

Decouverte du débogueur DDD

par [Hiko Seijuro](#) ([Autres articles](#))

Date de publication : 15/08/2007

Dernière mise à jour : 15/08/2007

Ce tutoriel vous permet d'aborder le débogueur DDD et suppose que vous savez déjà utiliser un débogueur. Cela sous-entend que vous savez ce qu'est une pile d'appels, un point d'arrêt, ...

- 1 - Introduction
- 2 - Préparer un projet pour le débogage
 - 2.1 - Pour déboguer, il faut compiler en... debug
 - 2.2 - Lancement de DDD
 - 2.3 - Ouverture de la partie à déboguer
 - 2.4 - Passer des arguments à la ligne de commande
- 3 - Les outils de débogage
 - 3.1 - Les espions
 - 3.1.1 - Les infobulles
 - 3.1.2 - La zone d'affichage des variables
 - 3.2 - La pile des appels
 - 3.3 - Les points d'arrêt
 - 3.3.1 - Points d'arrêt sur une entrée de fonction
 - 3.3.2 - Points d'arrêt sur une ligne précise du code
 - 3.3.3 - Se positionner sur une fonction précise
 - 3.4 - L'exécution pas à pas
 - 3.5 - Modifier & relancer l'exécution
- 4 - Configuration plus poussée de DDD
 - 4.1 - Rendre la sauvegarde de la configuration effective
 - 4.2 - Configurer DDD
 - 4.2.1 - Data
 - 4.2.2 - Helpers
- 5 - Les possibilités avancées
 - 5.1 - Visualisation des tableaux sous formes de courbes
 - 5.2 - Les points d'arrêts conditionnels
 - 5.3 - Associer des commandes à des points d'arrêts
- 6 - Conclusion
- 7 - Pour aller plus loin...
- 8 - Remerciements

1 - Introduction

Un débogueur est un outil qui permet d'exécuter un programme en le contrôlant presque intégralement. L'objectif d'un tel programme est de permettre au développeur de rechercher les erreurs de programmation qu'il a commise et qui ne sont pas détectées par le compilateur.

Le débogueur DDD est en fait une surcouche graphique des débogueurs en mode texte tel "gdb". Ce tutorial se base sur une approche C & C++ du débogueur, mais il peut être utilisé pour d'autres langages. Pour l'installation de cet outil, je vous laisse le soin de vous fier au mode d'installation de votre distribution favorite.

2 - Préparer un projet pour le débogage

2.1 - Pour déboguer, il faut compiler en... debug

Pour commencer, il faut utiliser le compilateur gcc/g++ suivant que vous conceviez une application C ou bien C++. A la ligne de compilation, il faut rajouter l'option "-g" comme par exemple :

```
g++ -o test test.cpp -g
```

Ce mode de compilation correspond au mode "debug" pour les utilisateurs de visual studio.

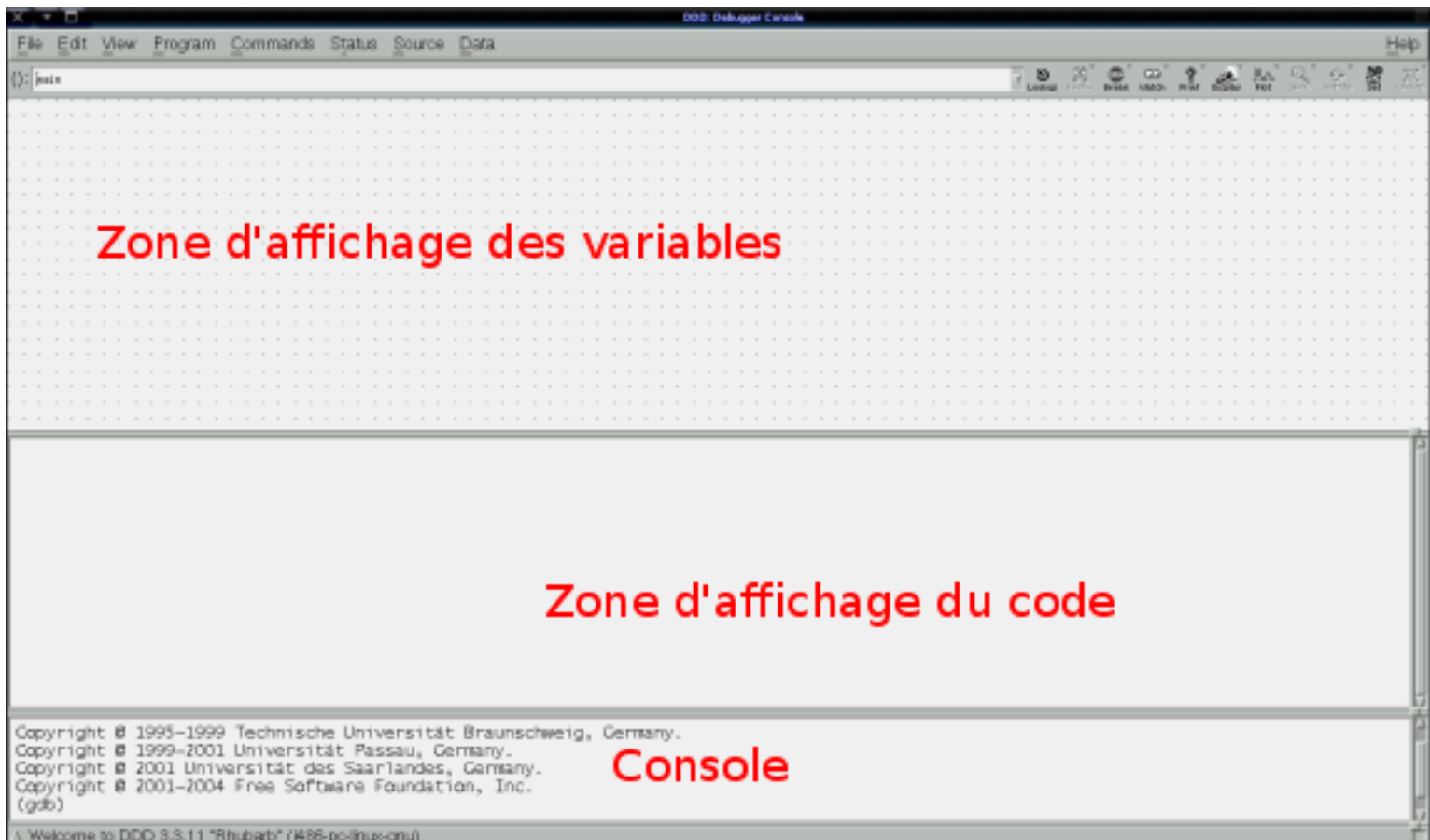
2.2 - Lancement de DDD

Maintenant que la compilation, est effectuée il faut lancer DDD, en y spécifiant ou non le chemin de l'exécutable à déboguer. Pour cela il suffit de saisir la commande suivante:

```
ddd [chemin_executable]
```

Il est possible d'y associer plusieurs options qui permettent de choisir le débogueur associé. Pour plus d'informations je vous laisse consulter le [manpage associé](#)

Voici ce que vous obtenez :



2.3 - Ouverture de la partie à déboguer

Il est possible de déboguer plusieurs choses : un exécutable, un dump, ... Pour ouvrir ce que vous souhaitez, rendez vous dans le menu fichier (si vous n'avez rien spécifié dans la ligne de commande pour lancer ddd).

