Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative
Istituto sistemi informativi e networking

SUPSI

Ambienti Operativi: Make

Amos Brocco, Ricercatore, DTI / ISIN

Un problema di compilazione

programma.adb

```
with Calcolatrice; use Calcolatrice;
with Stampante; use Stampante;

procedure Programma is
begin
  Stampa("5 + 2 = " &
Integer'Image(somma(5,2)));
  Stampa("5 * 2 = " &
Integer'Image(moltiplica(5,2)));
end Programma;
```

calcolatrice.ads

```
package Calcolatrice is
  function somma(a,b: in Integer) return Integer;
  function moltiplica(a,b: in Integer) return Integer;
end Calcolatrice;
```

stampante.ads

```
package Stampante is
  procedure stampa(s: in String);
end Stampante;
```

stampante.adb

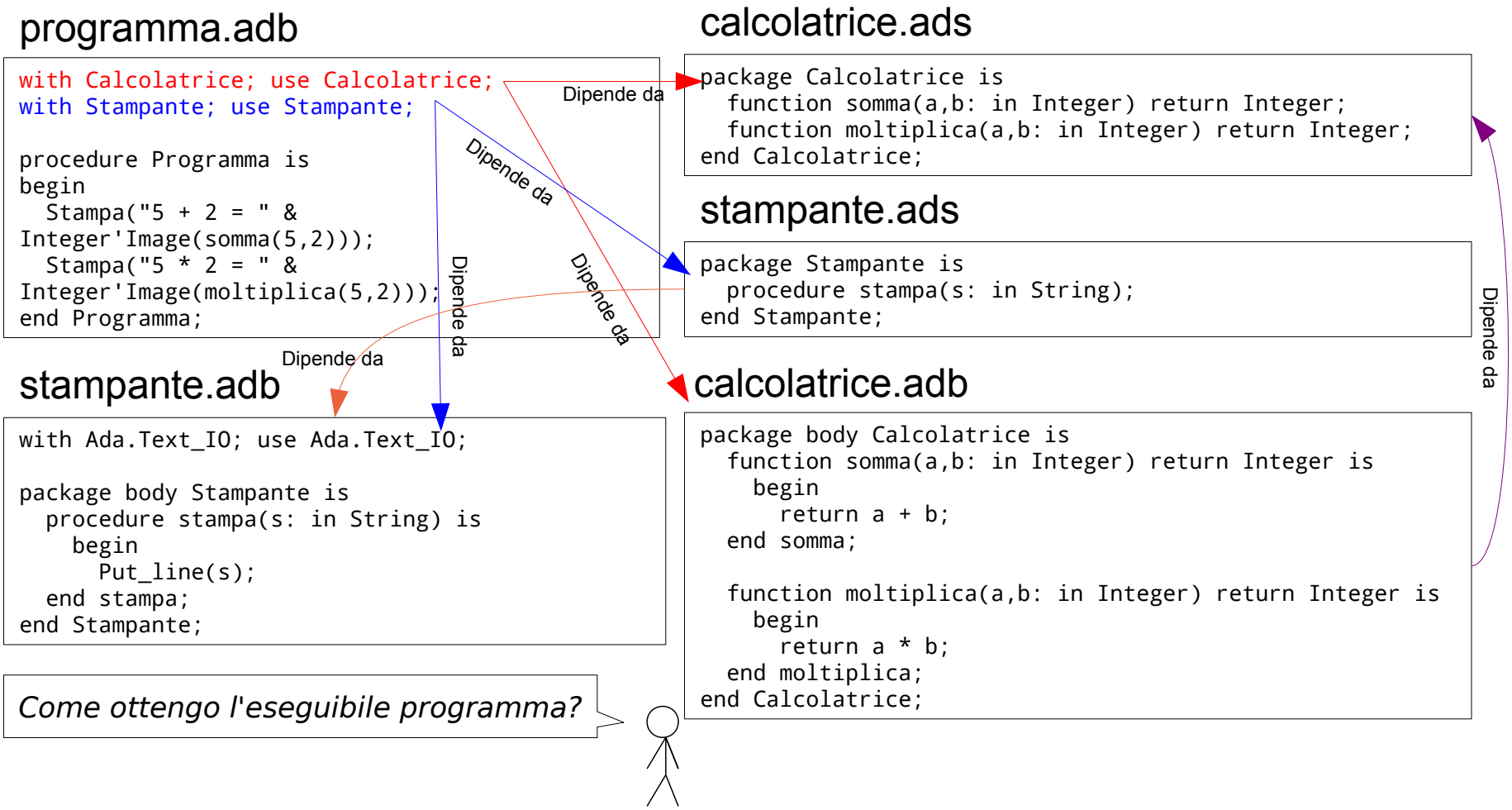
```
with Ada.Text_IO; use Ada.Text_IO;

package body Stampante is
  procedure stampa(s: in String) is
    begin
      Put_line(s);
  end stampa;
end Stampante;
```

