UNITELMA SAPIENZA

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Architettura degli Elaboratori
Lez. 6 – ASM: Funzioni ricorsive

Prof. Andrea Sterbini – sterbini@di.uniroma1.it

Argomenti

Argomenti della lezione

-Definizione di funzioni ricorsive

-Realizzazione in assembler

-Trasformazione di una funzione ricorsiva in iterativa

-Esempi di programmi

Funzione / procedura ricorsiva:

funzione che richiama, anche indirettamente, se stessa

Soluzioni ricorsive di problemi

Quando si può usare una funzione ricorsiva per risolvere un problema?

- se **esiste una soluzione conosciuta** per lo stesso problema di «piccole» dimensioni
- da questa ricaviamo il **caso base** della funzione

- e se esiste un modo di ottenere la soluzione di un problema di dimensione maggiore **a partire dalla soluzione dello stesso problema di dimensione minore**
- da questa definizione ricaviamo il **caso ricorsivo**, che è formato da 3 parti:
- **riduzione** del problema in problemi «più piccoli»
- **chiamata ricorsiva** della funzione per risolvere i casi «più piccoli»
- elaborazione delle soluzioni «piccole» per ottenere la **soluzione del problema originale**

Esempio

Funzione fattoriale

Definizione iterativa:

«il fattoriale di un numero intero N è il prodotto dei numeri 1..N»

Ovvero:

```
risultato = 1
for (i=N ; i>0 ; i--)
risultato *= i
```

Definizione ricorsiva:

caso base: «il fattoriale di 1 è 1»

riduzione del problema:

«per moltiplicare i numeri da 1 a N posso prima moltiplicare da 1 a N-1

costruzione della soluzione finale a partire da quella ridotta:

… e poi moltiplicare per N»

Ovvero

fattoriale(N) = 1 se N < 2 (caso base)

fattoriale(N) = N * fattoriale(N-1) altrimenti (caso ricorsivo)

Implementazione iterativa


```
   # moltiplico i numeri da N a 1
   # a0 = N
fattoriale:
   li  $v0, 1              # risultato = 1
ciclo:
   blez    $a0, fine          # se N <= 0 si è finito
   mul $v0, $v0, $a0       # risultato *= N
   sub $a0, $a0, 1         # N = N-1
   j   ciclo
fine:
   jr  $ra                 # torno  l'esecuzione al chiamante
```

Implementazione ricorsiva


```
   # a0 = N
fattoriale_ricorsivo:
   blez    $a0, caso_base          # se N <= 0 si è nel caso base

   ... qui preserviamo (su stack) $ra e $a0

   sub $a0, $a0, 1                 # N = N-1
   jal fattoriale_ricorsivo        # calcolo di fattoriale(N-1)

   ... qui ripristiniamo $a0 e $ra (dallo stack)

   mul $v0, $v0, $a0               # risultato *= N
   jr  $ra                         # torno  l'esecuzione al chiamante
caso_base:                         # caso base
   li  $v0, 1                      # risultato = 1
   jr  $ra                         # torno  l'esecuzione al chiamante
```

Impl. ricorsiva (completa)


```
   # a0 = N
fattoriale_ricorsivo:
   blez   $a0, caso_base          # se N <= 0 si è nel caso base
   subi   $sp, $sp, 8             # alloco 2 word su stack
   sw  $ra, 0($sp)              # salvo $ra
   sw  $a0, 4($sp)              # salvo $a0
   sub $a0, $a0, 1              # N = N-1
   jal fattoriale_ricorsivo     # calcolo di fattoriale(N-1)
   lw  $ra, 0($sp)              # ripristino $ra
   lw  $a0, 4($sp)              # ripristino $a0 (ovvero N)
   addi   $sp, $sp, 8             # disalloco 2 word dallo stack
   mul $v0, $v0, $a0            # risultato *= N
   jr  $ra                      # torno  l'esecuzione al chiamante
caso_base:                      # caso base
   li  $v0, 1                   # risultato = 1
   jr  $ra                      # torno  l'esecuzione al chiamante
```

Ricorsione multipla

È possibile che una funzione chiami se stessa più volte

Esempio: i numeri di Fibonacci

> «Su una isola c'è una coppia di giovani conigli,
> i coniglietti il primo anno sono troppo giovani e non fanno figli
> una coppia di conigli ogni anno a partire dal secondo fa un'altra coppia di conigli
> dopo N anni quante coppie sono presenti sull'isola?»