Dans notre cas, nous allons ouvrir un exécutable de base obtenu via le code source suivant :

```
#include <string>
#include <iostream>

using namespace std;

void printtest(string strCmd)
{
    cout << strCmd << endl;
}

int main(int argc, char **pArgs)
{
    if (argc > 1)
    {
        string strCmd = "le premier paramètre est : ";
        strCmd += pArgs[1];
        printtest(strCmd);
    }
    else
```

```
{  
    printtest("commande vide !");  
}  
  
return EXIT_SUCCESS;  
}
```

2.4 - Passer des arguments à la ligne de commande

Maintenant que DDD est lancé, il faut indiquer la ligne de commande qui va être passée à l'application en cours de débogage.

Attention, les arguments ne sont spécifiés qu'au lancement de l'application, il faut donc indiquer les points d'arrêts nécessaires avant.

Pour saisir la ligne d'arguments, rendez-vous dans le menu "Program" puis l'item "Run..." (ou plus simplement la touche F2). Vous obtenez alors ceci :



Il vous suffit de saisir la liste d'arguments dans la partie "Run with Arguments" puis de lancer l'exécution du programme en cliquant sur le bouton "run". Vous devriez voir dans la partie console soit votre argument, soit le message "commande vide!".

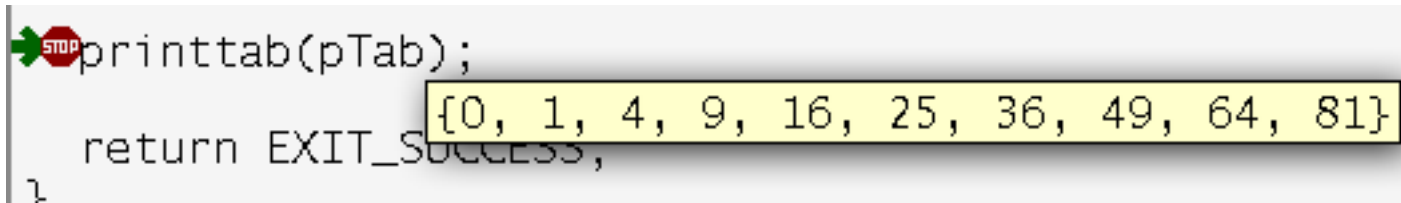
3 - Les outils de débogage

3.1 - Les espions

Une première fonctionnalité primordiale du débogueur est la possibilité de scruter la valeur des variables, et ce à tout moment. Il existe plusieurs façons de visualiser ces valeurs, lorsque le programme est stoppé en mode débogage.

3.1.1 - Les infobulles

La première manière de visualiser une variable est de simplement placer le pointeur de la souris sur celle-ci : une infobulle vous indiquera sa valeur. S'il s'agit d'un tableau ou d'une chaîne de caractères, vous pouvez même voir le contenu d'un tableau ou de la chaîne :

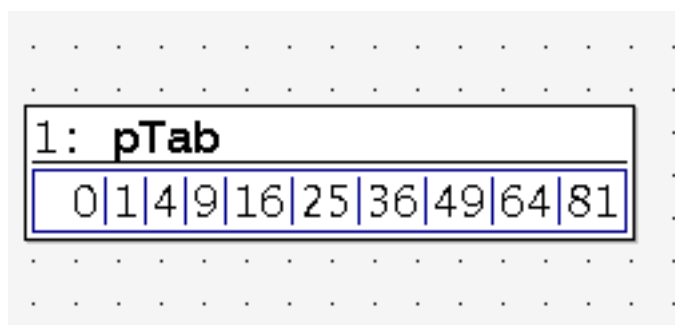


```

printtab(pTab);
return EXIT_SUCCESS;
  
```

3.1.2 - La zone d'affichage des variables

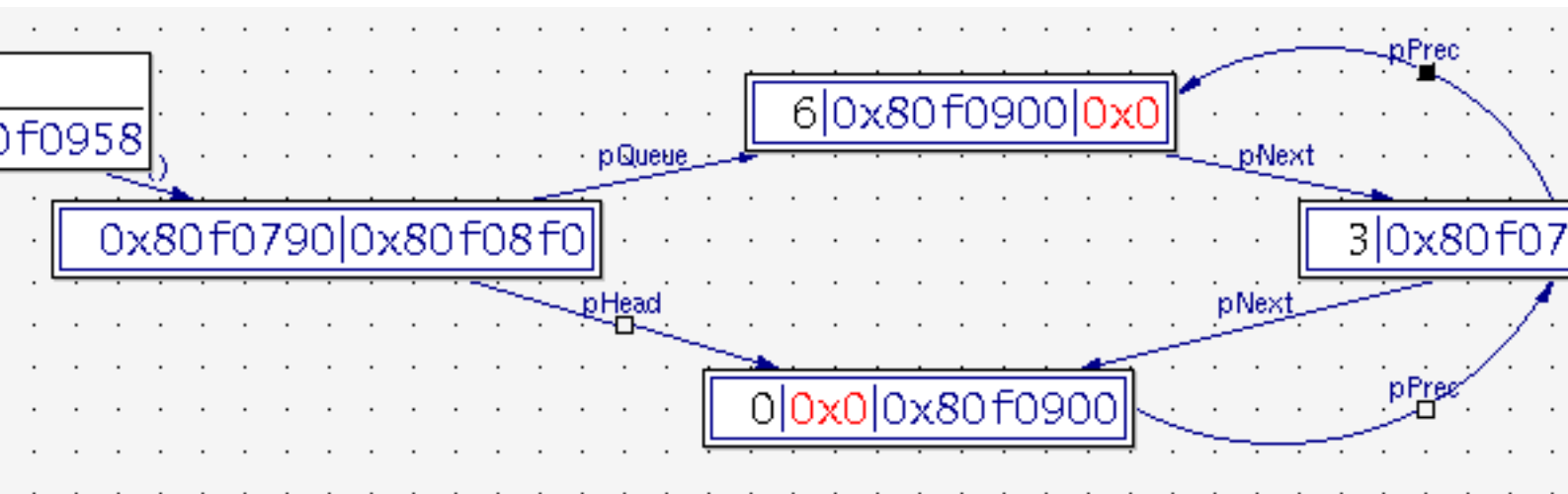
Il est possible d'afficher les variables de manière permanente pour pouvoir visualiser leurs évolutions. Pour cela, il faut effectuer un clic droit sur la variable désirée et sélectionner l'option "display". L'option "print" affiche juste dans la zone "console" la valeur de la variable. Les options "display *" et "print *" permettent, comme vous vous en doutez, non pas d'afficher la valeur de la variable mais la valeur contenu à l'adresse que contient la variable. Pour être plus clair, si vous prenez une variable représentant un tableau "display" montrera le contenu du tableau et "display *" le premier élément. Lorsque l'option "display" a été invoquée, vous verrez apparaître votre variable dans la zone de visualisation des variables :



```

1: pTab
0|1|4|9|16|25|36|49|64|81
  