calcolatrice.adb

```
package body Calcolatrice is
  function somma(a,b: in Integer) return Integer is
    begin
      return a + b;
  end somma;

  function moltiplica(a,b: in Integer) return Integer is
    begin
      return a * b;
  end moltiplica;
end Calcolatrice;
```

Come ottengo l'eseguibile programma?

Un problema di compilazione: dipendenze



Compilazione “manuale”

bash

```
[utente@host progetto]$ gcc -c programma.adb
```

Richiede: programma.adb Genera: programma.ali, programma.o

```
[utente@host progetto]$ gcc -c calcolatrice.adb
```

Richiede: calcolatrice.adb Genera: calcolatrice.ali, calcolatrice.o

```
[utente@host progetto]$ gcc -c stampante.adb
```

Richiede: stampante.adb Genera: stampante.ali, stampante.o

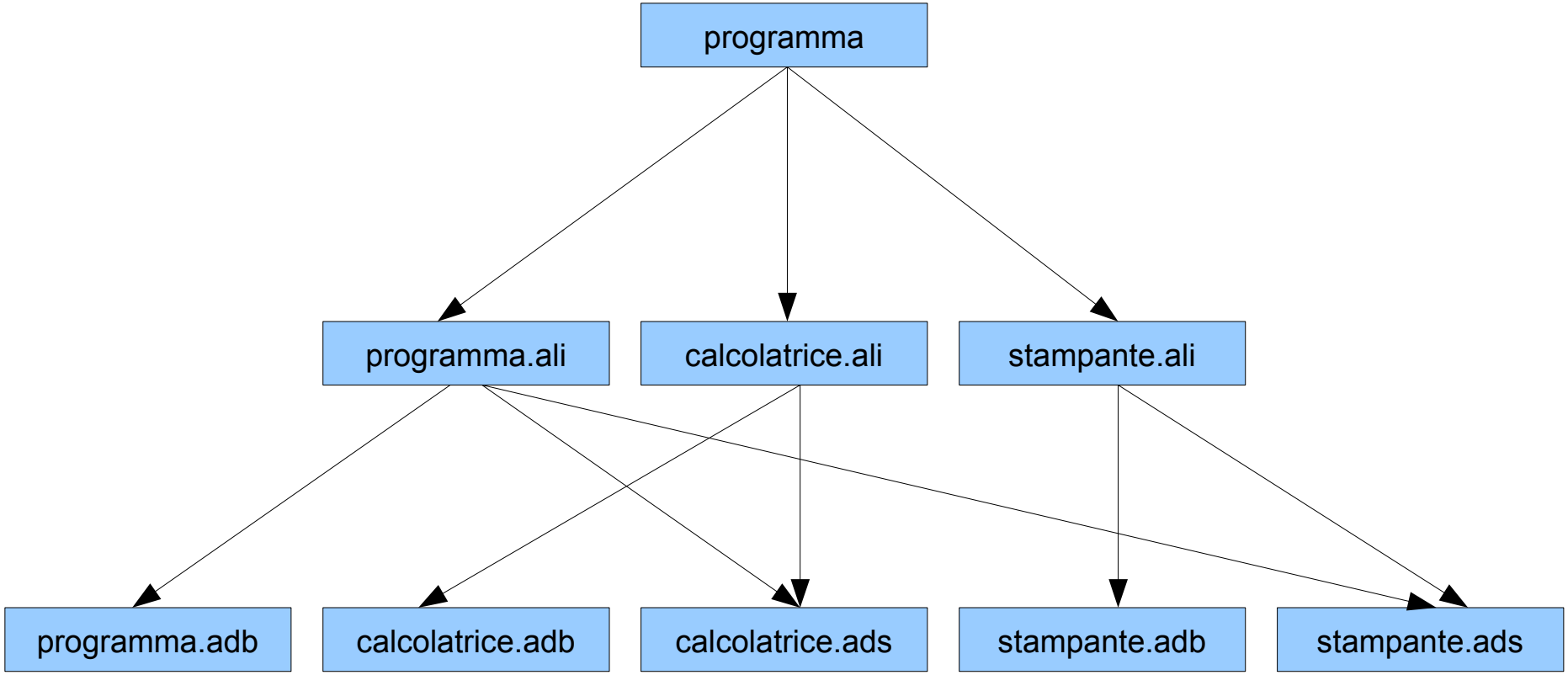
```
[utente@host progetto]$ gnatbind -x programma.ali
```

