TP N°1
Utilisation des entrées - Sorties

Objectifs :

- Compilation d'un programme en langage C / Utilisation du débuggeur
- Utilisation des entrées / sorties en langage C
- Création de fonctions
- Compilation séparée / Réalisation d'un makefile

1. Programme initial

Après avoir ouvert virtualBox et vous être logué sous l'environnement CentOS5, on vous demande de créer un répertoire `tp1` dans votre répertoire de travail `algo`. Vous recopierez ensuite le code décrit ci-dessous en Figure 1 (vous pouvez également télécharger sur l'intranet ce même programme) puis vous le nommerez `exo1.c` en prenant soin de la placer dans le répertoire `tp1`.

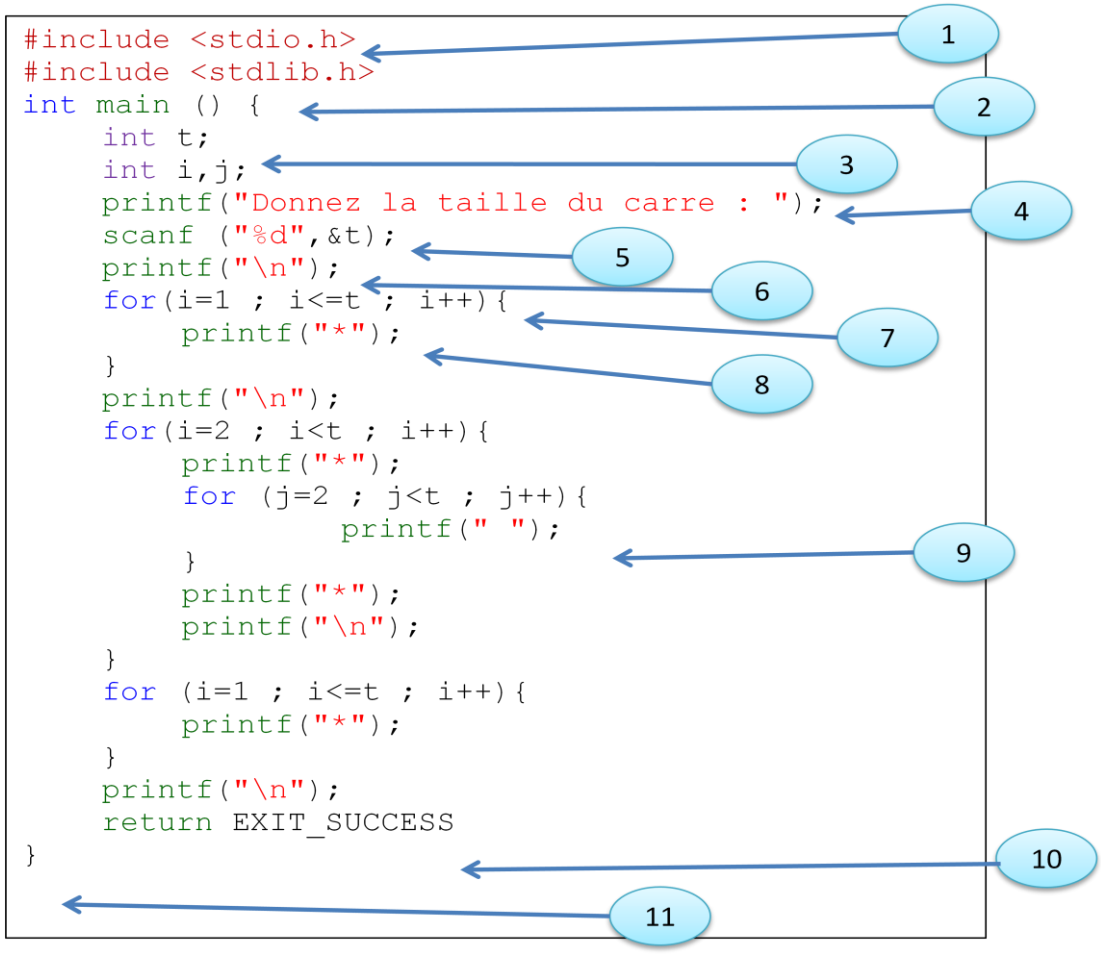

Figure 1 : programme initial

Quelques rappels :

