Unit: VI.
Libraries: creation and use

Herramientas Avanzadas para el Desarrollo de Aplicaciones

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante

lsi Department of Software and Computing Systems

Content

What is a library?

- In a short way we can say that a *library* is a set of resources: subprograms, classes, data, etc. . .
- When we distribute these resources inside a library we are improving ftheir use and reuse.
- Why?: In the case of *source code* we do not need to recompile since this is distributed inside the library in binary mode, already compiled; until now we only knew distributing it in source code mode.
- When using a library we need to *link* our code with that library, in this way we have access to its content.

Why using a binary format?

Several reasons:

- If for using it we need a binary format we avoid the user compiling the code.
- In some cases the compilation process to obtain a library is *costly* and may be not easy.
- In the case of dynamic-link libraries (DLL) we have the advantage of being able to modify them for solving problems without recompiling.

Simple example I

- Let's see an example of a library...
- We have a *monolíthic* code where the main program and the functions used are in a single file.
- It is an application that implements a simple algorithm for strings compression/decompression.
- Invoked in this way:

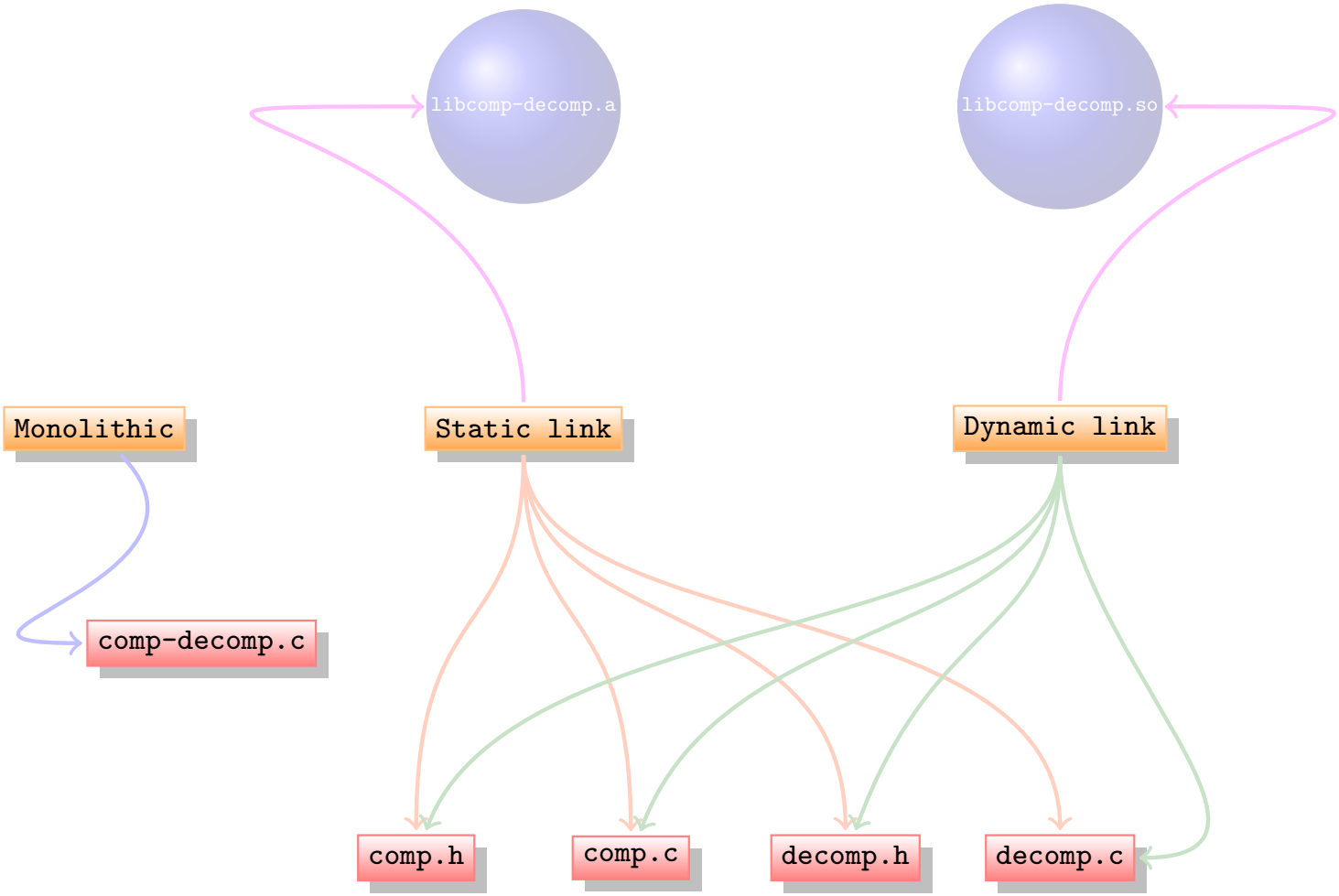
  ```
  comp-decomp -c ccccaasssssssssaaaaaaa
  ```

