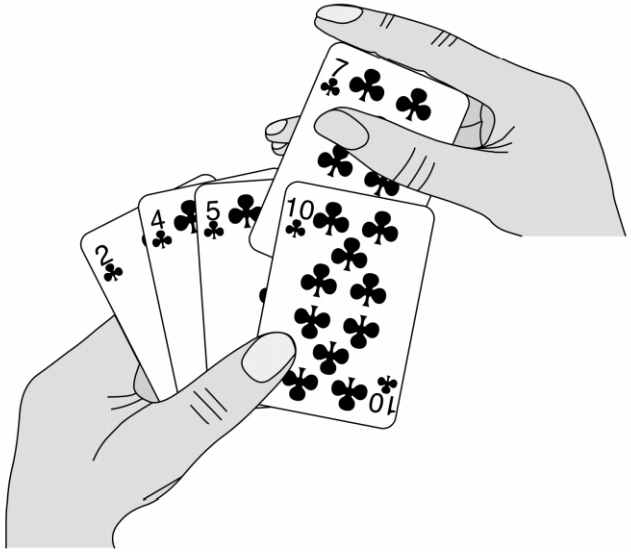

Projet 1 (tri) : feedback (code)

INFO2050 - Programmation avancée

Vérification d'inputs

- De quoi faut-il se prémunir ?
  - L'utilisateur est stupide : il faut vérifier tout ce qu'il entre !
    - (G)UI, données sensibles, etc.
    - Librairie ?
  - Les cas limites qui répondent à la spécification de la méthode : l'utilisateur comprend et sait ce qu'il fait
    - Ex. : `NULL` n'est pas un tableau valide (pour une taille non-nulle). Une taille de zéro est valide.
    - EX. : `NULL` n'est pas un pointeur acceptable de BST
    - Contexte d'expert (librairie ?)

Vérification d'inputs

- Comment ?
  - A qui appartient quelle responsabilité ?
  - Librairie
    - Ne pas forcer le programme principal à s'arrêter
    - Avertir de l'erreur
      - Code d'erreur (à défaut de mieux)
      - Exception (Java, C++, Python, etc.)
  - (G)UI
    - Ne pas s'arrêter en cas d'erreur mais avertir
      - Message de feedback
    - S'arrêter en cas d'erreur
      - Assertion, exception, exit_failure, …
      - Message de feedback aussi !

Fonctions auxiliaires

- Les fonctions doivent être commentées
  - Au niveau du prototype
  - Travail en collaboration, relecture personnelle, relecture par un tiers, …
- Mot clé "`static`" ***préférable***
  - Encapsulation (bonne pratique, pas de risque de confusions)
  - Pas de problème au *linkage*

size_t

- Quand ?
  - Taille/Longueur/capacité
  - Indice (par extension)
- Pourquoi ?
  - Plus clair quand on lit le code
  - Plus grande capacité (`unsigned`)
  - /!\ aux “wrap-arounds”

```
size_t length = 0;
int i = (int)(length - 1); //i ≠ -1
```

Gestion de la mémoire

- C99 :
  - `int Array[length];`
  - Allocation sur la pile d'un tableau dynamique (disparait à la fin de la scope)
    - Pas de fuite de mémoire
    - Mais impossible de le faire sortir de la fonction
- Sinon, `malloc` / `calloc`
  - Vérifier que l'allocation s'est bien passée (≠ `NULL`)
    - → Eviter une erreur de segmentation
  - Libérer ***toute*** la mémoire

Gestion de la mémoire

- Valgrind est ton ami
  - Sorties de tableaux (ex. : `if(array[i] > k && i < length)`)
  - Fuites de mémoire
  - Problèmes d'initialisation
- Compilation avec l'option "`-g`"

```
gcc tester.c NewSort.c Array.c --std=c99 --pedantic -
Wall -Wextra -Wmissing-prototypes -g -o test

valgrind ./test
```

Debugging

- GDB (compiler avec l'option `-g`)
  - Très puissant (au moins autant qu'un debuggeur intégré dans un IDE)
  - Mais pas *user-friendly...*
- `Printf`
  - Facile
  - Moins puissant
  - Risque de laisser des `printf` ou un `#include<stdio>`
    - Conseil : `#include<stdio>//TODO Remove`

      `printf("foo"); //TODO Remove`

Test du code

- Ecrire des scripts de tests
  - Se baser sur l'interface des fonctions
    - Possibilité d'écrire les tests avant le code !
  - Tester
    - Les cas limites (ex. : tableau vide ou un seul élément)
    - Les cas normaux (ex. : tailles 10, 1000, 1000000)
    - Sur des données aléatoires
    - /!\ Aucune garantie mais moins d'incertitude