```

Un cas spécifique ne peut être affiché qu'en utilisant cette zone : il s'agit de l'affichage d'une structure "récursive" (pile, liste, ...) En effet, comme il s'agit de pointeurs, il faut afficher la variable puis dérouler manuellement les pointeurs comme suit :



Pour dérouler manuellement, il suffit de double cliquer sur le membre de la structure que l'on souhaite afficher. Dans le cas présent voici la déclaration de la structure liste utilisée :

```
struct stElement
{
    int nValue;
    struct stElement *pNext;
    struct stElement *pPrec;
};

struct stList
{
    struct stElement *pHead;
    struct stElement *pQueue;
};

typedef struct stElement Element;
typedef struct stList List;
```

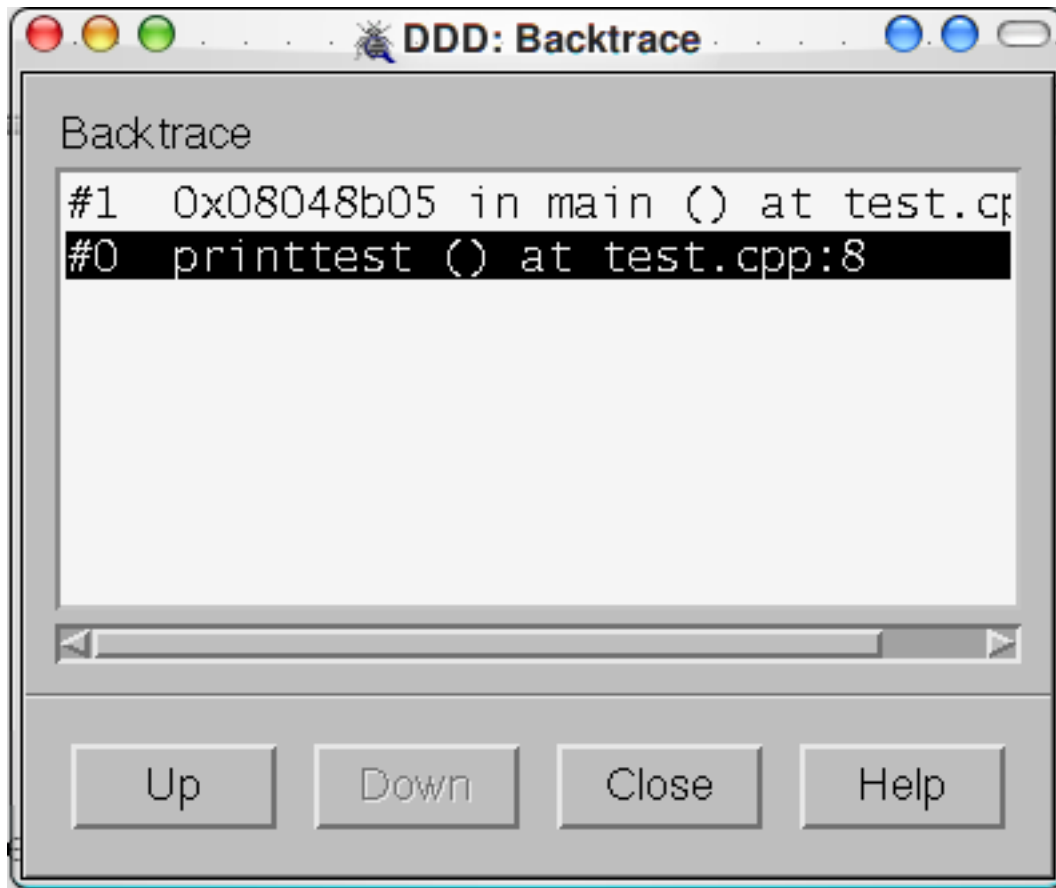
On retrouve donc les 3 membres par élément de la liste :

- un entier représentant la valeur
- un hexadécimal représentant l'adresse de l'élément suivant
- un hexadécimal représentant l'adresse de l'élément précédant

Comme vous pouvez vous en douter, l'adresse 0x0 correspond à la valeur NULL.

3.2 - La pile des appels

La pile des appels peut être obtenue grâce au menu "Status" puis à l'entrée "Backtrace". Vous arrivez ainsi sur une fenêtre de ce type :



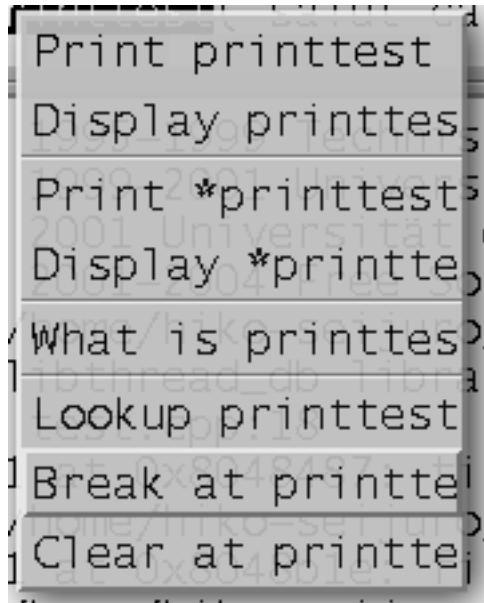
En double cliquant sur une entrée, DDD vous place automatiquement sur la ligne pointée par l'entrée dans la pile : la fonction appelante en général.

3.3 - Les points d'arrêt

On distingue 2 types de points d'arrêt : ceux qui sont à l'entrée d'une fonction et ceux qui sont placés à des points précis du code. En fait les premiers ne sont qu'un cas particulier des seconds.

