Relazione per L'esame di Programmazione Logica e Funzionale

Elia Renzoni, Gianmarco Beligni

Sessione Estiva 2024/2025

Indice

1 Specifica del Problema

Scrivere un programma ANSI C che acquisisce dalla tastiera due anni compresi tra 1900 e 2099, calcola il giorno e il mese in cui cadono il Martedì Grasso per il primo anno e il Giovedì Grasso per il secondo anno in base al calendario gregoriano (non è consentito prelevare le date da tabelle precompilate) e poi per ciascuna delle due date stampa sullo schermo le cifre del giorno e le prime tre lettere maiuscole del mese (ignorare tutte le lettere successive) con caratteri giganti ognuno formato da asterischi e occupante 5 posizioni sia in altezza che in larghezza.

2 Analisi Del Problema

2.1 Dati di Ingresso del Problema

I dati d'ingresso del problema sono costituiti da due anni compresi tra il 1900 e il 2099. Questi dovranno essere dei numeri naturali **N**.

2.2 Dati di Uscita del Problema

I dati d'uscita del problema sono caratterizzati dalle cifre del giorno e le prime tre lettere del mese in cui occorrono il Martedì Grasso, per il primo anno, e il Giovedì Grasso per il secondo anno.

2.3 Relazioni Intercorrenti

Per determinare le date del Martedì Grasso e del Giovedì Grasso bisogna partire dal giorno in cui occorre la Pasqua secondo gli anni di riferimento.
Trovare la data della Pasqua è fondamentale poichè togliendo i giorni della Quaresima è possibile risalire al Mercoledì delle Ceneri, ovvero il giorno che dà inizio al periodo di digiuno. I giorni sono 46 considerando anche le domeniche.
Il Martedì Grasso, ovvero la festa che conclude i giorni grassi di carnevale, precede sempre il Mercoledì delle Ceneri; ecco che occorre solamente aggiungere un giorno in più ai 46 per trovare la data del Martedì Grasso. Per il Giovedì Grasso, ovvero la festa che segna l'inizo dei festeggiamenti che terminano con il Martedì Grasso, il metodo è del tutto analogo a quello precedente, infatti il Giovedì Grasso precede la Pasqua di 52 giorni.
Un'eccezione che può verificarsi nel calcolo è costituita dagli anni bisestili, in quanto i giorni da scalare devono essere applicati ai giorni contenuti nei mesi a partire dal giorno di Pasqua, quindi è importante considerare le variazioni causate dagli anni bisestili.
Il calcolo quindi dipende dai seguenti aspetti:

- **Anni Bisestili:** in accordo con il Calendario Gregoriano gli anni bisestili sono quelli con un giorno in più a febbraio.
- **Pasqua**: ricorre sempre in giorni differenti in base all'anno a cui si fa riferimento, poichè è strettamente legata all'allineamento della Luna con il Sole e la Terra. Nonostante questo sono state fissate delle regole per determinare il periodo nel quale la Pasqua dovrà cadere seguendo il Calendario Gregoriano. Il principio cardine per il calcolo della Pasqua è la prima Luna piena di Primavera, in quanto la Pasqua cadrà la domenica successiva al primo plenilunio di primavera.

Per trovare il mese e il giorno in cui occorre la Pasqua abbiamo utilizzato il metodo creato dal matematico Carl Friedrich Gauss nel 1800.
L'algoritmo parte con il calcolo delle variabili `a`, `b`, `c`, `d` e `e`. Le prime tre si trovano calcolando il resto della divisione tra l'anno di riferimento, rappresentato attraverso la variabile `Y` e alcuni valori costanti, come spiegato di seguito:

$$a = Y \bmod 19$$

$$b = Y \bmod 4$$

$$c = Y \bmod 7$$

mentre le utlime due si trovano utilizzano le variabili precedenti.

$$d = (19a + M) \bmod 30$$

$$e = (2b + 4c + 6d + N) \bmod 7$$

Le costanti `M` e `N` presenti nei precedeni calcoli sono valori che cambiano in base agli anni considerati; nel caso della specifica gli anni sono compresi tra il 1900 e il 2099 quindi i valori delle due costanti sono 24, per `M`, e 5 per `N`.
Una volta calcolate le 5 variabili l'algoritmo passa alla stima effettiva del giorno e del mese:

- Se $(d + e) < 10$ allora la Pasqua cade nel mese di Marzo nel giorno $d + e + 22$.
- Se la condizione di prima non risultasse vera allora la Pasqua cadrà nel mese di aprile nel giorno $d + e - 9$.

L'algoritmo contempla anche la possibilità che si possano verificare le seguenti eccezioni:

- Se la data risultante dalla formula è il 26 aprile, allora la Pasqua cadrà il giorno 19 aprile.
- Se la data risultante dalla formula è il 25 aprile e contemporaneamente $d = 28$, $e = 6$ e $a > 10$, allora la Pasqua cadrà il 18 aprile.

3 Progettazione dell'Algoritmo

3.1 Scelte di Progetto

La stampa a caratteri giganti dovrà essere implementata, sia nel sorgente Prolog che in quello Haskell, inserendo ciascun carattere in una lista. Ogni elemento della lista dovrà contenere una frazione del carattere gigante.
Abbiamo inoltre deciso di definire nuovi tipi dati enumerativi, nel sorgente Haskell, per indicare i mesi, ovvero febbraio, marzo e aprile, e per rappresentare la scelta tra il martedì e il giovedì grasso. Infine abbiamo definito un nuovo tipo dato in grado di incapsulare le date composte dal giorno, il mese e l'anno. Nel sorgente Prolog non abbiamo definito nuovi tipi di dati, bensì abbiamo utilizzato predicati per racchiudere il giorno, il mese e l'anno di una specifica data.

3.2 Passi dell'Algoritmo

- Per ciascuna delle due date da calcolare.
  - Acquisisci e valida l'anno selezionato dall'utente.
  - Calcola il giorno e il mese in cui cade la Pasqua secondo l'anno scelto dall'utente.
  - Se l'anno scelto è il primo dei due possibili:
    - Calcola il giorno e il mese in cui cade il Martedì Grasso togliendo dal giorno della Pasqua 47 giorni e, fino a che il valore ottenuto non è positivo, aggiungendo i giorni dei mesi che mancano. Decrementare inoltre il numero del mese di partenza con il numero dei mesi che applico per il calcolo.
  - Altrimenti:
    - Calcola il giorno e il mese in cui cade il Giovedì Grasso togliendo dal giorno della Pasqua 52 e, fino a che il valore ottenuto non è posittivo, aggiungendo i giorni dei mesi che mancano. Decrementare il numero del mese con il numero dei mesi che applico per il calcolo.
  - Stampa le cifre del giorno e le prime tre lettere del mese.