  produces this output:

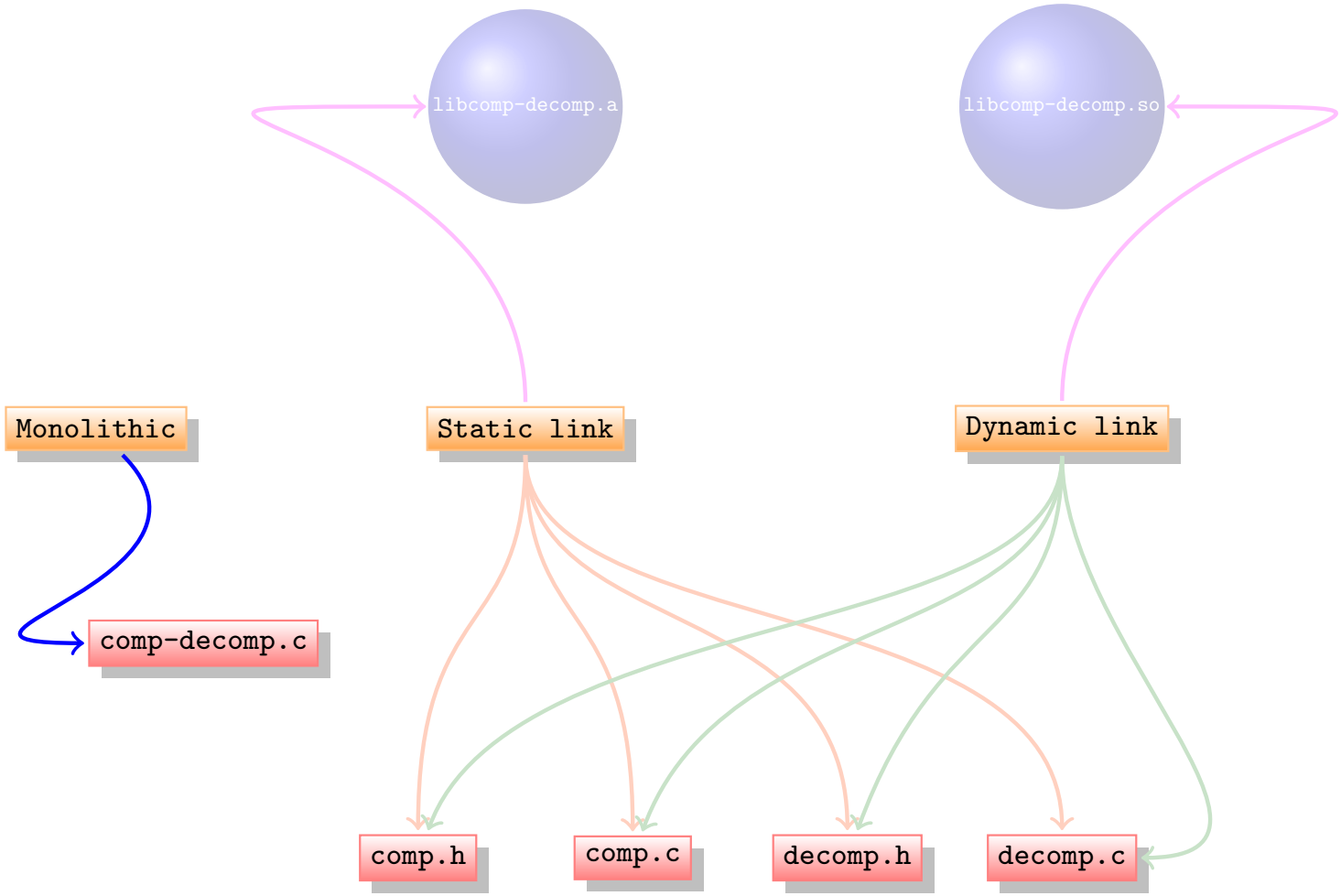
  ```
  Compresion de ‘‘ccccaasssssssssaaaaaaa’’(21) es ‘‘4caa8s7a’’(8)
  ```