3.3.1 - Points d'arrêt sur une entrée de fonction

Pour ajouter un point d'arrêt à l'entrée d'une fonction, on peut se placer dans le contexte d'une fonction appelante, sélectionner le nom de la fonction (un double clic sur ce nom suffit !) et effectuer un clic droit pour obtenir le menu suivant :



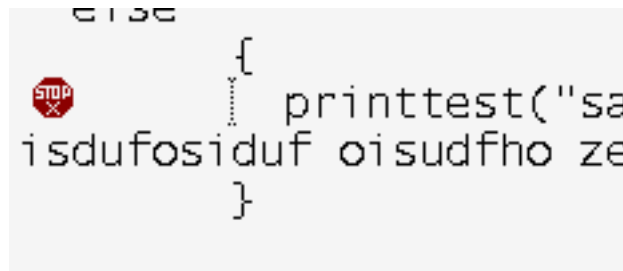
Il suffit alors de sélectionner la ligne "Break at ...", et le point d'arrêt sera placée sur la première ligne codante de la fonction comme par exemple :

```
void printtest(string strCmd)
{
    cout << strCmd << endl;
}
```

Il faut être conscient qu'à chaque appel de la fonction, un arrêt sera effectué et pas seulement celui effectué par la fonction appelante

3.3.2 - Points d'arrêt sur une ligne précise du code

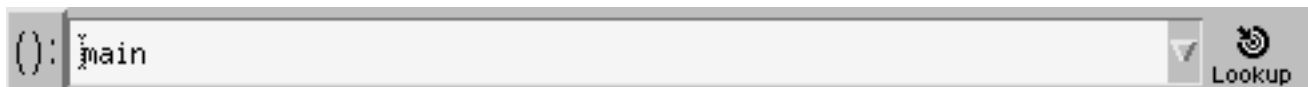
Pour assigner un point d'arrêt à une ligne de code précise, il faut effectuer un clic droit et choisir l'élément "Set Breakpoint". Il est possible d'appliquer un point d'arrêt temporaire (il se détruit dès qu'il atteint) avec la commande "Set Temporary Breakpoint" :



Comme vous pouvez le constater, la "petite croix" différencie un point d'arrêt temporaire d'un point d'arrêt classique.

3.3.3 - Se positionner sur une fonction précise

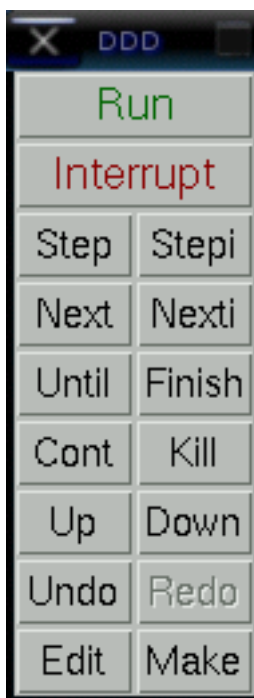
Pour se positionner dans une fonction précise, il est possible de faire une recherche de cette fonction. Pour cela, il faut se servir de la barre située en dessous de la barre de menu :



Lors du clic sur "Lookup" ou de la validation (en appuyant sur la touche "Entrée"), DDD place la première ligne de la fonction au milieu de la zone d'affichage du code. Ceci est très utile si on souhaite mettre un point d'arrêt dans une fonction n'étant pas appelée directement (ou de manière rapidement visible)

3.4 - L'exécution pas à pas

DDD possède une petite zone de commande qui permet de contrôler l'exécution du programme :



Voici la correspondance de chaque élément du panneau de contrôle :

- **Run** : permet de lancer l'exécution du programme.
- **Interrupt** : permet de stopper l'exécution du programme. **Il ne s'agit pas de mettre le programme en pause mais de l'arrêter complètement**
- **Step** : permet d'effectuer un pas en avant dans l'exécution y compris en entrant dans le corps d'une fonction (méthode) appelée
- **StepI** : permet d'effectuer un pas en avant dans l'exécution d'une instruction exactement
- **Next** : permet de continuer l'exécution sans rentrer dans le corps des fonctions appelées, le pas étant la **ligne de code**
- **NextI** : permet de continuer l'exécution sans rentrer dans le corps des fonctions appelées, le pas étant l'**opcode assembleur**
- **Until** : permet de laisser une boucle s'exécuter et continuer le débogage à partir de l'instruction qui suit.
- **Finish** : permet de continuer l'exécution jusqu'à rencontrer un "return"

- **Cont** : permet de continuer l'exécution jusqu'à rencontrer un nouveau point d'arrêt
- **Kill** : permet de stopper le programme de façon très brutal !
- **Up** : permet de remonter dans la pile d'appel
- **Down** : permet de descendre dans la pile d'appel
- **Undo** : permet d'annuler l'action effectuée
- **Redo** : permet de refaire l'action qui vient d'être annulée
- **Edit** : permet d'éditer le programme source
- **Make** : permet d'appeler le programme "make"

L'exécution "Pas à Pas" permet de scruter très précisément un programme. Cela a pour objectif, en général, de surveiller le comportement d'une variable localement.

3.5 - Modifier & relancer l'exécution

Il s'agit d'un des points faibles de DDD : il est impossible de modifier un programme et de reprendre directement où l'exécution s'est arrêté. En revanche, il est possible de modifier un programme, de le recompiler et de relancer avec les mêmes arguments. La procédure est très simple, il suffit d'éditer le source (avec la zone de commande "Edit" ou de manière extérieure), de le compiler (avec le bouton "Make" de la zone de commande, ou un appel manuel) et enfin utiliser le raccourci F2 et sélectionner les arguments.

