TP n°1 : Permutations et placement des dames sur un échiquier

Ce sujet fait directement référence au TD n°1-2 : Piles dynamiques et parcours d'arbre de décision.

Il a essentiellement pour but de préciser la démarche de réalisation d'un module de gestion du TAD pile dynamique contigüe et deux applications utilisant le même algorithme de parcours d'un arbre de décision.

1 Consignes générales pour le déroulement des TP SDA2

En début de séance, établissez un **répertoire de travail** intitulé `SDA2_TPx_MON_NOM` (avec x représentant le numéro de TP).

Pensez à analysez avant de coder. Votre code doit être préparé pour une génération de la **documentation doxygen**.

Pour chaque fonction développée, vous fournirez une **fonction de test**.

Au fur et à mesure de la séance, vous réaliserez des **captures d'écran** de vos tests que vous assemblerez dans un document word.

A la fin de la séance, vous ferez un **dépôt** de votre **répertoire compressé** contenant votre code ainsi qu'une trace des différents tests effectués.

Vous avez **une semaine pour déposez votre CR** et éventuellement une amélioration de votre code.

Tous vos programmes utiliserons la fonction `epilogue()` fournie par la plateforme moodle. Cette fonction devra s'exécuter à la fin de **chaque** exécution de votre programme. Elle sert à afficher certaines informations comme votre nom, la date, le poste de travail. Pour l'utiliser, il suffit d'écrire `atexit(epilogue) ;` comme première instruction du `main()` :

```
void epilogue() ;
int main ()
{
      atexit (epilogue) ;
      // votre code
}
```

2 Le module de gestion de pile dynamique

Écrire et tester le module de gestion de pile dynamique tel qu'il a été présenté en TD.

Vous devez développez un module de test pour la gestion de ce TAD.

À noter :

1. la définition dans *pile.h*, des constantes symboliques définissant les seuils d'allocation et de libération : par exemple 5 pour le premier et 10 pour le second.
2. la spécification différente de la fonction d'initialisation (voir **Erreur ! Source du renvoi introuvable.**)
3. l'ajout d'une fonction supplémentaire permettant d'obtenir la valeur du i*ème* élément empilé.

```
elt_t valeurSommet (pile_t Pile, unsigned int i) ;
```

3 Permutations

Les éléments à permuter sont les caractères d'un mot fourni par l'utilisateur lors de l'exécution du programme. Dans le programme, ces caractères seront désignés par leur **rang** dans le mot fourni par l'utilisateur.

Autrement dit, le programme ne manipulera pas directement les caractères mais leurs indices dans le processus de recherche des permutations : la pile utilisée est donc bien une pile d'entiers.

La stratégie de parcours de l'arbre de décision sera systématique et identique pour les deux applications à réaliser :

- l'opération *Passer au premier fils*, consistera à choisir le premier élément de l'ensemble à permuter (celui dont l'indice vaut 0 ou 1, selon la convention adoptée).
- et *Passer au frère suivant* permettra de choisir l'élément dont le n° suit.

Le contrôle de validité du nœud est toujours effectué a posteriori ; il est d'ailleurs différent de celui du placement des dames sur l'échiquier, comme le sera également la fonction d'affichage des solutions. Nous allons donc regrouper dans le fichier *parcours.h*, les spécifications des traitements et prédicats communs aux deux applications. Définir dans ce fichier, sous forme de macro-fonctions, les traitements suivants :

```
#define Passer1erFils(pile) ...
#define RemonterPere(pile) ...
#define PasserFrereSuivant(pile) ...
```

Définir sous forme de macro-fonctions, les prédicats suivants :

```
#define NaPlusDeFrere(pile, NbElt)
#define estTerminal(pile, NbElt)
#define CestPasFini(pile)
```

```
( ... )
```

Le fichier principal de cette première application, *permut.c* sera constitué des fonctions spécifiques à ce problème et de la fonction principale.

La fonction principale se charge :

- De lire le mot fourni par l'utilisateur, représentant l'ensemble à permuter
- De calculer le nombre d'éléments à permuter
- D'effectuer le parcours de l'arbre de décision en affichant les permutations obtenues
- D'afficher, lorsque l'exploration est terminée le nombre total de solutions affichées.

La fonction *estValide* détermine si le nœud courant est valide, c'est-à-dire si le dernier choix empilé n'avait pas déjà été effectué.

