*Amine Souki*

3B7

Groupe 3

***Atelier 0 : Initiation aux Outils de Développement C sous Linux***

1. Objectifs :

* Initiation à l'utilisation de quelques outils de développement avec le langage C sous Linux.
* Développement en C sous Linux à partir d'un terminal (en mode console).

1. Prérequis :

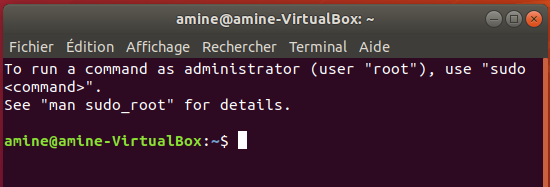
Dans cet atelier, nous allons utiliser la machine virtuelle Linux Ubuntu.

1. Outils de Développement :

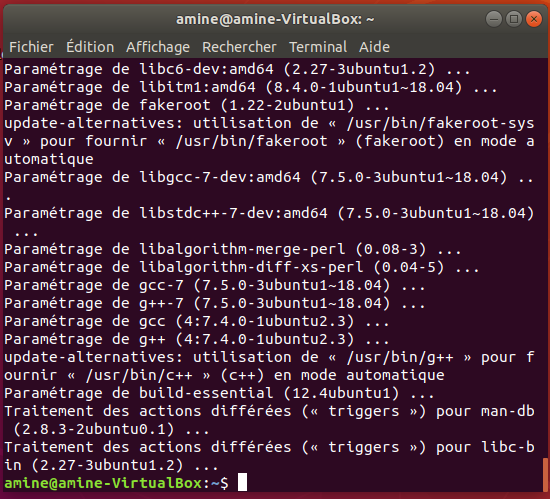
* GCC (GNU Compiler Collection) : est une suite de logiciels libres de compilation utilisé dans le monde Linux dès qu’on veut transcrire du code source en langage machine.
* Make (Gestionnaire de Compilation Make) : permet d'automatiser la phase de compilation, celle du programme mais aussi celle de la documentation associée.
* GDB (Gnu DeBugger) : est un debugger puissant dont l'interface est totalement en ligne de commande, c'est-à-dire avec une invite en texte.

1. Préparation de l'Environnement :

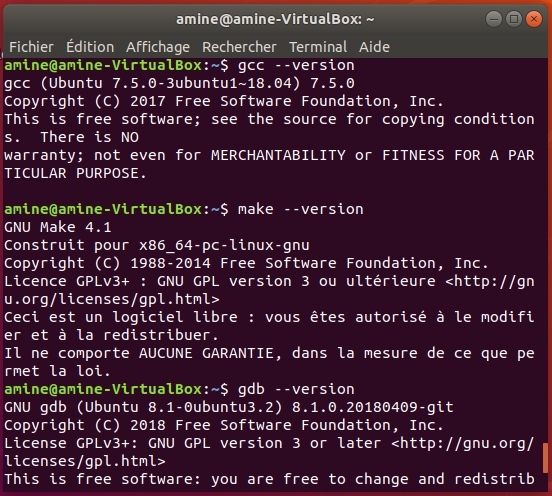
On va démarrer la machine virtuelle Linux Ubuntu et lancer un terminal (ctrl + alt + t).



Puis on va taper la commande **sudo apt-get update && sudo apt-get install build-essential gdb** pour installer les outils GCC, Make et GDB.



Après ça, on doit vérifier l’existence des outils GCC, Make et GDB dans le système d'exploitation à l’aide des commandes suivantes :

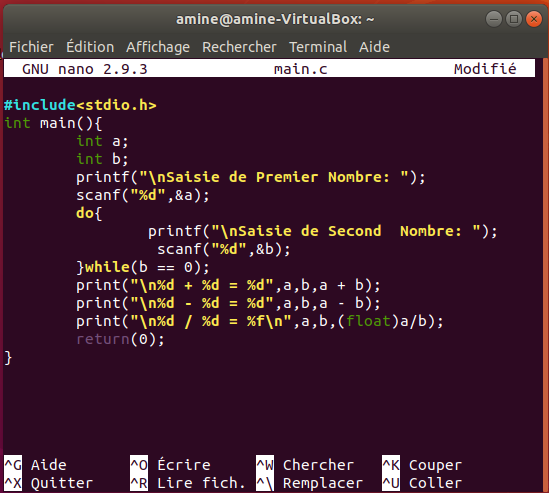
* gcc - - version
* make - - version
* gdb - - version

