Fondamenti di Informatica (Elettronici)

THINK JULIA – Capitolo 5 (seconda parte)

29 ottobre 2020

5. Recursion (parte b) [1]

1. Ricorsione
2. Diagrammi di stack per funzioni ricorsive
3. Ricorsione infinita
4. Input da tastiera
5. Debug
6. Glossario
7. Exercises

[1] Tratto da https://benlauwens.github.io/ThinkJulia.jl/latest/book.html, disponibile sotto Licenza 'Creative Commons Attribution-NonCommercial 3.0 Unported'.

Section 1

Ricorsione

Lavora magicamente

È legale per una funzione chiamarne un'altra; è anche legale che una funzione chiami se stessa.

Potrebbe non essere ovvio il motivo per cui questa è una cosa utile, ma risulta essere una delle cose più magiche che un programma possa fare.

Ad esempio, guarda la seguente funzione:

```
function countdown(n)
    if n <= 0
        println(`Blastoff!`)
    else
        print(n, ` `)
        countdown(n-1)
    end
end
```

Se "n" è zero oppure negativo, restituisce la parola "Blastoff!" (Lanciato!). Altrimenti, restituisce "n" e quindi chiama una funzione denominata countdown- essa stessa - passando "n-1" come argomento.

countdown ricorsivo

Cosa succede se chiamiamo questa funzione in questo modo?

```
julia> countdown(10)
10 9 8 7 6 5 4 3 2 1 Blastoff!
```

Esecuzione passo dopo passo

L'esecuzione di `countdown(3)` inizia con `n = 3`, e poiché `n` è maggiore di 0, restituisce il valore 3, quindi chiama se stessa . . .

L'esecuzione di `countdown` inizia con `n = 2`, e poiché `n` è maggiore di 0, restituisce il valore 2 e quindi chiama se stessa. . .

L'esecuzione di `countdown` inizia conn `= 1`, e poiché `n` è maggiore di 0, restituisce il valore 1 e quindi chiama se stessa. . .

L'esecuzione di `countdown` inizia conn `= 0`, e poiché `n` non è maggiore di 0, restituisce la parola, `Blastoff!` e poi ritorna.

Il `countdown` che ha ottenuto `n = 1` ritorna.

Il `countdown` che ha ottenuto `n = 2` ritorna.

Il `countdown` che ha ottenuto `n = 3` ritorna. E poi sei di nuovo in `Main`.

