Perfectionnement à la programmation en C
Fiche de TP 8

L2 Informatique 2020-2021

Casse-tête des huit dames

Ce TP se déroule en **une seule séance** et est à faire par binômes. Le travail réalisé doit être envoyé au plus tard exactement **une semaine après** le début de la séance de TP. Il sera disposé sur la plate-forme prévue à cet effet et constitué des programmes répondant aux questions et des éventuels fichiers annexes qui peuvent être demandés.

L'objectif de ce TP est d'implanter le casse-tête des *huit dames*. Nous utilisons et approfondissons ici les notions de

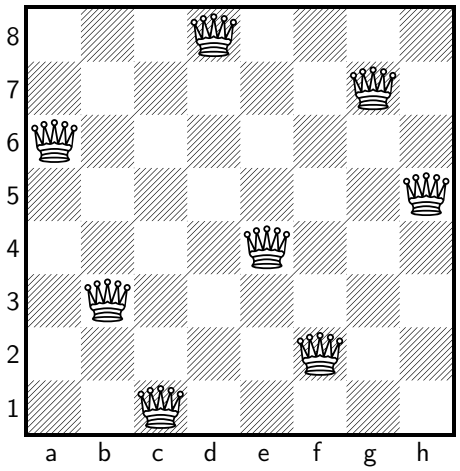
— découpage d'un projet en modules ;

— interface NCURSES ;

— opérateurs bit à bit.

1 Spécification

Un célèbre problème d'échecs consiste à placer huit dames (aux échecs, on dit « dame » et non pas « reine » !) sur un échiquier sans qu'aucune n'attaque une autre. On rappelle q'une dame attaque toutes les autres cases qui sont situées sur sa colonne, sa rangée ou ses diagonales. Voici, par exemple, une configuration gagnante :



L'objectif est de réaliser une interface graphique permettant à un utilisateur de placer à la souris (ou au clavier) des dames sur un échiquier. Le programme signale si le placement est gagnant ou non. Pour mener tout ceci à bien, il sera question — bien que ceci n'est pas évident de prime abord — de manipulation de bits et d'emploi d'opérateurs bit à bit. Nous verrons que représenter une situation de jeu par des suites de bits et opérer dessus à l'aide d'opérateurs bit à bit permet de gagner en efficacité.

2 Opérateurs bit à bit

Il est primordial d'avoir une bonne compréhension préliminaire des opérateurs bit à bit avant de commencer ce TP. Cette partie et le cours correspondant sont faits pour acquérir les bases dans ce domaine.

Commençons par poser quelques points de vocabulaire et de notation utilisés dans tout le TP. Nous notons toute suite de bits u de taille n bits par

$$u = u_{n-1} \dots u_1 u_0.$$

En toute généralité, u_i est le *bit d'indice* i de u. Ainsi par exemple, u_0 est le bit d'indice 0 de u et u_{n-1} est son bit d'indice $n-1$. De plus, on dira que le j^e *bit* de u est le bit u_{j-1}. Ainsi, u_0 est le premier bit de u et u_{n-1} est son n^e bit. On dira aussi que le bit d'indice 0 de u est le *bit de poids faible* de u, tandis que le bit d'indice $n-1$ est son *bit de poids fort*.

Par exemple, si $u = 01111001$, alors nous avons $n = 8$, $u_0 = 1$, $u_1 = 0$ et $u_{n-1} = 0$, Ainsi, le premier bit de u est 1, son 2^e bit est 0 et son n^e bit est 0.

Voici maintenant un exemple d'opération bit à bit. L'expression

```
1 ~(1 << 7);
```

a pour valeur l'entier possédant son 8^e bit à `0` et tous les autres à `1`. En effet, la sous-expression `1 << 7` a pour valeur l'entier dont le 8^e bit est à `1` et les autres à `0`. Le fait de considérer son complémentaire (opérateur `~`) produit le résultat attendu.

Exercice 1. (Opérations bit à bit)

Nous utilisons dans cet exercice les opérateurs bit à bit pour modifier des mots binaires de façon adéquate. Toutes les questions de cet exercice doivent **être résolues en une ligne** en C à la manière de l'exemple donné ci-dessus.

1. Créer un entier tel que son 15^e bit est à `1` et les autres sont à `0`.
2. Créer un entier tel que son 14^e bit est à `0` et les autres sont à `1`.
3. Mettre le 13^e bit d'un entier `x` à `1`.
4. Mettre le 12^e bit d'un entier `x` à `0`.

