

# Compilation, édition de liens, fichiers objets, bibliothèques statiques et dynamiques

Nous allons créer un fichier source de bibliothèque, comportant plusieurs fonctions, mais ne comportant pas de fonction main. Ce fichier source de bibliothèque pourra être compilé selon 3 niveaux différents :

- 1 - en un fichier objet (extension .o)
- 2 - en un fichier de bibliothèque statique (extension .a)
- 3 - en un fichier de bibliothèque dynamique (extension .so)

Nous verrons ensuite comment un programme principal en C (comportant une fonction main) peut faire appel aux différentes fonctions de la bibliothèque.

## Les fichiers sources

Voici le code source toto.c de la bibliothèque toto :

```
#include <stdio.h>

int ma_variable=5;

void ma_fonction()
{
    printf("Bienvenue dans Ma Fonction !\n");
}

int mon_addition(int i,int j)
{
    printf("Bienvenue dans Mon Addition !\n");
    return (i+j);
}
```

Et voici son fichier d'en-tête toto.h, ne déclarant que les prototypes des fonctions et des variables globales :

```
int ma_variable;
void ma_fonction();
int mon_addition(int i,int j);
```

Remarques :

- le fichier d'en-tête toto.h associé au source toto.c n'est pas généré automatiquement : il faut l'écrire "à la main"
- dans la liste des paramètres passés aux fonctions, il faut préciser explicitement le type de chaque variables, même si plusieurs paramètres sont de même type. Exemple : `int i,j` ne marche pas. Il faut écrire `int i, int j`

Voici le code source titi.c du programme principal. Ce programme invoque le fichier d'en-tête toto.h de la bibliothèque toto, et fait appel aux fonctions et aux variables globales de cette bibliothèque :

```
#include <stdio.h>
#include "toto.h"

int main(void)
{
    printf("---- Début du programme principal ----\n");
    ma_fonction();
    printf("Ma_variable vaut : %i\n",ma_variable);
    printf("6+7=%i\n",mon_addition(6,7));
    printf("---- Fin du programme principal ----\n");
    return 0;
}
```

Remarque :

`#include "toto.h"` indique que le fichier d'en-tête toto.h se trouve dans le répertoire courant (le répertoire de titi.c). Si on met `#include <toto.h>`, alors gcc cherchera le fichier d'en-tête toto.h dans le répertoire `/usr/include`

# La compilation des fichiers sources

## 1 - Création et utilisation d'un fichier objet

Pour compiler le fichier toto.c en un fichier toto.o, on utilise le paramètre -c de gcc, qui indique à gcc d'arrêter son travail après la compilation (sans appeler l'éditeur de lien) :

```
gcc -c toto.c
```

On obtient alors le fichier objet toto.o

Le programme **objdump** permet de savoir quelles fonctions sont présentes dans le fichier objet toto.o. Si on tape **objdump -t toto.o**, on obtient la table des symboles du fichier objet :

```
toto.o:      file format elf32-i386

SYMBOL TABLE:
00000000 l    df *ABS* 00000000 toto.c
00000000 l     d  .text 00000000
00000000 l     d  .data 00000000
00000000 l     d  .bss 00000000
00000000 l     d  .rodata 00000000
00000000 l     d  .note.GNU-stack 00000000
00000000 l     d  .comment 00000000
00000000 g     O  .data 00000004 ma_variable
00000000 g     F  .text 00000018 ma_fonction
00000000      *UND* 00000000 printf
00000018 g     F  .text 0000001e mon_addition
```

Comment compiler le programme principal titi.c ?

Commençons par essayer une compilation simple :

```
gcc titi.c
```

On obtient alors, sans grande surprise, l'erreur suivante de la part de l'éditeur de lien :

```
/tmp/ccIbKgIE.o(.text+0x21): In function `main':
: undefined reference to `ma_fonction'
/tmp/ccIbKgIE.o(.text+0x43): In function `main':
: undefined reference to `mon_addition'
collect2: ld returned 1 exit status
```

Cette erreur indique que le code des fonctions ma\_fonction et mon\_addition n'ont pas été trouvé par l'éditeur de lien. C'est normal car le code de ces fonctions est dans la librairie toto, et que seul le fichier d'en-tête de cette librairie a été indiqué dans le programme source : gcc ne connaît donc pas le corps des fonctions.

