POLYTECH LILLE

Université Lille1 Sciences et Technologies

Département IMA / 3A (S5) **Programmation Structurée 2012/2013**

http://laure.gonnord.org/pro/teaching/

TP8 - Pointeurs simples, traçage de programme avec ddd

La partie Linux - Commande find

Tester toutes les commandes dans un terminal situé à la racine de votre compte.
Pour chercher un fichier sur une machine de l'école, on peut utiliser la commande find :

```
find . -name "vecteurs.c"
```

permet de trouver le fichier `vecteurs.c`

Trouvons maintenant tous les fichiers C de notre répertoire. Ils terminent tous par `.c` :

```
find . -name "*.c"
```

On peut aussi chercher des fichiers par type :

```
find . -type d
```

(ici on trouve tous les répertoires) et par taille, date, ...

Pour chercher les gros fichiers de notre compte (et éventuellement faire le ménage), on utilise (par exemple) :

```
find . -type f -size +10000000c -printf '%s %p\n'
```

(trouve tout les fichiers à partir de . de taille >10Mo et imprime quoi ?)

1 Jouons avec les adresses

Exercice 1 *Écrire le programme suivant :*

```
int main()
{
   int a = 10;
   int b = 50;
   int* p;

    p=&a ;
    b=*p ;
    *p=42;

    return 0;
}
```

1. *Afficher après chaque instruction la valeur entière (printf avec %d) de :* `a`, `b`, `*p`, *et des adresses* `&a` *et* `p` *(printf avec %p). Vérifier que tout se passe comme prévu.*
2. *Commenter la ligne d'initialisation de p (ie mettre l'initialisation en commentaire, pour que l'initialisation ne soit plus prise en compte lors de la compilation).*
3. *Compiler avec -Wall et ignorer le Warning. Quelle erreur obtient-on à l'exécution ? (il y a deux possibilités).*

On n'ignorera plus JAMAIS les Warnings

Exercice 2 (Segmentation fault = problème accès mémoire) *Dans un autre fichier c, déclarer un tableau d'entiers de taille 2 initialisé avec des zéros, et imprimer la case 10. . . . .*

Observer que là encore, `clang -Wall` *met des Warnings. Les ignorer*[1]*, et exécuter. Observer les comportements possibles à l'exécution. Même exercice en accédant à une case d'indice 4200.*
On n'ignorera plus JAMAIS les Warnings

Exercice 3 (Pointeurs de pointeurs) *(source : http://diwww.epfl.ch/w3lsp/teaching/coursC/exercices/pointeurs.html Écrire un programme déclarant une variable entière v et l'initialisant. Déclarer un pointeur pv de type correspondant à cette variable et le faire pointer sur v. Déclarer un autre pointeur ppv de type approprié pour pouvoir pointer sur le pointeur pv. Faire pointer ppv sur pv. Modifier la valeur de v indirectement en utilisant le pointeur ppv, puis vérifier en imprimant v.*

2 Pointeurs comme paramètres modifiables

On écrira un programme C COMPLET par exercice, et on testera, *evidemment.*

Exercice 4 *Écrire une procédure* `minmax` *qui prend en paramètre deux entiers a et b et qui range dans a le min et b le max. Appeler cette fonction dans le main, après avoir déclaré deux variables : l'une valant 23 et l'autre valant 42. Vérifier.*

Exercice 5 (Bonus pour les rapides) *Écrire une procédure* `logint` *qui prend en paramètre deux entiers n et p, et retourne la plus grande puissance q de p dans n. Elle calcule en outre le coefficient multiplicateur d et le reste r tels que* $n = d \times p^q + r$ *avec* $r < p^q$ *et* $d < p$. *r et d seront passés par adresse. Il est interdit d'utiliser la fonction* `log`. *On testera à l'aide des égalités* $27 = 1 \times 2^4 + 11$, $98 = 9 \times 10^1 + 8$. O*n écrira d'abord soigneusement l'algorithme en pseudo-code sur une feuille*

3 Utilisation de gdb/ddd pour tracer des programmes

Gdb est un débugger : c'est un outil d'inspection de programmes écrits en C, C++, ... Il est beaucoup utilisé lors du développement de programmes, afin de trouver les bugs. Ici nous allons prendre contact avec gdb via une interface graphique (laide, rudimentaire, mais utile), nommée *ddd.*

Exercice 6 *Reprendre le programme de l'exercice 1, le copier en* `exo-ddd.c`.

