Examen final Projet de Programmation – IF142
DEUG
——

Nous vous rappellons que **les téléphones portables sont interdits** dans les salles d'examen. Veillez à éteindre le vôtre et à le ranger dans votre sac.

Pour cette épreuve, les calculatrices ne sont pas autorisées. En revanche tous les documents personnels sur support papier sont autorisés.

Les réponses aux questions devront être justifiées. La clarté et la précision de ces explications seront prises en compte dans la notation. **Les programmes devront impérativement être expliqués et commentés**, sous peine de ne pas être notés.

durée de l'épreuve : 2h30 juin 2003

Pour une question donnée, vous pouvez utiliser les fonctions demandées précédemment, même si vous ne les avez pas écrites.

1 Lecture de programme

▸ **Exercice 1.** Qu'affiche le programme suivant (justifier brièvement) ? 1+1+1+1

```
#include <iostream>

int main()
{
  int i = 0;
  int j = 1;
  int k = 2;
  int *p = &i;
  int *q = &j;
  int *r = &k;

  cout << i << " " << j << " " << k << endl;
  *r = *p;
  *p = *q;
  *q = *r;
  cout << i << " " << j << " " << k << endl;
  r = p;
  p = q;
  q = r;
  cout << i << " " << j << " " << k << endl;
  *r = *p;
  *p = *q;
  *q = *r;
  cout << i << " " << j << " " << k << endl;
}
```

2 Tetris

Le but de cette partie est d'implémenter quelques fonctions permettant de programmer un jeu de `Tetris` simplifié.

On dispose d'un "plateau" de jeu qui est composé de $n+1$ emplacements, numérotés de 0 à n ($n \in \mathbb{N}$ fixe). Au-dessus de chaque emplacement, on peut potentiellement empiler une infinité de pièces. Les pièces ont une forme rectangulaire : elles sont de hauteur 1 et de largeur un entier strictement positif. Les pièces tombent verticalement de haut en bas et s'arrêtent dès le premier obstacle rencontré en restant en position horizontale. Les mouvements latéraux ne sont pas autorisés (on supposera qu'avant la chute d'une pièce, le joueur précise l'emplacement sur lequel elle tombe). Ceci est illustré par la figure 1. On dit qu'une pièce tombe à l'emplacement k si la partie la plus à gauche de la pièce tombe dans la colonne k; autrement dit, la pièce s'étale de la colonne k à une colonne $k' \geq k$, k' dépendant de sa largeur.

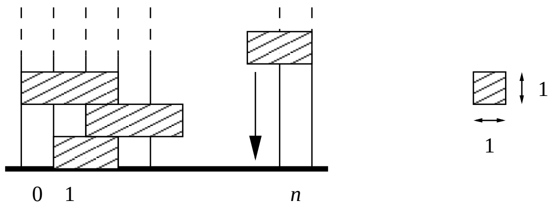

Figure 1: Une situation possible dans le `Tetris` simplifié

▸ **Exercice 2.** On s'intéresse au *profil supérieur* (ou *profil*) d'un jeu, c'est-à-dire à la suite finie de $n+1$ entiers, tel que l'entier numéro k est égal à la plus grande ordonnée occupée dans la colonne k. Ainsi, tant qu'aucune pièce n'est tombée, le profil du jeu est

$$p = (\underbrace{0, 0, \ldots, 0}_{n+1}).$$

Si une pièce de largeur 2 tombe à l'emplacement 1, le nouveau profil est

$$p' = (0, 1, 1, \underbrace{0, 0, \ldots, 0}_{n-2}).$$

Si p est un profil, on notera p_i $(0 \leq i \leq n)$ la i^{e} composante de p.

a. Proposer un type pour décrire un profil. On appellera `profil` ce type. 1

b. Soit un profil p. On suppose qu'une pièce de largeur l tombe à l'emplacement c. On veut calculer le nouveau profil p' du jeu. Pour chaque i entre 0 et n, donner la valeur de p'_i. 1

c. Ecrire une fonction

```
profil ajout_piece(profil p, int larg, int col);
```

qui renvoie le nouveau profil obtenu après la chute d'une pièce de largeur `larg` à l'emplacement `col` sur le profil `p`. 1

