

INF304 — TP2

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

```
cp -r /Public/304_INF_Public/TP2 .
```

Première partie : Tests et débogage

Écriture de tests

► Exercice 0

Prendre connaissance et comprendre le fonctionnement des différents modules du squelette de programme fourni.

Le but de ce TP est de tester soigneusement la procédure `tri_insertion` du paquetage `tri`.

► Exercice 1

Écrire un jeu de tests (c'est-à-dire, **plusieurs** fichiers de tests !) pour tester la fonction de tri.

Compilez le programme `exec_test_tri` à l'aide de la commande `make exec_test_tri` (ou simplement `make`).

Testez le programme `exec_test_tri` sur le jeu de tests que vous avez créé. La fonction `tri_insertion` est-elle correcte ?

Débogage

Les outils de *débogage* permettent de suivre l'exécution d'un programme pas à pas, et d'avoir accès à chaque pas à l'état de la mémoire (zones allouées, valeur des variables et paramètres). Un des outils de débogage les plus largement utilisés est l'outil `gdb`. Cet outil — en ligne de commande — est souvent utilisé *via* une interface graphique : VScode par exemple permet d'utiliser `gdb`.

L'utilisation d'un outil de débogage nécessite de compiler le code source avec une option particulière (`-g`).


1. compilez le programme `exec_test_tri` en tapant la commande `make exec_test_tri`
2. notez la taille des fichiers `.o` en tapant la commande `ls -l *.o`
3. modifiez la première ligne du fichier `Makefile` en ajoutant l'option `-g` :
`CC=clang -g`
4. recompilez le programme `exec_test_tri` en tapant les commande `make clean`, puis à nouveau `make exec_test_tri`
5. notez à nouveau la taille des fichiers `*.o`


L'option `-g` ajoute des informations dans le code compilé du fichier `.o` permettant de faire la correspondance avec les instructions du code source.

Utilisation de gdb avec VScode

Si vous ne l'avez pas fait lors du premier TP, copiez les deux fichiers de configurations `tasks.json` et `launch.json`, depuis le répertoire `/Public/304_INF_Public/.vscode`, dans votre propre répertoire `.vscode` (cf « [Mise en place des fichiers de configuration](#) »).

Ouvrez le fichier source `tri.c` (Fichier/Ouvrir un fichier source), placez un point d'arrêt au début de la fonction `tri_insertion` : un *point d'arrêt* se place en cliquant sur la partie gauche de l'éditeur (à gauche des numéros de ligne). Un point d'arrêt placé est signalé par un point rouge : il s'agit d'un point du programme où l'exécution va être arrêtée lors du débogage.

Une fois le programme compilé avec l'option `-g`, et le point d'arrêt placé, ouvrez le programme principal `exec_test_tri.c` dans une fenêtre, puis lancez le débogueur (bouton , ou menu Exécuter/Démarrer le débogage). VScode vous demande d'entrer un argument au programme débogué (ici, ce sera le nom du fichier d'entrée contenant un tableau à trier).

Observez les valeurs de `t`, `t->taille`, `i`. Observez le fonctionnement des différentes commandes disponibles (barre d'outils , ou menu Exécuter) : «pas à pas principal», «pas à pas détaillé», «pas à pas sortant».

► Exercice 2

Utilisez l'outil de débogage pour trouver et corriger les éventuelles erreurs de la fonction `tri_insertion`.

Automatisation des tests

► Exercice 3

On souhaite *automatiser* l'exécution de ces tests. Compléter la procédure `test_tri_insertion` (dans le fichier `test_tri.c`) afin que cette procédure, pour chaque nom de fichier fourni comme argument de la liste de commande :

1. lise le tableau contenu dans le fichier
2. trie ce tableau à l'aide de la procédure `tri_insertion`
3. écrive le tableau trié dans un nouveau fichier (dont le nom correspond par exemple au nom du fichier d'entrée, suffixé par `.out`).