1. Inclusion du fichier système `stdio.h` et `stdlib.h` . Le premier fichier décrit l'interface des fonctions `printf` et `scanf`, ces fonctions peuvent alors être appelées dans le programme que l'on écrit. Le deuxième fichier décrit (entre autres) la macro `EXIT_SUCCESS` utilisée en fin de programme

2. Fonction principale, main est la première fonction appelée lors de l'exécution. Ici, elle n'a pas de paramètres.

3. Déclarations de variables entières.

4. Affichage de texte à l'écran. Le texte est compris entre 2 guillemets.

5. Lecture d'un entier à partir de l'entrée standard. On précise le format de lecture `%d` => entier, et on indique l'emplacement où ranger l'entier lu : `&t` , adresse de l'emplacement associé à l'entier `t` précédemment déclaré.

6. \n est un caractère spécial : le saut de ligne. L'instruction `printf("\n");` permet donc de passer à la ligne suivante dans l'affichage à l'écran.

7. Pour i ← 1 à t faire ===> pour (i ← 1 ; tant que i ≤ t ; i ← i +1)

8. Affichage du caractère « étoile ». On aurait pu aussi écrire `printf("%c",'*');`

9. Affichage du caractère « espace » ou « blanc »

10. Le `return (EXIT_SUCCESS)` est là pour signifier au système d'exploitation que le logiciel a été exécuté sans erreur. `EXIT_SUCCESS` est une constante qui vaut `0`, elle est définie dans `stdlib.h`. On peut donc, si une erreur se produit, retourner au système une autre valeur que `0` par `return (x)`; où `x` est une variable qui contiendrait une valeur entière autre que `0`. On peut ainsi associer une certaine valeur à un certain type d'erreur. Il est conseillé de n'avoir qu'un seul point de sortie par fonction. Notez que la syntaxe `return x;` est également correcte, les parenthèses sont facultatives. Lorsque la fonction `main()` aura terminé d'exécuter ses instructions, le programme se terminera.

11. Toute fonction commence par une accolade ouvrante, et finit par une accolade fermante.

On vous demande de compiler puis d'exécuter le programme ci-dessus et d'interpréter le résultat. Vous pouvez utiliser le débuggeur pour observer ligne à ligne l'évolution du programme (voir annexe).

2. Rectangle paramétrable

Dans cette partie, on vous demande de transformer le programme précédent pour dessiner un rectangle de L x H étoiles ou L et H seront demandés à l'utilisateur du programme au moyen de `scanf`.

Sous l'éditeur de texte, utilisez « File » « Save » (ou « Save As » pour attribuer un nouveau nom de fichier) et découpez le programme précédent en deux fichiers :

- **`rect.c` contenant une fonction permettant de tracer le rectangle L*H**
- **`pp.c` contenant le programme principal qui demande L et H, puis appelle la fonction présente dans `rect.c`**

Créez un troisième fichier `rect.h` spécifiant les interfaces de `rect.c`, c'est à dire les éléments utilisables qui sont décrits en détail dans le fichier .c.

Ici, `rect.h` contiendra simplement : `void rectangle ( int L, int H ) ;` qui indique que l'on peut utiliser la procédure rectangle avec deux entiers.

Ajoutez `#include "rect.h"` à la suite de `#include <stdlib.h>` dans `pp.c`

Ceci permet au compilateur de « connaître » la procédure rectangle, et de pouvoir compiler `pp.c`. (Rappel : `<fichier_système.h>` `"fichier_utilisateur.h"`)

Compilez chacun des fichiers .c avec l'option –c pour la compilation séparée :

```
gcc –c pp.c
gcc –c rect.c
```

Compilez l'ensemble pour produire un exécutable :

```
gcc pp.o rect.o –o montp
```

(on peut aussi faire `gcc pp.c rect.o –o montp` , mais on recompile alors `pp.c`)

On vous demande d'écrire un `makefile` permettant d'exécuter automatiquement les commandes précédentes.

3. Croix de saint-André

La croix de saint André est une croix en forme de X. Ce symbole a été utilisé par de nombreux pays européens (France, Espagne, Belgique, Russie, Ecosse, Pays-Bas et Irlande)

Figure 2 : Croix de Saint-André dans le drapeau de Saint-Patrick

On vous demande de copier `rect.c` dans un nouveau fichier que vous nommerez `rect2.c` et de modifier le programme pour afficher une croix de Saint André à l'intérieur du rectangle.

