TP3 – Les tableaux

Projet de programmation M1

6 Octobre 2015

Un tableau est une structure de données permettant de stocker plusieurs éléments du même type. Un tableau a une longueur n fixée, c'est le nombre d'éléments qu'il contient. On peut accéder au $i$ème élément d'un tableau `t` en écrivant `t[i]` ; on dit que i est un indice. Le premier indice d'un tableau est le 0 (le dernier est donc $n-1$). Attention toutefois, en C, rien ne vous empêche d'écrire `t[25]` même si `t` est de longueur 10. GCC ne râlera pas et il se peut même que votre programme s'exécute sans planter. En revanche, vous risquez d'être confrontés à un moment ou un autre à des bugs inexpliqués : **faites toujours attention à rester dans les limites du tableau**.

On déclare un tableau ainsi : `type t[taille];`. Par exemple `int t[10];` désigne un tableau de dix entiers. On peut l'initialiser ainsi `int t[] = {1,2,3}` pour le tableau de taille 3 avec `t[0]=1, t[1]=2, t[2]=3`.

Tableaux et fonctions Il est fréquent de vouloir passer un tableau en argument d'une fonction. Il y a une première bonne habitude à prendre : **toujours prévoir un argument supplémentaire pour la longueur du tableau**. Supposons que vous vouliez écrire une fonction `max(t)` qui renvoie le maximum du tableau `t`. Comme il n'y a aucun moyen de connaître la longueur de `t` à partir de `t`, il vous faut nécessairement un argument qui contient la longueur de `t` : `max(t,n)`. En général, lorsque vous appellerez `max` dans votre programme, vous connaîtrez par avance la longueur de `t`. La déclaration d'une telle fonction s'écrit : `int max(int t[], int n)` {...}. Attention, passer un tableau en argument n'est pas exactement équivalent à passer une variable. Voir le premier exercice. On expliquera plus tard pourquoi cela se passe ainsi. Il est plus délicat de faire une fonction qui retourne un tableau. On verra plus tard comment faire. Pour l'instant, vous pouvez vous contenter de modifier le tableau passé en argument (voir Exercice 1).

Matrices On peut aussi définir des tableaux à plusieurs dimensions comme les matrices. Ce seront simplement des tableaux de tableaux. Il y a cependant quelques subtilités ici. Pour intialiser une matrice, mieux vaut indiquer ses dimensions au préalable : `int t[2][3] = {{1,2,3},{4,5,6}}`. On peut éventuellement se passer d'indiquer la première dimension (GCC la devinera en fonction de votre initialisation) : `int t[][3] = {{1,2,3},{4,5,6}}`, mais les autres sont obligatoires. Lorsque vous voulez utiliser des matrices comme argument de vos fonctions, vous devez de même indiquer toutes les dimensions (on expliquera plus tard pourquoi et comment on peut faire ça proprement). Bien entendu, vous ne pouvez pas connaître ces dimensions à l'avance. Une astuce est d'utiliser les autres arguments de la fonction pour initialiser cette valeur, ainsi :

```
void afficher_matrix(int n, int m, int t[n][m]) { ... }
```

Exercice 1. Que va afficher le programme suivant (vous pouvez le récupérer sur ma page : tab.c) ?

```c
#include <stdio.h>

void raz(int t[], int n){
  int i;
  for(i=0;i<n;i++)
    t[i]=0;
}

int main() {
  int t[] = {1,2,3};
  raz(t,3);
```

```
    printf("%d\n",t[0]);
    return 0;
}
```

Exercice 2. On va écrire quelques fonctions basiques sur les tableaux.

1. Écrire une fonction `search` qui cherche un élément dans un tableau d'entiers et renvoie sa position s'il est trouvé et -1 sinon.
2. Écrire une fonction `max` qui renvoie la position du maximum d'un tableau d'entiers.
3. Comme on l'a vu au premier exercice, une fonction peut modifier les valeurs d'un tableau. Écrire une fonction `swap` qui prend en argument un tableau `t` et deux entiers `a`,`b` et qui échange les valeurs des indices `a` et `b` du tableau.
4. Écrire une fonction `void tri_selection(int t[], int n)` qui implémente le tri sélection sur le tableau `t`. On rappelle que le principe du tri sélection est de trouver la plus grande valeur du tableau et de la placer à la fin du tableau. Puis de trouver la deuxième plus grande valeur et la placer à l'avant-dernière position etc.

Exercice 3. On va passer aux matrices. Écrire une fonction qui affiche à l'écran les éléments d'une matrice dans chacun des ordres ci-dessous :

1. De la première à la dernière ligne, chaque ligne étant écrite de la première à la dernière colonne.
2. De la dernière à la première colonne, chaque colonne étant écrite de la première à la dernière ligne.
3. En diagonal, comme dans l'exemple ci-dessous :

$$\begin{pmatrix} 0 & 1 & 3 & 6 \\ 2 & 4 & 7 & 9 \\ 5 & 8 & 10 & 11 \end{pmatrix}$$

Exercice 4. On va jouer à pierre-feuille-ciseau.

1. Écrire un petit programme qui demande en boucle à l'utilisateur d'entrer un nombre : 0 pour pierre, 1 pour ciseau et 2 pour feuille ; 3 pour quitter le programme. L'ordinateur joue en choisissant uniformément une valeur entre 0 et 2. Comptez les points et affichez-les à chaque tour.

On va rajouter une intelligence artificielle. Le principe est le suivant : on se rappelle des deux derniers coups i, j joués par l'utilisateur et on regarde parmi tous les coups joués précédemment lequel a été joué le plus souvent après i et j. Par exemple, supposons que le joueur ait déjà joué 00101211010001. On voit que le joueur a déjà joué trois fois les coups 0 et 1 d'affilé. Deux fois il a ensuite joué 0 et une seule fois il a ensuite joué 2. Le coup suivant le plus probable est alors 0 (pierre). L'ordinateur jouera donc 2 (feuille) pour espérer gagner.

Pour coder cela, on va utiliser un tableau à 3 dimensions `t[3][3][3]` tel que : `t[i][j][k]` vaut le nombre de fois où la suite ijk a été joué par l'utilisateur.

2. Écrire une fonction `coup_suivant` qui prend en argument l'historique `t` des coups joués (comme décrit ci-dessus) et les deux derniers coups joués par l'uilisateur et qui renvoie le coup le plus probable. S'il y a égalité entre plusieurs, on choisira aléatoirement parmi ceux-ci.
3. Implémentez cette intelligence artificielle (essayez de la faire jouer contre des joueurs qui ne connaissent pas la stratégie, vous verrez, ça marche pas mal).
4. Comment pourrait-on faire pareil mais en regardant les 3 derniers coups ? Plus difficile : comment pourrait-on faire pour regarder les k derniers coups où k est un entier que peut choisir l'utilisateur au début du programme ? (**Astuce :** comment représenter une suite de k nombres entre 0 et 2 par un seul entier ? Métastuce : vous savez le faire pour des nombres entre 0 et 1).

Pour le 13 octobre

Finir d'implémenter l'intelligence artificielle du pierre-feuille-ciseau (exercice 4, question 3). (Vous pouvez faire la question 4 mais ça demande un peu plus de réflexion car il faut changer un peu la façon dont on se rappelle des coups précédents).