Università degli Studi di Milano
Corso di Laurea in Informatica, A.A. 2017-2018

Procedure ricorsive

Homepage del corso

Turno A
Nicola Basilico
Dipartimento di Informatica
Via Comelico 39/41 - 20135 Milano (MI)
Ufficio S242
nicola.basilico@unimi.it
+39 02.503.16294

Turno B
Jacopo Essenziale
Dipartimento di Informatica
Via Celoria 20 - 20133 Milano (MI)
AISLab
jacopo.essenziale@unimi.it
+39 02.503.14010

Ricorsione

- La risoluzione di un problema P è costruita sulla base della risoluzione di un sottoproblema di P

- Esempio classico: il fattoriale di n

$$n! = \prod_{k=1}^{n} k = n \prod_{k=1}^{n-1} k = n \times (n-1)!$$

- il fattoriale di n è uguale a n moltiplicato per il fattoriale di n-1, non vero se n=0! Serve una regola aggiuntiva

$$n! = \begin{cases} n \times (n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0. \end{cases}$$

Ricorsione

- Applico la regola in cascata

$$n! = \begin{cases} n \times (n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0. \end{cases}$$

$$4! = 4 \times (3)!$$

Ricorsione

- Applico la regola in cascata

$$n! = \begin{cases} n \times (n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0. \end{cases}$$

$$4! = 4 \times \underbrace{(3)!}_{3! = 3 \times (2)!}$$

Ricorsione

- Applico la regola in cascata

$$n! = \begin{cases} n \times (n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0. \end{cases}$$

$$4! = 4 \times (3)!$$

$$3! = 3 \times (2)!$$

$$2! = 2 \times (1)!$$

Ricorsione

- Applico la regola in cascata

$$n! = \begin{cases} n \times (n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0. \end{cases} \quad \cdots$$

$$4! = 4 \times \underbrace{(3)!}$$

$$3! = 3 \times \underbrace{(2)!}$$

$$2! = 2 \times \underbrace{(1)!}$$

$$1! = 1 \times (0)!$$

Ricorsione

- Applico la regola in cascata

$$n! = \begin{cases} n \times (n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0. \end{cases}$$

$$4! = 4 \times (3)!$$

$$3! = 3 \times (2)!$$

$$2! = 2 \times (1)!$$

$$1! = 1 \times (0)!$$

$$0! = 1$$

Ricorsione

- Applico la regola in cascata

$$n! = \begin{cases} n \times (n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0. \end{cases}$$

$$4! = 4 \times (3)!$$

$$3! = 3 \times (2)!$$

$$2! = 2 \times (1)!$$

$$1! = 1 \times (0)!$$

$$0! = 1$$

$$= 4 \times 3 \times 2 \times 1 \times 1$$

Procedure ricorsive

- Procedura ricorsiva: è una procedura che per risolvere il problema P invoca se stessa per risolvere un sotto-problema di P

- In generale una procedura ricorsiva non è una procedura foglia: invoca se stessa per sua definizione

- Una procedura ricorsiva è:
  - Un callee: deve salvare i registri callee-saved ($s0, …, $ra, $fp)
  - Un caller: deve salvare i registri caller-saved ($t0, …, $a0, …, $v0, $v1)

- Al momento del ritorno dalla chiamata ricorsiva è necessario ripristinare il valore di `$ra`

Procedure ricorsive

Possono essere strutturate in diversi blocchi funzionali:

- Punto di ingresso
- Push sullo stack dei registri usati
- Check caso base / step ricorsivo
  - Caso base
  - Step ricorsivo
- Ripristino dei registri usati
- `jr $ra`

Esempio: somma elementi array

- S prende in input un array e il numero di elementi di quell’array; restituisce la somma di tutti gli elementi dell’array

```
int S( int arr[], int dim ) {

   if ( dim == 0 ) // caso base: array vuoto
      return 0 ;

   else // step ricorsivo
      return S( arr, dim - 1 ) + arr[ dim - 1 ] ;

}
```

Esempio: somma elementi array

```
S:
addi $sp, $sp, -8      # Salvo sullo stack $ra e dim-1
addi $t0, $a1, -1
sw   $t0, 0($sp)
sw   $ra, 4($sp)
```

Esempio: somma elementi array

```
S:
addi $sp, $sp, -8       # Salvo sullo stack $ra e dim-1
addi $t0, $a1, -1
sw   $t0, 0($sp)
sw   $ra, 4($sp)

bne  $a1, $zero, STEP_RICORSIVO        # branch ! ( size == 0 )
li   $v0, 0                            # Caso base: se array vuoto ritorna 0
addi $sp, $sp, 8                       # dealloco stack frame
jr $ra

STEP_RICORSIVO:
```

Esempio: somma elementi array

```
S:
addi $sp, $sp, -8       # Salvo sullo stack $ra e dim-1
addi $t0, $a1, -1
sw   $t0, 0($sp)
sw   $ra, 4($sp)

bne  $a1, $zero, STEP_RICORSIVO      # branch ! ( size == 0 )
li   $v0, 0                          # Caso base: se array vuoto ritorna 0
addi $sp, $sp, 8                     # dealloco stack frame
jr $ra

STEP_RICORSIVO:
move  $a1, $t0            # aggiorno secondo argomento  (il primo è il base addr. dell’array)
jal   S                   # chiamata ricorsiva

# (ora in $v0 ho S(arr, dim-1))
```