1. *Compiler ce programme avec l'option -g de clang (obligatoire pour utiliser gdb ou ddd) :*

```
clang -o exoddd exo-ddd.c -Wall -g
```

1. Oui, là, je vous autorise !

on a uniquement ajouté l'option -g

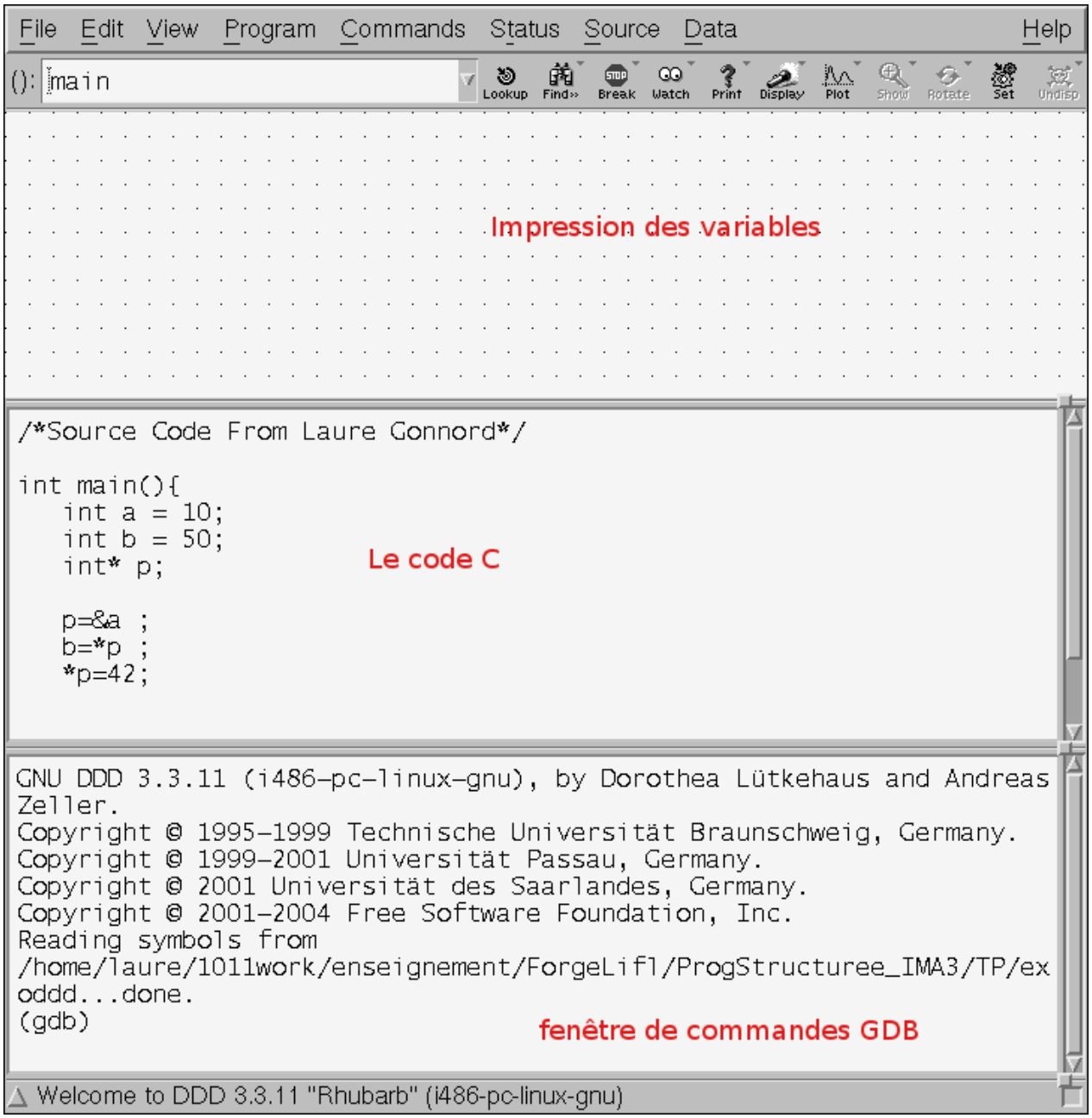
2. *Vérifier qu'il s'exécute toujours.*
3. *Lancer ddd, en remarquant qu'on lance sur le binaire :*

