Programmation C

PORTFOLIO S1

Emilie Marti – INFO1
Chargé de TP : FRANCIS Nadime

Vue d'ensemble

Tous les TP sont gérés sous git. Le lien d'accès est le suivant : https://github.com/hyliancloud/INFO_C.git . Tout est push sur la branche master.

Ma démarche pour apprendre à résoudre les TP est généralement la suivante :

- Si je connais l'algorithme, je tente ma méthode en premier. Si elle marche, je la teste sur plusieurs cas pour être sure qu'elle marche toujours (ou pour déterminer ses limites).
- Si je ne connais pas l'algorithme, je fais des recherches sur internet, notamment openclassroom qui a de bons tutoriels, puis je mets en pratique en C. Je consulte aussi d'autres sites comme stackoverflow et autres tutoriels online.
- Les erreurs de compilation sont corrigées en prenant connaissance du type d'erreur (si je ne le reconnais pas je cherche) et en corrigeant.
- Les erreurs sont réglées dans l'ordre, en commençant par la première : souvent c'est potentiellement l'erreur qui en induit d'autres.
- Si j'ai des erreurs de comportement, j'essaie de debugger avec des printf qui témoignent du devenir de la fonction à chaque pas.
- Si je suis bloquée, je demande au chargé de TP ou à un collègue.
- J'essaie d'implémenter les conseils d'optimisation du chargé de TP, tout en gardant les versions antérieures pour comparer.
- Certains algos (notamment les fonctions recursives qui traitent les tableaux) ont été mieux appréhendés en cours d'Algorithme I et corrigés pour certains TP plus tard.

Les TP ont été faits de façon individuelle.

J'essaie de garder mon code propre et lisible ; je pars du principe que mon code va être relu, soit par quelqu'un d'autre, soi par moi-même, et doit être facilement compris. J'évite de commenter mon code, car cela l'allonge et le rend lourd à relire. Il faut donc qu'il soit clair par lui-même.

Je me suis imposée des règles :

- Je code en anglais.
- Le nom des fonctions et des variables doivent être auto-explicatives ; rien que de les voir on doit comprendre ce qu'elles vont faire / à quoi elles vont servir.
- Les noms ont des formes définies :
    - Variables → nomVariable ;
    - Variables globales (utilisées le moins possible) → _nomVariable ;
    - Valeurs dans #define → NOM_VALEUR ;
    - Fonctions → nom_fonction().
- Les accolades vont à la ligne.
- Le code doit être factorisé.

- Un programme doit être bien organisé dans ses dossiers ; dans le dossier principal il n'y a que les .c, les autres types de fichiers (.h, images, txt …) vont dans leur dossier attribué.

J'ai défini ces règles petit à petit courant le semestre, donc peut-être qu'elles ne sont pas partout suivies, surtout au début (j'ai quand-même essayé de revoir mes codes). J'essaie de factoriser au plus mais c'est peut-être pas tout à fait le cas encore.

Pour le premier semestre, j'ai plus ou moins suivi le fil rouge. Les TP réalisés sont les suivants :

- TP3 (exos)
- TP4 (pile d'entiers)
- TP5 (tableaux fin -1)
- TP6 (backtracking)
- TP7 (sudoku graphique)
- TP8 (syracuse)
- TP9 (exos malloc)

À partir du TP6, j'ai utilisé gdb pour le debug ; lorsque les printf ne m'aidaient pas à comprendre pourquoi une fonction ne fonctionnait pas, j'ai appris à utiliser gdb et ça a été utile. Pour l'utiliser, il faut rajouter aux CFLAGS du makefile -g et on exécute avec gdb :

```
>> gdb ./test
>> r (run the program)
```

gdb s'arrête au moment où le programme plante et donne quelques indications sur la raison pourquoi il plante (généralement des seg-fault dans mon cas).

