

Uso di strumenti software per la compilazione : **Make e Makefile.**

A. Murli - Calcolo Scientifico
a.a.2004/2005

1



Un programma

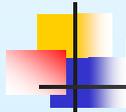
**Un programma scritto in un
qualsiasi **linguaggio ad alto livello**
si compone di diverse "unità di programma",
memorizzate generalmente in file diversi.**

Il programma chiamante

```
#include <stdio.h>
main() {
...
prog1(n,A);
prog2(A,B);
prog3(n,B);
}
main.c
```

Murli - Calcolo Scientifico
a.a.2004/2005

2



Un programma

Un programma scritto in un qualsiasi **linguaggio ad alto livello** si compone di diverse "unità di programma", memorizzate generalmente in file diversi.

Le routine ausiliarie.

```
#include <stdio.h>
main() {
...
prog1(n,A);
prog2(A,B);
prog3(n,B);
}
```

main.c

prog1.c
prog2.c
prog3.c

prog1.c

prog2.c

prog3.c

Murli - Calcolo Scientifico
a.a.2004/2005

3



Generazione di un programma eseguibile

La generazione di un programma **eseguibile** a partire da un programma scritto in un linguaggio di programmazione ad alto livello si compone essenzialmente di **due fasi**:

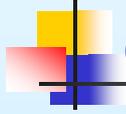
I FASE – Compilazione dei files sorgenti

```
cc -c main.c prog1.c prog2.c prog3.c
```

...dai file.c ai file.o

A. Murli - Calcolo Scientifico
a.a.2004/2005

4



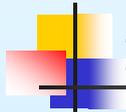
Generazione di un programma eseguibile

La generazione di un programma **eseguibile** a partire da un programma scritto in un linguaggio di programmazione ad alto livello si compone essenzialmente di **due fasi**:

II FASE - Creazione del programma eseguibile

```
cc -o ese.exe main.o prog1.o prog2.o prog3.o
```

...dai file.o al file.exe



In generale

La compilazione può essere molto laboriosa.



- **Compilazione di tutti i files sorgenti.**
- **Link di tutti i files oggetto.**
- **Link a opportune librerie.**

• ...



Esempio: utilizzo di un modulo di BLAS1

```
#include <stdio.h>
void dcopy_(int *,double *,int *, double *,int *);

main( ){
...
dcopy_(&n,dx,&incx,dy,&incy);
...
}
```

copia.c

I FASE – **Compilazione** dei files sorgenti

```
gcc -c main.c copia.c
```



Esempio: utilizzo di un modulo di BLAS1

```
#include <stdio.h>
void dcopy_(int *,double *,int *, double *,int *);

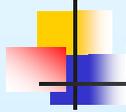
main( ){
...
dcopy_(&n,dx,&incx,dy,&incy);
...
}
```

copia.c

II FASE - **Creazione** del programma eseguibile

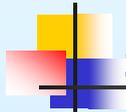
```
gcc -o ese main.o copia.o -L/path -lblas
```

path della
directory
della
libreria
libblas.a



Domanda

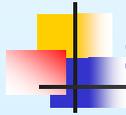
E' possibile rendere più
semplici ed automatiche
queste operazioni
?



Soluzione 1

La più "naturale"...

E' possibile scrivere
uno *script*
che esegua sequenzialmente
tutte le operazioni necessarie!



Soluzione 2

La più efficiente...

Utilizzare il
programma
make !



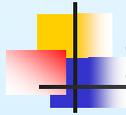
Il programma MAKE.

Il programma **make** esegue
sequenzialmente tutte le operazioni
descritte in un file generalmete
chiamato **makefile** o **Makefile**
(nome predefinito).

Un **makefile** è uno **script** specializzato per il
programma **make**.

Contiene:

- definizioni di macro
- descrizioni delle operazioni da compiere.

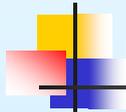


Il programma MAKE.

Il programma **make** esegue sequenzialmente tutte le operazioni descritte in un file generalmente chiamato **makefile** o **Makefile** (nome predefinito).

Per eseguire il makefile

```
make -f makefile
```



Esempio di makefile

```
# makefile di un programma
# prima versione

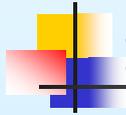
ese.exe: main.o prog1.o prog2.o prog3.o
    cc -o prog main.o prog1.o prog2.o prog3.o

main.o : main.c
    cc -c main.c

prog1.o : prog1.c
    cc -c prog1.c

prog2.o : prog2.c
    cc -c prog2.c

prog3.o : prog3.c
    cc -c prog3.c
```



In dettaglio

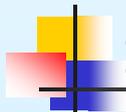
In un makefile si distinguono 2 tipi di linee:

- Linea di **dipendenza**

Definisce una relazione tra un **target** e uno o più file detti di **dipendenza**.

- Linea di **comando**

Definisce l'operazione da effettuare sui files di dipendenza per ottenere il corrispondente **target**.