- From the same code we are going to create three versions of the application:*monolíthic*, linked with a *static library* and linked with a *dynamic library*.

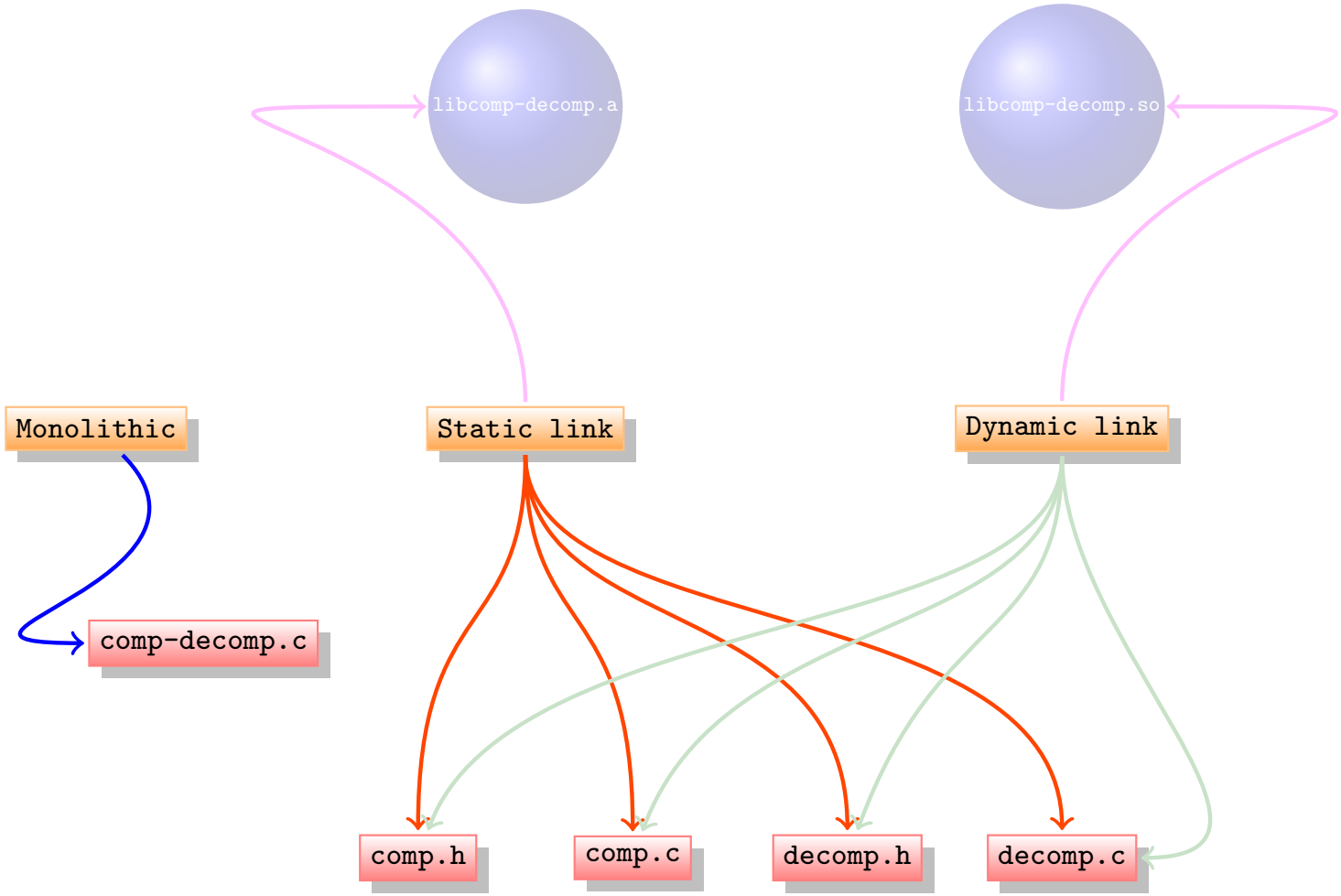
Simple example II



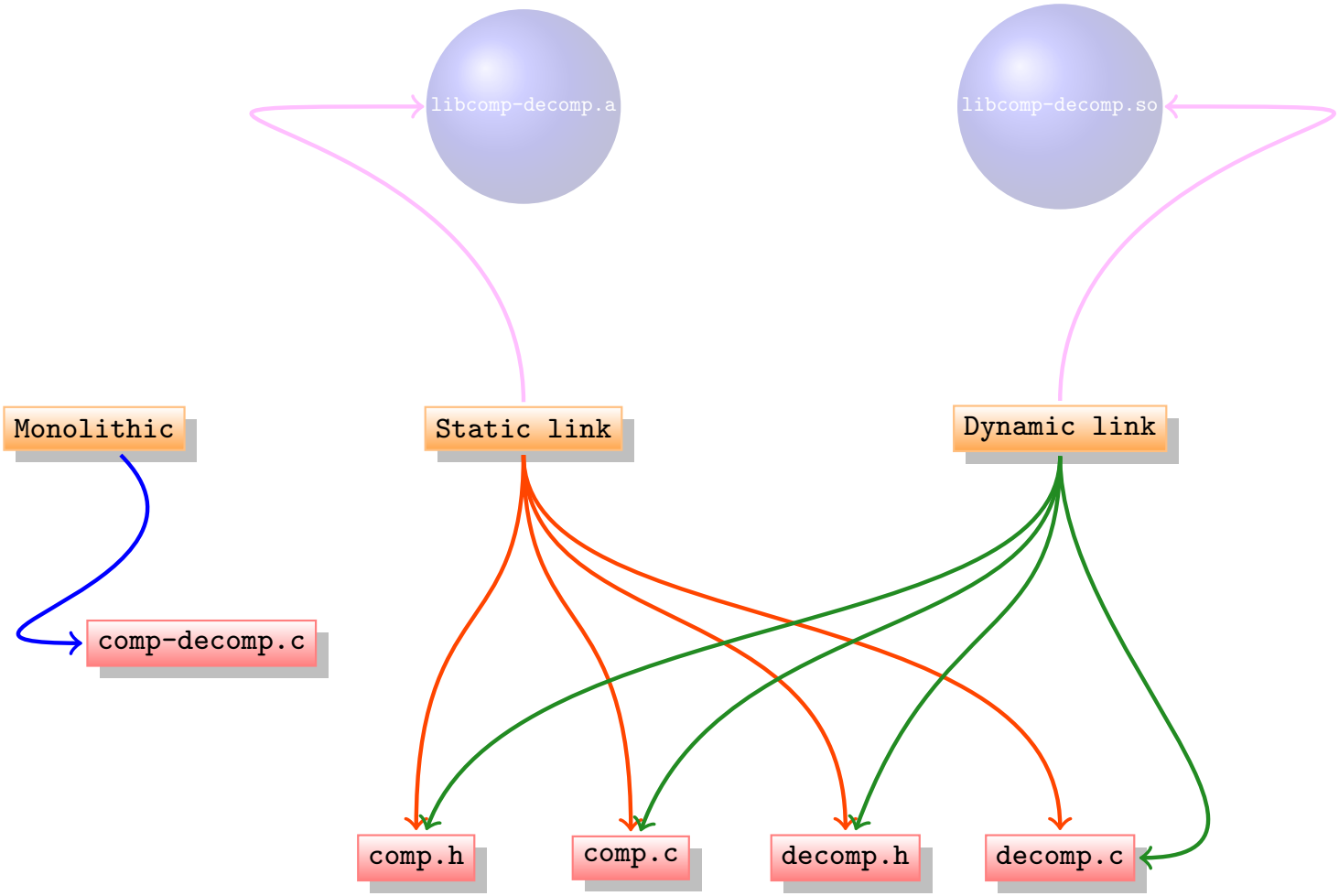
Simple example II