5. Tester si le 11^{e} bit d'un entier `x` est un `1`.
6. Tester si les 3 premiers bits d'un entier `x` sont tous `1`.
7. Tester si les 4 premiers bits d'un entier `x` sont tous `0`.
8. Tester l'égalité des 10^{e} bits de deux entiers `x` et `y`.
9. Tester si deux entiers `x` et `y` possèdent au moins un bit à `1` situé en un même indice.
10. Tester si tous les bits des deux entiers `x` et `y` qui sont en mêmes indices sont égaux.
11. Tester si tous les bits des deux entiers `x` et `y` qui sont en mêmes indices sont différents.

3 Écriture du programme

Exercice 2. (Conception du projet)

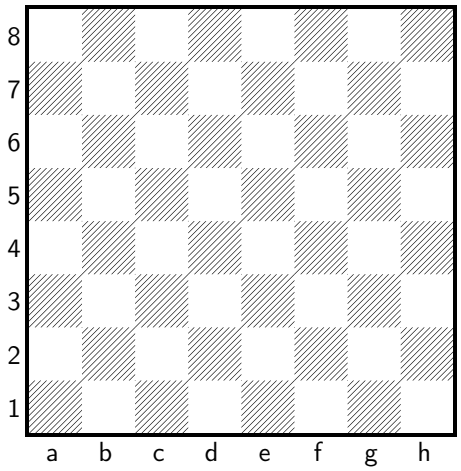
L'objectif de cet exercice est de concevoir une architecture viable pour le projet présenté.

1. Lire attentivement **l'intégralité du sujet** avant de commencer à répondre aux questions. La description du sujet constitue une spécification de projet.
2. Établir la liste de toutes les fonctionnalités demandées, option par option.
3. Une fois ceci fait, proposer un découpage en modules cohérent du projet. Pour chaque module proposé, décrire les types qu'il apporte ainsi que ses objectifs principaux.
4. Au fil de l'écriture du projet, il est possible de se rendre compte que le découpage initialement prévu n'est pas complet ou adapté. Si c'est le cas, mentionner l'historique de ses modifications.
5. Maintenir un `Makefile` pour compiler le projet.

Exercice 3. (Représentations des données)

Le but de cet exercice est de définir la façon dont va être représenté un échiquier en mémoire. L'information contenue dans un échiquier est la présence ou l'absence, case par case, de dames. Il est clair que cette information seule est suffisante pour représenter de manière fidèle une configuration de jeu.

Un échiquier est un quadrillage de taille 8×8 dont les cases sont, en première approche, indexées par des couples lettre / entier (voir la figure 1a). De manière plus compacte, chaque case peut être représentée par un entier de 0 à 63 comme le montre la figure 1b. Il est possible d'« aplanir » un échiquier en le voyant comme un suite de 64 cases dont l'élément d'indice i de la suite représente la case d'indice $63 - i$ de l'échiquier (voir la figure 1c).



(a) L'échiquier concret dans lequel chaque case est désignée par un couple.

56	57	58	59	60	61	62	63
48	49	50	51	52	53	54	55
40	41	42	43	44	45	46	47
32	33	34	35	36	37	38	39
24	25	26	27	28	29	30	31
16	17	18	19	20	21	22	23
8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7

(b) L'échiquier abstrait dans lequel chaque case est indexée par un entier.

63	62	61	60	59	58	57	56	55	54	53	…	…	…	…	4	3	2	1	0

(c) L'échiquier aplati. Il y a correspondance entre les indices des cases de l'échiquier et les indices des bits d'un mot binaire de 64 bits.

FIGURE 1 – L'échiquier, vu de trois façons différentes.

Toute case d'un échiquier peut prendre deux états : occupée par une dame ou bien libre. On décide de représenter une position[1] d'échecs par un entier de 64 bits : chaque bit code l'état de la case indiquée par son indice dans l'entier. Un bit à `1` (resp. `0`) spécifie que la case correspondante est occupée par une dame (resp. vide). De plus, le bit d'indice i code pour la case d'indice i de l'échiquier. Par exemple, la position située en première page du sujet est codée par l'entier de 64 bits suivant :

00001000 01000000 00000001 10000000 00010000 00000010 00100000 00000100.

