

Etudiant : Théo Dubus 22008507

## TP3 C++

---

### Ajouts dans cette version

- Support des fonctions curryfiées
- Ajout des tests sur les fonctions curryfiées
- Modification de la classe `Token` (suppression de `v()` qui n'était utile que pour les `Literal`)
- Ajout des tests de curryfication dans `testValgrind.txt`

### Difficultés rencontrés

L'ajout des fonction spécialisées, dont la prise en charge de la syntaxe `pow(_1, 2)` par exemple, aurait nécessité une refonte du projet et du fonctionnement du parseur et de la mémoire des variables. Le temps étant limité avec l'approche des partiels je n'ai pas pu finir d'implémenter cette dernière fonctionnalité dans mon projet.

### Choix d'implémentation

On a choisi d'ajouter les fonction curryfiées avec une 2ème mémoire qui contient le nom de la nouvelle fonction curryfiée, le nom de la fonction originale, et les arguments déjà spécifiés. Dans le lexer, la création de nouvelles fonctions curryfiées se fait avec `handleCurryfied()`. On peut ensuite appeler les fonctions comme des fonctions classique après avoir vérifié si la fonction est de type curryfiée et après avoir ajouté les paramètres prédéfinis le cas échéant. A noter que la fonction à nombre variables d'arguments `polynome()` supporte aussi la curryfication.

### Compilation du projet

Le projet utilise le système de build CMake. Un script `buildrun.sh` permet de compiler, lancer les test et la calculatrice en 1 commande.

Le projet compile sous :

- `clang 10.0.0` sur Windows/Ubuntu
- `gcc 7.5.0` sur Ubuntu ou WSL (sous-système Windows pour Linux)

### Tests

Les tests sont regroupés dans le répertoire `\test`. J'ai choisi d'utiliser la librairie de test Googletest qui est l'une des plus populaire ( Catch, Boost.Test et Ctest sont aussi de bons candidats).

Les test sont séparés en 2 fichiers `.cpp` :

- `expressionTest.cpp` : Les tests d'analyse de tokens, créés au début du projet pour vérifier le fonctionnement de `tokensFromString`
- `programTest.cpp` : Les tests du fonctionnement du programme complet

- programme en une ligne ( addition, priorités opératoires, parenthèses )
- programme multi lignes ( variables, affichage ou non avec ; , calcul du volume d'un cylindre...)

## Fuites mémoire

Le projet a été testé avec **valgrind** pour détecter la présence ou non de 'memleaks' et aucune fuite n'a été trouvée. Un script **testValgrind.sh** permet de tester les fuites sur un programme type composé de toutes les possibilités du langage.

## Continuous integration

Le repo de ce projet est hébergé sur github.com (en privé pour éviter la copie), j'en ai donc profité pour expérimenter la continuous intégration avec **Github Actions** qui permet d'effectuer une série de test pour valider mes commit à chaque push.

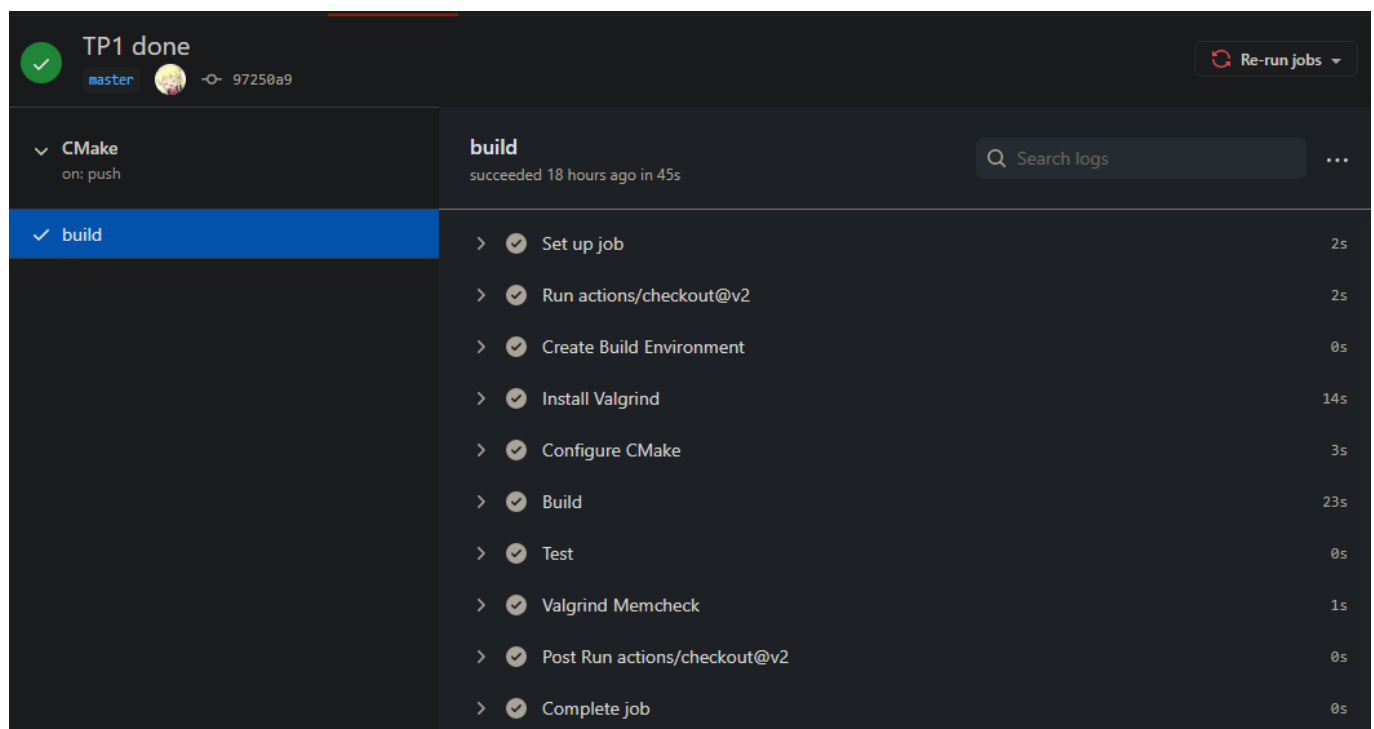
**.github/workflow/cmake.yml** contient le fichier yaml qui s'occupe de cette continuous integration.