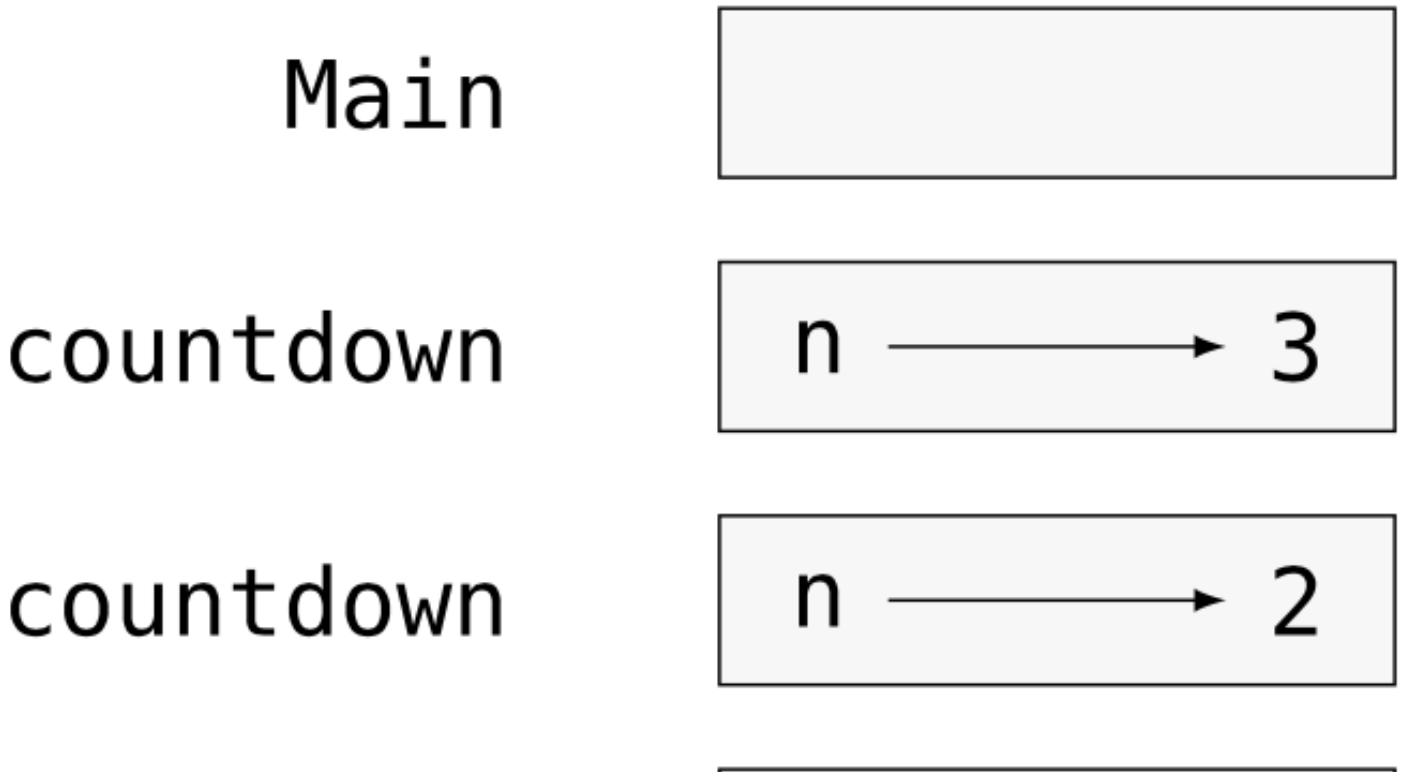
Section 2

Diagrammi di stack per funzioni ricorsive

Esempio di esecuzione

In Stack Diagrams, abbiamo utilizzato un* diagramma di stack per rappresentare lo stato di un programma durante una chiamata* di funzione. Lo stesso tipo di diagramma può aiutare a interpretare una funzione ricorsiva.

Ogni volta che una funzione viene chiamata, Julia crea un frame per contenere i parametri e le variabili locali della funzione. Per una funzione ricorsiva, potrebbe esserci più di un frame nello stack allo stesso tempo.



Section 3

Ricorsione infinita

Il programma non termina mai

Se una ricorsione non raggiunge mai un caso base, continua a effettuare chiamate ricorsive per sempre e il programma non termina mai.

Questo è noto come ricorsione infinita e generalmente non è una buona mossa.

Ecco un programma minimo con una ricorsione infinita:

```
function recurse()
    recurse()
end
```

controlla se puoi raggiungere un caso base.

Julia error message

Nella maggior parte degli ambienti di programmazione, un programma con ricorsione infinita non viene realmente eseguito per sempre.

Julia segnala un messaggio di errore quando viene raggiunta la profondità massima di ricorsione:

```
julia> recurse()
ERROR: StackOverflowError:
Stacktrace:
     [1] recurse() at ./REPL[1]:2 (repeats 79984 times)
```

Questo stacktrace è un po' più grande di quello che abbiamo visto nel capitolo precedente. Quando si verifica l'errore, ci sono 80000 frame di ricorsione nello stack!

Se si verifica accidentalmente una ricorsione infinita, rivedere la funzione per confermare che esista un caso base che non effettui una chiamata ricorsiva. E se c'è un caso base, che poi termini.

Section 4

Input da tastiera

Interazione utente

I programmi che abbiamo scritto finora non accettano alcun input da parte dell'utente. Fanno semplicemente la stessa cosa ogni volta.

Julia fornisce una funzione predefinita chiamata `readline` che arresta il programma e attende che l'utente digiti qualcosa. Quando l'utente preme RETURN o ENTER, il programma riprende, e `readline` restituisce ciò che l'utente ha digitato come stringa.

```
julia> text = readline()
What are you waiting for?
`What are you waiting for?`
```

Prima di ricevere input dall'utente, è una buona idea stampare un `prompt` che dica all'utente cosa digitare:

```
julia> print(`What...is your name? `); readline()
What...is your name? Arthur, King of the Britons!
`Arthur, King of the Britons!`
```

Un punto e virgola `;` permette di mettere più istruzioni sulla stessa riga. Nella `REPL` solo l'ultima istruzione restituisce il suo valore.

Immettere solo stringhe di cifre

Se ti aspetti che l'utente digiti un numero intero, puoi provare a convertire il valore restituito in Int64:

```
julia> println(`What...is the airspeed velocity of an unladen swallow?`);
speed = readline()
What...is the airspeed velocity of an unladen swallow?
42
`42`
julia> parse(Int64, speed)
42
```

Ma se l'utente digita qualcosa di diverso da una stringa di cifre, viene visualizzato un errore:

```
julia> println(`What...is the airspeed velocity of an unladen swallow? `); speed = 
What...is the airspeed velocity of an unladen swallow?
What do you mean, an African or a European swallow?
`What do you mean, an African or a European swallow?`
julia> parse(Int64, speed)
ERROR: ArgumentError: invalid base 10 digit 'W' in `What do you mean, an African or
[...]
```

Vedremo più avanti come gestire questo tipo di errore.

