IN 301 Langage C – TD

Version du 08 septembre 2020

Pierre Coucheney – pierre.coucheney@uvsq.fr
Coline Gianfrotta – coline.gianfrotta@ens.uvsq.fr
Maël Guiraud – mael.guiraud@uvsq.fr
Franck Quessette – franck.quessette@uvsq.fr
Yann Strozecki – yann.strozecki@uvsq.fr
Sandrine Vial – sandrine.vial@uvsq.fr

Table des matières

Durant ce semestre vous travaillerez sous l'environnement Linux de la machine virtuelle (ou mac OS) que ce soit pour les TDs ou pour le projet.

Dans cet environnement, vous utiliserez le terminal afin de vous déplacer dans l'arborescence des fichiers, d'éditer, compiler, débugger votre programme, et de mettre vos fichiers sous contrôle de version sur un répertoire distant.

Pour cela, vous utiliserez les programmes suivants :
— `geany` pour l'édition,
— `gcc` pour la compilation,
— `gdb` pour le débuggage,
— `git` pour le contrôle de version.

Biblio

— **Le langage C, 2ème édition** – Brian W. Kernighan, Dennis M. Ritchie
— **Programmer en langage C - 5ème édition** – Claude Delannoy

Plan du Cours

Dans toute la suite les sections ou exercices sont notés $\mathcal{P}$résentiel ou $\mathcal{D}$istanciel.

BLOC 1 : Fichiers

— Environnement de programmation, éditeur, compilateur, arborescence des fichiers, fenêtre shell, ligne de commande, commandes shell de base, commande de compilation.
— Entrées sorties dans les fichiers, sur l'écran, principe.
— Makefile.
— Tableaux statiques.

BLOC 2 : Adresses

Adresse mémoire, Opérateurs `*` et `&`, malloc, free.

Chaînes de caractères, Implémentation des tableaux multidimmensionnels, arguments de la ligne de commande, atoi, atof, constructeur de type (tableaux et `struct`).

BLOC 3 : Compil

Compilation séparée
Listes chaînées simples

BLOC 4 : Piles et Files

Implémentations de piles et files

Processus, `fork()`, `exec()`, Environnement d'exécution d'un processus

BLOC 5 :

BLOC 6 :

1 BLOC 1 : Fichiers – Environnement de programmation

1.1 $\mathcal{P}$ Lecture et écriture dans les fichiers

Dans toute cette section, comme dans la suite du poly, vous écrirez un `Makefile` permettant de compiler et d'exécuter les programmes.

$\mathcal{P}$ Exercice 1. Premier fichier `.h`

Créer un fichier `constantes.h` qui contient les lignes suivantes qui définissent trois constantes :

```
#define N 10000
#define MAX 1000000
#define NOMFIC "nombres.data"
```

$\mathcal{P}$ Exercice 2. Écriture dans un fichier

a. Écrire un programme `genere.c` qui génère `N` entiers positifs aléatoires compris entre zéro et `MAX`. Ces nombres seront écrits dans le fichier `NOMFIC`. Afin de pouvoir utiliser les constantes dans ce programme, ajouter au début du programme, la ligne :

```
#include "constantes.h"
```

L'ouverture, l'écriture et la fermeture du fichier se feront dans une fonction :

```
void ecrire ()
```

Dans le `main` l'appel se fera par :

```
ecrire ();
```

Formattez l'affichage des nombres avec `%6d` pour que le fichier généré soit plus lisible.

$\mathcal{P}$ Exercice 3. Lecture depuis un fichier

a. Écrire un programme `algos.c` qui déclare en variable globale un tableau de taille `N` :

```
int T[N];
```

et qui lit les nombres qui sont dans le fichier `NOMFIC` généré à l'exercice précédent et les stocke dans les cases du tableau de `0` à `N-1`.

Vous devez faire une fonction :

```
void lecture ()
```

qui lit dans le fichier `NOMFIC` fichier et remplit le tableau `T`. Vous devez comme dans l'exercice précédent écrire :

```
#include "constantes.h"
```

au début de votre programme.

b. Programmer une fonction qui écrit les nombres qui sont dans le tableau dans un fichier `nombres-verif.data`

c. Utiliser la commande `diff` pour vérifier que les fichiers sont identiques :

```
diff nombres.data nombres-verif.data
```

diff affiche les différences, donc si diff n'affiche rien c'est que les deux fichiers sont identiques.

d. Modifier le code en supposant que l'on ne connait pas le nombre de valeurs dans le fichier `NOMFIC`. On peut alors utiliser la valeur de retour de `fscanf` pour savoir «quand s'arrêter». Tapez

```
man 3 fscanf
```

et lisez la section `valeur de retour`.

