IN 301 Langage C – TD

Version du 18 septembre 2016

Pierre Coucheney – Pierre.Coucheney@uvsq.fr
Franck Quessette – Franck.Quessette@uvsq.fr
Yann Strozecki – Yann.Strozecki@uvsq.fr

Table des matières

Durant ce semestre vous travaillerez sous l'environnement Linux de la machine virtuelle que ce soit pour les TDs ou pour le projet. Dans cet environnement, vous utiliserez autant que possible le terminal afin de vous déplacer dans l'arborescence des fichiers, d'éditer votre programme, de compiler votre programme, de débugger votre programme, et de mettre vos travaux 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.

1 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.

Exercice 1. Terminal et compilation

a. Ouvrez un terminal (essayer la combinaison de touches `CTRL+ALT+T`). 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, 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
```

Exercice 2. Contrôle de version : git

Comme expliqué en cours, 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]. Alle 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.

Exercice 3. 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`.

2 Rappels

Exercice 4. Etoiles

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

Exercice 5. 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
```

Exercice 6. 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
```

Exercice 7. 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.

Exercice 8. Nombres premiers

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

Exercice 9. 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.

Exercice 10. Racines

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

Exercice 11. 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.

Exercice 12. 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;
}
```

Exercice 13. Calcul de suite

Calculer les valeurs successives de la suite :

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

Exercice 14. 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 ....

Exercice 15. 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

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

Exercice 16. 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.

3 Pointeurs, structures, listes chaînées

Exercice 17. Pointeurs

a. Afficher (et comprendre) la taille en mémoire des types suivants :

```
char; int; double; char*; void*; int*; double*; int**; int[10]; char[7][3]; int[]
```

Soit `tab` un tableau de 10 caractères déclaré par `char tab[10];`
Afficher la taille mémoire de

```
tab; tab[0]; &tab[0]; *&tab; *&tab[0]
```

Soit `p` une variable définie par `char (*p)[10] = &tab`
Afficher la taille mémoire de

```
p; *p; (*p)[2]; &(*p)[2];
```

b. Ecrire une fonction qui ne renvoie rien et dont l'effet de bord est de permuter la valeur de deux entiers passés en paramètres.

c. Que produit l'appel à la fonction suivante ?

```
void reinitPointeur(int* p){
   p = NULL;
}
```

Pour tester, on peut par exemple exécuter les instructions suivantes dans la fonction `main()` en ajoutant les instructions d'affichages adéquates :

```
int a = 1;
int* p = &a;
reinitPointeur(p);
```

Comment modifier la fonction pour que la réinitialisation du pointeur soit effective ?

Exercice 18. Structure, tableaux

a. Définir une structure appelée `Tableau` qui contient deux champs :

1. `taille` : un entier.
2. `tab` : un tableau de taille 100 dont les `taille` premiers entiers sont significatifs ;

b. Quelle est la taille mémoire de cette structure ?

c. Ecrire une fonction

```
int alea(int n)
```

qui renvoie un entier tiré aléatoirement entre `0` et `n`. Pour cela voir la documentation des fonctions `rand()` et `srand()` de la librairie `stdlib`.

d. Ecrire une fonction qui initialise une variable de la structure : `taille` vaut 10, et le tableau est rempli avec 10 entiers choisis aléatoirement entre 0 et 20.

e. Ecrire une fonction qui affiche les éléments du tableau.

f. Ecrire une fonction qui retourne le produit des éléments du tableau.

g. Ecrire une fonction qui retourne la valeur minimale du tableau.

h. Ecrire une fonction qui effectue un décalage de 1 case à droite de tous les éléments d'un tableau. La valeur 0 est affectée à la case à gauche. La taille du tableau est donc augmentée de 1.

i. Ecrire une fonction qui trie les éléments du tableau.

j. Ecrire une fonction qui insère une valeur dans un tableau trié. Après l'insertion, le tableau est toujours trié. La taille du tableau est augmentée de 1.