1. Définir un type énuméré `Case` qui contient, de `A1` à `H8` toutes les cases d'un échiquier. Il faut que la valeur numérique de la constante `A1` soit `0`, celle de `B1` soit `1`, …, et finalement celle de `H8` soit `63`. Utiliser les fonctions rechercher / remplacer de l'éditeur de texte pour ne pas perdre de temps ici.

2. On rappelle que le type `unsigned long long` permet de représenter des entiers de 64 bits[2]. Déclarer un type synonyme `Position` qui permet de représenter des positions selon la convention expliquée. Cette déclaration de type doit répondre aux explications précédentes et être la plus simple possible.

3. Définir une fonction

1. Comme dans les TP précédents, on appelle « position » aux l'ensemble des données représentant une situation de jeu.

2. Pour utiliser ce type sans avertissement de la part du compilateur, on compilera sans l'option `-pedantic`. Ceci est exceptionnel et ne concerne donc que ce TP.

```c
int est_case_occupee(Position pos, Case c);
```

qui renvoie `1` si la case `c` de la position `pos` est occupée par une dame et `0` sinon.

Exercice 4. (Cœur du programme)

Dans cet exercice, nous allons écrire les fonctions de base permettant de faire les différents tests nécessaires à la résolution du problème des huit dames. Nous utiliserons les opérations bit à bit vues dans l'exercice 1 et la représentation d'une position efficace introduite dans l'exercice 3.

1. Écrire une fonction

```c
int placer_dame_position(Position *pos, Case c);
```

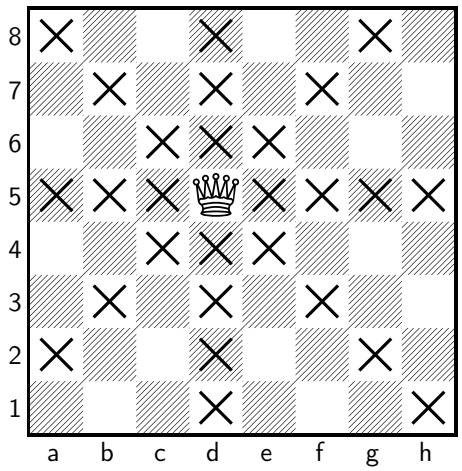
qui modifie la position pointée par `pos` pour qu'elle contienne une dame sur la case `c`, en laissant les contenus des autres cases inchangés. La fonction renvoie un entier pour signaler une éventuelle erreur.

2. Écrire une fonction

```c
int afficher_position(Position pos);
```

qui réalise un affichage de la position `pos` sur la sortie standard. La présence d'une pièce sera notée par un `'+'` et l'absence, par un `'.'`. La fonction renvoie un entier pour signaler une éventuelle erreur.

3. Une dame *attaque* toute les cases situées sur sa rangée, sa colonne et ses deux diagonales. Voici par exemple les cases attaquées (case contenant une ×) par une dame en d5 :



Écrire une fonction

```c
int calculer_cases_attaquees(Position *pos, Case c);
```

qui modifie la position pointée par `pos` pour qu'elle contienne exactement les cases attaquées par une dame située sur la case `c`. Ainsi, cette position résultat doit avoir des bits à `1` aux endroits correspondants à des cases attaquées et des bits à `0` ailleurs. La fonction renvoie un entier pour signaler une éventuelle erreur.

Attention : une dame n'attaque pas la case sur laquelle elle se trouve !

4. On souhaite à présent, pour gagner en efficacité, sauvegarder les résultats de la fonction précédente pour toutes les valeurs possibles de son paramètre `c`. Pour cela, déclarer une variable globale

```
Position tab_cases_attaquees[64];
```

   Ce tableau doit être initialisé à l'aide de la fonction `calculer_cases_attaquees` avant toute utilisation. Ce tableau statique, de taille 64, est ainsi indexé par des cases. Pour toute case `c`, `tab_cases_attaquees[c]` contient la position calculée par la fonction de la question précédente.

   Attention : l'utilisation de variables globales est à proscrire dans le cas général. Son utilisation est cependant ici justifiée.

5. Écrire une fonction

```
int est_sans_attaque_mutuelle(Position pos);
```

   qui renvoie `1` si les dames de la position `pos` ne s'attaquent pas mutuellement et `0` sinon.

