[21/22] 38090-MOD2 - PROGRAMMAZIONE AVANZATA

Maffeis Isaac 1041473

Relazione di progetto: Haskell

1. Introduzione

Per il progetto nel linguaggio di programmazione Haskell mi sono concentrato nel realizzare una semplice applicazione con i costrutti studiati a lezione.
In particolare mi sono focalizzato sul concetto della ricorsione, funzioni, funzioni curried, funzioni anonime, funzioni di ordine superiore, liste, tuple, tipi, equazioni di guardia, pattern matching.

L'applicazione realizzata si occupa di verificare se una stringa è ricorsiva e di restituire una lista contenente le ricorrenze dei caratteri da cui è composta.

È presente una parte di test interattiva da console nella quale si può testare il corretto funzionamento.

2. Caratteristiche

2.1 Funzionalità

L'applicazione realizzata riceve in ingresso dall'utente una stringa, verifica se è palindroma e restituisce una lista con le ricorrenze dei caratteri letterali o numerici da cui è composta la stringa.

es. amorale, d'ogni vin gode l'aroma.
è palindroma.
è composta da:
[[('a',4),('d',2),('e',2),('g',2),('i',2),('l',2),('m',2),('n',2),('o',4),('r',2),('v',1)]]

Il programma deve quindi permettere all'utente di scrivere una parola o frase, verificare se questa sia palindroma non considerando spazi e caratteri speciali come simboli e accenti (,.?!-:;\"\') .
Sempre non considerando questi caratteri deve successivamente passare a contare le ricorrenze dei caratteri presenti nella parola o frase e mostrarli all'utente in ordine alfabetico.

3. Composizione

3.1 Interprete

L'applicazione Haskell è stata realizzata tramite l'interprete e compilatore online replit.com

3.2 Funzioni

In un linguaggio funzionale le funzioni sono il cuore del programma, essendo esso non una lista di operazioni come in un linguaggio empirico, ma una composizione di funzioni.

Per realizzare il progetto sono servite le seguenti funzioni:

- filter :: (a -> Bool) -> [a] -> [a]
  filtra una lista con i soli elementi che soddisfano il predicato richiesto

- map :: (a->b)->[a]->[b]
  applica la funzione f a tutti gli elementi della lista.
- not :: Bool -> Bool
  svolge la funzione dell'operatore not logico.
- elem :: Eq a => a -> [a] -> Bool
  restituisce true se la funzione contiene l'elemento fornito.
- head :: [a] -> a
  restituisce il primo elemento di una lista.
- last :: [a] -> a
  restituisce l'ultimo elemento di una lista.
- init :: [a] -> [a]
  accetta una lista e ritona la lista senza il primo elemento.
- tail :: [a] -> [a]
  accetta una lista e ritona la lista senza l'ultimo elemento.
- length :: [a] -> Int
  restituisce la lunghezza di una stringa
- group :: Eq a => [a] -> [[a]]
  crea liste di caratteri uguali adiacenti da una lista di caratteri (stringa).
- removeWhiteSpaces :: String -> String
  rimuove gli spazi presenti in una frase.
- removeSymbols :: String -> String
  rimuove i simboli come , . ? ! - : ; \ " \ ' che possono essere presenti all'interno di una frase o parola.
- isPalindrome :: String -> Bool
  restituisce true o false in base a se la stringa fornita è palindroma o no.
- isPalindromeToString :: Bool -> String
  trasforma in una stringa il risultato true/false della valutazione della funzione precedente.
- quicksort :: (Ord a) => [a] -> [a]
  ordina una stringa in ordine alfabetico, o in generale ordina una lista di elementi ordinabili.
- func :: String -> [(Char,Int)]
  funzione di ordine superiore, è composta da più funzioni, permette disporre la stringa in ordine alfabetico, dividerla in liste di caratteri uguali adiacenti, creare le tuple composte dal primo carattere della lista e dalla lunghezza di quella lista

3.3 Ricorsione

Il linguaggio funzionale Haskell è basato sulla ricorsione, principio per cui grazie a delle ipotesi ricorsive si riduce il problema generale a un problema più semplice (caso base), si calcola la soluzione del caso base e passo per passo lo si compone con il problema generale per risolverlo.

In questa applicazione vengono definite tre funzioni ricorsive:

- isPalindrome :: String -> Bool
  caso base: stringa vuota oppure contenente un solo elemento;
  ipotesi ricorsiva: la lista senza il primo e l'ultimo elemento;
  passo: si controlla l'uguaglianza tra il primo e l'ultimo elemento della lista.

- group :: Eq a => [a] -> [[a]]
  caso base: stringa vuota;
  ipotesi ricorsiva: ci si basa su una funzione ausiliaria per memorizzare il risultato e si suppone si sapere il risultato del carattere successivo;
  passo: si controlla l'uguaglianza tra i caratteri adiacenti.
- quicksort :: (Ord a) => [a] -> [a]
  caso base: stringa vuota;
  ipotesi ricorsiva: si suppone si sapere il risultato del carattere successivo;
  passo: valutazione > o = di un carattere con il successivo.

3.3 Anonymous function

Una funzione anonima è una funzione senza nome, è una astrazione lambda λ, viene usata perché a volte è più conveniente definire una funzione in questo modo direttamente nel corpo di un'altra funzione, piuttosto che definirla con in nome.
Nel progetto sono presenti varie funzioni anonime e vengono definite per mezzo dell'operatore lambda (oppure con il simbolo '\') .

es. (\x -> x /= ' ') xs : applica a xs quella funzione tale che dato un x restituisce se è diverso dal carattere spazio

3.5 Curried Function

Il currying è una tecnica che trasforma una funzione di più argomenti in una funzione di 1 argomento che restituisce una funzione di 1 argomento che restituisce una funzione di 1 argomento... finché non restituisce un valore.