1. Développement C sous Linux :
   1. Édition du Code Source :

Pour installer l’éditeur nano, on taper dans un terminal la commande **sudo apt-get install nano.**

Pour éditer un fichier main.c on utilise la commande **nano main.c (sinon gedit main.c).**

On va écrire le code suivant :



Dès que terminer l’écriture de notre code, on va taper (ctrl + o) pour enregistrer et (ctrl + x) pour quitter l’éditeur.

* 1. Compilation du Code Source :

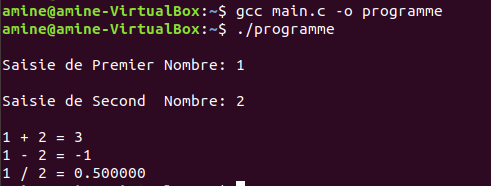
Pour compiler le fichier main.c, on va taper dans le terminal la commande **gcc main.c -o programme.**

* 1. Exécution du Fichier Binaire :

Or main.c est le fichier source et programme est le binaire (exécutable) résultant de l'opération de compilation.

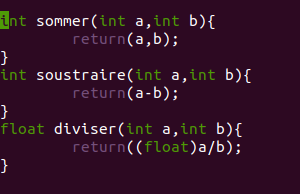
Afin de lancer l'exécution du fichier binaire issu de la compilation du code source main.c, taper dans le même terminal la commande

**./programme.**

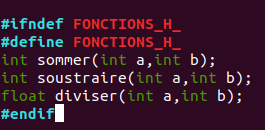


1. Compilation de Plusieurs Fichiers Sources :

On va créer deux nouveaux fichiers nommés fonctions.c et fonctions.h avec la commande **touch fonctions.c fonctions.h.** Puis Éditer le fichier fonctions.c et copier dedans le code suivant :



Après ça, on va éditer le fichier fonctions.h et copier dedans les prototypes des fonctions implémentés dans fonctions.c :



On ajouter **#include ''fontions.h''** dans **main.c** et évoquer les fonctions de ce module, puis on va compiler chaque module à part:

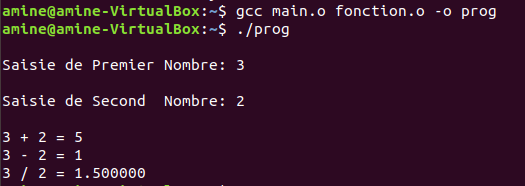


**-c** : permet de compiler le fichier en question sans passer à la phase d'édition de lien.

**-ls :** lister le contenu du répertoire courant.

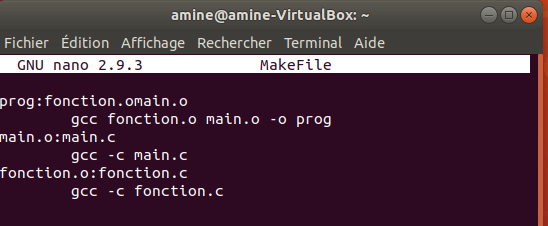
Après la compilation, pour lier les fichiers objets en un seul fichier binaire exécutable (prog) on utilise la commande **gcc main.o fonctions.o -o prog.**

Enfin on peut exécuter le binaire résultant avec la commande **./prog.**



1. Automatisation de la Compilation sous Linux :
   1. Création du Fichier MakeFile :

Dans le même terminal, on va créer un fichier **MakeFile**, puis on va éditer les règles suivantes dans le même fichier :

