Cours d'Algorithmique-Programmation 2^e partie (IAP2): programmation impérative et structures de données simples

Présentation du débogueur *gdb*

Sandrine Blazy

ensiie - $1^{ère}$ année

6 décembre 2007

école nationale supérieure d'informatique pour l'industrie et l'entreprise

ensiie

Mise au point de programmes

- Utilisation d'un langage de haut niveau
  - beaucoup d'erreurs détectées par le compilateur
  - exceptions
- Affichage de résultats intermédiaires et de messages d'erreur
  - modification du code
- Programmation défensive
  - vérification de propriétés sur les variables
- Utilisation d'un débogueur
  - apprentissage d'un nouvel outil
  - solution efficace

ensiie

Rôle d'un débogueur

Aide à la correction d'erreurs quand on veut :

- faire démarrer un programme en précisant des informations pouvant affecter son comportement,
- faire arrêter un programme dans des conditions particulières,
- examiner ce qui s'est passé quand le programme s'est arrêté
- modifier le programme.

ensiie

Débogueur

Examine un programme durant son exécution.

Exemples

- quel est le résultat de l'exécution de la ligne 26 ?
- quelle est la valeur de telle variable ?
- que fait telle fonction ?
- que se passe-t-il si j'impose telle valeur à telle variable ?

Notion de point de programme

ensiie

Compilateur et débogueur

- le compilateur fournit des informations au débogueur
- format commun
- le choix du débogueur est lié à celui du débogueur
- le débogueur exploite du code non optimisé

ensiie

Fonctionnement du débogueur *gdb*

- interprète les commandes de l'utilisateur
- option de compilation `-g`
- démarrage : `gdb exécutable` puis `run`
- utilisation : placement de points d'arrêt statiques ou dynamiques
- l'utilisateur contrôle l'exécution du programme
- affiche l'état de la mémoire
- sortie : `quit`

ensiie

Points d'arrêts

- Placement de points d'arrêts : commande `break`
  - `b ma_fonction`
  - `b 12`
    - connue par la commande `list`
    - `l ma_fonction` ...
  - `b 25 if (x != 7)`
- `continue`
- Suppression : `delete` et `clear`
  - `delete 1`
  - `clear ma_fonction`, `clear 12`
- Activation : `enable` et `disable`

ensiie

Points d'observation

- `watch (x == 5)`
- `watch ma_var`

ensiie

Contrôle de l'exécution

- `next` exécute entièrement une instruction
- `step` exécute une seule instruction
- `finish` exécute une instruction jusqu'à son point d'appel

ensiie

Gestion des données

- Affichage de valeurs
  - `print`
    - `print /x pt`, `print * pt`
  - `display`
- `set ma_var = ma_val`
- `whatis ma_var`
- `ptype ma_var`

ensiie

Différence entre ptype et whatis

```
struct complex {double real; double imag;} v;

(gdb) whatis v
type = struct complex
(gdb) ptype v
type = struct complex {
    double real;
    double imag;
}
```

La commande where

Fournit le chemin des appels de fonctions qui ont amené à l'endroit courant.

```
(gdb) where
#0  maxmin (t=0xbffffad0, n=5, pmax=0xbffffae4, pmin=0xbffffae8)
          at coursgdb.c:9
#1  0x00001f48 in main () at coursgdb.c:29
```

La commande info
info source

```
(gdb) info source
Current source file is pgm.c
Compilation directory is /home/sandrine/iap/
Located in /home/sandrine/iap/DBG/pgm.c
Contains 31 lines.
Source language is c.
Compiled with stabs debugging format.
```

ensiie

La commande info
info functions

```
(gdb) info functions
All defined functions:

File pgm.c:
int main();
void maxmin(int *, int, int *, int *);

Non-debugging symbols:
        080483dc  printf
        080483ec  __libc_init_first
        0804840c  scanf
        0804841c  exit
        0804842c  _start
```

ensiie

La commande info
info breakpoints

```
(gdb) info breakpoints
Num Type       Disp En Address What
1 breakpoint keep y 0x08842 in main at pgm.c:25
2 breakpoint keep y 0x048a6 in maxmin at pgm.c:8
3 breakpoint keep y 0x048df in maxmin at pgm.c:12
```

ensiie

Exécution d'alternatives

```
10      for (i=1; i<n; i++)
(gdb) n
11          {if (t[i]>max)
(gdb)
12              max=t[i];
(gdb)
13           if (t[i]<min)
(gdb)
10      for (i=1; i<n; i++)
(gdb) p t[i]
$4 = 3
(gdb) p max
$5 = 3
(gdb) p min
$6 = 2
```

ensiie

Pointeurs

```
(gdb) l
3       int main()
4       {int *pointeur;
5       pointeur=NULL;
6       pointeur=(int *)malloc(sizeof(int));
7       *pointeur=12;
8       return 0;}
(gdb) b main
Breakpoint 1 at 0x8048476: file pt.c, line 5.
(gdb) run
Starting program: /home/sandrine/iap/pt
Breakpoint 1, main () at pt.c:5
6       pointeur=NULL;
(gdb) s
7       pointeur=(int *)malloc(sizeof(int));
(gdb) p pointeur
$1 = (int *) 0x0
```

ensiie

Pointeurs
(suite)

```
6         pointeur=NULL;
(gdb) s
7         pointeur=(int *)malloc(sizeof(int));
(gdb) p pointeur
$1 = (int *) 0x0
(gdb) n
8         *pointeur=12;
(gdb) p pointeur
$2 = (int *) 0x80495b8
(gdb) p *pointeur
$3 = 0
```

ensiie

“segmentation fault”

```
[sb@lunix EX_GDB]$./mon_prog
Segmentation fault
[sb@lunix EX_GDB]$ gdb mon_prog
GNU gdb 6.6
Copyright 2006 Free Software Foundation, Inc.
...
(gdb) run
Starting program: /home/sb/GDB/mon_prog

Program received signal SIGSEGV, Segmentation
       fault.
0x0804848c in inserer (eltnouv=3, eltref=2,
              ll=0xbffff708) at mon_prog.c:41
41 nouv->donnee=eltnouv;
(gdb) l 41
38 ...
```

ensiie

“segmentation fault”
(suite)

```
Program received signal SIGSEGV, Segmentation
       fault.
0x0804848c in inserer (eltnouv=3, eltref=2,
              ll=0xbffff708) at mon_prog.c:41
41 nouv->donnee=eltnouv;
(gdb) l 41
38
39 /* allocation de la nouvelle cellule */
40 /* nouv = (Liste)malloc(sizeof(elt)); */
41 nouv->donnee=eltnouv;
42
43 /* recherche de la place dans la liste */
44 prech = *ll;
```

ensiie