

Compilazione separata

Come realizzare correttamente un
piccolo progetto su piu' file

Programmi C su più file

- Tutti i programmi che abbiamo visto finora erano composti da un unico file .c
- Nel caso di programmi più grandi è conveniente suddividere il codice in più file:
 - migliore suddivisione e organizzazione del codice
 - facilità di manutenzione e correzione degli errori
 - possibilità di riutilizzare il codice per progetti diversi

Programmi C su più file

- Il linguaggio C e l'ambiente Unix forniscono tre strade per suddividere il codice:
 1. inclusione diretta di file sorgente e compilazione unica;
 2. inclusione di header file e compilazione separata;
 3. Inclusione di header file e uso di librerie statiche o dinamiche.
- Vediamo i tre casi con un esempio

Includo il .c: esempio

```
int somma (int x, int y) {  
    return x + y;  
}
```

File `somma.c`

```
#include <stdio.h>  
#include "somma.c"  
int main(void) {  
    int a, b, c;  
    scanf("%d %d", &a, &b);  
    c = somma(a,b);  
    printf("%d\n", c); return 0;  
}
```

File `main.c`

Includo il .c: esempio

- Per compilare basta usare il comando:

```
gcc -Wall -pedantic main.c
```

Vediamo cosa succede

1. Preprocessore: sostituisce alle prime due righe del file `main.c` una copia del contenuto del file `stdio.h` e una la copia del file `somma.c`

Includo il .c: esempio

```
/* qua la copia di stdio.h*/  
int somma (int x, int y) {  
    return x + y;  
}  
  
int main(void) {  
    int a, b, c;  
    scanf ("%d %d", &a, &b) ;  
    c = somma (a,b) ;  
    printf ("%d\n", c) ; return 0 ;  
}
```

File
Effettivamente
compilato

Contiene tutto
Il codice della
Funzione somma

Includo il .c: esempio

- Per compilare basta usare il comando:

```
gcc -Wall -pedantic main.c
```

Vediamo cosa succede

1. Preprocessore: sostituisce alle prime due righe del file `main.c` una copia del contenuto del file `stdio.h` e una la copia del file `somma.c`
2. Il compilatore genera il codice binario sia per la funzione `somma()` che per il `main()` esattamente come se avessi scritto tutto in un singolo file dall'inizio

Includo il .c: esempio

- Svantaggi:
 - Devo avere il codice sorgente di tutte le funzioni che uso
 - Se fosse possibile solo questo dovrei avere sempre disponibile il codice di tutte le funzioni di libreria, ad esempio **printf()** , **scanf()**
 - Ogni volta devo ricompilare tutto dall'inizio anche se non ho modificato niente

Includo il .h: esempio

```
int somma (int x, int y) {  
    return x + y;  
}
```

File somma.c

```
int somma (int x, int y);
```

File somma.h

```
#include <stdio.h>  
#include "somma.h"  
int main(void) {  
    int a, b, c;  
    scanf("%d %d", &a, &b);  
    c = somma(a,b);  
    printf("%d\n", c); return 0;  
}
```

File main.c

Includo il .h: esempio

- In questo caso non è necessario avere disponibile il codice sorgente della funzione somma ma basta il precompilato (modulo oggetto):
 - Si può creare il modulo oggetto con
`gcc -Wall -pedantic -c somma.c`
Questo crea un file `somma.o` con il precompilato (binario)
 - e poi mettere a disposizione solo `somma.h` e `somma.o` a chi ha bisogno della funzione

Includo il .h: esempio

- Supponiamo quindi `somma.o` e `somma.h` disponibili e vediamo come compilare il main:
 - Basta creare il modulo oggetto anche per il main
- ```
gcc -Wall -pedantic -c main.c
```
1. Questo copia le informazioni del .h all'inizio del file

# Includo il .h: esempio

```
/* qua la copia di stdio.h*/
int somma (int x, int y);
int main(void) {
 int a, b, c;
 scanf("%d %d", &a, &b);
 c = somma(a,b);
 printf("%d\n", c); return 0;
}
```

**File**  
**Effettivamente**  
**compilato**

Contiene solo  
l'intestazione della  
Funzione somma

# Includo il .h: esempio

- Supponiamo quindi `somma.o` e `somma.h` disponibili e vediamo come compilare il main:
    - Basta creare il modulo oggetto per il main
- ```
gcc -Wall -pedantic -c main.c
```
1. Questo copia le informazioni del .h all'inizio del file
 2. Con le informazioni sul tipo della funzione il compilatore può controllare che la funzione sia chiamata in modo corretto e creare un modulo oggetto anche per il main `main.o`

Includo il .h: esempio

- Supponiamo ...vediamo come compilare il main:

- Basta creare il modulo oggetto per il main

```
gcc -Wall -pedantic -c main.c
```

- E poi collegarlo al codice del modulo oggetto di somma ed alle librerie (linking)

```
gcc main.o somma.o -o ese
```

Le due fasi possono anche essere raggruppate in un unico comando

```
gcc -Wall -pedantic main.c somma.o -o ese
```

Includo il .h: esempio

- Vantaggi:
 - Non devo avere il codice sorgente di tutte le funzioni che uso (bastano tipi e prototipi nel .h)
 - Ogni volta non devo ricompilare tutto dall'inizio anche se non ho modificato
- Svantaggi:
 - Se ho molti file.o devo ricordarmi di specificarli tutti altrimenti la compilazione fallisce
 - Soluzione, raggruppare tutti i .o in una libreria (come stdio.h)

Includo il .h + libreria: esempio

```
int somma (int x, int y) {  
return x + y;  
}
```

File somma.c

```
int somma (int x, int y)
```

File somma.h

```
#include <stdio.h>  
#include "somma.h"  
int main(void) {  
int a, b, c;  
scanf("%d %d", &a, &b);  
c = somma(a,b);  
printf("%d\n", c); return 0;  
}
```

File main.c

Includo il .h + libreria: esempio

- Anche in questo caso creo il modulo oggetto

- con

```
gcc -Wall -pedantic -c somma.c
```

- E poi creo la libreria `libSomma.a`

```
ar r libSomma.a somma.o
```

- Se avessi più `.o` potrei metterli tutti nella stessa libreria
es:

```
ar r libOpInt.a somma.o diff.o div.o
```

Includo il .h + libreria: esempio

- Supponiamo quindi **libSomma.a** disponibile vediamo come compilare il main:
 - Basta indicare la libreria dove prendere il codice per le funzioni mancanti

```
gcc -Wall -pedantic main.c -L. -lSomma
```

1. Questo crea il **main.o**
2. E va a prendere il codice per le funzioni usate ma non definite nei .o delle librerie, sia quelle standard che quelle linkate esplicitamente
 - Basta indicare la cartella e la parte centrale del nome (tutte le librerie C iniziano per **lib** e terminano per **.a**)