4 Implementazione dell'Algoritmo

4.1 Implementazione in Haskell

Nome del sorgente: `martedi_giovedi_grasso.hs`

```
{-
 - Progetto di Programmazione Logica e Funzionale
 - Studenti: Elia Renzoni e Gianmarco Beligni
 - N. Matricola: 319978 321069
 - Sessione Estiva a.a 2024/2025
 -}


module Main where

-- inclusione delle librerie
import Text.Read (readMaybe)
import Data.Char (digitToInt)

type Giorno = Int
type Anno = Int
-- definizione dei tipi Mese, Giorno e Calendario
data Mese = Febbraio | Marzo | Aprile
              deriving (Eq,Ord, Enum, Bounded, Show, Read)
data GiornoGrasso = MartedìGrasso | GiovedìGrasso
              deriving (Eq,Ord,Enum,Show,Read)
data Calendario = Calendario
  { giorno :: Giorno,
    mese :: Mese,
    anno :: Anno
  } deriving (Show)
{- Funzione per formattare un numero tra 1-99 incluso per avere sempre due cifre:
 - l'argomento    un numero tra 1-99 incluso-}
formattaADueCifre :: Int -> String
formattaADueCifre n | n < 10 = "0" ++ show n
                    | n > 99 || n < 0 = error $ show (n) ++ "il numero deve essere tra 0 e 99"
                    | otherwise = show n
{- Funzione per la creazione del tipo Data:
 - il primo argomento    il giorno
 - il secondo argomento    il Mese
 - il terzo argomento    l'anno-}
creaData :: Giorno->Mese->Anno->Calendario
creaData x mese anno | x>0 && x<=giorniDelMese mese anno = Calendario{giorno = x,mese = Febbraio,anno = anno}

{- Funzione che restituisce quanti giorno ci sono in un mese:
 - il primo argomento    il mese
 - il secondo argomento    l'anno (importante per Febbraio) -}
giorniDelMese :: Mese -> Anno -> Giorno
giorniDelMese Febbraio anno | controllaBisestile anno = 29
                            | otherwise = 28
giorniDelMese mese _ | mese == Aprile = 30
                     | mese == Marzo = 31

{- Funzione che restituisce la data di Pasqua:
 - il primo argomento    l'anno -}
calcoloPasqua :: Anno -> Calendario
calcoloPasqua anno
    -- Caso speciale 25 aprile -> 18 aprile
    | giorno == 25 && mese == Aprile && d == 28 && e == 6 && a > 10 = creaData 18 Aprile anno
    -- Caso speciale 26 aprile -> 19 aprile
    | giorno == 26 && mese == Aprile = creaData 19 Aprile anno
    | otherwise = creaData giorno mese anno
  where giorno | z < 10 = z + 22
               | otherwise = z - 9
        mese | z < 10 = Marzo
             | otherwise = Aprile
        z = d + e
        d = (19 * a + mGreg) `mod` 30
        e = (2*b+4*c + 6 * d + nGreg) `mod` 7
        mGreg = 24
```

```
        nGreg = 5
        a = anno `mod` 19
        b = anno `mod` 4
        c = anno `mod` 7
{- Funzione che restituisce se l'anno    bisestile:
 - l'argomento    l'anno da verificare -}
controllaBisestile :: Anno -> Bool
controllaBisestile anno = (anno `mod` 4 == 0 && anno `mod` 100 /= 0) || (anno `mod` 400 == 0)

{- Funzione che calcola la data del Martedì o Giovedì grasso
 - il primo argomento    il giorno grasso
 - il secondo argomento    la data di pasqua dell'anno in cui si vuole calcolare i giorni grassi -}
calcolaGiornoGrasso :: GiornoGrasso -> Calendario -> Calendario
calcolaGiornoGrasso giornoGrasso pasqua
  | giornoGrasso == MartediGrasso = sottraiGiorniDaData 47 pasqua
  | giornoGrasso == GiovediGrasso = sottraiGiorniDaData 52 pasqua

{- Funzione che calcola la sottrazione di giorni ad una data:
 - il primo argomento    il numero di giorno da sottrarre
 - il secondo argomento    la data a cui sottrarre i giorni -}
sottraiGiorniDaData :: Giorno -> Calendario -> Calendario
sottraiGiorniDaData giorniScalare calendario
  -- caso base: i giorni da scalare sono meno del giorno del calendario
  | (giorno calendario) - giorniScalare > 0 = creaData (giornoCorrente-giorniScalare) meseCorrente annoCorrente
  -- caso ricorsivo: aggiunge i giorni del mese precedente finch  la sottrazione sia maggiore di 0
  | otherwise = sottraiGiorniDaData 0 dataAggiornata
    where
      dataAggiornata = Calendario{giorno = giorniRimasti, mese = mesePrecedente, anno = annoCorrente}
      giorniRimasti = (giornoCorrente - giorniScalare) + (giorniDelMese mesePrecedente annoCorrente)
      mesePrecedente = pred (mese calendario)
      annoCorrente = (anno calendario)
      giornoCorrente = (giorno calendario)
      meseCorrente = (mese calendario)

{- Funzione che trasforma una data nella rappesentazione ASCII art:
 - l'argomento    una data -}
formattaDataAscii :: Calendario -> String
formattaDataAscii giornoCalendario = unlines dataCombinata
      where
        spazioTraCaratteri = [replicate 5 " "]
        -- lista di lista contenente le cifre del giorno se    ad una cifra aggiunge uno zero davanti
        cifreAsci :: [[String]]
        cifreAsci = map (cifraInAsciiArt . digitToInt) (formattaADueCifre $ giorno giornoCalendario)
        giornoAscii = take 1 cifreAsci ++ spazioTraCaratteri ++ drop 1 cifreAsci
        -- lista di lista contentente le 3 lettere del mese
        meseAscii :: [[String]]
        meseAscii = (meseInAsciiArt (mese giornoCalendario))
        spazioTraGiornoMese = replicate 5 $ replicate 5 " "
        -- Combina cifre spazio e lettere insieme
        dataCombinata =  foldl1 (zipWith(++)) $ giornoAscii ++ spazioTraGiornoMese ++ meseAscii

acquisisciAnno :: IO Int
acquisisciAnno = do
                anno <- getLine
                -- validazione dei dati acquisiti
                case readMaybe anno of
                  (Just anno) | controllaAnno anno -> return anno
                  _ -> do
                     putStrLn "Input non valido. L'anno deve essere tra 1900 e 2099."
                     acquisisciAnno
{- Funzione per controllare che l'anno sia tra 1900-2099
 - l'argomento    l'anno -}
controllaAnno :: Anno -> Bool
controllaAnno anno = anno >= 1900 && anno <= 2099


{- Funzione che converte Mese nella sua rappresentazione ASCII art:
 - l'argomento    il mese da cui prendere l'ASCII art-}
meseInAsciiArt ::Mese -> [[String]]
meseInAsciiArt mese | mese == Febbraio = [["***** ",
                                           "*     ",
                                           "***** ",
                                           "*     ",
```

```
                                    "*     "],
                                   ["***** ",
                                    "*     ",
                                    "***** ",
                                    "*     ",
                                    "***** "],
                                   ["**** ",
                                    "*   *",
                                    "**** ",
                                    "*   *",
                                    "**** "]]
                    | mese == Marzo =    [["*   * ",
                                    "** ** ",
                                    "* * * ",
                                    "*   * ",
                                    "*   * "],
                                   ["***** ",
                                    "*   * ",
                                    "***** ",
                                    "*   * ",
                                    "*   * "],
                                   ["*****",
                                    "*   *",
                                    "*****",
                                    "*  * ",
                                    "*   *"]]
                    | mese == Aprile =   [["***** ",
                                    "*   * ",
                                    "***** ",
                                    "*   * ",
                                    "*   * "],
                                   ["***** ",
                                    "*   * ",
                                    "***** ",
                                    "*     ",
                                    "*     "],
                                   ["**** ",
                                    "*   *",
                                    "**** ",
                                    "*  * ",
                                    "*   *"]]
                    | otherwise =         [["  ?  "],
                                    ["  ?  "],
                                    ["  ?  "],
                                    ["  ?  "],
                                    ["  ?  "]]

{- Funzione che restituisce ASCII art di una cifra:
 - l'argomento    la cifra da cui prendere l'ASCII art -}
cifraInAsciiArt :: Int -> [String]
cifraInAsciiArt n | n == 0 = ["*****",
                              "*   *",
                              "*   *",
                              "*   *",
                              "*****"]
                  | n == 1 = ["  *  ",
                              " **  ",
                              "* *  ",
                              "  *  ",
                              "*****"]
                  | n == 2 = ["*****",
                              "    *",
                              "*****",
                              "*    ",
                              "*****"]
                  | n == 3 = ["*****",
                              "    *",
                              "*****",
                              "    *",
                              "*****"]
                  | n == 4 = ["*   *",
                              "*   *",
                              "*****",
```

```
                    "    *",
                    "    *"]
                | n == 5 = ["*****",
                    "*    ",
                    "*****",
                    "    *",
                    "*****"]
                | n == 6 = ["*****",
                    "*    ",
                    "*****",
                    "*   *",
                    "*****"]
                | n == 7 = ["*****",
                    "    *",
                    "    *",
                    "    *",
                    "    *"]
                | n == 8 = ["*****",
                    "*   *",
                    "*****",
                    "*   *",
                    "*****"]
                | n == 9 = ["*****",
                    "*   *",
                    "*****",
                    "    *",
                    "*****"]
                | otherwise = ["  ?  ", "  ?  ", "  ?  ", "  ?  ", "  ?  "]


main :: IO ()

-- definizione della funzione principale
main = do

  putStrLn("Programma per il calcolo di Giovedì e Martedì Grasso secondo il calendario Gregoriano")
  -- acquisizione dei due anni
  putStrLn("Inserisci l'anno per il Martedì Grasso")
  primoAnno <- acquisisciAnno
  putStrLn("inserisci l'anno per il Giovedì Grasso")
  secondoAnno <- acquisisciAnno
  -- stampa della data del Martedì e Giovedì Grasso
  putStrLn (formattaDataAscii (calcolaGiornoGrasso MartedìGrasso (calcoloPasqua primoAnno)))
  putStrLn (formattaDataAscii (calcolaGiornoGrasso GiovedìGrasso (calcoloPasqua secondoAnno)))
```

