INF304 — TP3

Placez-vous dans votre répertoire INF304 et copiez les fichiers nécessaires à ce TP :

```
cp -r /Public/304_INF_Public/TP3 .
```

Placez-vous dans votre répertoire `TP3`.

Votre répertoire contient un programme `ensembles` composé de différents modules (`sac`, `operations_ens` et `ensembles`). Ces modules permettent de définir des ensembles à partir d'un «ensemble» de référence (le *sac*) puis de réaliser des opérations classiques sur les ensembles.

Un *sac* `S` de taille `n` contient $n^2 = n \times n$ valeurs entières (éventuellement répétées plusieurs fois), et chaque *ensemble* (type `ensemble`) est défini à partir du *sac* par son cardinal `card` et un tableau de couples d'indice `indices` faisant référence au tableau du sac. Les valeurs d'un sac sont rangées dans un tableau à deux dimensions (`n×n`), chaque dimension est indicée de 0 à `n-1`. Chaque ensemble peut être vide ou contenir des valeurs **toutes distinctes entre elles**.

Par exemple, à partir du *sac* correspondant au fichier `valeurs_16` :

```
4
32  56   9   2
 7  87   2  45
10  72  55  56
57  23  14  34
```

L'ensemble `E` défini par

```
E.card = 3;
E.indices[0] = {0,2};
E.indices[1] = {1,0};
E.indices[2] = {3,1};
```

correspond à l'ensemble {9,7,23}. Par contre

```
E.card = 3;
E.indices[0] = {0,3};
E.indices[1] = {1,2};
E.indices[2] = {3,1};
```

est incorrect, car ne correspond pas à un ensemble (deux valeurs égales).

Deux *sacs* sont fournis : `valeurs_16` (4×4=16 éléments) et `valeurs_256` (16×16=256 éléments). Le nom du fichier sac à utiliser est à fournir en argument du programme.

1. Compilation séparée

▶ Exercice 1

Lisez les fichiers sources fournis. Repérez la structure du programme et les dépendances entre les différents modules. À l'aide de ces dépendances, écrivez un fichier `Makefile` permettant d'automatiser la compilation du programme.

Essayez votre `Makefile` : compilez le programme, modifiez le fichier `sac.c`, recompilez : quels sont les modules recompilés ?

Modifiez le fichier `sac.h`, puis recompilez, toujours à l'aide du `Makefile` : quels sont cette fois les modules recompilés ?

2. Tests boîte blanche

Le but de cette partie est de détecter les erreurs présentes dans le module `operations_ens` qui gère les opérations ensemblistes.

Le programme principal `ensembles` permet de définir au clavier différents ensembles puis de réaliser les opérations de base implémentées dans le module `operations_ens`. Il est aussi possible de générer des ensembles particuliers à l'aide des procédures `init_alea`, `init_vide` et `init_plein`.

Le fichier `saisie_123` donne un exemple d'entrées pour définir différents ensembles.

▶ Exercice 2

À l'aide des différents outils de génération d'ensembles (saisie manuelle ou génération d'ensembles particuliers), détectez les erreurs dans les opérations ensemblistes du module `operations_ens`.

NB : il est bien demandé de **détecter** les erreurs et non de les **corriger**. Vous pouvez à la fin du TP essayer de corriger les erreurs détectées **s'il vous reste du temps**.

3. Tests boîte noire

Votre répertoire contient 5 exécutables appelés `boite_noire_1`, `boite_noire_2`, `boite_noire_3`, `boite_noire_4` et `boite_noire_5`.

Ces programmes lisent deux chaînes de caractères S1 et S2 dans un fichier dont le nom est donné en argument, et affichent :

- la position de S2 dans S1, si S2 est une sous-chaîne de S1 (la position 0 correspond au 1er caractère)
- -1, si S2 n'est pas une sous-chaîne de S1.

Le fichier d'entrée contient deux chaînes de caractères S1 et S2, chacune sur une seule ligne. S1 et S2 ne comportent que des caractères alpha-numériques (lettres de l'alphabet minuscules ou majuscules et chiffres), ou des espaces. Le caractère de retour chariot à la fin de la ligne n'est pas inclus dans la chaîne. S2 ne peut pas être la chaîne vide.

Par exemple, si le fichier d'entrée contient :

```
aaab
ab
```

alors, le programme affichera "2" ("ab" est une sous-chaîne de "aaab" et apparaît à la position 2).

Si le fichier d'entrée contient :

```
aaab
ba
```

alors, le programme affichera "-1" ("ba" n'est pas une sous-chaîne de "aaab").

▶ Exercice 3

Un seul de ces programmes est correct (... ou du moins, ne contient aucun «bug volontaire»). Votre mission consiste à détecter lequel, et à fournir le jeu de test mettant en évidence le fait que les autres ne sont pas corrects.

▶ Exercice 4 - Concours de tests !

Écrivez votre propre version de la fonction `recherche`, et introduisez un bug (le plus discret possible !). Vous pouvez reprendre ou vous inspirer de la fonction dans le fichier `recherche.c`, qui est correcte — du moins, qui ne contient aucun bug «volontaire».

Vérifiez que le jeu de tests écrit à l'exercice précédent «trouve» bien votre bug. Si ce n'est pas le cas, ajoutez au moins un test mettant ce bug en évidence.

À la fin de votre TP, vous allez soumettre sur Caséine :

- votre jeu de test ;
- votre version de `recherche.c`, contenant le «bug volontaire» que vous avez introduit.

Le concours consistera ensuite à «jouer» les jeux de tests les uns contre les autres : le but du jeu étant d'avoir un jeu de tests le plus complet possible (afin de «trouver» les bugs des fonctions des autres binômes), et le bug le plus «discret» possible (afin que les jeux de tests des autres binômes ne le trouve pas...).

Concrètement, ce «concours» se déroulera de la manière suivante, une fois tous les jeux de tests et fichiers `recherche.c` fournis :

Pour chaque paire de binômes A et B :

1. le «match aller» consiste à faire tourner les tests du binôme A sur le programme du binôme B
    - A gagne le match si ses tests sont suffisants pour trouver le bug du programme de B
    - B gagne le match si les tests de A ne sont pas suffisants et qu'aucun bug n'est trouvé
2. le «match retour» consiste à faire tourner les tests du binôme B sur le programme du binôme A

Ces «matchs» seront réalisés par les enseignant·e·s, à partir des fichiers rendus par les binômes ; toutefois rien ne vous interdit d'essayer entre vous pendant le TP...

Règles du jeu :

1. Tous les tests fournis doivent être des tests fonctionnels et donc **respecter la spécification**
2. Les tests fournis par un binôme doivent comporter au moins un test permettant de «trouver» le bug du programme de ce même binôme
3. Le programme «buggué» doit compiler correctement

Compte-rendu

À l'issue du TP, déposer sur la page moodle du cours :

- Un compte-rendu d'une page maximum, au format PDF :
    - Pour l'exercice 2 : pour le programme `ensembles`, indiquez les opérations erronées du module `operations_ens` avec pour chacune le jeu de tests qui l'a mise en défaut.
    - Pour l'exercice 3 : quelle est la boîte noire correcte ? Pour chacune des autres, joignez le jeu de test minimal détectant les erreurs.
- Le fichier `Makefile` de l'exercice 1
- Une archive nommée `Tests.zip` contenant les tests des exercices 3 et 4
- Le fichier `recherche.c` de l'exercice 4, contenant un «bug volontaire»

>