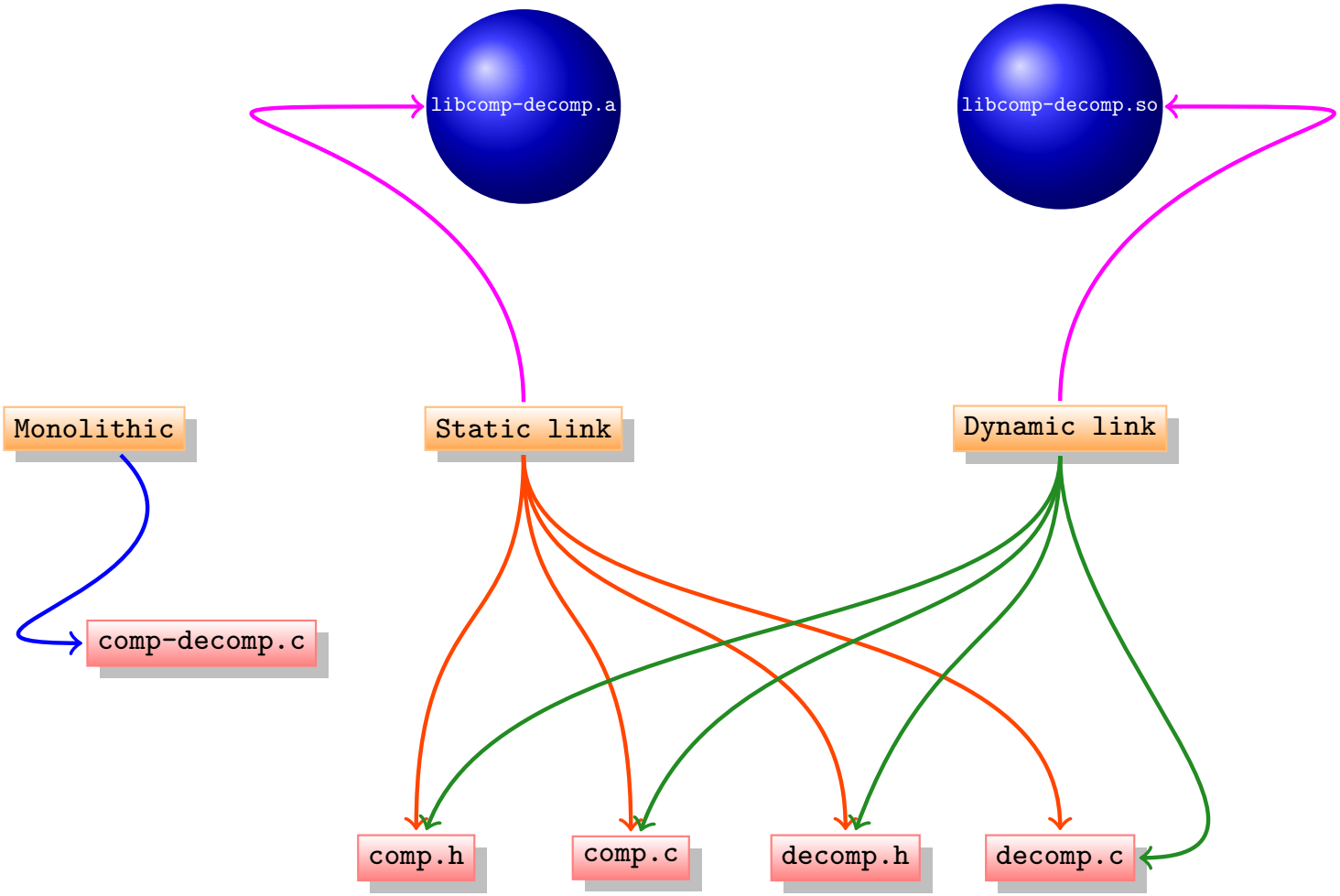
Simple example II



Simple example II



Simple example II



Code for monolithic example I (comp-decomp.c)

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

/* It returns the number of characters equals to the first one of 's'
   */
static int caracteres_iguales (char* s) {
  int l = 0;
  int cont = 1;

  if (s == NULL) return 0;

  l = strlen (s);
  if (l < 2)
    return l;
  else {
    int i = 1;
    while (s[0] == s[i++])
      cont++;
    return cont;
  }
}
```

Code for monolithic example II (comp-decomp.c)

