Le debugger symbolique DDD

`ddd`(data display debugger) est un outil permettant de débugger efficacement des programmes écrits en C ou C++ (entre autres). Il s'agit en fait d'une interface permettant d'appeler un debugger non graphique, tel que `gdb`.
Pour pouvoir débugger un programme à l'aide de `ddd`, il est impératif que le code source ait été compilé avec l'option -g. On lance ensuite `ddd` sur le programme `mon_programme` en tapant `ddd mon_programme`. Par exemple,

```
gcc -g -o mon_programme mon_programme.c
```

puis

```
ddd mon_programme
```

On voit alors s'ouvrir la fenêtre de la Figure 1.

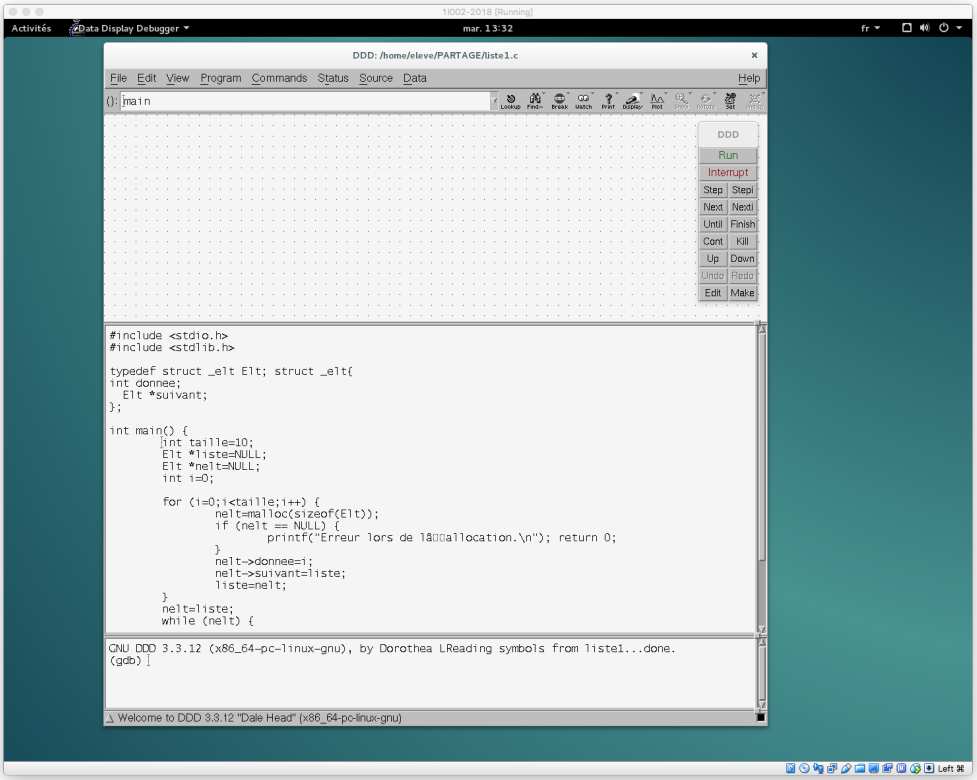

FIGURE 1 – Le debugger `ddd`

- la barre de menus contient des boutons permettant de configurer le déroulement de l'exécution. Lorsque le bouton est accompagné d'un triangle, il permet d'accéder à un sous-menu en maintenant le bouton gauche de la souris enfoncé.
- la fenêtre elle-même est divisée en trois sections : une zone d'affichage (*View* ▷ *Data Window*), la zone contenant le code source (*View* ▷ *Source Window*) et la console (*View* ▷ *GDB Console*) qui permet d'agir sur l'exécution. C'est là par exemple que l'utilisateur saisit des valeurs lorsque le programme en demande. On peut aussi y taper des commandes sous forme textuelle, ou y afficher des résultats.
- la sous-fenêtre DDD (*View* ▷ *Command Tool*) propose des boutons de commande permettant de contrôler l'exécution du programme.

Progression dans l'exécution du programme

Pour exécuter le programme avec `ddd`, il suffit de cliquer sur `Run`. Cette opération exécute le programme d'une traite : vous n'avez rien vu passer ! On définit donc des *points d'arrêt* (voir ci-dessous) dans le

programme. `Run` exécute alors le programme jusqu'au premier point d'arrêt. On peut ensuite poursuivre l'exécution de plusieurs manières :

Cont : poursuit l'exécution jusqu'au prochain point d'arrêt.

