Feuille de TD/TP n° 5

Algorithmes sur les scalaires

Exercice 1 : Soit l'algorithme (A) suivant calculant la racine carré entière d'un entier `N` :

```
Sqrt (N) =
   R ← 0
   tant que R×R < N faire
      R ← R + 1
   renvoyer R - 1
```

Soit l'algorithme (B) suivant :

```
Sqrt (N) =
   R ← N div 2 + 1;
   tant que R×R > N faire
      R ← R - 1
   renvoyer R
```

Questions :

- Indiquer la complexité de ces algorithmes en tours de boucle.
- Proposer une version récursive utilisant la recherche dichotomique (C).
- Programmer ces algorithmes en utilisant le type `Long_Integer`.
- Tester l'algorithme (A) pour déterminer la racine carrée de `9000000000000000005`.
- Tester l'algorithme dichotomique (C) pour déterminer cette racine carrée en utilisant comme intervalle de départ `[0 .. 3070000000]`.
- Peut-on tester l'algortihme (B) pour trouver cette racine ? Commenter.

Exercice 2 : Soit l'algorithme suivant calculant la multiplication de `A` par `B` :

```
Mult (A, B) =
   M ← 0
   tant que B > 0 faire
      si B modulo 2 = 1 alors
         M ← M + A
      A ← A * 2
      B ← B / 2
   renvoyer M
```

Questions :

- Indiquer la complexité de cet algorithme en tours de boucle. Comparer avec la multiplication par additions successives.

- Programmer cet algorithme en utilisant le type `Long_Integer`.
- Tester l'algorithme pour déterminer le produit de `2000000000` par `3000000000`. Comparer avec la multiplication par additions successives.

Exercice 3 : Définir une fonction de prototype :

```
function Dicho (Val, Min, Max : Long_Integer) return Long_Integer;
```

Qui effectue une recherche dichotomique sur l'intervalle `[Min .. Max]` d'une valeur `N` telle que, pour une fonction croissante `F` définie auparavant, `F (N) <= Val` et `F (N + 1) > Val`.

Utiliser cette fonction pour tester la recherche dichotomique du zéro d'une fonction. Utiliser cette fonction pour redéfinir la recherche dichotomique de la racine carrée dans l'exercice 1.

Exercice 4 : Soit l'algorithme suivant calculant la valeur approchée de π à ε près par développement limité :

```
PI ← 0
N ← 0
T ← 1
S ← 1
tant que 4 * T > EPSILON faire
   PI ← PI + T * S
   N ← N + 1
   T ← 1 / (2 * N + 1)
   S ← -S
PI ← 4 * PI
```

Questions :

- Programmer cet algorithme en utilisant le type `Long_Float`.
- Tester avec une précision de 5, 8 puis 10 chiffres après la virgule. Comparer avec le nombre π d'Ada (Ada.Numerics.PI).
- Utiliser l'approximation des réels représentés en machine pour modifier l'algorithme afin de déterminer π avec la précision maximale.