Richiede: programma.ali, calcolatrice.ali, stampante.ali Genera: b~programma.adb, b~programma.ads

```
[utente@host progetto]$ gnatlink programma.ali
```

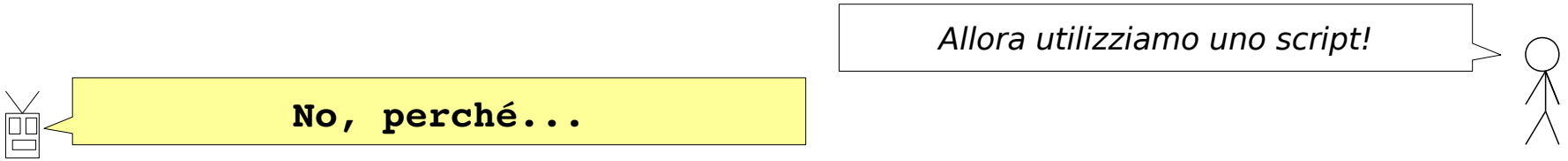
Richiede: b~programma.adb, b~programma.ads, programma.ali, calcolatrice.ali, stampante.ali, programma.o, calcolatrice.o, stampante.o
Genera b~programma.adb e programma

Dipendenze


programma
programma.ali
calcolatrice.ali
stampante.ali
programma.adb
calcolatrice.adb
calcolatrice.ads
stampante.adb
stampante.ads

Problemi

- Bisogna immettere una lunga sequenza di comandi ad ogni compilazione



- Se modifico un file devo andare a vedere anche tutti i sorgenti che dipendono da quel file e ricompilarli... e poi devo rifare il linking di tutti i file che dipendono da questi ultimi
  - Gestire manualmente queste dipendenze è complicato è può portare a errori

- Se la compilazione di un file richiede molto tempo non voglio che questo avvenga anche se non ce n'è bisogno
  - es. supponiamo che il comando “gcc -c stampante.adb” richieda molto tempo, se successivamente modifico il file “calcolatrice.adb” non c'è bisogno di ricompilare anche stampante.adb

Soluzione: Make e Makefile

- Un Makefile è un file interpretato dal tool Make che permette di organizzare la compilazione di un file

Makefile

```
obiettivo : dipendenze
            ricetta per generare "obiettivo" a partire dalle dipendenze
```

regola

Tab

Il file da generare è chiamato "**obiettivo**"

I file necessari per generare l'obiettivo eseguendo le regole sono detti "**dipendenze**"

Esempio:

Makefile

```
calcolatrice.ali: calcolatrice.adb calcolatrice.ads
            gcc -c calcolatrice.adb
```