Next : exécute l'instruction suivante. Si cette instruction est un appel de fonction, la fonction est exécutée d'une traite (i.e., le debugger ne déroule pas l'exécution de la fonction pas à pas.

Step : exécute l'instruction suivante. Si cette instruction est un appel de fonction, on entre dans la fonction et on s'arrête à la première instruction de celle-ci. On peut à tout instant terminer l'exécution de la fonction en cliquant sur Finish. *NB : `Step` est incompatible avec l'exécution des fonctions de bibliothèque, type `printf` ou `malloc`, dont on ne connaît pas le code source*.

Until : exécute le programme jusqu'à atteindre une instruction dont le numéro de ligne est strictement supérieur à celui de l'instruction courante. Cette commande permet en particulier de terminer l'exécution d'une boucle.

Insertion et destruction de points d'arrêt

L'insertion d'un point d'arrêt se fait avec le bouton `Break` après avoir placé le curseur au début de la ligne où l'on veut s'arrêter. Un stop s'affiche au début de la ligne. L'exécution du programme sera interrompue avant chaque exécution de l'instruction figurant sur cette ligne. En plaçant le curseur au niveau d'un point d'arrêt existant, on remarque que le bouton `Break` du menu se transforme en `Clear`. On peut alors détruire le point d'arrêt ou, en accédant au sous-menu, le désactiver ou modifier ses caractéristiques. En particulier, on peut rendre un point d'arrêt temporaire : le programme ne sera interrompu que lors de son premier (ou prochain) passage à cet endroit. Le menu *Source* ▷ *Breakpoints* permet d'accéder à la liste des points d'arrêt du programme, ce qui se révèle utile lorsqu'on veut en modifier plusieurs.

Visualisation de la valeur des variables

Il existe plusieurs manières de visualiser la valeur d'une variable. La plus simple consiste à amener le curseur dessus : au bout d'environ 1s, la valeur de la variable s'affiche. Un clic droit sur une variable permet aussi d'accéder à un menu d'affichage, en particulier aux fonctions :

Print : afffiche la valeur de la variable dans la fenêtre *GDB Console*. Si cette variable est un pointeur, la valeur affichée est une adresse. Il faut utiliser `Print*` pour accéder à la valeur de la variable pointée, c'est-à-dire au contenu de la case mémoire.

Display : affiche dans la fenêtre *Data Window* l'évolution de la valeur de la variable, mise à jour au fur et à mesure des modifications. Comme pour `Print`, `Display*` permet d'accéder à l'objet pointé lorsque la variable est un pointeur. L'affichage peut être supprimé au moyen du bouton `Undisp` (ou par un clic droit sur l'affichage).

Visualisation d'un tableau :
Il est possible de visualiser directement le contenu d'un tableau, en sélectionnant `Display` après un clic droit sur le nom du tableau. On peut passer d'un affichage vertical à horizontal (ou l'inverse), en sélectionnant `Rotate` au moyen d'un clic droit sur l'affichage du tableau.

À l'intérieur d'une fonction, pour afficher le contenu d'un tableau passé en paramètre, on tape dans la console GDB la commande *graph display tab[prem]@nbelem*, où *tab* est le nom du tableau, *prem* l'indice du premier élément affiché et *nbelem* le nombre d'éléments à afficher.

Visualisation d'une liste chaînée :
Lorsqu'un pointeur est affiché (`Display`), on peut sélectionner ce pointeur dans la zone d'affichage et cliquer sur `Disp*`. La valeur de l'objet pointé s'afffiche alors, liée au pointeur par une flèche. Lorsque différents pointeurs référencent une même case mémoire, la "factorisation" n'est pas automatique. Le menu *Data* ▷ *Detect Aliases* permet de ramener les différentes flèches sur le même objet.

Exécution d'un programme avec arguments

L'exécution de certains programmes nécessite des arguments passés par la ligne de commande. Ces paramètres ne peuvent pas être transmis lors de l'appel à `ddd`. Ils sont transmis lors du lancement de l'exécution en accédant au menu *Program* ▷ *Run* et en entrant l'ensemble des paramètres (séparés par un espace) dans la case *Run with Arguments*.