

Albero Genealogico

Progetto Linguaggi e Compilatori 2021/22

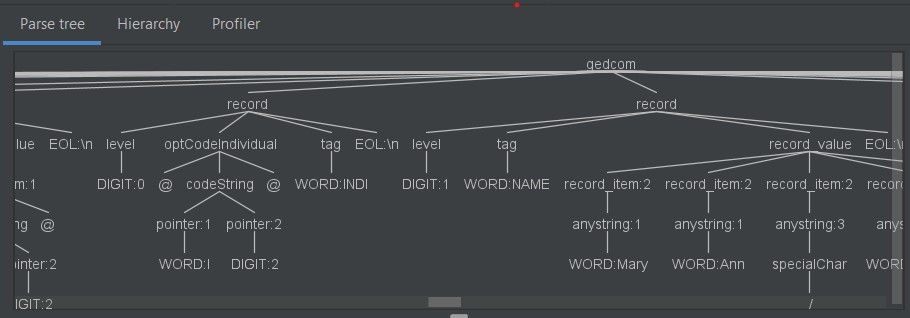
Francesco Chiocchi | Linguaggi e Compilatori | 20/06/2022

# Grammatica

Usando il tool di IntelliJ per ANTLR4 ho costruito una grammatica generica dove ho immaginato che un file di tipo Gedcom è formato da un insieme di record, i quali hanno un livello, un tag opzionale rappresentante l’inizializzazione di un individuo o di una famiglia, un tag obbligatorio e il valore del record che è opzionale.

Infine, come detto all’inizio, un file di tipo Gedcom deve contenere anche un ultimo record di tipo richiesta dal quale poi partiranno le operazioni di calcolo degli antenati/discendenti di un certo individuo.

(Nella cartella condivisa metterò a disposizione l’immagine completa dell’albero di parsing, che ovviamente essendo molto grande non entra in Word).

Esempio parziale di un albero di parsing per l’individuo @I2@ nel file Sample.ged:  


# Implementazione

Innanzitutto, inizialmente avevo scelto di usare un visitor anziché un listener durante la visita dell’albero perché potevo ridirigere a mio piacimento le visite dei nodi. Poi mi sono reso conto che per il mio scopo non serviva quindi ho deciso di rimodificare la classe da cui fare overriding e ho usato quindi un semplice listener.

Il core dell’applicazione è proprio quando viene scoperta la grammatica, quindi quando entro nel metodo enterGedcom(ctx). Qui si inizializzano le mappe relative agli individui e alle famiglie presenti nella classe FamilyTree con un ciclo for visitando tutti i record presenti nel file. Vengono quindi aggiunti ad elements e families tutti i codici degli Individui/Famiglie, in cui il record il cui livello è zero e il tag obbligatorio è rispettivamente “INDI”, per gli individui e “FAM” per le famiglie.

Contemporaneamente, per ogni record visitato, vengono impostate le caratteristiche di ogni individuo (nomi e cognomi se conosciuti, eventuali data e luogo di nascita, data e luogo di morte, luogo di sepoltura, puntatori al padre e alla madre se conosciuti) e ogni famiglia (eventuale data e luogo di Matrimonio, composizione di una famiglia).

Per ottenere poi tutti gli individui ho letto nell’ultima riga del file la richiesta, che può essere o di tipo ANCE o DESC, quando visito la regola enterRequest(ctx). Mi sono servito poi di un insieme generico Set<String> contenente tutti i codici degli individui risultanti dalla visita del file dopo aver preso la richiesta e chiamato i metodi getAncestorOf(Individual i) o getDescendantOf(Individual i), sempre a seconda del tipo di richiesta del file e poi li ho stampati a video quando la grammatica viene chiusa con exitGedcom(ctx).

Parti più importanti dell’enterGedcom(ctx):

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

# Testing

Si è passato infine all’ultima fase, quella di testing, dove si allegano i vari screenshots dei file EsempioRossi.ged e Sample.ged. Inoltre, ho aggiunto un ulteriore file Gedcom riguardante la famiglia dei Sayan del manga “**Dragon Ball**”.

**EsempioRossi.ged** 🡪Richiesta = 0 ANCE @I322382377075@

Immagine che contiene testo

Descrizione generata automaticamente

**EsempioRossi.ged** 🡪Richiesta = 0 DESC @I322381934457@

Immagine che contiene testo

Descrizione generata automaticamente

**Sample.ged** 🡪Richiesta = 0 DESC @I1@

Immagine che contiene testo

Descrizione generata automaticamente

**DragonBall.ged** 🡪 0 ANCE @I12@ (Gothen)

Immagine che contiene testo

Descrizione generata automaticamente

PROGETTO DISPONIBILE ANCHE IN: [ProjectLC](https://github.com/FrancescoChiocchi8/ProjectLC)

