

Calculator

Earvin Kayonga

earvin.kayonga@gmail.com

École d'Ingénieurs Généraliste du Numérique - EFREI

ABSTRACT

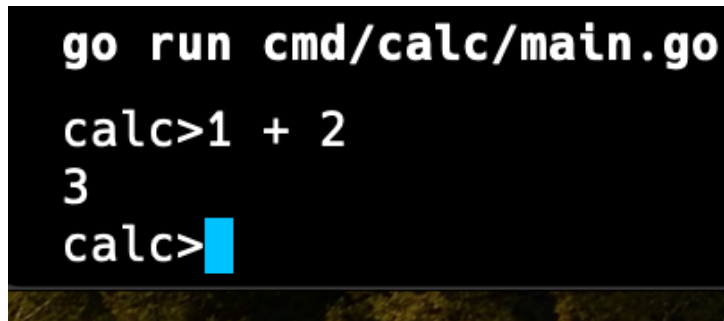
Le projet propose de compléter l'implementation des fonctions basiques d'une calculatrice en ligne de commande.

Contents

1 Introduction	1
2 Sujet	1
2.1 Évaluation	2
2.2 Bonus:	2
2.3 Remarques	2
3 Rendu:	3
4 Documentation:	3

1 Introduction

Calculator: Le programme doit accepter une string et afficher le résultat du calcul.



```
go run cmd/calc/main.go
calc>1 + 2
3
calc>
```

Figure 1: Exemple d'utilisation

2 Sujet

Vous êtes chargé(e) de développer et de tester les fonctionnalités d'une calculatrice en ligne de commande. Suivez les instructions ci-dessous pour implémenter les fonctionnalités et les tests unitaires requis

- **Ajout de la Fonctionnalité POWER** (calc/operation.go)

Implémentez une fonctionnalité pour l'opération de puissance (exponentiation) qui permet de calculer $2^2=4$.

Ajoutez le test unitaire pour cette fonctionnalité afin de vérifier son bon fonctionnement.

- **Correction du Test Test_ConvertInputToValue** (calc/input_test.go)

Analysez le code et implémentez le test unitaire Test_ConvertInputToValue pour garantir qu'il valide correctement la conversion des entrées utilisateur en valeurs numériques exploitables.

- **Correction du Test Test_Calculate** (calc/calc.go)

Corrigez la fonction Calculate pour que le test unitaire Test_Calculate passe.

- **Correction du Test Test_Run**

Réimplémentez la fonction Run pour que tous les cas de test dans le test unitaire Test_Run passe.

- **Ajout de Tests pour les Fonctions Multiply et Divide** (calc/operation_test.go)

Écrivez et ajoutez des tests unitaires pour les fonctions de multiplication (Multiply) et de division (Divide) afin de valider leur exactitude.

- **Ajout de la Fonctionnalité de Mémoire**

Implémentez une fonctionnalité de mémoire qui permet de stocker les résultats précédents des calculs.

2.1 Évaluation

L'application sera évaluée sur les critères suivants:

- **Fonctionnalité:** Respect des consignes et implémentation de toutes les fonctionnalités et leur **tests**.
- **Qualité du code:** Respect des bonnes pratiques de programmation et clarté du code.
- **Assez Documenté:** Facilement déployable et avec un README.md.

2.2 Bonus:

- Dockerisation: Lancement via Docker

2.3 Remarques

L'application doit être documentée et commentée. Le code source doit être propre, lisible et **testé**.

Testez et débugez votre code régulièrement. Améliorez l'interface utilisateur en ajoutant des fonctionnalités supplémentaires.

Ce PDF est accompagné du fichier go-calc.bundle.

```
git clone go-calc.bundle calc1 # pour accéder au code
Cloning into 'calc1'...
Receiving objects: 100% (15/15), done.
```

```
ls calc1
README.md calc      cmd      go.mod
```

```
cd calc1
go run cmd/calc/main.go # pour lancer l'application ligne de commande.
>quit # pour quitter
```

```
go test ./... # pour lancer les tests.
```

```
git add --all
git commit -m "message" # committez autant que vous voulez
```

```
git bundle create <prenom.nom>.bundle HEAD main # pour créer votre bundle de rendu
```

3 Rendu:

Vous allez devoir commit votre code, créer un fichier .bundle contenant vos changements, puis me l'envoyer par Teams ou mail earvin@earvinkayonga.com

Voici les instructions pour créer le fichier .bundle.

```
git add --all
git commit -m "message" # committez autant que vous voulez
git bundle create <prenom.nom>.bundle HEAD main # pour
```

4 Documentation:

Exemple d'utilisation de Go: <https://gobyexample.com/>