Le `Makefile` sera commun avec celui de l'exercice précédent.

$\mathcal{P}$ Exercice 4. premier algo

a. Dans le programme `algos.c` écrire une fonction :

```
int recherche (int x)
```

qui renvoie `1` si le nombre `x` est dans le tableau et `0` sinon.

b. Dans la fonction `recherche` ajouter un compteur pour compter le nombre de comparaisons que fait la fonction. Afficher ce nombre de comparaisons avant le retour de la fonction.

c. Écrire une fonction `stat_recherche(int x)` qui fait comme la fonction `recherche` mais renvoie la valeur du compteur du nombre de comparaisons.

d. Dans la fonction `main` appeler dix mille fois la fonction `stat_recherche` en lui donnant en argument une valeur aléatoire entre zéro et $5 \times$ `MAX`. Calculer le nombre comparaisons en moyenne. Ce nombre était-il prévisible ?

$\mathcal{P}$ Exercice 5. Commande wc

Programmer l'équivalent de la commande du terminal `wc -c` qui donne le nombre de caractères d'un fichier texte passé en argument. Le programme sera appelé de la manière suivante :

```
./monwc fichier
```

1.2 $\mathcal{D}$ Environnement de programmation

Dans un premier temps, vous allez apprendre les fonctionnalités de base du terminal. Les commandes que vous utiliserez sont :

```
man, ls, cd, pwd, mkdir, mv, rm, wc, cat
```

leur description peut être obtenue avec la commande `man commande`. Deux fonctionnalités du terminal qui s'avèrent très utiles à l'usage, et que vous vous efforcerez d'utilisez dès le début, sont la complétion automatique avec la touche tabulation, et le parcours de l'historique des commandes avec les flêches haut et bas.

$\mathcal{D}$ Exercice 6. Terminal et compilation

a. Ouvrez un terminal. Dans quel répertoire vous trouvez-vous ? Que se passe-t'il si vous appuyer sur la touche `w` puis deux fois sur `TAB` dans le terminal ? Puis `wh` et deux fois sur `TAB` ? Puis `whe` et une fois sur `TAB` ? Que fait la commande que vous venez de taper ? Essayer `whereis ls`.

b. Déplacez vous vers le répertoire `Bureau`. Revenez dans le répertoire initial, puis encore dans le répertoire `Bureau` sans taper complètement au clavier `Bureau`.

c. Affichez ce que contient le répertoire `Bureau`. Créez un répertoire `essai`, et vérifiez que celui-ci a bien été créé.

d. Déplacez vous dans le répertoire `essai`, puis éditez un fichier vide `fic1` avec `geany` en tapant la commande :

```
geany fic1 &
```

puis fermez `geany`. Que contient le répertoire maintenant ?

e. Copiez le fichier vers un fichier `fic2`. Changez le nom de `fic2` en `fic3`. Supprimez le fichier initial `fic1`.

f. Supprimez tous les fichiers et ajoutez le programme `debug.c` qui est sur `e-campus2`. Affichez le programme dans le terminal. Combien de lignes et de caractères contient-il ?

g. Nous allons maintenant compiler le programme. Taper la commande

```
gcc debug.c
```

Vérifiez qu'un fichier `a.out` a été créé. Executez le programme en tapant la commande `./a.out`. Manifestement ce programme contient un bug... mais vous ne le corrigerez pas pour l'instant.

h. Pour choisir un nom à votre exécutable, tapez la commande `gcc -o nom debug.c`. Vérifiez qu'un exécutable ayant le nom que vous avez choisi a bien été créé, et exécutez le.

i. Compilez en ajoutant les warnings de compilation avec la commande

```
gcc -o nom -Wall debug.c
```

Que constatez-vous ?

Notez bien que pour compiler un programme qui utilise la librairie mathématique, il faut ajouter l'option `-lm`. Au final, cela donne la commande suivante :

```
gcc -o nom -Wall debug.c -lm
```

$\mathcal{D}$ Exercice 7.

