Cahier de TP n°4

TP 1 : Rôle des fonctions

Gilbert strikes back

C'est en rentrant bien tardivement d'une soirée festive que Gilbert a eu un éclair de génie et a écrit les deux fonctions suivantes :

```
char *VdbG4SF_B_35tB(char *F3e_4GiK_7MP0)
{return new char[strlen(F3e_4GiK_7MP0)+1];}

char *fSJFsh(char *a43P_kN6j_m1)
{
char *csAGv_r, *jerVAa, *rezbh;
for(jerVAa=csAGv_r=VdbG4SF_B_35tB(rezbh=a43P_kN6j_m1);*csAGv_r++=*a4
3P_kN6j_m1++;);return jerVAa;}
```

Elles sont correctes au niveau algorithmique, compilent sans problème et donnent bien le résultat voulu, mais il reste à savoir ce qu'elles peuvent bien réaliser…

a) Ecrivez un fichier .c contenant les définitions des fonctions proposées;

b) Ecrivez un fichier .h contenant les prototypes de ces deux fonctions;

c) Ecrivez un programme principal dans un troisième fichier .c. Ce programme devrait vous aider à trouver ce que font ces fonctions. Il faut donc essayer de voir quelle fonction appeler et quel(s) argument(s) lui fournir. (Vous ne devriez normalement pas avoir besoin de décortiquer l'algorithme réalisé par les fonctions, ce qui, finalement, n'est peut être pas plus mal…).

Test sur les variables locales

Pour vérifier que les variables de toutes les fonctions sont bien des variables locales, le plus simple est de vérifier qu'elles sont bien situées à des adresses différentes. Pour cela, écrivez un programme principal comportant 2 variables que vous initialiserez et dont vous afficherez les adresses. Ecrivez une fonction avec un paramètre portant le même nom que l'une des variables du programme principal, et une variable locale portant le même nom que l'autre variable du programme principal. La fonction doit afficher les adresses de son paramètre et de sa variable locale. Que constatez-vous à l'exécution ?

Changez le nom du paramètre et de la variable locale de la fonction. Exécutez de nouveau le programme; Que constatez-vous ?

Tableaux et paramètres

Enfin, écrivez un programme principal dans lequel est défini un tableau dynamique d'entiers (défini par un pointeur à une dimension). Faites une allocation dynamique de 20 entiers et rangez dans le tableau les valeurs de 1 à 20. Affichez les valeurs stockées dans le tableau, l'adresse de ces valeurs et l'adresse du pointeur utilisé. Ecrivez ensuite une fonction qui ajoute 1 à toutes les valeurs stockées dans le tableau. Cette fonction aura donc comme paramètre un pointeur vers des entiers. Affichez dans la fonction : la valeur de ce pointeur, et l'adresse de ce pointeur : que constatez-vous ?

Retourner l'adresse d'une variable locale

Ecrivez une fonction qui renvoie l'adresse de l'une de ses variables (locale donc). Affichez la valeur récupérée dans le programme principal et utilisez cette valeur pour accéder à la mémoire. Que se passe-t-il ? Qu'en pensez-vous ?

TP 2 : Fonctions utilitaires

Le but de ce thème est d'écrire quelques fonctions utilitaires qui seront ensuite regroupées dans un module.

Saisie sécurisée : écrire une fonction qui réalise la saisie sécurisée d'un réel : il doit être compris entre 2 valeurs a et b (a : borne inférieure; b : borne supérieure). La fonction renverra donc une valeur réelle qui sera comprise (au sens large) entre a et b.

Paramètres de main()

La définition de la fonction main() peut s'écrire sous la forme :

```
int main(int argc, char *argv[])
{
// instructions du programme principal
}
```

Puisque `main()` est une fonction comme les autres, cela signifie entre autres que `argc` et `argv` sont deux paramètres.

Les valeurs contenues dans `argc` et `argv` sont les valeurs que l'on peut fournir en entrée au programme lorsqu'on l'exécute à partir d'une fenêtre de commandes DOS (la ligne alors entrée dans la fenêtre DOS s'appelle "ligne de commande"). `argc` contient le nombre de mots écrits sur la ligne de commande, `argv` contient ces mots.

