INFO0947: Récursivité et Elimination de la Récursivité

Peissone Dumoulin, s193957

1 Formulation Récursive

1.1 Cas de base

Si $n = 1$:

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

1.2 Cas récursif

Si $n > 0$:

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

2 Spécification

```
1 //PréConditions : hexa != NULL, n >= 0
2 //PostConditions : hexa_dec_rec = decimal ∧ hexa = hexa0 ∧ n = n0
3 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 pas être strictement négatif

```
1 unsigned int hexa_dec_rec(char *hexa, int n){
2     assert(hexa != (void*)0 && n >= 0);
3     // {PréConditions ≡ hexa ≠ NULL ∧(length(hexa) ≥ 0 ⟹ n ≥ 0)}
4 }
```

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 ∧(length(hexa) ≥ 0 ⟹ n ≥ 0)}
if(n == 1 && i == n){ // i == n permet de différencier le cas de base du cas récursif
    // {hexa ≠ NULL ∧ n = 1 ∧ i = n}
    if(convert(hexa[n - 1]) != (unsigned int) -1)
    //Vérif de la valur de retour de convert()
        return convert(hexa[n - 1]);
        // {hexa_dec_rec = convert(hexa[n - 1]) ∧ hexa = hexa₀ ∧ n = n₀}
        // {⟹ PostCondition}
    else
        return -1;//dans le cas où l'un des char n'existe pas en hexadécimal
        // {hexa_dec_rec = -1 ∧ hexa = hexa₀ ∧ n = n₀}
        // {⟹ PostCondition}
}
```

3.3 Cas Récursifs

Il y a un seul cas récursif, selon que n est strictement positif. $\{\text{PréConditions}_{REC}\}$ et $\{\text{PostConditions}_{REC}\}$ sont respectivement les PréConditions et les PostConditions de l'appel récursif.

```
}else if(n > 0){
    // {hexa ≠ NULL ∧ n > 0}
    if(convert(hexa[n - 1]) != (unsigned int) -1){
    //Vérif de la valur de retour de convert()
        // {⟹ PréConditions_REC}
        intermediaire += convert(hexa[n - 1]) * puissance;
        // {hexa ≠ NULL ∧ n > 0 ∧intermediaire = intermediaire + convert(hexa[n − 1])}
        puissance *= 16;
        // {hexa ≠ NULL ∧ n > 0 ∧ intermediaire = intermediaire + convert(hexa[n − 1])
        // ∧ puissance = puissance ∗ 16}
        i = 2;
        // {hexa ≠ NULL ∧ n > 0 ∧ intermediaire = intermediaire + convert(hexa[n − 1])
        // ∧ puissance = puissance ∗ 16 ∧ i = 2}
        return hexa_dec_rec(hexa, n - 1);
        // {PostConditions_REC ≡ hexa_dec_rec = hexa_dec_rec ∧ n = n - 1 ∧ hexa = hexa₀ }
        // {⟹ PostConditions}
    }else
        return -1;//dans le cas où l'un des char n'existe pas en hexadécimal
        // {hexa_dec_rec = -1 ∧ hexa = hexa₀ ∧ n = n₀}
        // {⟹ PostConditions}
}
```

3.4 Code complet

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

    static int puissance = 1, intermediaire = 0, i = 1;
    int decimal;

    if(n == 1 && i == n){
        if(convert(hexa[n - 1]) != (unsigned int) -1)
            return convert(hexa[n - 1]);
        else
            return -1;//dans le cas où l'un des char n'existe pas en hexadécimal
    }else if(n > 0){
        if(convert(hexa[n - 1]) != (unsigned int) -1){
            intermediaire += convert(hexa[n - 1]) * puissance;
            puissance *= 16;
            i = 2;
            return hexa_dec_rec(hexa, n - 1);//récursivité
        }else
            return -1;//dans le cas où l'un des char n'existe pas en hexadécimal
    }

    decimal = intermediaire;
    puissance = 1;
    intermediaire = 0;

    return decimal;//retourne la valeur décimale finale
}
```

4 Traces d'Exécution

5 Complexité

6 Dérécursification