UNIVERSITÀ DEGLI STUDI DI URBINO
DIPARTIMENTO DI SCIENZE PURE E APPLICATE
CORSO DI LAUREA IN INFORMATICA APPLICATA

Progetto di Programmazione Logica e Funzionale

di Giaconi Christian, Giacomo Rossi
Matricola 314045, 314671
Anno di corso: terzo

Anno Accademico 2023/2024 - Sessione invernale

Docente: Prof. Marco Bernardo

Indice

1 Specifica del problema

Scrivere un programma in Haskell e un programma in Prolog per implementare un sistema di raccomandazione di canzoni. Il sistema suggerisce canzoni a un utente basandosi sulle sue preferenze musicali e utilizza un punteggio di gradimento per ordinare le canzoni più popolari o rilevanti.

2 Analisi del problema

2.1 Dati in ingresso

- Un file contenente le canzoni nel formato:

```
Titolo,Artista,Genere,Punteggio
```

- Una lista di generi preferiti.
- Pesi numerici assegnati ai generi preferiti.

2.2 Dati in uscita

- Una classifica ordinata di canzoni basata sui punteggi ponderati.
- Eventuali messaggi di errore o conferma nelle interazioni utente.

2.3 Relazioni tra i dati

Ogni canzone è caratterizzata da un titolo, un artista, un genere, e un punteggio. Il punteggio ponderato è calcolato come:

$$Punteggio\ Ponderato = Punteggio \times Peso\_Genere$$

se il genere della canzone appartiene ai generi preferiti, altrimenti è uguale al punteggio originale.

3 Progettazione dell'algoritmo

3.1 Scelte di progetto

Per sviluppare il sistema di raccomandazione musicale, abbiamo adottato approcci distinti per ciascuno dei linguaggi richiesti, Haskell e Prolog, al fine di sfruttare al meglio le peculiarità di ciascuno.

In **Haskell**, abbiamo deciso di rappresentare le canzoni utilizzando tipi di dati strutturati. Questa scelta ci consente di garantire una gestione chiara e sicura delle informazioni musicali, sfruttando le capacità del sistema di tipi statico per ridurre errori e facilitare l'implementazione delle operazioni. Le canzoni vengono caricate da un file di testo, per consentire l'integrazione di dataset esterni senza modificare il codice sorgente. La scelta di utilizzare un file di testo è stata motivata dalla necessità di mantenere il sistema flessibile e facilmente adattabile a dataset di dimensioni variabili.

In **Prolog**, invece, abbiamo optato per una rappresentazione tramite predicati dinamici. Le canzoni sono caricate all'interno del programma attraverso il predicato `carica_canzoni/0`, che utilizza `assertz/1` per definire dinamicamente ogni canzone come un fatto nella base di conoscenza. Questo approccio ci permette di manipolare i dati con la semantica dichiarativa propria di Prolog, evitando problematiche legate all'importazione di file esterni. Ogni canzone è rappresentata da un predicato `canzone/4`, che descrive il titolo, l'artista, il genere e il punteggio. La struttura dati utilizzata è completamente dinamica, consentendo aggiunte o modifiche in tempo reale senza la necessità di ridefinire l'intero dataset.

- **Haskell:**
  - L'uso di file di testo consente di estendere il dataset semplicemente aggiungendo nuove righe al file.
  - Le funzioni per la manipolazione delle stringhe e la costruzione di tipi di dati strutturati in Haskell rendono questa operazione relativamente semplice e modulare.
  - Inoltre, Haskell permette un trattamento funzionale dei dati, come la mappatura e il filtraggio, che è particolarmente utile per calcolare e ordinare i punteggi ponderati.
- **Prolog:**
  - La rappresentazione tramite predicati è particolarmente adatta per le relazioni tra canzoni, generi e preferenze dell'utente.
  - Caricare le canzoni come fatti dinamici permette di utilizzare direttamente la logica del linguaggio per effettuare operazioni come il filtraggio e l'ordinamento basato sui punteggi.
  - Questo approccio evita la necessità di definire una sintassi di input complessa, sfruttando la semplicità dichiarativa propria di Prolog.
- **Gestione dell'input e output:**
  - In Haskell, il file di testo richiede un parsing iniziale per costruire una lista di canzoni.
  - In Prolog, invece, la definizione diretta nel predicato `carica_canzoni/0` semplifica l'esecuzione e permette di focalizzarsi sull'elaborazione delle raccomandazioni piuttosto che sulla gestione dell'input.

3.2 Passi dell'algoritmo

I passi principali dell'algoritmo, comuni sia per l'implementazione in Haskell che in Prolog, sono i seguenti:

1. **Caricare le canzoni:**
    - In Haskell, le canzoni vengono caricate da un file di testo, che permette la costruzione di una lista di canzoni.
    - In Prolog, le canzoni sono definite all'interno del programma attraverso il predicato `carica_canzoni/0`.
2. **Inserire le preferenze dell'utente:** L'utente fornisce i generi musicali preferiti e i relativi pesi, che influenzeranno il calcolo dei punteggi ponderati.
3. **Calcolare i punteggi ponderati:** Per ogni canzone, viene calcolato un punteggio combinando il peso associato al genere con il punteggio individuale della canzone.
4. **Ordinare le canzoni per punteggio ponderato:** Le canzoni vengono ordinate in ordine decrescente rispetto al punteggio ponderato, così da ottenere una classifica delle raccomandazioni.
5. **Stampare la classifica:** L'algoritmo produce in output la lista ordinata delle canzoni con i rispettivi punteggi, offrendo una chiara visualizzazione per l'utente.

