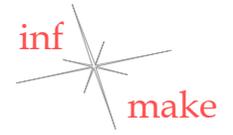




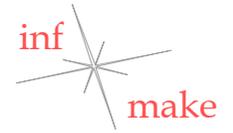
Make



- Strumento che esegue un insieme di azioni in base ad una serie di “regole di dipendenza”
- In genere usato per produrre e mantenere un eseguibile, a partire da numerosi moduli
- Le regole di dipendenza su cui si basa make sono generalmente contenute in un file
 - nomi di default del file: `Makefile` oppure `makefile`
 - se si usano altri nomi: `make -f nomefile`



Moduli per prove



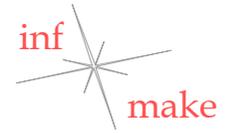
```
reverse.h  
void reverse (char *,char *);
```

```
reverse.c  
#include <stdio.h>  
#include "reverse.h"  
  
void reverse (char *prima, char *dopo)  
{  
    int i,j,lung;  
  
    lung = strlen (prima);  
    for (j=lung-1, i=0; j>=0; j--, i++)  
        dopo[i] = prima[j];  
    dopo[lung] = NULL;  
}
```

```
main.c  
#include <stdio.h>  
#include "reverse.h"  
  
main()  
{  
    char str [100];  
  
    reverse ("pippo",str);  
    printf ("reverse (\"pippo\") = %s\n", str);  
    reverse ("aerea",str);  
    printf ("reverse (\"aerea\") = %s\n", str);  
}
```



Compilazione separata



```
% cc -c reverse.c main.c
```

```
reverse.c:
```

```
main.c:
```

```
% ll -F
```

```
total 10
```

```
-rw-r--r--  1 dandrea  216 Feb 27 17:15 main.c
-rw-r--r--  1 dandrea  880 Feb 27 17:17 main.o
-rw-r--r--  1 dandrea  224 Feb 27 17:16 reverse.c
-rw-r--r--  1 dandrea   12 Feb 27 17:12 reverse.h
-rw-r--r--  1 dandrea  708 Feb 27 17:17 reverse.o
```

compilazione separata dei moduli
produce due file oggetto (.o)

```
% cc -o prova reverse.o main.o
```

```
itnhp2% ll -F
```

```
total 42
```

```
-rw-r--r--  1 dandrea  216 Feb 27 17:15 main.c
-rw-r--r--  1 dandrea  880 Feb 27 17:17 main.o
-rwxr-xr-x  1 dandrea 16384 Feb 27 17:18 prova*
-rw-r--r--  1 dandrea  224 Feb 27 17:16 reverse.c
-rw-r--r--  1 dandrea   12 Feb 27 17:12 reverse.h
-rw-r--r--  1 dandrea  708 Feb 27 17:17 reverse.o
```

link: collegamento dei moduli in un eseguibile

```
% ./prova
```

```
reverse ("pippo") = oppip
```

```
reverse ("aerea") = aerea
```

esecuzione ...



Altro materiale di esempio



palindromi.c

```
#include <stdio.h>
#include "reverse.h"
#include "palindromi.h"

int palindromi (char *stringa)
{
    char rovesciata [100];
    reverse (stringa, rovesciata);
    return (strcmp (stringa, rovesciata) == 0);
}
```