Exemples :

```
Donnez la largeur : 12
Donnez la hauteur : 10

     ************
     **        **
     * *      * *
     *  *    *  *
     *   *  *   *
     *    **    *
     *    **    *
     *   *  *   *
     *  *    *  *
     ************
```

```
Donnez la largeur : 9
Donnez la hauteur : 11

*********
**     **
* *   * *
*  * *  *
*   *   *
*  * *  *
* *   * *
**     **
*       *
*       *
*********
```

Attention, selon que l'on compile `pp.c` avec l'un ou l'autre des fichiers rectangle, on obtient un exécutable différent.

4. Bornes maximum

En partant de l'exercice précédent on souhaite borner l'affichage du dessin par deux constantes LMAX et HMAX dont on définira les valeurs dans un fichier d'en-tête par exemple « limites.h ». Ces constantes sont définies au moyen de : `#define constante valeur`.

Définir dans `limites.h`, deux constantes LMAX et HMAX avec des valeurs arbitraires. Inclure ensuite le fichier `limites.h` là où c'est utile par la directive `#include "limites.h"` ajoutée par exemple après l'inclusion de `stdio.h`.

Copier et modifiez le programme, au moyen de boucles tant que, pour que l'affichage du rectangle et de la croix ne s'effectue que sur la zone LMAX x HMAX.

Exemple : avec LMAX et HMAX définies dans `limites.h` avec les valeurs respectives 20 et 5.

```
Donnez la largeur : 22
Donnez la hauteur : 11

   ********************
   **
   * *                 *
   *  *               *
   *   *             *
```

5. Affichage dans un fichier

Reprendre l'exercice précédent et enregistrer vos résultats dans un fichier.

Vous pourrez utiliser la commande `man` pour obtenir les informations détaillées concernant les fonctions d'accès aux fichiers:

`fopen(), fprintf(), fscanf() et fclose()`

6. Annexes

Compilation et génération d'exécutable

Afin d'obtenir, à partir de code en langage C, un programme que l'on peut réellement exécuter sur un ordinateur, deux étapes principales sont nécessaires :

- La compilation :
  - **Le code source** écrit en C est d'abord envoyé à travers un préprocesseur qui effectue l'inclusion des fichiers d'en-tête ainsi que le remplacement des différentes macro-instructions déclarées
  - Le code obtenu est ensuite effectivement compilé pour donner un code en assembleur
  - Le code assembleur est ensuite assemblé afin de générer un **fichier objet** associé lisible par la machine.
- Edition de lien :
  - La totalité des fichiers objets correspondants au programme complet (codes objets provenant des sources écrites par l'utilisateur et codes objets correspondants à des bibliothèques disponibles sur la machine) sont réunis pour générer un **exécutable**.

Sous l'environnement Linux, le compilateur libre gcc (GNU Compiler Collection) permet d'effectuer l'ensemble de ces opérations en une ou plusieurs étapes au moyen de la commande `gcc`

Ligne de commande	Opérations effectuées
`gcc fich.c`	Compile le fichier `fich.c` et génère un exécutable nommé par défaut a.out
`gcc fich.c -o fich.x`	Compile le fichier `fich.c` et génère un exécutable nommé `fich.x`
`gcc -c fich.c`	Compile le fichier `fich.c` et produit un code objet nommé `fich.o`
`gcc fich.o`	Edition de liens du fichier objet `fich.o` et génération d'un exécutable nommé par défaut a.out

De nombreuses autres options existent, tapez `man gcc` pour davantage d'informations.

Exécution du programme

Une fois la compilation et l'édition de liens réalisées avec succès, vous pouvez lancer votre exécutable directement dans votre terminal en tapant la commande :

```
./{nom executable}
```

Utilisation du débuggeur DDD

Le débogage est l'activité qui consiste à diagnostiquer et corriger des bugs. Lors du débogage en ligne, le programmeur exécute le programme pas à pas et effectue après chaque pas une série de vérifications.

Nous allons utiliser le logiciel **DDD** qui constitue une interface graphique pour le débogueur open source **GNU gdb**.