La ricorsività è utilizzata in entrambi i linguaggi per garantire la gestione dinamica dei dati:

- **Haskell:** Le funzioni ricorsive vengono impiegate per elaborare le liste di canzoni. Ad esempio, il calcolo dei punteggi ponderati viene implementato tramite una funzione che scorre ricorsivamente l'intera lista, applicando il peso corrispondente al genere di ciascuna canzone.
- **Prolog:** L'ordinamento delle canzoni si basa su un predicato ricorsivo che inserisce ogni elemento in una lista ordinata. La dinamica di Prolog consente inoltre di aggiungere o aggiornare i generi preferiti senza ridefinire l'intera base di conoscenza.

4 Implementazione dell'algoritmo

4.1 Implementazione in Haskell

Il file `raccomandazioni.hs` implementa l'algoritmo in Haskell. Un esempio di calcolo dei punteggi ponderati:

```haskell
-- ########################################################
-- # Corso di Programmazione Logica e Funzionale          #
-- # Progetto di raccomandazione di canzoni               #
-- # Studente: Giaconi Christian, Giacomo Rossi           #
-- # Matricola: 314045, 314671                            #
-- ########################################################

{- Specifica:
    Scrivere un programma in Haskell per implementare un sistema
       avanzato di raccomandazione di canzoni.
    Il sistema suggerisce canzoni a un utente in base a:
    - Preferenze per uno o pi  generi musicali specificati.
    - Un sistema di punteggio ponderato per dare priorit  a
       canzoni  pi  rilevanti.
    L'utente deve fornire un file di testo con le canzoni nel
       seguente formato:
        Titolo,Artista,Genere,Punteggio
    Dove "Punteggio"    un intero da 1 a 10.
    Le canzoni saranno ordinate in base al punteggio ponderato e
       filtrate per genere.
-}

module Main where

import Data.List (sortOn, nub, intercalate)
import Data.Maybe (mapMaybe)
import Data.Ord (Down(..))
import qualified Data.Map as Map
import System.IO.Error(isDoesNotExistError)
import Control.Exception (catch, IOException)
import Text.Read (readMaybe)

-- ########################################################
-- Definizioni dei tipi di dati
-- ########################################################

-- | La struttura 'Canzone' rappresenta una canzone con:
-- - titolo: il titolo della canzone.
-- - artista: l'artista che la interpreta.
-- - genere: il genere musicale della canzone.
-- - punteggio: un punteggio assegnato (da 1 a 10).
data Canzone = Canzone
    { titolo    :: String
    , artista   :: String
    , genere    :: String
    , punteggio :: Int
    } deriving (Show, Eq)
```

```haskell
-- | PesiGeneri    una mappa che associa un genere musicale a un
   peso
-- che influenza la priorit  delle raccomandazioni.
type PesiGeneri = Map.Map String Double

-- ##########################################################
-- Main: Menu interattivo
-- ##########################################################

-- | Funzione principale che avvia il menu interattivo.
main :: IO ()
main = menuLoop Nothing Map.empty

-- | Gestisce il menu principale, mantenendo lo stato del sistema:
-- - maybeCanzoni: un elenco opzionale delle canzoni caricate.
-- - pesi: i pesi dei generi preferiti, gestiti dall'utente.
menuLoop :: Maybe [Canzone] -> PesiGeneri -> IO ()
menuLoop maybeCanzoni pesi = do
    putStrLn "\n--- Sistema di Raccomandazione Musicale ---"
    putStrLn "1. Carica un file con le canzoni"
    putStrLn "2. Gestisci i generi preferiti (aggiungi o modifica)"
    putStrLn "3. Stampa la classifica delle canzoni"
    putStrLn "4. Stampa i generi preferiti con il relativo
       punteggio"
    putStrLn "5. Esci"
    putStrLn "Seleziona un'opzione:"
    scelta <- getLine
    case scelta of
        "1" -> caricaCanzoni >>= (`menuLoop` pesi) . Just
        "2" -> selezionaGeneriPreferitiEImpostaPesi maybeCanzoni
           pesi >>= menuLoop maybeCanzoni
        "3" -> raccomandaCanzoni maybeCanzoni pesi >> menuLoop
           maybeCanzoni pesi
        "4" -> visualizzaGeneriPreferiti pesi >> menuLoop
           maybeCanzoni pesi
        "5" -> putStrLn "Grazie per aver usato il sistema di
           raccomandazione. Arrivederci!"
        _   -> putStrLn "Opzione non valida. Riprova." >> menuLoop
           maybeCanzoni pesi

-- ##########################################################
-- Funzioni di caricamento e gestione dei dati
-- ##########################################################

-- | Carica un file di testo, legge i dati delle canzoni e li
-- trasforma in una lista di Canzone.
-- Il file deve avere un formato valido: Titolo,Artista,Genere,
   Punteggio.
caricaCanzoni :: IO [Canzone]
caricaCanzoni = do
    nomeFile <- chiediNomeFile
    contenuto <- readFile nomeFile
    let canzoni = mapMaybe analizzaCanzone (lines contenuto)
    if null canzoni
        then putStrLn "Errore: il file non contiene dati validi!
           Riprova." >> caricaCanzoni
```

```
        else putStrLn "File caricato con successo!" >> return
            canzoni

-- | Richiede all'utente di inserire il nome del file con le
   canzoni
-- e ne effettua una validazione dell'input tramite la funzione
   validaFile.
chiediNomeFile :: IO FilePath
chiediNomeFile = do
    putStrLn "Inserire il nome del file:"
    nomeFile <- getLine
    esito_lettura <- validaFile nomeFile
    case esito_lettura of
        Right () -> return nomeFile  -- Restituisce il nome del
            file se valido
        Left err -> do
            putStrLn $ "Errore: " ++ err
            chiediNomeFile

-- | Controlla se il nome del file    espresso
-- correttamente e se tale file esiste.
validaFile :: FilePath -> IO (Either String ())
validaFile nomeFile =
    catch (readFile nomeFile >> return (Right ()))
          (\e -> if isDoesNotExistError e
                 then return $ Left "File non trovato!"
                 else return $ Left "Errore durante l'apertura del
                     file.")

-- | Permette all'utente di scegliere
-- i generi preferiti e assegnare un peso a ciascuno di essi.
selezionaGeneriPreferitiEImpostaPesi :: Maybe [Canzone] ->
   PesiGeneri -> IO PesiGeneri
selezionaGeneriPreferitiEImpostaPesi Nothing pesi = do
    putStrLn "Errore: nessun file caricato. Carica un file prima di
        continuare."
    return pesi
selezionaGeneriPreferitiEImpostaPesi (Just canzoni) pesi = do
    let generiDisponibili = nub $ map genere canzoni
    putStrLn $ "Generi disponibili: " ++ intercalate ", "
        generiDisponibili
    generiSelezionati <- raccogliGeneri generiDisponibili
    aggiornaPesi generiSelezionati pesi

-- | Consente all'utente di inserire i generi
-- preferiti uno alla volta, terminando con "fine".
raccogliGeneri :: [String] -> IO [String]
raccogliGeneri generiDisponibili = do
    putStrLn "Inserisci i generi preferiti uno alla volta. Scrivi '
        fine' per terminare."
    loop []
  where
    loop acc = do
        putStrLn "Inserisci un genere preferito:"
        input <- getLine
        if input == "fine"
```

```
            then return (nub acc)
            else if input `elem` generiDisponibili
                 then putStrLn ("Genere '" ++ input ++ "' aggiunto
                    ai preferiti.") >> loop (input : acc)
                 else putStrLn "Genere non valido. Riprova." >>
                    loop acc

-- | Consente all'utente di modificare i pesi dei generi preferiti.
-- Se il genere ha gi  un peso, l'utente pu  scegliere di
   mantenerlo o aggiornarlo.
aggiornaPesi :: [String] -> PesiGeneri -> IO PesiGeneri
aggiornaPesi [] pesi = return pesi
aggiornaPesi (g:gs) pesi = do
    let pesoCorrente = Map.findWithDefault 1.0 g pesi
    putStrLn $ "Peso corrente per il genere '" ++ g ++ "': " ++
       show pesoCorrente
    putStrLn "Vuoi aggiornare il peso? (s/n)"
    risposta <- getLine
    if risposta == "s"
        then do
            putStrLn $ "Inserisci il nuovo peso per il genere '" ++
                g ++ "':"
            nuovoPeso <- leggiPesoValido
            aggiornaPesi gs (Map.insert g nuovoPeso pesi)
        else do
            putStrLn $ "Peso per il genere '" ++ g ++ "' invariato.
               "
            aggiornaPesi gs pesi

-- ############################################################
-- Raccomandazioni
-- ############################################################

-- | Genera e stampa una lista di canzoni consigliate
-- basandosi sui pesi dei generi e sui punteggi delle canzoni.
raccomandaCanzoni :: Maybe [Canzone] -> PesiGeneri -> IO ()
raccomandaCanzoni Nothing _ = putStrLn "Errore: nessun file
   caricato. Carica un file prima di continuare."
raccomandaCanzoni (Just canzoni) pesi = do
    let raccomandate = raccomanda pesi canzoni
    if null raccomandate
        then putStrLn "Nessuna canzone trovata con i pesi attuali."
        else stampaClassifica raccomandate

-- ############################################################
-- Funzioni ausiliarie
-- ############################################################

-- | Converte una riga di testo in un oggetto Canzone.
-- Restituisce Nothing se la riga non    formattata correttamente.
analizzaCanzone :: String -> Maybe Canzone
analizzaCanzone riga =
    case separaTaglia ',' riga of
        [titolo, artista, genere, punteggioStr]
            | "" `notElem` [titolo, artista, genere, punteggioStr]
                -- Controlla che tutte le parti siano non vuote
```

```
            , Just punteggio <- readMaybe punteggioStr  -- Prova a
                leggere il punteggio
            , punteggio >= 1 && punteggio <= 10 -> Just (Canzone
                titolo artista genere punteggio)  -- Verifica che
                il punteggio sia valido
        _ -> Nothing  -- Restituisce Nothing se la riga non
            valida

-- | Divide una stringa in una lista di stringhe, usando un
   delimitatore.
separa :: Char -> String -> [String]
separa _ "" = []
separa delimiter string =
    let (primo, resto) = break (== delimiter) string
    in primo : case resto of
        [] -> []
        x -> separa delimiter (dropWhile (== delimiter) (tail x))

-- | Divide una stringa in campi separati, pulendo gli spazi.
separaTaglia :: Char -> String -> [String]
separaTaglia delimiter string = map (filter (/= ' ')) (separa
   delimiter string)

-- | Legge un valore di peso valido inserito dall'utente.
leggiPesoValido :: IO Double
leggiPesoValido = do
    input <- getLine
    case readMaybe input of
        Just peso | peso > 0 -> return peso
        _ -> putStrLn "Peso non valido. Riprova." >>
            leggiPesoValido

-- | Calcola il punteggio ponderato per ogni canzone e le ordina.
raccomanda :: PesiGeneri -> [Canzone] -> [(Double, Canzone)]
raccomanda pesi canzoni =
    let arricchite = arricchisci pesi canzoni
    in sortOn (Down . fst) arricchite

-- | Calcola il punteggio ponderato per ogni canzone.
arricchisci :: PesiGeneri -> [Canzone] -> [(Double, Canzone)]
arricchisci pesi canzoni =
    [ (fromIntegral (punteggio c) * Map.findWithDefault 1.0 (genere
        c) pesi, c) | c <- canzoni ]

-- | Stampa le canzoni ordinate con il loro punteggio ponderato.
stampaClassifica :: [(Double, Canzone)] -> IO ()
stampaClassifica raccomandate =
    mapM_ stampaConPosizione (zip [1..] raccomandate)
    where
        stampaConPosizione (pos, (punteggioPonderato, Canzone
            titolo artista genere _)) = do
            putStrLn $ "#" ++ show pos ++ " - " ++ titolo
            putStrLn $ "   Artista: " ++ artista
            putStrLn $ "   Genere: " ++ genere
            putStrLn $ "   Punteggio ponderato: " ++ show
                punteggioPonderato
```

```
            putStrLn "-----------------------------------------"

-- | Visualizza i generi preferiti e i pesi associati.
visualizzaGeneriPreferiti :: PesiGeneri -> IO ()
visualizzaGeneriPreferiti pesi
    | Map.null pesi = putStrLn "Nessun genere ancora definito."
    | otherwise = do
        putStrLn "I tuoi generi preferiti e pesi associati sono:"
        mapM_ stampaGenere (Map.toList pesi)

-- | Stampa il genere, concatenato al peso suo relativo
stampaGenere :: (String, Double) -> IO ()
stampaGenere (genere, peso) = putStrLn $ genere ++ ": " ++ show
   peso
```