Organisation de vos dossiers Afin de de stocker correctement vos fichiers vous devez créer les dossiers suivants :

- un dossier `IN301`
- dans ce dossier un dossier par TD, commencez donc par un dossier `TD 01 - Environnement de programmation` correspondant au TD de cette section
- Vous pouvez également créer un dossier par séance de TD avec la commande :

```
for x in $(seq  1 12); do mkdir "td_$x"; done
```

- Vous pouvez créer un dossier `Notes de Cours` et un dossier `Projet`.
- Pour cette séance vous travaillerez dans le dossier `TD 01 - Environnement de programmation`.
- Vous sauvegarderez tous les fichiers de ce TD dans ce dossier.

$\mathcal{D}$ Exercice 8.

Utilisation de commandes shell

a. Écrire un programme C qui génèrer dix milles nombres aléatoires entre 0 et 1 000. Utiliser la commande :

```
man 3 rand
```

pour voir la syntaxe des fonctions `rand()` et `srand()`. Dans votre programme pour afficher à l'écranla syntaxe est :

```
int a;
a = rand();
printf("%d\n",a);
```

b. Utiliser `srand()` avec comme argument `getpid()` pour changer l'initialisation des nombres aléatoires. Faites `man 2 getpid()`

c. Quelle est la commande de compilation correcte ?

d. Utiliser une redirection pour rediriger le résultat dans un fichier `nombres.data`.

e. Utiliser la command `wc` pour compter le nombre de nombres générer

f. Utiliser la command `sort -n` (faites `man sort` pour voir la doc) pour trier le fichier et rediriger le tri vers un autre fichier `nombres_tries.data`

g. Utiliser la command `uniq` pour supprimer les doublons et mettre le résultat dans `nombres_uniq.data`. Combien y a-t-il de nombres différents ?

h. mettre toutes ces commandes dans un fichier, le rendre exécutable et éécutez le.

i. Enchaîner toutes les commandes en une seule ligne pour compter le nombre de nombres différents.

$\mathcal{D}$ Exercice 9. Contrôle de version : git

Vous allez utiliser un gestionnaire de version qui s'appelle `git`, et un dépôt distant qui s'appelle `github`. Il y a (au moins) 2 intérêts à cela par rapport à une sauvegarde classique : d'une part vous conservez l'historique de toutes les versions et pouvez revenir en arrière en cas d'erreur ou de suppression inopinée de votre travail, et d'autre part, vous faites des sauvegardes sur un dépôt distant qui vous prémunit des accidents que pourrait toucher votre machine personnelle. L'objectif de cet exercice est de vous familiariser avec l'utilisation de cet outil que vous utiliserez tout au long du semestre (et au-delà je l'espère !).

a. Ouvrez un navigateur et allez sur la page web : https://github.com/

b. Créez un compte en mémorisant bien votre nom d'utilisateur et votre mot de passe.

c. Une fois que vous êtes connecté, créez un dossier qui s'appelle `IN301` en cliquant sur la croix en haut à droite de l'écran (create new repository). Votre dépôt distant sous gestionnaire de version git est créé, vous allez maintenant le récupérer sur votre machine.

d. Dans votre terminal, déplacez vous à l'endroit où vous voulez ajouter le dossier `IN301`. Créez une copie locale en tapant la commande suivante dans le terminal :

```
git clone https://github.com/moi/IN301
```

et en remplaçant `moi` par votre nom d'utilisateur. Un répertoire `IN301` a normalement été créé, ce que vous pouvez vérifier en tapant la commande `ls`.

e. Déplacez vous dans le répertoire `IN301`. Créez le dossier `td0` (commande `mkdir`) [1]. Allez dans le dossier `td0` et créez le fichier vide `essai` avec la commande `touch essai`. Nous allons maintenant ajouter ce fichier au répertoire distant.

f. Pour mettre un nouvel élément en contrôle de version, il faut utiliser la commande `git add fichier`, donc, ici, `git add essai`. Pour valider cet ajout, utiliser la commande `git commit essai`, ou la commande `git commit -a` qui validera toutes les modifications faites sur des fichiers qui sont sous contrôle de version. Une fenêtre s'ouvre pour que vous renseigniez une description de la modification. Ecrivez, par exemple, "ajout fichier td0/essai", cliquez ensuite sur `CTRL+x`, puis sur `o` (pour OUI) et `ENTREE`. Pour propager cela au répertoire distant, il reste à taper la commande `git push`. Vérifiez ensuite sur votre compte `github` que le dossier a bien été ajouté.

g. Ajoutez le fichier `debug.c` que vous avez manipulé dans l'exercice précédent dans le répertoire `td0`. En suivant les mêmes instructions que pour `td0` faites en sorte de mettre ce fichier sous gestion de version et de l'ajouter dans `github`.

h. Une fois que cela a bien été réalisé, supprimez le répertoire `IN301` de votre machine (commande `rm -rf IN301`) et vérifiez que cela a bien fonctionné. Faites maintenant un clône de votre dépôt distant `github`, et vérifiez que vous avez bien récupéré le dossier `td0` et qu'il contient le fichier `debug.c`.

i. Ajoutez du texte dans le fichier `essai` et sauvegardez. Testez alors la commande `git status`. Commitez le changement. Que renvoie la commande `git status` ? Poussez votre changement sur `github` et testez de nouveau `git status`.