k. Ecrire une fonction qui inverse les éléments d'un tableau. Cette inversion s'effectue sur le tableau lui-même sans utiliser de tableau intermédiaire.

l. Ecrire une fonction qui supprime un élément choisi au hasard dans le tableau. La taille du tableau est diminuée de 1.

m. Ecrire une fonction qui élimine les valeurs en double (ou plus) du tableau.

n. Programmer le tri par selection, le tri par insertion, le tri à bulle.

o. Instrumenter le code pour que les fonctions de tri renvoie le nombre d'accès en lecture et écriture du tableau.

p. Pour une taille de tableau fixée, générer plusieurs tableaux et calculer le nombre moyen de comparaisons effctuées.

q. Faire varier la taille des tableaux (par exemple de 100 à 1500 par pas de 100), et afficher à chaque fois le nombre moyen de comparaisons effectuées par les différents méthodes de tri. Les valeurs obtenues peuvent être affichées sur une courbe de la manière suivante :

1. Faire en sorte que l'affichage soit sur 2 colonnes où
   — la première colonne contient la taille des tableaux générés
   — la deuxième colonne contient le nombre moyen de comparaisons
2. Rediriger les affichages dans le terminal vers un fichier. Si l'exécutable s'appelle `a.out`, cela se fait avec la commande

   ```
   ./a.out > resultat
   ```

3. Démarrer le logiciel `gnuplot` dans le terminal.
4. Afficher les résultats avec la commande

   ```
   gnuplot> plot "./resultat" with lines notitle
   ```

5. Pour comparer la courbe obtenue avec le graphe d'une fonction `f` (ici le nombre de comparaison "théorique") :

   ```
   gnuplot> replot f(x)
   ```

Cette méthode pourra être utilisée par la suite pour visualiser le nombre d'opérations moyen effectuées par les tris.

r. Implementer un algorithme de tri en temps linéaire quand les valeurs des tableaux sont restreintes à l'intervalle $[0 \dots 100]$.

s. Implémenter le tri rapide.

Exercice 19. Listes chaînées

Dans cette section, on programmera les fonctions de manipulation de listes suivantes en utilisant des fonctions récursives lorsque cela est possible :

a. Définir la structure de données `Liste` qui implémentera la liste chaînée.

b. Les fonctions de base : créer une liste vide, tester si une liste est vide, afficher une liste, libérer la mémoire.

c. Fonctions d'ajout : ajout au début, ajout à la fin, ajout trié. Pour la dernière insertion, on écrira au préalable une fonction qui vérifie que la liste est bien triée. Pour tester ces fonctions, on créera une liste en ajoutant des entiers générés aléatoirement entre 0 et n jusqu'à ce que 0 soit tiré.

d. Retourner le nombre d'éléments de la liste.

e. Recherche un élément.

f. Supprimer un élément.

g. Concaténer deux listes.

h. Entrelacer deux listes triées pour former une nouvelle liste triée.

i. Implémenter le tri à bulles sur une liste.

j. Renverser une liste chaînée.

k. Implémenter le tri fusion dans une liste d'entiers.

l. Implémenter le crible d'Eratosthene avec une liste afin de trouver tous les nombres premiers plus petits qu'une valeur passée en argument du programme.

m. Étant donnée une liste d'entiers, on cherche à les classer selon leur valeur modulo un entier K passé en paramètre de la fonction que vous écrirez. Par exemple, si $K = 2$, on veut séparer les valeurs paires des valeurs impaires. Proposer une structure adaptée au classement et implémenter le classement.

4 Chaînes de caractères

En C, il est possible d'initialiser un tableau de caractères par une chaîne de caractère de la manière suivante :

```
char c[] = "salut"
```

Le tableau se termine alors par le caractère de fin de chaîne `'\0'`. Sur l'exemple, il s'agit donc un tableau de taille 6. Cela permet l'usage de fonctions adaptées aux chaînes de carctères, fournies dans la librairie `string.h`, telle que la fonction `strlen()` qui donne la taille du tableau sans le caractère de fin de chaîne.