J'ai eu deux principaux problèmes d'organisations lors des TP, je vais tenter de ne pas me laisser avoir au deuxième semestre :

- À partir du TP5, j'ai pas fait les comptes rendus de suite ; j'ai juste continué à faire les TP suivants. Ça m'a obligé à relire mes codes pour me rappeler de ce que j'avais fait pour rédiger les comptes rendus.
- Quand j'ai eu des difficultés avec la fin des derniers TP, j'ai décidé de finir plus tard et tenté de faire les TP suivants. Mauvaise tactique.

TP3 – Quelques exercices

Emilie Marti

Ce TP contient des exercices différents et indépendants les uns des autres ; on aurait pu les faire dans des fichiers indépendants, cependant j'ai choisi de les faire sous forme de modules qui vont être appelés par le même main.c, en rajoutant une forme modulaire à mon programme. Aussi, le premier exercice étant l'implantation de la fonction puissance, elle a pu être réutilisée plus tard lors des conversions de base, ce qui montre qu'elle marche plutôt bien.

Au niveau de la puissance, j'ai implémenté une version optimisée de la fonction de puissance récursive.

J'essaie de factoriser mon code ; c'est plus facile à comprendre et à debugger si besoin (chaque fonctionnalité a sa propre fonction, on évite de tout faire dans une fonction).

De façon générale, j'ai pris connaissance de certains algorithmes (la puissance en récursif, les tris dichotomique et à bulles et la conversion des bases) et à les mettre en place en C, ainsi qu'à trouver une manière d'optimiser une fonction récursive (pour qu'elle trouve un résultat « intelligemment » en ne passant pas par tous les cas), dans le cas des puissances.

Les compétences suivantes étaient à valider :

- I/O

Compétence mise en place à travers les entrées sur la sortie standard (avec des scanf), contrôlés avec des conditions pour ne pas prendre en compte n'importe quel input donné par l'utilisateur. Le flux de sortie utilisé est principalement stdout.

- Type

Les types utilisés sont les entiers et les caractères, soit sous sa forme, soit sous la forme de pointeurs.

- Programme

Les fonctions sont utilisées et testées dans le main.

- Compilation

Un makefile est créé pour compiler facilement avec la simple commande « make ». Les modules sont convertis en fichiers objets .o puis utilisés pour créer l'exécutable de sortie (qui fonctionne sur linux).

- Récursivité

La récursivité est utilisée sur la puissance et sur la recherche dichotomique. Pour la puissance, deux fonctions récursives sont proposées ; l'une est plus optimisée

par rapport à l'autre car elle va faire moins de tours pour calculer la puissance que la première.

- Tableaux

Les tableaux 1D sont utilisés pour les chaînes de caractères qu'il faut traiter et convertir puis pour les tableaux d'entiers qu'il va falloir trier lors des deux derniers exercices.

Les compétences supplémentaires mises en œuvre sont les suivantes :

- Bibliothèque

Les bibliothèques utilisées sont : stdio, stdlib (pour les fonctions standard), string pour les chaînes de caractères, math pour des fonctions mathématiques puis time pour avoir le nombre de secondes depuis le début de l'existence de l'informatique (pour changer la seed lorsqu'on veut avoir des nombres random).

- Module

Pour rendre le rendu plus propre, j'ai décidé de faire un seul main qui testera les fonctions de tous les exercices, mais celles-ci seront placées dans un fichier source différent, qui sera rappelé sur le main via son fichier header (.h).

Les .h ont toujours cette forme :

```
#ifndef __HEADER__H
#define __HEADER__H

void prototype_fonction() ;

#endif
```

Ceci évite au compilateur de recompiler deux fois le même .c, ce qui lui ferait penser qu'il y a plusieurs définitions de la même fonction et la compilation n'aboutirais pas.

TP4 – Pile d’entiers

Emilie Marti

Sur ce TP, on apprend à utiliser la pile pour stocker des valeurs en mémoire et les récupérer dans l’ordre. Il s’agira d’un programme modulaire dont le header (.h) est fourni ; les fonctions sont déclarées, et on doit les coder en conservant le prototype. En gros on codera la pile qui va être ensuite utilisée par une calculatrice graphique (codée avec MLV) qui va utiliser ses fonctions. Finalement, certaines fonctionnalités vont être rajoutées à la calculatrice.