In dettaglio

In un makefile si distinguono 2 tipi di linee:

- Linea di **dipendenza**

L'assegnazione di un target avviene con ":"

```
main.o : main.c
```

- Linea di **comando**

La linea di comando comincia dopo un carattere di tabulazione

```
<TAB>cc -c main.c
```



Esempio: linee di dipendenza

```
# makefile di un programma
# prima versione

ese.exe: main.o prog1.o prog2.o prog3.o
    cc -o prog main.o prog1.o prog2.o prog3.o

main.o : main.c
    cc -c main.c

prog1.o : prog1.c
    cc -c prog1.c

prog2.o : prog2.c
    cc -c prog2.c

prog3.o : prog3.c
    cc -c prog3.c
```

Makefile

Linee di dipendenza

a.a.2004/2005

17



Esempio: linee di comando

```
# makefile di un programma
# prima versione

ese.exe: main.o prog1.o prog2.o prog3.o
    cc -o ese.exe main.o prog1.o prog2.o prog3.o

main.o : main.c
    cc -c main.c

prog1.o : prog1.c
    cc -c prog1.c

prog2.o : prog2.c
    cc -c prog2.c

prog3.o : prog3.c
    cc -c prog3.c
```

Makefile

Linee di comando

a.a.2004/2005

18

Esempio: linee di commento

```
# makefile di un programma
# prima versione

ese.exe: main.o prog1.o prog2.o prog3.o
    cc -o ese.exe main.o prog1.o prog2.o prog3.o

main.o : main.c
    cc -c main.c

prog1.o : prog1.c
    cc -c prog1.c

prog2.o : prog2.c
    cc -c prog2.c

prog3.o : prog3.c
    cc -c prog3.c
```

Makefile

Linee di
commento

a.a.2004/2005

19

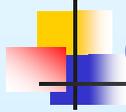
Osservazione 1

**Non è sempre necessario
specificare
in un Makefile tutti i target ed i
comandi da eseguire.**

**Esistono delle regole predefinite che
il make è in grado di riconoscere
senza che esse siano specificate!**

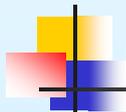
A. Murli - Calcolo Scientifico
a.a.2004/2005

20



Osservazione 2

Il make ricorre alle regole predefinite solo se nel makefile una linea di dipendenza non è seguita da una linea di comando



Esempio di makefile

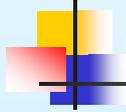
```
# makefile di un programma
# seconda versione

ese.exe: main.o prog1.o prog2.o prog3.o
    cc -o ese.exe main.o prog1.o prog2.o prog3.o

main.o : main.c
prog1.o : prog1.c
prog2.o : prog2.c
prog3.o : prog3.c
```

Il make riconosce i files con suffisso ".c" e fa eseguire se necessario la loro compilazione

Makefile



Esempio di makefile

```
# makefile di un programma  
# terza versione  
  
ese.exe: main.o prog1.o prog2.o prog3.o  
cc -o ese.exe main.o prog1.o prog2.o prog3.o
```

Makefile

**Il make riconosce automaticamente
la dipendenza di ciascun file ".o"
dal corrispondente file ".c"**



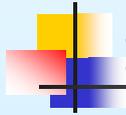
Osservazione 3

**In un Makefile per sintetizzare le
linee di dipendenza e di
comando è possibile
definire delle **MACRO**.**



Le **MACRO possono specificare:**

- Elenchi di file.
- Opzioni dei compilatori.
- Librerie
- Comandi.



In dettaglio

- Definizione di una **macro**

L'assegnazione di una macro avviene con "="

```
PATH = /usr/local/lib
```

```
CC = cc
```

```
...
```



Esempio di makefile

```
# makefile di un programma  
# terza versione  
  
oggetti = main.o prog1.o prog2.o prog3.o  
  
ese.exe: $(oggetti)  
    cc -o ese.exe $(oggetti)
```

Makefile

Definizione di una macro



Esempio di makefile

```
# makefile di un programma
# terza versione

oggetti = main.o prog1.o prog2.o prog3.o

ese.exe: $(oggetti)
    cc -o ese.exe $(oggetti)
```

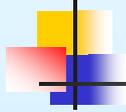
Makefile

Il carattere "\$" indica il valore della macro fra le parentesi (...)



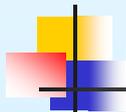
Macro predefinite

CC = cc	compilatore C
AS = as	assemblatore
CFLAGS = ...	Flag del compilatore del C



Macro speciali

\$*	Nome del file dipendente privato dell'eventuale suffisso
\$@	Nome per esteso del file dipendente corrente
\$<	Nomi per esteso di tutti i file di dipendenza



Domanda

**E se utilizziamo moduli di librerie come ad esempio le
BLAS
?**

Esempio makefile per BLAS

```
# definizione della macro che specifica
# il path della libreria BLAS
BLASLIB = /usr/local/lib/libblas.a

# definizione delle macro per i compilatori
CC = gcc
FC = g77

# definizione delle macro degli oggetti
OBJ = main.o copy.o max.o

# definizione degli oggetti e del comando
# per la compilazione
ese.exe: $(OBJ)
    $(CC) -o ese.exe $(OBJ) $(BLASLIB)
Makefile 1/2
```

A. Murli - Calcolo Scientifico
a.a.2004/2005

31

Esempio makefile per BLAS

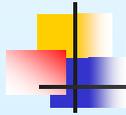
```
# definizione della dipendenza dei file sorgenti
# e specifica del comando per la compilazione
.f.o:
    $(FC) -c $*.f
.c.o:
    $(CC) -c $*.c

# definizione dell'istruzione per rimuovere il
# file core, i file.o e l'eseguibile
clean:
    rm -f core *.o ese.exe
Makefile 2/2
```

Tale target si esegue col
comando **make clean**

A. Murli
a.a.2004/2005

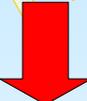
32



Esecuzione di un target

In generale in un Makefile
possono essere definiti
diversi target.

Per **eseguire un target**
descritto in un makefile
si utilizza il comando



```
make nome_target
```



In conclusione

Vantaggi nell'utilizzo del programma make

- Il make **non ricompila** files sorgenti non modificati.
- Il make **ricompila sempre** i files sorgenti modificati.



**L'utente non deve preoccuparsi
di tenere un elenco degli aggiornamenti effettuati.**