

# L3 MI / PROG

## TP#4: Racine carrée entière

2023-09-27

### Exercice 1 : Racine carrée entière & bibliothèques dynamiques

L'objectif de cet exercice est de développer une petite bibliothèque partagée dynamique qui implémente une fonction `uint64_t isqrt(uint64_t x)`, qui calcule et renvoie  $\lfloor \sqrt{x} \rfloor$ , la racine carrée de  $x \in \llbracket 0, 2^{64} - 1 \rrbracket$  arrondie à l'entier inférieur.



Cet exercice est issu de l'examen terminal 2022 ; il est inspiré du *Hacker's Delight*<sup>1</sup>, chapitre 11-1.

#### Q.1 :

1. Écrivez une fonction «leading zero count» `uint64_t lzc(uint64_t x)` ; qui renvoie le nombre (compris entre 0 et 64) de *zéros de tête* dans l'écriture binaire de  $x$ . Autrement dit, cette fonction renvoie le nombre de zéros rencontrés avant le premier bit à 1 dans  $x$ , en partant du bit de poids fort. Autrement dit, si  $x$  représente l'entier  $\sum_{i=0}^{63} x_i 2^i$ , si  $j \in \llbracket 0, 64 \rrbracket$  est la valeur minimum de  $i$  telle que  $k \geq j \Rightarrow x_k = 0$ , cette fonction renvoie  $64 - j$ .

EXEMPLE :

- Pour  $x = 0x0000000000000000ULL$ , `lzc(x)` retourne 64 (et  $j$  vaut 0).
- Pour  $x = 0x0000000000000001ULL$ , `lzc(x)` retourne 63 (et  $j$  vaut 1).
- Pour  $x = 0x8000000000000000ULL$ , `lzc(x)` retourne 0 (et  $j$  vaut 64).
- Pour  $x = 0x1000000000000000ULL$ , `lzc(x)` retourne 3 (et  $j$  vaut 61).

REMARQUE. Sur les processeurs x86 modernes, il existe une instruction `lzcnt` qui calcule exactement cette fonction. Vous pouvez l'utiliser si vous le souhaitez, mais devez néanmoins d'abord implémenter votre propre fonction.

#### Q.2 :

1. Écrivez une fonction `uint64_t isqrta(uint64_t x)` ; qui calcule et renvoie la plus petite puissance de deux supérieure ou égale à  $\lceil \sqrt{x} \rceil$ , et 0 si  $x$  vaut zéro. Vous pourrez exploiter le fait que pour  $x$  non nul, cette puissance vaut  $2^i$  avec  $i = 32 - \text{lzc}(x-1)/2$ .

ATTENTION : On rappelle que le littéral `1` a pour type `int`.

ATTENTION : Votre fonction doit bien renvoyer la *puissance* de deux demandée, et non son logarithme.

---

1. Henry S. Warren, Jr., 2013.

**Q.3 :**

1. Implémentez `isqrt` en utilisant l'algorithme « Babylonien » suivant (dont on admettra la correction) pour calculer  $\lfloor \sqrt{x} \rfloor$  pour  $x > 0$  :

(a)  $u_0 := g$  avec  $g \geq \lceil \sqrt{x} \rceil$ .

(b) Calculez  $u_{n+1} = \lfloor (u_n + \lfloor x/u_n \rfloor) / 2 \rfloor$ , et si  $u_{n+1} \geq u_n$ , retournez  $u_n$ .

Prenez garde à bien choisir votre valeur initiale pour  $u_0$  ; celle-ci devra être correcte et « la meilleure possible » (dans la mesure du raisonnable).

**Q.4 :** Pour cette question, on conseille très fortement d'utiliser un Makefile bien configuré. On rappelle que les flags `CPPFLAGS` et `LDFLAGS` sont généralement utilisés pour respectivement renseigner les options `-I` et `-L` du compilateur ou linkeur. Il peut également être pratique d'utiliser la commande `$(shell toto)` qui au sein d'un Makefile est remplacée par l'évaluation de la commande `toto` exécutée dans le shell où `make` est invoqué.

1. Compilez l'ensemble de vos fonctions vers une bibliothèque partagée dynamique `libisqrt.so` (en utilisant éventuellement une autre extension en fonction de votre système), et créez un fichier `isqrt.h` correspondant. Prenez garde à ne déclarer et à ne rendre visible *que* la ou les fonctions nécessaires.
2. Créez un programme de test qui invoque votre fonction `isqrt` depuis la bibliothèque partagée. Vérifiez qu'une modification de la fonction `isqrt` (par exemple l'introduction d'un bug volontaire) change bien les résultats renvoyés par votre programme de test *sans que celui-ci n'ait besoin d'être recompilé*.

**Q.5 :**

1. Ajoutez à votre programme de test une fonction qui tire un grand nombre (par exemple  $2^{23}$ ) entiers positifs entre 0 et  $2^{32}$ , et vérifie que `isqrt(x)` renvoie le même résultat que `floor(sqrt(x))`, avec `sqrt` de signature `double sqrt(double x)` la fonction *racine carrée* de la bibliothèque de mathématiques standard, et `floor` de signature `double floor(double x)` la fonction d'arrondi à l'entier inférieur de cette même bibliothèque.
2. Expliquez pourquoi pour certaines valeurs  $x \in \llbracket 0, 2^{64} - 1 \rrbracket$ , les deux expressions de la question précédente peuvent être différentes ?
3. En cas de différence, quelle fonction renvoie le résultat correct ?
4. Vérifiez vos hypothèses.

**Q.6 (bonus) :**

1. Démontrez le fait utilisé dans la question **Q.2**. **INDICE :** Faites un raisonnement par cas, et exploitez le fait que si  $x = 2^i$ ,  $\text{lzc}(x) = 64 - (i + 1)$ .