```
1   /* Returns in  'cs' the compressed string of 's'. */
    void comprime (char* s, char* cs) {
3     int l = cont = i = 0;
      char num[5], *s2;
5
      if (s == NULL)  return;
7
      l = strlen (s);
9     if (l < 3) strcat (cs, s);
      else {
11      int csl = 0;
        cont = caracteres_iguales (s);
13      if (cont > 2) {
          sprintf (num, "%d", cont);
15        strcat (cs, num);
          csl = strlen(cs);
17        cs[csl++] = s[0];
          cs[csl] = '\0';
19      } else {
          csl = strlen(cs);
21        for (i = 0; i < cont; i++)
            cs[csl++] = s[0];
23        cs[csl] = '\0';
        }
25
        s2 = &s[cont];
27      comprime (s2, cs);
      }
29  }
```

Code for monolithic example III (comp-decomp.c)

```
1  /* Returns in 's' the decompressed string of 'cs'. */
   void descomprime (char* cs, char* s) {
3    /* por hacer */
     strcat (s, cs);
5  }

7  int main(int argc, char *argv[])
   {
9    char c[100];

11   if (argc != 3) {
       printf("Uso: comp-decomp [-c|-d] cadena\n");
13     return 1;
     } else {
15     if ( (strcmp(argv[1],"-c") != 0) && (strcmp(argv[1],"-d") != 0) ) {
         printf("Uso: comp-decomp [-c|-d] cadena\n");
17       return 2;
       }
19   }
     if (strcmp(argv[1],"-c") == 0) {
21     c[0] = '\0';
       comprime (argv[2], c);
23     printf ("Compresion de \"%s\"(%d) es \"%s\"(%d)\n", argv[2], strlen(argv[2]), c,
             strlen(c));
     }
25   if (strcmp(argv[1],"-d") == 0) {
       c[0] = '\0';
27     descomprime (argv[2], c);
       printf ("Descompresion de \"%s\"(%d) es \"%s\"(%d)\n", argv[2], strlen(argv[2]),
             c, strlen(c));
29   }
     return 0;
31 }
```

Creation of a library

- For the creation of a dynamic-link library or static, first we need to divide the previous code.
- We put in one or several files the functions and/or data structures that provide the main functionality.
- We leave our of these files the code of the main program, which will be in a different file which will 'consume' the code of the library.
- In our case, the division into files provides us the following files:
  - `comp.h` and `comp.c` Contain the code related with the compression function.
  - `decomp.h` and `decomp.c` Contain the code related with the decompression function.
  - `comp-decomp-driver.c` It is the main program in which the arguments used to call it will be analyzed and the corresponding function is called.

comp.h and comp.c I

```c
/* comp.h */

void comprime (char* s, char* cs);
```

```c
/* comp.c */

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

/* It is private, the library does not export it. (modifier static) */
static int caracteres_iguales (char* s) {
  int l = 0, cont = 1;

  if (s == NULL) return 0;
  l = strlen (s);
  if (l < 2) return l;
  else {
    int i = 1;
    while (s[0] == s[i++]) cont++;
    return cont;
  }
}
```

comp.h and comp.c II

```
1  /* comp.c */

3  void comprime (char* s, char* cs) {
     int l = cont = i = 0;
5    char num[5], *s2;

7    if (s == NULL) return;

9    l = strlen (s);
     if (l < 3) strcat (cs, s);
11   else {
       int csl = 0;
13     cont = caracteres_iguales (s);
       if (cont > 2) {
15       sprintf (num, "%d", cont);

17       strcat (cs, num);
         csl = strlen(cs);
19       cs[csl++] = s[0];
         cs[csl] = '\0';
21     } else {
         csl = strlen(cs);
23       for (i = 0; i < cont; i++) cs[csl++] = s[0];
         cs[csl] = '\0';
25     }
       s2 = &s[cont];
27     comprime (s2, cs);
     }
29 }
```

decomp.h and decomp.c

```
1   /* decomp.h */

3   void descomprime (char* s, char* cs);
```

```
1   /* decomp.c */

3   #include <string.h>

5   /* Returns in 's' the decompressed string of 'cs'.
        */
    void descomprime (char* cs, char* s) {
7     /* por hacer */
      strcat (s, cs);
9   }
```

comp-decomp-driver.c