Pour calculer les modulo, il faut taper « m » ; en effet, le symbole du module, % , n’est pas géré par la librairie graphique MLV et taper ce symbole au clavier fait directement planter le programme.

Les compétences à acquérir pour ce TP sont les suivantes :

- I/O

Les input des utilisateurs sont gérés via les fonctions de la librairie MLV.

- Type

Les types mis en œuvre sont les entiers, les flottants et les caractères.

- Programme

Les fonctions sont codées dans son modules et utilisées dans un main qui va aussi gérer les fonctions graphiques pour que le programme marche sur la calculatrice graphique.

- Module

Le programme est sous la forme modulaire ; toutes les fonctions sont dans des fichiers spécifiques qui dépendent de sa fonctionnalité.

- Compilation

Un makefile est mis en place pour une compilation simple ; un seul « make » est à écrire sur le terminal et le makefile créera les fichiers objets qui seront utilisés pour créer l’exécutable de sortie. Les fichiers objets sont ensuite éliminés car non utilisés pour plus de clarté dans le dossier.

- Bibliothèque

Pour ce TP, la bibliothèque graphique MLV a été utilisée ; les fonctions principales étant données, il a fallu les adapter pour mettre en œuvre les fonctionnalités. Les fonctions de la pile peuvent être associées à une mini-bibliothèque utilisée sur la calculatrice.

TP5 – Tableaux fin -1

Emilie Marti

Sur ce TP, on apprend à manipuler des tableaux d'entiers, triés ou pas, qui ont la particularité de se terminer par -1, pour simuler le caractère de fin de chaîne '\ 0' ; ceci permet de ne pas devoir passer en paramètre des fonctions la taille du tableau (elle peut être recalculée à chaque fois).

- I/O

L'utilisateur peut décider la taille des tableaux qui vont être utilisés avec les fonctions créées dans ce programme.

- Type

Les types utilisés sont des entiers et des tableaux d'entiers (pointeur sur la première case).

- Programme

Un fichier main.c appelle les fonctions et ne contient que les tests.

- Compilation

Pour compiler, il suffit d'écrire « make » dans le terminal et un appel à echo dit à l'utilisateur le nom de l'exécutable (./TP5 dans ce cas).

- Récursivité

La récursivité est utilisée dans la fonction merge_sort. En effet, merge sort va s'appeler elle-même tant que les sous-tableaux sont de taille > 1. Les autres fonctions sont itératives par défaut ; c'est plus performant et la récursivité risque de faire exploser la pile bêtement alors qu'en itératif non.

- Tableaux

Toutes les fonctions sont sur des tableaux d'entiers avec caractère de fin.

- Pointeurs

Les tableaux d'entiers sont souvent utilisés sous la forme d'un pointeur.

Un fichier supplémentaire, exo0.c, contient des fonctions non spécifiées dans la consignes mais utiles au bon fonctionnement du programme. Les compétences mises en œuvre sont l'allocation de mémoire et la modulation du programme, les fonctions étant séparées dans des fichiers selon la consigne.

TP6 – Backtracking
Emilie Marti

Ce TP m'a appris à résoudre un jeu du type sudoku avec de la brute force (on teste les valeurs jusqu'à bloquer, puis on revient à l'arrière pour tester de nouvelles valeurs ; ansi de suite jusqu'à résolution). Ce n'est pas une méthode optimisée mais elle marche très bien pour un simple sudoku de 9x9 cases.

Avec la première partie du TP j'ai découvert la technique du backtracking avec un tableau à une dimension et j'ai du l'appliquer sur un tableau à deux dimensions au sudoku (deuxième partie du TP). La partie la plus difficile à été déjà de comprendre ce que c'était le backtracking, puis l'appliquer. Ensuite, il a fallu faire gaffe au tableau à deux dimensions (ne pas inverser lignes et colonnes surtout, ce qui peut tout faire foirer bêtement).