"Per generare l'obiettivo 'calcolatrice.ali' servono 'calcolatrice.adb' e 'calcolatrice.ads', e utilizzo il comando 'gcc -c calcolatrice.adb'

Makefile in cucina

- Le regole di un Makefile possono essere viste come le istruzioni per preparare una pietanza

Makefile

```
pietanza: ingrediente_1 ingrediente_2 … ingrediente_n
	ricetta
	…
	ricetta
```

Generare un obiettivo

- Richiamando il comando 'make' il file chiamato **Makefile** nella directory corrente viene elaborato:
  - Bisogna specificare quale obiettivo si intende generare
  - Se non viene specificato nessun obiettivo, viene considerato il primo obiettivo trovato nel file

[X] bash

```
utente@host progetto$ make
gcc -c calcolatrice.adb
```

oppure

[X] bash

```
utente@host progetto$ make calcolatrice.ali
gcc -c calcolatrice.adb
```

Nota: il Makefile deve terminare con una linea vuota
Nota 2: se la configurazione non si trova nel file Makefile è possibile specificare il nome del file con ***make -f nomefile***

Dipendenze

- Un obiettivo viene rigenerato **solo** se le dipendenze cambiano (o il obiettivo non esiste più)
  - es. se modifico calcolatrice.adb

[X] bash

```
utente@host progetto$ make
make: `calcolatrice.ali' is up to date.
```

[X] bash

```
utente@host progetto$ touch calcolatrice.ali
utente@host progetto$ make
gcc -c calcolatrice.adb
```

- Per determinare se una dipendenza è cambiata Make utilizza unicamente la data/ora di modifica di ogni file:
  - `if (“data/ora di modifica di obiettivo” < “data/ora di modifica di dipendenze”) {`
    `rigenera target`
    `}`

Più regole e obiettivi

- È possibile avere più regole all'interno dello stesso Makefile, quando si esegue make senza specificare un obiettivo, la generazione comincia dal primo obiettivo
- Le regole devono essere separate da una linea vuota

Makefile

```
programma: programma.ali calcolatrice.ali stampante.ali
	gnatbind -x programma.ali
	gnatlink programma.ali

programma.ali: programma.adb calcolatrice.ads
	gcc -c programma.adb

calcolatrice.ali: calcolatrice.adb calcolatrice.ads
	gcc -c calcolatrice.adb

stampante.ali: stampante.adb stampante.ads
	gcc -c stampante.adb

clean:
	rm -f programma calcolatrice.ali calcolatrice.o programma.ali programma.o \\
	stampante.ali stampante.o
```

Un obiettivo può anche non essere un file!
Si parla allora di *phony target (**obiettivo virtuale**)*

Obiettivi virtuali di uso comune

Obiettivo	Descrizione
`all`	Effettua tutte le operazioni richieste per generare il programma
`install`	Installa l'applicazione creata nei percorsi predefiniti (Unix)
`clean`	Cancella i file binari prodotti durante il make
`distclean`	Cancella tutti i file che non facevano parte dei sorgenti originali
`check`	Esegue i test contenuti nei sorgenti dell'applicazione

Variabili

- È possibile specificare delle variabili (es. per rendere più flessibile il Makefile)
  - Per definire una variabile utilizzare **`NOME=valore`**
  - Per richiamare una variabile utilizzare **`$(NOME)`**

Makefile

```
ADABIND=gnatbind -x
ADABINDOPS=-x
ADALINK=gnatlink
ADACOMPILER=gcc -c

programma: programma.ali calcolatrice.ali stampante.ali
            $(ADABIND) $(ADABINDOPS) programma.ali
            $(ADALINK) programma.ali

programma.ali: programma.adb calcolatrice.ads
            $(ADACOMPILER) programma.adb

calcolatrice.ali: calcolatrice.adb calcolatrice.ads
            $(ADACOMPILER) calcolatrice.adb

stampante.ali: stampante.adb stampante.ads
            $(ADACOMPILER) stampante.adb

clean:
            rm -f programma calcolatrice.ali calcolatrice.o programma.ali programma.o \\
            stampante.ali stampante.o
```

Espansione delle variabili

- In make ci sono due tipi di variabili:
  - espansione ritardata (con **=**, predefinite)
    - vengono interpretate nel momento in cui devono essere usate
  - espansione immediata (con **:=**):
    - vengono interpretate non appena vengono lette da make

Makefile

```
B = “ciao”
A = $(B)
B = “mondo”
```

Sia A che B contengono “mondo”.