Afin de rendre possible le débogage il est nécessaire d'insérer lors de la compilation des informations supplémentaires appelés symboles de debug. Pour ce faire il vous suffit de rajouter l'option –g lors de la compilation avec gcc :

```
gcc -g -c exo1.c
gcc exo1.o
```

Ou bien directement :

```
gcc -g exo1.c
```

Après avoir correctement compilé votre programme avec l'option –g, lancez le debuggeur au moyen de la ligne suivante :

```
ddd a.out &
```

Cette commande lance le programme de débogage ddd sur l'exécutable a.out. La commande « & » en fin de ligne permet de garder la main sur le terminal (nous verrons en fin d'année que cette commande permet en fait de créer un 2ème processus qui se déroule en parallèle du terminal).

Vous devez obtenir l'affichage suivant :

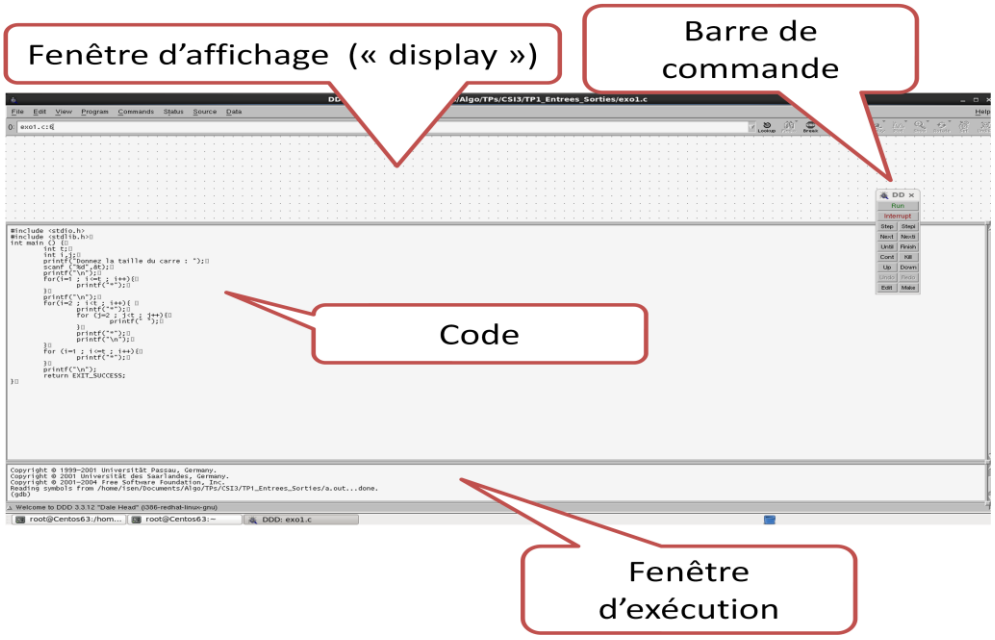

Figure 3 : aperçu de DDD

Afin de découvrir l'utilisation du débogueur ddd on vous propose les actions suivantes :

- ☑ **Dans le menu Edit/Preferences, onglet source, cochez l'option « Display source line number » afin de faire apparaitre les numéros de ligne du programme.**
- ☑ **Sélectionnez la ligne 6 et cliquez droit pour insérer un « breakpoint »**

