Professeur : Dr El Hadji Bassirou TOURÉ

COMPTE RENDU DU TP-1 : SETUP ENVIRONNEMENT DE DEVELOPPEMENT

Installation DOCKER et Configuration

Prénom : Assatou Dramé

Nom : KEITA

Classe : L3 GLSI B

Année Universitaire : 2025 - 2026

Introduction

Ce travail pratique a pour objectif de mettre en place un environnement de développement standardisé basé sur Docker, destiné à la programmation en langage C dans le cadre du cours de Programmation C Avancée.

Docker a été choisi afin de disposer d'un environnement isolé, reproductible et identique pour tous les étudiants, quel que soit leur système d'exploitation. Autrement dit, ça va nous éviter des « **Chez moi ça marche, mais pas chez toi** », ….

Avant de procéder à l'installation de Docker et à la configuration de l'environnement fourni, il est nécessaire de présenter le matériel et le contexte logiciel dans lequel ce TP a été réalisé.

1. Prérequis

Avant d'effectuer l'installation, les travaux ont été réalisés sur la configuration matérielle et logicielle suivante :

- **Processeur** : x86_64 (AMD)
- **RAM** : 16 GB
- **Espace disque (C :)** : 143 GB libres
- **Système d'Exploitation** : Windows 11 (64 - bits, avec WSL2 activé)

2. Installation de Docker (sur Windows)

Cette section décrit les étapes suivies pour installer Docker Desktop sur Windows avec WSL2.

Étape 1 : Activer WSL2

- Exécution de la commande **wsl --install** dans PowerShell en mode administrateur suivi du redémarrage du système :

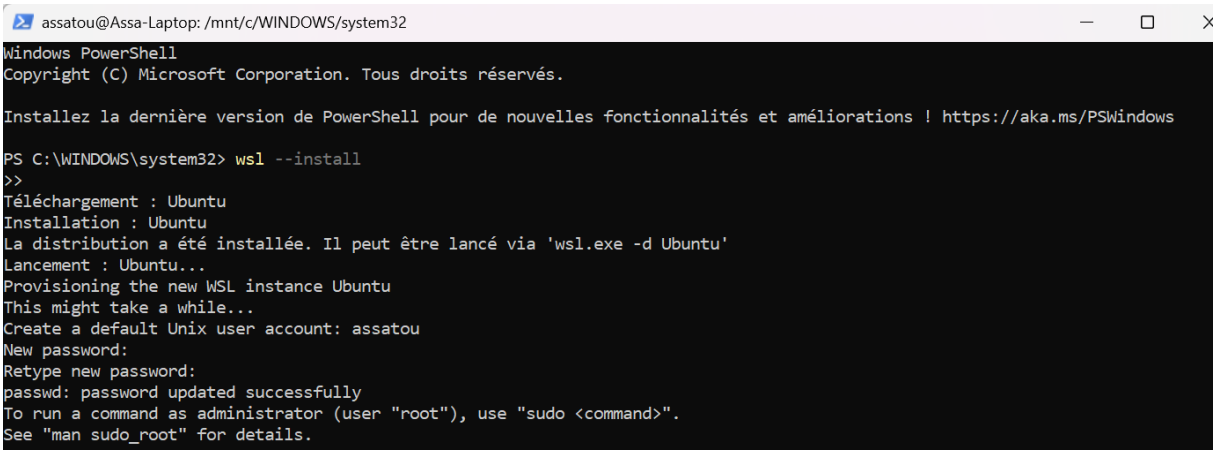


Image 1: Activation de WSL 2

Après l'exécution de la commande (*Image 1*), la distribution **Ubuntu** a été téléchargée et installée automatiquement. Le système a ensuite effectué la mise en place de la distribution, créant l'environnement Linux nécessaire et configurant un compte utilisateur par défaut (**assatou**). Le terminal affiche alors le **prompt Linux (***Image 2***)** :

Image 2: Prompt Linux

Ce prompt indique que l'utilisateur « assatou » est **connecté à Ubuntu via WSL2** et que le système est prêt à exécuter des commandes Linux.

Cette étape confirme que WSL2 (**Windows Subsystem for Linux version 2**) est correctement installé et fonctionnel sur le système Windows.

Étape 2 : Télécharger Docker Desktop

- Ouverture du navigateur Web et accès à l'adresse officielle : https://www.docker.com/products/docker-desktop/

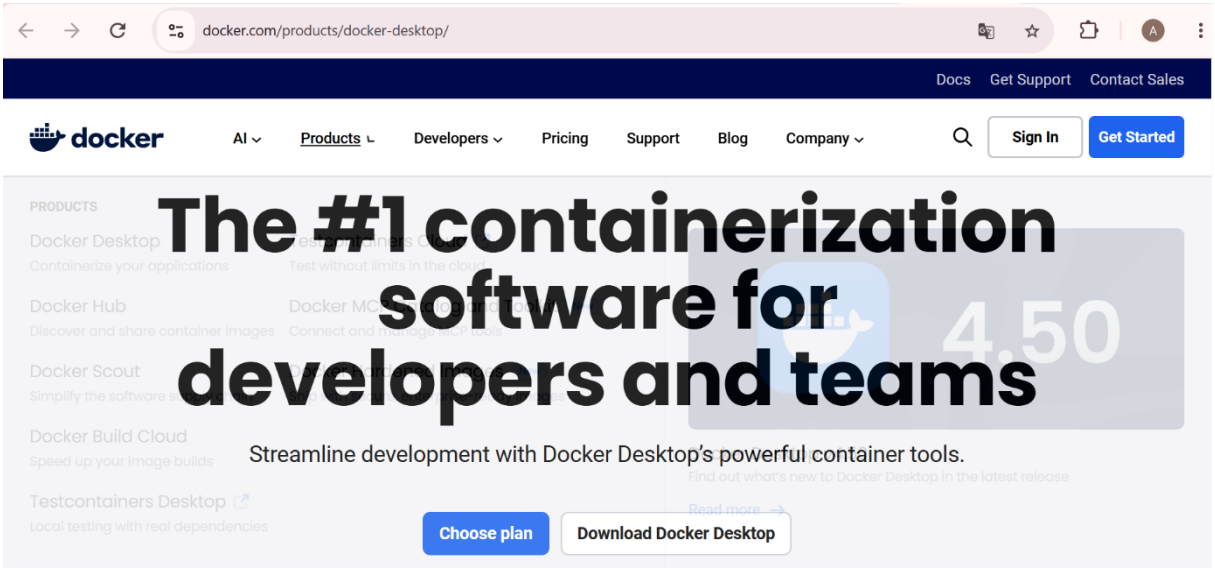


Image 3 : Page Officielle Docker Desktop

- Téléchargement de Docker Desktop en cliquant sur « **Docker Desktop for Windows** » (**AMD** dans mon cas vu que c'est mon type de processeur) :