   Indication : il est possible d'utiliser l'algorithme suivant. On commence par calculer un entier `attaques` *de 64 bits qui contient l'ensemble des cases attaquées par toutes les dames de* `pos`*. Cet entier est calculé en parcourant toutes les cases de la position qui contiennent une dame, en exploitant les valeurs du tableau* `tab_cases_attaquees`*. Ensuite, comparer par le bon opérateur bit à bit les entiers* `attaques` *et* `pos`*. De cette comparaison, en déduire le résultat à renvoyer.*

Exercice 5. (Interface graphique)

Le but de cet exercice est de réaliser une interface graphique utilisant la bibliothèque graphique NCURSES qui permet de s'essayer au problème des huit dames.

1. Écrire une fonction paramétrée par une position et qui permet de l'afficher.

2. Écrire une fonction qui permet à l'utilisateur d'essayer de résoudre le problème. L'interface indiquera par un moyen de quelconque (visuel, sonore, *etc.*) si un placement illégal (c.-à-d. lorsque deux dames s'attaquent mutuellement) a été fait. Dans le cas où les huit dames ont été placées sans qu'elles ne s'attaquent mutuellement, l'interface indiquera que l'utilisateur a gagné.

   Il y a peu de spécifications et d'obligations pour l'interface graphique tant qu'elle répond à la problématique initiale de l'application et qu'elle est utilisable.

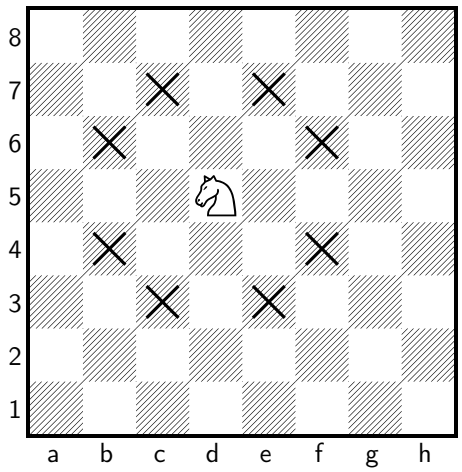
   Optionnel : on permettra à l'utilisateur d'annuler un nombre de coups arbitraire en utilisant un bouton *précédent*. Cette fonction peut par exemple utiliser un tableau de huit positions représentant les différentes étapes du placement des dames.

4 Améliorations

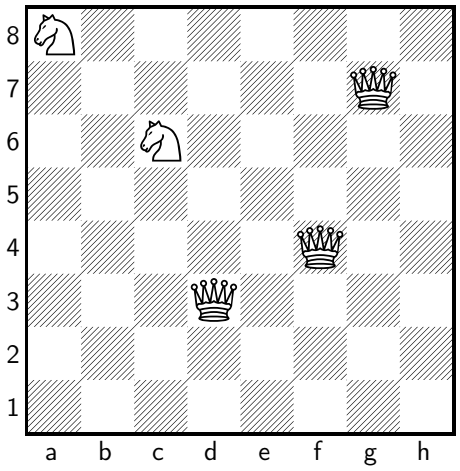
Exercice 6. (Des cavaliers en plus)

Cet exercice est optionnel, il n'est a envisager que lorsque tout le reste fonctionne parfaitement.

Un cavalier (aux échecs, attention, on dit « cavalier » et non pas « cheval » !) attaque toutes les cases situées « en `L` » par rapport à celle où il se trouve. Voici par exemple les cases attaquées (cases contenant une ×) par un cavalier en d5 :



1. Refaire l'ensemble du TP avec des cavaliers à la place de dames. Le but du jeu est maintenant de placer le maximum de cavaliers sans qu'aucun n'attaque un autre.
2. Mélanger maintenant les dames et les cavaliers dans une même position. L'utilisateur peut placer, selon ses envies, des dames et des cavaliers dans une même position. L'objectif est de construire des configurations ayant un maximum de dames et de cavaliers qui ne s'attaquent pas mutuellement. Par exemple, la position suivante est valide :



Indication : une position est maintenant représentée non pas par un seul nombre de 64 bits mais par deux. L'un contient les cases occupées par les dames tandis que l'autre contient les cases occupées par les cavaliers.

Pour ceux qui veulent aller plus loin, ces entiers de 64 bits que nous avons utilisés pour représenter un ensemble de pièces aux échecs sont connus sous le nom de *bitboards*. Les programmes de classe mondiale actuels qui jouent aux échecs utilisent des bitboards (sous certains détails plus ou moins sophistiqués) pour représenter des positions de manière véritablement efficace.