Section 5

Debug

I messaggi di errore a volte danno un'indicazione sbagliata

Quando si verifica un errore di sintassi o di runtime, il messaggio di errore contiene molte informazioni, ma può essere fuorviante

Le parti più utili sono solitamente: di che tipo di errore si trattava, e dove si è verificato.

Gli errori di sintassi sono generalmente facili da trovare, ma servono alcuni trucchi. In generale, i messaggi di errore indicano dove è stato rilevato il problema, ma l'errore effettivo potrebbe essere precedente nel codice, a volte su una riga precedente.

Lo stesso vale per gli errori di runtime. Supponi di voler calcolare un rapporto segnale / rumore in decibel. La formula è

$$SNR_{db} = 10 \log_{10} \frac{P_{signal}}{P_{noice}}$$

Esempio

In Julia, potresti scrivere qualcosa del genere:

```
signal_power = 9
noise_power = 10
ratio = signal_power ÷ noise_power
decibels = 10 * log10(ratio)
print(decibels)
```

E ottieni:

```
-Inf
```

Questo non è il risultato che ti aspettavi.

Per trovare l'errore, potrebbe essere utile stampare il valore del rapporto, che risulta essere 0. Il problema è nella riga 3, che utilizza la **divisione intera* invece della divisione in virgola mobile.

Dovresti dedicare del tempo alla lettura attenta dei messaggi di errore, ma non dare per scontato che tutto ciò che dicono sia corretto.

Section 6

Glossario

Glossario I

divisione intera Operatore, indicato con ÷, che divide due numeri e arrotonda per difetto (verso l'infinito negativo) a un numero intero.

Glossario I

divisione intera Operatore, indicato con ÷, che divide due numeri e arrotonda per difetto (verso l'infinito negativo) a un numero intero.

operatore modulo Operatore, indicato con un segno di percentuale (%), che funziona su numeri interi e restituisce il resto quando un numero viene diviso per un altro.

Glossario I

divisione intera Operatore, indicato con ÷, che divide due numeri e arrotonda per difetto (verso l'infinito negativo) a un numero intero.

operatore modulo Operatore, indicato con un segno di percentuale (%), che funziona su numeri interi e restituisce il resto quando un numero viene diviso per un altro.

espressione booleana Un'espressione il cui valore è `true` o `false`.

Glossario I

divisione intera Operatore, indicato con ÷, che divide due numeri e arrotonda per difetto (verso l'infinito negativo) a un numero intero.

operatore modulo Operatore, indicato con un segno di percentuale (%), che funziona su numeri interi e restituisce il resto quando un numero viene diviso per un altro.

espressione booleana Un'espressione il cui valore è `true` o `false`.

operatore relazionale Uno degli operatori che confronta i suoi operandi: ==, ! =, >, <, > = E <=.

Glossario I

divisione intera Operatore, indicato con ÷, che divide due numeri e arrotonda per difetto (verso l'infinito negativo) a un numero intero.

operatore modulo Operatore, indicato con un segno di percentuale (%), che funziona su numeri interi e restituisce il resto quando un numero viene diviso per un altro.

espressione booleana Un'espressione il cui valore è `true` o `false`.

operatore relazionale Uno degli operatori che confronta i suoi operandi: ==, ! =, >, <, > = E <=.

operatore logico Uno degli operatori che combina espressioni booleane: && (and), || (or) e ! (Not).

Glossario I

divisione intera Operatore, indicato con ÷, che divide due numeri e arrotonda per difetto (verso l'infinito negativo) a un numero intero.

operatore modulo Operatore, indicato con un segno di percentuale (%), che funziona su numeri interi e restituisce il resto quando un numero viene diviso per un altro.

espressione booleana Un'espressione il cui valore è `true` o `false`.

operatore relazionale Uno degli operatori che confronta i suoi operandi: ==, ! =, >, <, > = E <=.

operatore logico Uno degli operatori che combina espressioni booleane: && (and), || (or) e ! (Not).

dichiarazione condizionale Un'istruzione che controlla il flusso di esecuzione a seconda di una condizione.