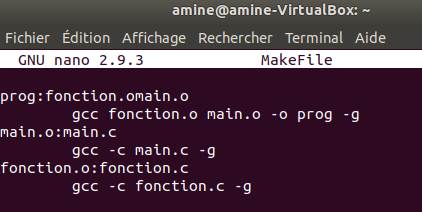


On doit enregistrer le fichier **MakeFile** et quitter nano **(ctrl + o et ctrl + x)** puis on va compiler le projet en invoquant l'outil **Make** et enfin on peut Exécuter la cible prog avec la commande **./prog.**

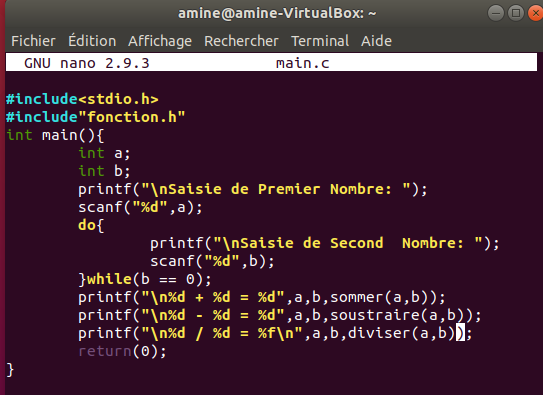
1. Exécution et Débogage :

Dans cette partie on va éditer à nouveau le fichier **MakeFile** avec la commande **nano MakeFile** après, on ajouter l'option **-g** à chaque fois qu’on évoque la commande **gcc**, puis on doit enregistrer le fichier

**MakeFile** et quitter :

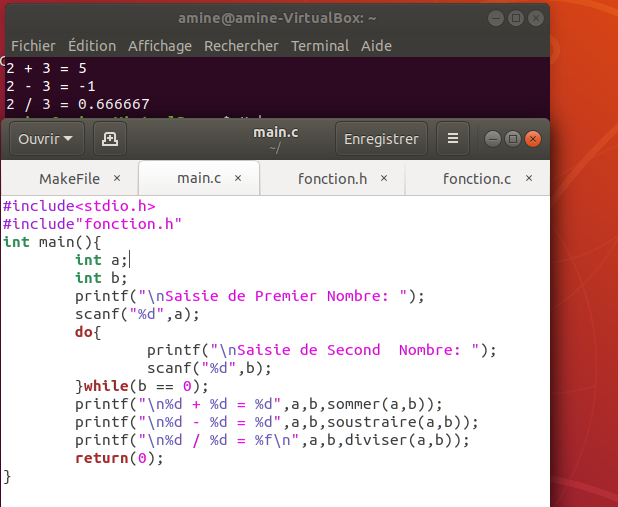


Maintenant, on va rééditer le fichier **main.c** et remplacer par ce code qui présente un bug lors de l’exécution.



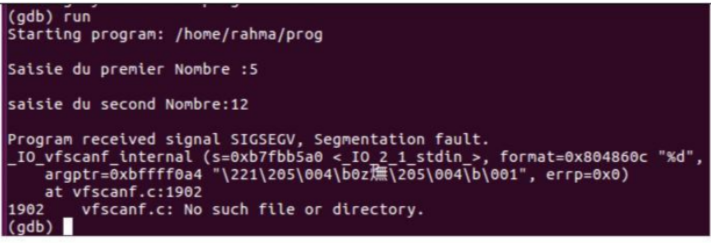
Quand on recompiler à nouveau le projet avec la commande **make,** on remarque que la compilation est faite sans probléme.

Après on va exécuter le binaire résultant prog avec la commande **./prog.**

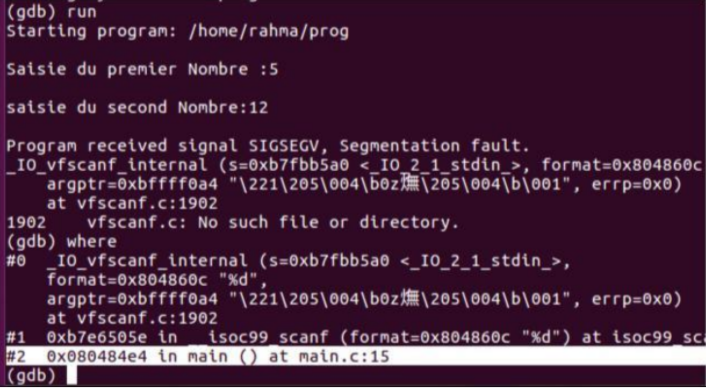


La seule façon de trouver les erreurs en cours d'exécution est de débogueur le binaire grâce au débuggeur GDB à travers l’utilisation de commande **gdb prog**.

