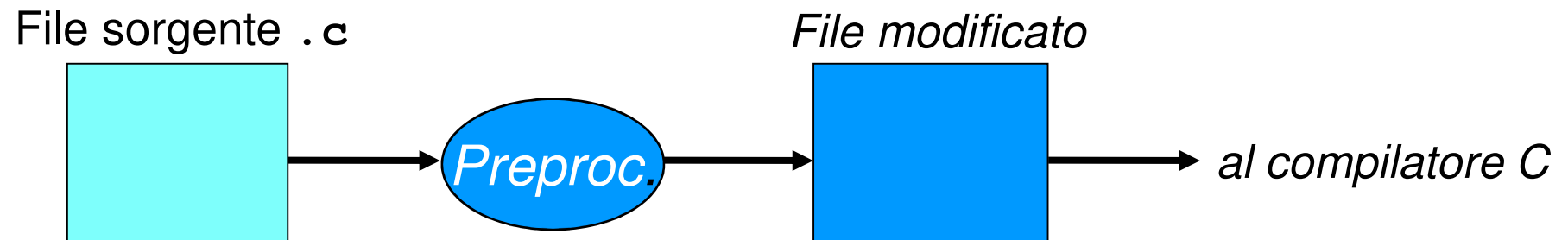


Preprocessore C

- Agisce ***prima del compilatore*** e processa direttamente il file sorgente
- ***Esegue direttive*** possibilmente contenute nel file sorgente
- Il compilatore riceve una versione “riveduta” del file sorgente



Preprocessore C

- Il preprocessore non è un compilatore:
 - Non conosce il linguaggio C
 - Non può interpretarne le istruzioni
 - Non può verificare la correttezza del programma
- Agisce esclusivamente sul testo del programma
 - Potrebbe manipolare qualunque testo...
 - ...anche codice sorgente di altri linguaggi!

Preprocessore C – che cosa e come?

- ***includere altre porzioni di testo***, prese da altri file
 - effettuare ***ricerche e sostituzioni*** (più o meno sofisticate, ad es. ***parametriche***) sul testo
 - ***inserire o sopprimere parti del testo*** a seconda del verificarsi di certe condizioni
- Comandato da ***direttive contenute nel sorgente stesso***
- Le direttive non sono istruzioni C e non ne hanno la sintassi
 - Le direttive vengono soppresse una volta elaborate (il compilatore non ne ha visibilità)

Preprocessore C - Direttive

- includere altre porzioni di testo

#include *nomefile*

- effettuare *ricerche e sostituzioni*

#define *testo1 testo2*

- *inserire o sopprimere parti del testo*

#ifdef *cond*

#ifndef *cond*

...testo...

...testo...

#endif

#endif

La direttiva `#define`

Sintassi:

```
#define    testo1    testo2
```

Effetto:

definisce una *regola di ricerca e sostituzione*:
ogni occorrenza di *testo1* verrà sostituita da
testo2

Scopo:

definire costanti simboliche (per convenzione, `testo1` si tende a scrivere con tutte maiuscole)

La direttiva #define

Prima del pre-processing:

```
#define RADICEDI2 1.4142F
int main() {
    float lato = 18;
    float diagonale = lato * RADICEDI2;
}
```

Dopo il pre-processing:

```
int main() {
    float lato = 18;
    float diagonale = lato * 1.4142F;
}
```

Preprocessore C

Attenzione:

- nell'effettuare ricerche e sostituzioni, il preprocessore **si limita a sostituire testo con altro testo**
- **non effettua controlli di nessun tipo**, né può farli: non è un compilatore, e dunque *non conosce la sintassi del C*
- Quindi, direttive sbagliate possono produrre *risultati privi di senso*

La direttiva `#define` – (controesempio)

Prima del pre-processing:

```
#define RADICEDI2 1.414paperino
int main() {
    float lato = 18;
    float diag = lato * RADICEDI2;
}
```

Dopo il pre-processing (errore sintattico):

```
int main() {
    float lato = 18;
    float diag = lato * 1.414paperino;
}
```


Le macro

La regola di ricerca e sostituzione introdotta dalla direttiva `#define` si chiama **macro**

Regole semplici, come le precedenti:

```
#define MAX 10
```

```
#define RADICEDIDUE 1.4142F
```

definiscono **macro semplici**

La direttiva `#define` permette però **anche di definire regole più complesse**, che vanno sotto il nome di **macro parametriche** (*che non vedremo in questo corso*)

Preprocessore C – #include

Sintassi:

```
#include <libreria.h>  
#include "miofile.h"
```

Effetto:

include il contenuto del file specificato
esattamente nella posizione in cui si trova la
direttiva stessa

(La differenza tra le due scritture sopra verrà discussa più avanti;
impatta solo su ***dove*** –in quale direttorio- il file incluso verrà cercato
dal preprocessore)