Preprocessore

Corso di Programmazione di Sistema

Nicola Bicocchi

DIEF/UNIMORE

Aprile 2021

Hello World! Direttiva include

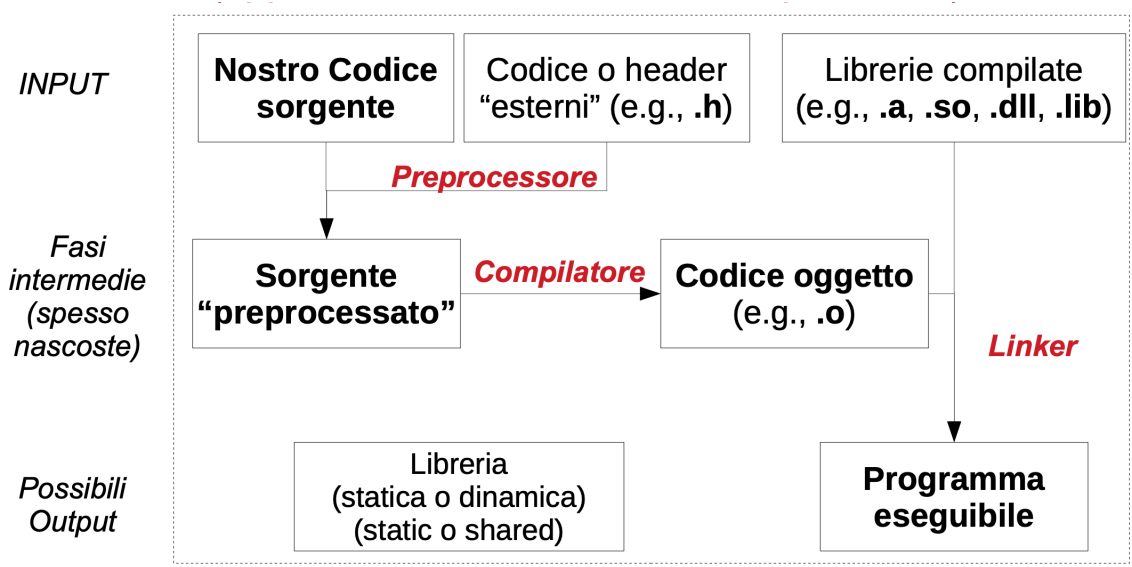
Linea 1: **#** introduce una direttiva del pre-processore che **include (importa)** un file (**stdio.h**) da un percorso standard (**< >**)

```
1 #include <stdio.h>
2
3 int main(){
4     printf("Hello, World!\n");
5     return 0;
6 }
```

Direttive preprocessore

- Il simbolo **#** (diesis, cancelletto, o hash) precede tutte le operazioni che vengono gestite dal *preprocessore*
- Sono operazioni eseguite durante il processo di compilazione
- Servono, in particolare, a manipolare il codice sorgente *prima* della compilazione vera e propria. In particolare, sono utilizzate per aggiungere, modificare ed escludere parti di codice sorgente

Processo di compilazione



Preprocessore

- Non è un comando o programma aggiuntivo, ma solo un termine con cui si definisce questa fase del processo di compilazione
- La si può immaginare come una procedura di manipolazione di testo: l'input è codice sorgente l'output è codice sorgente
- Le direttive al preprocessore non esistono più nel codice sorgente che viene effettivamente compilato nè tantomento nel codice compilato. Si tratta di un meccanismo per manipolare il codice sorgente prima della compilazione
- Per mostrare/salvare solo l'output del preprocessore si può usare:

```
1 $ gcc -E filename.c
```

Direttive rilevanti

#include aggiunta di codice

#define modifica del codice

#ifdef (#ifndef) #else #endif esclusione condizionata di codice

Direttiva #include

- Aggiunge codice sorgente presente in file esterni
- Utilizzato di solito con file header con estensione .h

```
1 #include <stdio.h>
```

- Il preprocessore genera un file sorgente intermedio in cui la direttiva #include <stdio.h> è sostituita dal contenuto dell'intero file indicato (/usr/include, /usr/local/include)

```
1 $ cat /usr/include/stdio.h
```

