TP 8

Manipulation 1 (Utiliser le débuggueur gdb)**.**
Le debuggueur `gdb` *permet d'analyser l'exécution d'un programme, notamment pour en trouver les bugs. On peut en particulier exécuter les instructions pas à pas, poser des points d'arrêt* breakpoints*, afficher le contenu des variables et afficher la pile d'appel.*

Cet exercice a pour but de vous familiariser un peu avec l'utilisation de `gdb` *et éventuellement de vous donner envie d'aller plus loin.*

Le premier programme calcule naïvement quelques termes de la suite de Fibonacci.

simpleforgdb.c

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

int somme(int x, int y) {
  return x + y;
}

int fibo(int x){
  if (x <= 1) return 1;
  return somme(fibo(x - 1), fibo (x - 2));
}

int main () {
  int i;
  for (i = 0; i < 5; i++){
    printf("u(%d) = %d\n", i, fibo(i));
  }
  exit(0);
}
```

1. *Compiler avec l'option* `-g` *(commande* `gcc -g <source.c>`*) et lancer gdb sur l'exécutable créé (ici* `a.out`*) à l'aide de la commande* `gdb a.out`*.*
2. *Vérifer que cela fonctionne : en tapant la commande* `run` *dans* `gdb` *vous devriez voir le programme s'exécuter comme dans le shell.*
3. *On va maintenant poser un point d'arrêt à l'entrée de la fonction* `somme()`*. Taper* `b somme`*, puis* `r` *(raccourci pour* `run`*). Le programme s'arrête sur le premier appel de* `somme` *en affichant la valeur des arguments.* `p x` *(*`print x`*) affiche la valeur de* `x`*. Vous pouvez utiliser* `continue` *(ou* `c`*) pour continuer,* `step` *(ou simplement* `s`*) pour exécuter l'instruction suivante,* `next` *(*`n`*) pour exécuter la ligne suivante (sans rentrer dans les appels de fonctions). La commande* `delete` *(*`d`*) efface tout les points d'arrêt.*
4. *Essayer* `whatis somme`*,* `whatis y`*,* `whatis main`*,* `whatis i` *(il est normal que les variables locales à un bloc soient inconnues dans un autre bloc).*
5. *La commande* `continue` *(*`c`*) vous permet de continuer l'exécution jusqu'au point d'arrêt suivant. Allez jusqu'au troisième appel de la fonction* `somme` *à l'aide de* `continue`*.*
6. *Afficher la pile d'appel :* `backtrace` *(*`bt`*). Même chose pour le quatrième appel de* `somme`*.*
7. *Pour terminer l'exécution courante :* `kill`*. Pour quitter :* `quit`*.*
8. *Vous pouvez obtenir de l'aide avec* `help`*.*

Debugguez les autres programmes à l'aide de `gdb`*.*

simpleforgdb3.c

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

# define N 4

int main () {
  int i, j;
  char *s, buf[1024];
  s = buf;
  /* trouve les i, j tels que :
     i - j divise i + j. */
  for (i = 1; i <= N; i++) {
    for (j = N; j > 0; j--) {
      if ( 0 == ( (i + j) % (i - j) ) ) {
      s += sprintf(s, "(%d, %d), ", i, j);
      }
    }
  }
  s[-2] = '\n';
  s[-1] = '\0';
  printf("liste = %s", buf);
  exit(0);
}
```

bugnote.c

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

void erreur() {
  int tableau[16];
  tableau[27] = 18;
}

int main(){
  int note;
  note = 5;
  erreur();
  printf("Votre note: %d/20\n", note);
  exit(EXIT_SUCCESS);
}
```

simpleforgdb2.c

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

void foo() {
  char buf[] = "Hello world\n";
  printf(buf);
  sprintf(buf,"Bonjour tout le monde (en francais)\n");
  printf(buf);
}

int main () {
  int x = 0;
  foo();
  printf("x = %d\n", x);
  exit(0);
}
```

buggcc.c

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

int main () {
  int x = 2;
  int y = 1;
  int z = -1;
  int ok = 0;
  int fd[2];
  if ( pipe(fd) < 0 ) perror("Pipe");
  if (dup2(fd[1], 1) < 0 ) perror("dup2");
  if ( (y * abs(x - 1)) == ( (-z) * abs(x - 1)) ) {
    printf("ok\n");
    ok++;
  }
  if ( (y * abs(x - 1)) != ( (-y) * abs(x - 1)) ) {
    printf("ok\n");
    ok++;
  }
  if ( (2 * abs(x - 1)) != ( (-2) * abs(x - 1)) ) {
    printf("ok\n");
    ok++;
  }
  if (ok < 3) fprintf(stderr, "Erreur\n");
  exit(EXIT_FAILURE);
}
```