   ```
   ddd exoddd &
   ```

4. *Observer l'environnement en se reportant à la feuille annexe.*
5. *Mettre des points d'arrêts aux endroits suivants : ligne après les déclarations, puis après chaque instruction.*
6. *Lancer ddd : run dans la fenêtre du bas.*
7. *Afficher (display) au premier point d'arrêt les variables a, b, p, $*p$.*
8. *Pour passer d'un point d'arrêt à un autre, taper next (ou n) dans la fenêtre de commandes gdb. Observer la fenêtre du haut lors de cette commande.*

EXERCICE 7 *Utiliser ddd pour observer le comportement des variables de vos programmes écrits dans ce tp.*

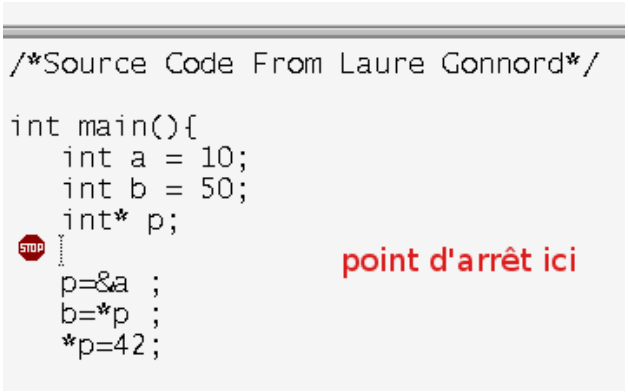
Fenêtre Initiale GDB Après lancement de ddd sur le binaire, on obtient une fenêtre :



Exécution du code Dans la fenêtre de commandes gdb :

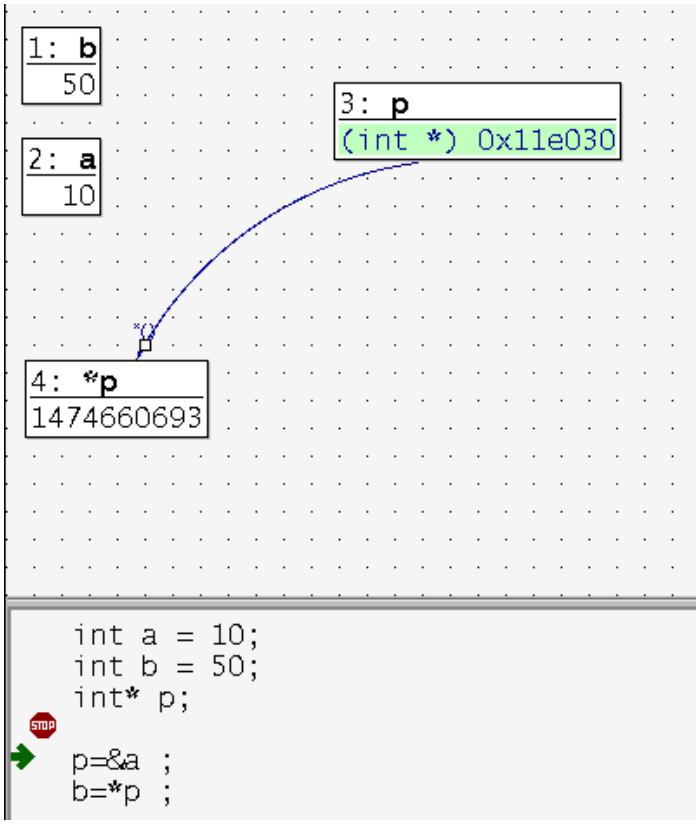
```
(gdb) run (+entrée)
```

BreakPoints Pour stopper l'exécution à un endroit précis du code, on met un point d'arrêt : clic dans le code à l'endroit en question, puis clic droit, set breakpoint.



Si il y a des breakpoints, l'exécution s'arrête au premier rencontré. On est alors capable d'afficher les variables.

Display En cours d'exécution, si on est arrêté au milieu du programme, on peut afficher les variables disponibles à cet instant, à l'aide de display. **attention, pas de clic droit dans la fenêtre du haut !** Pour le point d'arrêt montré plus haut, on peut afficher a, b, p et $*p$ (click droit sur la variable dans la fenêtre de code, et display). On obtient alors des jolis dessins dans la fenêtre du haut.



Remarque : si on double-clic sur une adresse dans la fenêtre du haut, le pointeur se "déroule".