Makefile

```
B := “ciao”
A := $(B)
B := “mondo”
```

A contiene “ciao”, B contiene “mondo”.

Assegnazione di variabili

- Assegnazione condizionata (**?=**)
  - il valore viene assegnato alla variabile solo se la variabile non è già stata definita precedentemente. La variabile creata è ad espansione ritardata.

Makefile
```
OUTPUT_DIR ?= $(PROJECT_DIR)/out
```

- Aggiunta (**+=**)
  - aggiunge ulteriore contenuto ad una variabile già presente. Il tipo di variabile non viene modificato se già presente, altrimenti viene creata una variabile ad espansione ritardata.

Makefile
```
OBJECTS += test.o
```

Variabili automatiche

Variabile	Descrizione
$@	Contiene il nome dell'obiettivo
$<	Contiene il nome del file del primo requisito
$?	Contiene i nomi dei requisiti che sono più nuovi dell'obiettivo, separati da spazi
$^	Contiene il nome di tutti i requisiti, separati da spazi. Se uno o più requisiti sono duplicati, saranno indicati solo una volta.
$+	Come $^, ma non rimuove i duplicati.
$*	Contiene il nome dell'obiettivo, privo del suffisso

Variabili automatiche

Makefile

```
ADABIND=gnatbind -x
ADABINDOPS=-x
ADALINK=gnatlink
ADACOMPILER=gcc -c

programma: programma.ali calcolatrice.ali stampante.ali
            $(ADABIND) $(ADABINDOPS) programma.ali
            $(ADALINK) programma.ali

programma.ali: programma.adb calcolatrice.ads
            $(ADACOMPILER) $<

calcolatrice.ali: calcolatrice.adb calcolatrice.ads
            $(ADACOMPILER) $<

stampante.ali: stampante.adb stampante.ads
            $(ADACOMPILER) $<

clean:
            rm -f programma calcolatrice.ali calcolatrice.o programma.ali programma.o \\
            stampante.ali stampante.o
```

Metacaratteri

- Posso usare dei metacaratteri sia nelle ricette di generazione che nelle dipendenze
  - *** , ?, ~, gruppi di caratteri [...]**
- Le sostituzioni nelle ricette vengono effettuate dalla shell

Makefile

```
ADABIND=gnatbind -x
ADABINDOPS=-x
ADALINK=gnatlink
ADACOMPILER=gcc -c

programma: programma.ali calcolatrice.ali stampante.ali
            $(ADABIND) $(ADABINDOPS) programma.ali
            $(ADALINK) programma.ali

programma.ali: programma.ad?
            $(ADACOMPILER) programma.adb

calcolatrice.ali: calcolatrice.ad?
            $(ADACOMPILER) calcolatrice.adb

stampante.ali: stampante.ad?
            $(ADACOMPILER) stampante.adb

clean:
            rm -f programma *.ali *.o
```

Metacaratteri: esempi

- Genera un programma usando tutti i file che terminano con .c:

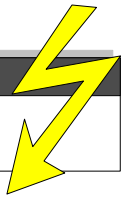
Makefile

```
programma: *.c
            gcc -o $@ $^
```

- È anche possibile usare metacaratteri all'interno dell'obiettivo:
  - **l'espansione avviene al momento dell'esecuzione del Makefile**

Makefile

```
*.ali: dipendenza.ads
```



  - Aggiunge ai file con suffisso .ali nella directory corrente la dipendenza al file dipendenza.ads. Tuttavia, all'esecuzione di questa istruzione potrebbero non esistere questi file (e quindi la regola non è applicata)

