

Capitolo 13 – Il preprocessore

Outline

Introduzione

La direttiva #include

La direttiva #define: costanti simboliche

La direttiva #define: le macro

Compilazione condizionata

Le direttive #error e #pragma

Gli operatori # e

Numeri di linea

Costanti simboliche predefinite

Assertzioni

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Obiettivi

- In questo capitolo, impareremo a:
 - utilizzare #include per lo sviluppo di grandi programmi.
 - Utilizzare #define per la creazione di macro.
 - Capire la compilazione condizionata.
 - Visualizzare messaggi di errore durante la compilazione.
 - Utilizzare le asserzioni per testare se i valori di una espressione sono corretti.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Introduzione

- Preprocessing
 - Si verifica prima di compilare un programma
 - Inclusione di altri file
 - Definizione di costanti simboliche e macro
 - Compilazione condizionata
 - Esecuzione condizionata delle direttive
- Formato delle direttive
 - Linee che iniziano con #
 - Solo caratteri di spazio prima delle direttive su una linea

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

La direttiva #include

- #include
 - Copia uno specifico file dove compare la direttiva
 - #include <filename>
 - Inclusione di librerie standard
 - #include "filename"
 - Inclusione di file definiti dall'utente
 - Utilizzata per:
 - Programmi con più file sorgenti da compilare assieme
 - File Header – hanno delle definizioni e delle dichiarazioni comuni (classi, strutture, prototipi di funzione)
 - #include in ogni file

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

La direttiva #ifndef: costanti simboliche

- #define
 - Direttiva utilizzata per creare costanti simboliche e macro
 - Costanti simboliche
 - Quando un programma viene compilato, tutte le occorrenze della costante simbolica vengono rimpiazzate nel testo
 - Formato

```
#define identifier replacement-text
```
 - Esempio:

```
#define PI 3.14159
```
 - Ciò che è a destra dell'identificatore rimpiazza il testo

```
#define PI = 3.14159
```

 - Rimpiazza "PI" con "= 3.14159"
 - Non si possono ridefinire le costanti simboliche una volta

create
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

La direttiva #include: le macro

- Le macro
 - Operazioni definite in #define
 - Una macro senza argomenti viene trattata come una costante simbolica
 - Una macro con argomenti ha i suoi argomenti sostituiti al testo degli argomenti, quando espansa
 - Effettua la sostituzione del testo – nessun data type checking
 - La macro

```
#define CIRCLE_AREA( x ) ( PI * ( x ) * ( x ) )
```

causa

```
area = CIRCLE_AREA( 4 );
```

per diventare

```
area = ( 3.14159 * ( 4 ) * ( 4 ) );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

La direttiva #include: le macro

- Uso di parentesi
 - Senza parentesi la macro

```
#define CIRCLE_AREA( x ) PI * ( x ) * ( x )
```

causerebbe

```
area = CIRCLE_AREA( c + 2 );
```

Per diventare

```
area = 3.14159 * c + 2 * c + 2;
```
- Argomenti multipli

```
#define RECTANGLE_AREA( x, y ) ( ( x ) * ( y ) )
```

causerebbe

```
rectArea = RECTANGLE_AREA( a + 4, b + 7 );
```

Per diventare

```
rectArea = ( ( a + 4 ) * ( b + 7 ) );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

La direttiva #include: le macro

- #undef
 - Elimina la definizione di una costante simbolica o di una macro
 - Se una costante simbolica o una macro non era definita può essere ridefinita

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Compilazione condizionata

• Compilazione condizionata

- Controllano le direttive e la compilazione
- Espressioni di cast, si zeof, enumerazioni di costanti non possono essere valutate nelle direttive
- Struttura simile alla `i f`

```
#i f !defi ned( NULL )
#defi ne NULL 0
#endi f
```

 - Determina se la costante simbolica NULL è stata definita
 - se NULL è definita, `defi ned(NULL)` vale 1
 - se NULL non è definita, questa funzione definisce NULL a 0
- Ogni `#i f` deve terminare con `#endi f`
- `#i fdef` è una abbreviazione per `#i f defi ned(name)`
- `#i fndef` è una abbreviazione per `#i f !defi ned(name)`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Compilazione condizionata

• Altre istruzioni

- `#el i f` – è equivalente a `el se i f` in una istruzione `i f`
- `#el se` – è equivalente a `el se` in una istruzione `i f`

• Codice "Comment out"

- Non si può utilizzare `/* ... */`
- Utilizzare

```
#i f 0
    code commented out
#endi f
```
- Per attivare il codice, cambiare 0 con 1

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Compilazione condizionata

• Debugging

- ```
#defi ne DEBUG 1
#i fdef DEBUG
 cerr << "Vari abl e x = " << x << endl ;
#endi f
```
- Definendo DEBUG a 1 attiva il codice
  - Dopo aver corretto il codice, rimuovere l'istruzione `#defi ne`
  - Le istruzioni di debugging ora vengono ignorate

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Le direttive `#error` e `#pragma`

### • `#error` tokens

- I tokens sono sequenze di caratteri separate da spazi
  - `"I l i ke C++"` ha 3 token
- Visualizza un messaggio che include i token specificati come messaggio d'errore
- Si ferma e previene la compilazione del programma

### • `#pragma` tokens

- provoca un'azione definita dall'implementazione (consult compiler documentation)
- una direttiva `pragmas` non riconosciuta dal compilatore sarà ignorata

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Gli operatori # e ##

- #
  - Causa un rimpiazzamento di testo convertito in stringa fra virgolette
  - L'istruzione

```
#define HELLO(x) printf("Hello, " #x "\n");
```

causa

```
HELLO(John)
```

Per diventare

```
printf("Hello, " "John" "\n");
```
  - Le stringhe separate da spazi bianchi vengono concatenate quando si usa la `printf`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Gli operatori # e ##

- ##
  - Concatena due token
  - L'istruzione

```
#define TOKENCONCAT(x, y) x ## y
```

causa

```
TOKENCONCAT(0, K)
```

Per diventare

```
OK
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Numeri di linea

- `#line`
  - Rinumerata le successive linee di codice, iniziando col valore intero
  - Possono essere inclusi i nomi di file
  - `#line 100 "myFile.c"`
    - Le linee sono numerate da 100 iniziando dal successivo file sorgente
    - I messaggi del compilatore faranno riferimento a "myFile.c"
    - Rende gli errori più significativi
    - I numeri di linea non appaiono nel file sorgente

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Costanti simboliche predefinite

- Quattro costanti simboliche predefinite
  - Non si possono usare in `#define` o `#undef`

| Symbolic constant     | Description                                                                                                        |
|-----------------------|--------------------------------------------------------------------------------------------------------------------|
| <code>__LINE__</code> | The line number of the current source code line (an integer constant).                                             |
| <code>__FILE__</code> | The presumed name of the source file (a string).                                                                   |
| <code>__DATE__</code> | The date the source file is compiled (a string of the form " <b>Mmm dd yyyy</b> " such as " <b>Jan 19 2001</b> "). |
| <code>__TIME__</code> | The time the source file is compiled (a string literal of the form " <b>hh:mm:ss</b> ").                           |

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Asserzioni

- `assert` macro
  - Header `<assert.h>`
  - Valore di test di una espressione
  - Se 0 (falso) stampa un messaggio di errore e invoca `abort`
  - Esempio:

```
assert(x <= 10);
```
  - se `NDEBUG` è definita
    - Tutte le successive istruzioni `assert` vengono ignorate