Dorénavant, pour ceux qui travaillent avec une machine en prêt, vous pourrez récupérer vos travaux en début de séance en clonant votre répertoire distant.

A tout moment, vous pouvez "commiter" vos changements (ne pas oublier de faire `git add` pour les dossiers et fichiers nouvellement créés), et les propager avec la commande `git push`. Localement vous pouvez vérifier si vos fichiers sont bien enregistré avec la commande `git status`.

Enfin, notez qu'il ne faut commiter que les fichiers textes (typiquement les programmes terminant par l'extension .c) ainsi que les répertoires qui les contiennent, et non les exécutables.

$\mathcal{D}$ Exercice 10. Compilation et débug

Récupérez le fichier `debug.c` de l'exercice précédent. Nous allons utiliser le logiciel `gdb` afin de débugger ce programme.

a. Sans éditer le programme `debug.c`, compilez et exécutez le (voir exercice 1).

Les commandes de base du débugger `gdb` sont

```
run, quit, break, bt, print, step, next
```

b. Pour exécuter le programme dans le débugger, lancer la commande `gdb ./progDebug` dans le terminal, puis la commande `run`. Que se passe-t'il ?

c. Essayez d'ajoutez un point d'arrêt à la ligne 11 (commande `break 11`). Cela ne doit pas être possible car l'exécutable ne sait pas faire référence au fichier source. Il faut pour cela ajouter l'option `-g` à la compilation. Pour cela quittez `gdb` (commande `quit`) et recompilez en ajoutant l'option.

1. Dorénavant, pour la feuille de td i, vous créerez un dossier `tdi` à cet emplacement

d. Retournez dans `gdb` et ajoutez un point d'arrêt à la ligne 11 puis lancez l'exécution du programme. Afficher la pile des appels (commande `bt`), et la valeur des variables dans la fonction courante. Continuez l'exécution du programme pas à pas en affichant les valeurs des variables régulèrement.

e. Corrigez la fonction `factorielle`, puis faites de même pour les fonctions `somme` et `maximum` en vous aidant si nécessaire de `gdb`.

A Révisions du L1

Les exercices de cette section sont faits pour se mettre à jour du niveau de début de ce cours.

$\mathcal{P}$ Exercice 11. Etoiles

Écrire un programme qui affiche à l'écran 10 étoiles sous la forme suivante :

```
         *
        *
       *
      *
     *
    *
   *
  *
 *
*
```

$\mathcal{P}$ Exercice 12. Conversions ....

Ecrire un programme qui convertit un temps donné en secondes en heures, minutes et secondes (avec l'accord des pluriels).

Exemple d'exécution :

```
3620 secondes correspond à 1 heure 0 minute 20 secondes
```

$\mathcal{P}$ Exercice 13. Multiplication Egyptienne

Pour multiplier deux nombres, les anciens égyptiens se servaient uniquement de l'addition, la soustraction, la multiplication par deux et la division par deux. Ils utilisaient le fait que, si X et Y sont deux entiers strictement positifs, alors :

$$X \times Y = \begin{cases} (X/2) \times (2Y) & \text{pour X pair} \\ (X-1) \times Y + Y & \text{pour X impair} \end{cases}$$

Ecrire un programme qui, étant donnée deux nombres (dans l'exemple 23 et 87), effectue la mutliplication égyptienne, en affichant chaque étape de la façon suivante :

```
  23 x 87
= 22 x 87 + 87
= 11 x 174 + 87
= 10 x 174 + 261
= 5 x 348 + 261
= 4 x 348 + 609
= 2 x 696 + 609
= 1 x 1392 + 609
= 2001
```

$\mathcal{P}$ Exercice 14. Limites ....

Calculez la limite de la suite

$$S_n = \sum_{i=1}^{i=n} \frac{1}{i^2}$$

en sachant que l'on arrête l'exécution lorsque $|S_{n+1} - S_n| < \epsilon$, ϵ étant la précision fixée à l'avance par une constante.