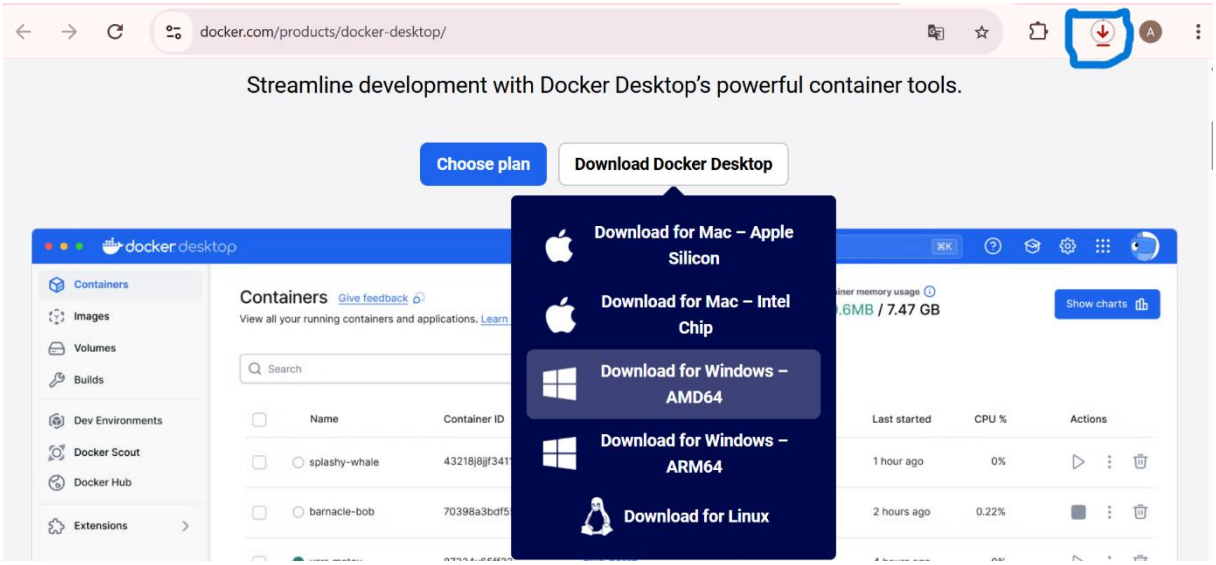


Image 4 : Téléchargement Docker Desktop

Étape 3 : Installer Docker Desktop

- Exécution du fichier d'installation (Image 5) précédemment téléchargé en faisant un double-clic :

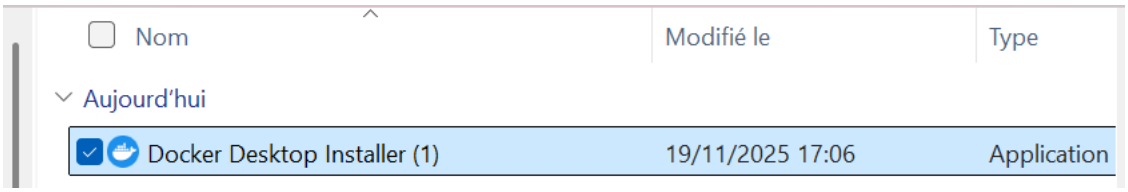


Image 5 : Fichier d'installation

- Suivi des instructions de configuration (*Image 6*) en cochant l'option "**Use WSL 2 instead of Hyper-V**" () :

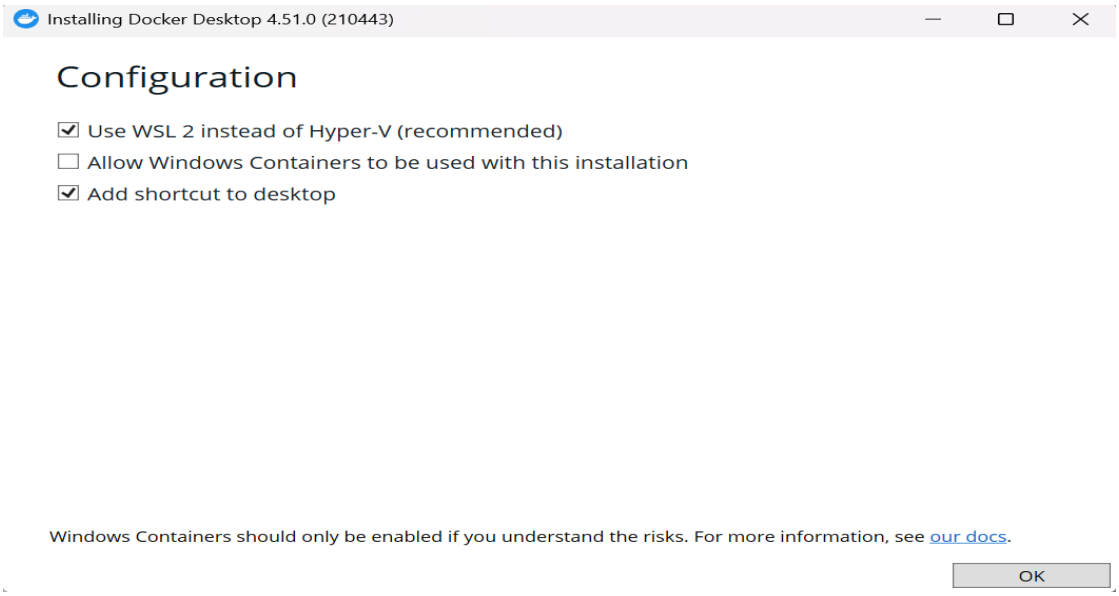


Image 6 : Configuration avant l'installation

Après la configuration, on clique sur le bouton « **OK** » pour finaliser l'installation (*Image 7*) puis on obtient :

