



Université
de Limoges

FACULTÉ
DES SCIENCES
ET TECHNIQUES

Département d'Informatique

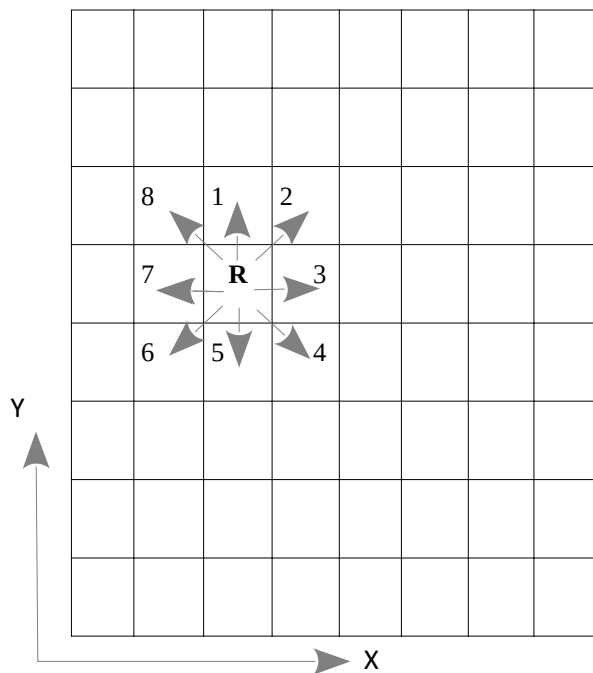
Devoir Blanc Novembre 2014

Durée : 1 heure 30 Cours, documents autorisés

NB : une rédaction soignée des algorithmes est exigée.

Question 1 (7 points)

Sur un échiquier, le Roi se déplace dans toutes les directions d'**une seule case** comme ci-dessous.



Écrire un algorithme qui :

- demande à l'utilisateur la position initiale du Roi et vérifie que la position est sur l'échiquier. Si ce n'est pas le cas, on redemandera la position à l'utilisateur.
- puis en boucle 10 fois de suite :
 - demande la direction à prendre :1, 2, 3, 4, 5, 6, 7 ou 8 et vérifie que la direction soit valide (si ce n'est pas le cas, on redemandera une direction à l'utilisateur) ;
 - puis soit calcule et affiche sa nouvelle position, soit envoie un message d'erreur si le déplacement aboutit à faire sortir la pièce de l'échiquier.
- Affiche « Gagné ! » si les 10 déplacements ont pu être effectués sans que le Roi ne sorte de l'échiquier et « Perdu après B coups ! » sinon (où B est le nombre de coups joués avant de sortir de l'échiquier).

Remarque : Évidemment si le Roi aboutit hors de l'échiquier lors d'un de ses déplacements, l'algorithme ne redemandera pas l'utilisateur de saisir un nouveau déplacement.

Question 2 (3 points)

Exprimer les conditions suivantes en les développant sous leur forme la plus simple (c'est à dire en les simplifiant au maximum). Il est d'ailleurs peut être possible de donner directement le résultat (**true** ou **false**) de certaines de ces expressions. On suppose que X est une variable de type **char** et que A et B sont de type **int**. On notera que and, or et ! peuvent s'écrire respectivement **&&**, **||** et **not**.

1. ! ((X=='C') and ((B>A) or (A>2)))
2. ! (!(2==2) and (A<10))
3. ! ((X=='X') or (X!='X'))
4. ! (!(!(A / B!= 0) or (!(A<2) or !(X=='C'))))
5. ! (false or (!(A==B) and (B>=10)))
6. ! (((A>B) and (B<10)) and !((A>B) and (B<10)))

Question 3 (4 points)

Transformez la structure «switch» suivante en une équivalence utilisant seulement les structures «if ... else ...» et/ou «if ... » sachant que x est de type int. (*utilisez cette feuille pour répondre*)

```
switch (x)
{
    case 2: x=3;
           break;
    case 3: x=5;
    case 1: if (x==5)
           x=3;
           break;
    case 7: x=x+1;
           break;
    default: x=1;
}
```

Question 4 (6 points) :

La suite de Fibonacci a historiquement été définie pour calculer la croissance de populations de lapins. Le $i^{\text{ème}}$ terme de cette suite représente la population de lapins après i semaines. Le $i^{\text{ème}}$ terme de la suite est défini comme étant la somme des deux termes précédents (pour $i \geq 2$ avec $\text{Fibonacci}(0) = 0$ et $\text{Fibonacci}(1) = 1$).

- 1) Déclarez un tableau de taille 100 et remplissez les cases du tableau avec les termes de la suite de Fibonacci. La $i^{\text{ème}}$ case du tableau doit représenter la population de lapin à la $i^{\text{ème}}$ semaine (c.a.d. la $i^{\text{ème}}$ case du tableau doit contenir $\text{Fibonacci}(i)$).
- 2) Un éleveur de lapins souhaite avoir une application permettant de connaître le nombre de semaines qu'il lui faudra attendre pour avoir au moins N lapins. Proposez une application permettant cette fonctionnalité. L'utilisateur doit pouvoir :
 - Entrer un nombre de lapins souhaités et savoir le nombre de semaines nécessaires pour avoir autant de lapins.
 - Choisir de recommencer cette opération autant de fois qu'il le souhaite (Utilisez pour cela un menu laissant le choix à l'utilisateur de recommencer le calcul).
- 3) (Bonus) Notre éleveur est particulièrement affamé et décide certaines semaines de manger certains lapins. Modifiez votre application pour laisser le choix à l'utilisateur de manger k (à définir par l'utilisateur) lapins lors de la semaine t (à définir par l'utilisateur) et recalculez la suite de Fibonacci à partir de la semaine t .