Il y est effectué en autre:

- la compilation du projet avec CMake.
- Le lancement des tests Googletest.
- La vérification de présence des fuites mémoires avec Valgrind.

Exemples d'execution de CI : \

- Execution du "workflow" avec succès :



The screenshot shows a GitHub Actions interface for a workflow named 'TP1 done'. The workflow is in a 'succeeded' state, having completed 18 hours ago in 45 seconds. The 'build' job is selected and shows a list of steps, all of which were successful. The steps include: Set up job (2s), Run actions/checkout@v2 (2s), Create Build Environment (0s), Install Valgrind (14s), Configure CMake (3s), Build (23s), Test (0s), Valgrind Memcheck (1s), Post Run actions/checkout@v2 (0s), and Complete job (0s). A 'Re-run jobs' button is visible in the top right corner.

Step	Duration
Set up job	2s
Run actions/checkout@v2	2s
Create Build Environment	0s
Install Valgrind	14s
Configure CMake	3s
Build	23s
Test	0s
Valgrind Memcheck	1s
Post Run actions/checkout@v2	0s
Complete job	0s

- Echec de la tache Valgrind Memcheck (ce n'est qu'un exemple, le projet ne contient pas de memleaks) :

working parenthesis

master 42e2f03

Re-run jobs

▼ CMake

on: push

1

build

failed 3 days ago in 52s

Q Search logs

...

× build

▼ Valgrind Memcheck6s

23 ==3448== by 0x10A068: main (in /home/runner/work/TP\_CPP\_M1/build/src/calculator\_run)

24 ==3448== If you believe this happened as a result of a stack

25 ==3448== overflow in your program's main thread (unlikely but

26 ==3448== possible), you can try to increase the size of the

27 ==3448== main thread stack using the --main-stacksize= flag.

28 ==3448== The main thread stack size used in this run was 8388608.

29 ==3448==

30 ==3448== HEAP SUMMARY:

31 ==3448== in use at exit: 1,168 bytes in 25 blocks

32 ==3448== total heap usage: 28 allocs, 3 frees, 82,064 bytes allocated

33 ==3448==

34 ==3448== LEAK SUMMARY:

35 ==3448== definitely lost: 0 bytes in 0 blocks

36 ==3448== indirectly lost: 0 bytes in 0 blocks

37 ==3448== possibly lost: 0 bytes in 0 blocks

38 ==3448== still reachable: 1,168 bytes in 25 blocks

39 ==3448== suppressed: 0 bytes in 0 blocks

40 ==3448== Rerun with --leak-check=full to see details of leaked memory

41 ==3448==

42 ==3448== For counts of detected and suppressed errors, rerun with: -v

43 ==3448== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)

44 /home/runner/work/\_temp/b7bc3157-c4ba-4987-a3ba-9656cecb6c82.sh: line 1: 3448 Segmentation fault

45 (core dumped) valgrind ./src/calculator\_run 100/2+33-7

45 Eval:

46 Error: Process completed with exit code 139.

> Post Run actions/checkout@v21s

> Complete job0s