TP4 Pointeurs et piles

On veut réaliser un programme permettant à une souris (repérée par le symbole S) de trouver un morceau de fromage (repéré par F) dans un labyrinthe contenant des emplacements vides (repérés par un espace) et des obstacles infranchissables (repérés par *).

```
**********
* * **  S*
***  * * *
* * ** * *
* **   * *
*  **  * *
* *   * **
*   * * **
*F* **** *
**********
```

À partir d'une case de départ spécifiée dans le labyrinthe, la souris se déplace de case en case jusqu'à arriver sur la case où se trouve le fromage (si elle arrive à le trouver). La souris ne se déplace que d'une case à la fois dans une direction : nord, sud, est, ouest ; elle n'a pas de vue globale du labyrinthe et ne « voit » que les cases contigües à sa position.

1) Le labyrinthe est représenté par un tableau carré à deux dimensions, chaque case du tableau est identifiée par un couple de coordonnées. Définir une constante **TailleLabyrinthe** comme un entier valant 10 et le type **Labyrinthe** comme représentant un tableau à deux dimensions de caractères, de taille TailleLabyrinthe x TailleLabyrinthe.
2) Écrire un sous-programme **void labyrintheCreerVide(Labyrinthe & lab)** qui initialise un labyrinthe avec des obstacles ("*") placés autour du labyrinthe et des emplacements vides (" ") à l'intérieur.
3) Écrire un sous-programme **void labyrinthePlacerObstacles(Labyrinthe & lab, int pobs)** prenant comme paramètre un pourcentage d'obstacles (pobs compris entre 0 et 100) à placer aléatoirement sur les emplacements vides du labyrinthe.
4) Écrire un sous-programme **void labyrinthePlacerSourisFromage(Labyrinthe & lab)** plaçant dans le labyrinthe la souris ("S") et le fromage ("F") sur des emplacements libre déterminés aléatoirement.
5) Écrire un sous-programme **void labyrintheAfficher(Labyrinthe lab)** affichant à l'écran le labyrinthe.
6) Une position dans le labyrinthe est repérée par un couple de coordonnées : colonne, ligne. Définir un type **Position** repérant une position dans le labyrinthe. Écrire un sous-programme **positionDepart** retournant la Position correspondant à l'emplacement de la souris.
7) Le chemin parcouru par la souris sera mémorisé dans une pile de Position. Une pile est une liste dont les seules opérations applicables sont : ajout d'une position en tête de liste (*empiler*), accéder à la première position de la liste (appelée le *sommet* de la pile), enlever la première position de la pile (*dépiler*). Définir un type **PilePosition** permettant de représenter une pile de positions.

8) Écrire les sous-programmes **pileInitialiser** créant une pile de positions vide, **pileEstVide** testant si une pile est vide, **pileEmpiler** permettant d'empiler une position dans une pile, **pileSommet** retournant le sommet d'une pile, **pileDepiler** dépilant la position au sommet d'une pile, **pileVider** qui vide totalement une pile.

9) Écrire une fonction **positionAExplorer** qui à partir d'un Labyrinthe et d'une position dans ce labyrinthe retourne une position contigüe (c'est-à-dire au nord, au sud, à l'est ou à l'ouest de la position passée) dans laquelle la souris peut se déplacer (c'est-à-dire que la case correspondante du labyrinthe est " " ou "F"). Si la souris ne peut se déplacer dans aucune position contigüe, la fonction retournera la position particulière (-1,1).

10) Écrire un sous-programme **labyrintheResolution** qui à partir d'un Labyrinthe va déplacer la souris jusqu'à ce qu'elle trouve le fromage ou qu'elle se rende compte qu'il n'y a pas de solution pour le trouver. L'idée de la résolution est la suivante : tant que la souris n'a pas trouvé le fromage, elle essaie de se déplacer dans une case contigüe dans laquelle elle n'est pas encore passée. Afin de savoir dans quelles cases elle est déjà passée, quand elle quitte une case, elle laisse une marque dans la case qu'elle quitte afin de repérer qu'elle est passée par là, ce qui va être représenté par un "." placé dans la case du labyrinthe. Dans certains cas, aucune case contigüe ne sera accessible : soit parce qu'il s'agit d'un cul-de-sac, soit parce que toutes les cases contigües ont déjà été explorées. Dans un cas comme dans l'autre, elle devra donc revenir sur ses pas (en utilisant la pile contenant les dernières positions explorées) à la recherche d'une case contigüe accessible. Si par contre, elle trouve une position contigüe accessible, elle empilera la position quittée afin d'être capable de revenir en arrière si besoin. Après chaque mouvement de la souris, qui se traduit par un déplacement du "S" dans le labyrinthe, vous afficherez le labyrinthe, ce qui vous aidera à corriger un comportement incorrect de la souris. Réfléchir aux deux cas dans lesquels la résolution se termine.