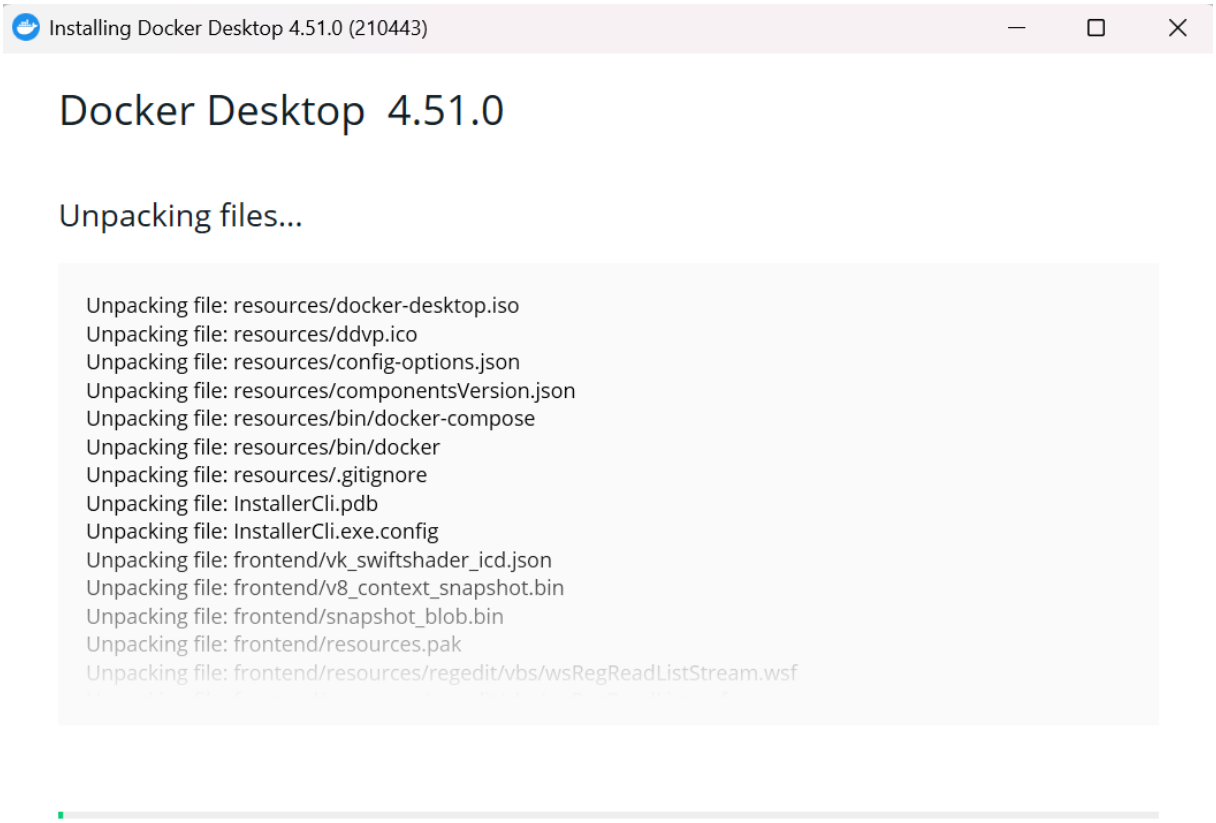


Image 7 : Dernière étape avant installation

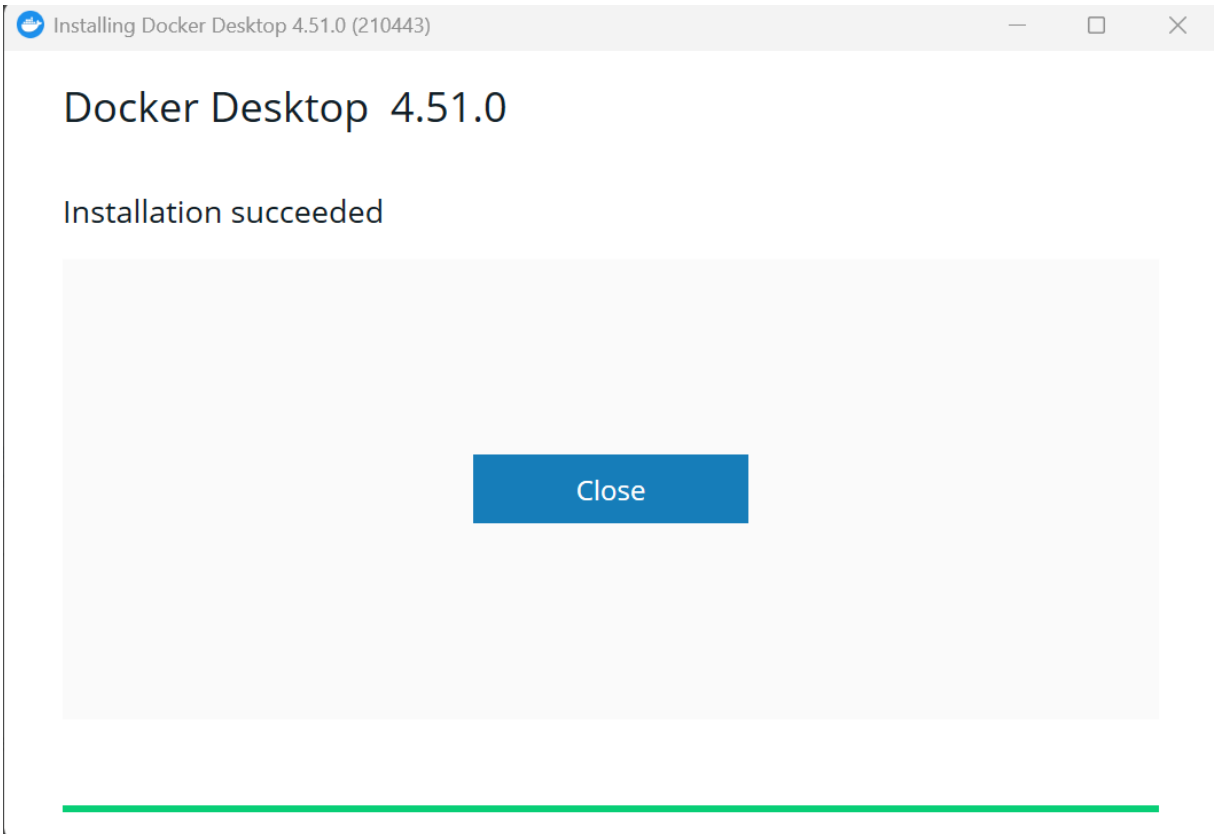


Image 8 : Après installation

Après cette étape (*Image 8*), on doit **redémarrer le système**.

L'installation de **Docker Desktop** étant achevée et l'**option WSL2** ayant été activée, il est nécessaire de procéder à une vérification afin de s'assurer que Docker est correctement configuré et opérationnel.

Étape 4 : Vérifier l'installation de Docker

- Ouverture du PowerShell et exécution des deux commandes :
  - **docker --version** permet d'afficher la version de Docker installée sur le système, ce qui confirme que le client Docker est correctement reconnu par Windows :

Image 9 : Version Docker installée

  - **docker run hello-world** permet d'exécuter un conteneur de test fourni par Docker. Lorsque le moteur Docker est actif, cette commande télécharge l'image *hello-world* et affiche un message de bienvenue :

Image 10 : Vérification du succès de l'installation

Après l'exécution de cette dernière commande, on constate l'affichage du message surligné (*Image 10*), qui constitue **le message de bienvenue de Docker** spécifiant ainsi que l'installation a réussi.