main2.c

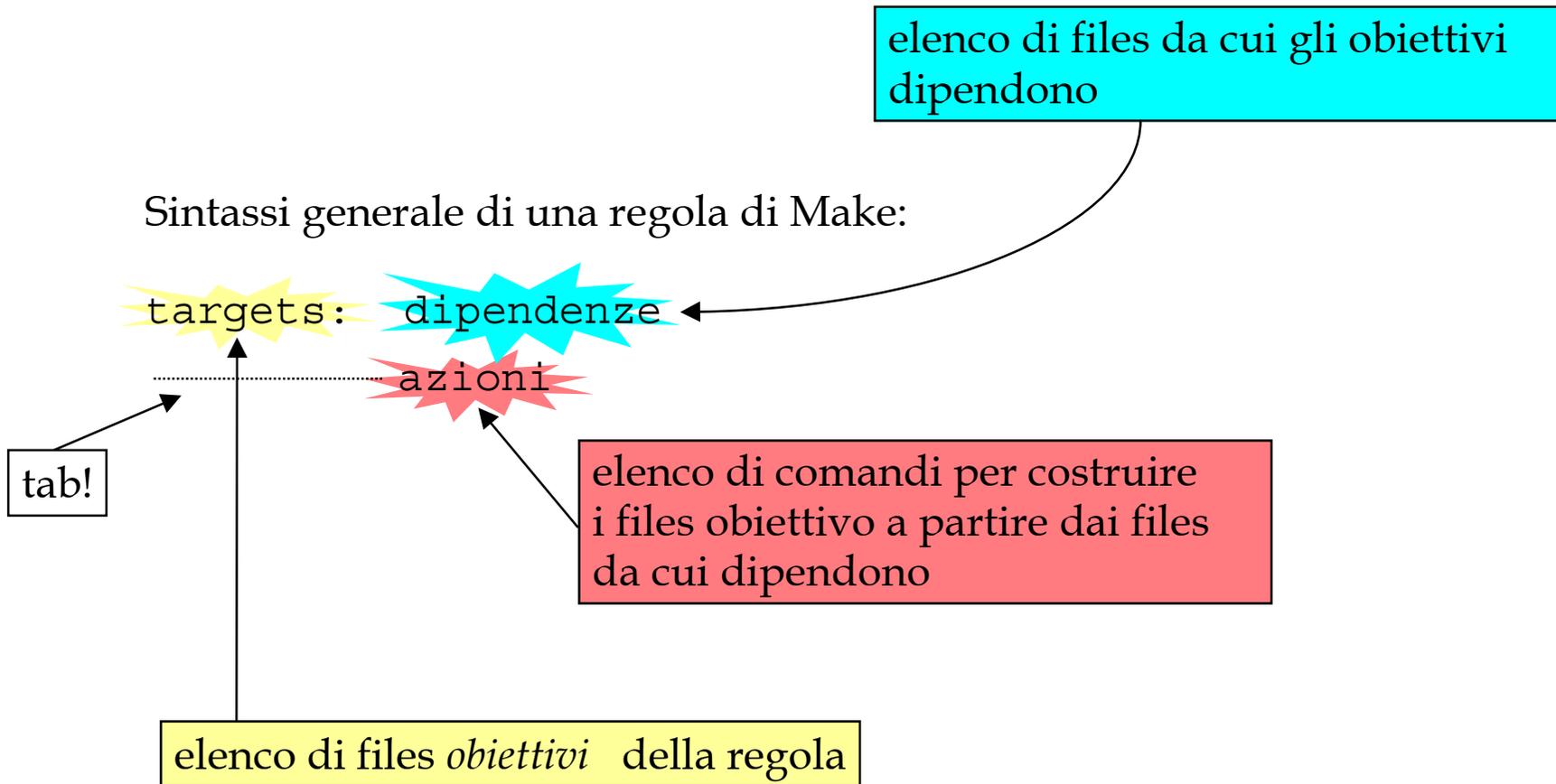
```
#include <stdio.h>
#include "palindromi.h"

main()
{
    printf ("palindromo (\\"pippo\\") = %d\n",
           palindromi("pippo") );
    printf ("palindromo (\\"aerea\\") = %d\n",
           palindromi("aerea") );
}
```

```
% cc -c palindromi.c main2.c reverse.c
% cc -o prova main2.o palindromi.o reverse.o
% ls -F
main.c main.o main2.c main2.o palindromi.c palindromi.h
palindromi.o prova* reverse.c reverse.h reverse.o
% ./prova
palindromo ("pippo") = 0
palindromo ("aerea") = 1
```



Sintassi delle regole





Esempio



```
Makefile

prova:      main.o reverse.o
            cc -o prova main.o reverse.o

main.o:    main.c reverse.h
            cc -c main.c

reverse.o: reverse.c reverse.h
            cc -c reverse.c
```

Nome di default del file delle regole di make

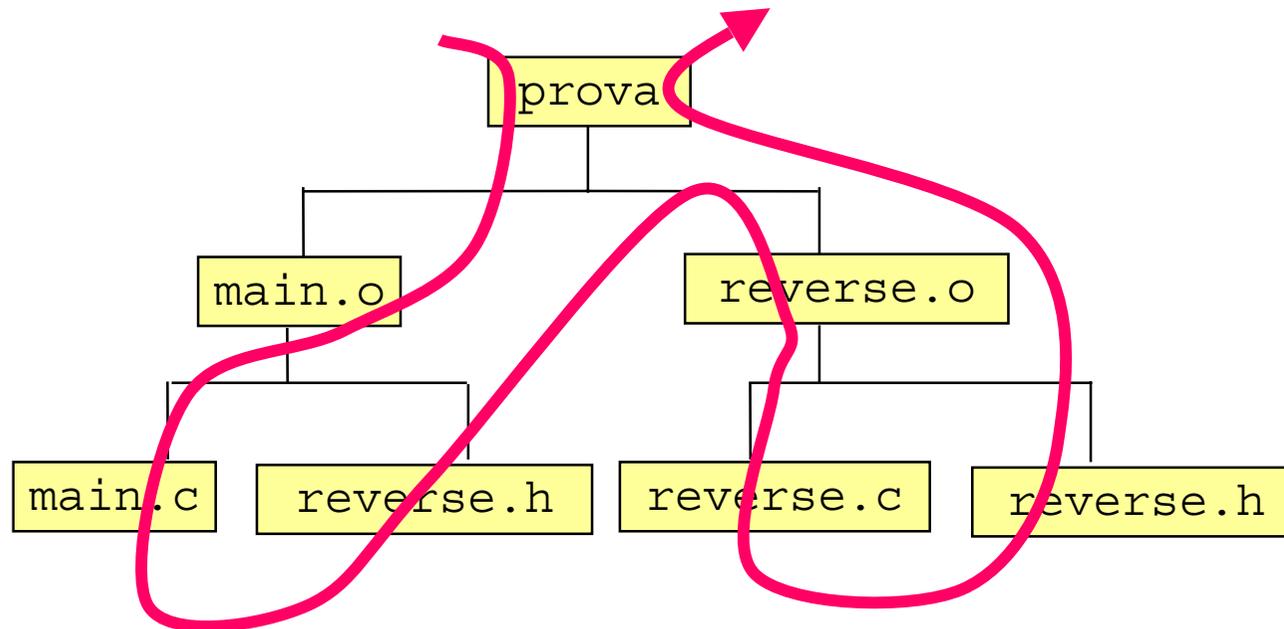
per ottenere "prova" ci vogliono "main.o" e "reverse.o", se uno di questi è più recente di "prova" allora il comando da eseguire per aggiornare "prova" è "cc -o .."

