Progetto di Compilatori 2022-2023

Raffaele Terracino

11 luglio 2023

1 Grammatica progettata

Il primo passo per lo sviluppo del progetto è stato costruire una grammatica context-free che genera il linguaggio descritto dalla traccia. L'alfabeto della grammatica è l'insieme delle lettere maiuscole e minuscole dell'alfabeto italiano, delle cifre da 0 a 9, e dei caratteri speciali [;.:"%!-]. L'assioma della grammatica è "input", difatti la principale produzione è:
`input -> data%%biblioteca!!lista_pres`
La produzione rappresenta la generica forma di un file di input descritto dalla traccia. Seguono le altre produzioni della grammatica.

```
data -> GGMM-GGMM-AA
biblioteca -> scrittore->lista_libri --- biblioteca | ϵ
scrittore -> NAME
lista_libri-> Q_NAME DOT WS BOOK_CODE SEMICOLON WS lista_libri
| ϵ
lista_pres -> utente LEFT_PAR pres_utente RIGHT_PAR lista_pres
| ϵ
utente -> Q_NAME WS SEP WS CF WS
pres_utente -> BOOK_CODE COLON WS data_prestito SEMICOLON pres_utente
| WS pres_utente | ϵ
data_prestito -> GGMM SLASH GGMM SLASH AA
```

I simboli terminali sono riportati in maiuscolo e corrispondono ai token descritti nella successiva sezione. Vengono ignorati i caratteri di ritorno a capo e tabulazione, mentre gli spazi assumono un importante ruolo nella formattazione del file.

2 Analisi lessicale

I lessemi più semplici sono i separatori delle sezioni del file, a cui corrispondono le espressioni regolari "%%" e "!!" e, rispettivamente, i token `BEG_LIB` e `BEG_PRES`. Il token per i giorni e mesi dell'anno è `GGMM`, descritto dall'espressione regolare `[0-9]{2}`, mentre per gli anni il token corrispondente è `AA`, descritto dall'espressione `[0-9]{4}`. Spetta all'analizzatore sintattico il compito di effettuare i controlli richiesti per la data. Il lessema per gli scrittori, il cui to-

ken corrispondente è NAME, è descritto dall'espressione regolare ([A-z][a-z]*[ ]?){2,}, che include la possibilità di avere scrittori con più nomi o cognomi. I libri dei titoli e i nomi utente sono descritti dall'espressione regolare "[a-zA-Z0-9]+", corrispondente al token Q_NAME. Per quanto riguarda i codici dei libri, si utilizza l'espressione [0-9]{3}-[0-9]{5}[A-Z]{2}, legata al token BOOK_CODE. Il codice fiscale degli utenti, il cui token è CF è descritto dall'espressione [A-Z0-9]{16}. Gli altri simboli da tokenizzare, quali i simboli di interpunzione, le parentesi e -> hanno come espressione regolare i simboli stessi e come token il rispettivo nome in inglese. Lo scanner, realizzato con flex e descritto nel file scanner.fl, ha il compito di riconoscere i lessemi mediante le espressioni descritte e passare i corrispondenti token al parser.

3 Analisi sintattica

Il parser, realizzato con Bison, inizia con l'inclusione dei file di intestazione delle strutture dati utilizzate, ossia la tabella dei simboli e uno stack di stringhe, descritte nella successiva sezione, e la dichiarazione di un puntatore a carattere, char *autore, necessario per il corretto inserimento dei libri nella tabella dei simboli. Viene inoltre dichiarata una union contenente il puntatore a carattere str. Il parser effettua le seguenti azioni in corrispondenza di alcune produzioni.

