TP4 : Organisation de la mémoire, Makefiles et tableaux

CSH : Initiation au C et au shell

★ **Exercice 1: Organisation mémoire d'un processus**

On souhaite explorer la disposition mémoire d'un processus C, c'est-à-dire les adresses mémoire où sont placés les différents éléments possible. Pour afficher l'adresse à laquelle se trouve une variable `i` quelconque, il faut utiliser : `printf("variable i: %p",&i);`

▷ **Question 1:** Dans un fichier appelé ***exo1.c***, déclarer les variables suivantes :

(1) une chaîne de 10 caractères globale et statique ;
(2) un double local au main ;
(3) un tableau de 10 entiers global ;
(4) un caractère local statique ;
(5) un entier nommé `ex` déclaré en externe et sans valeur initiale.

Dans la fonction main(), également dans ***exo1.c***, faites écrire les adresses des variables. Dans un deuxième fichier appelé exo1bis.c mettre la définition de l'entier `ex` que vous initialiserez à la valeur 20.

▷ **Question 2:** Compilez ce programme avec `gcc -c exo1.c` `gcc -c exo1bis.c` puis réalisez l'édition de liens avec `gcc -o exo1 exo1.o exo1bis.o`. Exécutez `exo1` et concluez sur la place des différentes variables. Quelle est l'effet du modificateur `static` ? Qu'en est-il de `extern` ?

▷ **Question 3:** Modifiez votre programme pour qu'il affiche l'adresse de la fonction main() elle-même. Obtenir l'adresse d'une fonction est similaire à ce qu'on fait pour une variable : `printf("%p\n",&main);` Où est placé le code de cette fonction ?

▷ **Question 4:** Ajoutez une variable de type `char*` dont le contenu est l'adresse d'un bloc alloué dynamiquement avec l'appel `malloc(512);`. Où est placée cette variable ? Où est placé le bloc alloué ?

★ **Exercice 2: Compilation séparée et Makefile.** Récupérez le code fourni sur la page web du cours (`http://www.loria.fr/~quinson/Teaching/CSH/`).

▷ **Question 1:** Vérifiez que les commandes `gcc -c code1.c` et `gcc -c code2.c` ne produisent pas de message d'erreur et que les fichiers ***code1.o*** et ***code2.o*** ont été créés.

▷ **Question 2:** À partir de ces deux fichiers et de ***codage.c***, créez le fichier exécutable ***codage*** par la commande `gcc -o codage codage.c code1.o code2,o`

▷ **Question 3:** Exécutez-le, puis supprimez les fichiers ***code1.o***, ***code2.o*** et ***codage***.

▷ **Question 4:** Écrivez un fichier Makefile permettant de compiler ce projet. On rappelle qu'un tel fichier est composé de différentes règles, chacune suivant la syntaxe suivante (avec un caractère tabulation et non des espaces au début des lignes de commande) :

```
1 <fichier cible> : <liste des dependances>
2         <commande pour fabriquer la cible a partir des dependances>
```

▷ **Question 5:** Vérifiez que votre makefile exécute les bonnes actions dans les cas suivants :

- Aucun binaire n'existe, tout est à reconstruire ;
- Effacement de l'un des `.o` ;
- Effacement du binaire ;
- Modification de l'un des fichiers source ;

★ **Exercice 3: Dice Poker** Ce jeu est une variation du Poker qui utilise des dés au lieu de cartes. Vous allez maintenant implémenter un programme permettant de jouer à ce jeu contre l'ordinateur.

Le jeu consiste en plusieurs manches. Dans chaque manche, l'ordinateur et l'humain lancent 5 dés. Certaines combinaisons de valeur de dés permettent de l'emporter sur d'autres. La liste suivante énumère les combinaisons de la plus "puissante" à la moins "puissante" :

Id	Nom	Description
7	Full	Les cinq dés ont la même valeur
6	Quatre d'un coup	Quatre dés sur cinq ont la même valeur, et le dernier est différent
5	House	Trois dés ont la même valeur, et les deux autre ont la même autre valeur
4	Straight	Cinq valeurs consécutives (même dans le désordre)
3	Trois d'un coup	Trois dés ont la même valeur, tandis que les deux autres sont différents
2	Deux paires	Deux paires de dés ont la même valeur
1	Paire	Deux dés ont la même valeur, et les autres sont différents
0	Rien	Toutes les autres situations

▷ **Question 1:** Observez le code du programme ***dice_poker.c*** fourni. Consultez la page man des fonctions utilisées que vous ne connaissez pas. Compilez-le et exécutez-le plusieurs fois pour essayer.

▷ **Question 2:** Ajoutez une fonction `int manche()` à ce programme, qui crée deux tableaux de 5 entiers tirés aléatoirement, et les affiche sous la forme suivante :

```
Moi : 3 5 4 4 5
Vous: 3 3 6 1 2
```

Vous utiliserez une fonction `void affiche(char *name, int hand[5])` pour afficher chaque ligne.