Observation lors de la première exécution

Lors de l'exécution de la commande « **docker run hello-world** », une erreur a été affichée indiquant que le client Docker ne parvenait pas à se connecter au moteur Docker (erreur surlignée dans l'*Image 11*) :

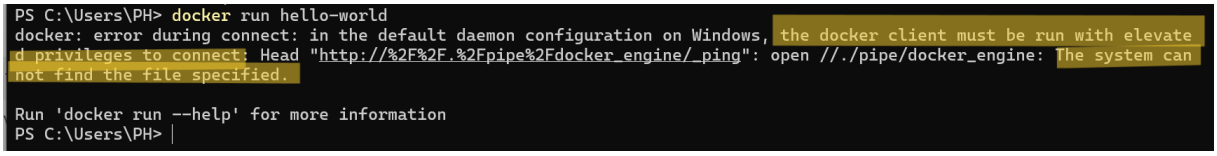


Image 11 : Erreur Docker_run lors de la vérification

Cette situation était due au fait que le moteur Docker n'était pas encore démarré. Après le **lancement manuel** de Docker Desktop, le moteur s'est initialisé correctement et la commande a pu être exécutée avec succès.

Remarque :

Toutes les commandes Docker nécessitent que Docker Desktop soit lancé et que le moteur Docker soit actif. En cas d'arrêt ou après un redémarrage de Windows, Docker Desktop doit être ouvert avant toute utilisation des commandes Docker.

Cette étape marque la fin de l'installation et la validation de l'environnement Docker sous Windows avec WSL2.

3. Téléchargement de l'image du cours

- Le téléchargement de l'image s'effectue via la commande :
  docker pull elbachirtoure/c-advanced-course:v1.0

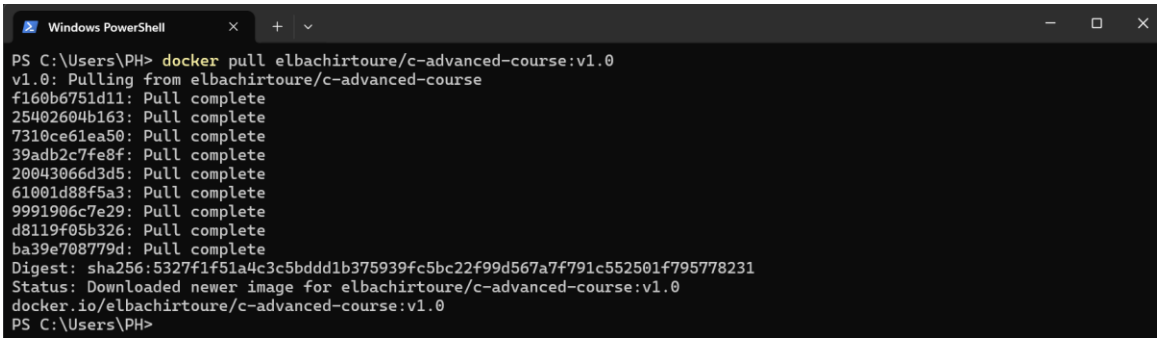


Image 12 : Affichage du téléchargement des couches Docker ("Pull complete")

- Exécution de la commande **docker images** pour vérifier que l'image est bien enregistrée localement :

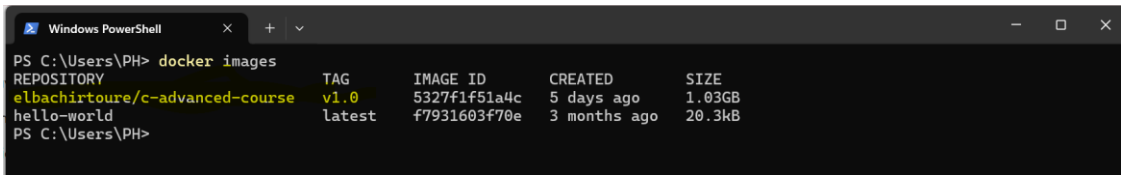


Image 13 : Affichage de la liste des images Docker, montrant "elbachirtoure/c-advanced-course v1.0"

L'image **elbachirtoure/c-advanced-course**, avec le tag **v1.0**, apparaît alors dans la liste des images disponibles donc le téléchargement s'est déroulé correctement.

4. Utilisation de Base

- **Démarrage du Container :** Ouverture du PowerShell dans le répertoire où on veut travailler (notre workspace), puis exécution de la commande (Sous Windows) :
docker run -it -v ${PWD}:/workspace elbachirtoure/c-advanced-course:v1.0

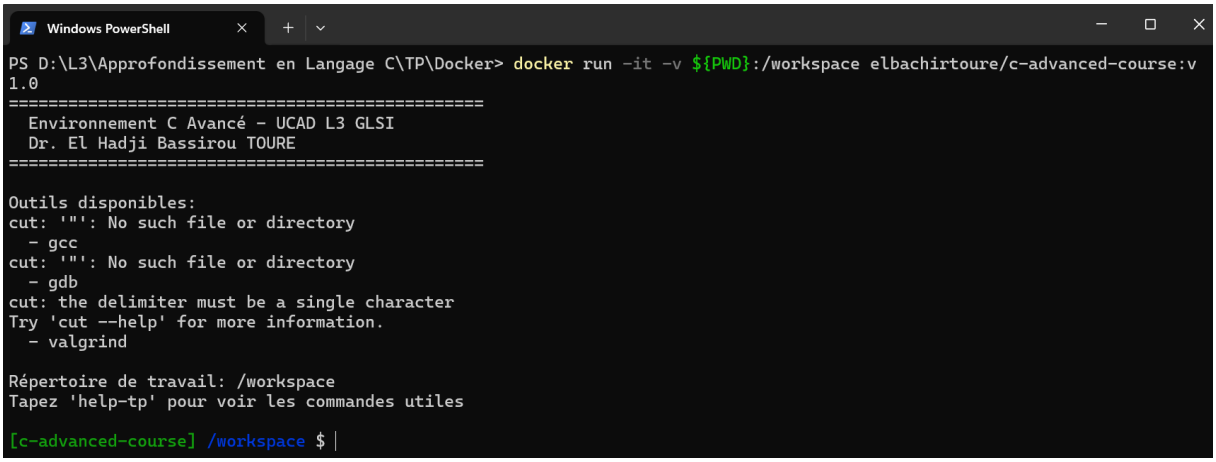


Image 14 : Invite Linux du Conteneur

- **Navigation dans le Container:**
  - Accéder au workspace (fichiers partagés) avec la commande : **cd /workspace**

Image 15 : Accès Workspace

Remarque : Le conteneur du cours s'ouvre directement dans le répertoire /workspace, qui correspond au dossier Windows monté via **l'option -v**. Il n'est donc pas nécessaire de se déplacer dans ce répertoire avec cd /workspace.