$\mathcal{P}$ Exercice 15. Nombres premiers

Écrire un programme qui teste si un nombre est premier ou pas.

$\mathcal{P}$ Exercice 16. Nombres amis

Soit n et m, deux entiers positifs. n et m sont dits *amis* si la somme de tous les diviseurs de n (sauf n lui-même) est égale à m et si la somme de tous les diviseurs de m (sauf m lui-même) est égale à n.

Écrire une fonction qui teste si deux entiers sont des nombres amis ou non.

Écrire une fonction qui, étant donné un entier positif $nmax$ affiche tous les couples de nombres amis (n, m) tels que $n \leq m \leq nmax$.

Aide : 220 et 284 sont amis.

$\mathcal{P}$ Exercice 17. Racines

Ecrivez un programme qui calcule la racine d'un nombre à une erreur ε fixée par une méthode de dichotomie.

$\mathcal{P}$ Exercice 18. Les suites de Syracuse

On se propose de construire un petit programme qui permet d'étudier les suites dites de Syracuse :

$$u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}$$

La conjecture de Syracuse dit que quelle que soit la valeur de départ, la suite finit par boucler sur les valeurs 4,2,1,4,2,1,...

a. Construire un programme qui, à partir d'une valeur de départ u_0, affiche les valeurs successives jusqu'à tomber sur la valeur 1 ;

b. modifier le programme pour qu'il compte le nombre d'itérations, sans affichage intermédiaire ;

c. modifier le programme pour qu'il affiche le nombre d'itérations pour toutes les valeurs de départ entre 1 et une valeur fixée.

$\mathcal{P}$ Exercice 19. Factorielle

Pourquoi la fonction suivante peut donner des résultats faux ?

```
int factorielle (int n)
{
    int f;
    if (n <= 1)
        f = 1;
    else
        f = n * factorielle(--n);
   return f;
}
```

$\mathcal{P}$ Exercice 20. Calcul de suite

Calculer les valeurs successives de la suite :

$$u_n = \sqrt{1 + \sqrt{2 + \sqrt{\ldots + \sqrt{n}}}}, \text{ pour } 1 \leq n \leq N.$$

$\mathcal{P}$ Exercice 21. Table ASCII

Le code ASCII permet de coder chaque caractère (imprimable par une machine) à l'aide d'un octet. Ecrire un programme qui affiche le code ASCII. Par exemple, `65` code pour `A`, `48` code pour `0`, etc ....

$\mathcal{P}$ Exercice 22. Tableau

Dans la suite vous utiliserez un tableau d'entiers comportant `N` cases où `N` est une constante.

a. Ecrire une fonction qui initialise toutes les cases du tableau à 1.

b. Ecrire une fonction qui affiche le produit des éléments d'un tableau.

c. Ecrire une fonction qui retourne le minimum d'un tableau

d. Ecrire une fonction qui effectue un décalage de 1 case à droite de tous les éléments d'un tableau. La case à gauche est affectée à 0. Le dernier élément est supprimé du tableau.

e. Ecrire une fonction qui insère une valeur dans un tableau trié. Après l'insertion, le tableau est toujours trié. Le dernier élément du tableau est supprimé.

f. Ecrire une fonction qui inverse les éléments d'un tableau. Cette inversion s'effectue sur le tableau lui-même (n'utilisez pas de tableau intermédiaire).

g. Ecrire une fonction qui élimine les valeurs en double (ou plus) d'un tableau d'entiers positifs en remplaçant ces valeurs en double par leur valeur négative. La première apparition de la valeur reste inchangée.

h. On suppose que le tableau est découpé en section de nombres significatifs. Chaque section est séparée par un ou plusieurs 0. Ecrire une fonction qui calcule **la moyenne des produits** de chaque section. Par exemple, si l'on dispose du tableau suivant : `1 2 3 0 0 5 4 0 0 8 0 10 11`. Il y a 4 sections. La moyenne des produits correspond à

$$\frac{(1.2.3) + (5.4) + 8 + (10.11)}{4} = 36,0$$

$\mathcal{P}$ Exercice 23. Tri

a. Remplir un tableau de valeurs aléatoires comprises entre 0 et 99.

b. Calculer le nombre de valeurs différentes dans le tableau.

c. Calculer le tableau d'entiers de taille 10 dont l'élement indicé par i contient le nombre de valeurs aléatoires dont la division par 10 vaut i.