`argc` est donc de type entier (int) et `argv` est un tableau contenant des pointeurs vers des caractères : c'est un tableau de caractères à deux dimensions.

Valeurs de `argc` et `argv`

Soit un programme nommé `toto.exe`.

Dans une fenêtre DOS, vous pouvez l'exécuter en donnant simplement son nom :

```
Z:\>toto
```

Dans ce cas, `argc` vaudra 1, car il n'y a qu'un mot sur la ligne de commande

`argv[0]` contiendra ce mot.

Vous pouvez donner des valeurs en entrée au programme via la ligne de commande : exemple :

```
Z:\>toto a b c d
```

a) Exécution à partir de Dev-c++ ou Code::Blocks

Ecrivez un programme principal ayant en paramètre `argc` et `argv`. Ce programme devra juste afficher les valeurs de `argc` et `argv[0]` (qui est une chaîne de caractères). Exécutez-le à partir de Dev-C++. Que constatez-vous ?

b) Exécution à partir du DOS

Ouvrez une fenêtre DOS et exécutez votre programme en lui donnant des entrées (qui ne sont bien sur pour l'instant pas traitées); Vérifiez que `argc` a bien la valeur prévue. Améliorez ce programme pour qu'il affiche maintenant tous les mots donnés sur la ligne de commande.

c) exploitation de la ligne

Améliorez votre programme pour qu'il accepte en entrée deux entiers donnés sur la ligne de commande et qu'il affiche le résultat de leur somme. Exemple : si votre programme s'appelle `toto.exe`, on devra pouvoir, à partir du DOS, obtenir :

```
Z:\>toto 12 54
Le résultat est : 66
```

c) information sur le programme

Améliorez votre programme pour qu'il affiche un message expliquant la manière de s'en servir si :

- on ne marque que son nom : indiquer qu'il n'y a pas assez d'arguments

ou

- le mot /help se trouve sur la ligne de commande : mode d'emploi du programme

exemple du résultat attendu :

```
z:\>toto
pas assez d'arguments.
```

ou

```
Z:\>toto /help
usage : toto entier1 entier 2
réalise l'addition des deux entiers
```

TP 3 : synthèse numéro 1 : le jeu du mastermind

Le sujet de ce thème est la réalisation d'un jeu relativement simple en procédant petit à petit par l'écriture de fonctions. Le sujet de ce thème consistera donc à réaliser un certain nombre de fonctions que l'on vous demandera de tester individuellement les unes après les autres. Il vous restera à réaliser le programme principal qui appellera ces fonctions dans le bon ordre pour réaliser le jeu.

Pour cela, nous allons procéder en deux étapes : réaliser un premier module de jeu puis un module d'interface graphique sommaire. Le jeu pourra fonctionner en mode texte simple. Pour profiter au maximum de ce TP, il est impératif que vous suiviez pas à pas la réalisation demandée.

Réalisation du module jeu_master : jeu en mode texte

Une combinaison du master mind est une suite de 5 pions de couleur, sachant qu'il existe 8 couleurs différentes : rouge, bleu, vert , jaune, marron, blanc, noir, rose, et qu'une combinaison peut comporter plusieurs pions de la même couleur. Le but est de retrouver, en un nombre minimal d'essais, une combinaison choisie au hasard par l'ordinateur.

A chaque essai, vous proposez une combinaison de 5 couleurs, et l'ordinateur vous indique le nombre de pions de la bonne couleur et le nombre de pions bien placés par rapport à la combinaison à découvrir.

On utilise, dans un premier temps, des chiffres pour remplacer les couleurs: une combinaison sera donc un ensemble de 5 chiffres, chaque chiffre étant comprise entre 1 et 8. On choisit de stocker une combinaison dans un tableau de 5 entiers.

Etape 1 : création des fichiers jeu_master.h et jeu_master.c

Créez ces deux fichiers dans votre projet et ajoutez, dans le fichier jeu_master.h, la ligne suivante : `#define TAI 5`.