Pour indiquer à gcc d'utiliser le fichier objet toto.o lors de l'édition de lien, la ligne de commande est :

```
gcc titi.c toto.o
```

Il n'y a alors plus d'erreur de lien, et on obtient un fichier a.out exécutable. Si on lance a.out, on obtient

```
---- Début du programme principal ----
Bienvenue dans Ma Fonction !
Ma_variable vaut : 5
Bienvenue dans Mon Addition !
6+7=13
---- Fin du programme principal ----
```

Le programme titi.c a donc réussi à utiliser les fonctions ma\_fonction et mon\_addition dont le code est dans le fichier objet toto.o. Remarque : la déclaration de la variable globale ma\_variable est présente (et suffisante) dans le fichier d'en-tête toto.h.

## 2 - Création et utilisation d'une librairie statique

Sous Linux, un fichier de librairie statique porte l'extension .a

Pour convertir le fichier objet toto.o en une librairie statique libtoto.a, il faut utiliser les 2 lignes de commande suivantes :

```
ar cr libtoto.a toto.o
ranlib libtoto.a
```

On obtient un fichier **libtoto.a**, dont la table des symboles donnée par **objdump -t libtoto.a** est :

```
toto.o:          file format elf32-i386

SYMBOL TABLE:
00000000 l      df *ABS*  00000000 toto.c
00000000 l      d    .text  00000000
00000000 l      d    .data  00000000
00000000 l      d    .bss   00000000

00000000 l      d    .rodata      00000000
00000000 l      d    .note.GNU-stack  00000000
00000000 l      d    .comment    00000000
00000000 g      O    .data  00000004 ma_variable
00000000 g      F    .text  00000018 ma_fonction
*UND*  00000000 printf
00000018 g      F    .text  0000001e mon_addition
```

Pour compiler titi.c en faisant appel aux fonctions de la librairie statique libtoto.a, il suffit de passer en paramètre à gcc le nom du fichier de la librairie statique (exactement comme pour un fichier objet) :

```
gcc titi.c libtoto.a
```

On obtient alors un fichier a.out exécutable. Si on lance a.out, on obtient :

```
---- Début du programme principal ----
Bienvenue dans Ma Fonction !
Ma_variable vaut : 5
Bienvenue dans Mon Addition !
6+7=13
---- Fin du programme principal ----
```

Remarque : dans les 2 cas précédents (utilisation d'un fichier objet et utilisation d'une librairie statique), le code des fonctions ma\_fonction et mon\_addition provenant initialement de la librairie toto.c a été introduit dans le fichier exécutable a.out.

Avantage : le fichier a.out n'est plus dépendant des librairies installées dans le système  
Inconvénient : si on modifie la librairie toto.c, il faudra alors recompiler titi.c pour obtenir un nouveau programme exécutable a.out utilisant la nouvelle librairie

## 3 - Création et utilisation d'une librairie dynamique

Sous Linux, un fichier de librairie dynamique porte l'extension .so

Pour convertir le fichier objet toto.o en une librairie dynamique libtoto.so, la ligne de commande est :

```
gcc -o libtoto.so -shared toto.o
```

On obtient alors un fichier de librairie dynamique libtoto.so

La commande **objdump -t libtoto.so** permet d'obtenir la table des symboles de la librairie dynamique :

```
libtoto.so:      file format elf32-i386

SYMBOL TABLE:
000000b4 l      d    .hash  00000000
000001f0 l      d    .dynsym      00000000
```

