIcon :