Università degli Studi di Bergamo

SCUOLA DI INGEGNERIA
Corso di Laurea Magistrale in Ingegneria Informatica

Progetto Haskell: Matching Brackets

Studente
Davide Salvetti
Matricola 1057596

Anno Accademico 2021–2022

1 Introduzione

Il programma sviluppato consente di analizzare i file con estensione ".cpp" e ".hpp" e di verificare che tutte le parentesi contenute nei file siano chiuse e accoppiate in modo corretto.

2 Funzionamento

Una volta lanciato il programma, vengono selezionati tutti i file all'interno della *directory* in cui il programma viene eseguito. Successivamente, i file vengono filtrati e vengono mantenuti solo quelli che hanno estensione ".cpp" e ."hpp". A questo punto, ognuno di questi file viene letto e ne viene estratto il contenuto. Vengono poi create delle coppie costituite da nome del file e contenuto del file, e vengono mostrati sullo schermo i nomi dei file che hanno parentesi correttamente accoppiate e quelli che invece non hanno passato tale test.

3 Funzioni implementate in Haskell

Di seguito vengono riportate le funzioni che sono state implementate.

3.1 main

Il main è la funzione principale da cui parte l'esecuzione. Come anticipato, viene selezionata la *directory* corrente in cui viene eseguito il programma. Vengono poi estratti tutti i file e vengono filtrati. A questo punto, grazie alla funzione `mapM` viene eseguita la funzione `loadFileStrict` su tutti i file contenuti in `all`. A questo punto viene eseguita prima la funzione `getCorrectFiles` e poi `getIncorrectFiles` per verificare quali tra questi file ha le parentesi inserite in modo corretto. L'argomento in ingresso a queste funzioni è il risultato della funzione `zip`, che viene utilizzata per creare una lista di tuple composte da due stringhe, in cui nella prima stringa c'è il nome del file e nella seconda il suo contenuto.

```
main = do
  currentDir <- getCurrentDirectory
  putStrLn "--- Analyzing files in:"
  print currentDir
  all <- getDirectoryContents ""
  let filesFound = onlycppandhpp all
  putStrLn "--- Files Found:"
  print filesFound
  contents <- mapM loadFileStrict $ filesFound
  putStrLn "--- Files with correct matching brackets:"
  print(getCorrectFiles $ zip filesFound contents)
  putStrLn "--- Files with incorrect matching brackets:"
  print(getIncorrectFiles $ zip filesFound contents)
```

3.2 onlycppandhpp

Questa funzione serve per eseguire il filtro dei file in ingresso. Riceve come parametri una lista di stringhe e ritorna a sua volta una lista di stringhe. Per ogni stringa ricevuta in ingresso viene eseguito un controllo per verificare che contenga ".cpp" oppure ".hpp" e se contiene uno dei due viene inserita in una lista. Per far ciò è stata utilizzata una *list comprehension* e la funzione `stringContains` spiegata in seguito.
Se necessario, si potrebbe modificare tale funzione in modo da accettare in input anche

una lista di stringhe che contengono le estensioni da filtrare e richiedere all'utente quali file vuole che siano analizzati. Così facendo si potrebbero analizzare anche file con estensioni diverse da ".cpp" e ".hpp".

```
{- Filters only cpp and hpp files. -}
onlycppandhpp :: [String] -> [String]
onlycppandhpp xs = [x | x <- xs, or [(stringContains ".cpp" x), (
    stringContains ".hpp" x)]]
```

3.3 loadFileStrict

Questa funzione serve per estrarre da un file il suo contenuto. La particolarità di questa funzione è che evita la *Lazy Evaluation* di Haskell, andando a leggere la lunghezza del file prima di leggere il contenuto. In questo modo si è certi che tutto il contenuto del file venga letto.

```
{- Function to read a file without lazy evaluation. -}
loadFileStrict :: FilePath -> IO String
loadFileStrict f = do
    s <- readFile f
    length s `seq` return s
```

3.4 stringContains

La funzione `stringContains` riceve in ingresso prima la stringa che deve essere contenuta e poi la stringa che la deve contenere, e restituisce un booleano: `True` se il primo parametro è contenuto nel secondo, `False` altrimenti.
Per fare ciò sono stati utilizzati il *pattern matching* e la ricorsione. Inoltre, è stata utilizzata una funzione di supporto `check`: questa funzione verifica, sempre tramite ricorsione e *pattern matching*, se la tutti i caratteri della stringa passata come primo argomento combaciano con quelli iniziali della stringa passata come secondo argomento. Questa funzione di supporto viene chiamata dalla funzione `stringContains` con secondo parametro diminuito del suo primo elemento ogni volta. In questo modo viene scandita tutta la stringa.

```
stringContains :: String -> String -> Bool
stringContains (_:_) [] = False
stringContains xs ys
    | check xs ys = True
    | stringContains xs (tail ys) = True
    | otherwise = False

check :: String -> String -> Bool
check [] _ = True
check (_:_) [] = False
check (x:xs) (y:ys) = (x == y) && check xs ys
```

3.5 getCorrectFile e getIncorrectFile

Queste funzioni, tramite *list comprehension*, ritornano una lista di stringhe che soddisfano la condizione `arePaired` (o `not(arePaired)` per i file non corretti).

```
{- Returns only the files with correct brackets. -}
getCorrectFiles :: [(String, String)] -> [String]
getCorrectFiles tuples = [a | (a, b) <- tuples, arePaired b]

{- Returns only the files with incorrect brackets. -}
getIncorrectFiles :: [(String, String)] -> [String]
getIncorrectFiles tuples = [a | (a, b) <- tuples, not( arePaired b)]
```

3.6 arePaired

Questa è la funzione principale del programma. Riceve in ingresso una stringa (il contenuto del file) e ritorna un valore booleano. Per fare ciò si appoggia alla funzione `removeMatching` ed alla funzione di libreria `foldl`. Innanzitutto, dalla stringa in ingresso vengono filtrate solo le parentesi. Poi viene applicata la funzione `removeMatching` che riceve in ingresso una stringa ed un carattere, e ritorna una stringa. Questa funzione riceve in ingresso la stringa contenente tutte le parentesi presenti nel file, ed un carattere (che è necessariamente un tipo di parentesi perchè sono stati filtrati solo quegli elementi). A seconda della parentesi che viene passata, la funzione ritorna la stringa in ingresso meno la coppia di parentesi definite dal carattere che gli è stato passato. Se invece non trova il *pattern matching* corretto, ritorna la stringa con le parentesi senza eliminarle. Tutto questo viene eseguito tramite *pattern matching*.
Grazie alla funzione `foldl` è possibile applicare la funzione a tutta la stringa partendo da destra verso sinistra: per questo motivo i caratteri che può ricevere in ingresso la funzione `removeMatching` sono parentesi chiuse.
Se il risultato finale della funzione *removeMatching* è una stringa vuota, allora viene ritornato il valore booleano `True` ed il contenuto del file passato in origine contiene parentesi accoppiate correttamente, altrimenti viene ritornato il valore booleano `False`.

```
arePaired :: String -> Bool
arePaired xs = null $ foldl removeMatching [] $ filter (`elem` "({[]})") xs

removeMatching :: String -> Char -> String
removeMatching ('(' : xs) ')' = xs
removeMatching ('{' : xs) '}' = xs
removeMatching ('[' : xs) ']' = xs
removeMatching (xs) x         = x:xs
```

3.7 Esempio di esecuzione

Per poter eseguire il programma è stato installato il compilatore *ghc*. Di seguito viene mostrato un esempio di esecuzione del programma.

```
PS C:\Users\Davide\Desktop\MatchingBrackets\MatchingBrackets> ./MatchingBrackets
--- Analyzing files in:
"C:\\Users\\Davide\\Desktop\\MatchingBrackets\\MatchingBrackets"
--- Files Found:
["swordsman.cpp","map.hpp","map.cpp","knight.cpp","game.cpp","characterfactory.cpp"]
--- Files with correct matching brackets:
["map.cpp","knight.cpp","game.cpp","characterfactory.cpp"]
--- Files with incorrect matching brackets:
["swordsman.cpp","map.hpp"]
```