4.2 Implementazion in Prolog

Nome del sorgente: `martedi_giovedi_grasso.pl`

```
/*
 * Progetto di Programmazione Logica e Funzionale
 * Studenti: Elia Renzoni e Gianmarco Beligni
 * N. Matricola: 319978 321069
 * Sessione Estiva a.a 2024/2025
*/

mese(febbraio).
mese(marzo).
mese(aprile).
precedente(febbraio, marzo).
precedente(marzo, aprile).
precedente(aprile, maggio).
successivo(Mese1, Mese2):-
  precedente(Mese2, Mese1).

/* regola che restituisce la data di Pasqua:
   gli argomenti sono il giorno, mese e anno */
pasqua(Giorno, Mese, Anno) :-
    A is Anno mod 19,
    B is Anno mod 4,
```

```
    C is Anno mod 7,
    D is (19*A + 24) mod 30,
    E is (2*B + 4*C + 6*D + 5) mod 7,
    Z is D + E,

    % Calcolo della Data
    (Z < 10 ->
        BaseG is 22 + Z, BaseM = marzo
    ;
        BaseG is Z - 9, BaseM = aprile
    ),

    % Applica Eccezioni sulla data calcolata
    (BaseG =:= 26, BaseM = aprile ->
        Giorno = 19, Mese = aprile
    ; BaseG =:= 25, BaseM = aprile, D =:= 28, E =:= 6, A > 10 ->
        Giorno = 18, Mese = aprile
    ;
        Giorno = BaseG, Mese = BaseM
    ).


/* regola per controllare che l'anno sia bisestile:
   l'argomento    l'anno
*/
controlla_bisestile(Anno) :-
    (Anno mod 400 =:= 0;
     (Anno mod 100 =\= 0, Anno mod 4 =:= 0)).


crea_data(Giorno, Mese, Anno) :-
    integer(Anno), Anno >= 1900, Anno =< 2099,
    integer(Mese), Mese >= 1, Mese =< 12,
    integer(Giorno), Giorno >= 1,
    giorni_del_mese(GiornoMax, Mese, Anno),
    Giorno =< GiornoMax.

/* regola che restituisce quanti giorno ci sono in un mese:
   il primo argomento    il giorno
   il secondo argomento    il mese
   il terzo argomento    l'anno (importante per Febbraio)
*/
giorni_del_mese(28, febbraio, Anno) :- \+ controlla_bisestile(Anno).
giorni_del_mese(29, febbraio, Anno) :- controlla_bisestile(Anno).
giorni_del_mese(31, Mese, _) :- Mese = marzo.
giorni_del_mese(30, Mese, _) :- Mese = aprile.


/* regola che calcola la sottrazione di giorni ad una data:
   il primo argomento    il numero di giorno da sottrarre
   il secondo argomento    la data a cui sottrarre i giorni
   il terzo argomento    il risultato della sottrazione
*/
sottrai_giorni(Sottraendo, data(Giorno, Mese, Anno), DataArrivo) :-
    GiornoSufficiente is Giorno - Sottraendo,
    (GiornoSufficiente > 0 ->
        DataArrivo = data(GiornoSufficiente, Mese, Anno)
    ;
       precedente(NuovoMese, Mese),
       giorni_del_mese(MaxGiorni, NuovoMese, Anno),
       NuovoGiorno is Giorno - Sottraendo + MaxGiorni,
       sottrai_giorni(0, data(NuovoGiorno, NuovoMese, Anno), DataArrivo)
    ).

/* regola che calcola la data del Martedì o Giovedì grasso:
   il primo argomento    il giorno grasso
   il secondo argomento    la data di pasqua dell'anno in cui si vuole calcolare i giorni grassi
*/
calcola_martedi_grasso(Anno, MartediGrasso) :-
    pasqua(PGiorno, PMese, Anno),
    sottrai_giorni(47, data(PGiorno, PMese, Anno), MartediGrasso).

calcola_giovedi_grasso(Anno, GiovediGrasso) :-
```

```
    pasqua(PGiorno, PMese, Anno),
    sottrai_giorni(52, data(PGiorno, PMese, Anno), GiovediGrasso).

calcola_martedi_giovedi_grasso(martedi, Anno, MartediGrasso, _) :-
    calcola_martedi_grasso(Anno, MartediGrasso).

calcola_martedi_giovedi_grasso(giovedi, Anno, _, GiovediGrasso) :-
    calcola_giovedi_grasso(Anno, GiovediGrasso).

/* regola che restituisce ASCII art di una cifra:
   l'argomento    la cifra da cui prendere l'ASCII art
*/
giorno_ascii(Num, GiornoCodificato) :-
    (Num == 0 -> GiornoCodificato = ['*****',
                                     '*   *',
                                     '*   *',
                                     '*   *',
                                     '*****']);
    (Num == 1 -> GiornoCodificato = ['  *  ',
                                     ' **  ',
                                     '* *  ',
                                     '  *  ',
                                     '*****']);
    (Num == 2 -> GiornoCodificato = ['*****',
                                     '    *',
                                     '*****',
                                     '*    ',
                                     '*****']);
    (Num == 3 -> GiornoCodificato = ['*****',
                                     '    *',
                                     '*****',
                                     '    *',
                                     '*****']);
    (Num == 4 -> GiornoCodificato = ['*   *',
                                     '*   *',
                                     '*****',
                                     '    *',
                                     '    *']);
    (Num == 5 -> GiornoCodificato = ['*****',
                                     '*    ',
                                     '*****',
                                     '    *',
                                     '*****']);
    (Num == 6 -> GiornoCodificato = ['*****',
                                     '*    ',
                                     '*****',
                                     '*   *',
                                     '*****']);
    (Num == 7 -> GiornoCodificato = ['*****',
                                     '    *',
                                     '    *',
                                     '    *',
                                     '    *']);
    (Num == 8 -> GiornoCodificato = ['*****',
                                     '*   *',
                                     '*****',
                                     '*   *',
                                     '*****']);
    (Num == 9 -> GiornoCodificato = ['*****',
                                     '*   *',
                                     '*****',
                                     '    *',
                                     '*****']).

/* regola che converte Mese nella sua rappresentazione ASCII art:
   il primo argomento    il mese da cui prendere l'ASCII art
   il secondo argomento    il mese codificato in ASCII art
*/
mese_ascii(Mese, MeseCodificato) :-
  (Mese == febbraio -> MeseCodificato = [['*****',
                                          '*    ',
                                          '*****',
                                          '*    ',
```

```
                                          '*    '],
                                         ['*****',
                                          '*    ',
                                          '*****',
                                          '*    ',
                                          '*****'],
                                         ['****  ',
                                          '*   *',
                                          '****  ',
                                          '*   *',
                                          '****  ']]);
  (Mese == marzo -> MeseCodificato = [['*   *',
                                      '* * *',
                                      '*   *',
                                      '*   *',
                                      '*   *'],
                                     ['*****',
                                      '*   *',
                                      '*****',
                                      '*   *',
                                      '*   *'],
                                     ['*****',
                                      '*   *',
                                      '*****',
                                      '*  * ',
                                      '*   *']]);
  (Mese == aprile -> MeseCodificato = [['*****',
                                       '*   *',
                                       '*****',
                                       '*   *',
                                       '*   *'],
                                      ['*****',
                                       '*   *',
                                       '*****',
                                       '*    ',
                                       '*    '],
                                      ['*****',
                                       '*   *',
                                       '*****',
                                       '*  * ',
                                       '*   *']]).

/* regola per acquisire l'anno dall'utente:
   l'argomento    l'anno che rientra nei limiti definiti dalla regola controlla_anno
   questa regola viene chiamata ricorsivamete fino a quando non    soddisfatta la regola controlla_anno
*/
acquisisci_anno(AnnoScelto) :-
  catch(
    (
      read(Anno),
      controlla_anno(Anno, AnnoScelto)
    ),
    _Errore,
    (
      stampa_errore,
      acquisisci_anno(AnnoScelto)
    )
  ).


/* regola per controllare l'anno:
   gli utlimi due parametri indicano l'input inserito dall'utente
   e l'anno da restituire.
*/
controlla_anno(AnnoLetto, AnnoRestituire) :-
  (   AnnoLetto >= 1900,
      AnnoLetto =< 2099
  ->  AnnoRestituire = AnnoLetto
  ;   stampa_errore,
      acquisisci_anno(AnnoRestituire)
  ).

/* regola per controllare l'anno
```

```
   il primo argomento    l'anno da controllare,
   il secondo argomento    l'anno
*/
stampa_errore :-
  write('Input non valido. L''anno deve essere tra 1900 e 2099.\n').

/* regola per stampare caratteri in formato gigante:
   il primo argomento    la lista dei caratteri della prima parola,
   il secondo argomento    la lista dei caratteri della seconda parola
   esempio: stampa_gigante("CIAO", "MONDO")
*/
stampa_gigante(Caratteri1, Caratteri2) :-
  stampa_righe_giganti(0, Caratteri1, Caratteri2).
/* regola ricorsiva per stampare righe di caratteri giganti:
   il primo argomento    l'indice della riga corrente (0-4),
   il secondo argomento    la lista dei caratteri della prima parola,
   il terzo argomento    la lista dei caratteri della seconda parola,
   caso base: quando raggiunge la 5a riga (indice 5) termina con cut (!)
*/
stampa_righe_giganti(5, _, _) :- !.

/* caso ricorsivo per stampare una riga di caratteri giganti:
   il primo argomento    l'indice della riga corrente,
   il secondo argomento    la lista dei caratteri della prima parola,
   il terzo argomento    la lista dei caratteri della seconda parola,
   - stampa la riga per la prima parola
   - aggiunge 4 spazi
   - stampa la riga per la seconda parola
   - va a capo
   - incrementa l'indice e chiama ricorsivamente
*/
stampa_righe_giganti(RigaIndice, Caratteri1, Caratteri2) :-
  stampa_riga(RigaIndice, Caratteri1),
  write('    '),  % Spazio tra primo e secondo gruppo
  stampa_riga(RigaIndice, Caratteri2),
  nl,
  Successivo is RigaIndice + 1,
  stampa_righe_giganti(Successivo, Caratteri1, Caratteri2).
/* regola per stampare una singola riga di caratteri:
   caso base: lista vuota, termina
*/
stampa_riga(_, []).
/* caso ricorsivo per stampare una riga di caratteri:
   il primo argomento    l'indice della riga da stampare,
   il secondo argomento    la lista dei caratteri,
   - estrae la riga specifica dal carattere corrente
   - stampa la riga con uno spazio
   - procede ricorsivamente con il resto della lista
*/
stampa_riga(Index, [Lettera | Resto]) :-
  nth0(Index, Lettera, Riga),
  write(Riga), write(' '),
  stampa_riga(Index, Resto).


/* regola che trasforma una data nella rappesentazione ASCII art:
   l'argomento    una data
*/
stampa_caratteri_giganti(Giorno, Mese) :-
  Unita is Giorno // 10,
  Decina is Giorno mod 10,
  giorno_ascii(Unita, UnitaCodificata),
  giorno_ascii(Decina, DecinaCodificata),
  mese_ascii(Mese, MeseCodificato),
  stampa_gigante([UnitaCodificata, DecinaCodificata], MeseCodificato).


% definizione della regola principale
programma :-
  write('Programma per il calcolo di Giovedì e Martedì Grasso secondo il calendario Gregoriano\n'),
  write('Inserire l'' anno per calcolare il Martedì Grasso\n'),
  acquisisci_anno(PrimoAnno),
  write('Inserire l'' anno per calcolare il Giovedì Grasso\n'),
```

```
acquisisci_anno(SecondoAnno),
calcola_martedi_grasso(PrimoAnno, MartediGrasso),
calcola_giovedi_grasso(SecondoAnno, GiovediGrasso),
data(Giorno, Mese, _) = MartediGrasso,
stampa_caratteri_giganti(Giorno, Mese),
data(GiornoG, MeseG, _) = GiovediGrasso,
stampa_caratteri_giganti(GiornoG, MeseG).
```

