Programmazione (imperativa)

Corso di Laurea in Informatica

Roberto Cordone

DI - Università degli Studi di Milano

Lezioni:	Lunedì 12.00 - 13.00 e 14.00 - 16.00	Mercoledì 14.00 - 17.00
Laboratorio:	Giovedì 12.00 - 13.00 e 14.00 - 17.00	
Ricevimento:	su appuntamento	
Tel.:	02 503 16235	
E-mail:	roberto.cordone@unimi.it	
Web page:	http://homes.di.unimi.it/~cordone/courses/2015-prog/2015-prog.html	

Macro dotate di parametri

`#define` *identificatore*$(p_1, \ldots, p_n)$ *espressione*

dove $p_1, \ldots, p_n$ si chiamano parametri

Una volta definita, si può invocare la macro scrivendo:

identificatore$(a_1, \ldots, a_n)$

dove $a_1, \ldots, a_n$ si chiamano argomenti

Il precompilatore

- sostituisce l'identificatore con l'espressione ovunque appaia nei file che includono la macro (come per le macro semplici)
- sostituisce i parametri con gli argomenti della chiamata: p_1 con a_1, ..., p_n con a_n

N.B.: Non lasciare spazi fra identificatore e (. Perché?

Macro e parentesi

Anche le macro parametriche sono brutali sostituzioni di testo

È buona norma quindi racchiudere fra parentesi

- l'espressione dell'intera macro
- le occorrenze dei parametri nel testo della macro

per evitare effetti indesiderati delle regole implicite di precedenza quando il precompilatore procede a espandere la macro

Data la macro

```
#define PARI(n) (n % 2 == 0)
```

la chiamata

```
if (PARI(i+2)) i++;
```

diventa

```
if ((i+2 % 2 == 0)) i++;
```

dove `i+2 % 2` vale `i+0`, che in genere non è nullo

Esempi

Date le seguenti macro:

```
#define MAX(x,y) ((x) > (y) ?  (x) :  (y))
#define PARI(n) ((n) % 2 == 0)
```

è possibile chiamarle scrivendo:

```
i = MAX(j+k,j-k);
if (PARI(i+2)) i++;
```

Il precompilatore sostituisce le chiamate con

```
i = ((j+k) > (j-k) ?  (j+k) :  (j-k));
if (((i+2) % 2 == 0)) i++;
```

Si noti l'abbondanza di parentesi: sono necessarie?

Macro e funzioni

Rispetto alle funzioni, le macro

- sono trattate dal precompilatore (le funzioni dal compilatore)
- sono rimpiazzate testualmente dalla definizione (non chiamate)
- non hanno celle di memoria dedicate a parametri e variabili locali
- sono più veloci (nessuna allocazione e passaggio di parametri)
- creano codice più grosso (il corpo si ripete a ogni chiamata)
- non controllano il tipo degli argomenti
- funzionano con argomenti di tipo indeterminato

Si tratta di oggetti completamente diversi