00000470	1	d	.dynstr	00000000	
00000532	1	d	.gnu.version	00000000	
00000584	1	d	.gnu.version_r	00000000	
000005b4	1	d	.rel.dyn	00000000	
000005fc	1	d	.rel.plt	00000000	
0000060c	1	d	.init	00000000	
00000624	1	d	.plt	00000000	
00000660	1	d	.text	00000000	
000007b4	1	d	.fini	00000000	
000007e0	1	d	.rodata	00000000	
00000820	1	d	.eh_frame	00000000	
00001824	1	d	.data	00000000	
00001830	1	d	.dynamic	00000000	
000018f8	1	d	.ctors	00000000	
00001900	1	d	.dtors	00000000	
00001908	1	d	.jcr	00000000	
0000190c	1	d	.got	00000000	
0000192c	1	d	.bss	00000000	
00000000	1	d	.comment	00000000	
00000000	1	d	.debug_aranges	00000000	
00000000	1	d	.debug_info	00000000	
00000000	1	d	.debug_abbrev	00000000	
00000000	1	d	.debug_line	00000000	
00000000	1	d	*ABS*	00000000	
00000000	1	d	*ABS*	00000000	
00000000	1	d	*ABS*	00000000	
00000000	1	df	*ABS*	00000000	/home/gb/rpm/BUILD/glibc-
.3.3/build-i586-linux/csu/crti.S					
00000000	1	df	*ABS*	00000000	/home/gb/rpm/BUILD/glibc-
.3.3/build-i586-linux/csu/defs.h					
00000000	1	df	*ABS*	00000000	initfini.c
00000000	1	df	*ABS*	00000000	/home/gb/rpm/BUILD/glibc-
.3.3/build-i586-linux/csu/crti.S					
00000000	1	df	*ABS*	00000000	<command line>
00000000	1	df	*ABS*	00000000	/home/gb/rpm/BUILD/glibc-
.3.3/build-i586-linux/config.h					
00000000	1	df	*ABS*	00000000	<command line>
00000000	1	df	*ABS*	00000000	<built-in>
00000000	1	df	*ABS*	00000000	/home/gb/rpm/BUILD/glibc-
.3.3/build-i586-linux/csu/crti.S					
00000660	1	F	.text	00000000	call_gmon_start
00000000	1	df	*ABS*	00000000	crtstuff.c
000018f8	1	O	.ctors	00000000	__CTOR_LIST__
00001900	1	O	.dtors	00000000	__DTOR_LIST__
00001908	1	O	.jcr	00000000	__JCR_LIST__
00001828	1	O	.data	00000000	p.0
0000192c	1	O	.bss	00000001	completed.1
00000690	1	F	.text	00000000	__do_global_dtors_aux
00000700	1	F	.text	00000000	frame_dummy
00000000	1	df	*ABS*	00000000	crtstuff.c
000018fc	1	O	.ctors	00000000	__CTOR_END__
00001904	1	O	.dtors	00000000	__DTOR_END__
00000820	1	O	.eh_frame	00000000	__FRAME_END__
00001908	1	O	.jcr	00000000	__JCR_END__
00000780	1	F	.text	00000000	__do_global_ctors_aux
00000000	1	df	*ABS*	00000000	/home/gb/rpm/BUILD/glibc-
2.3.3/build-i586-linux/csu/crtn.S					
00000000	1	df	*ABS*	00000000	/home/gb/rpm/BUILD/glibc-
2.3.3/build-i586-linux/csu/defs.h					
00000000	1	df	*ABS*	00000000	initfini.c
00000000	1	df	*ABS*	00000000	/home/gb/rpm/BUILD/glibc-
2.3.3/build-i586-linux/csu/crtn.S					
00000000	1	df	*ABS*	00000000	<command line>
00000000	1	df	*ABS*	00000000	/home/gb/rpm/BUILD/glibc-
2.3.3/build-i586-linux/config.h					
00000000	1	df	*ABS*	00000000	<command line>

```

00000000 l      df *ABS*  00000000      <built-in>
00000000 l      df *ABS*  00000000      /home/gb/rpm/BUILD/glibc-
2.3.3/build-i586-linux/csu/crti.S
00000000 l      df *ABS*  00000000      toto.c
00001824 l      O .data  00000000      .hidden __dso_handle
00000760 g      F .text  0000001e      mon_addition
00001830 g      O *ABS*  00000000      _DYNAMIC
0000182c g      O .data  00000004      ma_variable
0000060c g      F .init  00000000      _init
0000192c g      *ABS*  00000000      __bss_start
00000000      F *UND*  0000002b      printf@@GLIBC_2.0
000007b4 g      F .fini  00000000      _fini
00000000 w      F *UND*  0000008f      __cxa_finalize@@GLIBC_2.1.3
0000192c g      *ABS*  00000000      _edata
0000190c g      O *ABS*  00000000      _GLOBAL_OFFSET_TABLE_
00001930 g      *ABS*  00000000      _end
00000748 g      F .text  00000018      ma_fonction
00000000 w      *UND*  00000000      _Jv_RegisterClasses
00000000 w      *UND*  00000000      __gmon_start__

