Déboguage avec les outils GNU : utilisation de gdb / Eric Bachard avril 2004

Important : ceci est la première version d'un document d'accompagnement aux TP d'initiation à la programmation en C de LO22. Celui-ci est donc forcément incomplet et contient peut-être des erreurs. Des exemples viendront bientôt compléter ce mini-guide d'utilisation de gdb, afin de le rendre moins « obscur ». Si vous trouvez une ou plusieurs erreurs, merci de me le signaler.

Eric Bachard, Avril 2004

Bibliographie utilisée pour la rédaction de ce document :

[1] http://www.xxxxxxxxxxxxxxxxx : exemple d'utilisation de gdb.

[2] Programmer avec les Outils GNU de Mikes Moukides et Andy Oram, traduit de l'anglais par Manuel et Nat Makarévitch. Editions O'Reilly, Paris 1997, ISBN 2-84177-010-9

[3] Méthodologie de la programmation en C, de Jean-Pierre Braquelaire, Editions Dunond, Paris 2000, ISBN : 2 10 0047825

1) Préparatifs

Avant de pouvoir déboguer un programme **en utilisant gdb, il faut le compiler avec l'option -g** de gcc, ou **mieux encore, avec l'option -ggdb (avec optimisation pour gdb).** Une table étendue des symboles sera ainsi générée, permettant d'utiliser pleinement les possibilités de gdb

Exemple :

```
eric@alube:~/LO22/P04/C/TP_C_P04/TP1$ gcc -ggdb exo15.c
eric@alube:~/LO22/P04/C/TP_C_P04/TP1$ gdb ./a.out
GNU gdb 6.1-debian
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "powerpc-linux"...Using host libthread_db library
"/lib/libthread_db.so.1".

(gdb) list
1       #include <stdio.h>
2
3       #define TRUE                    1
4       #define FALSE                   0
5
6       int main (void)
7       {
8               const int pas = 5;
9               int ligne, colonne;
10              char operation;
(gdb)
```

N.B : help list pour en savoir plus...

Ainsi, si on oublie de compiler avec l'option -ggdb, c'est autre chose.... :

```
eric@alube:~/LO22/P04/C/TP_C_P04/TP1$ gcc exo15.c
eric@alube:~/LO22/P04/C/TP_C_P04/TP1$ gdb -q ./a.out
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) list
1       ../sysdeps/powerpc/powerpc32/elf/start.S: Aucun fichier ou répertoire de ce type.
        in ../sysdeps/powerpc/powerpc32/elf/start.S
(gdb)
```

N.B. : Il est aussi possible d'optimiser un code tout en le compilant pour gdb. Dans ce cas, les options -O et -g seront toutes deux passées lors de la compilation. En effet, ces deux options ne sont pas incompatibles avec gdb (mais peuvent l'être avec d'autres débogueurs). Le problème qui survient néanmoins, est que le code ainsi généré ne correspond plus tout à fait au code source (car optimisé, avec l'option -O).

Il est, en général, raisonnable d'optimiser les code créé **après** l'avoir correctement débogué. Dans certains cas rares, il est nécessaire d'optimiser le code à déboguer, les erreurs n'apparaissant que dans le code optimisé.

2) Lancement de gdb : **options possibles**

- Cet utilitaire peut s'éxécuter dans une console (tty--> "teletype") , et/ou un terminal (pts -->pseudo-terminal-slave), ou encore être appelé par un programme «fenêtré» du type ddd, frontend à gdb.

- La commande gdb peut être entrée sans ou avec arguments, sans ou avec options.

- Les options les plus intéressantes de gdb sont les suivantes :

-d : (pour directory)demande à gdb de rechercher les fichiers sources dans le répertoire
-x <fichier> : (pour execute). Avant d'accepter une première commande, le débogueur lit et exécute les commandes contenues dans <fichier>, ce qui permet de lancer une suite d'opérations (mise en place de points d'arrêts, initialisation de variables, ...) dès le lancement de gdb. Dans le cas où ce sont toujours les même commandes qui sont lancées, il peut être intéressant de placer ces commandes dans le fichier ressources de gdb .gdbinit de votre répertoire personnel, lu au lancement de gdb.

-nx : Abréviation de « **N**o e**X**ecute ». interdit l'exécution des commandes présentes dans le fichier de configuration .gdbinit

-q : cette option, qui veut dire « quiet » prend tout son sens après quelques lancements de gdb. La première fois que gdb se lance, gdb se présente très longuement. Apèrs quelques lancements, cela devient parfaitement inutile, et l'option -q devient alors intéressante.

-help : affiche un message informatif décrivant brièvement chaque option disponible avant de se terminer.

Le premier argument est le nom du programme à tester.

3) Guide des commandes principales

Pour mémoire, le tableau ci-dessous qui résume les commandes les plus utilisées

Abréviation	***Commande***	***Fonction***
b	break	Insère un point d'arrêt
c	continue	Reprend l'exécution après un point d'arrêt
d	delete	Supprime un point d'arrêt
f	frame	Affiche le bloc de pile courant
h	help	Affiche des informations sur une commande
i	info	Affiche des informations sur divers composants internes et externes au programme
l	list	Liste le code source
n	next	Active le mode pas à pas (chaque fonction est dans ce cas exécutée en une seule passe)
p	print	Affiche la valeur d'une variable ou d'une expression
q	quit	Termine la session gdb
r	run	Lance le programme à déboguer
s	step	Active le mode pas à pas (chaque fonction est en ce cas exécutée ligne après ligne)

N.B. : gdb est capable de compléter tous les noms de commande entrés, comme avec la complétion du shell (Bash par exemple).

Exemple :

```
eric@alube:~/LO22/P04/C/TP_C_P04/TP1$ gdb -q a.out
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) li
1       #include <stdio.h>
2
3       #define TRUE                    1
4       #define FALSE                   0
5
6       int
7       main (void)
8       {
9         const int pas = 5;
10        int ligne, colonne;
(gdb)
```

Dans cet exemple gdb a exécuté « list » avec seulement li d'entré, car aucune autre commande ne commence par li

3.1) La pose de points d'arrêts

(b)reak [line] : «b» + numéro de la ligne permet de placer un point d'arrêt à la ligne indiquée.

(b)reak [fonc] : «b» + nom de la fonction permet de placer un point d'arrêt à la fonction spécifiée.

info break : indique où sont définis les points d'arrêt.

3.2) Exécution d'un programme

run < file : lance le programme avec une redirection de l'entrée standard

(n)ext : exécute une instruction en entrant dans le code des fonctions

(c)ount : continue l'exécution du programme

jump (line) : saute à la ligne indiquée (modifie le compteur ordinal)

(l)ist : liste le code source du programme

3.3) Examen des données

(p)rint [expr] : affiche la valeur de l'expression

(p)rint [*tab@num] : affiche la valeur de l'expression après chaque arrêt

undisplay [num] : supprime un display

set [expr=value] : modifie la valeur d'une variable.