

Clément Rouvroy

MP2I – Planche de Colle n° 3

Introduction

Cette planche de colle porte sur les thématiques abordées lors du début du second semestre :

- La discipline de programmation : spécification, préconditions et postconditions,
- La programmation défensive et les assertions,
- La pile d'appels, la preuve et l'analyse de complexité de programmes récursifs,
- L'introduction à la programmation en OCaml.

Les exercices sont de difficulté variable et ont pour objectif de vérifier votre compréhension et votre maîtrise de ces notions.

Exercices

Cours

Exercice 1 (Difficulté: 1)

Reprécisez les notions de **précondition** et de **postcondition**.

- Donnez un exemple de fonction (dans le pseudo-code de votre choix) avec une précondition et une postcondition explicites.

Exercice 2 (Difficulté: 1)

Reprécisez la notion de **pile d'appels**.

Exercice 3 (Difficulté: 2)

Expliquez la différence entre un code **naïf** et un code **défensif**. Illustrez votre propos à l'aide d'un exemple concret (par exemple que ferait un programme défensif qui prend en entrée un type *?).

Exercice 4 (Difficulté: 3)

Décrivez les mécanismes fondamentaux de la **pile d'appels**.

- Expliquez ce qui se passe lors de l'appel d'une fonction (création d'une nouvelle pile d'exécution, ...).
- Montrez comment la pile évolue lors de l'appel de plusieurs fonctions successives ou récursives.

Exercices de colle

Exercice 5 (Difficulté: 2)

Considérez la fonction suivante (pseudo-code ou langage au choix) qui calcule le minimum dans un tableau :

```
function minArray(A):  
    minimum = A[0]  
    for i in 1 to length(A)-1:  
        if A[i] < minimum:  
            minimum = A[i]
```

```
return minimum
```

- Proposez une **spécification** formelle (précondition et postcondition).
- Indiquez en quoi la precondition peut être vérifiée par une assertion.

Exercice 6 (Difficulté: 3)

Prouvez la correction de la fonction `fact(n)` qui calcule la factorielle de `n` par récursion.

```
function fact(n):  
    if n <= 1:  
        return 1  
    else:  
        return n * fact(n - 1)
```

- Établissez la **spécification** (précondition et postcondition).
- Décrivez la **preuve par induction** de la postcondition.

Exercice 7 (Difficulté: 3)

- Analysez la **complexité** temporelle de la fonction `fact(n)` ci-dessus.
- Faire de même pour la complexité spatiale.

Exercice 8 (Difficulté: 3)

Écrivez une fonction `fibonacci(n)` qui renvoie le n -ième terme de la suite de Fibonacci en utilisant la récursion naïve.

- Déterminez la complexité de cette version récursive.
- Proposez une version **optimisée** (par exemple, avec mémoïsation ou itération) et sa complexité.

Exercice 9 (Difficulté: 2)

- Écrivez une fonction `carre` (en OCaml) qui prend un entier `n` et renvoie son carré.
- Donnez la **signature de cette fonction**.
- Ecrire la fonction `fibonacci`, en donner sa signature.

Exercice 10 (Difficulté: 3)

- En OCaml, écrivez la fonction `fact_defensif` qui calcule la factorielle de `n`, mais qui :
- Vérifie en entrée que `n` n'est pas négatif.
 - Calcule la factorielle de manière récursive.

Ecrire des *tests* qui couvrent l'ensemble des cas possibles de votre code. Justifier.

Exercice 11 (Difficulté: 2)

En C, écrivez une fonction `safeAlloc(size_t n)` qui alloue dynamiquement un tableau de `n int` et renvoie son adresse.

- Vérifiez la precondition sur l'entrée avec `assert`.
- Vérifiez que vous ne renvoyez pas quelque chose qui n'a pas pu être alloué, toujours avec `assert`.

Exercice 12 (Difficulté: 3)

En C, écrivez une fonction récursive `isPalindrome` pour tester si une chaîne de caractères est un palindrome.

- Pensez à gérer le cas `NULL`.
- Proposez une preuve rapide de terminaison (le cas de base, la réduction de la taille de la chaîne).
- Discutez de la complexité de la solution.

Exercice 13 (Difficulté: 2)

En C, écrivez une fonction `copyString` qui recopie une chaîne source dans une destination, en vérifiant notamment que la destination est assez grande.

- Définissez la spécification (précondition, postcondition).
- Expliquez comment vous avez géré le cas où `destSize < strlen(src) + 1`.

Exercice 14 (Difficulté: 2)

En C, écrivez un programme simple qui lit un entier `n` au clavier et calcule **récur-
sivement** le **facteur** de `n` (sans vérification), puis améliorez-le de façon à :

- Vérifier que `n` est bien dans l'intervalle acceptable (par ex. $0 \leq n \leq 12$ pour éviter les débordements d'entiers `int`).
- Gérer proprement les cas d'erreur (au-delà d'une valeur limite, affichage d'un message, etc.).

Exercice 15 (Difficulté: 3)

En C, écrivez une fonction `sumArray` qui calcule la somme des éléments d'un tableau de `n` entiers, en mode récursif.

- Utilisez des assertions pour vérifier que l'entrée n'est pas vide.
- Calculez sa complexité et décrivez le coût en mémoire.

Petits exercices OCaml

Exercice 16 (Difficulté: 1)

Écrivez une fonction `compose` qui prend deux fonctions `f` et `g` et renvoie la composition de ces fonctions. En particulier, la sortie est bien une fonction et non le résultat de l'application de la composition sur une entrée.

Exercice 17 (Difficulté: 1)

Écrivez une fonction `map` qui applique une fonction à tous les éléments d'une liste et renvoie la liste résultante. N'utilisez pas la fonction `List.map` de la bibliothèque standard.

Exercice 18 (Difficulté: 1)

Écrivez une fonction `filtre` qui prend un prédicat (fonction renvoyant un booléen) et une liste, et renvoie une nouvelle liste contenant uniquement les éléments qui satisfont le prédicat. N'utilisez pas la fonction `List.filter` de la bibliothèque standard. Par exemple le filtre `pair` sur `[1;2;3]` renvoie `[2]`.

Exercice 19 (Difficulté: 1)

Écrivez une fonction `est_palindrome` qui vérifie si une liste est un palindrome (se lit de la même façon dans les deux sens).