Makefile

- Un target in generale e' un file. Usando `.PHONY: target` specifichiamo a make che il target non e' un file

- `make target -C ./dir` in pratica equivale all'esecuzione del comando `make target` all'interno della directory `./dir`. Parafrasando prima di leggere il makefile ed eseguire ogni altra operazione cambia directory.

Librerie

In Linux ci sono due tipi di librerie:

- Quelle statiche (.a) che diventano parte dell'applicazione
- Quelle linkate dinamicamente “shared objects” (.so). Queste librerie possono essere usate in due modi diversi.
  - Linkate dinamicamente a run-time, ma devono essere presenti al momento della compilazione/link dell'eseguibile.
  - Vengono caricate e scaricate dinamicamente e licate a run-time, usando le funzioni del loader di di sistema.

Librerie statiche

Una libreria statica e' semplicemente una "collezione" di file oggetto. In pratica e' necessario creare i vari file oggetto e poi archiviarli mediante l'utility ar (ar - create, modify, and extract from archives).

Durante la fase di linking i linker estrae dalla libreria le "funzioni" richieste e le rende parte dell'eseguibile.

Vediamo ad esempio la directory 1.

```
$ nm libutil.a

alloca.o:
0000000000000000 T dalloca_
                 U fprintf
                 U malloc
                 U stdout

free.o:
                 U free
0000000000000000 T free_
```

Librerie statiche

Chiaramente e' possibile usando ar riestrarre gli oggetti di cui e' composta una libreria:

```
$ ar x libutil.a
$ ls
alloca.c  alloca.o  free.c  free.o  libutil.a  Makefile
```

Librerie “dinamiche”

```
gcc -O0 -g -Wall -W -fPIC -c -o alloca.o alloca.c
gcc -O0 -g -Wall -W -fPIC -c -o free.o free.c
gcc -shared -Wl,-soname,libutilmine.so.1 -o
                  libutilmine.so.1.0.0 alloca.o free.o
ln -s libutilmine.so.1.0.0 libutilmine.so
ln -s libutilmine.so.1.0.0 libutilmine.so.1
```

Opzioni al compilatore:

- -fPIC: Direttiva al compilatore necessaria a produrre “position independent code”, una caratteristica essenziale per le librerie shared.
- -shared: Produce uno “ shared object” che puo' dunque essere “linked” con altri oggetti a formare un eseguibile.
- -W1: serve a passare opzioni al linker . In questo caso l'opzione che viene passata al linker e': "-soname libutilmine.so.1" (`objdump -x libutilmine.so.1.0.0 | grep "SONAME"`). Il nome passato con l'opzione -o e' invece passato al gcc

Successivamente Il link simbolico a libutilmine.so serve a fare funzionare il flag -lutilmine del compilatore. Mentre il link libutilmine.so.1 serve a runtime

Librerie “dinamiche”

Per la compilazione del programma “dipendente” dalla libreria “dinamica” si veda la directory 2.

L'eseguibile risultante ovviamente “dipende” dalla libreria stessa. La lista delle dipendenze puo' essere visualizzata mediante ldd:

```
$ ldd ./test
        libutilmine.so.1 => not found
...
```

Librerie “dinamiche”

Per fare in modo che l'eseguibile possa trovare le librerie necessarie a run-time si deve “istruire” il sistema. Ci sono tre modi di farlo:

1. Aggiungere la directory dove si trova la librerie shared al file /etc/ld.so.conf, e di seguito eseguire (con i permessi di root) il comando ldconfig.
2. Sempre come root eseguire “ldconfig -n directory”. Aggiunge la directory nella “library cache”. Chiaramente l'infomazione sara' persa al prossimo reboot del sistema.
3. Oppure usare la variabile di environment LD_LIBRARY_PATH. Ad esempio:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:directory
```

Librerie “dinamiche”

Loading e unloading dinamico di librerie shared mediante l'uso di **libdl**, vedi directory 3.

```
void *dlopen(const char *filename, int flag);
```

- filename e' il nome della libreria “shared”
- flag noi useremo come valore RTLD_LAZY, risolve i simboli solo al momento dell'esecuzione del codice

```
void *dlsym(void *handle, const char *symbol);
```

- handle e' il “opaque handle" ritornato da dlopen
- symbol e' il nome del simbolo definito nella libreria shared
- ritorna il puntatore alla locazione dove e' stato caricato il simbolo

Librerie “dinamiche”

```
char *dlerror(void);
```

- ritorna una stringa che descrive l'errore piu' recente verificatosi. La stessa funzione ritorna NULL in caso non si sia verificato nessun errore dall'ultima chiamata a dlerror.

```
int dlclose(void *handle);
```

- la librearia viene “unloaded” se non piu' necessaria

Librerie “dinamiche”

Nello stesso esempio vediamo anche l'uso di void __attribute__ ((constructor)) e void __attribute__ ((destructor)):

```
gcc -O0 -g -Wall -W -fPIC   -c -o subs.o subs.c
gcc -shared -Wl,-soname,libtest.so.1 -o libtest.so.1.0.0 subs.o
```

Compilazione del programma per l'uso di libdl:

```
gcc -I./lib -rdynamic -o test1 main1.c -ldl
```

e senza libdl:

```
gcc -I./lib -o test2 main2.c -L./lib -ltest
```

(ispezione del file /proc/PID/maps, ricordate il comando lsof ?)

Esercizio

libreria (sia statica che dinamica) con funzioni per allocare, riempire con numeri random, deallocare e stampare una matrice. Relativo programma (main) che ricevuto in input la dimensione della matrice la allochi, la riempia e ne stampi il contenuto.