3.6 Funzioni di ordine superiore

Le funzioni di ordine superiore possono prendere altre funzioni come parametro e addirittura ritornare delle funzioni, sul principio delle curried functions.

Nel progetto il concetto di funzione curried e di ordine superiore è stato usato per sviluppare la funzione:

- func :: String -> [(Char,Int)]
  func xs = map (\a -> (head a, length a)) $ group $ quicksort xs
  funzione che prende come parametro la stringa xs, esegue il quicksort su di essa per ordinarla in ordine alfabetico, dal risultato del quicksort esegue la funzione group per creare liste di caratteri adiacenti, su questo risultato applica la funzione anonima che crea una tupla composta dalla testa della lista e dalla sua lunghezza, a tutti gli elementi della lista.

Il carattere $ serve per considerare ciò che viene dopo quel carattere come singolo argomento della funzione presente prima del carattere

3.7 Input / Output

La fase di Input e Output è svolta nella funzione principale main.

- L'input è ottenuto grazie al costrutto do, (main = do) e per via del comando <- getLine si può memorizzare in una variabile una stringa scritta dall'utente.

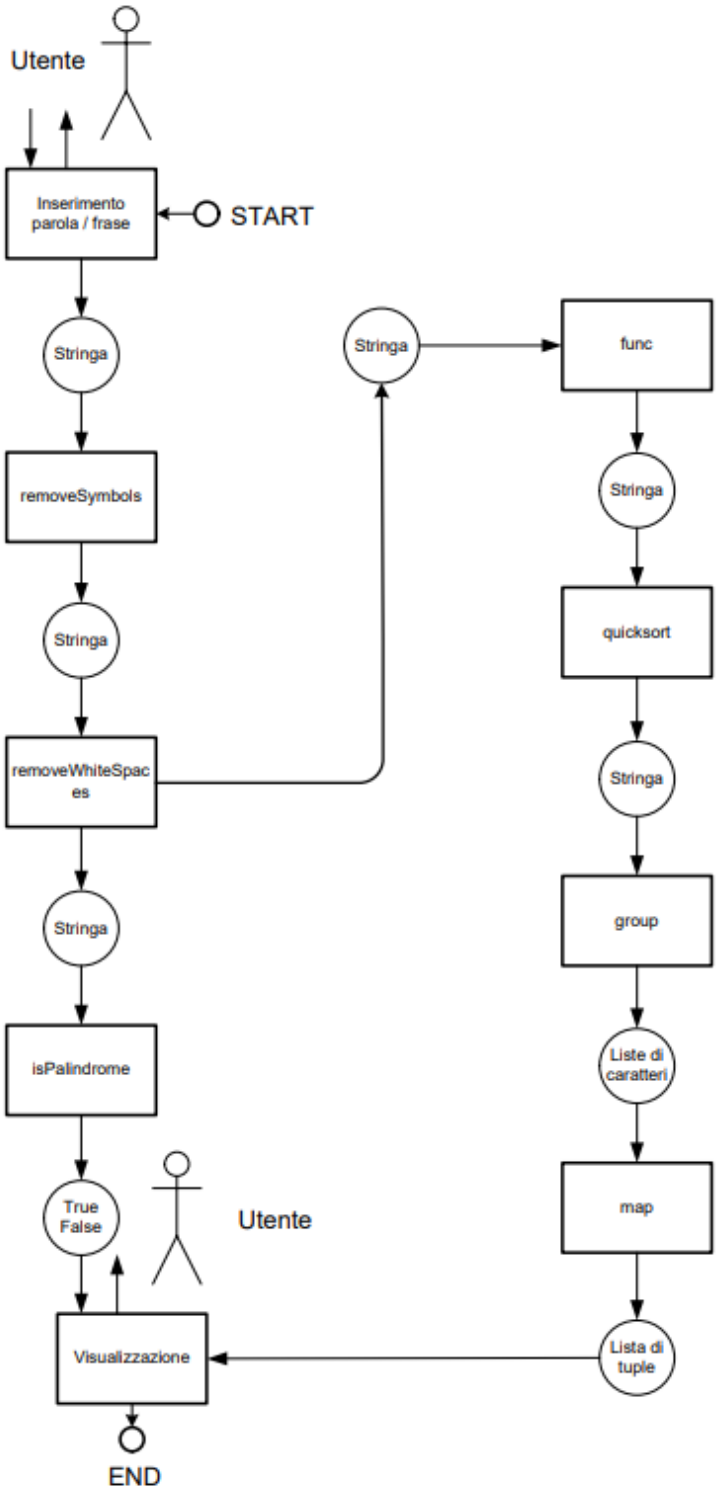
- L’Output è ottenuto semplicemente con il comando putStrLn che visualizza su console la stringa fornita e con la funzione show che permette di convertire interi in stringhe per poterle stampare.

4. Funzionamento

Il funzionamento è descritto tramite la rete di Petri in figura nella pagina successiva, realizzata con un basso livello di astrazione e assumendo che tutte le funzioni siano realizzate in linguaggio Haskell, tutte sullo stesso server.

5. Testing

La fase di test è realizzata nella funzione principale main, vengono svolti da prima test funzionali, considerando degli esempi di stringhe palindrome. È previsto anche un test interattivo in cui si simula la procedura del cliente tramite input/output da console inserendo parole o frasi a piacimento per verificarne il giusto comportamento.


Utente
Inserimento parola / frase
START
Stringa
removeSymbols
Stringa
removeWhiteSpaces
Stringa
isPalindrome
True False
Utente
Visualizzazione
END
Stringa
func
Stringa
quicksort
Stringa
group
Liste di caratteri
map
Lista di tuple