▸ **Exercice 3.** Quand une ligne du jeu est pleine, c'est-à-dire qu'à une hauteur donnée toutes les colonnes sont occupées, on veut la faire disparaître du jeu et décaler d'une unité vers le bas tout ce qui se trouve au-dessus, cela en défiant les lois de la pesanteur, puisque certaines pièces peuvent alors se trouver au-dessus d'un trou sans y tomber. La figure 2 illustre ce principe.

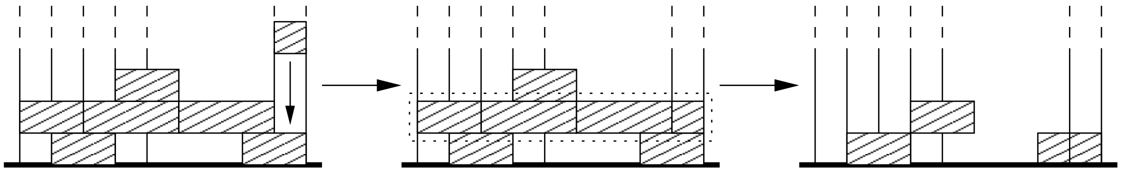

Figure 2: Disparition d'une ligne pleine

a. Pour pouvoir supprimer une ligne, on a besoin de savoir quelles sont les cases occupées. Pour cela, on modifie le type `profil` introduit à l'exercice 2, pour qu'en plus de la hauteur associée à chaque colonne, on ait la liste des emplacements occupés (sous forme de listes chaînées, une liste chaînée par colonne). Ecrire ce nouveau type et adaptez la fonction
`profil ajout_piece(profil p, int larg, int col);`
de l'exercice 2 (sans suppression de ligne). 2

Désormais, le type `profil` est le nouveau type que vous venez d'introduire.

b. Ecrire une fonction
`bool case_occupee(profil p, int haut, int col);`
qui renvoie `true` si et seulement si la case de la colonne `col` à la hauteur `haut` est occupée dans le profil `p`. 1

c. Ecrire une fonction
`bool ligne_occupee(profil p, int haut);`
qui renvoie `true` si et seulement si la ligne de hauteur `haut` est entièrement occupée. 1

d. Ecrire une fonction
`profil supprime_ligne(profil p, int haut);`
qui renvoie le profil obtenu après suppression de la ligne de hauteur `haut` dans le profil `p`. 1,5

e. Ecrire une fonction
`profil tetris(profil p, int larg, int col);`
qui renvoie le nouveau profil obtenu après la chute d'une pièce de largeur `larg` à l'emplacement `col` sur le profil `p` et suppression des éventuelles lignes pleines. 1,5

3 Serpent

On s'intéresse à un autre jeu bien connu.

Dans ce jeu, une entité représentant un serpent se déplace sur votre écran. Le joueur ne peut agir que sur sa direction, en appuyant sur une des touches H (haut), B (bas), G (gauche), D (droite). Lorsqu'une direction est choisie, le serpent la suit au cours du temps, jusqu'à ce qu'une nouvelle direction lui soit indiquée. Il est donc possible que le serpent vienne rencontrer un des bords de l'écran. Il a alors perdu.

Au cours d'une phase normale de jeu, le corps du serpent a une taille stable plutôt importante, et son déplacement se fait de proche en proche : si on décompose son corps en une suite de pixels, le premier d'entre eux étant sa tête, lors du mouvement la tête va avancer d'un pixel dans la direction courante, puis chaque pixel du corps va prendre la place de celui qui le précède : le second venant prendre la place de la tête, le troisième la place du second etc... le dernier la place de l'avant dernier. La figure 3 illustre ce principe.

