Alessio Bechini
- Corso di Fondamenti di Informatica II -

Cenni sul preprocessore e il suo utilizzo

Il preprocessore

- Storicamente, i compilatori C operavano con passaggi multipli.
- Il primo passaggio eseguiva una *elaborazione preventiva* del testo del programma, e produceva un file intermedio destinato ai successivi passaggi di compilazione vera e propria.
- Il meccanismo responsabile di questa prima passata prende il nome di *preprocessore*.
- Si possono dare istruzioni al preprocessore attraverso apposite direttive, da inserire direttamente nel codice sorgente del programma.


Il preprocessore: operazioni effettuate
Fondamenti di Informatica II
• Inserimento dei *file di inclusione*, ovvero di file di testo che contengono prototipi di funzioni di librerie standard o definite dall'utente.
• Esame delle *direttive di compilazione* per effettuare compilazioni condizionali (per aumentare la portabilita` e per le sessioni di debugging).
• Espansione delle *macro*, che riducono lo sforzo di programmazione e migliorano la leggibilita` del codice
Il preprocessore: cenni 3


Le direttive
Fondamenti di Informatica II
• Si possono dare istruzioni al preprocessore attraverso le cosiddette *direttive*.
• Qualsiasi riga che inizi con il carattere # viene considerata una direttiva (eccezion fatta per la sua presenza in una stringa, in un commento, ecc.).
• La direttiva piu` semplice e` la direttiva nulla, ovvero la riga contenente esclusivamente il carattere #. La direttiva nulla viene sempre ignorata dal preprocessore.
Il preprocessore: cenni 4

Direttiva #define

Fondamenti di Informatica II

- La direttiva #define definisce una *macro*, ovvero permette di individuare un identificatore che verra` espanso dal preprocessore secondo quanto specificato nella direttiva.

```
//         *** ESEMPIO ***
#define SALUTO “Buona giornata”
#define vuoto “”
[..]
cout << SALUTO;
  // espanso: out << “Buona giornata”;
cout << “vuoto”;
  // non viene espanso
```


Macro per sostituzione di costrutti

Fondamenti di Informatica II

- Utilizzare macro per la sostituzione su larga scala di costrutti C++ e` indice di cattiva programmazione

```
//  *** ESEMPIO ***

#define BEGIN {
#define END }
#define MAX_LIM
[..]
while(k < MAX_LIM)
   BEGIN
      [..]
   END
[..]
```

UTILIZZO CONSIGLIATO

UTILIZZO VIVAMENTE SCONSIGLIATO

Insidie nell'uso di #define

Fondamenti di Informatica II

```
//        *** ESEMPIO ***

#include <iostream.h>
#define COMP_1    2
#define COMP_2    5
#define COMP_TOT  COMP_1 + COMP_2

main() {
   cout << "Quadrato di COMP_TOT: " \
      << COMP_TOT * COMP_TOT << \n;
   return(0);
}
```

COSA STAMPA?
COSA VOLEVA IL PROGRAMMATORE
COME RISULTATO?

Il preprocessore: cenni 7

#define e #undef

Fondamenti di Informatica II

```
//             *** ESEMPIO ***

#define BLOCK_SIZE  512
[..]
dimBuff=nBlocks*BLOCK_SIZE;
  // espanso in dimBuff=nBlocks*512;
[..]
#undef BLOCK_SIZE
  // l'uso di BLOCK_SIZE non e` ora consentito
[..]
#define BLOCK_SIZE 128
[..]
dimBuff=nBlocks*BLOCK_SIZE;
  // espanso in dimBuff=nBlocks*128;
[..]
```

ERRORE COMUNE:
`#define BLOCK_SIZE = 512`

Il preprocessore: cenni 8

Macro con parametri

Fondamenti di Informatica II

- La sintassi per definire macro con parametri e` la seguente:
  #define *id_macro*(<*elenco_arg_formali*>) *seq_token*
- Per richiamare tali macro si utilizza la forma:
  id_macro<*spazio_vuoto*>(<*elenco_arg_attuali*>)
- Una chiamata di macro provoca due serie di sostituzioni:
  - l'id_macro e i relativi argomenti vengono sostituiti dalla corrispondente sequenza di token
  - gli argomenti formali nella sequenza di token sostituita vengono rimpiazzati dai corrispondenti argomenti attuali