Pour afficher les informations sur les erreurs, on doit taper la commande **run** ou bien **r.**



Les erreurs ne sont pas précises, donc on peut faire mieux en localisant la ligne causant l'erreur dans le code, il suffit de taper la commande **where** dans le prompt du **GDB** :



*Les Breakpoints :*

Les breakpoints ce un point d’arrêt permet d’arrêter l’exécution de code dans un certain ligne.

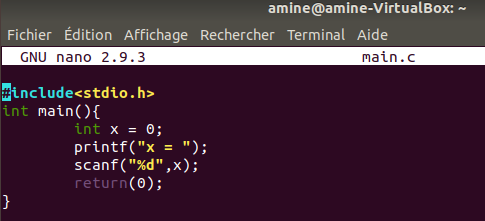
*Exemple :*

**(gdb) break 3**: Permettra d'arrêter l'exécution à la ligne 3.

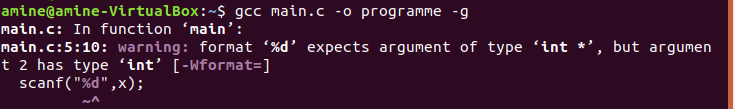
**(gdb) b Somme: 6** : Permet de faire un breakpoint dans la fonction Somme, à la ligne 6.

*Exemple d’Utilisation des Breakpoints:*

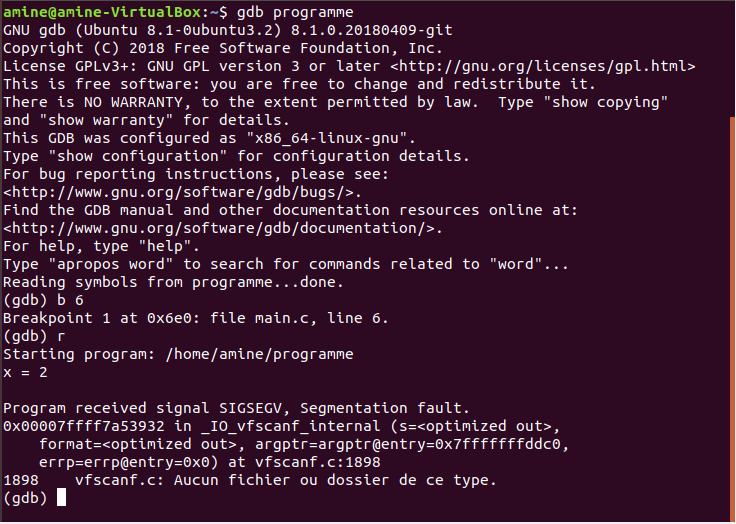
* On taper sur le terminal **nano main.c,** puis on va creé le code suivant.



* Puis on va taper la commande **gcc main.c –o programme –g :**



* Ensuite, on va exécuter sur le GDB de programme avec la commande **gdb programme**, puis on va taper les commandes suivantes d’une maniéré successive **(gdb) b 6** et **(gdb) r**.



*Examiner la situation lors d’un Breakpoint:*

On peut examiner l’état de la mémoire à l’aide de la commande **print** : **(gdb) p x.**

Aussi, on peut remémorer l’endroit dans le code où l’exécution a été interrompue grâce à l’utilisation de la commande **list.**

*Effacer un Breakpoint:*

Pour indiquant un numéro de ligne ou un nom de fonction permet d'effacer le breakpoint, on doit utiliser la commande **clear.**



La commande **delete** permet d'effacer un breakpoint en indiquant son numéro.

Pour quitter, on va taper la commande **quit.**