```

Pour compiler titi.c en utilisant la librairie dynamique libtoto.so il faut indiquer à gcc 2 informations :

- 1 - quelle librairie dynamique l'éditeur de lien doit-il utiliser : paramètre `-l`
- 2 - dans quel répertoire se trouve le fichier .so de la librairie dynamique : paramètre `-L`

La compilation de titi.c en utilisant la librairie dynamique toto se fait alors avec la ligne de commande suivante :

```
gcc -ltoto -L. titi.c
```

`ltoto` indique à l'éditeur de lien d'utiliser le fichier de librairie dynamique libtoto.so  
`-L.` indique que le fichier de librairie dynamique se trouve dans le répertoire courant

Si le paramètre `-L` n'est pas passé à gcc, l'éditeur de lien recherche alors les fichiers .so de librairie dynamiques dans le répertoire `/usr/lib`

## 4 - Remarque et récapitulation concernant la compilation avec gcc

`gcc toto.c` -> compilation + édition de lien -> on obtient un fichier exécutable `a.out`

`gcc -c toto.c` -> compilation seulement -> on obtient un fichier objet `toto.o`

`gcc toto.o` -> édition de lien seulement (utilisation du linqueur LD) -> on obtient un exécutable `a.out`

Pour obtenir un exécutable, il faut une et une seule fonction main dans l'ensemble des sources et des librairies. Si un programme fait appel à des fonctions extérieures qui ne sont pas définies dans son code source, il faut l'indiquer à gcc sur la ligne de commande. Exemple : le programme titi.c fait appel à des fonctions contenues dans toto.c. Il y a alors 4 manières de compiler titi.c :

- 1 - Avec le fichier source de toto.c : `gcc titi.c toto.c`
- 2 - Avec le fichier objet toto.o de toto.c : `gcc titi.c toto.o`
- 3 - Avec la librairie statique libtoto.a de toto.c : `gcc titi.c libtoto.a`
- 4 - Avec la librairie dynamique libtoto.so de toto.c : `gcc titi.c -ltoto`

Ces 4 lignes de commande génèrent un exécutable a.out dont le point d'entrée est la fonction main() de titi.c. Enfin, le programme objdump permet d'aller voir les symboles présents dans :

- un fichier objet
- une librairie statique
- une librairie dynamique
- un fichier exécutable ELF32 (a.out)

## 5 - Utilitaires supplémentaires pour l'administration des bibliothèques partagées

- la commande **ldd** `<prog>` permet de savoir quelle(s) bibliothèque(s) partagée(s) le programme `<prog>` utilise ;
- la commande **ldconfig** régénère tous les liens symboliques dans les répertoires contenant des bibliothèques partagées (ces répertoires sont répertoriés dans le fichier `/etc/ld.so.conf` ). Les répertoires `/lib` et `/usr/lib` sont pris en charge par défaut. L'usage des liens symboliques permet de conserver plusieurs versions de chaque bibliothèque, tout en passant de l'une à l'autre en un seul appel système : ainsi la mise à jour d'une bibliothèque peut se faire sans arrêter le système ! **ldconfig -v** affiche l'ensemble des librairies installées dans le système.
- La commande **export LD\_librarie\_PATH=`pwd`** permet de rajouter le répertoire courant au chemin de recherche des librairie partagées (exactement comme l'option **-L** de gcc)
- La commande **ltrace** permet de suivre les appels à des fonctions de bibliothèques dynamiques effectués par un programme, exactement de la même façon que **strace** le fait pour les appels système : de même que **strace**, c'est un outil précieux pour étudier le comportement d'un programme qui ne fonctionne pas correctement.

