Workshop GDB :
À la découverte de GDB, un outil de débogage puissant !

Plan du workshop :

- Présentation du PowerPoint (1ère partie)
- Expérimentation des premières commandes

Installation de gdb et clonage du dépôt (si ce n'est pas déjà fait bien sûr).

Dans un premier temps nous allons cloner le dépôt pour avoir tous les fichiers destinés à la bonne réalisation de ce workshop :

- **git clone git@github.com:lbarreteau/workshop-gdb.git**

Une fois les fichiers récupérer il faut vérifier si gdb est bien installer, entrez la commande suivante

- **gdb –version**

💡 Si ce n'est pas le cas suivez le guide suivant : How to Install GDB? | GDB Tutorial

Lancement de gdb et exécuter un programme

Avant de pouvoir trouver les bugs d'un programme et de lancer gdb il faut tout d'abord ajouter un flag de compilation permettant le débogage, ce flag va permettre 2 choses, premièrement de pouvoir connaitre les lignes où se trouvent les problèmes que l'on veut résoudre, et deuxièmement de pouvoir exécuter le programme étape par étape ce qui va nous être précieux par la suite.

- **Ajouter le flag de débogage (-g)**

Une fois ce flag de compilation ajouté, nous allons pouvoir lancer gdb

- **gdb**

Tips 💡 : Si vous voulez éviter que gdb vous harcèle d'informations inutiles lorsque vous le lancez, vous pouvez rajouter -silent en paramètre.

Magie ! vous êtes dans gdb

Maintenant essayons de lancer notre programme !

Pour lancer votre programme vous devez utiliser la commande run

- **run**

Aie ❌, cela n'a pas marché. Pourquoi ? Nous avons demandé à gdb de se lancer mais sans aucun binaire, cela ne peut donc pas fonctionner. Nous allons donc donner à gdb notre binaire.

- **Générer le binaire avec make**
- **file [nom du binaire]**

Nous pouvons maintenant relancer la commande

Exécuter un programme avec des arguments et rediriger les sorties

Pour ajouter des arguments au lancement de notre programme, cela fonctionne comme un binaire.

- **run [Args]**

Essayons de mettre par exemple de mettre 10 en argument

- **run 10**

Magique, ça marche !

Si l'on veut rediriger la sortie de notre programme, c'est similaire à un lancement d'un binaire en temps normal la redirection vers un fichier peut être possible comme ceci.

- Run 5 > [votre fichier]

💡 L'inverse peut être aussi possible. Si l'on souhaite envoyer des informations en argument dans le cadre d'une possible utilisation de « getline ». Il suffit juste de changer le sens du chevron.

Quitter gdb

Pour quitter gdb, c'est simple il suffit d'écrire soit exit soit q

- **quit**

Gdb va vous demander s'il faut tuer le processus en cours, cela correspond à notre lancement de programme.

- **yes**

Live coding sur les breaks points

Expérimentation des commandes d'affichage

Maintenant que vous maitrisez les bases et que nous pouvons nous balader dans le déroulement d'un programme. Nous allons afficher des variables, il a deux manières de faire cela :

- La première est une façon est temporaire -> **print**
- Le deuxième est définitif -> **display**

Cela pour être utiliser comme ceci :

- **[print/display] [nom_de_la_variable]**

- Exercice 1 & 2 sur les boucles infinies et segfault

- Présentation du PowerPoint (2ème partie), découverte des fonctionnalités plus poussées ainsi que conclusion du workshop

💡 Petit memo sur toutes les commandes ainsi que leurs raccourcis pour gdb

commande	raccourci	effet
run	r	lance le programme (s'arrête au prochain point d'arrêt)
continue	c	relance le programme (s'arrête au prochain point d'arrêt)
~~	~~~	~~~
break [yyy.c:]xx	b [yyy.c:]xx	place un point d'arrêt à la ligne xx du fichier yyy.c (si indiqué)
info breakpoints	info breakpoints	liste les points d'arrêts
delete [x]	d [x]	efface les points d'arrêts si pas d'argument, ou le point d'arrêt correspondant au n° x
~~	~~~	~~~
next	n	exécute une instruction (ne rentre pas dans les fonctions) peut-être suivi du nombre de ligne à exécuter
step	s	exécute une instruction (rentre potentiellement dans les fonctions)
finish	f	exécute les instructions jusqu'à la sortie de la fonction
list	l	affiche 10 lignes de code centrée sur la ligne à exécuter
until xx	u xx	exécute les instructions jusqu'à la ligne xx