Aide à la détection d’erreurs

Alain Cournier

alain.cournier@u-picardie.fr

Université de Picardie, UFR Sciences

De l’importance des erreurs

- https://fr.wikipedia.org/wiki/Vol_501_d%27Ariane_5

Présentation générale

Exemple : Suites de Syracuse (Wikipédia)

- Une suite de Syracuse est une suite d'entiers naturels définis de la façon suivante :
  - On part d'un entier naturel non nul quelconque. On défini le terme suivant de la suite comme :
    - Si le terme précédent est pair, on le divise par 2
    - Si le terme précédent est impair, on le multiplie par 3 et on ajoute 1
    - On répète l'opération à l'infini
  - Exemple : 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1…
- Remarque : à partir de 1 on obtient la suite périodique 1, 4, 2, 1…

Exemple : Suites de Syracuse (Wikipédia)

- Cette famille de suites poses de nombreuses questions :
  - Quel que soit l'entier strictement positif de départ, la suite de Syracuse devient-elle périodique à partir d'un certain rang ?
  - Quel que soit l'entier strictement positif de départ, la suite de Syracuse atteint-elle l'entier naturel 1 en un temps fini ?

Exemple : Programme de Syracuse

Fonction avec résultat entier Syracuse (Donnée i : entier naturel)

Variables : entier nb init 0; entier a init i;

DébutCodeFonction

Tant que a > 1 faire

Écrire(a); nb ←nb+1

Si a est pair alors a ← a div 2

Sinon a ← (3*a) +1

Finsi

Fintq

Ecrire(a); renvoyer(nb)

FinCodeFonction

Exemple : Programme de Syracuse

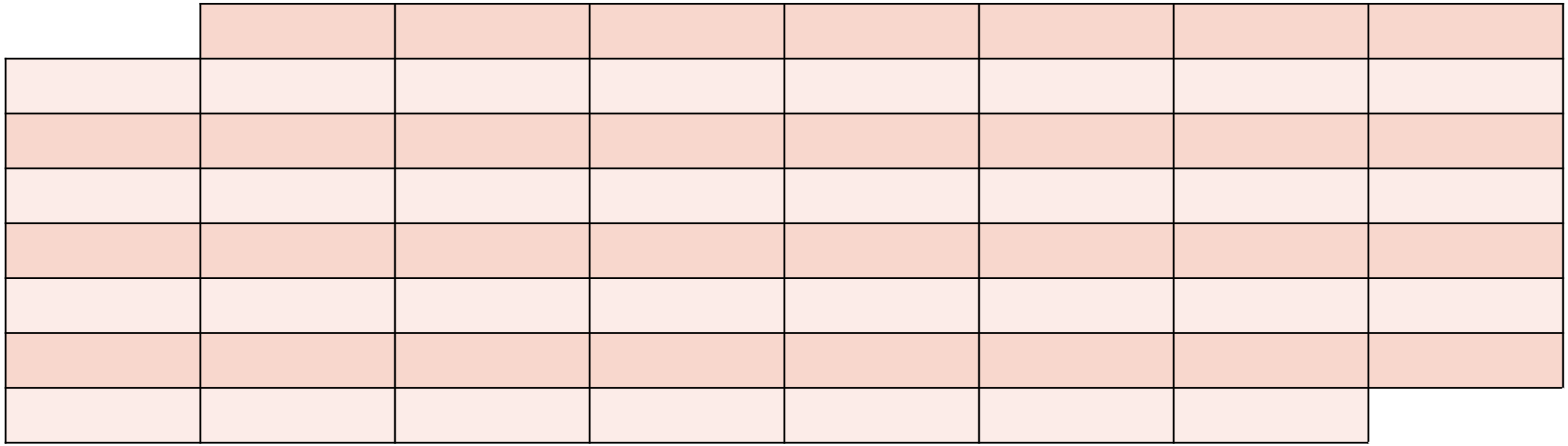
- Question 1 : Montrer que la fonction Syracuse renvoie toujours le bon résultat (Le nombre de termes qui précède 1 ou 0 dans la suite).

- Question 2 : Montrer que la fonction Syracuse s’arrêtera pour toute valeur de i.

Exemple : Programme de Syracuse

```
Fonction avec résultat entier Syracuse (Donnée i : entier naturel)
    Variables : entier nb init 0; entier a init i;
DébutCodeFonction
    Si a < 2 alors Ecrire(a); renvoyer(0)
    Sinon Écrire(a);
        Si a est pair alors Renvoyer(1+Syracuse(a div 2))
        Sinon Renvoyer(1+Syracuse((3*a)+1))
        FinSi
    FinSi
FinCodeFonction
```

Exemple 2 :



Peut-on couvrir ce tableau avec des dominos à deux cases sans que les dominos ne se recouvrent les uns les autres ? (Pb de pavage)

Introduction

Étapes de la conception d’un algorithme

1. Compréhension du problème
2. Spécifications
3. Idées et justifications
4. Algorithme
5. Démonstrations
    1. Arrêt
    2. Bon résultat
6. Complexités
7. Écriture dans un langage de programmation

Compréhension du problème

- C’est mieux…
- Il faut comprendre les données fournies
- Les modes opératoires
- Comprendre le résultat que souhaite avoir le client

Spécifications

- Spécifier c'est exprimer le lien (la fonction) qui lie la donnée fournie et le résultat attendu.

- Les spécifications doivent être indépendantes du processus de calcul.

- Les spécifications ne doivent pas laisser de places aux interprétations.
  - La langue française est ambiguë : « l'instituteur dit l'inspecteur est un âne »
  - Par exemple on n'imagine pas que le calcul de la tva dépende du magasin dans lequel elle est calculée (Voir https://wiki.dolibarr.org/index.php/R%C3%A8gles_de_calcul_et_arrondi_de_TVA)

- Les données doivent elles aussi être spécifiées :
  - Par leurs type (entier, réel, chaîne de caractères, …)
  - Par leur domaine de définition, le domaine de définition des opérateurs associés (*e.g.*, on ne divise pas par zéro)

Idées et justifications

- Dans cette étape on va préciser et spécifier chacune des étapes du processus de calcul.

- On cherche les cas particuliers qui méritent une attention soutenue dans le but de pouvoir les tester en priorité
  - Attention les tests sont nécessaires mais pas suffisant !

Algorithme

- Ici nous allons traduire les idées de l’étape précédente dans le langage algorithmique.

- Attention: un algorithme doit être indépendant de l’architecture de la machine

Preuve d’arrêt (termination)

- On doit prouver que l’algorithme que nous avons écrit s’arrête **pour toute donnée valide** (dans le domaine de définition).

- Cette preuve doit être écrite car un algorithme qui ne se fini pas, ne renvoie jamais de résultat.

Preuve du bon résultat (partial correctness)

- On démontre dans cette étape que le résultat calculé par l'algorithme coïncide avec les spécifications.

- But : Prouver que notre algorithme renvoie le bon résultat s'il termine (efficace).

Complexités

- Ici on évalue le coût en ressources de votre processus de calcul
  - En mémoire
  - En temps d'exécution (Utilisation du processeur)
  - En utilisation de la bande passante
  - …

- But : savoir si notre algorithme est **efficient**

- Attention la simplicité n'implique pas l'efficacité ! (https://fr.wikipedia.org/wiki/Fonction_d%27Ackermann)

Écriture dans un langage de programmation

- A vos claviers
- Oui je sais c’est dur d’attendre si longtemps.
- Attention: le code écrit n’est pas toujours le code exécuté :
  - Pb d’arrondi
  - Optimisation
  - …