NB: on peut recopier une chaîne de caractères dans une autre à l'aide de la fonction `strcpy`, et concaténer une chaîne à une autre à l'aide de la fonction `strcat` (déclarées dans le module `string.h`) :

```
#include

void exemple_concatenation(char * nom) {
    char nouveau_nom[256];

    strcpy(nouveau_nom, nom);
    strcat(nouveau_nom, ".out");
    ...
}
```

► Exercice 4

Il peut être fastidieux de vérifier «à la main» (ou plutôt «aux yeux»...), pour chaque test, que le résultat de la procédure est correct.

Écrivez un *oracle* pour la procédure de tri, c'est-à-dire une procédure ou une fonction dont le rôle est de *vérifier* que le résultat rendu par la procédure testée correspond bien à sa spécification. Quelles sont les propriétés à vérifier pour une procédure de tri ?

Complétez la procédure `test_tri_insertion`, en utilisant cet oracle, afin d'afficher automatiquement quel(s) tests échouent.

► Exercice 5

On souhaite maintenant vérifier que le jeu d'essai écrit est pertinent. Modifiez l'algorithme afin d'y introduire une erreur (si possible assez subtile...). L'erreur introduite est-elle découverte par votre jeu d'essai ?

Deuxième partie : génération aléatoire de tests

On souhaite maintenant générer de manière aléatoire une partie des tests.

Génération aléatoire et pseudo-aléatoire

En informatique, pour simuler le hasard, on utilise des générateurs de nombres pseudo-aléatoires : “Un générateur de nombres pseudo-aléatoires, *pseudorandom number generator (PRNG)* en anglais, est un algorithme qui génère une séquence de nombres présentant certaines propriétés du hasard. Par exemple, les nombres sont supposés être approximativement indépendants les uns des autres, et il est potentiellement difficile de repérer des groupes de nombres qui suivent une certaine règle (comportements de groupe)” (cf. wikipedia).

En pratique, les PRNG sont initialisées grâce à une *seed* (semence, graine). Toute la suite de nombres produite par le générateur découle de façon déterministe de la valeur de la graine.

Pseudo-aléatoire : aspects pratiques C

Le paquetage `stdlib` définit deux fonctions permettant de tirer une séquence aléatoire de nombres :

- `rand()` renvoie un entier pseudo-aléatoire entre 0 et la constante `RAND_MAX`
- `srand(int)` : cette fonction prend un entier en paramètre, entier utilisé pour *initialiser* le générateur pseudo-aléatoire. Cette fonction doit être appelée une seule fois, au début de l'exécution du programme : la séquence de nombres pseudo-aléatoires est entièrement déterminée par cette valeur d'initialisation.

On peut obtenir un nombre pseudo-aléatoire sur l'intervalle $[0, N-1]$ à l'aide de l'expression C `rand() % N`.

Génération aléatoire de tests

► Exercice 6

Écrire une procédure permettant d'initialiser un tableau avec des valeurs aléatoires.

Compléter la procédure `test_tri_insertion_alea` afin de tester le tri à l'aide de tests générés de manière aléatoire. Vous pouvez utiliser à nouveau l'oracle de l'exercice 3.

Évaluez votre jeu de tests à l'aide d'une fonction `tri_insertion` erronée. Essayez de faire varier le nombre et la taille des tableaux générés.

Quel jeu de tests vous paraît le plus complet et pertinent : celui que vous avez écrit, ou celui généré aléatoirement ?

Compte–rendu

À l'issue du TP, déposer sur [la page moodle du cours](#) :

- Un compte–rendu d'une page maximum, au format PDF :
 - Expliquez votre démarche pour construire le jeu de tests de l'exercice 1
 - Décrire le fonctionnement de l'oracle de l'exercice 4, en particulier quelles propriétés sont testées
 - Les résultats et conclusions de vos deux phases de test.
- Le fichier `test_tri.c` modifié.