Il preprocessore: cenni 9

Macro con parametri: esempio

Fondamenti di Informatica II

```
//              *** ESEMPIO ***

#define CUBO(x) ((x)*(x)*(x))
[..]
int n, y;
[..]
n = CUBO(y);
[..]
```

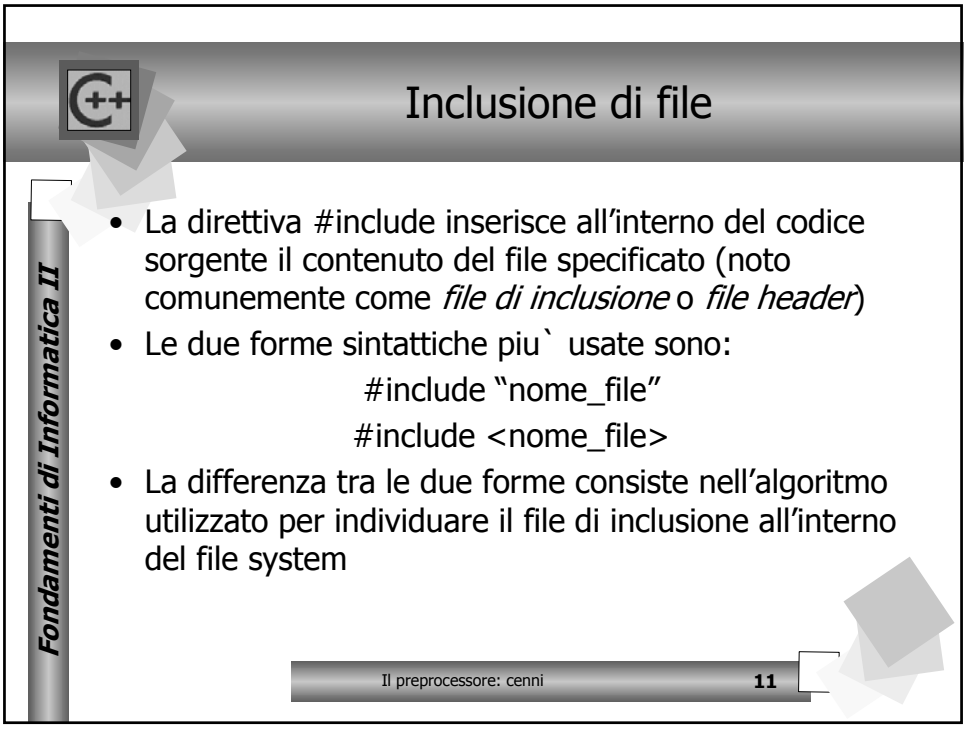
Perche' tale abbondanza di parentesi?

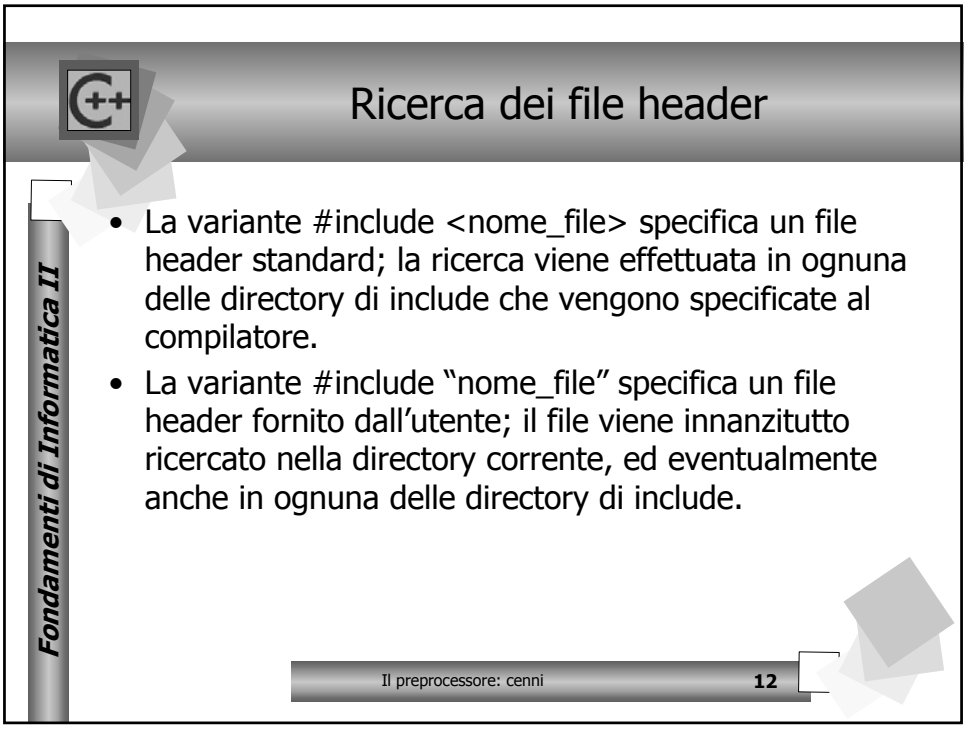
ATTENZIONE:
questa sembra una chiamata di funzione, ma non si effettuano controlli di tipo!