Progetto composto di più file

- Mettiamo i nostri header in percorsi di sistema (e.g., /usr/include, /usr/local/include). Approccio utile solo per la distribuzione di librerie, non durante lo sviluppo
- Indichiamo al compilatore gcc di cercare in percorsi arbitrari con l'opzione di compilazione -I

```
1 $ ...
```

- Indichiamo al compilatore gcc di cercare localmente il file

```
1 #include "mymath.h"
```

Esempio (mymath.h e mymath.c)

- All'interno del file .h posizioniamo i prototipi delle funzioni

```c
unsigned mypow(unsigned int base, unsigned int exp);
```

- All'interno del file .c posizioniamo le implementazioni delle funzioni

```c
#include "mymath.h"

unsigned mypow(unsigned int base, unsigned int exp){
    unsigned int result;
    for(result=1; exp>0 ; exp--){
        result*=base;
    }
    return result;
}
```

Esempio (main.c)

```c
#include <stdio.h>
#include "mymath.h"

int main(){
    unsigned int base=3,exp=5;
    printf("%u^%u=%u\n", base, exp, mypow(base, exp));
    return 0;
}
```

```
cmake_minimum_required(VERSION 3.15)
project(mymath C)
set(CMAKE_C_STANDARD 99)

add_executable(main main.c mymath.c)
```

Direttiva #define

- La direttiva #define INPUT OUTPUT modifica il codice sorgente sostituendo tutte le occorrenze di INPUT con OUTPUT
- *Attenzione: #define non definisce variabili e/o funzioni globali!*

```c
#define INPUT OUTPUT
```

```c
#include <stdio.h>
#define MAX 10
int main() {
    printf("Il valore massimo é %d\n", MAX);
    return 0;
}
```

Direttiva #define parametrica

- Il valore di sostituzione può anche essere parametrico

```c
#define N(x) (10*(x))
int main(){
    int a=N(5);
    return 0;
}
```

- *Attenzione: N non è una funzione!*
- Il preprocessore crea un nuovo codice sorgente sostituendo tutte le occorrenze di N(argomento) con (10 * (argomento)) senza effettuare alcun controllo!
- E' importante utilizzare in modo opportuno le parentesi per evitare problemi di priorità!

Parentesi

- Una definizione robusta si ottiene mettendo tra parentesi sia gli argomenti da sostituire che la macro stessa

```
1 #define SQUARE(a) a*a
2
3 SQUARE(3);  // Output:9
4 SQUARE(1+2) // Sostituito in s = 1+2*1+2 -> Output:5
```

- Definizione robusta

```
1 #define SQUARE(a) ((a)*(a))
```

Direttiva #undef

- La direttiva #define non ha scope
- Il preprocessore non conosce le funzioni o altri costrutti di aggregazione/visibilità del linguaggio. Legge i sorgenti e opera in modo sequenziale dall'inizio alla fine del file sorgente
- Per eliminare una definizione è necessario utilizzare la direttiva #undef in modo esplicito

```
1 #define MAX 128
2 ...
3 printf("%d\n", MAX);
4 ...
5 #undef MAX
6 ...
7 printf("%d\n", MAX);     /* Errore! */
```

Direttiva #if

- E' possibile abilitare o disabilitare porzioni di codice attraverso l'uso del preprocessore e la direttiva #if
- Il preprocessore non può valutare il contenuto di variabili o di codice a runtime
- Le uniche *variabili* nel preprocessore sono quelle definite attraverso la direttiva #define (macro)
- Utile per: (a) scegliere tipi diversi di implementazione, (b) stabilire tipi di dati utilizzati, (c) stabilire comportamenti speciali per la fase di sviluppo (debug)

```
1 #define MAX 10 \
2 #if (MAX==1024)
3     // implementazione 1
4 #else
5     // implementazione 2
6 #endif
```

Direttiva #if (Esempi)

- message(S) stampa il messaggio solo se la MACRO DEBUG è definita

```
#ifndef DEBUG
#define message(S) printf(S) \
#else
#define message(S)
#endif
```

- Include guards

```
#ifndef MYMATH_H
#define MYMATH_H

unsigned mypow(unsigned, unsigned);

#endif // MYMATH_H
```