  - Lister les fichiers présents avec la commande : **ls -la**

Image 16 : Fichiers Présents

- Afficher le répertoire courant avec la commande : **pwd**

Image 17 : Affichage répertoire courant

- Créer un répertoire avec la commande : **mkdir mon_projet** (nom du projet)

Image 18 : Création d'un répertoire

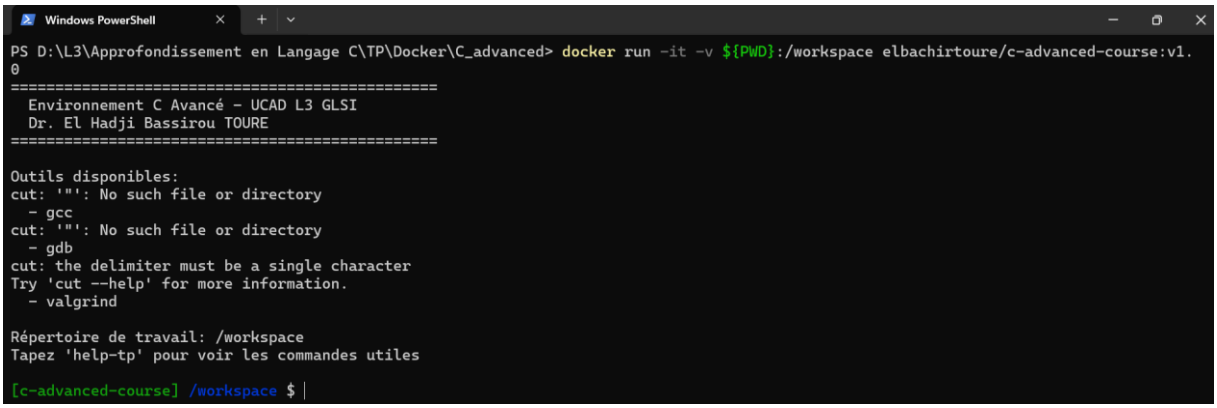
- Quitter le container avec la commande : **exit**

Image 19 : Départ du Container

5. Prise en main de Vim

Avant de commencer, on doit nous assurer que notre conteneur est lancé depuis notre répertoire de travail avec la commande :

docker run -it -v ${PWD}:/workspace elbachirtoure/c-advanced-course:v1.0



<u>Exercice 1</u> : Premiers pas avec Vim

L'objectif de cet exercice est de créer et d'éditer un fichier C avec les opérations de base.

Étape 1 : Créer un fichier dans le container

On exécute la commande **vim hello.c** et on obtient le résultat :

Image 20: Terminal après création du fichier

Vim est en mode **NORMAL** donc on ne peut pas encore saisir du texte.

Étape 2 : Passer en mode INSERT pour taper du code

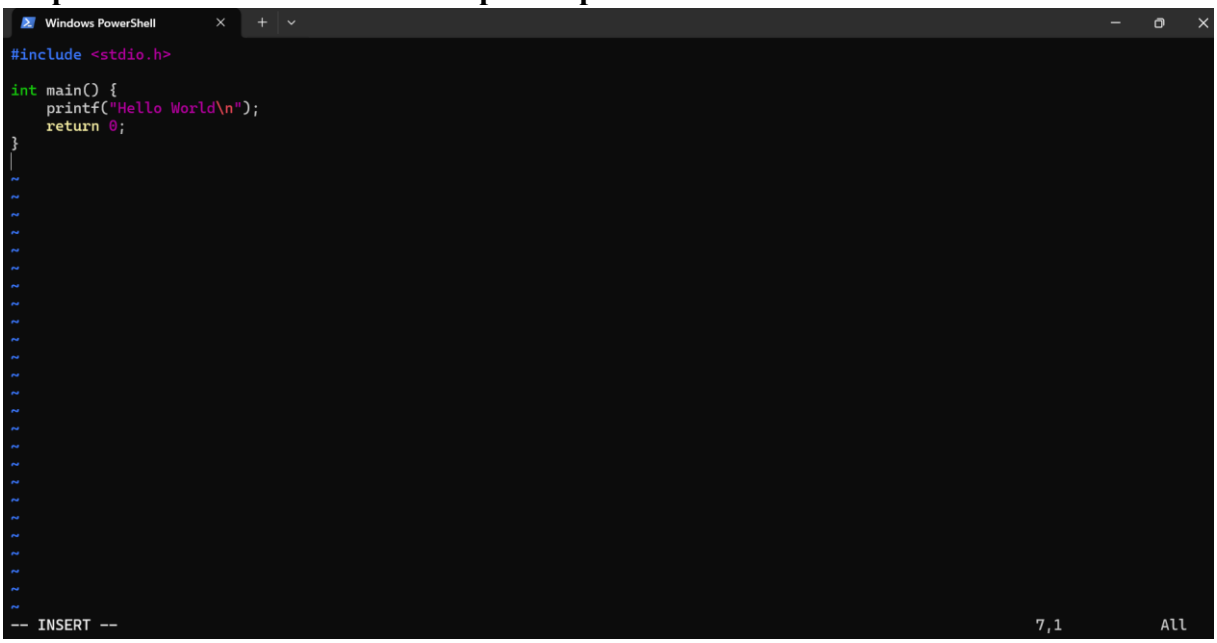


Image 21: Taper du code avec Vim

Étape 3 : Sauvegarder et quitter

On appuie sur **ESC** pour revenir au mode NORMAL puis taper :wq et enfin la touche **« ENTREE »**