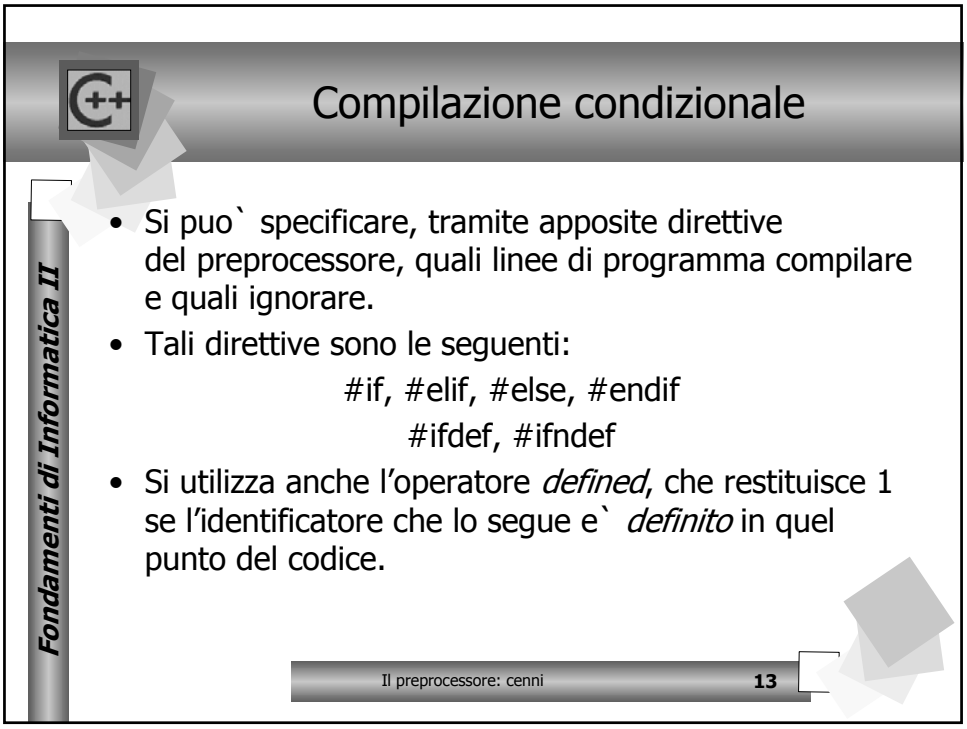
Cosa accade se si richiama
n = CUBO(y++);
?

