

Tests unitaires démo pratique

1 - Un exemple avec des tests

Considérez l'exemple d'une structure de données "file" (ou *queue* en anglais), tel que les éléments les premiers entrés sont aussi les premiers sortis (First In, First Out) :

<https://git.vainsta.fr/share/queue>.

Ce petit projet se compose de plusieurs fichiers, dont voici une brève description :

- le module *queue* (`queue.c` + `queue.h`)
- un exemple d'utilisation de la queue (`sample.c`)
- un fichier de tests du module *queue* (`test_queue.c`)
- le fichier `CMakeLists.txt` pour compiler ce projet

La compilation du projet et l'exécution des tests se fait de la manière suivante :

```
$ mkdir build ; cd build
$ cmake .. ; make      # compilation
$ make test           # lancement des tests
```

La commande `make test` va lancer l'exécution de tous les tests définis dans `CMakeLists.txt` et qui sont implémentés dans le fichier `test_queue.c`. La fonction `main()` prend en paramètre le nom du test à exécuter (*init_free*, *push*, *pop*, *length*, *empty*) et appelle la fonction de test associée.

Ainsi, l'exécution de la commande ci-dessous exécute un test, qui a pour objectif de vérifier le code de la fonction `queue_length()` de notre module *queue* :

```
$ ./test_queue length      # execution du test "length"
$ echo $?                  # afficher le code de retour de la commande précédente
0                           # 0 => success!
```

On considère qu'un test est réussi (*success*) si et seulement si le code de retour du programme de test est égal à 0, c'est-à-dire que la fonction `main()` renvoie la valeur `EXIT_SUCCESS` (ou 0). Toutes les autres valeurs de retour correspondent à des cas d'erreur !

En résumé :

- 0 ou `EXIT_SUCCESS`, l'exécution du test s'est terminée normalement en ne détectant aucun problème.
- 1 ou `EXIT_FAILURE`, l'exécution du test s'est terminée normalement en détectant un problème.
- Des valeurs supérieures à 128 correspondent à des terminaisons anormales (programme de test tué par un signal), liés à une erreur grave comme un accès illégal à la mémoire (*Segmentation Fault*, 139), une division par zéro (*Floating Point Exception*, 136), ...

Plus l'écriture d'un test est "précise", plus il doit être facile de conclure qu'il y a un bug dans la fonction testée (et non pas dans une autre fonction). L'ensemble des tests doit *couvrir* toutes les fonctions de notre module et être le plus exhaustif possible !

Notez que la commande `make test` va appeler le framework *CTest* (intégré à *CMake*) qui affiche un petit rapport d'exécution de tous les tests...

```
$ make test
Running tests...
Test project /home/orel/Documents/pt2/misc/queue/build
  Start 1: test_queue_new_free
1/8 Test #1: test_queue_new_free ..... Passed    0.00 sec
  Start 2: test_queue_push_head
2/8 Test #2: test_queue_push_head ..... Passed    0.00 sec
  Start 3: test_queue_pop_head
3/8 Test #3: test_queue_pop_head ..... Passed    0.00 sec
  Start 4: test_queue_push_tail
4/8 Test #4: test_queue_push_tail ..... Passed    0.00 sec
  Start 5: test_queue_pop_tail
5/8 Test #5: test_queue_pop_tail ..... Passed    0.00 sec
  Start 6: test_queue_length
6/8 Test #6: test_queue_length ..... Passed    0.00 sec
  Start 7: test_queue_empty
7/8 Test #7: test_queue_empty ..... Passed    0.00 sec
  Start 8: test_queue_clear
8/8 Test #8: test_queue_clear ..... Passed    0.00 sec

100% tests passed, 0 tests failed out of 8

Total Test time (real) =  0.03 sec
```

2 - Annexes

Un exemple de fonction test

Voici un exemple de test (à compléter) pour la fonction `game_is_empty()` du jeu Takuzu :

```
bool test_is_empty(void)
{
    game g = game_default();
    bool test1 = game_is_empty(g, 0, 0);
    bool test2 = !game_is_empty(g, 5, 5);
    game_delete(g);
    return test1 && test2;
}
```

Revision #5

Created 17 November 2024 22:09:48 by Nicolas

Updated 31 January 2025 23:10:10 by Nicolas