5 Testing del Programma

In questa sezione abbiamo riportato 10 test significativi che provano il funzionamento della soluzione al problema. Abbiamo inserito 5 test sia per il sorgente in Haskell che per quello in Prolog. L'ultimo test di ogni batteria dimostra l'affidabilità del programma in caso di errori di inserimento.

5.1 Testing del Sorgente Haskell

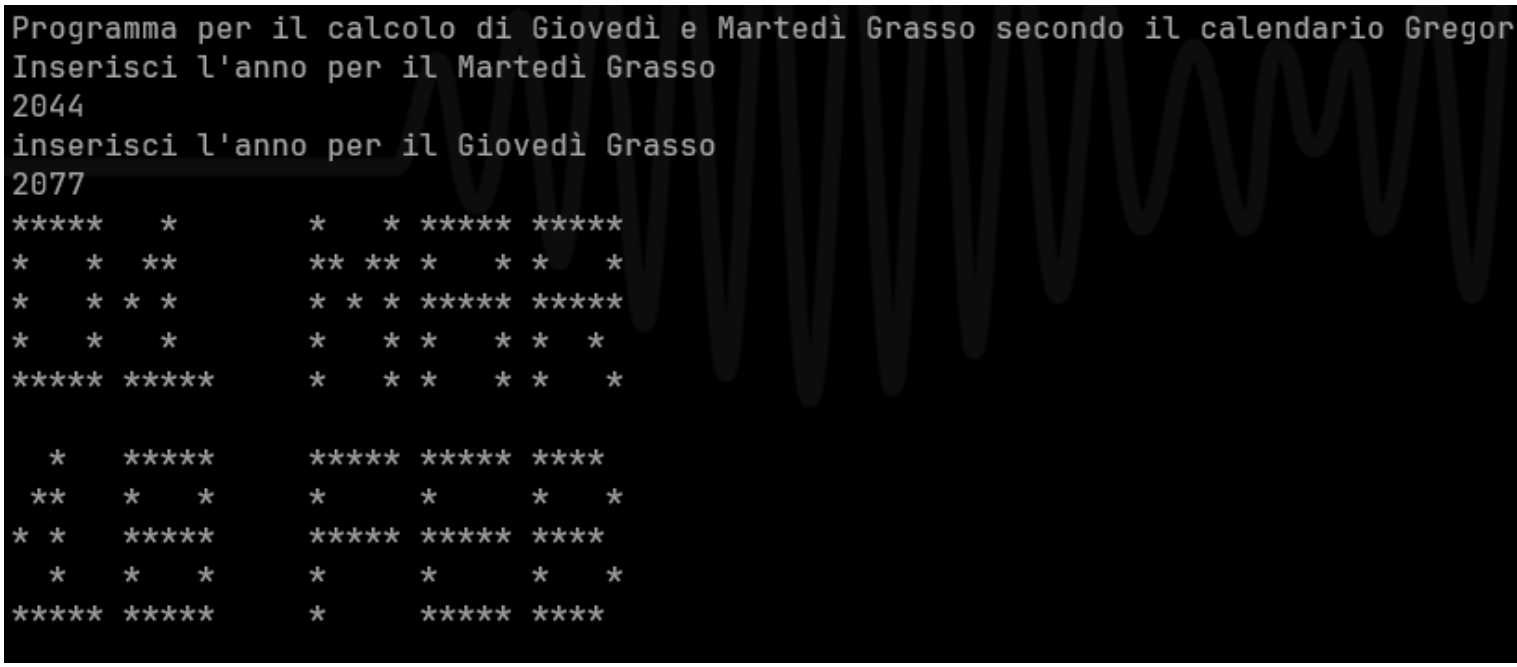

Figura 1: Test Haskell 1

Figura 2: Test Haskell 2

Figura 3: Test Haskell 3