Son prototype sera le suivant : `int estValide (const pile_t *pPile)`

Enfin, la dernière fonction présentée ici, mais à sans doute développer et tester en premier, est la fonction permettant l'affichage de la solution correspondant au nœud terminal atteint et au mot initial fourni par l'utilisateur. Elle retournera comme solution le numéro de la solution affichée.

Son prototype sera le suivant :

```
unsigned int affiche1Sol (const pile_t *pPile, const char *pMot);
```

À noter :

> Cette fonction utilisera la fonction `valeurSommet` qui a été ajoutée au module de gestion de pile. Il est conseillé dans un premier temps de ne pas s'occuper du résultat à retourner ni de la présentation des résultats.

L'utilisation d'une variable statique dans la fonction `affiche1Sol,` déclarée comme suit : `static unsigned int numSol = 0 ;` permettra de retourner le numéro de la solution affichée et sera utile pour faire en sorte d'afficher *n* solutions par ligne, avec *n* dépendant de la longueur du mot. Pour cela, la fonction d'affichage génèrera un saut de ligne lorsque le numéro de la solution affichée sera un multiple du nombre de termes pouvant être affichés par ligne sur l'écran de la console d'affichage.

Note : En mode console, on peut afficher jusqu'à 80 caractères par ligne.

Exemple

```
Entrez le mot à permuter: lacet
lacet lacte laect laetc latce latec lcaet lcate lceat lceta lctae lctea
leact leatc lecat lecta letac letca ltace ltaec ltcae ltcea lteac lteca
alcet alcte alect aletc altce altec aclet aclte acelt acetl actle actel
aelct aeltc aeclt aectl aetlc aetcl atlce atlec atcle atcel atelc atecl
claet clate cleat cleta cltae cltea calet calte caelt caetl catle catel
celat celta cealt ceatl cetla cetal ctlae ctlea ctale ctael ctela cteal
elact elatc elcat elcta eltac eltca ealct ealtc eaclt eactl eatlc eatcl
```

```
eclat eclta ecalt ecatl ectla ectal etlac etlca etalc etacl etcla etcal
tlace tlaec tlcae tlcea tleac tleca talce talec tacle tacel taelc taecl
tclae tclea tcale tcael tcela tceal telac telca tealc teacl tecla tecal
Durée de la recherche = 0.110 s
Il y a 120 solutions.
```

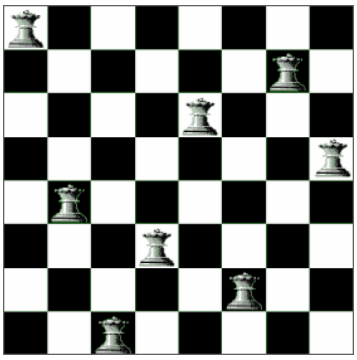
4 Les huit dames

Il s'agit maintenant de concevoir et de réaliser le programme permettant d'obtenir les positions de 8 dames sur un échiquier en faisant en sorte qu'aucun couple de dames ne soit en « prise ».

On rappelle que dans le jeu d'échecs, une dame peut se déplacer en ligne droite sur toute ligne, colonne ou diagonale de l'échiquier. Deux dames sont donc « en prise » si elles sont situées sur la même ligne ou la même colonne ou encore la même diagonale.

Ce problème a été posé pour la première fois en 1848 par **Max Bezzel** dans un journal d'échecs de la *Berliner Schachgesellschaft*. Il a été résolu en 1850 par **Franz Nauck** et publié dans la revue allemande *Illustrierten Zeitung*. **Carl Friederich Gauss** (1777-1855), intéressé par cette « colle » n'aurait trouvé que 72 solutions.

4.1 Principe



En partant du principe que l'on ne peut mettre qu'une seule reine par ligne, le problème se ramène à la recherche des numéros de colonnes successifs utilisés pour placer les huit dames.

Indication

Le programme de recherche des permutations précédent permet le placement de n tours sur un échiquier $n \times n$. Une simple adaptation de ce programme permettra de traiter le problème des dames.

Cette seconde application réutilisera le module de gestion de pile ainsi que le fichier *parcours.h*. Seul le fichier principal sera différent même si sa structure générale est identique à celle de l'application précédente.

