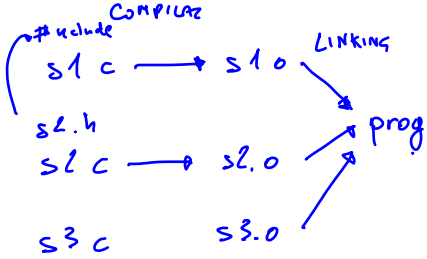


Make

Marco Alberti

Dipartimento
di Matematica
e Informatica

Università
degli Studi
di Ferrara

PROGETTI

SISTEMI DI
BUILD

Programmazione e Laboratorio, A.A. 2020-2021

Sistemi di costruzione

(o sistemi di build)

Programmi che automatizzano la ~~compilazione~~ COSTRUZIONE di un progetto software, secondo le indicazioni di un file di progetto. Vantaggi:

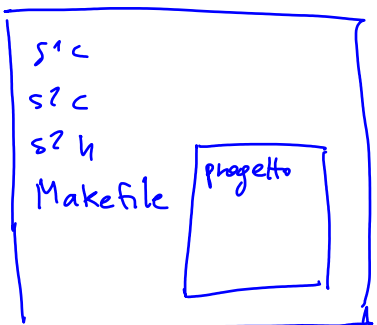
- Ricompilano solo i file modificati (importante per progetti grandi)
- Non dimenticano modifiche

Esempi:

- Make (che vedremo), usato per C e C++ (e non solo)
- MSBuild per C# e Visual Basic
- Ant, Maven, Gradle per Java

Che cos'è Make

Make è un sistema di build molto diffuso in ambiente UNIX, usato principalmente (ma non solo) per progetti C e C++.
Il file di progetto si chiama, convenzionalmente, `Makefile` e si trova nella stessa cartella dei file sorgente.
Il comando `make` esegue la costruzione del progetto secondo il contenuto di `Makefile`.



Formato Makefile

Una sequenza di target, ognuno specificato su due righe consecutive come segue:

```
__target__: __dipendenza1__ ... __dipendenzaN__
	__comandoCompilazioneTarget__
```

- La prima riga specifica da quali file dipende il target.
- La seconda qual è il comando (tipicamente di compilazione) per ottenere il target a partire dai file dal quale dipende.

Esempio

```
main: main.c
	gcc -o main main.c
```

[1] carattere di tabulazione

Esempio più articolato

Scrivere un Makefile per il progetto che deve produrre un eseguibile di nome `prog` a partire da questi sorgenti: .

file1.h

```
void p(int i);
```

file2.h

```
void q(char);
```

file3.h

```
char f(char x);
```

main.c

```
#include “file1.h”
#include “file2.h”
main()
{ p(1); q(2); }
```

file1.c

```
#include<stdio.h>
#include “file3.h”
void p(int i)
{ printf(“%d”,f(i));}
```

file2.c

```
#include<stdio.h>
#include “file3.h”
void q(char i)
{ printf(“%c”,f(i));}
```

file3.c

```
char f(char x)
{ return x+1; }
```

(una) Soluzione

Università degli Studi di Ferrara

155_make/Makefile

```
1  prog: main.o file1.o file2.o file3.o
2          gcc -o prog main.o file1.o file2.o file3.o
3
4  main.o: main.c file1.h file2.h
5          gcc -c main.c
6
7  file1.o: file1.c file3.h
8          gcc -c file1.c
9
10 file2.o: file2.c file3.h
11         gcc -c file2.c
12
13 file3.o: file3.c
14         gcc -c file3.c
```