Cela vous permettra d'utiliser TAI comme constante pour la taille utile des tableaux dans les fonctions. Ainsi, vous pourrez écrire :

```
for(cpt = 0; cpt < TAI; cpt++)
{
      instructions de la boucle …
}
```

Fonction numéro 1 : affichage d'une combinaison

Ecrire une fonction affiche_combi qui affiche tous les chiffres d'une combinaison séparés par un '.'. exemple : si la combinaison est 54642, on doit voir à l'écran :

```
5 . 4 . 6 . 4 . 2
```

Fonction numéro 2 : Saisie d'une combinaison

Voici le prototype de la fonction à écrire :

```
long saisi_comb(long tablo[]);
```

Ecrire la définition de la fonction `saisi_comb`, qui propose à l'utilisateur de saisir une combinaison sous la forme d'un nombre entier : la fonction devra retrouver les chiffres individuels dans le nombre et les ranger dans le tableau. Si le nombre ne comporte pas exactement 5 chiffres ou si l'un des chiffres est 0 ou 9, alors la fonction retournera la valeur 0. Si la combinaison est valide, la fonction retournera la valeur 1.

Fonction numéro 3 : génération aléatoire d'une combinaison

On vous fournit le prototype de la fonction qui génère aléatoirement une combinaison de 5 chiffres :

```
void genere_combi(long tablo[]);
```

Ecrire la définition de cette fonction, vous utiliserez la fonction `rand()`, n'oubliez pas que vous devrez utiliser la fonction `srand()` dans le programme principal : Inclure les fichiers stdlib.h et time.h au début de votre programme (`#include <stdlib.h>` ainsi que `#include <time.h>`), puis en première instruction de votre programme principal, utiliser la commande :

```
srand((unsigned)time(NULL));
```

Cette commande sert à initialiser le générateur de nombres aléatoires, cela permet d'obtenir des valeurs aléatoires différentes entre deux exécutions du programme.

Pour effectuer un tirage au hasard entre 1 et 8, on utilise la commande `rand()`.

```
long value; // variable stockant le chiffre au hasard de la combinaison
value = 1 + (rand() % 8); // % : modulo, donc on obtient une valeur entre 0
                          // et 7, on y ajoute 1 : entre 1 et 8
```

fonction numéro 4 : nombre de valeurs bien placées

prototype : `long bien_places(long combi[], long propos[]);`

Cette fonction prend en entrée : le tableau comprenant la combinaison à découvrir, le tableau comprenant la proposition faire par l'utilisateur. Elle retourne le nombre de chiffres bien placés dans la combinaison

Exemple : la combinaison à découvrir est : 5 **4** 8 7 2

La proposition est: 8 **4** 1 2 3

Seul le 4 est bien placé, donc la fonction retournerait 1 dans ce cas précis.

Ecrivez la définition de cette fonction.

Fonction numéro 5 : nombre de valeurs présentes mais mal placées

prototype : `long presentes(long combi[], long propos[]);`

Cette fonction prend en entrée : le tableau comprenant la combinaison à découvrir, le tableau comprenant la proposition faire par l'utilisateur. Elle retourne le nombre de chiffres présents mais mal placés dans la combinaison.

Exemple : la combinaison à découvrir est : 5 4 **8** 7 **2**

La proposition est: **8** 4 1 **2** 3

Le 2 et le 8 sont présents dans la combinaison à découvrir, mais sont mal placés. Attention, on ne compte pas les chiffres bien placés ici ! La fonction retournerait donc 2 dans le cas présent.

Ecrivez la définition de cette fonction.

Fonction numéro 6 : résultat de l'essai

Prototype : `void resultat(long, long);`

Le premier paramètre est le nombre de chiffres présents mais mal placés, le second est le nombre de chiffres bien placés. La fonction doit afficher :

```
X présents - Y bien placés.
```

(Où X et Y sont les valeurs des deux paramètres).

Ecrivez la définition de cette fonction.

Dernière étape : écriture du programme principal

Ecrivez maintenant le programme principal qui appelle dans le bon ordre les différentes fonctions écrites. Le programme principal doit afficher le nombre de tentatives faites pour découvrir la combinaison cachée. Le nombre maximum de tentatives est de 20, si l'utilisateur n'a pas trouvé à la 20ème tentative, il a perdu. Dans le programme principal, on créera, pour les combinaisons proposées par l'utilisateur, un tableau `propositions` à deux dimensions de 20 lignes et 5 colonnes pour stocker toutes les propositions successives du joueur.