J'ai pas su trouver toutes les possibilités de résolution d'un sudoku à plusieurs solutions … La fonction retourne lorsqu'elle a trouvé la première solution, donc je devrait peut-être changer cela facilement (peut-être pour le deuxième semestre).

- I/O

J'ai affiché le tableau du sudoku sur la sortie standard et ça rend plutot bien ; ça a été plutôt facile surtout parce que les nombres sont de 1 à 9 (ou rien), du coup ils occupaient le même espace.

- Type

Les types utilisés sont principalement des int.

- Programmation

- Module

Le programme est divisé en modules ; le main est tout seul dans son fichier, et les fonctions classées selon ce qu'elles gèrent.

- Compilation

Le makefile va compiler le programe avec un simple make et va donner le nom de l'exécutable sur la sortie standard.

- Récursivité

La technique du backtracking (brute force) est codée en récursive.

- Tableaux

On travaille sur des tableaux 1D pour la première partie, puis 2D pour la deuxième partie.

- Pointeurs

Les tableaux sont sous la forme d'un pointeur qui pointe sur la première case d'un tableau.

- Fichiers

Le programme va récupérer les modèles de sudoku non-résolus dans des .txt. À partir de là un tableau à 2 dimension est rempli.

Pour retrouver l'emplacement de la case, j'utilise une seule variable position et je retrouve les coordonnées, par exemple pour un tableau de 9x9, comme suit :

row = position/9 ;
column = position%9 ;

Ceci me permet de trouver une condition d'arrêt simplement ; en effet, lorsque la position = 9x9 = 81, je viens de sortir du tableau (qui va de 0 à 80) et cela veut dire que le sudoku a été complêtement rempli et je peux terminer ma fonction.
Cela me permet aussi de n'avoir qu'une variable à passer en paramètre lorsque je veux appeler une fonction pour une case donnée.

TP7 – Sudoku graphique
Emilie Marti

Peut-être le TP le plus fun à faire mais la gestion des événements (tours ou devenir du jeu) a été vraiment difficile à prendre en main pour moi.

Pour le déroulé du jeu on travaillera avec deux tableaux 9x9 et un tableau 3x3 : le premier tableau est la base du sudoku à résoudre (à tenir en compte car l'utilisateur ne doit pas pouvoir modifier une valeur de base), le deuxième est le tableau qui va s'afficher et se mettre à jour selon les input du joueur et le troisième est juste un clavier numérique qui va permettre au joueur de choisir le numéro à mettre dans la case.

Le main a le moins de fonctions possibles ; on initialise notre terrain de jeu, on appelle la fonction qui contient la boucle while du jeu puis une fonction qui va libérer la mémoire allouée.

Les fonctions sont distribuées par fonctionnalités dans des fichiers différents ; on doit pouvoir trouver facilement toutes les fonctions.

- I/O

Ici doivent êtres gérés les input du joueur, en l'occurence les clicks sur le tableau du jeu, avec les fonctions de la libMlv ; le click est récupéré au pixel près (toujours en fonction de la taille de la fenêtre, jamais en dur ou si une valeur change ce ne s'adaptera pas).

- Type

Les types utilisés sont principalement des int et des pointeurs sur int (pointeur sur la première case du tableau).

- Programme

Il s'agit d'un jeu, un programme entier qui doit être organisé en modules pour la bonne lisibilité et pouvoir facilement trouver les fonctions en cas de debug.

- Module

Les modules sont en fonction des fonctionnalités ; en effet, il y a plusieurs fichiers tels que le board qui affiche le terrain de jeu et gère les fonctions graphiques, le inputManager qui gère les entrées du joueur, le sudoku qui sert à solver le sudoku (fait par backtracking au TP6).

