1. Guida veloce a `make`

L'utility `make` è un program manager per gestire un gruppo di moduli di un programma, una raccolta di programmi o un sistema completo. Permette di esprimere dipendenze fra file.
Se dovessimo mantenere molti file sorgenti, come ad esempio:

```
programma.c     funzione1.c     funzione2.c
```

e sapessimo che programma.c (detto *target*) dipende da fuzione1.c e funzione2.c (le quali si dice formano una *dependency list*), potremmo compilare questi file utilizzando il *gcc*. Rimane però un problema di fondo. Se abbiamo già creato qualche file oggetto (`.o`) che vogliamo linkare durante la compilazione globale del programma:

- non server ricompilare un modulo per il quale abbiamo già il file oggetto, dovremmo compilare solo i file modificati dopo una determinata data;
- per linkare un oggetto, è spesso necessario scrivere lunghi comandi sul terminale, e indurre all'errore.

L'utility `make` fornisce gli strumenti per fare questi controlli e garantisce una compilazione solo di quei moduli di cui non esiste già il file oggetto aggiornato.

A tale scopo, `make` usa un file di testo di nome "makefile". Questo file contiene **regole** (dette *make rule*) che esprimomo cosa deve fare il sistema per aggiornare un target se uno dei file nella dependency list è stato modificato.

Una **regola** ha due parti, una sinistra (*target*) ed una destra (*dipendenze*), separate dai due punti. Fra le regole ci deve essere almeno una riga vuota.

```
target : [lista_target] dependency_list
Tab comando_1
.
.
.
Tab comando_n
```

- **target** è il nome di un programma che deve essere creato (per esempio l'eseguibile da produrre come risultato della compilazione), oppure il nome di un'azione (per esempio *install*, *clean*...);
- **lista_target** contiene una lista (opzionale) di altri target
- **dependency_list** contiene i nomi dei file da cui dipende il target (sorgenti, header o dati);
- **comando_1**, ... ,**comando_n** sono i comandi da eseguire per ottenere il target, obbligatoriamente preceduti da una tabulazione (una pressione del tasto TAB).

Un makefile è eseguito tramite l'esecuzione del comando `make` da terminale. Se eseguito senza specificare alcun target (scrivendo semplicemente 'make' seguito da Invio), viene eseguito il primo target nel makefile. Il comando controlla se qualcuno fra i file della dependency_list è stato modificato più recentemente del target. In caso affermativo esegue i comandi.

Operativamente, *make* costruisce l'albero delle dipendenze a partire dalla prima regola del makefile. Ogni nodo nella dependency_list della radice viene appeso come figlio. Poi si visitano le foglie e si aggiungono le dipendenze allo stesso modo. La generazione dell'albero termina quando non ci sono più regole che hanno una foglia come target.
L'albero viene poi usato tramite una visita bottom up. Per ogni nodo X si controlla che l'istante dell'ultima modifica del padre sia successivo all'istante dell'ultima modifica di X. Se non è così, va eseguita la command list della regola che ha come target il padre.

La processazione avviene come segue:

1. Viene letto il makefile, capendo così quali file hanno bisogno di essere solamente linkati e quali hanno invece bisogno di essere ricompilati;
2. Si controlla la data e l'ora di un file oggetto e se esso risulta "antecedente" rispetto ai file che lo compongono, si ricompila il file, altrimenti si linka solamente;
3. Nella fase finale si controlla data e ora di tutti i file oggetto e se anche uno solo risulta più recente del file eseguibile, allora si ricompilano tutti i file oggetto.

Solitamente il makefile definisce almeno i seguenti target:

- Il primo target (quello eseguito con 'make' senza parametri) che compila i sorgenti e produce gli eseguibili;
- *install* che provvede a spostare gli eseguibili nelle cartelle apposite (installazione);
- *clean* elimina i file prodotti dalla compilazione.

2. Come creare un makefile

Supponiamo di avere i file simili all'esempio precedente:

```
programma.c      funzione1.c      funzione2.c      header.h
```

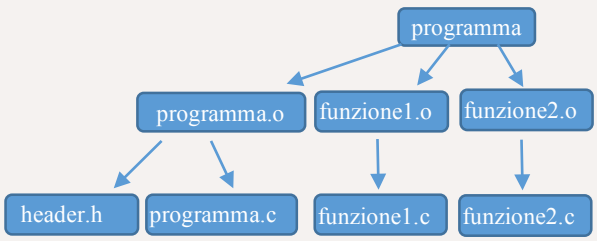
e di voler far sì che ci siano delle regole che impongano di ricompilare se cambia qualche file, ecco come, ad esempio, potrebbe essere strutturato un makefile (modificabile con un qualsiasi editor di testo):

```
programma : programma.o funzione1.o funzione2.o
        gcc programma.o funzione1.o funzione2.o

programma.o : header.h programma.c
        gcc -c programma.c

funzione1.o : funzione1.c
        gcc -c funzione1.c

funzione2.o : funzione2.c.
        gcc -c funzione2.c
```



L'interpretazione è la seguente:

- "programma" dipende da tre file oggetto, "programma.o", "funzione1.o" e "funzione2.o"; se anche uno solo di questi tre file risulta cambiato, i file devono essere linkati nuovamente;
- "programma.o" dipende da due file, "header.h" e "programma.c", se uno di questi due file è stato modificato, allora bisogna ricompilare il file oggetto. Considerazioni analoghe si fanno per per gli ultimi due comandi.

Per quanto riguarda i **commenti**, si usa il carattere cancelletto (#), così tutto ciò che è sulla medesima riga viene ignorato.

Per **eseguire un makefile** è sufficiente digitare `make` dalla linea di comando. Sarà quindi cercato nel current folder il file di nome `makefile` da eseguire.

3. Semplice makefile per il progetto

Data la semplicità dell'applicazione da realizzare, si possono semplicemente creare due sorgenti `server.c` e `client.c`. Il makefile può semplicemente essere il seguente:

```
# make rule primaria con dummy target 'all'--> non crea alcun file all ma fa un complete build
#                                              che dipende dai target client e server scritti sotto
all: client server

# make rule per il client
client: client.o
        gcc –Wall client.o –o client

# make rule per il server
server: server.o
        gcc –Wall server.o –o server

# pulizia dei file della compilazione (eseguito con 'make clean' da terminale)
clean:
        rm *o client server
```

IMPORTANTE: per la consegna del progetto, l'opzione –Wall è sempre obbligatoria.