I Rappel pour préparer le déverminage

On se donne le code suivant pour illustrer les passage du source au binaire, par deux étapes : le pré-processeur et l'assembleur.

```c
#include <stdio.h>
#define DEFINED_CONSTANT 7
int main()
{
    int i;

    for (i = 0; i < DEFINED_CONSTANT; i++)
    {
        printf("Hello there %d\n", i);
    }
    return 0;
}
```

1. La première étape, pour voir la sortie du préprocesseur :

   ```
   gcc main.c -E >> main_p.c
   ```

   qui produit le texte suivant :

   ```
   main_p.c:
   ......
   extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__ ,
           __leaf__));
   # 944 "/usr/include/stdio.h" 3 4

   # 2 "main.c" 2

   # 5 "main.c"
   int main()
   {
    int i;

    for (i = 0; i < 7; i++) {
     printf("Hello there %d\n", i);
    }

    return 0;
   }
   ```

   On remarque que tous les #includes ont été intégrés, et le #define DEFINED_CONSTANT a été remplacé par sa valeur 7 dans la boucle.

2. Deuxième étape, on compile en code assembleur :

   ```
   $ gcc main_p.c -S
   ```

   qui produit le fichier assembleur main_p.s :

```
    .file   "main.c"
    .section    .rodata
.LC0:
    .string "Hello there %d\n"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $16, %rsp
    movl    $0, -4(%rbp)
    jmp .L2
.L3:
    movl    -4(%rbp), %eax
    movl    %eax, %esi
    leaq    .LC0(%rip), %rdi
    movl    $0, %eax
    call    printf@PLT
    addl    $1, -4(%rbp)
.L2:
    cmpl    $6, -4(%rbp)
    jle .L3
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (Debian 6.3.0-18+deb9u1) 6.3.0 20170516"
    .section    .note.GNU-stack,"",@progbits
```

3. Finalement on compile complètement :

```
gcc main_p.s -o main
```

pour obtenir le fichier exécutable

```
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  03 00 3e 00 01 00 00 00  80 05 00 00 00 00 00 00  |..>.............|
00000020  40 00 00 00 00 00 00 00  00 1a 00 00 00 00 00 00  |@...............|
00000030  00 00 00 00 40 00 38 00  09 00 40 00 1f 00 1e 00  |....@.8...@.....|
00000040  06 00 00 00 05 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00000050  40 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |@.......@.......|
00000060  f8 01 00 00 00 00 00 00  f8 01 00 00 00 00 00 00  |................|
00000070  08 00 00 00 00 00 00 00  03 00 00 00 04 00 00 00  |................|
00000080  38 02 00 00 00 00 00 00  38 02 00 00 00 00 00 00  |8.......8.......|
00000090  38 02 00 00 00 00 00 00  1c 00 00 00 00 00 00 00  |8...............|
000000a0  1c 00 00 00 00 00 00 00  01 00 00 00 00 00 00 00  |................|
000000b0  01 00 00 00 05 00 00 00  00 00 00 00 00 00 00 00  |................|
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000d0  cc 08 00 00 00 00 00 00  cc 08 00 00 00 00 00 00  |................|
000000e0  00 00 20 00 00 00 00 00  01 00 00 00 06 00 00 00  |.. .............|
.....
```

II Utiliser le dévermineur

Vous lancerez le debugger (cf. CM) pour comprendre puis corriger un programme simple. Le code source du fichier exerciceDebug_.c compile sans erreur. Cependant, à l'exécution on observe que :

(a) la fonction echange n'echange pas les valeurs des variables,

(b) une erreur de segmentation ("segmentation fault") se produit.

4. Placez un breakpoint au début de la fonction main (break main ou b main), et echange.
5. Lancez le programme (run). A partir de là vous suivrez pas à pas l'exécution (n ou next)
6. Juste avant la sortie de la fonction echange, examinez l'état des paramètres formels a et b.
7. De retour dans le main : que valent valeur1 et valeur2 (dits "paramètres effectifs" dans l'appel echange) ?
8. Proposez une solution pour que ces valeurs soient échangées même en sortie de echange et retestez.
9. Placez des breakpoints pour afficher la valeur de la variable valeur3 et du contenu "pointé" par cette variable jusqu'à l'instruction d'affectation de *valeur3. Que constatez vous ? Comment corriger cette erreur ?
10. Quittez le programme, corrigez, relancez, et vérifiez qu'il se termine sans erreur.

III Fonctions itératives et récursives

11. On rappelle la suite de Fibonacci définie par :
$u_0 = 0$
$u_1 = 1$
$u_n = u_{n-1} + u_{n-2}$ *si* $n > 1$

(a) Ecrire une fonction fibonacciIt calculant itérativement le terme de rang n de la suite dont le prototype est : int fibonnacciIt(int n) ;

(b) Ecrire une fonction fibonacciRec calculant récursivement le terme de rang n de la suite dont le prototype est : int fibonnacciRec(int n) ;

(c) Ecrire ensuite une fonction main demandant la valeur de n à l'utilisateur et affichant le terme correspondant de la suite.

12. Fonction PGCD Ecrire une fonction (non récursive) pgcd, à deux paramètres entiers, retournant le pgcd de ses paramètres. On rappelle que le pgcd est défini par les relations suivantes (a et b étant des entiers naturels) :
— $pgcd(a, 0) = a$
— $pgcd(a, b) = pgcd(b, r)$ avec $r = a \ mod \ b$, si $b \neq 0$
13. Fonction Factorielle

(a) Ecrire une fonction factorielleIt qui calcule itérativement et retourne la valeur de n ! (1 x 2 x 3 x...x n).

(b) Ecrire une fonction factorielleRec qui calcule récursivement et retourne la valeur de n ! (1 x 2 x 3 x...x n).

(c) Ecrire une fonction factorielleBis à un paramètre entier m qui calcule et retourne la valeur du plus petit entier positif n tel que n ! (factorielle de n) soit supérieur à m.