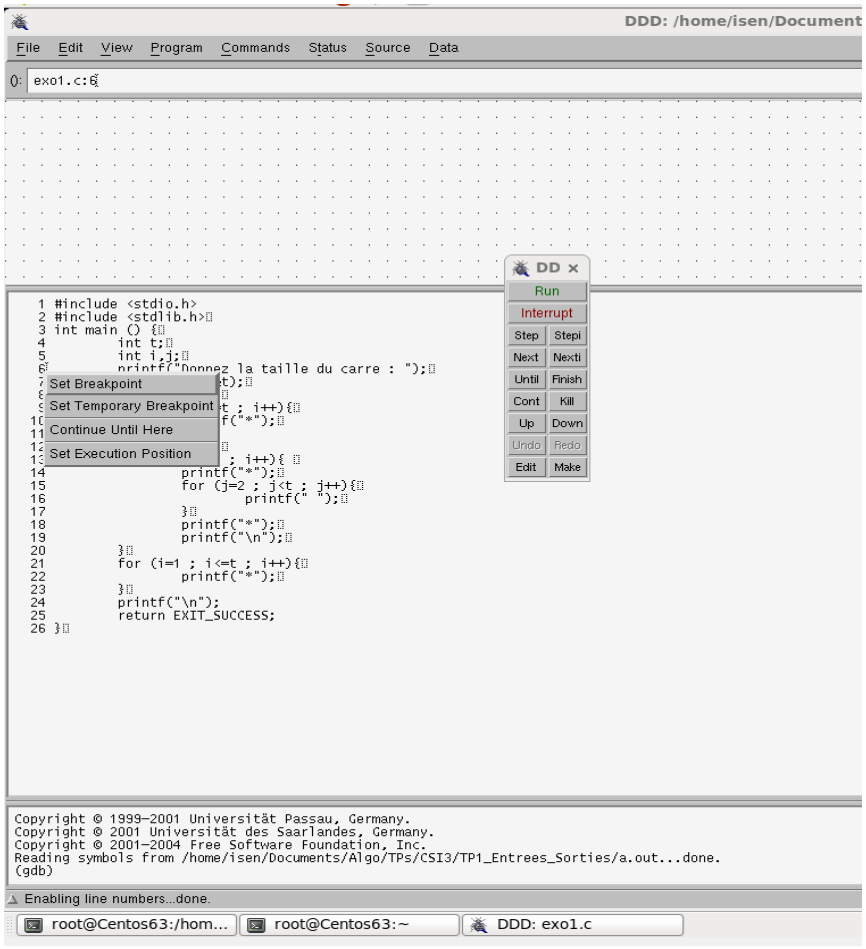

Figure 4 : insertion d'un breakpoint

- ☑ **Dans la barre de commande, appuyez sur le bouton « run », le débogueur avance jusqu'au « breakpoint » que vous avez spécifié en ligne 6. Passez le pointeur de la souris sur les différentes variables, vous verrez leurs valeurs s'afficher en info-bulles**

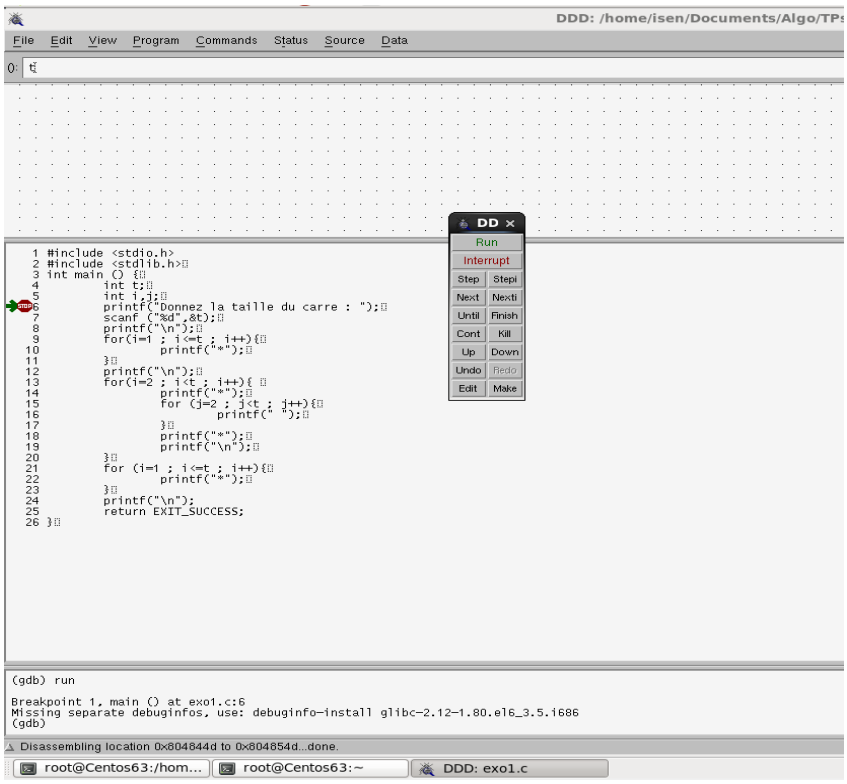

Figure 5 : commande Run

☑ **Avancez à présent d'une ligne, en appuyant sur le bouton « Step » de la barre de commande, puis avancez d'une autre ligne comme ceci :**