-

- Compilation

Un makefile est prévu pour compiler avec une simple commande « make ». Pour lancer le jeu il faut appeler en paramètres une des bases de sudoku comme suit :

>> ./sudoku data/grid1.txt

- Tableaux

Le programme se base sur des tableaux d'entiers à deux dimensions principalement.

- Structures

Une structure board, qui est principalement un typedef d'un tableau d'entiers 2D est utilisée sur tout le programme.

- Pointeurs

Vu qu'on travaille sur les tableaux, on travaille avec des pointeurs aussi.

- Fichiers

Le programme fait appel à des fichiers .txt pour avoir une base de sudoku.

- Bibliothèque

On utilise la libMlv pour ce TP.

Par manque de temps, **ce TP n'est pas terminé**. J'ai bloqué au niveau de la gestion des tours et fait l'erreur de passer à un autre TP avant de le terminer (j'ai fait les TP 8 et 9).

Ce sera le premier TP que je terminerais dès le S2.

TP8 – Syracuse
Emilie Marti

Ce TP avait l'air plus compliqué que ce qu'il était vraiment. Cependant il était intéressant à faire, notamment au niveau de l'optimisation de la fonction qui calcule la hauteur de la suite en utilisant un tableau pour mettre en cache les valeurs déjà calculées.

- Type

Deux nouveaux types ont été utilisés : long int et long unsigned int. En demandant à l'utilisateur de rentrer un nombre sur l'entrée standard, je le range dans un long int, que je caste en long unsigned int pour la première partie du TP. J'utilise ensuite un long int car le tableau du cache se remplie avec des -1 (pour simuler une case non remplie).

- Programmation

Le main appelle les fonctions et les teste.

- Compilation

Le makefile compile le programme avec make et donne le nom de l'exécutable.

- Récursivité

Les fonctions afficant la suite de Syracuse et donnant la hauteur sont récursives. La deuxième partie du TP m'a appris à optimiser une fonction avec un cache (chose que je devrais essayer de faire systématiquement).

- Tableaux

Le cache est un tableau d'entiers longs remplis avec la fonction récursive opti_fly_syracuse, alloué dynamiquement avec malloc avec une taille adaptée à la fonction (il va prendre la taille minimum possible).

Les compétences supplémentaire utilisées sont l'allocation dynamique (le tableau du cache est alloué avec malloc) et la modulation du programme.

J'ai eu quelques difficultés à savoir comment remplir et quand appeler le cache dans la fonction qui calcule la hauteur de la suite, mais j'ai finalement réussi à trouver (et à comprendre aussi).

TP9 – Exos malloc
Emilie Marti

Ce TP est principalement des exercices sur malloc et une introduction à valgrind, un outil de débug qui permet de vérigier si la mémoire allouée est bien libérée et montre où un segfault a été produit. Très utile aussi pour l'algorithmique.

J'ai aussi appris à allouer de la mémoire pour des tableaux à deux dimensions de façon « optimale » (lorsque les tableaux n'ont pas tous la même taille). Je ne savais pas qu'on pouvait faire des malloc sur des doubles pointeurs.

- I/O

Le programme utilise des arguments de la ligne de commande qui exécute le programme, demande des entiers sur l'entrée standard et affiche son contenu sur la sortie standard.

- Type

On travaille sur des char et des entiers principalement (avec des pointeurs pour les tableaux).

- Compilation

Il y a un makefile pour compiler le programme avec la simple commande « make ».

- Tableaux

On travaille sur des tableaux d'entiers et char à 1 et 2 dimensions.

- Pointeurs

Lorsqu'on utilise malloc on traite les tableaux comme des pointeurs. Dans ce TP, on utilise même des doubles pointeurs.

- Allocation

Le TP porte sur de l'allocation de mémoire.

Par manque de temps, **il manque le dernier exercice** ; demandant une certaine réflexion pour bien le réaliser (comprendre le fonctionnement de malloc et free), il sera terminé au S2 juste après le TP7.