Exemple : si le programme **a.out** a été compilé en utilisant la bibliothèque partagée **libtoto.so**, alors la commande **ltrace a.out** renvoie :

```
__libc_start_main(0x0804850c, 1, 0xbffff6c4, 0x08048580, 0x080485d0 <unfinished ...>
printf("---- D\351but du programme princ"...---- Début du programme principal ----
)
= 39
ma_fonction(0x40016640, 0x080485d0, 0xbffff698, 0x4003e95d, 1 <unfinished ...>
printf("Bienvenue dans Ma Fonction !\n"Bonjour dans Ma Fonction !
)
= 29
<... ma_fonction resumed> )
= 29
printf("Ma_variable vaut : %i\n", 5Ma_variable vaut : 5
)
= 21
mon_addition(6, 7, 0xbffff638, 0x08048531, 0x40016640 <unfinished ...>
printf("Bienvenue dans Mon Addition !\n"Bonjour dans Mon Addition !
)
= 30
<... mon_addition resumed> )
= 13
printf("6+7=%i\n", 136+7=13
)
= 7
printf("---- Fin du programme principal "...---- Fin du programme principal ----
)
= 37
+++ exited (status 0) +++
```

La commande **ltrace -l libtoto.so a.out** renvoie :

```
---- Début du programme principal ----
ma_fonction(0x40016640, 0x080485d0, 0xbffff698, 0x4003e95d, 1Bonjour dans Ma Fonction !
)
= 29
Ma_variable vaut : 5
mon_addition(6, 7, 0xbffff638, 0x08048531, 0x40016640Bonjour dans Mon Addition !
)
= 13
6+7=13
---- Fin du programme principal ----
+++ exited (status 0) +++
```

Et la commande **ldd a.out** renvoie :

```
linux-gate.so.1 => (0xffffe000)
libtoto.so => /usr/lib/libtoto.so (0x40027000)
libc.so.6 => /lib/tls/libc.so.6 (0x40029000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

On voit bien à travers cette dernière commande que le programme **a.out** fait appel à la librairie partagée **libtoto.so**.

# Utilisation de bibliothèques existantes dans le système

La plupart des fonctions élémentaires dont un programmeur a besoin sont généralement déjà programmées dans des bibliothèques installées. Par exemple, pour afficher une image JPG il n'est nullement besoin de ré-écrire un décodeur JPG : il suffit de faire appel à une fonction d'affichage d'image JPG. Oui, mais laquelle, et dans quelle bibliothèque se trouve-t-elle ? Pareil pour lire un fichier MP3, ou encore pour accéder au réseau.

## 1 - Comment connaître les bibliothèques disponibles dans le système ?

La commande `ldvconfig -v` affiche l'ensemble des bibliothèques installées dans le système. Recherchons une bibliothèque contenant des routines relatives au traitement des images JPEG :

```
ldvconfig -v | grep jpeg
```

On constate que la bibliothèque `libjpeg.so` est installée dans le système. Où se trouve le fichier de bibliothèque dynamique `libjpeg.so` ?

```
whereis libjpeg.so
```

On obtient comme résultat l'emplacement `/usr/lib/libjpeg.so`

## 2 - Comment connaître les fonctions disponibles dans une bibliothèque du système ?

Quelles fonctions contient cette bibliothèque `libjpeg.so` ? Pour cela, affichons sa table de symboles grâce à `objdump` :

```
objdump -T /usr/lib/libjpeg.so
```

Parmi tous les résultats affichés, les fonctions de la bibliothèque sont précédées du mot `Base`. Filtrons la sortie de `objdump` grâce à `grep` afin d'affiner le résultat :

