Comando `make` & Makefile

- Comando **`make`** per automatizzare la compilazione dei sorgenti

- Il comando `make` si basa su un *file di testo* chiamato Makefile

Comando `make` & Makefile

- Un Makefile è un file contenente al suo interno delle regole

- Le regole hanno la seguente struttura:

```
obiettivo: prerequisiti
	comando1
	comando2
	...
```

Comando `make` & Makefile

```
obiettivo: prerequisiti
(TAB)comando
```

- `obiettivo`: (in genere) il file da creare
- `prerequisiti`: i file necessari per creare l'obiettivo
- `comando`: i comandi necessari per creare l'obiettivo

NOTA: il carattere di tabulazione (TAB) obbligatorio all'inizio della riga corrispondente al comando

Comando `make` & Makefile

Sintassi make:

`make` : esegue la prima regola presente nel Makefile;

`make` **`target`** : esegue la regola relativa all'obiettivo target;

`make` [target] -f nomefile : l'opzione -f consente di specificare un file diverso da utilizzare come Makefile;

NOTA: `make` controlla se qualcuno dei `prerequisiti` è stato modificato più recentemente rispetto all'`obiettivo` e in caso affermativo esegue i `comandi`.

Comando `make` & Makefile

Esempio: (1/3)

`imperatore.txt` : file di testo contenete la riga 'Giovambattista Ianni';

`vassalli.txt` : file di testo contenete le righe 'Alessandra Martello' e 'Claudio Panetta';

`servi_gleba.txt` : file di testo contentente le righe 'studente1', 'studente2', ...

`gerarchia.txt` : file di testo con il contenuto di 'imperatore.txt', 'vassalli.txt' e 'servi_gleba.txt'

Comando `make` & Makefile

Esempio: (2/3)

Makefile:

```
gerarchia.txt: imperatore.txt vassalli.txt servi_gleba.txt
    cat imperatore.txt \
    vassalli.txt \
    servi_gleba.txt > gerarchia.txt
```

NOTA: il carattere ”back slash” (”\”) consente di spezzare il comando su più linee.

Comando `make` & Makefile

Esempio: (3/3)

da linea di comando scriviamo:

`make` (oppure `make gerarchia.txt`)

su output otteniamo:

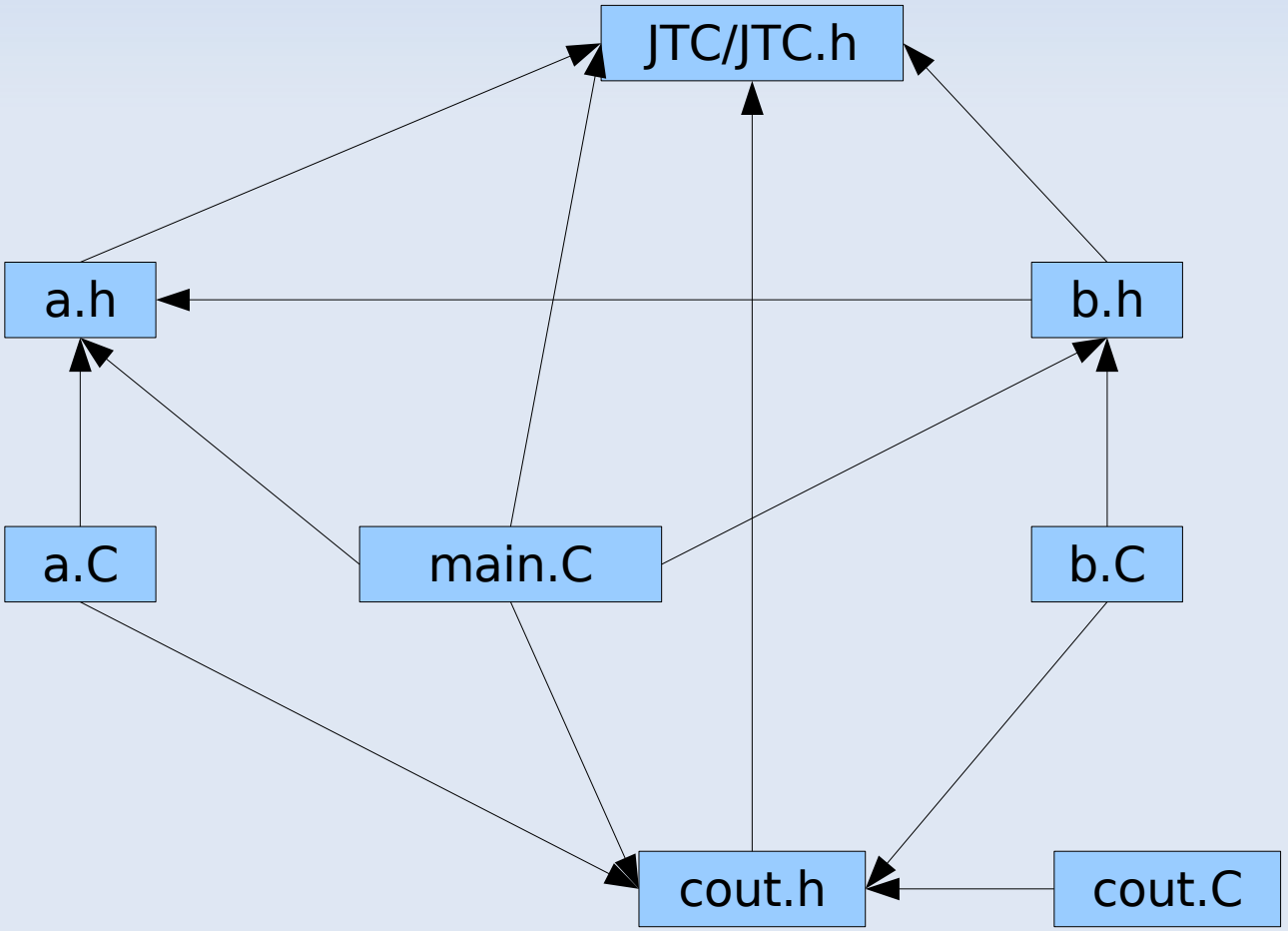
```
cat imperatore.txt \
        vassalli.txt  \
        servi_gleba.txt > gerarchia.txt;
```

oppure:

```
make: 'gerarchia.txt' is up to date.
```

Comando `make` & Makefile

Compilare i sorgenti?

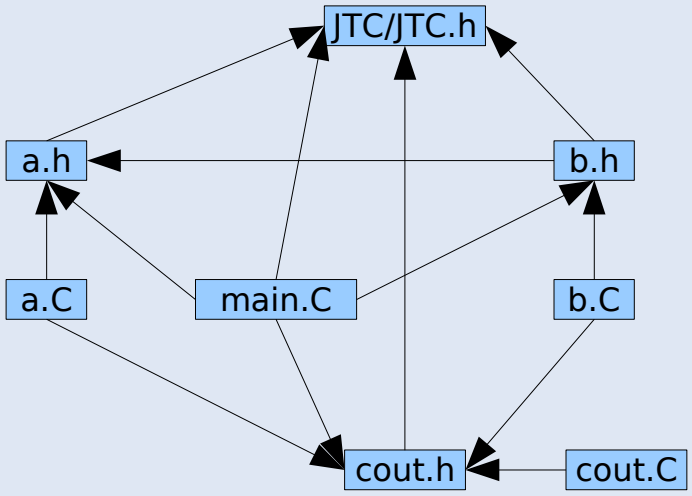


Comando `make` & Makefile

Compilare i sorgenti?

```
> g++ a.C b.C cout.C main.C -o eseguibile -lJTC -lpthread
```

```
> g++ -c main.C
> g++ -c a.C
> g++ -c b.C
> g++ -c cout.C
> g++ main.o a.o b.o cout.o -o eseguibile -lJTC -pthread
```



