Pas de ToC (alors qu'elle est présente dans le template)

INFO0947: Récursivité et Elimination de la Récursivité

Peissone Dumoulin, s193957

1 Formulation récursive

↳ · Indiquer le raisonnement est toujours une bonne idée

· la synthèse doit se présenter comme une notation (ça sera + simple pour spécif & approche constructive) : $f(\cdot,\cdot) = \begin{cases} \text{—} & \text{, cas de base} \\ \text{—} & \text{, cas général} \end{cases}$

· OK avec la formulation, mais si elle ne suit pas la formule de l'énoncé (cf. expo)

Spécif :

↳ la postcondition doit s'exprimer à l'aide de la notation introduite en Sec. 2

Approche constructive

↳ la mauvaise postcondition t'empêche de tirer des conclusions après le retour

Postcond rec est du correct

Complexité

Tu dois commencer par décomposer ton code en 3 morceaux => cf. ma découpe + explication Sec. 3.4

Il faut ensuite résoudre ton équa de récurrence

Tu ne peux pas te contenter de conclure sans prendre la peine de justifier

· Dire

1) Tu dois suivre les algos vus en cours (cf. Chap 9 + Pseudo Code)

Tu dois justifier chaque étape !

1 Formulation Récursive

explication ?
justification ?

1.1 Cas de base

Si $n = 1$:

$$hexa\_dec\_rec(hexa, n) = convert(hexa[n-1])$$

1.2 Cas récursif

Si $n > 1$:

$$hexa\_dec\_rec(hexa, n) = convert(hexa[n-1]) + 16 * hexa\_dec\_rec(hexa, n-1)$$

2 Spécification

c'est qui ?

```
//PréConditions : hexa != NULL, n > 0
//PostConditions : hexa_dec_rec = decimal ∧ hexa = hexa_0 ∧ n = n_0
unsigned int hexa_dec_rec(char *hexa, int n);
```

3 Construction Récursive

3.1 Programmation Défensive

On vérifie que la précondition est respectée en interdisant à hexa d'être NULL et n ne peut être négatif

```
unsigned int hexa_dec_rec(char *hexa, int n){
    assert(hexa != (void*)0 && n > 0);
    // {PréConditions ≡ hexa ≠ NULL ∧ (longueur(hexa) > 0 ⟹ n > 0)}
}
```

c'est qui ?
vient d'où ?

3.2 Cas de Base

On gère le cas de base où $n = 1$ après s'être assuré que les préconditions sont bien respectées.

```
// {PréConditions ≡ hexa ≠ NULL ∧ (longueur(hexa) > 0 ⟹ n > 0)}
if(n == 1)
    // {hexa ≠ NULL ∧ n = 1}
    return convert(hexa[n - 1]);
    // {hexa_dec_rec = convert(hexa[n - 1]) ∧ hexa = hexa_0 ∧ n = n_0}
    // {⟹ PostCondition}
}
```

Impossible de conclure sans savoir de la façon dont tu exprimes la Post.

3.3 Cas Récursif

Il y a un seul cas récursif, lorsque n est strictement supérieur à 1. {PréConditions$_{REC}$} et {PostConditions$_{REC}$} sont respectivement les PréConditions et les PostConditions de l'appel récursif.

```
else
    // {hexa ≠ NULL ∧ n > 1}
    // {⟹ PréConditions_REC}
    return convert(hexa[n - 1]) + 16 * hexa_dec_rec(hexa, n - 1);
    // {PostConditions_REC ≡
    // hexa_dec_rec(hexa, n) = convert(hexa[n - 1]) + 16 * hexa_dec_rec(hexa, n-1)
    // ∧ n = n - 1 ∧ hexa = hexa_0}
    // {⟹ PostConditions}
```

ça peut être la Post$_{rec}$

↳ incohérent par rapport à la postcond (cf. sec 2)

3.4 Code complet

```
unsigned int hexa_dec_rec(char *hexa, int n){
  assert(hexa != (void*)0 && n > 0);//Préconditions

  if(n == 1)
    return convert(hexa[n - 1]);//Cas de base
  else
    return convert(hexa[n - 1]) + 16 * hexa_dec_rec(hexa, n - 1);//Cas récursif
}
```

A

b

$T(n-1)$

⇒ Complexité :

$$T(1) = A$$

$$T(n) = b * T(n-1)$$

4 Traces d'Exécution

Voici les traces d'exécution concernant les 3 exemples du fichier main-hexadécimal.c

Difficile de comprendre comment évolue la pile (le template contenait du code LaTeX pour le faire)

4.1 hexa_dec_rec("27", 2)

hexa_dec_rec("27", 2)	7 + 16 * 2 = 39

4.2 hexa_dec_rec("A23", 3)

hexa_dec_rec("A23", 3)	3 + 16 * (2 + 16 * 10) = 2595

4.3 hexa_dec_rec("A78E", 4)

hexa_dec_rec("A78E", 4)	14 + 16 * (8 + 16 * (7 + 16 * (10))) = 42894

5 Complexité

→ ??

Avec $b \in [0, 15]$, $n \in \mathbb{N}$, on a :

$T(n) = 16 * T(n-1) + b$ → correspond normalement à la constante + pivot avant incrément unique

5.1 Cas de base

Dans le cas où $n = 1$:

$T(n)$ est linéaire car $\cancel{T(1) = b}$ et $b \in [0, 15]$

5.2 Cas récursif

Dans le cas où $n > 1$:

La fonction hexa_dec_rec(hexa, n) va s'appeler récursivement en décrémentant la valeur courante de n à chaque appel jusqu'à atteindre le cas de base. On a donc n - 1 appels récursifs.

$T(n) = 16 * T(n-1) + b \rightarrow O(n)$) Pq ?

La complexité de la fonction hexa_dec_rec(hexa, n) est **linéaire**.

6 Dérécursification

Pour procéder à la dérécursification, on va utiliser le pseudo langage qu'on a vu dans les gamecodes associés.

6.1 Code récursif

```
hexa_dec_rec(String hexa, int n):
  if(n=1)
    then
      r ← convert(hexa[n - 1]);
    else
      r ← convert(hexa[n - 1]) + 16 * hexa_dec_rec(hexa, n - 1);
```

6.2 Code dérécursivé

Comment tu passes de l'une à l'autre ?

```
hexa_dec_rec'(String hexa, int n):
    s ← hexa;
    u ← n;
    until u = 1 do
        s ← s;
        u ← u - 1;
    end
    r ← convert'(hexa[u - 1]);
```

Incorrect, de toute façon