4.2 Implementazione in Prolog

Il file `raccomandazioni.pl` implementa l'algoritmo in Prolog. Esempio di ordinamento delle canzoni:

```
/* ######################################################## */
/* # Corso di Programmazione Logica e Funzionale          # */
/* # Progetto di raccomandazione di canzoni               # */
/* # Studente: Giaconi Christian, Giacomo Rossi           # */
/* # Matricola: 314045, 314671                            # */
/* ######################################################## */

/* ================================================
   Predicati dinamici
   ================================================ */

/* Predicato che 'canzone/4' memorizza informazioni relative
   alle canzoni caricate. Ogni canzone    rappresentata
   dai seguenti argomenti: Titolo, Artista, Genere e Punteggio. */
:- dynamic(canzone/4).

/* Predicato che 'genere_preferito/2' associa un peso preferito
   a ciascun genere musicale. Il primo argomento    il Genere,
   il secondo    il Peso associato a quel genere. */
:- dynamic(genere_preferito/2).

/* ================================================
   Predicati principali
   ================================================ */

/* Predicato che 'main'    il punto di ingresso principale.
   Inizializza il programma e avvia il menu interattivo
   per l'utente. */
main :-
    nl,
    write('Benvenuto nel sistema di raccomandazione musicale!'),
    carica_canzoni,
    loop_menu.

/* Predicato che 'loop_menu' gestisce la selezione delle azioni
   da parte dell'utente nel menu principale. Ogni opzione del menu
   chiama un predicato specifico per eseguire l'azione corrispondente.
      */
loop_menu :-
    nl,
    write('======================================'), nl,
    write('Scegli un\'azione: '), nl,
    write('1. Gestisci i generi preferiti (aggiungi o modifica)'), nl,
    write('2. Stampa la classifica delle canzoni'), nl,
    write('3. Stampa la lista dei generi preferiti'), nl,
    write('4. Esci'), nl,
    write('======================================'), nl,
    write('Inserisci la tua scelta: '), nl,
    read(Scelta),
    (   Scelta = 1 -> gestisci_generi_preferiti
    ;   Scelta = 2 -> stampa_classifica
    ;   Scelta = 3 -> mostra_generi_preferiti
```

```
    ;   Scelta = 4 -> write('Arrivederci!\n'), halt
    ;   write('Scelta non valida. Riprova.\n')
    ),
    loop_menu.

carica_canzoni :-
    assertz(canzone('Despacito', 'Luis Fonsi', 'Reggaeton', 8)),
    assertz(canzone('All Eyez On Me', 'Tupac', 'HipHop', 8)),
    assertz(canzone('Danza Kuduro', 'Don Omar', 'Reggaeton', 9)),
    assertz(canzone('Song 2', 'Blur', 'Alternative/Indie', 6)),
    assertz(canzone('Bachata Rosa', 'Juan Luis Guerra', 'Bachata', 9)),
    assertz(canzone('Notturno op 55 no 1', 'Chopin', 'Classica', 6)),
    assertz(canzone('Free Bird', 'Lynyrd Skynyrd', 'Rock', 8)),
    assertz(canzone('Thunderstruck', 'AC/DC', 'Rock', 7)),
    assertz(canzone('Come As You Are', 'Nirvana', 'Rock', 8)),
    assertz(canzone('La Gota Fria', 'Carlos Vives', 'Vallenato', 7)),
    assertz(canzone('Stronger', 'Kanye West', 'HipHop', 9)),
    assertz(canzone('Californication', 'Red Hot Chili Peppers', '
       Alternative/Indie', 6)),
    assertz(canzone('Upper Echelon', 'Travis Scott', 'Trap', 7)),
    assertz(canzone('El Cantante', 'Hector Lavoe', 'Salsa', 9)),
    assertz(canzone('Suavemente', 'Elvis Crespo', 'Merengue', 10)),
    assertz(canzone('La Vaca', 'Los Toros Band', 'Merengue', 9)).

/* ================================================
   Predicati per la gestione dei generi preferiti
   ================================================ */

/* Predicato che permette all'utente di selezionare
   e gestire i generi musicali preferiti. */
gestisci_generi_preferiti :-
    mostra_generi_disponibili,
    write('Inserisci i tuoi generi preferiti tra apici, uno per volta.
       Scrivi "fine" per terminare.\n'),
    chiedi_generi_preferiti([]).

/* Predicato che raccoglie i generi preferiti inseriti
   dall'utente e li aggiunge alla lista di preferiti. */
chiedi_generi_preferiti(GeneriPreferiti) :-
    write('Inserisci un genere preferito: '),
    read(Genere),
    (   Genere == fine
    ->  chiedi_peso_generi(GeneriPreferiti)
    ;   findall(GenereDisponibile, canzone(_, _, GenereDisponibile, _),
        GeneriDisponibili),
        elimina_duplicati(GeneriDisponibili, GeneriUnici),
        (   membro(Genere, GeneriUnici)
        ->  append(GeneriPreferiti, [Genere], NuoviGeneri),
            chiedi_generi_preferiti(NuoviGeneri)
        ;   write('Genere non valido. Ecco i generi disponibili:\n'),
            scrivi_lista(GeneriUnici),
            chiedi_generi_preferiti(GeneriPreferiti)
        )
    ).
```

```
/* Predicato che chiede all'utente di inserire
   un peso per ciascun genere musicale preferito. */
chiedi_peso_generi([]).
chiedi_peso_generi([Genere | Altri]) :-
    format('Inserisci il peso per il genere ~w: ', [Genere]),
    read(Peso),
    (   number(Peso), Peso > 0
    ->  assertz(genere_preferito(Genere, Peso)),
        chiedi_peso_generi(Altri)
    ;   write('Peso non valido. Riprova.\n'),
        chiedi_peso_generi([Genere | Altri]) ).

/* ================================================
   Predicati per la raccomandazione e la classifica
   ================================================ */

/* Predicato che calcola il punteggio ponderato per ogni canzone
   in base al suo genere e al suo punteggio originale.
   Poi stampa la classifica ordinata delle canzoni. */
stampa_classifica :-
    nl, write('Calcolando la classifica...'), nl,
    findall(PunteggioPonderato-Titolo, calcola_punteggio_ponderato(
       Titolo, PunteggioPonderato), Punteggi),
    (   Punteggi == []
    ->  nl, write('Nessuna canzone trovata con punteggio ponderato.'),
       nl
    ;   ordina_lista(Punteggi, PunteggiOrdinati),
        nl, write('Ecco la classifica delle canzoni:'), nl,
        stampa_canzoni_ordinate(PunteggiOrdinati, 1)
    ).

/* Predicato che stampa le canzoni ordinate in base al punteggio
   ponderato, elencandole con la posizione, il titolo,
   l'artista e il punteggio ponderato. */
stampa_canzoni_ordinate([], _).
stampa_canzoni_ordinate([PunteggioPonderato-Titolo | Rest], Posizione)
   :-
    canzone(Titolo, Artista, Genere, _),
    format('~d# ~w (Artista: ~w, Genere: ~w, Punteggio ponderato: ~2f)\
       n',
           [Posizione, Titolo, Artista, Genere, PunteggioPonderato]),
    Posizione1 is Posizione + 1,
    stampa_canzoni_ordinate(Rest, Posizione1).

/* Predicato che calcola il punteggio ponderato
   di una canzone in base al suo genere (e al peso preferito associato)
   moltiplicato per il punteggio originale della canzone. */
calcola_punteggio_ponderato(Titolo, PunteggioPonderato) :-
    canzone(Titolo, _, Genere, Punteggio),
    peso_genere(Genere, Peso),
    PunteggioPonderato is Punteggio * Peso.

/* ================================================
   Predicati ausiliari
   ================================================ */
```

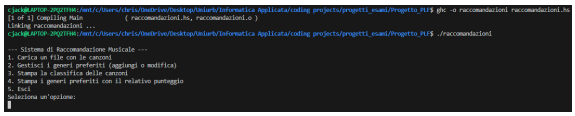
```
/* Predicato che mostra i generi preferiti associati
   con il rispettivo peso. */
mostra_generi_preferiti :-
    findall(Genere-Peso, genere_preferito(Genere, Peso), Generi),
    (   Generi == []
    ->  write('Non    stato definito alcun genere preferito.\n')
    ;   write('I tuoi generi preferiti e i loro pesi:\n'),
        stampa_generi(Generi)
    ).

/* Predicato che restituisce una lista dei generi musicali
   presenti nel database delle canzoni, evitando duplicati. */
mostra_generi_disponibili :-
    findall(Genere, canzone(_, _, Genere, _), Generi),
    elimina_duplicati(Generi, GeneriUnici),
    write('Generi disponibili:\n'),
    scrivi_lista(GeneriUnici).

/* Predicato che elimina duplicati da una lista. */
elimina_duplicati([], []).
elimina_duplicati([H|T], [H|T1]) :-
    non_membro(H, T),
    elimina_duplicati(T, T1).
elimina_duplicati([H|T], T1) :-
    membro(H, T),
    elimina_duplicati(T, T1).

/* Predicato che verifica se un elemento    membro della lista. */
membro(X, [X|_]).
membro(X, [_|T]) :-
    membro(X, T).

/* Predicato che verifica se un elemento NON    membro della lista. */
non_membro(_, []).
non_membro(X, [H|T]) :-
    X \= H,
    non_membro(X, T).

/* Scrive una lista elemento per elemento */
scrivi_lista([]).
scrivi_lista([H|T]) :-
    write('- '), write(H), nl,
    scrivi_lista(T).

/* Predicato che stampa la lista dei generi preferiti. */
stampa_generi([]).
stampa_generi([Genere-Peso | Rest]) :-
    format('~w: ~w\n', [Genere, Peso]),
    stampa_generi(Rest).

/* Predicato che 'peso_genere' restituisce il peso di un genere.
   Se non    specificato, viene utilizzato un peso di 1. */
peso_genere(Genere, Peso) :-
    (   genere_preferito(Genere, Peso)
    ->  true
    ;   Peso = 1 ).
```

```
/* Predicato che ordina una lista in ordine decrescente. */
ordina_lista(Lista, Ordinata) :-
    ordina_lista(Lista, [], Ordinata).

/* Predicato che ordina la lista ricorsivamente. */
ordina_lista([], Acc, Acc).
ordina_lista([X | Xs], Acc, Ordinata) :-
    inserisci_decrescente(X, Acc, NuovoAcc),
    ordina_lista(Xs, NuovoAcc, Ordinata).

/* Predicato che inserisce un elemento in una lista mantenendo
   l'ordine decrescente. */
inserisci_decrescente(Punteggio1-Titolo1, [], [Punteggio1-Titolo1]).
inserisci_decrescente(Punteggio1-Titolo1, [Punteggio2-Titolo2 | Rest],
    [Punteggio1-Titolo1, Punteggio2-Titolo2 | Rest]) :-
    Punteggio1 >= Punteggio2.
inserisci_decrescente(Punteggio1-Titolo1, [Punteggio2-Titolo2 | Rest],
    [Punteggio2-Titolo2 | NewRest]) :-
    Punteggio1 < Punteggio2,
    inserisci_decrescente(Punteggio1-Titolo1, Rest, NewRest).
```

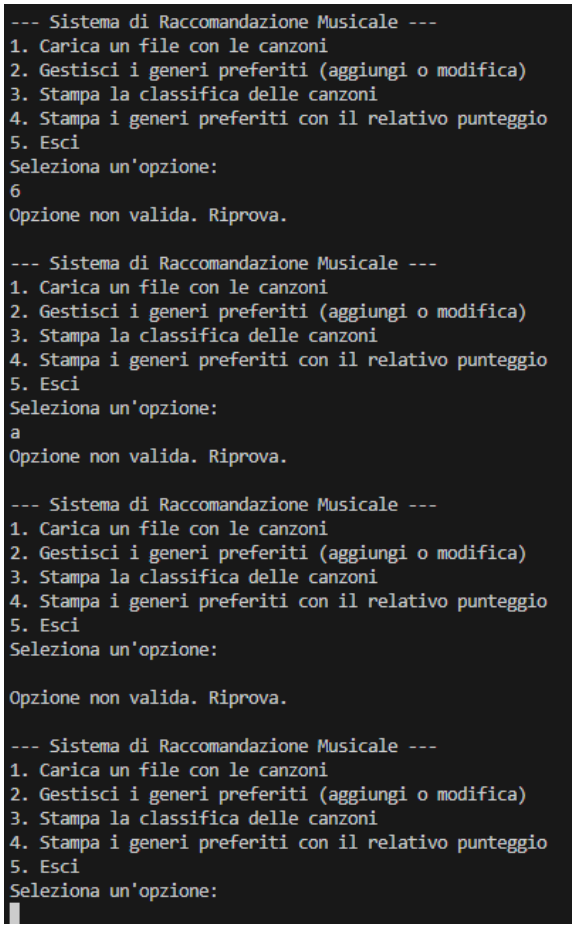
5 Testing

5.1 Testing del programma in Haskell

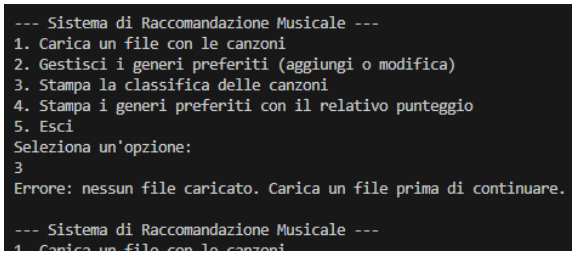
Test 1



Test 2



Test 3



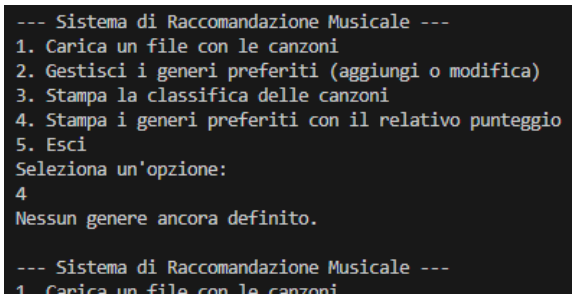
Test 4

```
--- Sistema di Raccomandazione Musicale ---
1. Carica un file con le canzoni
2. Gestisci i generi preferiti (aggiungi o modifica)
3. Stampa la classifica delle canzoni
4. Stampa i generi preferiti con il relativo punteggio
5. Esci
Seleziona un'opzione:
1
Inserire il nome del file:
pippo
Errore: File non trovato!
Inserire il nome del file:
canzoni.txt
File caricato con successo!
```