Raggruppare obiettivi con lo stesso suffisso: le regole generali

- Con il carattere **%** è possibile definire degli obiettivi che hanno lo stesso suffisso, creando una regola generale

Makefile

```
ADABIND=gnatbind -x
ADABINDOPS=-x
ADALINK=gnatlink
ADACOMPILER=gcc -c

programma: programma.ali calcolatrice.ali stampante.ali
	$(ADABIND) $(ADABINDOPS) programma.ali
	$(ADALINK) programma.ali

programma.ali: programma.adb calcolatrice.ads
	$(ADACOMPILER) programma.adb

%.ali: %.adb %.ads
	$(ADACOMPILER) $<

clean:
	rm -f programma *.ali *.o
```

Più obiettivi nella stessa regola

- Possiamo avere più obiettivi nella stessa regola

Makefile

```
others=c.txt d.txt
otherbak=a.bak c.bak

programma: programma.ali calcolatrice.ali stampante.ali a.txt b.txt c.txt d.txt
	gnatbind -x programma.ali
	gnatlink programma.ali

programma.ali: programma.adb calcolatrice.ads
	gcc -c programma.adb

calcolatrice.ali: calcolatrice.adb calcolatrice.ads
	gcc -c calcolatrice.adb

stampante.ali: stampante.adb stampante.ads
	gcc -c stampante.adb

clean:
	rm -f programma *.ali *.o *.txt

a.txt b.txt $(others): b.bak d.bak $(otherbak)
	cp a.bak a.txt
	cp b.bak b.txt
	cp c.bak c.txt
	cp d.bak d.txt
```

Più regole per un obiettivo

- Quando più regole sono definite per (o corrispondo a) un obiettivo Make le raggruppa e le esegue

Makefile

```
programma: programma.ali calcolatrice.ali stampante.ali
	gnatbind -x programma.ali
	gnatlink programma.ali

programma.ali: programma.adb calcolatrice.ads
	gcc -c programma.adb

calcolatrice.ali: calcolatrice.adb calcolatrice.ads
	gcc -c calcolatrice.adb

stampante.ali: stampante.ads

stampante.ali: stampante.adb
	gcc -c stampante.adb

clean:
	rm -f programma *.ali *.o
```

Obiettivi e percorsi

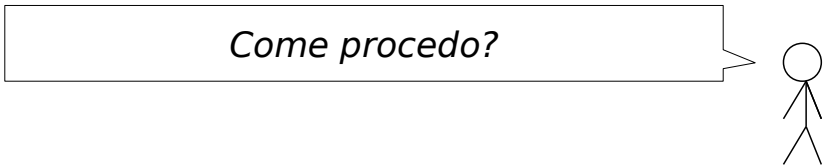
- Un obiettivo può anche fare riferimento a file in un altro percorso

Makefile

```
ADABIND=gnatbind -x
ADABINDOPS=-x
ADALINK=gnatlink
ADACOMPILER=gcc -c

programma: programma.ali calcolatrice.ali stampante.ali
            $(ADABIND) $(ADABINDOPS) programma.ali
            $(ADALINK) programma.ali

programma.ali: programma.adb calcolatrice.ads prova/test.bak
            $(ADACOMPILER) programma.adb

prova/test.bak: prova/test.txt
            cp prova/test.txt prova/test.bak

%.ali: %.adb %.ads
            $(ADACOMPILER) $<

clean:
            rm -f programma *.ali *.o prova/test.bak
```

Un altro esempio

- Vogliamo generare un archivio “thumbnails.tar.gz” con delle miniature delle seguenti immagini:
  - `gerwinski-gnu-head.png philosophical-gnu.png hitflip-gnu1.jpg hurd_mf.jpg`
- Per generare le miniature utilizziamo il comando convert
  - `convert -thumbnail 200 inputimage.png thumbnail.png`
- Per creare un file .tar utilizzo
  - `tar -cf nomefile.tar [file da inserire]`
- Per creare un archivio .gz di un file utilizzo
  - `gzip nomefile`