Figura 4: Test Haskell 4

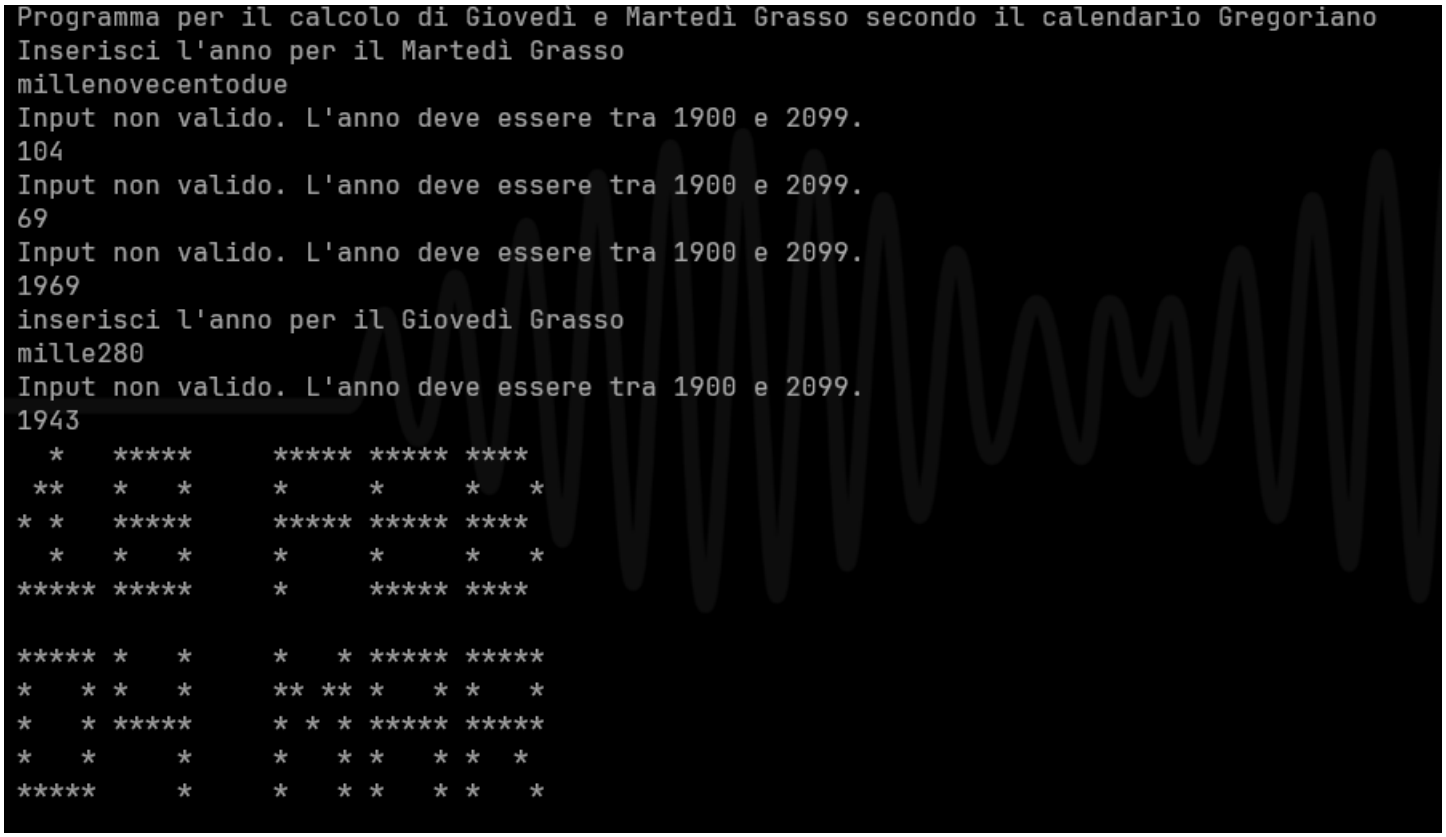

Figura 5: Test Haskell 5

5.2 Testing del Sorgente Prolog

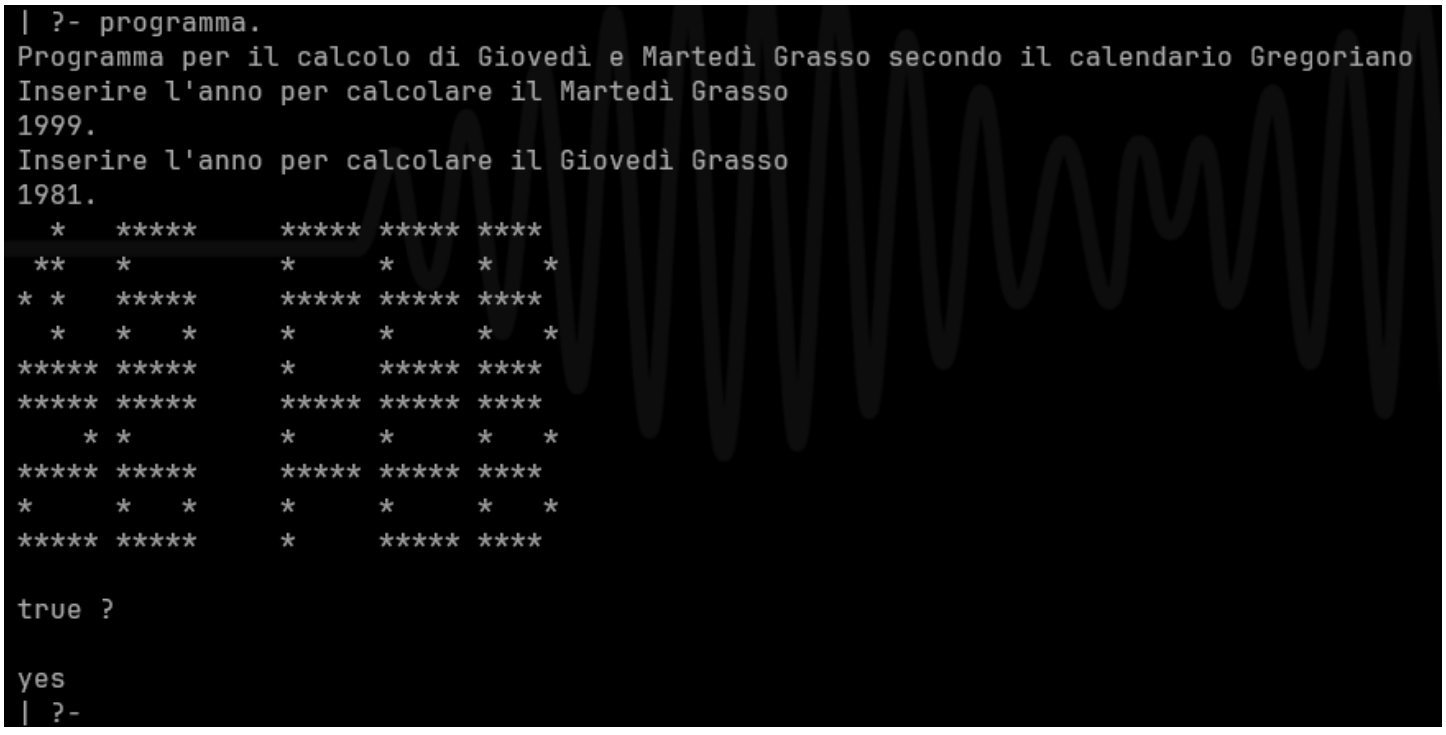

Figura 6: Test Prolog 1

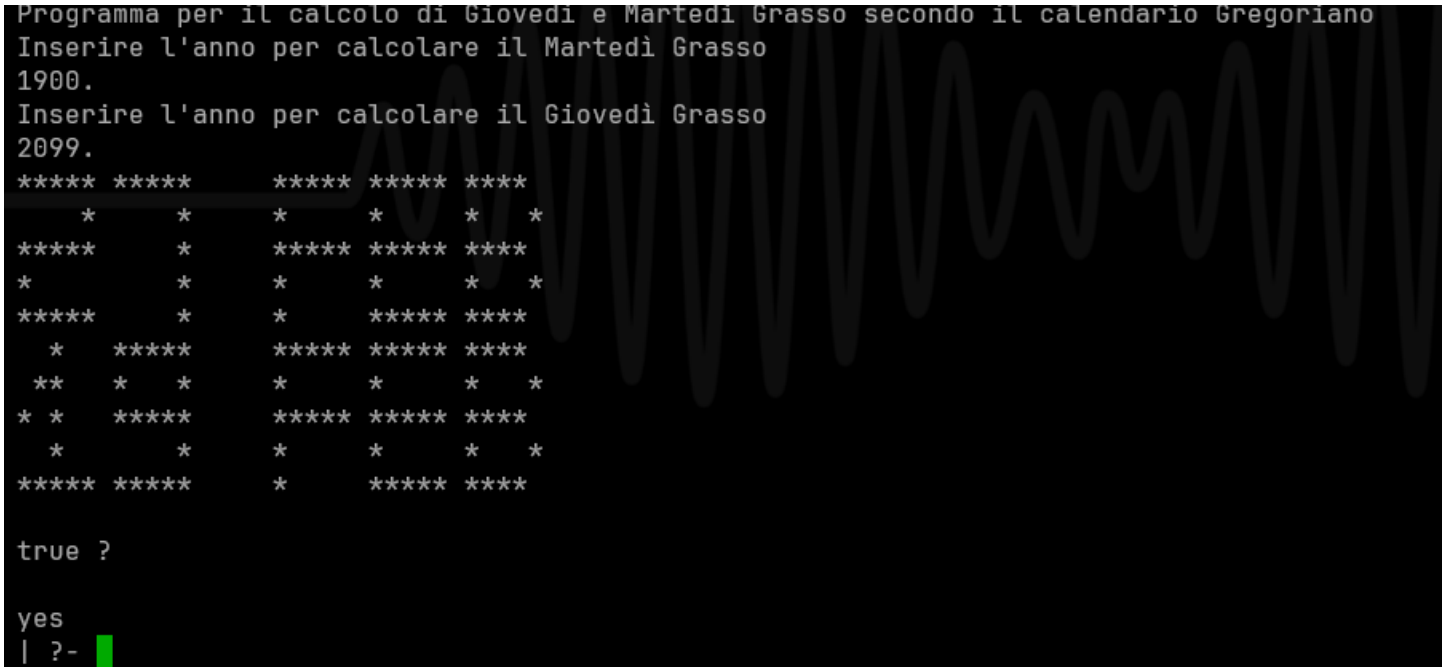