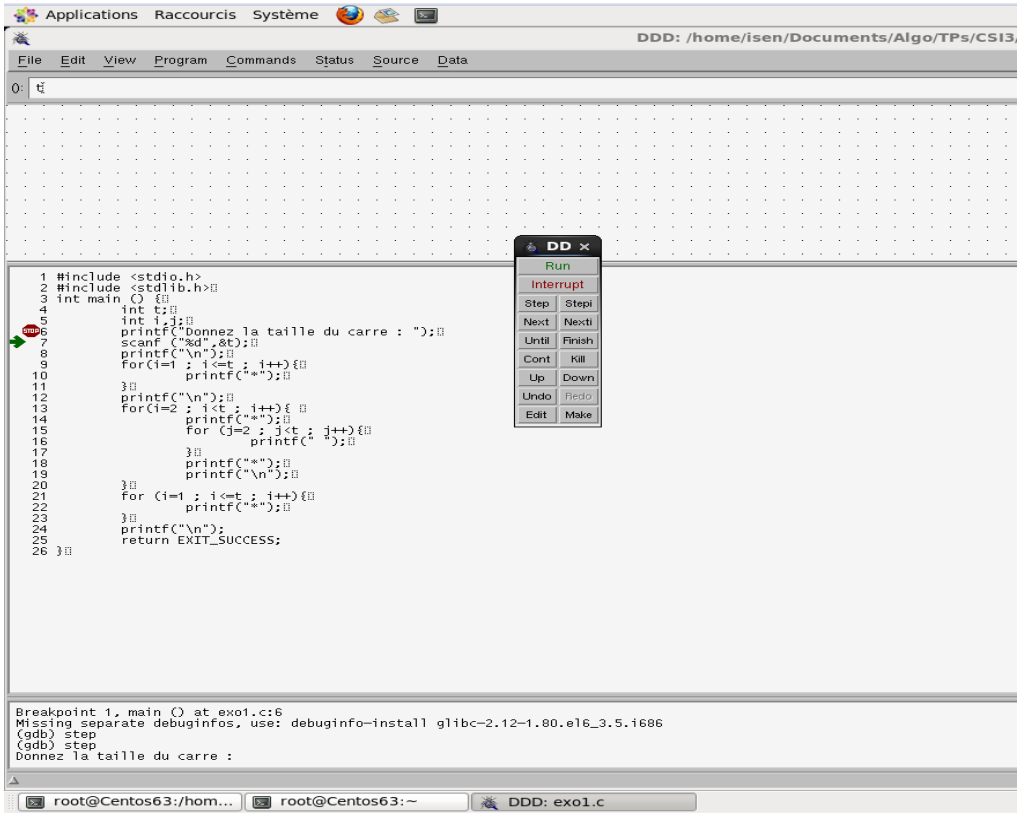

Figure 6 : fenetre d'exécution

- ☑ **Dans la fenêtre d'exécution apparaît le message « donnez la taille du carré ». Votre programme est en fait bloqué car il attend de l'utilisateur qu'il rentre une valeur dans la fonction `scanf`.**
- ☑ **Rentrez une valeur dans la fenêtre d'exécution, le débogueur passe à la ligne suivante car le programme est débloqué.**
- ☑ **Sélectionnez à présent la variable i à ligne 9 et faites clique droit sur la souris, comme ci-dessous :**