```
objdump -T /usr/lib/libjpeg.so | grep Base
```

On obtient alors la liste des 110 fonctions disponibles dans la bibliothèque dynamique `libjpeg.so` :

00009320	g	DF	.text	000002b5	Base	jpeg_gen_optimal_table
00002d30	g	DF	.text	0000028b	Base	jpeg_copy_critical_parameters
000068d0	g	DF	.text	00000103	Base	jinit_c_prep_controller
0000d640	g	DF	.text	0000007c	Base	jinit_input_controller
0000bf00	g	DF	.text	000000ae	Base	jpeg_read_scanlines
0001afa0	g	DF	.text	00000020	Base	jpeg_get_small
000147c0	g	DF	.text	000004c2	Base	jpeg_idct_float
000039e0	g	DF	.text	000003fd	Base	jpeg_set_colorspace
000035e0	g	DF	.text	0000003a	Base	jpeg_quality_scaling
00003ef0	g	DF	.text	0000036b	Base	jpeg_simple_progression
0000ae60	g	DF	.text	00000245	Base	jpeg_fdct_ifast
0000ff00	g	DF	.text	00000394	Base	jpeg_make_d_derived_tbl
0001e4fc	g	DO	*ABS*	00000000	Base	_DYNAMIC
00015b00	g	DF	.text	00000040	Base	jpeg_idct_1xl
000103f0	g	DF	.text	0000014a	Base	jpeg_huff_decode
00014c90	g	DF	.text	00000764	Base	jpeg_idct_islow
0000cf00	g	DF	.text	00000064	Base	jinit_master_decompress
0000c610	g	DF	.text	000002ce	Base	jpeg_calc_output_dimensions
00003550	g	DF	.text	00000082	Base	jpeg_set_linear_quality
000029a0	g	DF	.text	0000001f	Base	jpeg_write_m_byte
00019ac0	g	DF	.text	0000002d	Base	jround_up
00003800	g	DF	.text	00000149	Base	jpeg_set_defaults
0001b050	g	DF	.text	00000003	Base	jpeg_mem_init
00019aa0	g	DF	.text	00000011	Base	jdiv_round_up
0000a6d0	g	DF	.text	0000005f	Base	jinit_phuff_encoder
0001b030	g	DF	.text	00000019	Base	jpeg_open_backing_store

0000b780	g	DF .text	00000020	Base	jpeg_abort_decompress
0000bce0	g	DF .text	00000117	Base	jpeg_start_decompress
00007970	g	DF .text	000001ed	Base	jinit_color_converter
00019a00	g	DF .text	0000003b	Base	jpeg_destroy
00019af0	g	DF .text	0000006c	Base	jcopy_sample_rows
0000ce60	g	DF .text	00000092	Base	jpeg_new_colormap
0000c1c0	g	DF .text	000000ef	Base	jpeg_read_coefficients
00019a70	g	DF .text	0000002b	Base	jpeg_alloc_huff_table
00016880	g	DF .text	000001cd	Base	jinit_color_deconverter
00006170	g	DF .text	000000f4	Base	jinit_c_main_controller
00001fd0	g	DF .init	00000000	Base	_init
0001e300	g	DO .data	000001fc	Base	jpeg_std_message_table
000028b0	g	DF .text	0000008c	Base	jpeg_write_marker
000102a0	g	DF .text	0000014f	Base	jpeg_fill_bit_buffer
00019050	g	DF .text	000001c0	Base	jinit_2pass_quantizer
000198a0	g	DF .text	000000d5	Base	jinit_merged_upsampler
00009740	g	DF .text	00000064	Base	jinit_huff_encoder
0000b0b0	g	DF .text	00000234	Base	jpeg_fdct_float
0000b760	g	DF .text	00000020	Base	jpeg_destroy_decompress
00019e00	g	DF .text	00000077	Base	jpeg_std_error
0000b640	g	DF .text	0000011f	Base	jpeg_CreateDecompress
0001b960	g	DO .rodata	00000140	Base	jpeg_natural_order
0001b060	g	DF .text	00000001	Base	jpeg_mem_term
00012480	g	DF .text	0000012c	Base	jinit_d_main_controller
00013da0	g	DF .text	000000f5	Base	jinit_d_post_controller
00005fa0	g	DF .text	0000007d	Base	jinit_marker_writer
0000b2f0	g	DF .text	0000034a	Base	jpeg_fdct_islow
00002bd0	g	DF .text	000000d9	Base	jpeg_write_raw_data
000178b0	g	DF .text	000000ee	Base	jinit_lpass_quantizer
0001ae20	g	DF .text	0000017c	Base	jinit_memory_mgr
0000bbb0	g	DF .text	00000047	Base	jpeg_has_multiple_scans
000141d0	g	DF .text	000005ec	Base	jpeg_idct_ifast
0001b020	g	DF .text	00000005	Base	jpeg_mem_available
00019ba0	g	DF .text	00000030	Base	jzero_far
00014110	g	DF .text	000000b1	Base	jinit_inverse_dct