Figura 7: Test Prolog 2

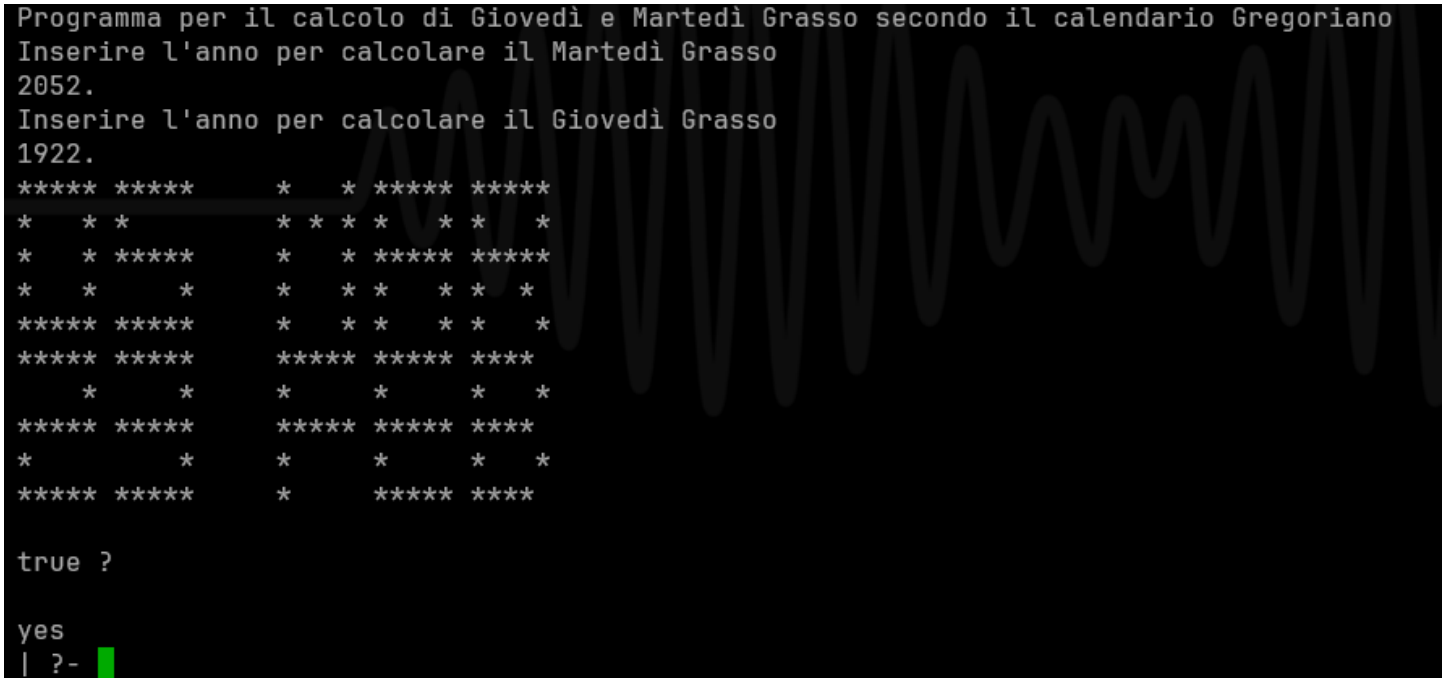

Figura 8: Test Prolog 3

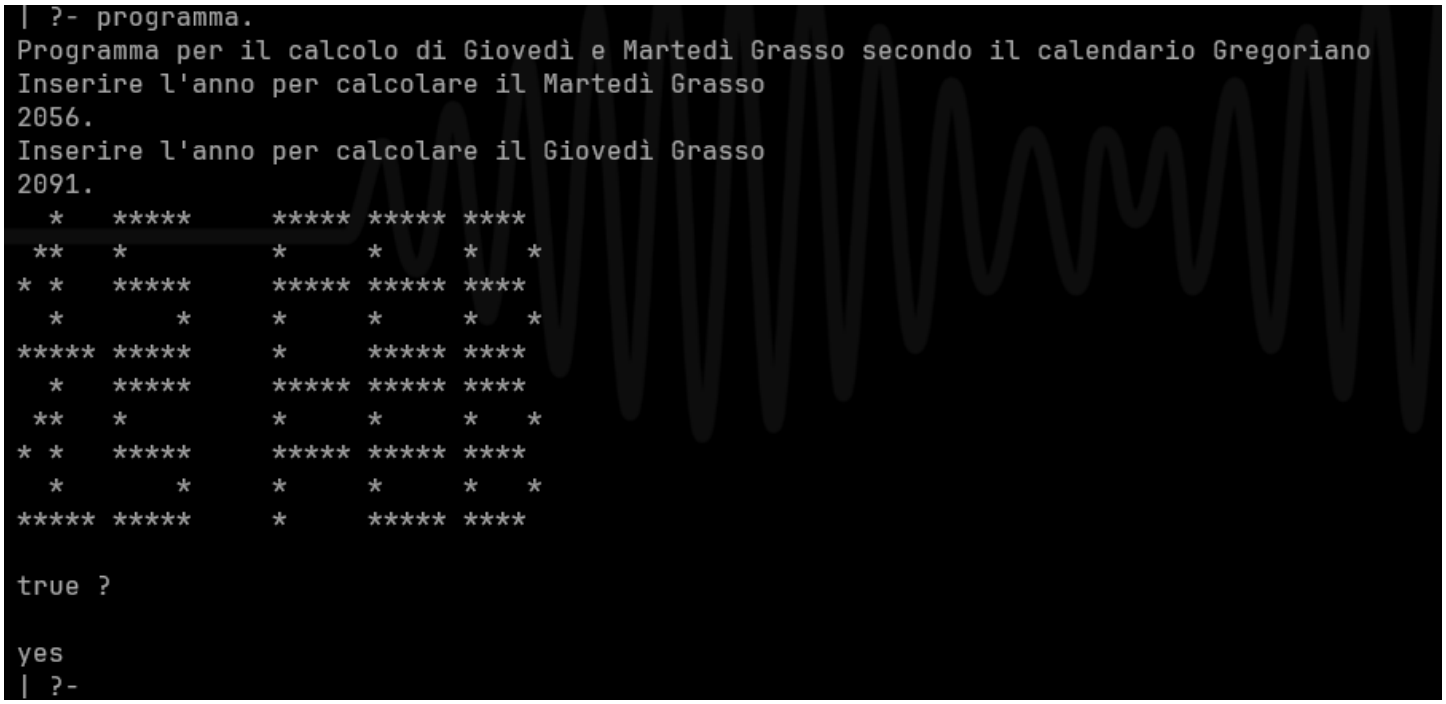

Figura 9: Test Prolog 4

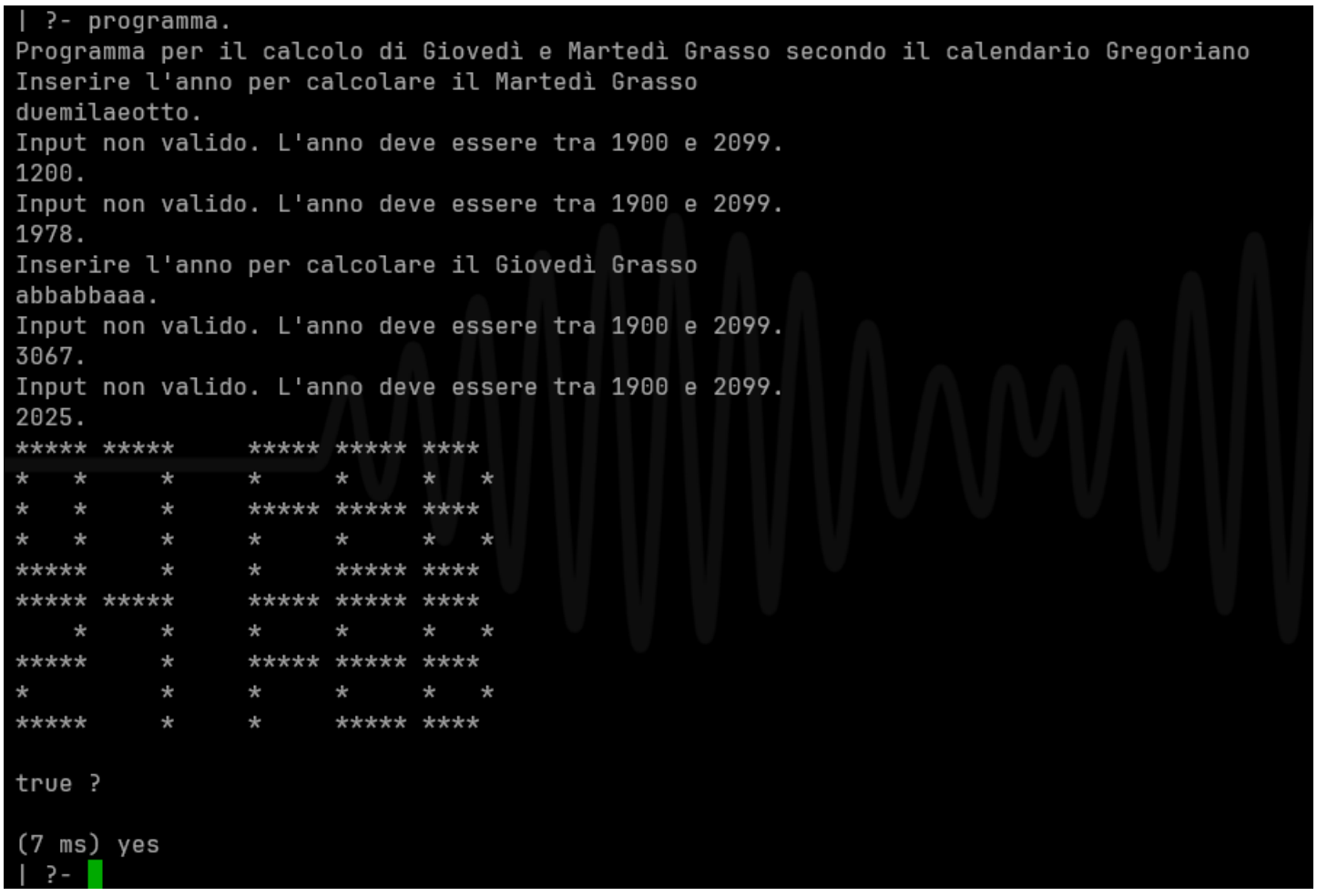

Figura 10: Test Prolog 5