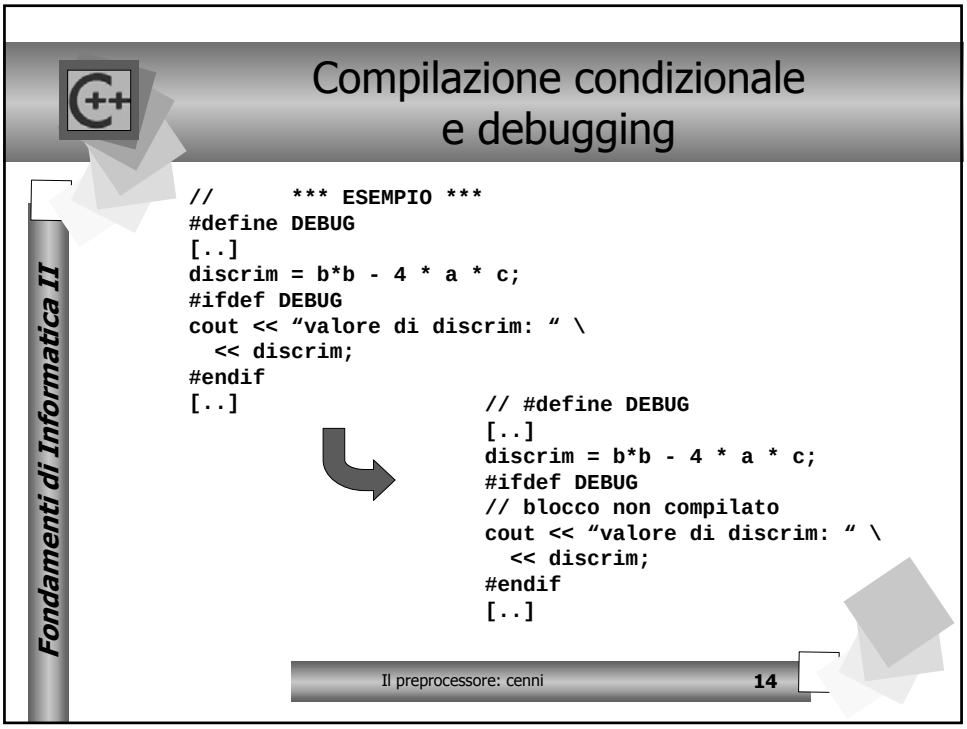
USO INSIDIOSO

Il preprocessore: cenni 10


Inclusione di file
Fondamenti di Informatica II
• La direttiva #include inserisce all'interno del codice sorgente il contenuto del file specificato (noto comunemente come *file di inclusione* o *file header*)
• Le due forme sintattiche piu` usate sono:
#include "nome_file"
#include <nome_file>
• La differenza tra le due forme consiste nell'algoritmo utilizzato per individuare il file di inclusione all'interno del file system
Il preprocessore: cenni
11


Ricerca dei file header
Fondamenti di Informatica II
• La variante #include <nome_file> specifica un file header standard; la ricerca viene effettuata in ognuna delle directory di include che vengono specificate al compilatore.
• La variante #include "nome_file" specifica un file header fornito dall'utente; il file viene innanzitutto ricercato nella directory corrente, ed eventualmente anche in ognuna delle directory di include.
Il preprocessore: cenni
12


Compilazione condizionale
Fondamenti di Informatica II
• Si puo` specificare, tramite apposite direttive del preprocessore, quali linee di programma compilare e quali ignorare.
• Tali direttive sono le seguenti:
#if, #elif, #else, #endif
#ifdef, #ifndef
• Si utilizza anche l'operatore *defined*, che restituisce 1 se l'identificatore che lo segue e` *definito* in quel punto del codice.
Il preprocessore: cenni 13


Compilazione condizionale e debugging
Fondamenti di Informatica II
// *** ESEMPIO ***
#define DEBUG
[..]
discrim = b*b - 4 * a * c;
#ifdef DEBUG
cout << "valore di discrim: " \
<< discrim;
#endif
[..]
// #define DEBUG
[..]
discrim = b*b - 4 * a * c;
#ifdef DEBUG
// blocco non compilato
cout << "valore di discrim: " \
<< discrim;
#endif
[..]
Il preprocessore: cenni 14

Progetti su più file

Fondamenti di Informatica II

- Il meccanismo di inclusione dei file viene correntemente usato per suddividere un programma su piu` file.
- In C++, tipicamente per ogni classe si crea un file di codice (.cpp o .cc) e uno con le dichiarazioni (.h).
- In un file si possono includere altri file, indipendentemente dal fatto che contengano dichiarazioni o definizioni.
- Nel caso di progetti software complessi, si deve evitare di fare inclusioni multiple.

Evitare inclusioni multiple

Fondamenti di Informatica II

```
/*** file principale ***/

#include "MIO_MODULO.h"
#include "MIO_SOTTOMODULO.h"

[..]
```

```
/*** file MIO_MODULO.h ***/

#include "MIO_SOTTOMODULO"

[..]
```

```
/*** file MIO_SOTTOMODULO.h ***/

#ifndef MIO_SOTTOMODULO
#define MIO_SOTTOMODULO

[..]

#endif
```