0000ba80	g	DF .text	000000d7	Base	jpeg_consume_input
00015850	g	DF .text	000002a1	Base	jpeg_idct_2x2
0000bb60	g	DF .text	00000047	Base	jpeg_input_complete
00013870	g	DF .text	00000182	Base	jinit_d_coef_controller
000026d0	g	DF .text	00000054	Base	jpeg_suppress_tables
00010b20	g	DF .text	00000068	Base	jinit_huff_decoder
000026b0	g	DF .text	00000020	Base	jpeg_abort_compress
0001e748	g	D *ABS*	00000000	Base	__bss_start
0000fcd0	g	DF .text	00000071	Base	jpeg_set_marker_processor
00003450	g	DF .text	000000fc	Base	jpeg_add_quant_table
00019980	g	DF .text	00000074	Base	jpeg_abort
<b>0000b9d0</b>	<b>g</b>	<b>DF .text</b>	<b>000000a2</b>	<b>Base</b>	<b>jpeg_read_header</b>
00002a30	g	DF .text	00000092	Base	jpeg_start_compress
00019a40	g	DF .text	0000002b	Base	jpeg_alloc_quant_table
00002690	g	DF .text	00000020	Base	jpeg_destroy_compress
000029c0	g	DF .text	00000062	Base	jpeg_write_tables
00003950	g	DF .text	0000008a	Base	jpeg_default_colorspace
00011be0	g	DF .text	00000098	Base	jinit_phuff_decoder
0001afe0	g	DF .text	00000020	Base	jpeg_get_large
0000bc00	g	DF .text	000000d2	Base	jpeg_finish_decompress
0000f950	g	DF .text	00000154	Base	jpeg_resync_to_restart
0000c4c0	g	DF .text	000000b6	Base	jpeg_stdio_src
00002580	g	DF .text	00000102	Base	jpeg_CreateCompress
0001b0a4	g	DF .fini	00000000	Base	_fini
0001b000	g	DF .text	00000020	Base	jpeg_free_large
0000ad90	g	DF .text	000000c3	Base	jinit_forward_dct
00004470	g	DF .text	000000fc	Base	jinit_compress_master
0000faf0	g	DF .text	0000008c	Base	jinit_marker_reader
00002730	g	DF .text	00000174	Base	jpeg_finish_compress
0000bfb0	g	DF .text	000000c1	Base	jpeg_read_raw_data
0001e748	g	D *ABS*	00000000	Base	_edata



0001e5d8 g	DO	*ABS*	00000000	Base	_GLOBAL_OFFSET_TABLE_
0001e74c g	D	*ABS*	00000000	Base	_end
00015400 g	DF	.text	0000044f	Base	jpeg_idct_4x4
00002940 g	DF	.text	00000054	Base	jpeg_write_m_header
000084e0 g	DF	.text	0000024a	Base	jinit_downsampler
00003620 g	DF	.text	0000003a	Base	jpeg_set_quality
00004400 g	DF	.text	0000006f	Base	jpeg_stdio_dest
0000fb80 g	DF	.text	0000014a	Base	jpeg_save_markers
00002ad0 g	DF	.text	000000fb	Base	jpeg_write_scanlines
00019b60 g	DF	.text	00000033	Base	jcopy_block_row
00002cb0 g	DF	.text	00000076	Base	jpeg_write_coefficients
0001afc0 g	DF	.text	00000020	Base	jpeg_free_small
0000c110 g	DF	.text	000000ad	Base	jpeg_finish_output
00005280 g	DF	.text	00000135	Base	jinit_c_master_control
0000c080 g	DF	.text	00000087	Base	jpeg_start_output
00007310 g	DF	.text	00000120	Base	jinit_c_coef_controller
00008950 g	DF	.text	0000026a	Base	jpeg_make_c_derived_tbl
000160e0 g	DF	.text	000002dd	Base	jinit_upsampler

Parmi toutes ces fonctions on trouve la fonction **jpeg\_read\_header** (en gras ci-dessus) qui permet de lire l'en-tête d'un fichier JPEG, afin de connaître, par exemple, la résolution en pixels de l'image. Avant d'utiliser cette fonction pour afficher la résolution d'une image JPEG, il nous faut trouver dans le système le fichier d'en-tête (avec l'extension .h) relatif à cette librairie partagée **libjpeg.so**.

### 3 - Où retrouver le fichier d'en-tête relatif à une librairie ?

Les fichiers d'en-tête relatifs aux librairies partagées du système, et que l'utilisateur peut utiliser dans ses programmes en C, sont généralement enregistrés dans le répertoire `/usr/include`. Recherchons dans ce répertoire un fichier dont le nom contient jpeg :