```c
/* comp-decomp-driver.c -- It is the main program. */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "comp.h"
#include "decomp.h"

int main(int argc, char *argv[])
{
  char c[100];

  if (argc != 3) {
    printf("Uso: comp-decomp [-c|-d] cadena\n");
    return 1;
  } else
  if ( (strcmp(argv[1],"-c") != 0) &&
       (strcmp(argv[1],"-d") != 0) )
       {
         printf("Uso: comp-decomp [-c|-d] cadena\n");
         return 2;
       }
  if (strcmp(argv[1],"-c") == 0) {
    c[0] = '\0';
    comprime (argv[2], c);
    printf ("Compresion de \"%s\"(%d) es \"%s\"(%d)\n", argv[2], strlen(argv[2]), c,
          strlen(c));
  }
  if (strcmp(argv[1],"-d") == 0) {
    c[0] = '\0';
    descomprime (argv[2], c);
    printf ("Descompresion de \"%s\"(%d) es \"%s\"(%d)\n", argv[2], strlen(argv[2]),
          c, strlen(c));
  }
  return 0;
}
```

Department of Software and Computing Systems

How do we generate a static-link library?

- We compile the files that compose it in a separated way. We obtain from each of them its '`.o`'.
- With the application '`ar`' -similar to `tar`- we create the file'biblioteca' with extension'`.a`' -**a** of *file*- like this:

```
ar crs libcomp-decomp.a comp.o decomp.o
```

- Where the string '`crs`' represents the options used to call it:
  - `c` Create the file with name: `libcomp-decomp.a`
  - `r` Substitutes -if already exists- in the file '`.a`' each of the files specilied next.
  - `s` Creates an index in the file '`.a`' so it is faster than accessing the files it contains.

How do we link with a static-link library? I

We can do it in several ways:

- Specifying its complete path, in the same way as with a file '`.o`': '`/ruta/hasta/fichero.a`'.
- Additionally, if its name follows the next syntax: libnombre-bib.a, the linker will recognize it and we can put in this way in the linking line: `-lnombre-bib`.
- The *linker* is a different application from the compiler, independent of the programming language used. Traditionally in o.s. of the UNIX family this program is called '`ld`'.

How do we link with a static-link library? II

```
#########################
# Static library        #
#########################################################
comp-decomp-estatico: comp-decomp-driver.o libcomp-decomp.a
    $(CC) -static comp-decomp-driver.o -L . -lcomp-decomp -o comp-decomp-estatico

comp-decomp-driver.o: comp-decomp-driver.c
    $(CC) -c $(CFLAGS) comp-decomp-driver.c

libcomp-decomp.a: comp.o decomp.o
    $(AR) crs libcomp-decomp.a comp.o decomp.o

comp.o: comp.c
    $(CC) -c $(CFLAGS) comp.c

decomp.o: decomp.c
    $(CC) -c $(CFLAGS) decomp.c
```

How do we generate a dynamic-link library?

- In the same way as before we compile the files that compose it separately. We obtain from each file its '`.o`' but we have to use the option `-fpic`. `pic`: Position Independent Code.
- Dynamic-link libraries use the extension '`.so`' in o.s. of the UNIX family. They are equivalent to the '`DLL`' in Windows.
- For generating them we do not need any new program as before, it is enough with the linker providing it the option '`-shared`':

```
gcc -shared -o libcomp-decomp.so comp-pic.o decomp-pic.o
```

How do we link a dynamic-link library?

```
1  ###################################
   # Dynamic-link library #
3  ###################################################################
   comp-decomp-dinamico: comp-decomp-driverdin.o libcomp-decomp.so
5      $(CC) comp-decomp-driverdin.o -L . -lcomp-decomp -o comp-decomp-dinamico

7  comp-decomp-driverdin.o: comp-decomp-driverdin.c
       $(CC) -fpic -c $(CFLAGS) comp-decomp-driverdin.c
9
   libcomp-decomp.so: comp-pic.o decomp-pic.o
11     $(CC) -shared -o libcomp-decomp.so comp-pic.o decomp-pic.o