4 - Configuration plus poussée de DDD

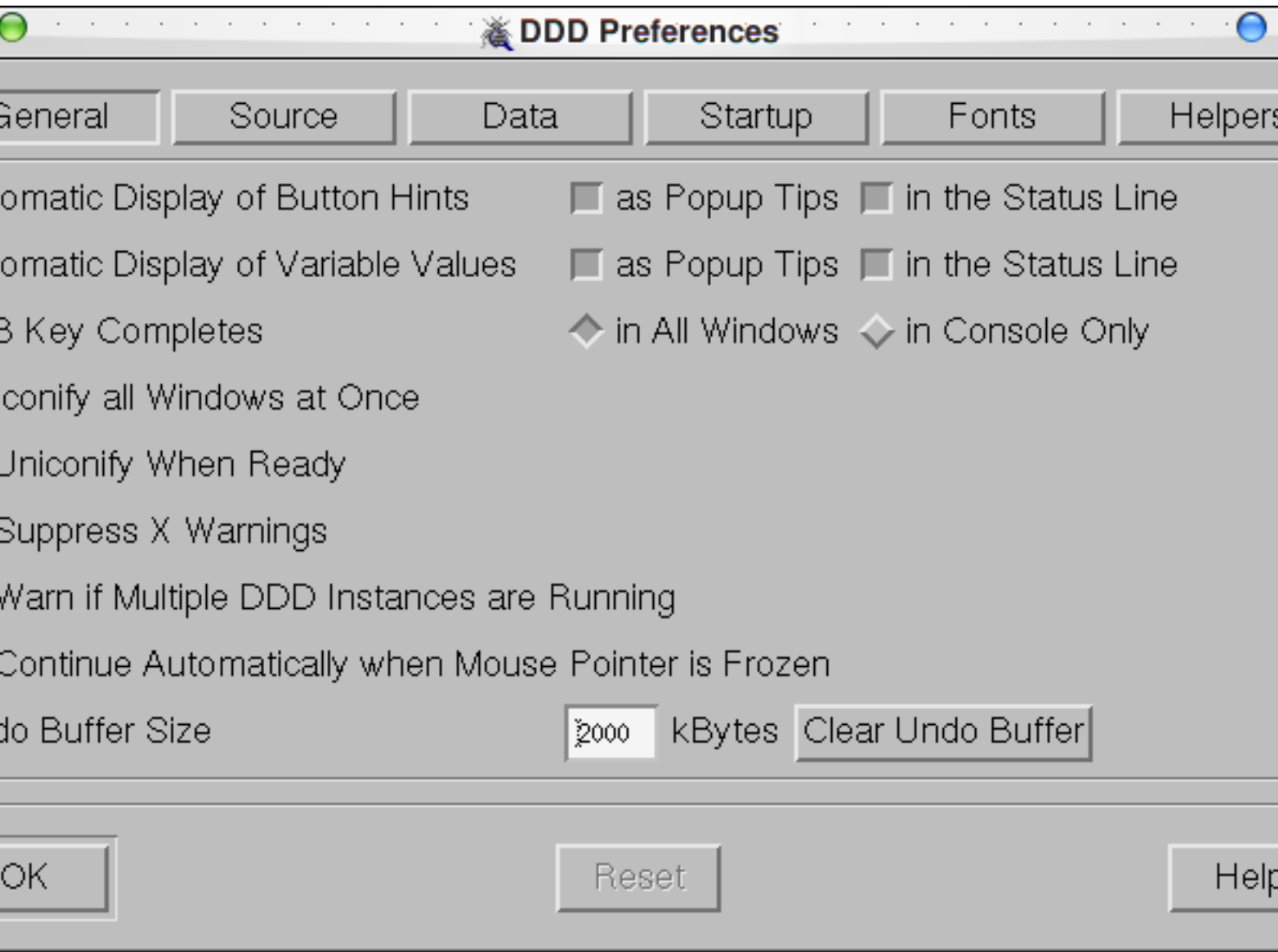
Le but de cette section est de présenter comment adapter DDD à vos besoins. Comme pour la plupart des logiciels sous linux, ces configurations seront sauvées dans le répertoire ~/.ddd. Si vous devez effectuer un formatage ou une autre opération supprimant vos configurations n'oubliez pas de copier ce répertoire si vous souhaitez conserver les modifications apportées à DDD.

4.1 - Rendre la sauvegarde de la configuration effective

Il faut effectuer une petite manipulation pour que la sauvegarde soit effective : il faut cocher l'option du menu **Edit** *Save Options*.

4.2 - Configurer DDD

Pour configurer DDD, rendez-vous dans le menu **Edit->Preferences** pour arriver sur cette fenêtre :



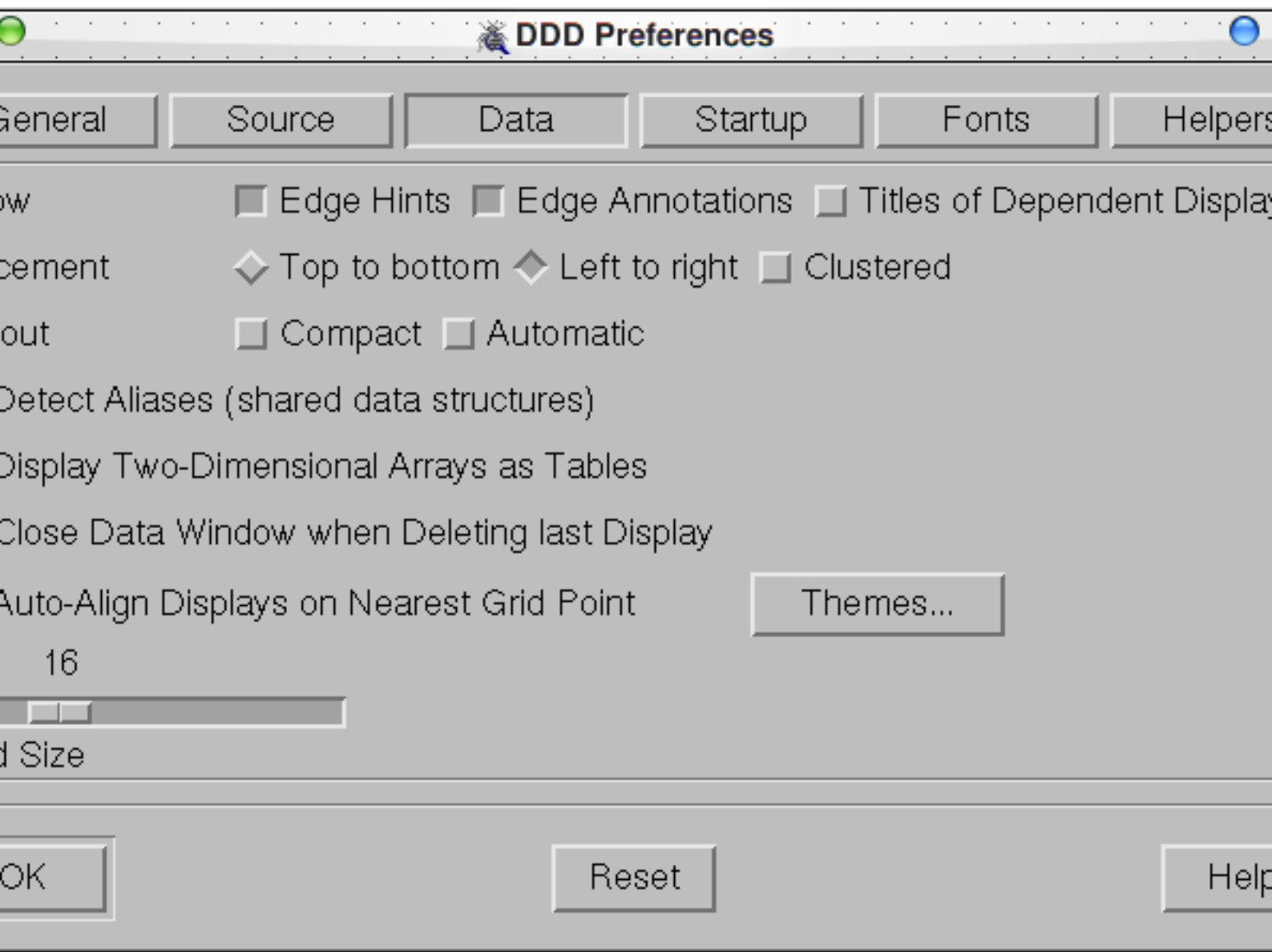
Chaque onglet correspond à un point précis de configuration :