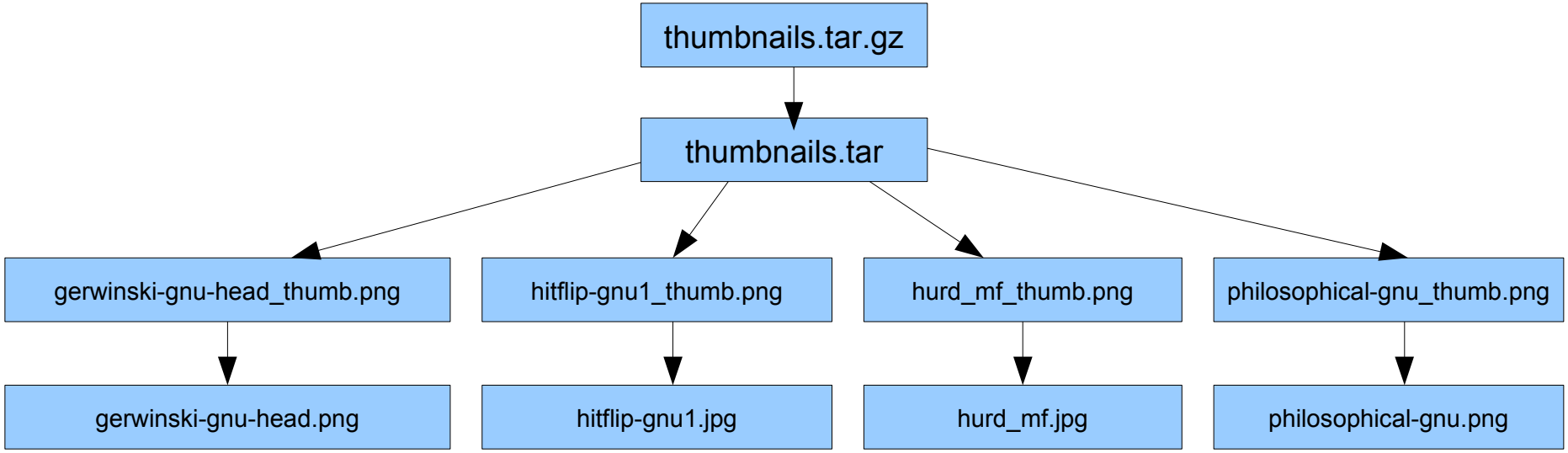
Quali sono i comandi da eseguire per ottenere l'archivio?

Bash

```
utente@host immagini$ convert -thumbnail 200 gerwinski-gnu-head.png gerwinski-gnu-head_thumb.png
utente@host immagini$ convert -thumbnail 200 hitflip-gnu1.jpg hitflip-gnu1_thumb.png
utente@host immagini$ convert -thumbnail 200 hurd_mf.jpg hurd_mf_thumb.png
utente@host immagini$ convert -thumbnail 200 philosophical-gnu.png philosophical-gnu_thumb.png
utente@host immagini$ tar -cf thumbnails.tar *_thumb.png
utente@host immagini$ gzip thumbnails.tar
```

1. Determinare, a grandi linee, la sequenza di comandi da eseguire

Quali sono le dipendenze?



2. Determinare le dipendenze

Il Makefile

Makefile

```
IMAGES=gerwinski-gnu-head_thumb.png hitflip-gnu1_thumb.png hurd_mf_thumb.png philosophical-
gnu_thumb.png

thumbnails.tar.gz: thumbnails.tar
            gzip thumbnails.tar

thumbnails.tar: $(IMAGES)
            tar -cf thumbnails.tar *_thumb.png

%_thumb.png: %.png
            convert -thumbnail 200 $< $*_thumb.png

%_thumb.png: %.jpg
            convert -thumbnail 200 $< $*_thumb.png

clean:
            rm -f *_thumb.png
            rm -f thumbnails.tar
            rm -f thumbnails.tar.gz
```

3. Scrivere le regole

Debugging

- Con l'opzione **-d** posso verificare quello che fa Make