Analyse d'un fichier objet

Soit le code suivant :

```c
// fichier memoire.c
#include <stdio.h>
#include <math.h>
// D'après Marc de Falco

const int a  = 42;
int b[] = { 1, 2, 3 };
int c;

int f(int x, int y)
{
    int z = x;
    z = z * y;
    return z;
}

int main(int argc, char **argv)
{
    const int d = 1664;
    //c = f(a, d);

    printf("%f\n", cos(d));
    return 0;
}
```

Faisons-en un fichier objet :

```
$ gcc -c memoire.c
```

Lançons le programme `objdump` qui affiche des informations sur un fichier objet :

```
$ objdump -x memoire.o
```

On obtient :

```
memoire.o:     format de fichier elf64-x86-64
memoire.o
architecture: i386:x86-64, fanions 0x00000011:
HAS_RELOC, HAS_SYMS
adresse de départ 0x0000000000000000

Sections :
Idx Name          Taille    VMA                 LMA                 Off fich  Algn
  0 .text         00000057  0000000000000000  0000000000000000  00000040  2**0
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         0000000c  0000000000000000  0000000000000000  00000098  2**3
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  0000000000000000  0000000000000000  000000a4  2**0
                  ALLOC
  3 .rodata       00000008  0000000000000000  0000000000000000  000000a4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment      0000002a  0000000000000000  0000000000000000  000000ac  2**0
                  CONTENTS, READONLY
  5 .note.GNU-stack 00000000 0000000000000000 0000000000000000 000000d6 2**0
                  CONTENTS, READONLY
  6 .eh_frame     00000058  0000000000000000  0000000000000000  000000d8  2**3
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 memoire.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l    d  .rodata        0000000000000000 .rodata
0000000000000000 l    d  .note.GNU-stack        0000000000000000 .note.GNU-stack
0000000000000000 l    d  .eh_frame      0000000000000000 .eh_frame
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000000000 g     O .rodata        0000000000000004 a
0000000000000000 g     O .data  000000000000000c b
0000000000000004       O *COM*  0000000000000004 c
0000000000000000 g     F .text  000000000000001f f
000000000000001f g     F .text  0000000000000038 main
0000000000000000         *UND*  0000000000000000 _GLOBAL_OFFSET_TABLE_
0000000000000000         *UND*  0000000000000000 cos
0000000000000000         *UND*  0000000000000000 printf

RELOCATION RECORDS FOR [.text]:
OFFSET           TYPE              VALUE
000000000000003b R_X86_64_PLT32    cos-0x0000000000000004
0000000000000042 R_X86_64_PC32     .rodata
000000000000004c R_X86_64_PLT32    printf-0x0000000000000004

RELOCATION RECORDS FOR [.eh_frame]:
OFFSET           TYPE              VALUE
0000000000000020 R_X86_64_PC32     .text
0000000000000040 R_X86_64_PC32     .text+0x000000000000001f
```

BSS On observe que le segment `bss` occupe une taille nulle :

```
  2 .bss 00000000 0000000000000000 0000000000000000 000000a4 2**0
```

C'est parce que, usuellement, la section `bss` n'existe pas jusqu'à ce que le programme commence son exécution. C'est la raison pour laquelle on ne peut pas retouver son contenu statiquement avec `objdump` alors qu'on s'attend à y trouver `c`

La variable globale non initialisée `c` sera contenue dans le segment `bss` lorsque le programme sera exécutée. On quand même a une infomation à ce propos :

```
0000000000000004       O *COM*  0000000000000004 c
```

Le symbole `COM` désigne des varaibles globales non initialisées : excatement ce qu'est `c`.

.data La variable globale `b` est initialisée et apparaît logiquement dans le segment `data`.

```
0000000000000000 g     O .data  000000000000000c b
```

.rodata La constante a est initialsée et apparaît donc dans le segment `.rodata` :

```
0000000000000004       O *COM*  0000000000000004 c
```

text Le code des fonctions comme `main` ou `f` est dans le fichier objet binaire. On le retrouve donc dans le segment de code `.text` :

```
0000000000000000 g     F .text  000000000000001f f
000000000000001f g     F .text  0000000000000038 main
```

Les symboles extérieurs Le symbole `UND` désigne un lien externe, un identificateur non défini explicitement dans le fichier objet courant.

```
0000000000000000         *UND*  0000000000000000 cos
0000000000000000         *UND*  0000000000000000 printf
```

La fonction `cos` est en effet définie dans la bibliothèque mathématique et `printf` vient de la bibliothèque d'entrées-sorties.