Lorsque le serpent rencontre certains obstacles présents sur l'écran, son corps s'allonge, et l'obstacle réapparaît ailleurs. Le joueur a pour but de ramasser le plus possible de ces obstacles.

Le corps s'allongeant, il devient possible que le joueur guide la tête du serpent vers un emplacement déjà occupé par une partie de son corps. On choisit alors de faire perdre le joueur.

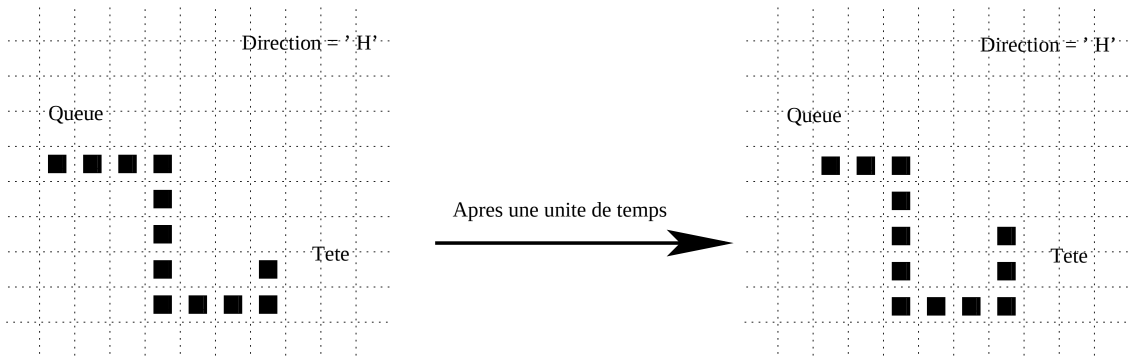

Figure 3: Déplacement du serpent un pas en avant

▸ **Exercice 4.**

a. Définir un type serpent (et ses sous types) comme étant une structure contenant son corps sous la forme d'une liste, et sa direction. 1

b. On suppose qu'on dispose des fonctions et procédures suivantes :

`bool boom(serpent s);` qui retourne vrai si le serpent rencontre un obstacle qui le fait perdre
`char get_key_pressed();` qui permet de savoir quelle est la dernière touche appuyée
`void avance_serpent(serpent s);` qui s'occupe du déplacement du serpent

Ecrire la fonction `main` qui intègre ces fonctions et procédures, et qui forme la partie principale du jeu. 1

c. Ajouter des informations à votre programme pour intégrer des limites pour le cadre de l'écran et écrire la fonction `boom`, qui teste si le serpent rencontre le cadre, ou une partie de son corps. 1

d. Ecrire une version de `avance_serpent`, conforme à ce qui a été décrit dans la présentation. 1

e. On gère à présent le contact du serpent avec l'obstacle dont il se nourrit pour s'allonger (la pastille). Vous avez accès à des procédures `affiche_pastille(int x_pastille, int y_pastille)`, et `new_pastille(int &x_pastille,int &y_pastille, serpent s)` qui crée une nouvelle pastille à un endroit libre.
Lors du contact avec la pastille tout se passera comme si seule la tête du serpent avançait, sans que le reste de son corps ne bouge. Il faudra tout de même que le corps du serpent reste cohérent et que sa taille augmente d'une unité. Modifier votre programme pour prendre en compte la gestion de la pastille. Il devra toujours y avoir une pastille présente à l'écran. 2

f. On cherche maintenant à améliorer la gestion du mouvement du serpent. Tout part de la remarque qu'il est suffisant d'effacer le bout de la queue du serpent, et d'ajouter une tête, c'est-à-dire ne modifier la liste qui représente le corps du serpent qu'à ses extrémités.

Définir un type liste chaînée, qui contienne un pointeur début et un pointeur fin, pointant respectivement vers le début et la fin de la liste. Si la fin de la liste représente la tête du serpent, et la tête de la liste sa queue, écrire une nouvelle version de `avance_serpent`. 2