- **General** permet de configurer l'interface de DDD
- **Source** permet de configurer la zone de commande ainsi que la zone du code source
- **Data** permet de configurer l'affichage des variables dans la zone dédiée
- **Startup** permet de configurer la mise en place de DDD lors de son démarrage
- **Fonts** permet de configurer les polices de caractères utilisées
- **Helpers** permet de configurer les programmes extérieurs qui peuvent être appelés

Les "importants" sont *Data* et *Helpers*. Nous allons donc expliquer ces 2 entités. Pour plus de précision sur les autres, je vous conseille d'aller lire le manuel de DDD dont l'adresse figure à la fin de cet article.

4.2.1 - Data

Voici à quoi ressemble la partie data :



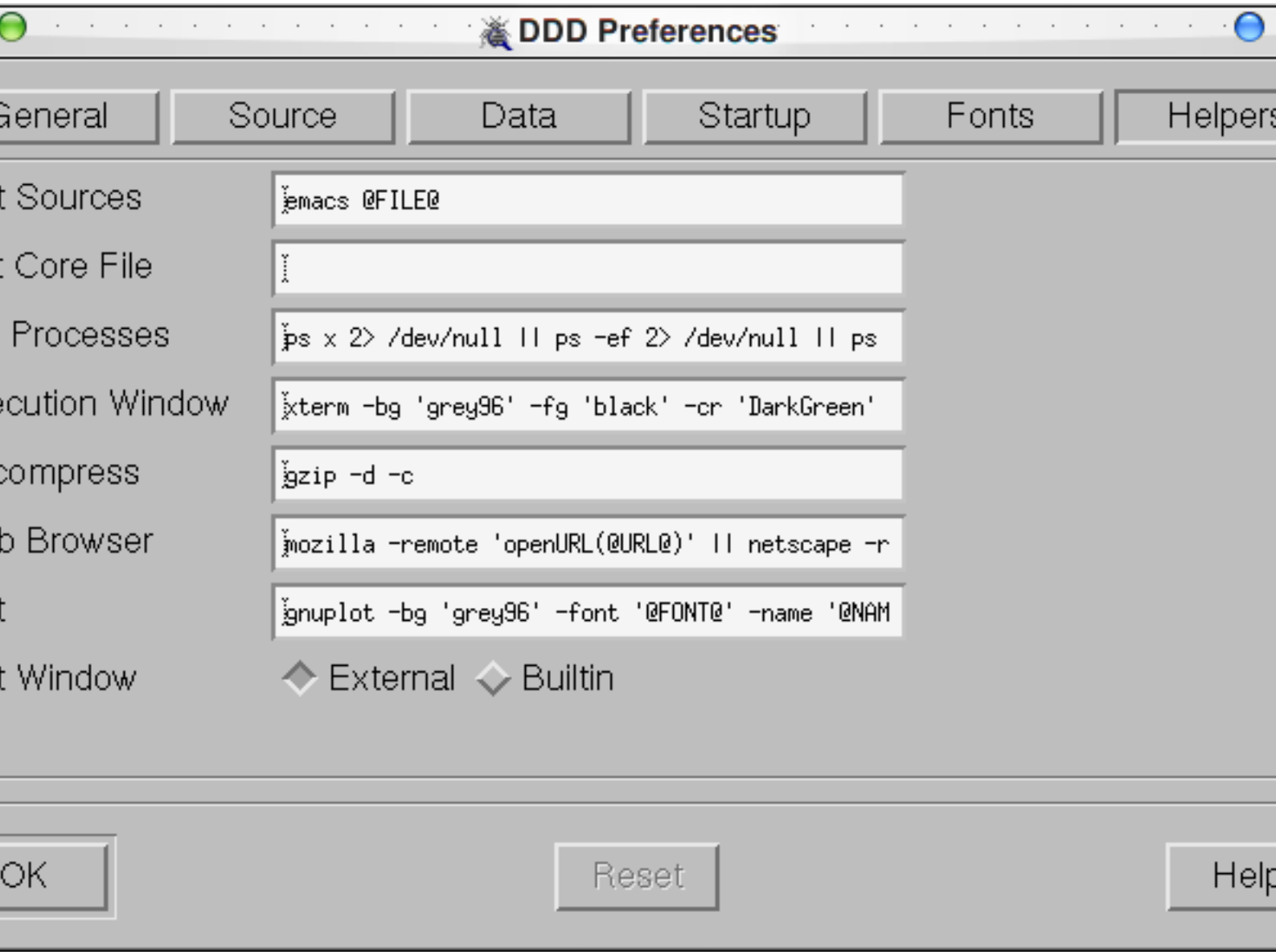
Les éléments les plus importants sont :

- **Show** permet de choisir les éléments à afficher surtout concernant les structures arborescentes : les liens (edge hints), le nom des liens (edge annotations)
- **Placement** permet de définir comment les variables seront disposées par défaut (de haut en bas, de gauche à droite, ...)
- **Display Two-dimensional Arrays as Table** permet, comme son nom l'indique, d'afficher les tableaux à 2 dimensions sous la forme d'une table. Attention cela ne marche pas pour les tableaux alloués dynamiquement !

Le bouton **Themes** permet de pousser la configuration de l'affichage des variables plus loin, je vous laisse le soin d'y consacrer une recherche sur le manuel si vous souhaitez y consacrer un peu de temps.