Exercice 20. Arguments de la ligne de commande

Une utilisation des chaînes de caractères est le passage d'argument au programme dans la ligne de commande. Pour cela, le prototype de la fonction `main()` doit être adapté de la manière suivante :

```
int main(int argc, char** argv)
```

où `argv` est un tableau de chaînes de caractères, et `argc est la taille du tableau`.

a. Tester et comprendre le programme suivant :

```
#include <stdio.h>
int main( int argc, char** argv){
  int i;
  for(i=0; i<argc; i++)
    printf("%s\n", argv[i]);
  return 0;
}
```

en l'appelant avec une ligne de commandes qui contient n'importe quel nombre d'arguments, par exemple :

```
./a.out 6 toto salut2016 ?!:;
```

b. Pourquoi la variable `argv` est-elle de type `char**` ?

c. En utilisant la fonction de conversion `atof()` (voir le manuel), écrire un programme qui affiche la somme des nombres passés en arguments. Par exemple la commande

```
./a.out 6.2 -8 2.0
```

doit afficher 0.2.

d. En utilisant la fonction `isdigit()` (voir le manuel), vérifier au préalable que les arguments passés sont bien des nombres flottants éventuellement négatifs.

Exercice 21. Tableau et liste de caractères

L'objectif est d'écrire des fonctions simples de manipulation des chaînes de caractères. Nous travaillerons avec deux chaînes de caractères qui seront passées en argument de l'appel au programme de la manière suivante :

```
./programme chaine1 chaine2
```

La première chaîne sera stockée de manière classique sous forme de tableau de caractère terminé par le caractère `'\0'`. La deuxième sera implémentée par une liste chaînée écrite dans la mémoire dynamique. Vous pouvez réutiliser/adapter les fonctions de manipulation des chaînes que vous avez vues.

a. Définir la structure de données `ListeC` permettant de stocker la chaîne de caractères sous forme de liste chaînée, et écrire une fonction d'ajout d'un caractère dans la liste. Ecrire la fonction `afficheChaine` qui affiche la liste chaînée, et la fonction

```
ListeC ajoutChaine(char* t, ListeC l)
```

qui écrit la chaîne de caractères `t` dans la liste `l` de manière *récursive*.

b. Écrire la fonction

```
void remplitMot(char** argv, int argc, char** t, ListeC* l)
```

qui affecte la première chaîne passée en argument du programme dans `t` et la deuxième dans `l`. Vous vérifierez au prélable que le nombre d'arguments du programme est correct, sinon le programme doit se terminer avec un message d'erreur.

c. Ecrivez une fonction *récursive* qui compare les deux chaînes (resp. `t` et `l`) selon l'ordre lexicographique : le comportement de la fonction imite celui de la fonction `strcmp()` (voir le manuel) pour la valeur de retour.

d. Ecrivez une fonction *recursive* qui teste si les deux chaînes (resp. `t` et `l`) sont symétriques l'une de l'autre.

e. Ecrire une fonction qui renvoie une variable de type `ListeC` qui contient le plus grand préfixe commun entre les deux chaînes.

5 Compilation et Makefile

Exercice 22. Écrire un fichier makefile

Créer un répertoire `Exo1`. Récupérer le fichier `exemple_decoupe.tar` sur e-campus et copier-le dans le répertoire `Exo1`. Décompresser l'archive à l'aide de la commande `tar xvf exemple_decoupe.tar`.

a. Faites un schéma de dépendances entre les différents fichiers.

b. Ecrivez dans un fichier `makefile` les commandes qui permettent de compiler chacun des fichiers séparément.

c. Rajouter une règle qui permet de faire l'édition de liens entre les fichiers objets et de créer un fichier exécutable.