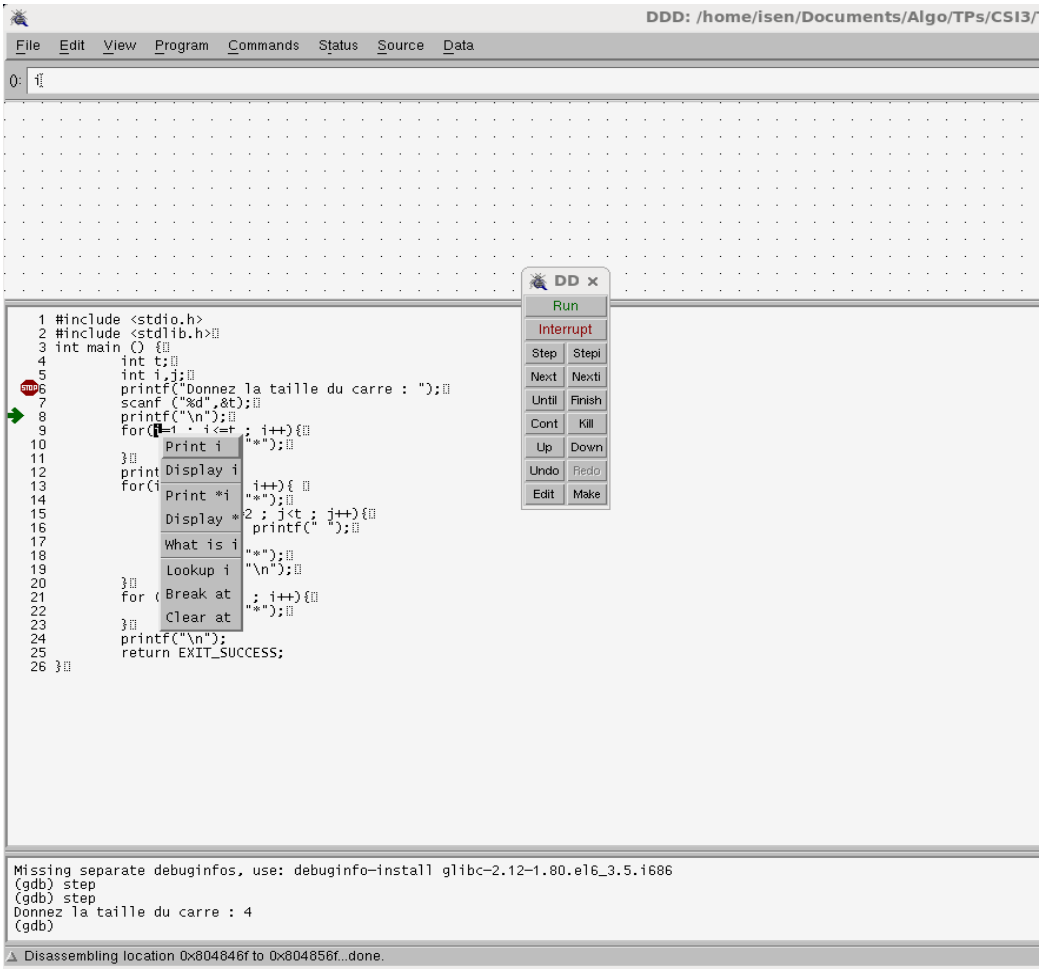

Figure 7 : Affichage du contenu de la variable i

- ☑ **Appuyez sur « Display i », le contenu de la variable i apparait dans la fenêtre d'affichage (Display). Vous pouvez à présent suivre pas à pas l'évolution du contenu des différentes variables, comme le montre la figure suivante :**