Test 5

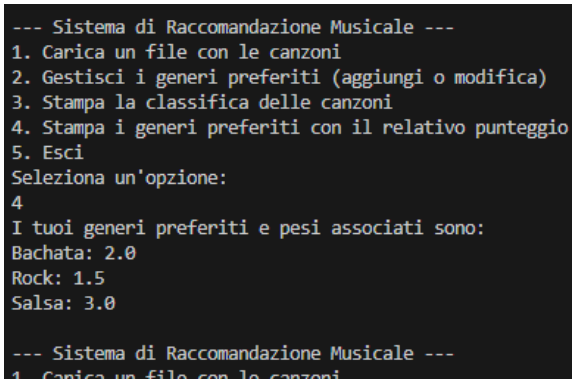
```
--- Sistema di Raccomandazione Musicale ---
1. Carica un file con le canzoni
2. Gestisci i generi preferiti (aggiungi o modifica)
3. Stampa la classifica delle canzoni
4. Stampa i generi preferiti con il relativo punteggio
5. Esci
Seleziona un'opzione:
3
#1 - Suavemente
   Artista: ElvisCrespo
   Genere: Merengue
   Punteggio ponderato: 10.0
----------------------------------------
#2 - ManyMen(WishDeath)
   Artista: 50Cent
   Genere: HipHop
   Punteggio ponderato: 10.0
----------------------------------------
#3 - DanzaKuduro
   Artista: DonOmar
   Genere: Reggaeton
   Punteggio ponderato: 9.0
----------------------------------------
#4 - BachataRosa
   Artista: JuanLuisGuerra
   Genere: Bachata
   Punteggio ponderato: 9.0
----------------------------------------
#5 - Stronger
   Artista: KanyeWest
   Genere: HipHop
   Punteggio ponderato: 9.0
----------------------------------------
#6 - ElCantante
   Artista: HectorLavoe
   Genere: Salsa
   Punteggio ponderato: 9.0
----------------------------------------
#7 - LaVaca
   Artista: LosTorosBand
```