13 comp-pic.o: comp.c
       $(CC) -c -o comp-pic.o -fpic $(CFLAGS) comp.c
15
   decomp-pic.o: decomp.c
17     $(CC) -c -o decomp-pic.o -fpic $(CFLAGS) decomp.c
```

Useful applications for files '.o', '.a' and '.so' I

They are applications that allow us extracting information of these type of files or even modify them in some way:

- **nm** Lists the symbols inside a binary file. Moreover it gives us certain information of each of them.
- **ranlib** Creates the index in a library done using '`ar`', this means, it has the same function as the option '`s`' of '`ar`'.
- **strip** Deletes certain symbols from the binary file for reducing its size.

Useful applications for files '.o', '.a' and '.so' II

Let's see the output done with '`nm`' on the static-link library previously created:

```
1   Ejecutamos libcomp-decomp.a

3   comp.o:
    0000000000000000 t caracteres_iguales
5   0000000000000078 T comprime
                     U sprintf
7                    U strcat
                     U strlen
9
    decomp.o:
11  0000000000000000 T descomprime
                     U strcat
```

Useful applications for files '.o', '.a' and '.so' III

- Which effect has '`strip`' in our binary files?
- Basically the final size of them, compare:

  before `ls -l libcomp-decomp.a` $= 3838$ bytes
  after `strip libcomp-decom.a`,
  `ls -l libcomp-decomp.a` $= 2536$ bytes

- In general we can use `strip` with any binary file generated in the process of compilation/linking, even with a final executable:

  before `ls -l comp-decomp` $= 8658$ bytes
  after `strip comp-decom`,
  `ls -l comp-decomp` $= 6008$ bytes

- After knowing '`strip`' what do you think that are the 'targets' `Debug` and `Release` of VisualStudio?

Useful applications for files '.o', '.a' and '.so' IV

- In the case of executable files dynamically linked with some library, the command '`ldd`' is specially useful.
- It tells us with which libraries our application is dynamically linked and if some is left; moreover it tells us the path where each of them is located in the file system:

```
    ldd comp-decomp
2       linux-vdso.so.1 =>  (0x00007fffa57ff000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f38c1b46000)
4       libcomp-decomp.so => not found
        /lib64/ld-linux-x86-64.so.2 (0x00007f38c1ef6000)
```

How to create and use a library in Vala

- We separate the code following the same criteria as in the example in‘C’.
- For generating the library we use a command such as follows:

```
valac --library=comp-decomp -H comp-decomp.h comp.vala -X -fPIC -X -shared -o libcomp-decomp.so
```

- For linking it we use the command such as:

```
valac comp-decomp.vapi comp-decomp-driver.vala -X libcomp-decomp.so -X -I. -o comp-decomp-driver
```

- The files ‘`.vapi`’ are in ‘`Vala`’ similars to the‘`.h`’ in ‘`C`’.

The example code in Vala I

```
1   /* comp.vala */
    namespace CompDecomp {
3     private int caracteres_iguales (string s) {
        int l = 0;
5       int cont = 1;

7       l = s.length;
        if (l < 2) return l;
9       else {
          int i = 1;
11        while (s[0] == s[i++]) cont++;
          return cont;
13      }
      }
15
      public void comprime (string s, ref string cs) {
17      int l = 0, cont = 0;
        string num, s2;
19
        l = s.length;
21      if (l < 3) cs += s;
        else {
23        cont = caracteres_iguales (s);
          if (cont > 2) {
25          num = cont.to_string();
            cs += num; cs += s[0].to_string();
27        } else { cs += string.nfill (cont, s[0]); }
          s2 = s[cont:s.length];
29        comprime (s2, ref cs);
        }
31    }
    }
```

The example code in Vala II

```
/* comp-decomp-driver.vala */
using CompDecomp;

int main(string[] args) {
  if (args.length != 3) {
    stdout.printf ("Uso: comp-decomp [-c|-d] cadena\n");
    return 1;
  } else if ( (args[1] != "-c") && args[1] != "-d") ) {
    stdout.printf("Uso: comp-decomp [-c|-d] cadena\n");
    return 2;
  }
  if (args[1] == "-c") {
    string c="";
    comprime (args[2], ref c);
    stdout.printf ("Compresion de \"%s\"(%d) es \"%s\"(%d)\n",
                   args[2], args[2].length,
                   c,       c.length);
  }

  return 0;
}
```