```
ls /usr/include/*jpeg*.h
```

On obtient alors le fichier `/usr/include/jpeglib.h`

ATTENTION : le fichier de librairie partagée s'appelle **libjpeg.so** mais le fichier d'en-tête s'appelle **jpeglib.h**

Pour utiliser les fonctions et les types de la librairie **libjpeg.so** il suffit de rajouter `#include <jpeglib.h>` au début du programme.

### 4 - Exemple d'utilisation de la librairie libjpeg.so :

Le programme en C suivant affiche la résolution d'un fichier image toto.jpg :

```
/*          Ce programme test.c affiche la résolution          */
/* d'une image toto.jpg en utilisant la librairie libjpeg.so */
#include <stdio.h>
#include <jpeglib.h>

int main(void)
{
    struct jpeg_decompress_struct cinfo;
    struct jpeg_error_mgr jerr;
    FILE *file;
    char *fichier="toto.jpg";

    cinfo.err = jpeg_std_error(&jerr);
    jpeg_create_decompress(&cinfo);

    if ((file=fopen(fichier,"rb"))==NULL)
    {
        fprintf(stderr,"Erreur : impossible d'ouvrir le fichier texture.jpgn");
        exit(1);
    }
}
```

```
jpeg_stdio_src(&cinfo, file);
jpeg_read_header(&cinfo, TRUE);
printf("La largeur de l'image est %i pixels\n",cinfo.image_width);
printf("La hauteur de l'image est %i pixels\n",cinfo.image_height);
return (0);
}
```

Pour le compiler on utilisera la ligne de commande suivante (l'éditeur de lien fait appel à la librairie `libjpeg.so`) :

```
gcc -ljpeg test.c
```

Et si on lance l'exécutable `a.out` obtenu, on obtient à l'écran :

```
La largeur de l'image est 400 pixels
La hauteur de l'image est 342 pixels
```

## Où trouver de l'aide

Les fichiers d'aides (.doc) des différentes librairies installées dans le système se trouvent dans le répertoire `/usr/share/doc`.

## Pour aller plus loin

### Utilisation de librairies partagées en Perl

Le programme `h2xs` permet de convertir un fichier d'en-tête .h en un module Perl .pm afin qu'un programme Perl puisse utiliser une librairie partagée écrite en C. Utilisons la librairie `toto.c` et son fichier d'entête décrits ci-dessus :

```
h2xs toto.h
```

### Distribution d'une librairie partagée dans un fichier RPM

L'option `-ba` du programme RPM permet de se fabriquer ses propres fichiers RPM, en ligne de commande.

**Retrouvez de nombreux autres articles  
concernant la programmation Linux  
sur le site [www.gecif.net](http://www.gecif.net)**