Esempio: somma elementi array

```
S:
addi $sp, $sp, -8      # Salvo sullo stack $ra e dim-1
addi $t0, $a1, -1
sw   $t0, 0($sp)
sw   $ra, 4($sp)

bne  $a1, $zero, STEP_RICORSIVO      # branch ! ( size == 0 )
li   $v0, 0                          # Caso base: se array vuoto ritorna 0
addi $sp, $sp, 8                     # dealloco stack frame
jr $ra

STEP_RICORSIVO:
move  $a1, $t0         # aggiorno secondo argomento (il primo è il base addr. dell’array)
jal   S                # chiamata ricorsiva

# (ora in $v0 ho S(arr, dim-1))

lw    $t0, 0($sp)      # ripristino dim-1
mul  $t1, $t0, 4       # lo moltiplico per 4 e lo metto in $t1
add   $t1, $t1, $a0    # indirizzo di arr[dim-1]
lw    $t2, 0($t1)      # t2 = arr[dim-1]
add   $v0, $v0, $t2    # $v0 = + S(arr, dim-1) + arr[dim-1]
```

Esempio: somma elementi array

```
S:
addi $sp, $sp, -8        # Salvo sullo stack $ra e dim-1
addi $t0, $a1, -1
sw   $t0, 0($sp)
sw   $ra, 4($sp)

bne  $a1, $zero, STEP_RICORSIVO     # branch ! ( size == 0 )
li   $v0, 0                         # Caso base: se array vuoto ritorna 0
addi $sp, $sp, 8                    # dealloco stack frame
jr $ra

STEP_RICORSIVO:
move  $a1, $t0          # aggiorno secondo argomento (il primo è il base addr. dell'array)
jal   S                 # chiamata ricorsiva

# (ora in $v0 ho S(arr, dim-1))

lw    $t0, 0($sp)       # ripristino dim-1
mul   $t1, $t0, 4       # lo moltiplico per 4 e lo metto in $t1
add   $t1, $t1, $a0     # indirizzo di arr[dim-1]
lw    $t2, 0($t1)       # t2 = arr[dim-1]
add   $v0, $v0, $t2     # $v0 = + S(arr, dim-1) + arr[dim-1]

lw    $ra, 4($sp)       # ripristino $ra
addi  $sp, $sp, 8       # dealloco stack frame
jr $ra
```

Esercizio 8.1

- Nome del file sorgente *fattorialeFibonacci.asm*

- Si scriva un programma che, dato un intero positivo n, stampi a video
  – il fattoriale di n
  – l'n-esimo numero di Fibonacci Φ_n dove

$$\Phi_n = \begin{cases} \Phi_{n-2} + \Phi_{n-1} & \text{if } n > 2 \\ 1 & \text{if } n = 2 \\ 0 & \text{if } n = 1. \end{cases}$$

  – Il calcolo del fattoriale e del numero di Fibonacci venga operato con l'uso di procedure ricorsive

Esercizio 8.2

- Nome del file sorgente *stampaContrario.asm*

- Si scriva un programma che dato un array di interi stampi l'array al contrario (dall'ultimo numero nell'array al primo). Il programma faccia uso di una procedura ricorsiva.

Esercizio 8.3

- Nome del file sorgente *binarySearch.asm*
- Si scriva un programma che esegua una binary search su un array **ordinato** di interi
- Dato un array A di N interi **ordinati** e un intero k, la procedura b(A,k,N) restituisce 1 se k è contenuto in A. Restituisce 0 altrimenti.

- Definizione operativa ricorsiva ?

$|A|$: numero di elementi in un array A
e_i: elemento di A in posizione i
A_L^i: array ottenuto selezionando da A gli elementi in posizione $j < i$
A_R^i: array ottenuto selezionando da A gli elementi in posizione $j > i$

$$b(A,k,N) = \begin{cases} b(A_L^{\lceil N/2 \rceil}, k, \lceil \frac{N}{2} \rceil - 1) & \text{if } k < e_{\lceil N/2 \rceil} \\ b(A_R^{\lceil N/2 \rceil}, k, N - \frac{N}{2}) & \text{if } k > e_{\lceil N/2 \rceil} \\ 1 & \text{if } k = e_{\lceil N/2 \rceil} \\ 0 & \text{if } k \neq e_{\lceil N/2 \rceil} \text{ e N=1} \end{cases}$$

Università degli Studi di Milano
Laboratorio di Architettura degli Elaboratori II
Corso di Laurea in Informatica, A.A. 2017-2018

Nicola Basilico
Dipartimento di Informatica
Via Comelico 39/41 - 20135 Milano (MI)
Ufficio S242
nicola.basilico@unimi.it
+39 02.503.16294

Hanno contribuito alla realizzazione di queste slides:
- Iuri Frosio
- Jacopo Essenziale