Controle Continu IF122

Les documents sont interdits. La durée de l'épreuve est de 1h50. Le barème est donné à titre indicatif. Bonne chance.

Exercice 1. Pointeurs (10 minutes, 4 points)

On considère le programme suivant, qui comporte quelque lignes en commentaire :

```
#include <iostream>
//void toto( ... x, ... y){
//  **x = *y;
//  }
int main(){
 int i=2;
 int j=12;
 int *pi= &i;
 int *ln= new int, e[]={3,1,7,9,14};

 pi = ln;
 *ln = 4;
 cout << *pi << " , " << *ln << endl;
 *pi=j+1;
 pi=&i;
 cout << *pi << " , " << *ln << endl;
 ln=e;
 cout << *(ln+*pi) << " , " << *ln + *pi << endl;

 // toto ( & pi, e);
 // cout << e[0] << endl;
  return 0;
}
```

1. Qu'affiche t'il ? Justifier rapidement chaque étape de votre réponse.
2. Ce programme comporte une erreur, laquelle ?
3. On veut décommenter les lignes, quel type doit on donner aux arguments de **toto** ? Que produit le dernier affichage ?

Exercice 2. Récursion (20 minutes) (4 points)

Un esprit tordu a écrit une partie du code suivant :

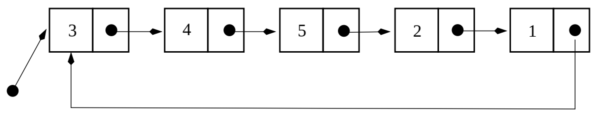
```
void affiche(int T[],int n) // procedure qui affiche les n elements du tableau
                            // n est suppose < dimension du tableau T

// procedure bizarre
void proc (int T[],int n, int c){   // n est < dim de T
int i;
if (c<=0) affiche(T,n);
 else {
   for (i=1;i<=n;i++){
   T[c-1]=i;
   proc(T,n,c-1);
   }
 }
}
```

1. Ecrivez le code pour la procedure affiche. En itératif, puis en récursif.
2. Qu'est ce qui est affiché lors de l'exécution de **proc(T,2,2)** ? Il est demandé de justifier avec précaution votre réponse, par exemple en dessinant un arbre associé à l'exécution.

Exercice 3. Liste - (Presque question de cours $<$ 20 minutes) (4 points)

Une **liste circulaire** se définit presque comme une liste chainée : elle est composée d'une liste de cellules chainées entre elles, sauf que ce qui pourrait être le dernier élément de cette liste a pour successeur le premier élément. Voici un exemple :



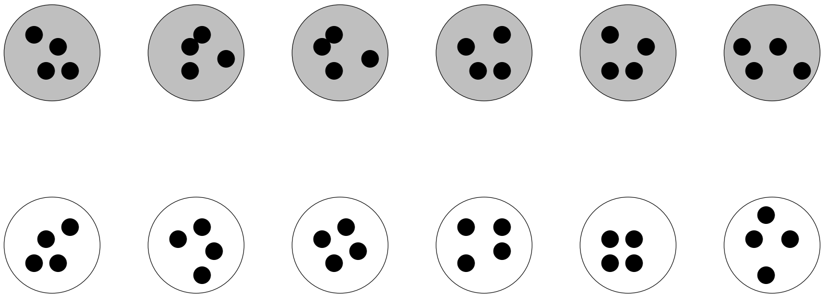
Il était facile de parcourir une liste chainée, puisque sa fin était marquée par NULL. Il faudra maintenant trouver une alternative.

1. Ecrivez les types nécessaires pour une liste d'entiers circulaire.
2. Ecrivez une procédure qui affiche une liste d'entier circulaire.

Exercice 4. Problème : Structure de données - Modélisation (1h00) (8 points)

On presente ici le **mancala** : ce jeu se joue à deux joueurs, face à face, avec un plateau entre eux (voir figure). Le plateau est composé de 2 lignes de 6 cases, initialement chacune des cases contient 4 graines, la ligne la plus proche d'un joueur est son camp.

Joueur 2



Joueur 1

Chacun à son tour un joueur va prendre toutes les graines d'une des cases de son camp, et va les distribuer dans le sens inverse des aiguilles d'une montre, en partant de la case suivante de celle qu'il à vidé.

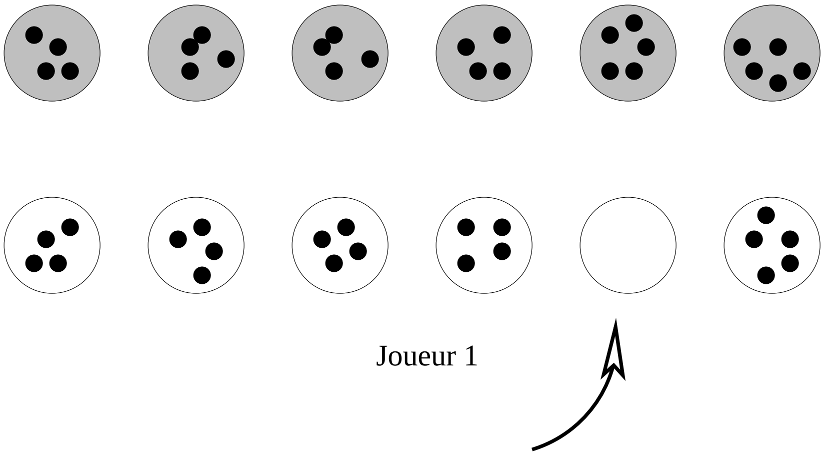
Sauf que pendant la redistibution des graines, lorsqu'il passe d'une case de son camps à une case du camps adverse, il engrange définitivement l'une des graines en se servant parmis celles qui lui reste à distribuer, puis il poursuit la distribution.

Symétriquement si pendant la distribution il doit passer d'une case du camps adverse à une de son camp, alors l'adversaire engrange l'une de ces graines.

Le jeu s'arrête lorsqu'un joueur ne peut plus jouer. On fait alors pour chaque joueur la somme des graines qu'il a mis de coté, avec celles qui lui reste dans son camp. Le gagnant est celui qui en a le plus.

Un exemple de phase de jeu : si le joueur 1, joue la case marquée par une fleche, alors après la distribution, il aura mis une graine de coté, et laissera le plateau suivant pour le joueur 2 :

Joueur 2



Joueur 1

1. A votre convenance, définissez un type de données qui permette de modéliser à la fois l'état du plateau, les gains engrangés, et le tour de jeu. **Attention**, comme il y a de nombreuses solutions, vous devrez défendre votre choix et bien expliquer à quoi correspondent votre (ou vos) types. (2 points)
2. Ecrire une fonction **reset** qui permettra de commencer avec un jeu dans sa configuration initiale. (1 point)
3. Ecrire une fonction **test_fin** qui permet de savoir si le jeu est fini ou non. (1 point)
4. Ecrire une fonction **qui_gagne** qui retourne le numero du joueur gagnant lorsque le jeu est terminé, -1 si on ne sais pas encore, et 0 en cas d'egalité. (1 point)
5. Ecrire une procédure **joue**, avec deux arguments : l'état du jeu, et un moyen d'identifier le coup proposé, qui a pour effet de changer l'état en fonction du coup. (2 points)
6. Donner un programme complet. (1 point)