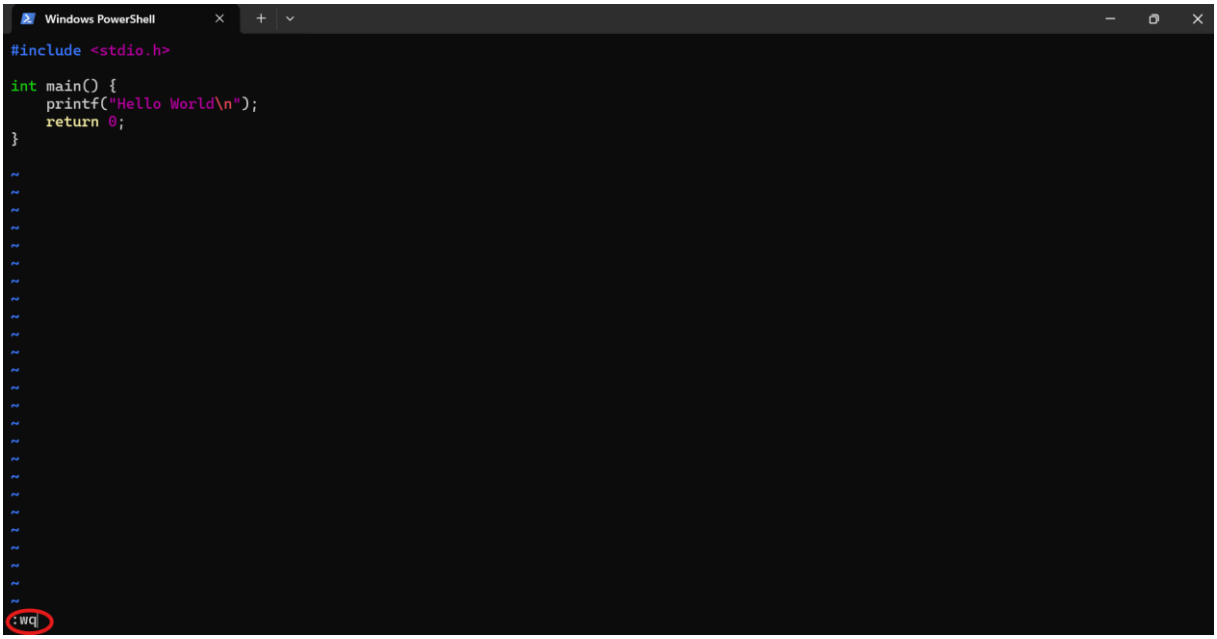


Image 22 : Sauvegarde et Exit

Étape 4 : Vérifier que le fichier existe

```
[c-advanced-course] /workspace $ cat hello.c
#include <stdio.h>

int main() {
    printf("Hello World\n");
    return 0;
}

[c-advanced-course] /workspace $ gcc hello.c -o hello
[c-advanced-course] /workspace $ ./hello
Hello World
[c-advanced-course] /workspace $
```

Image 23 : Vérification de l'existence du fichier "hello.c"

L'affichage du message **« Hello World »** nous permet de confirmer que le fichier existe bel et bien.

<u>Exercice 2</u> : Édition et Navigation

L'objectif de cet exercice est de modifier le fichier déjà existant **hello.c** et de naviguer dans le code.

Étape 1 : On réouvre le fichier avec la commande « vim hello.c » (Voir fichier : *Image 21*)

Étape 2 : Navigation (en mode NORMAL)

On doit placer le curseur sur la ligne **printf** en utilisant des lettres.

On peut procéder comme suit :

Sachant qu'on est à la première ligne après le code complet, on appuie sur **« k » trois fois** (monter) pour atterrir sur la ligne du printf (**le curseur** va alors se trouver exactement entre le **« t » et le « f »** du printf), enfin on clique sur **« h » cinq fois** pour se déplacer vers la gauche, au début de la ligne contenant le « printf ».

Étape 3 : Ajouter une nouvelle ligne

Image 24 : Ajout nouvelle ligne

Étape 4 : Copier-coller une ligne

Image 25 : Copie colle

Étape 5 : Supprimer une ligne

On va supprimer le « **return 0 ;** » de notre code :

Image 26 : Ligne supprimée

Étape 6 : Annuler et Refaire

Nous allons annuler la suppression de la ligne ci-dessus (*Image 26*) en cliquant sur « **u** » puis nous allons la resupprimer en faisant « **ctrl + r** ».

Image 27 : Annulation de la suppression du ligne

On constate qu'on a «**1 more line** » ce qui veut dire que la ligne qui a été supprimé a été rajouté par la suite.

Image 28 : Re suppression de la ligne

On resupprime et on a «**1 line less** ».

Étape 7 : Corriger l'indentation

On va au début de notre fichier (**contient 8 lignes**) en cliquant **gg** puis **=G** pour tout indenter ; on obtient :

Image 29 : Fichier indenté

Étape 8 : Sauvegarder et tester

On va compiler et exécuter le code ci-dessus (*Image 25*) :

Image 30 : Compilation et Exécution du code

<u>Exercice 3</u> : Recherche et Remplacement

L'objectif de cet exercice est de pouvoir rechercher et modifier du texte rapidement.

Étape 1 : Créer un fichier avec du code à modifier en exécutant la commande vim calcul.c

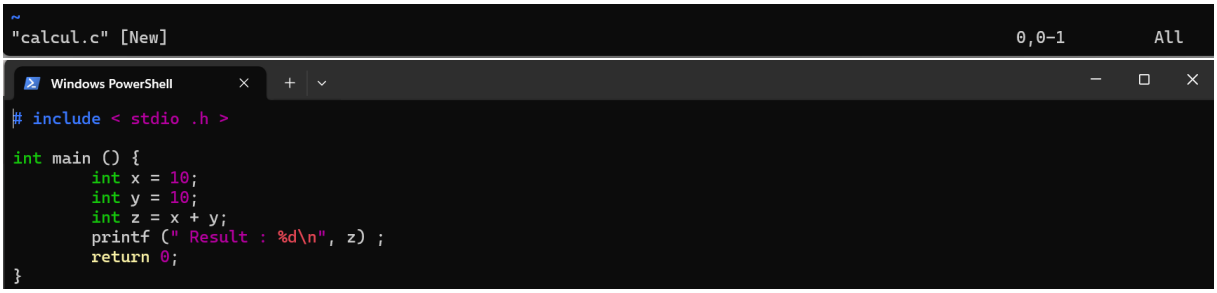


Image 31 : Nouveau fichier créé

Étape 2 : Rouvrir le fichier créé et rechercher ce qui est demandé

- Nous allons taper **/int** puis **ENTER** pour chercher **"int" :**

Image 32 : Après recherche de la première occurrence du mot "int"

- Ensuite lorsqu'on clique sur « **n** », ça nous ramène à l'occurrence suivante et « **N** » on revient à la dernière occurrence.

Étape 3 : Remplacer toutes les occurrences de "10" par "20"

- On tape en mode **NORMAL, :%s/10/20/g** puis ENTER

Image 33 : Remplacement de toutes les occurrences de "10" par "20"

Étape 4 : Sauvegarder et tester

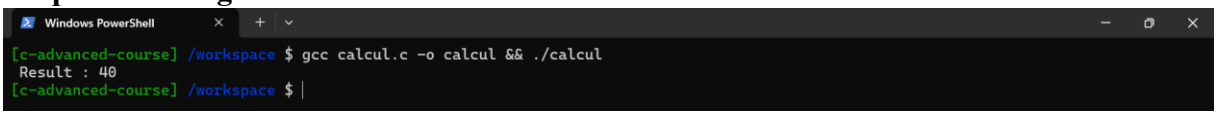


Image 34 : Test du Code

6. Premier programme C

Exercice 1 : Hello World

L'objectif de cet exercice est de compiler et d'exécuter un programme C dans le container.