▷ **Question 3:** Réalisez une fonction `int identifie(int hand[5])` retournant l'identificateur du type de main obtenu (voir la colonne Id ci-dessus).

Pour cela, il faudra calculer la fréquence de chaque valeur de la main (dans un tableau temporaire). Ainsi, pour la main {3,5,4,4,5} la table de fréquence est {0,0,1,2,2,0} (il y a zéro fois la valeur 1, zéro fois la valeur 2, une fois la valeur 3, et ainsi de suite). Le cas du "Straight" se traite à part.

▷ **Question 4:** Modifiez la fonction `affiche()` pour qu'elle affiche le type de main obtenu par chacun.

▷ **Question 5:** Modifiez `manche()` pour qu'elle affiche le gagnant de la manche et renvoi -1 si l'ordinateur gagne, 0 sur match nul et 1 si l'humain gagne.

```
Moi : 3 5 4 4 5 (deux paires)
Vous: 3 3 6 1 2 (paire)
JE GAGNE !
```

▷ **Question 6:** Modifiez la fonction principale pour jouer 5 manches, compter les points acquis à chaque manche et afficher le résultat final.

▷ **Question 7:** Modifiez votre programme pour autoriser les jeux avec un nombre arbitraire de dés, spécifié en ligne de commande. La fonction d'évaluation doit naturellement être modifiée (on peut laisser les Straight de coté dans un premier temps). Il faut pour cela consulter non seulement la table des fréquences de valeur, mais également la table des fréquences de la table des fréquences.

Par exemple, {4,6,2,4,3,6,6,6,3} a la table de fréquence suivante : {0,1,2,2,0,5} et la table de fréquence des fréquences suivante : {2,1,2,0,0,1,0,0,0,0}. Il y a deux valeurs qui ne sont pas représentées (le 1 et le 5), une valeur unique (le 2), deux paires (3 et 4), et un groupe de cinq valeurs identiques (les 6).

Valeur	1	2	3	4	5	6
Fréquence	0	1	2	2	0	5

(a) Table des fréquences

Fréquence	0	1	2	3	4	5	6	7	8	9
Nombre de valeurs ayant cette fréquence	2	1	2	0	0	1	0	0	0	0

(b) Table des fréquences de fréquence

Un *full* d'une main de 9 dés a la table de fréquence des fréquences suivantes : {5,0,0,0,0,0,0,0,0,1} (5 valeurs ne sont pas représentées du tout tandis qu'une valeur est représentée neuf fois).

On peut considérer cette table comme les coefficients d'un polynôme, dont il convient d'ignorer les deux premiers termes. La main du premier exemple correspond ainsi à $2 \times X^2 + X^5$ (on ne retient que la présence de deux paires et d'un groupe de cinq valeurs identiques) tandis que le full est X^9.

Nous voila prêts à modifier la fonction `affiche()` pour donner une forme textuelle de la combinaison obtenue. On utilisera les abréviations grecques ou latines pour les valeurs supérieures à 4. La main précédente est "2 paires, 1 penta" tandis que $X^4 + X^6$ est "1 carré, 1 hexa" et le full "1 nona".

Pour la valeur de la combinaison, on évaluera le polynôme pour $X = 2$ (attention, 4 paires sont plus fortes qu'un triple) ou $X = 10$ (attention aux débordements de capacité, on peut obtenir des valeurs de main négatives si on s'y prend mal).

▷ **Question 8:** Faites en sorte de permettre à l'utilisateur de demander à relancer des dés (comme on redemande des cartes au Poker) en indiquant 5 booléens après chaque lancé. Dans l'exemple suivant, les caractères soulignés sont tapés par l'utilisateur qui demande à relancer les 3 derniers dés.

```
Moi : 3 5 4 4 5 (deux paires)
Vous: 3 3 6 1 2 (paire)
Que voulez vous faire? 0 0 1 1 1
Vous: 3 3 3 4 4 (house)
VOUS GAGNEZ !
```

▷ **Question 9:** Limitez les cas d'égalité en faisant en sorte qu'entre deux Fulls, le gagnant soit celui dont les dés marquent la plus grande valeur. Faites de même pour les combinaisons "N d'un coup". Pour départager deux "House", on regarde d'abord la valeur des triplettes avant de regarder la valeur des paires si les triplettes sont équivalentes. On départage les "Deux paires" de façon similaire. Un "Six Straight" (2-6) est plus fort qu'un "Five Straight" (1-5).

▷ **Question 10:** *(mini projet)* Dotez l'ordinateur d'une stratégie pour lui aussi relancer certain de ses dés. Cette décision sera prise par le biais d'une fonction `int *rejoue(int *hand, int hand_size)`.

▷ **Question 11:** Implémentez plusieurs fonctions de ce type (tout rejouer, rien rejouer, rejouer un dé une fois sur deux, ainsi que des "vraies" stratégies), et faites jouer les AI les unes contre les autres sur un grand nombre de parties pour déterminer la meilleure approche. Faites jouer votre AI contre celle de votre voisin.