Pour ceux qui ont fini la première partie : intégration du module display pour le jeu en mode graphique

Nous allons une fois de plus utiliser ce bon vieux module conio pour intégrer une interface graphique minimale : cette interface ne changera rien aux fonctions utilisées pour le jeu, sauf peut être les fonctions d'affichage et de saisie.

Code couleur utilisé :

chiffre	Couleur
1	WHITE
2	RED
3	GREEN
4	BLUE
5	YELLOW
6	RED
7	BROWN
8	MAGENTA

Le module DISPLAY réunissant les différentes fonctions nécessaires est en fait téléchargeable à l'adresse :

http://perso.efrei.fr/~flasque/promo2010/Algo_et_C/TP/mastermind/

Il vous faut donc l'intégrer à votre programme principal pour obtenir un petit jeu de mastermind en mode graphique.

TP 4 : synthèse numéro 2 : Home Sweet Home.

Deux frères jumeaux (nommons les Gil & Bert), décident de faire une course dans Villejuif pour rentrer chez eux après avoir consommé moult jus d'orange et autres boissons. Il fait très noir et l'éclairage public est en panne…C'est donc à l'aveuglette que les deux frères vont tenter de se rendre à leur domicile. La carte de la ville où ils habitent est téléchargeable à http://perso.efrei.fr/~flasque/promo2010.Algo_et_C/TP/homesweethome/, le fichier s'appelle : villejuif.map, c'est un tableau à 2 dimensions contenant des entiers où :

- 0 symbolise une route
- 1 symbolise un bâtiment quelconque
- 2 symbolise un poste de police
- 3 symbolise la maison de Gil & Bert

Gil & Bert peuvent se déplacent d'une case par tour, Gil se déplaçant en premier, car c'est l'aîné (il est né 2 minutes avant Bert). A chaque tour, le sens du déplacement (haut, bas, gauche, ou droite) est tiré au hasard. Si la case de destination du mouvement est une route, pas de problème, le déplacement se fait. Si la case de destination est un bâtiment quelconque, le déplacement ne peut pas se faire. S'il s'agit d'un poste de police, le frère y entre et y reste, il ne peut plus bouger. S'il s'agit de leur maison, celui qui y arrive en premier est déclaré gagnant et a gagné une bonne nuit de sommeil. Si les deux frères échouent à un poste de police, la course se termine par un match nul. Si la case de destination est occupée (par l'autre frère), alors on choisit un autre déplacement au hasard pour ce tour. Si la case de destination est située hors du tableau, le frère continue sa promenade dans la banlieue et revient dans Villejuif 2 tours plus tard par l'endroit où il est sorti.

Gil part des coordonnées (16,7) et Bert des coordonnées (16,8), en commençant la numérotation des lignes et des colonnes à 0.

Ecrire les fonctions suivantes :

AfficheCarte qui représente les routes en noir, les bâtiments quelconques en gris, un poste de police par un P blanc sur fond bleu et la maison de Gil & Bert par un M blanc sur fond vert.

PlaceFrères qui affiche la position de Gil (G vert sur fond noir) et de Bert (B jaune sur fond noir). Attention, si l'un des deux est sur un poste de police, il n'apparaît plus à l'écran.

AuHasard qui retourne une direction tirée au hasard entre 0 et 3 (code pour les directions haut/bas/gauche/droite : 0 : haut, 1 : bas, 2 : gauche, 3 : droite)

MajFreres qui met à jour la position de Gil et de Bert en fonction du déplacement.

Intégrez ces fonctions au programme principal qui doit permettre de visualiser la "course" ainsi que le résultat.

Amélioration : une patrouille de police (X blanc sur fond bleu) parcourt les rues en se basant sur un tableau de directions contenu dans le fichier patrouille.map. Si la patrouille croise Gil ou Bert (si elle se situe à une case voisine), le frère est emmené directement au poste de police et la patrouille continue. La patrouille part des coordonnées (11,13).