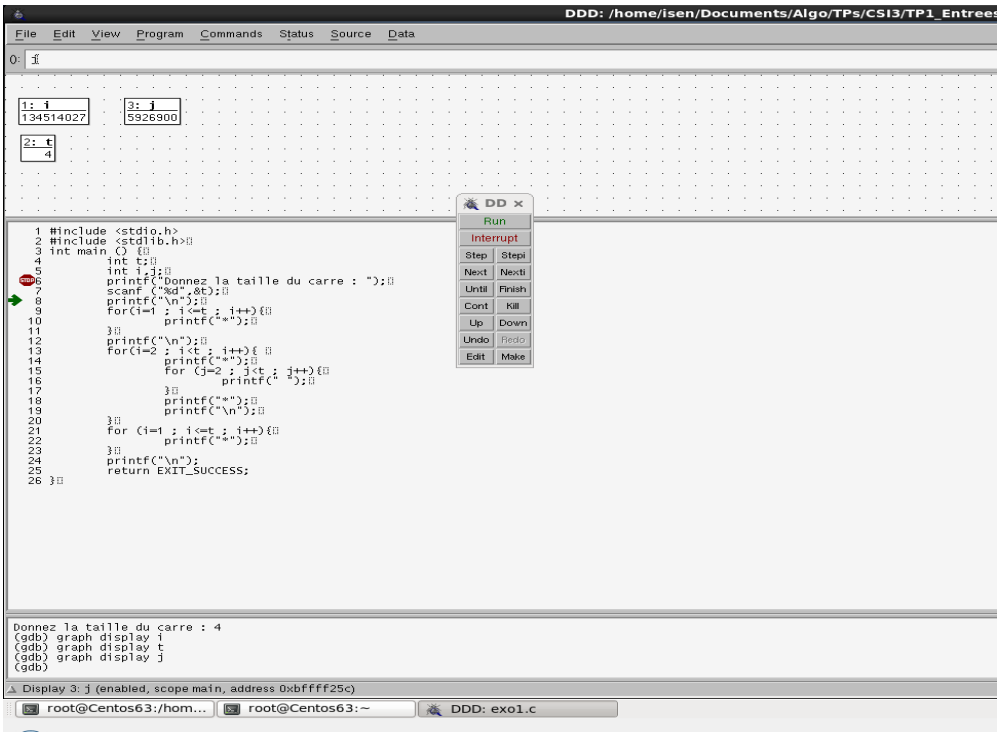

Figure 8 : affichage du contenu des variables

☑ **Enfin en cliquant droit dans la fenêtre de display et en sélectionnant « new display », vous pouvez même afficher n'importe quel contenu y compris la valeur d'un pointeur (`&t` par exemple)**

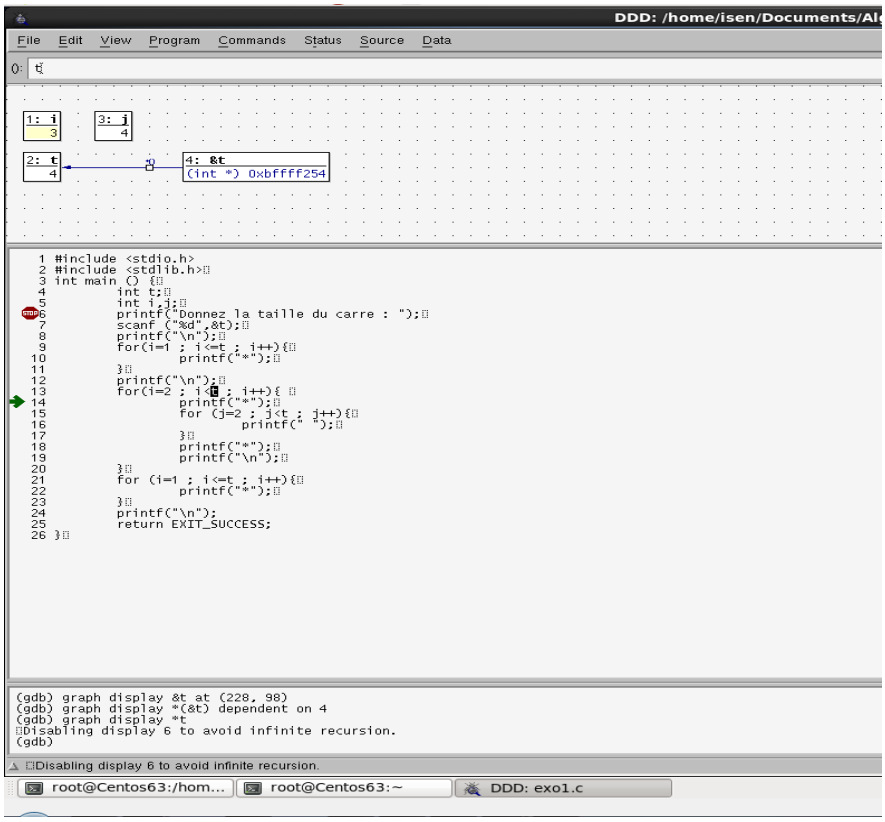

Figure 9 : Affichage de l'adresse de la variable t

- ☑ **Pour arrêter l'exécution du débogueur, appuyez sur le bouton « interrupt » de la barre de commande**
- ☑ **Si vous modifier votre programme, ne fermez pas ddd, recompilez simplement votre programme avec gcc (en n'oubliant pas l'option –g) et allez dans le menu source/reload source**