- Per le produzioni data e data_prestito, si controlla che giorni e mesi dell'anno siano corretti, ovvero che il giorno sia un intero compreso tra 0 e 31 e che il mese sia compreso tra 0 e 12;
- per la produzione biblioteca -> ϵ, si controlla se il puntatore autore, che viene aggiornato nella produzione scrittore -> NAME, sia nullo, caso in cui si stampa un errore sullo standard ouput e si esce dal programma. Difatti il testo richiede che la lista di libri sia non vuota;
- per la produzione non vuota della lista di libri, attraverso la funzione strncpy e l'aritmetica dei puntatori, si tolgono le virgolette dal titolo del libro e lo si inserisce nella tabella dei simboli, passando anche codice e scrittore;
- riconosciuta la produzione data_prestito, se valida, si effettua un push della data in una pila. Questa operazione è necessaria in quanto si è notato che, con la grammatica progettata, le date apparivano in ordine inverso rispetto ai prestiti;
- riconosciuto un prestito, si effettua un pop dalla pila delle date e si aggiunge, per il libro specificato, un prestito in quella data.

La funzione main si aspetta due argomenti, rispettivamente il path del file di input e quello del file di output. Se il numero di parametri è minore di 2, il programma stampa un errore e termina. Succede la stessa cosa se il file di input specificato non esiste. Dopo la chiamata a yyparse(), se non vi sono

errori, attraverso le funzioni della tabella dei simboli, vengono stampate sul file di output specificato la lista dei libri disponibili e quella dei libri in prestito, nel formato specificato dalla traccia. Infine, viene liberata la memoria allocata per la tabella dei simboli, e il programma termina con successo stampando un avviso sulla creazione del file.

4 Strutture dati implementate

4.1 Strutture di base

Il componente base per la gestione di libri e prestiti è la struct `Book`, che incapsula titolo, autore e codice come stringhe, un booleano isLoaned che indica se il libro è in prestito e, infine, la data del prestito come stringa. Si ha che isLonaed è 0 se e solo se la data del prestito è NULL. Si è scelta questa modalità di gestione dei prestiti per evitare problemi di null dereferencing. La struttura BookNode rappresenta una lista concatenata di Libri, in quanto incapsula un puntatore a Book e un puntatore a BookNode `next`.

4.2 Tabella dei simboli

La tabella dei simboli è implementata attraverso una hash table con chaining e con funzione hash basata sul metodo della divisione. Si stabilisce una dimensione massima, `HASHSIZE`, pari a 101. Tale dimensione è usata nella dichiarazione un array di puntatori `hashTable`, rappresentante la vera e propria tabella dei simboli, e nella funzione hash, incapsulata dalla funzione `hash(char *codice)`. Poichè ogni codice identifica univocamente un determinato libro, si è scelto di basare la funzione hash su tale campo. Le principali funzioni della tabella dei simboli sono `lookup(char *bookCode)`, che restituisce un puntatore alla lista `hashTable[hash(bookCode)]`, `insert(char *title, char *bookCode, char *author)`, che inserisce il libro in coda alla lista restituita da `lookup`, `int addLoan(char *bookCode, char *date)` che aggiorna un libro inserendo la data del prestito. Seguono le funzioni `void printAvailableBooks(FILE *stream)` e `void printLoanedBooks(FILE *stream)`, che rispettivamente stampano su file la lista dei libri disponibili e quella dei libri in prestito. Infine vi sono le funzioni atte a liberare la memoria allocata, quali `freeHashTable, freeList e freeBook`. La tabella dei simboli è contenuta nei file symboltable.c e symboltable.h

4.3 Stack

Lo stack implementato per risolvere la problematica descritta nella sezione precedente è contenuto nei file stack.c e stack.h. La struttura su cui si appoggia è `StackNode`, che incapsula un `char*` e un puntatore a StackNode chiamato next. Le funzioni implementate sono tre: `stackPush(char *string, char *stackPop(void) e void stackIsEmpty(void)`. Non vi è bisogno di una funzione che liberi la memoria dello stack in quanto nel parser, ogni volta che si

inserisce un prestito, si libera la memoria usata per la stringa estratta dallo stack.

5 Note finali

Il progetto è stato testato sui sistemi operativi "Windows 10" e "Ubuntu 22.04 LTS". I file di input per testare il programma risiedono nella directory "Input-Files", che comprende due esempi funzionanti e due esempi per cui il parser rileva un errore.