4.2.2 - Helpers

La partie "Helpers" permet de configurer les programmes appelés par DDD :



Les programmes extérieurs les plus utilisés sont :

- **Edit source** qui permet de choisir le programme qui va être appelé lors du clic sur le bouton "Edit" de la zone de commande. La syntaxe à respecter est la suivante : **<nomeditteur> @FILE@**
- **Plot** permet de définir la ligne de commande que va utiliser gnuplot. Je conseille de laisser cette ligne de commande telle quelle mais le manuel pourra vous aider si souhaitez manipuler de manière plus poussée gnuplot.
- **Plot Window** permet de définir si gnuplot est lancé en interne ou en externe. Je conseille de le mettre en externe car DDD peut planter facilement si il est lancé en interne.

5 - Les possibilités avancées

5.1 - Visualisation des tableaux sous formes de courbes

Pour cette partie, il faut installer un outil complémentaire qui s'appelle **gplot**. Nous allons utiliser le programme exemple suivant.

```
#include <stdlib.h>
#include <stdio.h>

void printtab(const int *pTab)
{
    int i;
    for(i=0;i<10;i++)
    {
        printf("tab[%d] = %d\n", i, pTab[i]);
    }
}

int main(int argc, char **pArgs)
{
    int pTab[10];
    int i;

    for(i=0;i<10;i++)
    {
        pTab[i] = i*i;
    }

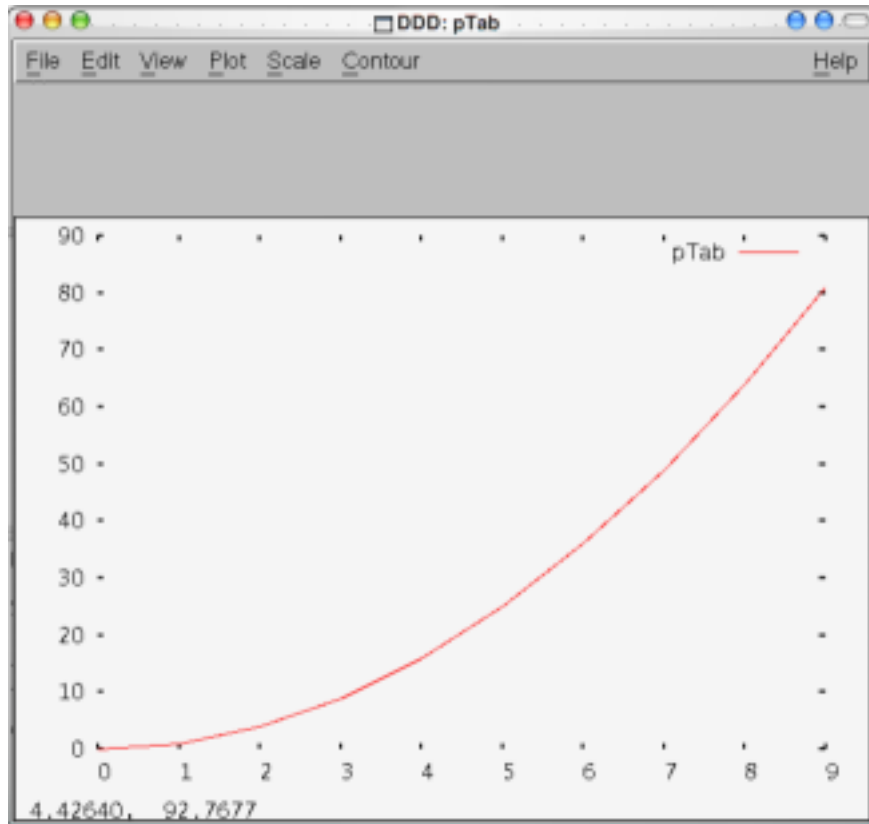
    printtab(pTab);

    return EXIT_SUCCESS;
}
```

Si on place un point d'arrêt sur la fonction "printtab", qu'on sélectionne la variable pTab (comme si on sélectionnait



un texte) puis on choisit l'élément "plot" () dans la barre d'outil. On obtient alors ceci :



Il est aussi possible d'afficher des matrices sous gnuplot :

