PROG L3-MI
ET

2022-12-07

Consignes. *La durée de cet examen est de deux heures. Aucun document n'est autorisé.* ***Dans tous les exercices, il est possible d'admettre le résultat d'une question pour répondre aux questions suivantes****. Toute réponse doit être justifiée. Les programmes doivent être écrits en C.*

Exercice 1 (environ 2 points) : `memcpy`

Q.1 :

1. Écrivez une fonction `memcpy` de signature :

   ```
   void *memcpy(void *restrict dst, const void *restrict src, size_t n);
   ```

   qui prend en argument deux pointeurs `dst` et `src` pointant tous deux vers des zones mémoire de taille au moins `n` octets ne se chevauchant pas, copie les `n` premiers octets de la zone pointée par `src` dans la zone pointée par `dst`, et renvoie la valeur de son argument `dst`.

Exercice 2 (environ 10 + 2 points) : Racine carrée entière

L'objectif de cet exercice est d'implémenter une fonction `uint64_t isqrt(uint64_t x);` qui calcule et renvoie $\lfloor\sqrt{\mathtt{x}}\rfloor$, la racine carrée de $\mathtt{x} \in [\![0, 2^{64} - 1]\!]$ arrondie à l'entier inférieur.

Cet exercice est inspiré du *Hacker's Delight*[1], 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 $\sum_{i=0}^{63} \mathtt{x}_i 2^i$, si $j \in [\![0, 64]\!]$ est la valeur minimum de i telle que $k \geq j \Rightarrow \mathtt{x}_k = 0$, 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).

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

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 $\lceil\sqrt{\texttt{x}}\rceil$, et `0` si `x` vaut zéro. Vous pourrez exploiter le fait que pour `x` non nul, cette puissance vaut 2^i avec $i =$ `32 - 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.

Q.3 :

1. Implémentez `isqrt` en utilisant l'algorithme « Babylonien » suivant pour calculer $\lfloor\sqrt{x}\rfloor$ pour $x > 0$ (dont on admettra la correction) :
   (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 justifier votre choix initial pour u_0, qui devra être correct et « le meilleur possible » (dans la mesure du raisonnable).

Q.4 :

1. Expliquez pourquoi pour `x` $\in [\![0, 2^{64}-1]\!]$, on peut avoir `isqrt(x)` différent de `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. En cas de différence, quelle fonction renvoie le résultat correct ?

Q.5 *(bonus)* :

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

Exercice 3 (environ 9 points) : Convolution discrète en deux dimensions

L'objectif de cet exercice est de calculer (naïvement) une convolution discrète en deux dimensions sur des octets non signés. Soit en entrée un rectangle R de n_r lignes et n_c colonnes dont les éléments sont notés $R[x][y]$, $0 \leq x < n_r$, $0 \leq y < n_c$, et un noyau de convolution K (que l'on fixera dans notre cas comme un carré de dimension 3) dont les éléments sont notés $K[x][y]$, $-1 \leq x \leq 1$, $-1 \leq y \leq 1$, et à valeur dans $[\![-255, 255]\!]$, on souhaite calculer le nouveau rectangle R' défini par :

$$R'[x][y] = \overline{\overline{\wedge}}\left(\sum_{d_x=-1}^{1}\sum_{dy=-1}^{1} R[x+d_x][y+d_y] \times K[d_x][d_y]\right) \tag{1}$$

où :

— les sommes sont calculées dans les entiers relatifs $\mathbb{Z}$;
— l'on considère que pour x, y tels que $R[x][y]$ n'est pas défini, on prend $R[x][y] := 0$;
— $\overline{\overline{\wedge}}$ est un opérateur de *saturation*, $\mathbb{Z} \to [\![0, 255]\!]$ défini par :

$$\overline{\overline{\wedge}} : x \mapsto \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \in [\![0, 255]\!] \\ 255 & \text{si } x > 255 \end{cases}$$

Q.1 :

1. Écrivez une fonction `uint8_t saturate(int32_t x)` qui implémente l'opérateur $\overline{\overline{\wedge}}$ pour un argument de type `int32_t`.

Q.2 :

1. Écrivez une fonction `getv` de signature :

   ```
   uint8_t getv(const uint8_t *in, int64_t nrows, int64_t ncols, int64_t x,
   ↪  int64_t y)
   ```

   qui renvoie $R[x][y]$ ou zéro en fonction de la valeur des arguments x et y (suivant la convention établie ci-dessus), et où R de dimension $n_r =$ `nrows`, $n_c =$ `ncols` est représenté par un pointeur `in` tel que $R[x][y]$ se trouve à l'adresse de base (stockée dans `in`) incrémentée de $x \times n_c + y$. (Autrement dit, R est stocké « en ligne ».)

Q.3 :

1. Proposez une convention de représentation de K par `kern`, de type `const int16_t kern[3][3]`.
2. Écrivez une fonction `dconv2` de signature :

   ```
   uint8_t *dconv2(const uint8_t *in, int64_t nrows, int64_t ncols, const
   ↪  int16_t kern[3][3])
   ```

   qui alloue une zone mémoire permettant de stocker le résultat R' de la convolution de R (représenté par `in` comme dans la question précédente) par K (représenté suivant votre convention), y écrit le résultat calculé suivant (1) (en utilisant le même format de représentation que pour R), et retourne l'adresse de la zone mémoire allouée.

Q.4 : On souhaite généraliser le calcul de convolution à des cas où le *noyau de convolution* K a des coefficients possiblement non entiers, ce qui n'est pas permis par la fonction `dconv2` telle que spécifiée ci-dessus.

1. Proposez une modification de `dconv2` qui permette de résoudre ce problème. Il devra notamment être possible d'utiliser cette nouvelle fonction pour que $R'[x][y]$ soit (une approximation de) la moyenne des neuf valeurs dans le voisinage de $R[x][y]$.

   Vous pourrez si vous le souhaitez appuyer votre réponse d'extraits de programmes, mais il n'est pas demandé de fournir une implémentation complète.

   Indice : Vous pouvez modifier le domaine de définition des coefficients de K et le type de `kern` et/ou la définition de l'opérateur de saturation utilisé.
2. Donnez les coefficients d'un noyau (et d'éventuels paramètres additionnels) pouvant être utilisé avec votre nouvelle fonction pour calculer la moyenne mentionnée ci-dessus.

Remarque : Les convolutions discrètes en deux dimensions sont notamment utilisées en traitement d'image.