Exercice 23. Écrire un ensemble de fichiers et un fichier Makefile

On désire écrire un programme pour calculer la circonférence et la surface d'un cercle étant donné son rayon. Le programme doit être composé de plusieurs fichiers :

`principal.c`	contient la fonction `main`
`circonference.c`	contient la fonction `circonference`
`surface.c`	contient la fonction `surface`
`pi.h`	contient la définition de la valeur approchée de π

a. Écrivez le contenu des différents fichiers décrits ci-dessus.

b. Ecrivez le contenu des fichiers `conference.h` et `surface.h`.

c. Faites un schéma de dépendances entre les différents fichiers.

d. Écrivez ensuite le fichier `Makefile` permettant de compiler chacun des fichiers `.c` séparément et ensuite de faire l'édition de liens entre les différents fichiers objets.

e. Ajoutez une cible `clean` dans le `Makefile` pour supprimer tous les fichiers intermédiaires ainsi que le fichier résultat.

Exercice 24. Avec du graphisme

Récupérez l'archive `graphisme.tar` qui contient un fichier `main.c`, l'ensemble des fichiers nécessaires à l'utilisation de la bibliothèque `graphics.c` et un `Makefile` qui permet de compiler cette bibliothèque.

a. Complétez le `Makefile` pour que le programme puisse être compilé. Pour cela, la librairie `SDL` doit être incluse au moment de l'édition de lien avec la commande

```
`sdl-config --libs` -lSDL_ttf
```

b. Ajouter une règle `run` qui permet de lancer l'exécution du programme. Celui-ci fait les choses suivantes (lire le code pour le constater) : à chaque clic dans la fenêtre graphique, un point est ajouté dans une liste et affiché relié aux points précédents jusqu'à ce que `NB_POINTS` soient affichés. A tout moment, en appuyant sur la touche `a`, on peut supprimer le dernier point ajouté.

c. Découper le fichier `main.c` dans 3 fichiers :
- un fichier `liste.c` qui contient les opérations sur les listes, y compris l'affichage dans la fenêtre graphique ;
- un fichier `simul.c` qui contient la fonction qui génère la simulation ;
- un fichier `main.c` qui contient la fonction principale `int main()`.

Vous ajouterez les fichiers d'en-tête nécessaires. Les constantes (`#define`) et les définitions des structures seront également placées à des endroits adaptés.

d. Ajoutez les règles `all` et `clean` dans le `Makefile`.

e. Dans la fonction de simulation, vous ajouterez les fonctionnalités suivantes :
- quand la touche `z` est appuyée, le point *le plus ancien* qui a été ajouté est supprimé ;
- quand une flèche est appuyée, le dessin de la chaîne se décale d'un pixel dans la position correspondante ;
- quand la touche `t` est appuyée, les points sont triés par ordre d'abscisse dans la fenêtre graphique.

6 Lecture et écriture dans les fichiers

Exercice 25. Buffer

Récupérez le programme `buffer.c`.

a. Après avoir redirigé la sortie standard du programme dans un fichier, lisez le temps que le programme met à s'exécuter.

b. Tester les différents mode du buffer en modifiant les commentaires. Dans quel mode semble être paramétré le flux `stdout` par défaut ?

Exercice 26. Commande wc

Programmer 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 :

```
./a.out fichier
```

Exercice 27. Ça se dégrade

Dans cet exercice, vous devrez créer une image au format ppm ASCII. Un tel fichier contient 3 lignes d'en-tête :

— P3 sur la première ligne ce qui correspond au type de fichier (ppm en mde texte),
— la largeur de l'image et la hauteur de l'image séparées par un espace,
— le nombre de couleurs.

Ensuite les pixels de l'image sont donnés ligne par ligne de gauche à droite. Un pixel est représenté par trois nombres entre 0 et 255 qui représentent son intensité en couleur pour les trois couleurs de base, rouge, vert et bleu. Toutes ces valeurs sont séparées par un retour à ligne dans un fichier ppm. Voir http://fr.wikipedia.org/wiki/Portable_pixmap pour la documentation sur ce formats.

a. Créer une image de 256×256 où le point en haut à gauche est noir, celui en haut à droite est bleu, celui en bas à gauche est rouge et celui en bas à droite est magenta. Tous les autres points seront des dégradés de rouge et de bleu proportionnels à leurs coordonnées.

b. Quelle taille fait le fichier ?

c. Visualiser les images.