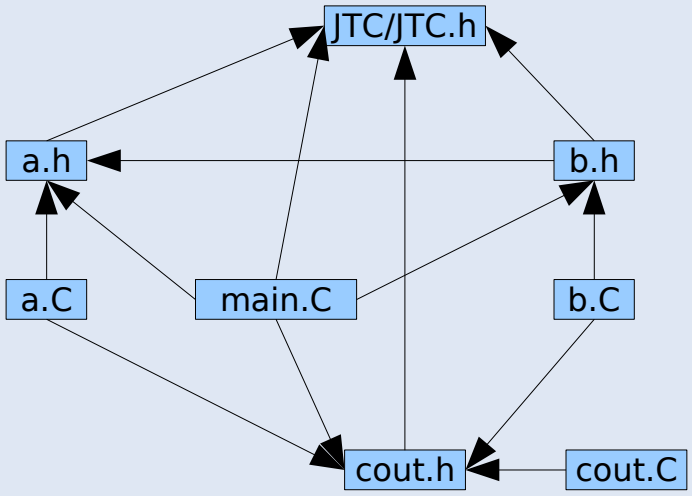
Comando `make` & Makefile

Compilare i sorgenti?

```
> g++ a.C b.C cout.C main.C -o eseguibile -lJTC -lpthread
```

Makefile:

```
eseguibile: a.C b.C cout.C main.C
   g++ a.C b.C cout.C main.C -o eseguibile -lJTC -lpthread
```



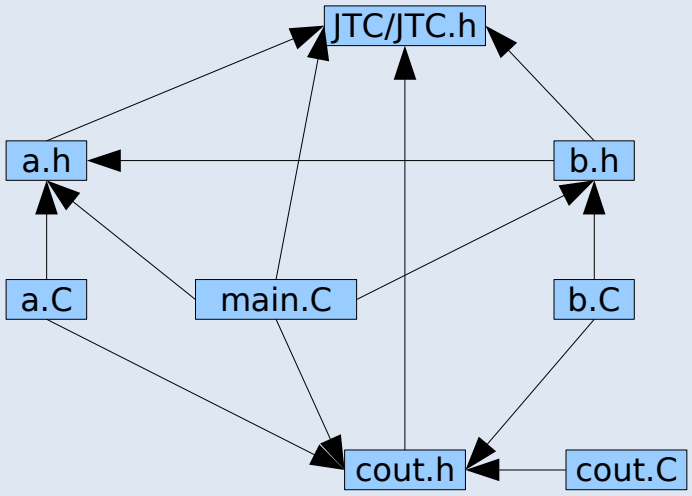
Comando `make` & Makefile

Compilare i sorgenti?

```
> g++ -c main.C
> g++ -c a.C
> g++ -c b.C
> g++ -c cout.C
> g++ main.o a.o b.o cout.o -o eseguibile -lJTC -pthread
```

Makefile:

```
eseguibile: a.o b.o cout.o main.o
   g++ a.o b.o cout.o main.o -o eseguibile -lJTC -lpthread
a.o: a.C cout.h a.h
   g++ -c a.C
b.o: b.C b.h a.h
   g++ -c b.C
cout.o: cout.C cout.h
   g++ -c cout.C
main.o: main.C a.h b.h cout.h
   g++ -c main.C
```



Comando `make` & Makefile

Variabili:

```
OBJS = a.o b.o cout.o main.o
LFLAG = -lJTC -lpthread

eseguibile: $(OBJS)
    g++ $(OBJS) -o eseguibile $(LFLAG)
```

Variabili speciali:

- `$@` : nome dell'obiettivo
- `$<` : nome del primo prerequisito
- `$^` : elenco di tutti i prerequisiti
- `$*` : prefisso in comune tra obiettivo e prerequisito
- `$?` : nome del prerequisito modificato

Comando `make` & Makefile

Regole implicite e regole di dipendenza:

```
OBJS = a.o b.o cout.o main.o
LFLAG = -lJTC -lpthread

eseguibile: $(OBJS)
   g++ $(OBJS) -o eseguibile $(LFLAG)

%.o: %.C
   g++ -c $< -o $@

a.o: a.C cout.h a.h

b.o: b.C b.h a.h

cout.o: cout.C cout.h

main.o: main.C a.h b.h cout.h
```