Cosa succede:

Fibonacci(0) = 1 la prima coppia è stata messa sull'isola

Fibonacci(1) = 1 al primo anno la prima coppia è ancora giovane

Fibonacci(2) = 1 + 1 al secondo anno la coppia iniziale genera una seconda coppia

Fibonacci(3) = 2 + 1 la seconda coppia non è matura, la prima coppia genera ancora

Fibonacci(4) = 3 + 2 tutte le coppie che hanno almeno 2 anni generano altre coppie

Fibonacci(5) = 5 + 3 ovvero oltre alle Fibonacci(N-1) coppie dell'anno precedente,
si aggiungono Fibonacci(N-2) nuove coppie ogni anno

Definizione ricorsiva

Fibonacci(0) = 1

Fibonacci(1) = 1

Fibonacci(N) = Fibonacci(N-1) # quelli vivi l'anno precedente

\+ Fibonacci(N-2) # quelli ormai maturi (nati da almeno 2 anni)

Realizzazione in Python:

```
def fibonacci(N):
    if N < 2:
            return 1                                    # caso base
    else:
            return  fibonacci(N-1)+fibonacci(N-2)       # caso
    ricorsivo
```

Implementazione ricorsiva


```
fibonacci: # a0 = N
   ... salvataggio dei registri su stack
   li  $t0, 2
   blt $a0, $t0, caso_base    # se N < 2 si è nel caso base
   sub $a0, $a0, 1            # N-1
   jal fibonacci              # calcolo di fibonacci(N-1)
   move    $v1, $v0               # tengo da parte il risultato
   sub $a0, $a0, 1            # N-2
   jal fibonacci              # calcolo di fibonacci(N-2)
   ... ripristino dei registri dallo stack
   add $v0, $v0, $v1          # fibonacci(N-1) + fibonacci(N-2)
   jr  $ra                    # torno  l'esecuzione al chiamante
caso_base:                    # caso base
   li  $v0, 1                 # risultato = 1
   ... ripristino dei registri dallo stack
   jr  $ra                    # torno  l'esecuzione al chiamante
```

Salvataggio su stack

Cosa va salvato? (tutto assieme così gli offset restano uguali e non faccio errori)

- **$ra** perché la funzione chiama un'altra funzione (se stessa)
- **$a0** perché la funzione ha bisogno di N per calcolare N-1 e N-2
- **$v1** perché la funzione deve ricordare F(N-1) per poi sommarlo a F(N-2)
- **$t0** (opzionale) se vogliamo modificare il minor numero di registri possibili

Quindi:

```
fibonacci: # a0 = N
    subi  $sp, $sp, 16   # alloco 4 word su stack
    sw  $ra, 0($sp)# salvo $ra
    sw  $a0, 4($sp)# salvo $a0
    sw  $v1, 8($sp)# salvo $v1
    sw  $t0, 12($sp)   # salvo $t0
    li  $t0, 2
    ble $a0, $t0, caso_base    # se N < 2 si è nel caso base
    sub $a0, $a0, 1# N-1
    jal fibonacci  # calcolo di fibonacci(N-1)
```

Ripristino da stack


```
move    $v1, $v0              # tengo da parte il risultato
sub $a0, $a0, 1               # N-2
jal fibonacci                 # calcolo di fibonacci(N-2)
lw  $ra, 0($sp)               # recupero $ra
lw  $a0, 4($sp)               # recupero $a0
lw  $v1, 8($sp)               # recupero $v1
lw  $t0, 12($sp)              # recupero $t0
subi    $sp, $sp, 16          # disalloco 4 word da stack
add $v0, $v0, $v1             # fibonacci(N-1) + fibonacci(N-2)
jr  $ra                       # torno  l'esecuzione al chiamante
```

