Projet Algo - 1ère Année
Informations complémentaires

1 Librairie Ncurses

Lorsque l'on utilise la librairie Ncurses, tout ce qui est affiché à l'écran doit l'être par l'intermédiaire de cette librairie. Il n'est pas possible d'avoir simultanément dans un terminal des caractères affichés de manières standard et des caractères affichés en utilisant le mode Ncurses. En d'autres termes, il n'est pas possible de commencer à afficher un menu dans le terminal en utilisant des `printf` puis de décider d'afficher quelque chose au milieu de l'écran en passant en mode Ncurses et en utilisant un `printw`. En effet, le passage au mode Ncurses initialise le terminal, ce qui efface le menu précédemment affiché.

2 Temporisation

Pour ralentir l'évolution du jeu de la vie, il suffit d'insérer une temporisation dans la boucle principale du jeu. Ralentir ou accélérer le jeu revient donc à augmenter ou à diminuer la durée de la temporisation. En `C`, la fonction `void usleep(int micro_seconds)` permet de réaliser une temporisation en précisant en paramètre un nombre de micro-seconde. Pour utiliser cette fonction, il suffit d'ajouter la directive de compilation suivante : `#include <unistd.h>`.

3 Tableaux à deux dimensions et passages de paramètres

Les crochets que l'on utilise pour encadrer les index d'un tableau ne sont, en réalité, qu'une façon de déréférencer un pointeur.

Si `ptr` est un pointeur, et `n` un nombre entier, alors l'expression `ptr[n]` est strictement équivalente à `*(ptr + n)`. Vous pourrez vérifier que ces deux notations sont équivalentes par vous-même.

Ces deux notations permettent de déréférencer un pointeur sur un tableau à une dimension. Interressons-nous maintenant au cas d'un tableau à deux dimensions défini de la manière suivante :

```
int matrice[NB_LIGNES][NB_COLONNES].
```

Dans ce cas, l'expression `matrice[ligne][colonne]` est strictement équivalente à :

```
matrice[(NB_COLONNES * ligne) + colonne]
```

elle-même strictement équivalente à :

```
*(matrice + (NB_COLONNES * ligne) + colonne)
```

Ainsi, le compilateur a besoin du nombre de colonnes d'un tableau à deux dimensions pour pouvoir réinterpréter une notation du type `matrice[ligne][colonne]`. C'est pourquoi cette dimension doit être précisée lors du passage d'un tableau à deux dimensions comme paramètre d'une fonction. En fait, lors de la transmission d'un vrai tableau à deux dimensions, le paramètre effectivement transmis est un pointeur sur le premier *sous-tableau*, et non un pointeur sur le premier élément.

Le passage du tableau `matrice` à une fonction devra donc se faire de la façon suivante :

```
void fonction(int (*matrice)[NB_COLONNES]);
```

Une autre notation possible, bien qu'elle masque la véritable nature du paramètre transmis, est :

```
void fonction(int matrice[][NB_COLONNES]);
```

L'inconvénient évident est que les fonctions ainsi définies ne sont absolument pas générales puisqu'elles en peuvent manipuler qu'un tableau de `NB_COLONNES`. Le prix de la généricité est l'abandon du vrai tableau au profit de l'allocation dynamique comme nous l'avons vu au TP16.