Glossario I

divisione intera Operatore, indicato con ÷, che divide due numeri e arrotonda per difetto (verso l'infinito negativo) a un numero intero.

operatore modulo Operatore, indicato con un segno di percentuale (%), che funziona su numeri interi e restituisce il resto quando un numero viene diviso per un altro.

espressione booleana Un'espressione il cui valore è `true` o `false`.

operatore relazionale Uno degli operatori che confronta i suoi operandi: ==, ! =, >, <, > = E <=.

operatore logico Uno degli operatori che combina espressioni booleane: && (and), || (or) e ! (Not).

dichiarazione condizionale Un'istruzione che controlla il flusso di esecuzione a seconda di una condizione.

condizione L'espressione booleana in un'istruzione condizionale che determina quale ramo viene eseguito.

Glossario II

dichiarazione composta Un'istruzione composta da un'intestazione e un corpo. Il corpo termina con la parola chiave `end`.

Glossario II

dichiarazione composta Un'istruzione composta da un'intestazione e un corpo. Il corpo termina con la parola chiave `end`.

ramo Una delle sequenze alternative di istruzioni in un'istruzione condizionale.

Glossario II

dichiarazione composta Un'istruzione composta da un'intestazione e un corpo. Il corpo termina con la parola chiave `end`.

ramo Una delle sequenze alternative di istruzioni in un'istruzione condizionale.

condizionale concatenato Una dichiarazione condizionale con una serie di diramazioni alternative.

Glossario II

dichiarazione composta Un'istruzione composta da un'intestazione e un corpo. Il corpo termina con la parola chiave `end`.

ramo Una delle sequenze alternative di istruzioni in un'istruzione condizionale.

condizionale concatenato Una dichiarazione condizionale con una serie di diramazioni alternative.

condizionale annidato Un'istruzione condizionale che appare in uno dei rami di un'altra istruzione condizionale.

Glossario II

dichiarazione composta Un'istruzione composta da un'intestazione e un corpo. Il corpo termina con la parola chiave `end`.

ramo Una delle sequenze alternative di istruzioni in un'istruzione condizionale.

condizionale concatenato Una dichiarazione condizionale con una serie di diramazioni alternative.

condizionale annidato Un'istruzione condizionale che appare in uno dei rami di un'altra istruzione condizionale.

dichiarazione di ritorno Un'istruzione che fa terminare immediatamente una funzione e tornare al chiamante.

Glossario II

dichiarazione composta Un'istruzione composta da un'intestazione e un corpo. Il corpo termina con la parola chiave `end`.

ramo Una delle sequenze alternative di istruzioni in un'istruzione condizionale.

condizionale concatenato Una dichiarazione condizionale con una serie di diramazioni alternative.

condizionale annidato Un'istruzione condizionale che appare in uno dei rami di un'altra istruzione condizionale.

dichiarazione di ritorno Un'istruzione che fa terminare immediatamente una funzione e tornare al chiamante.

ricorsione Il processo di chiamata della funzione attualmente in esecuzione.

Glossario II

dichiarazione composta Un'istruzione composta da un'intestazione e un corpo. Il corpo termina con la parola chiave `end`.

ramo Una delle sequenze alternative di istruzioni in un'istruzione condizionale.

condizionale concatenato Una dichiarazione condizionale con una serie di diramazioni alternative.

condizionale annidato Un'istruzione condizionale che appare in uno dei rami di un'altra istruzione condizionale.

dichiarazione di ritorno Un'istruzione che fa terminare immediatamente una funzione e tornare al chiamante.

ricorsione Il processo di chiamata della funzione attualmente in esecuzione.

caso base Un ramo condizionale in una funzione ricorsiva che non effettua una chiamata ricorsiva.

Glossario II

dichiarazione composta Un'istruzione composta da un'intestazione e un corpo. Il corpo termina con la parola chiave `end`.

ramo Una delle sequenze alternative di istruzioni in un'istruzione condizionale.

condizionale concatenato Una dichiarazione condizionale con una serie di diramazioni alternative.

condizionale annidato Un'istruzione condizionale che appare in uno dei rami di un'altra istruzione condizionale.

dichiarazione di ritorno Un'istruzione che fa terminare immediatamente una funzione e tornare al chiamante.

ricorsione Il processo di chiamata della funzione attualmente in esecuzione.

caso base Un ramo condizionale in una funzione ricorsiva che non effettua una chiamata ricorsiva.

ricorsione infinita Una ricorsione che non ha un caso di base o non lo raggiunge mai. Alla fine, una ricorsione infinita causa un `errore di runtime`.

Section 7

Exercises

aaaa

aaaa