Exercice 28. Cryptographie

Le but de cet exercice est de cacher du texte dans une image.

Pour cacher une chaîne de n caractères dans une image, on va modifier les n premiers pixels (r, v, b) de l'image de la manière suivante : on coupe la valeur du caractère lu en trois ; ainsi, le caractère a représenté par 11000001 se coupe en $11 = 3$, $000 = 0$ et $001 = 1$. Ensuite on remplace les trois bits de poids faible, c'est-à-dire les trois derniers bits de v et b et les deux derniers bits de r par ces valeurs. Par exemple si $r = 10111101$, $v = 01010110$ et $b = 00010011$, alors le pixel modifié par le caractère a vaut $r = 10111111$, $v = 01010000$ et $b = 00010001$.

a. Quelle formule permet d'obtenir le pixel modifié (r', v', b') en fonction de la valeur ASCII du caractère lu ?

b. Ouvrir l'image `chat` fournie sur ecampus avec un visionneur d'image. L'ouvrir ensuite avec un éditeur de texte comme Geany.

c. Compilez le fichier `decrypt.c`, puis exécutez le sur le fichier `chatCrypte` de la manière suivante :

```
./a.out chatCrypte
```

Que dit le chat ? Voit-on une différence entre les images `chat` et `chatCrypte` ? Comprendre le code de `decrypt.c`.

d. Écrire le programme `crypt.c` qui affiche dans le terminal le fichier avec le message crypté. Celui-ci sera appelé par la commande suivante dans le terminal :

```
./a.out chat "message a cacher"
```

Si vous voulez l'écrire dans un fichier pour pouvoir le lire, vous redirigerez la sortie standard de la manière suivante :

```
./a.out chat "message a cacher" >fichierCrypte
```

Vous vérifierez que votre programme est correct en décryptant le fichier que vous avez obtenu.

Dans les 3 exercices qui suivent, vous répondrez aux questions suivantes :

1. Quelle taille fait le fichier généré ?
2. Dans le shell lancer la commande `file` avec le nom de fichier en argument, quel est le résultat ?
3. Ouvrir le fichier avec geany, que lisez-vous ?

Exercice 29. Mon premier fichier, que du texte, ça c'est sûr.

Écrire dans un fichier en mode texte :
— Sur la première ligne les lettres de l'alphabet en minuscule sans espace ;
— Sur la deuxième ligne les lettres de l'alphabet en majuscule sans espace ;
— Sur les lignes suivantes les nombres de 0 à 999 inclus en affichant chaque nombre sur 4 caractères et en revenant à la ligne tous les 10.

Exercice 30. Mon deuxième fichier que du binaire ?

Écrire dans un fichier en mode binaire sans séparateurs :
— Les lettres de l'alphabet en minuscule ;
— Les lettres de l'alphabet en majuscule ;
— Écrire le caractère '\n' ;
— Écrire les nombres de 0 à 999.

Exercice 31. Est-ce du binaire ou du texte ?

Écrire dans un fichier en mode binaire sans séparateurs :
— Les lettres de l'alphabet en minuscule ;
— Les lettres de l'alphabet en majuscule ;
— Écrire le caractère '\n' ;

Exercice 32. J'écris donc je lis

Choisissez le mode (binaire ou texte) qui vous semble le plus adapté pour effectuer les opérations suivantes :

a. Dans un premier programme, générer 10^6 entiers aléatoires positifs et les écrire dans un fichier `a.txt`.

b. Dans un deuxième programme, lire le fichier `a.txt` et écrire les nombres dans `pair.txt` ou `impair.txt` en fonction de leur parité.

Exercice 33. Patron de boîte.

On reprend le cadre de l'exercice 3.

a. Créer un patron de cube permettant de visualiser les différentes couleurs. Les huit sommets du cube doivent être noir, rouge, vert, bleu, jaune, cyan, magenta, blanc. Chaque face devant présenter le dégradé sur deux des trois composantes RGB.

b. Écrire ce patron dans un fichier au format ppm binaire.