Étape 1 : Sur notre système (pas dans Docker), nous allons créer un répertoire pour le TP

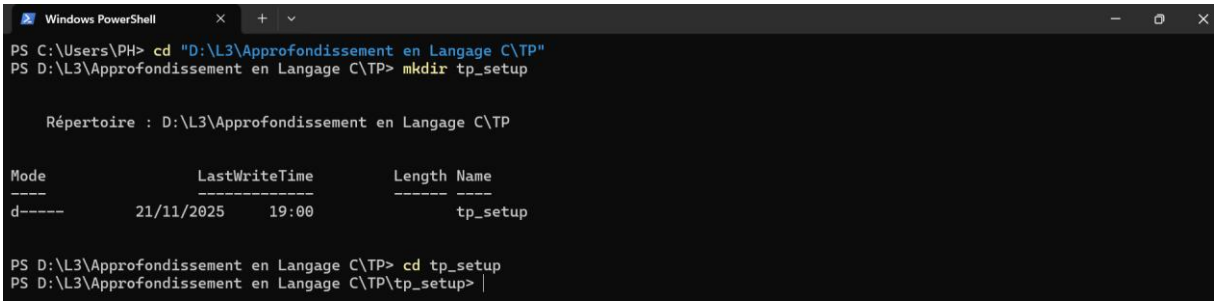


Image 35 : Création du répertoire de notre TP

Étape 2 : Nous allons créer un fichier « hello.c » avec votre éditeur préféré (Visual Studio Code pour moi)

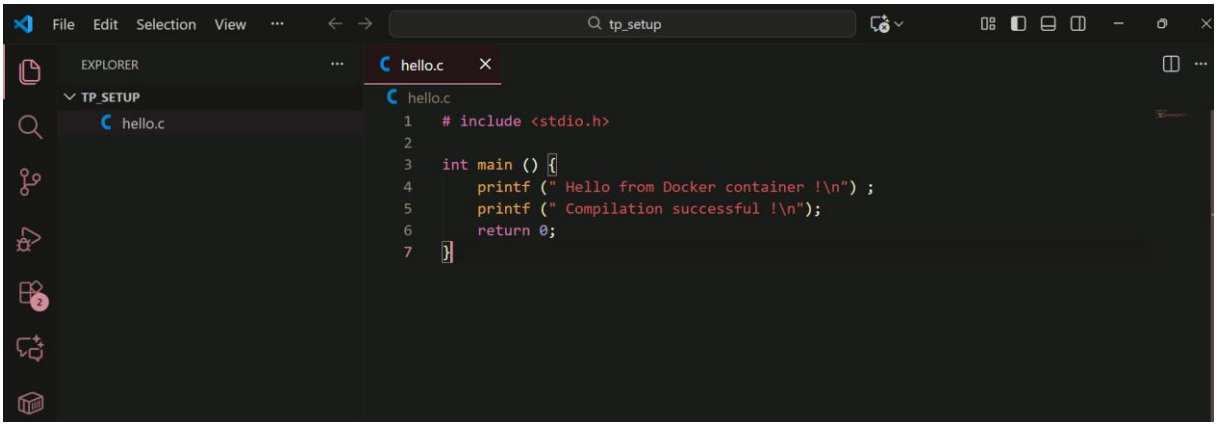


Image 36 : Création fichier dans VSCode

Étape 3 : Nous allons lancer le container Docker depuis le répertoire « tp_setup »

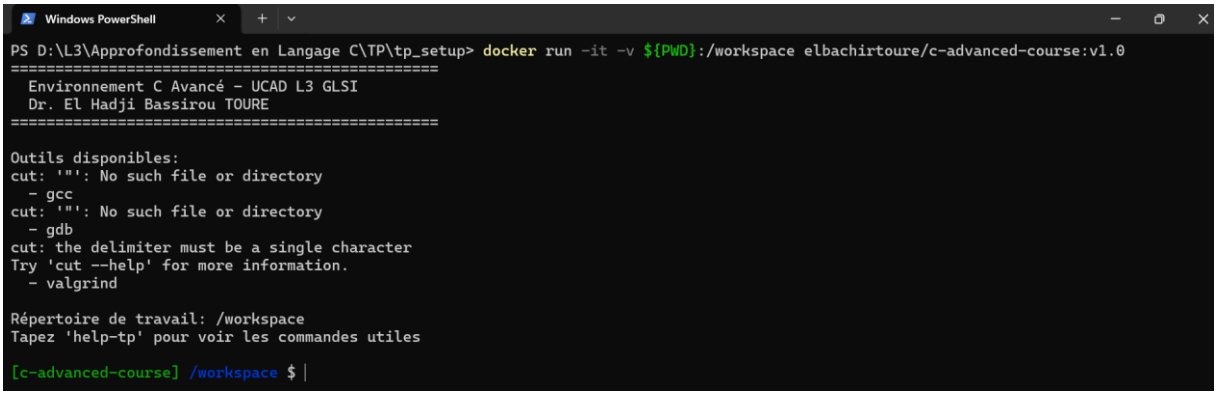


Image 37 : Lancement du container depuis notre répertoire de travail

Étape 4 : Enfin on teste le programme C

Image 38 : Résultat après compilation et exécution du programme

Exercice 2 : Vérification des outils

L'objectif de cet exercice est de tester tous les outils disponibles dans le container.

- **Compiler avec warnings stricts en utilisant la commande :**
  gcc -Wall -Wextra -pedantic -std=c11 hello.c -o hello