Caso base


```
caso_base:                  # caso base
   li  $v0, 1               # risultato = 1
   lw  $ra, 0($sp)          # recupero $ra
   lw  $a0, 4($sp)          # recupero $a0
   lw  $v1, 8($sp)          # recupero $v1
   lw  $t0, 12($sp)         # recupero $t0
   subi    $sp, $sp, 16        # disalloco 4 word da stack
   jr  $ra                  # torno  l'esecuzione al chiamante
```

NOTA: per ogni possibile percorso di esecuzione nella funzione (caso base o caso ricorsivo) dev'essere eseguita una coppia di salvataggio-ripristino identica altrimenti la gestione della stack diviene irregolare

Ricorsione -> iterazione

È sempre possibile trasformare un problema ricorsivo in uno iterativo (e viceversa)

Caso semplice, se la funzione ricorsiva ha una sola chiamata ricorsiva:

funzione(argomenti)
    if caso_base
    istruzioni del **caso base**
    return risultato
    else
    istr. prima della ricorsione
    calcolo di argomenti1
    funzione(argomenti1)
    istr. dopo la ricorsione return risultato

L'esecuzione si svolgerà più o meno così:

funzione(argomenti0)
    istr. prima della ricorsione
    calcolo di argomenti1
    istr. prima della ricorsione
    calcolo di argomenti2
    istr. prima della ricorsione
    calcolo di argomenti3
    …
    istruzioni del caso base
    …
    istr. dopo la ricorsione
    istr. dopo la ricorsione
    istr. dopo la ricorsione

… versione iterativa

Che è possibile realizzare con due cicli che eseguono nell'ordine:

- la ripetizione delle **istruzioni precedenti** alla chiamata ricorsiva, (compresa di **aggiornamento degli argomenti** della chiamata)
- il **caso base**
- e poi la ripetizione (per lo stesso numero di volte) delle **istruzioni seguenti** la chiamata ricorsiva (se necessario riottenendo il valore che avevano gli **argomenti**)

```
Funzione( argomenti)
    N=0
    while (not caso base)
        N += 1
        istr. prima della ricorsione
        aggiornamento degli argomenti
    istruzioni del caso base
    for (i=0 ; i<N ; i++)
        ricostruzione degli argomenti
        istr. dopo la ricorsione
```

NOTA se la ricorsione è multipla un contatore non è sufficiente a ricostruire la struttura delle chiamate e può essere necessario usare una stack per simulare la gestione corretta delle chiamate

Esempio


```
def fattoriale( N ):
    if N < 2:
    return 1      # caso base
    else:
    N1 = N – 1   # aggiorno args
    F1 = fattoriale( N1 )      # ricorsione



    F2 = F1 * N # istr. dopo ric.
    return F2
```

NOTA: possiamo semplificare la versione iterativa usando la commutatività dell'operazione prodotto per svolgere i prodotti da N a 1 invece che da 1 a N ed eliminando il conteggio non più necessario

```
def fattoriale(N):
    i=0 # contatore
    while (N >1):
    i = i + 1# conto  chiamate
    N = N – 1     # aggiorno args
    F1 = 1   # caso base
    for j in range(i):
    N = N + 1    # ricostruisco args
    F1 = F1 * N # istr. dopo ric.
    return F1
def fattoriale(N):
    F1 = 1   # caso base
    while (N >1):
    N = N – 1     # aggiorno args
    F1 = F1 * N # istr. dopo ric.
    return F1
```

Compito per casa

Realizzare in assembler, sia in forma ricorsiva che in forma iterativa, la funzione GCD (massimo comun divisore di due interi positivi) definita come segue:

GCD(x, y) = x se x = y

GCD(x, y) = GCD(y, x) se x > y

GCD(x, y) = GCD(x, y – x) se x < y

Esempio di esecuzione

GCD(120, 105) =

GCD(105, 120) =

GCD(105, 15) =

GCD(15, 105) =

GCD(15, 90) =

GCD(15, 75) =

GCD(15, 60) =

GCD(15, 45) =

GCD(15, 30) =

GCD(15, 15) = 15