Test 6

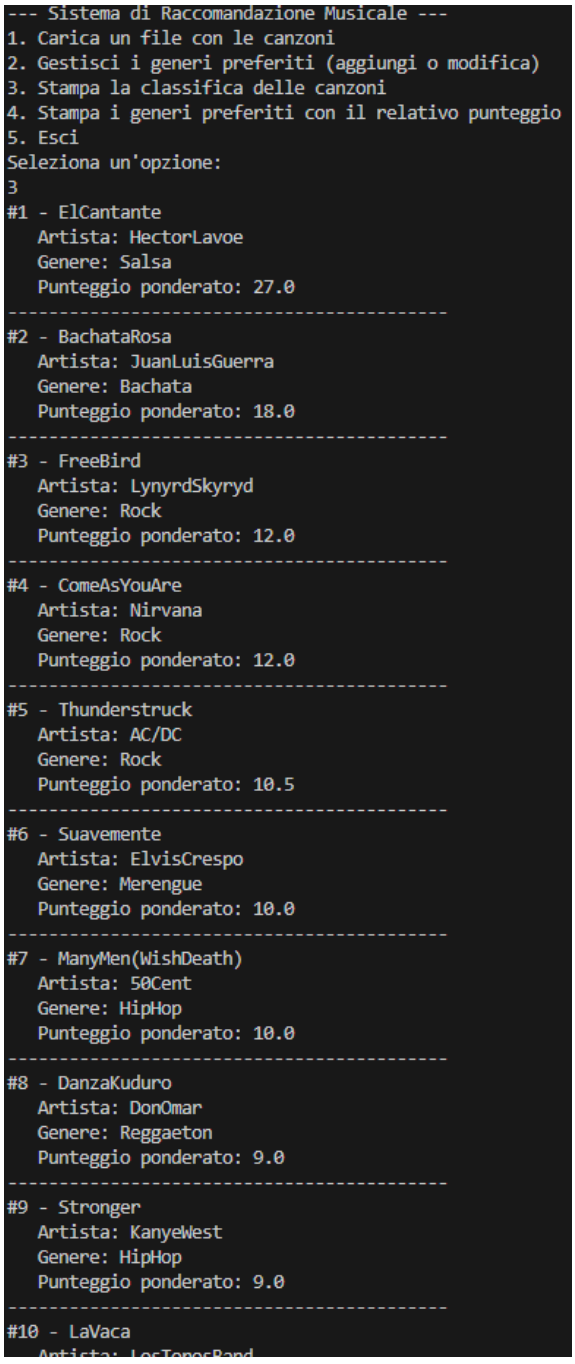


Test 7

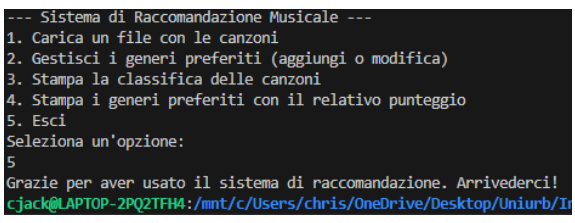
Test 8



Test 9

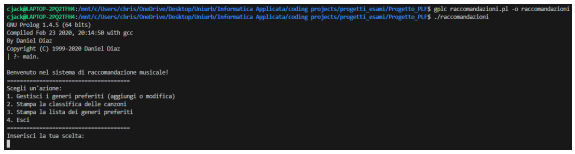


Test 10

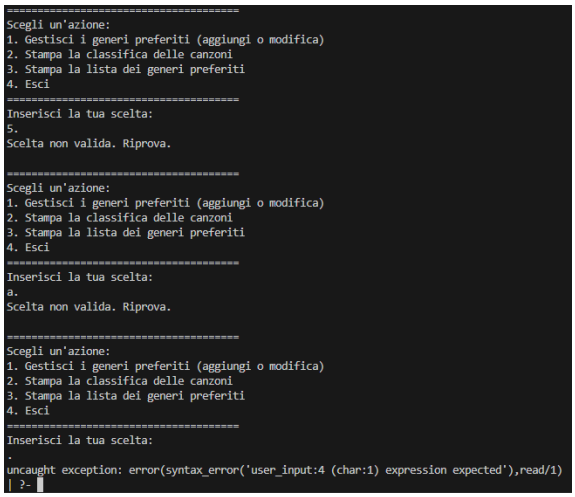


5.2 Testing del programma in Prolog

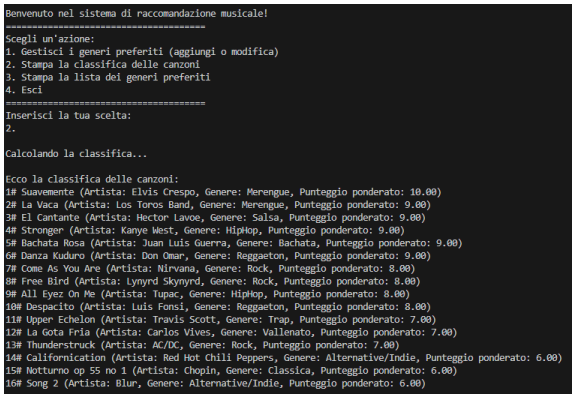
Test 1



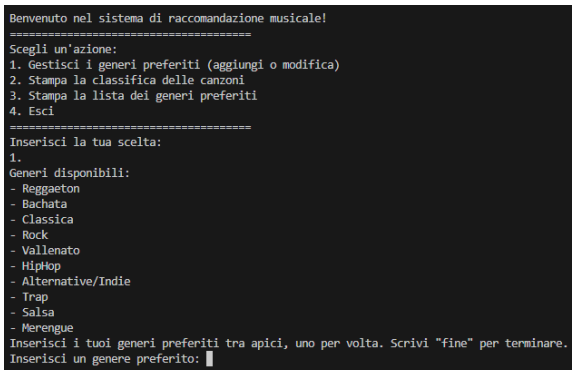
Test 2



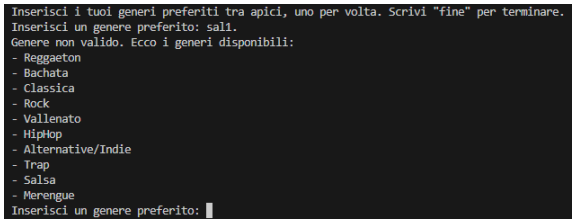
Test 3



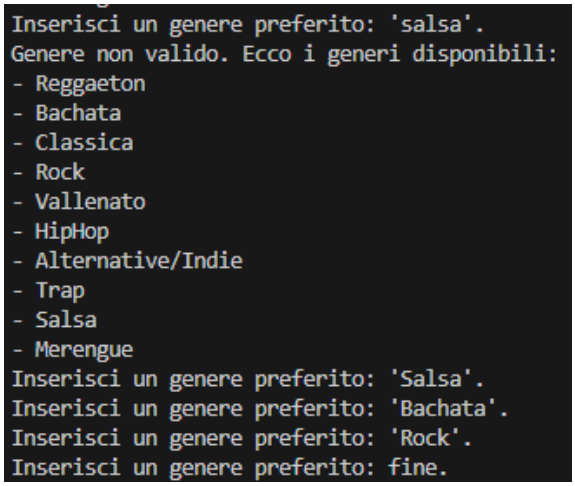
Test 4



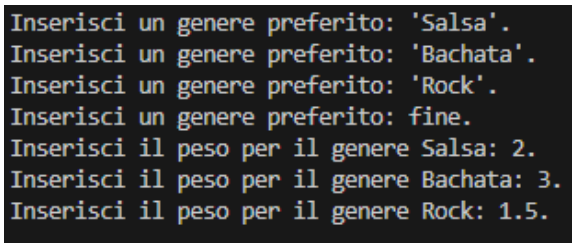
Test 5



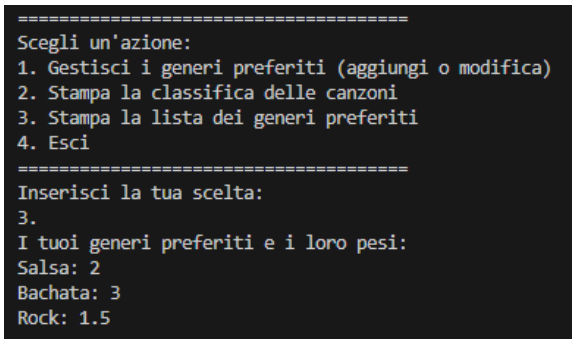
Test 6



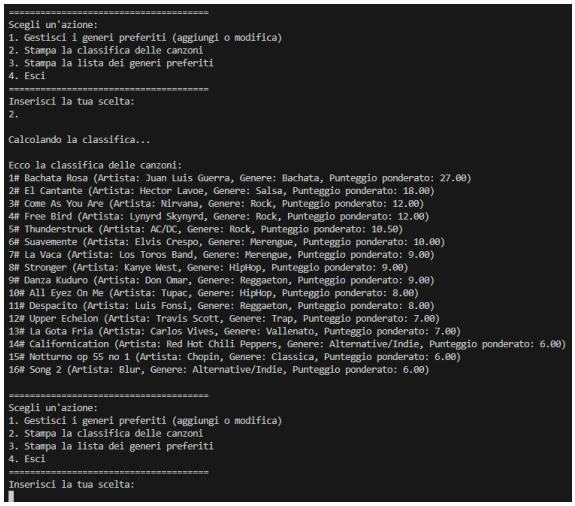
Test 7



Test 8



Test 9



Test 10