Esegue le regole contenute nel file Makefile nella directory corrente

```
% make
cc -c main.c
cc -c reverse.c
cc -o prova main.o reverse.o
```



Albero delle dipendenze



- L'albero viene visitato in ordine deep-first left-to-right.
- Se qualcuno dei file da cui dipende il target è stato cambiato più recentemente del target, allora viene eseguita l'azione



Azioni



- Una azione con un target ma senza dipendenze viene sempre eseguita

```
clean:  
    rm *.o core
```

- Le azioni si possono specificare di seguito alle dipendenze, separate da ;
TAB

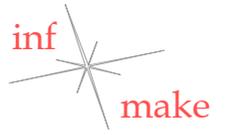
```
clean:; rm *.o core
```

- Il comando associato ad una azione può proseguire nelle linee seguenti usando il carattere \ (le linee di continuazione devono iniziare con TAB)

```
clean:  
    rm *.o \  
    core
```



Azioni e macro



Più azioni uguali possono essere raggruppate

```
targ1:      cc targ1
targ2:      cc targ2
```

Equivale a:

```
targ1 targ2:
          cc $*
```

La macro `$*` indica il nome del target corrente.



Macro



Una definizione di macro ha la forma:

```
NOMEMACRO = STRINGA-DI-SOSTITUZIONE
```

L'utilizzo può avvenire in qualunque azione, specificando \$ (NOMEMACRO) verrà invece inserita nel comando la STRINGA-DI-SOSTITUZIONE

Esempio:

```
CFLAGS = -g
```

```
tip: tip.c  
    cc $(CFLAGS) -o tip tip.c
```

Equivale a:

```
tip: tip.c  
    cc -g -o tip tip.c
```



Macro - Defaults



Alcune macro hanno un valore di default:

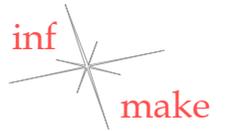
```
CC      = cc
CFLAGS  = -O
MAKE    = make
```

Altre vengono inizializzate in base ai valori delle variabili di environment:

```
SHELL   = /bin/bash
EDITOR  = /usr/bin/vi
HOME    = /home/koscar
LOGNAME = koscar
PATH    = ...
TERM    = xterm
```



Shell & Azioni



Per ciascuna azione make lancia una shell che esegue l'azione:

```
tip: tip.c  
cd miadir  
cc -o tip tip.c
```

NON FUNZIONA!

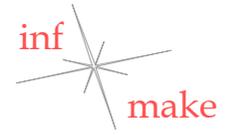
Si può fare raggruppando i comandi:

```
tip: tip.c  
cd miadir; cc -o tip tip.c
```

in questo caso l'intera linea viene passata ad una shell che esegue i due comandi in sequenza.



Regole predefinite (es.)



Alcune regole sono predefinite da make

Ad esempio si può creare un `makefile` come segue:

```
prova:      main.o reverse.o palindromi.o
            cc -o prova main.o reverse.o palindromi.o

main.o:     palindromi.h
reverse.o:  reverse.h
palindromi.o: palindromi.h reverse.h
```

la dipendenza di `main.o` da `main.c` e la azione associata vengono gestite da una regola di default
(idem per `reverse.o` e per `palindromi.o`)



Regole predefinite



La regola di default utilizzata nell'esempio precedente è una "suffix rule":

```
.C.O:  
    $(CC) $(CFLAGS) -c $<
```

Le suffix rule sono regole di tipo

DsTs:
 regola

dove **Ts** indica il suffisso del target e **Ds** indica il suffisso della dipendenza. In una suffix rule la macro predefinita \$< indica il nome della dipendenza.



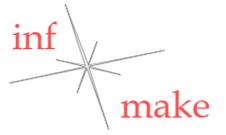
Opzioni di make



- `make -f miofile`
utilizza il file `miofile` invece di `Makefile`
- `make -n` visualizza le azioni senza compierle
- `make -i` ignora gli errori e prosegue fino in fondo
- `make -p` visualizza le macro e le regole di default
- `make -s` non visualizza le azioni che compie



Esercizi



- Provare, con le opzioni -n e -s, gli esempi fatti
- Scrivere un makefile per main2/palindromi
- Scrivere un makefile per reverse (cfr lucido 13)
- Unire i precedenti in un unico makefile (target ALL)
- Scrivere un makefile per altri esercizio svolti
- Scrivere un makefile per i prossimi esercizi ...