Image 39 : Compilation avec warnings stricts

- **Vérifier la version des outils**

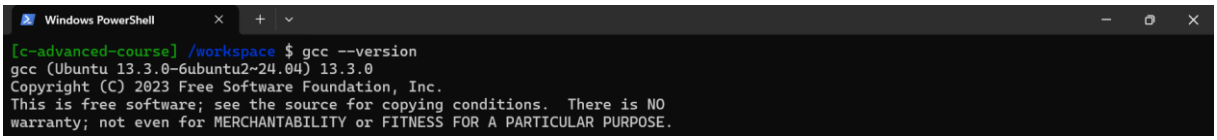


Image 40 : Version de gcc

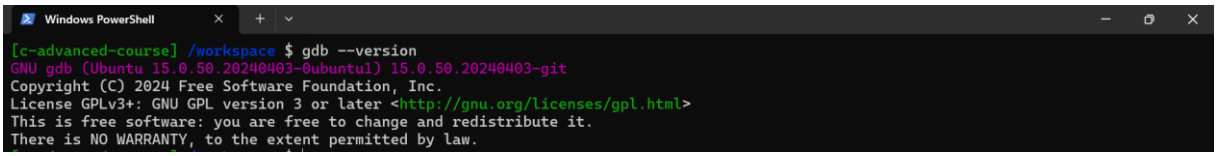


Image 41 : Version de gdb

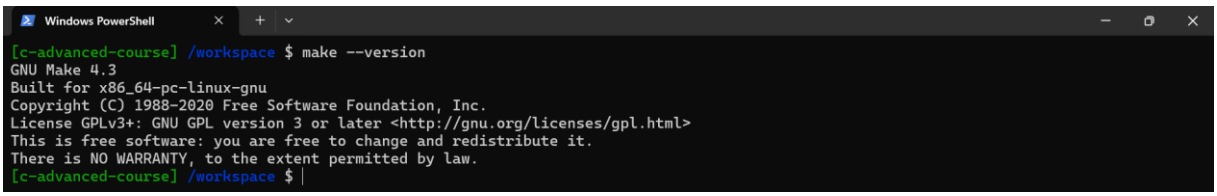


Image 42 : Version de make

Image 43 : Version de valgrind

➢ **Tester GDB**

Image 44 : Compilation avec debug

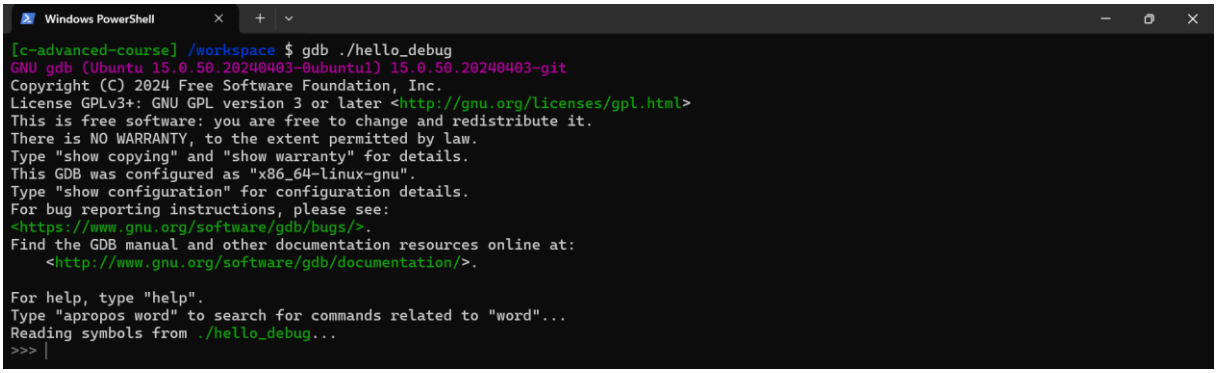


Image 45 : Lancement de GDB

Image 46 : Test dans GDB

➢ **Tester Valgrind**

Image 47: Test de Valgrind

7. Création d'un Makefile

<u>Exercice 3</u> : Makefile de base

L'objectif de cet exercice est d'automatiser la compilation avec **Make**.

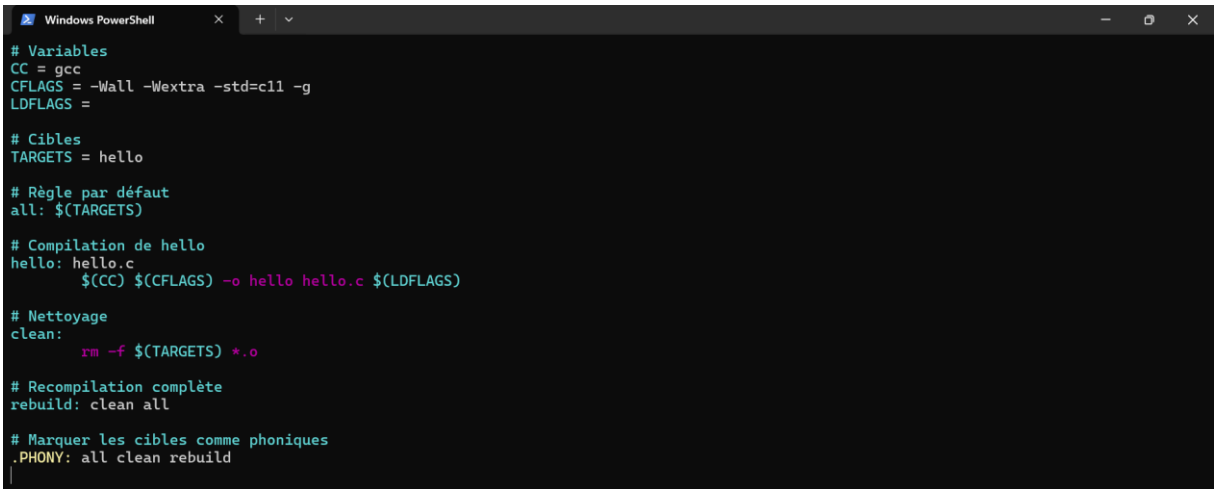


Image 48 : Création du fichier Makefile

Image 49 : Compilation automatique de "hello.c" et création de l'exécutable "hello"

Image 50 : Nettoyage du répertoire

Image 51:Recompilation de zéro

Test: Modifier **hello.c**, puis taper simplement **make**.

- Modification de **hello.c** en ajoutant printf("Test Make\n")

Image 52 : Modification du fichier

- Taper make

Image 53 : Utilisation de make pour les fichiers modifiés

Nous constatons que la commande « make » **détecte les changements et recompile** uniquement les fichiers modifiés ce qui constitue un gain de temps surtout pour les projets plus gros.

8. Validation de l'installation

Exercice 4 : Test complet

L'objectif de cet exercice est de valider que tout fonctionne correctement.

Windows PowerShell

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    printf("=== Test Environnement ===\n");

    // Test 1: Compilation
    printf("[OK] Compilation reussie\n");

    // Test 2: Allocation memoire
    int *ptr = malloc(10 * sizeof(int));
    if (ptr != NULL) {
        printf("[OK] malloc fonctionne\n");
        free(ptr);
    }

    // Test 3: Arithmetique
    int result = 42 + 27;
    if (result == 69) {
        printf("[OK] Arithmetique correcte\n");
    }

    printf("\nEnvironnement pret pour le cours !\n");
    return 0;
}
```

Image 54 : Création du programme "test_env.c"

```
[c-advanced-course] /workspace $ gcc -Wall -Wextra -g test_env.c -o test_env
[c-advanced-course] /workspace $ ./test_env
=== Test Environnement ===
[OK] Compilation reussie
[OK] malloc fonctionne
[OK] Arithmetique correcte

Environnement pret pour le cours !
[c-advanced-course] /workspace $
```

Image 55 : Compilation et Exécution

```
[c-advanced-course] /workspace $ valgrind --leak-check=full ./test_env
==73== Memcheck, a memory error detector
==73== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==73== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==73== Command: ./test_env
==73==
=== Test Environnement ===
[OK] Compilation reussie
[OK] malloc fonctionne
[OK] Arithmetique correcte

Environnement pret pour le cours !
==73==
==73== HEAP SUMMARY:
==73==     in use at exit: 0 bytes in 0 blocks
==73==   total heap usage: 2 allocs, 2 frees, 1,064 bytes allocated
==73==
==73== All heap blocks were freed -- no leaks are possible
==73==
==73== For lists of detected and suppressed errors, rerun with: -s
==73== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[c-advanced-course] /workspace $
```

Image 56 : Test des fuites de mémoire avec Valgrind