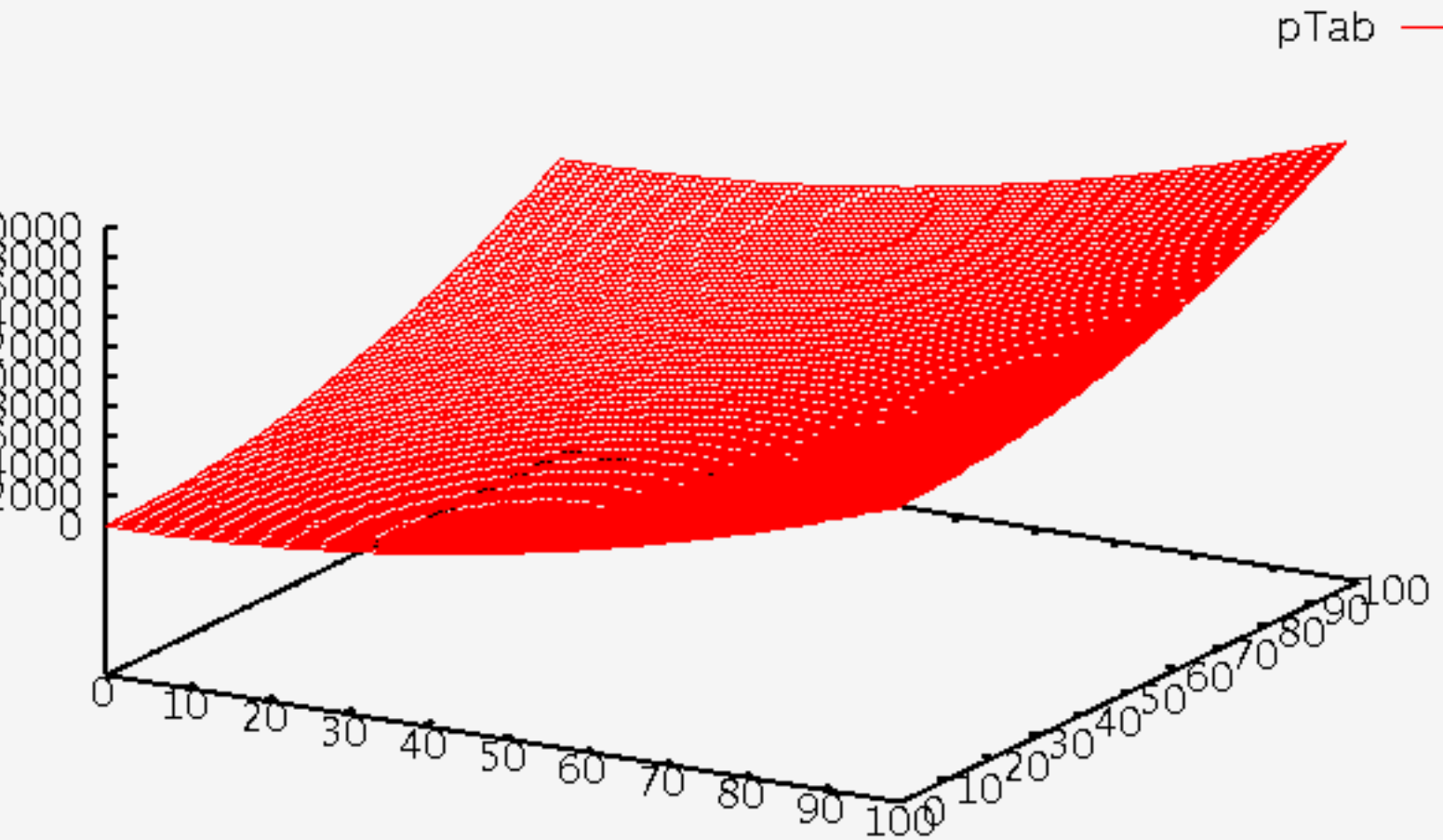
```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char **pArgs)
{
    int pTab[100][100];
    int i,j;

    for(i=0;i<100;i++)
    {
        for (j=0;j<100;j++)
        {
            pTab[i][j] = j*j+i*i;
        }
    }

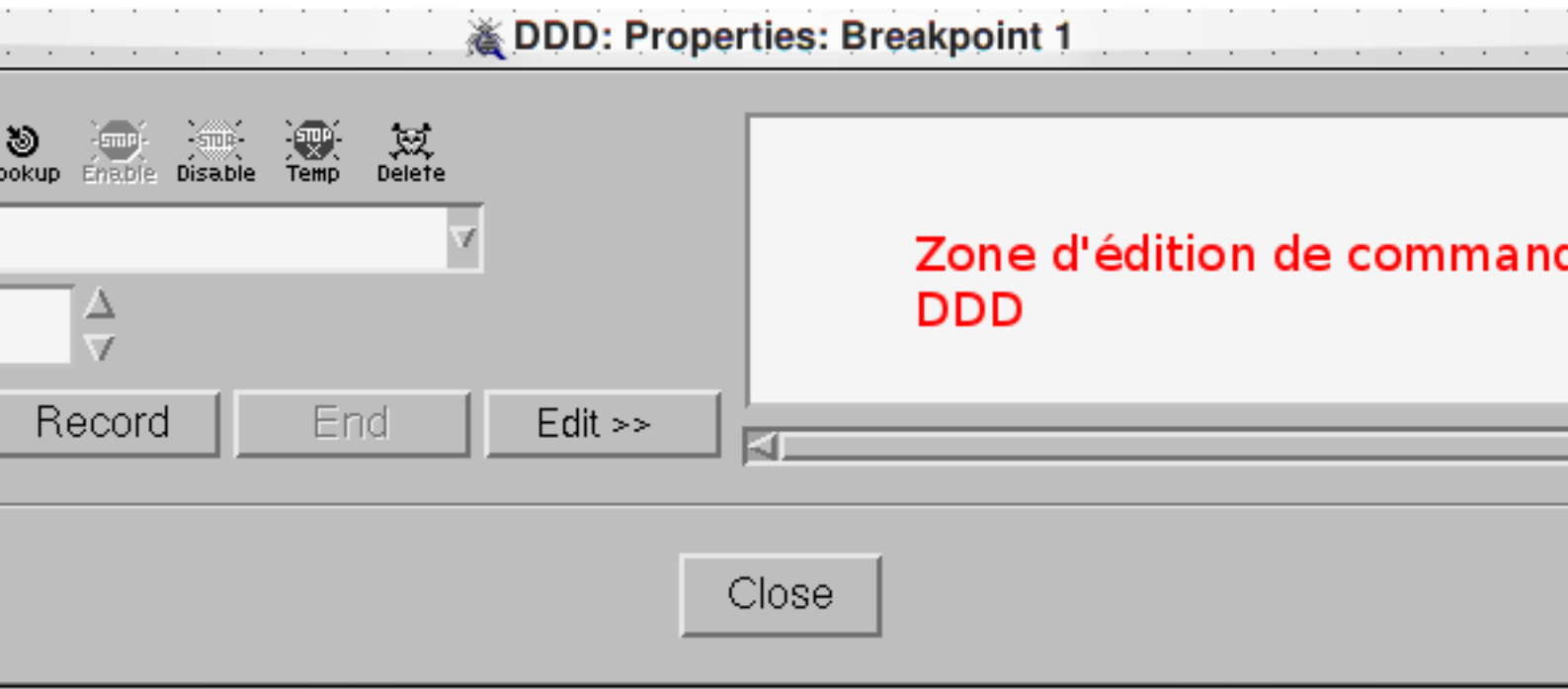
    return EXIT_SUCCESS;
}
```

Avec un point d'arrêt placé sur l'instruction return et lors de la mise en place de la suite de la procédure on obtient :



5.2 - Les points d'arrêts conditionnels

Il est possible de positionner des points d'arrêts qui ne seront actifs que sous certaines conditions (variable = une valeur précise, ...). Pour cela, il faut créer un point d'arrêt comme vu précédemment, puis effectuer un clic droit sur le point d'arrêt et sélectionner l'option "properties". Vous obtiendrez la boîte de dialogue suivante :



Dans la zone "condition", vous pouvez saisir la condition qui rendra le point d'arrêt actif. Attention il s'agit de condition sous la forme "C" (égalité est "==").

5.3 - Associer des commandes à des points d'arrêts

Dans la boîte de dialogue des propriétés d'un point d'arrêt, il y a une zone "Command". Cette zone permet de décrire une série de commandes spécifiques à DDD lors de l'arrêt du programme sur le point correspondant. Pour rendre visible cette zone, si elle ne l'est pas, il faut cliquer sur le bouton "edit".

6 - Conclusion

En conclusion, on peut dire que DDD n'est peut pas aussi évolué que visual c++ mais possède beaucoup atouts. Certaines fonctionnalités du débogueur de Microsoft ne sont pas présentes mais le développeur, dès qu'il maîtrise bien DDD, peut s'en passer. L'outil est donc un incontournable du monde libre.

De plus, il faut bien se rappeler qu'il s'agit d'un "front end" qui peut s'adapter à d'autres debugger tel jdb,... ce qui, évidemment, se fait en occultant certaines fonctionnalités précises pour chaque débbugger.

7 - Pour aller plus loin...

Si vous voulez utiliser DDD de manière plus poussée, je vous conseille de vous rendre sur le manuel de DDD **DDD**

8 - Remerciements

Je tiens à remercier Laurent Gomila, Farscape, Nico-Pyright pour leur aide et leur relecture