Conseil : Créer une copie de *permut.c* que vous nommerez *dames.c*

4.2 Travail à effectuer.

1. Modifiez la fonction *estValide* de manière à ce qu'elle contrôle également les deux diagonales et testez le programme avec cette seule modification. La longueur du mot tapé au clavier servira à fixer la taille de l'échiquier et le nombre de dames à placer.
2. Modifiez le programme principal, de façon à n'avoir que la taille de l'échiquier et le mode d'affichage (voir plus bas) à fournir.
3. Développez une fonction d'affichage d'une solution en notation échiquéenne,
4. Développez une fonction d'affichage pseudo-graphique d'une solution.
5. Prévoir un mode sans affichage des solutions, uniquement destiné à déterminer le nombre de solutions et une durée de recherche ne tenant pas compte des durées d'affichage.
6. Le compte-rendu présentera le nombre de solutions obtenues et la durée de la recherche (dans le mode sans affichage) pour toutes les tailles d'échiquier allant de 4×4 et 15×15.
7. Vous donnerez une estimation de la durée de la recherche pour un échiquier 30×30.

4.3 Annexes : quelques exemples d'affichage...

4.3.1 Exemple 1

```
Entrez la taille de l'échiquier (15x15 max): 7
Choix de l'affichage
        Tapez 1 pour un affichage en notation échiquéenne
        Tapez 2 pour un affichage pseudo graphique
        Tapez 3 pour ne pas afficher les solutions
Votre choix : 1
a1c2e3g4b5d6f7   a1d2g3c4f5b6e7   a1e2b3f4c5g6d7   a1f2d3b4g5e6c7
b1d2a3g4e5c6f7   b1d2f3a4c5e6g7   b1e2a3d4g5c6f7   b1e2c3a4g5d6f7
b1e2g3d4a5c6f7   b1f2c3g4d5a6e7   b1g2e3c4a5f6d7   c1a2f3b4e5g6d7
c1a2f3d4b5g6e7   c1e2g3b4d5f6a7   c1f2b3e4a5d6g7   c1g2b3d4f5a6e7
c1g2d3a4e5b6f7   d1a2c3f4b5g6e7   d1a2e3b4f5c6g7   d1b2g3e4c5a6f7
d1f2a3c4e5g6b7   d1g2c3f4b5e6a7   d1g2e3b4f5a6c7   e1a2d3g4c5f6b7
e1a2f3d4b5g6c7   e1b2f3c4g5d6a7   e1c2a3f4d5b6g7   e1g2b3d4f5a6c7
e1g2b3f4c5a6d7   f1a2c3e4g5b6d7   f1b2e3a4d5g6c7   f1c2a3d4g5e6b7
f1c2e3g4a5d6b7   f1c2g3d4a5e6b7   f1d2b3g4e5c6a7   f1d2g3a4c5e6b7
g1b2d3f4a5c6e7   g1c2f3b4e5a6d7   g1d2a3e4b5f6c7   g1e2c3a4f5d6b7
Durée de la recherche =     0.000 s
Il y a 40 solutions
```

4.3.2 Exemple 2

```
Entrez la taille de l'échiquier (15x15 max): 4
Choix de l'affichage
        Tapez 1 pour un affichage en notation échiquéenne
        Tapez 2 pour un affichage pseudo graphique
```

```
        Tapez 3 pour ne pas afficher les solutions
Votre choix : 2
Solution 1
          A   B  C   D
        +---+---+---+---+
     1  | . | R | . | . |
        +---+---+---+---+
     2  | . | . | . | R |
        +---+---+---+---+
     3  | R | . | . | . |
        +---+---+---+---+
     4  | . | . | R | . |
        +---+---+---+---+
Solution 2
          A   B  C   D
        +---+---+---+---+
     1  | . | . | R | . |
        +---+---+---+---+
     2  | R | . | . | . |
        +---+---+---+---+
     3  | . | . | . | R |
        +---+---+---+---+
     4  | . | R | . | . |
        +---+---+---+---+
Durée de la recherche = 0.000 s
Il y a 2 solutions
```

4.3.3 Exemple 3

```
Entrez la taille de l'échiquier (15x15 max): 14
Choix de l'affichage
        Tapez 1 pour un affichage en notation échiquéenne
        Tapez 2 pour un affichage pseudo graphique
        Tapez 3 pour ne pas afficher les solutions
Votre choix : 3
Durée de la recherche =    38.725 s
Il y a 365596 solutions
```