Relazione progetto

Compilatori

Giovanni Casano

Emanuele Castronovo

Tournée

Dipartimento di matematica e informatica
Università degli Studi di Palermo
Palermo
A.a. 2023/24

Indice

Capitolo 1

Introduzione

Il progetto prevede la realizzazione di un programma, con l'ausilio di Flex e Bison, che supporti il responsabile delle tappe italiane della tournée di un famoso gruppo musicale. Il file di input è costituito da tre sezioni distinte.

1.1 Formato del File di Input

Sezione 1: Tappe del Tour La prima sezione rappresenta le tappe del tour. Ogni tappa è descritta nel seguente formato:

```
Città: nome_citta;
Location: nome_location;
Data: data_tappa1;
Biglietti tipo A: (elenco numero biglietti per categoria);
Biglietti tipo B: (elenco numero biglietti per categoria);
Biglietti tipo C: (elenco numero biglietti per categoria);
Biglietti tipo D: (elenco numero biglietti per categoria);
********************
```

I dettagli di questa sezione includono:

- **nome città** - Può essere una o più parole, separate da uno spazio, tutte in lettere maiuscole e apostrofi (es. PALERMO, L'AQUILA, SAN BENEDETTO DEL TRONTO).
- **nome location** - Una o più parole, separate da uno spazio, con la prima lettera maiuscola e le altre minuscole (es. Palasport, Stadio San Siro, Teatro Metropolitan).

- **data tappa** - Data nel formato GG-MM-AAAA.
- **elenco biglietti per categoria** - Lista del numero di biglietti staccati per categoria. Le categorie sono: Ordinario, Musicisti, Junior, Senior, Disabili, Autorità, indicate dalla loro iniziale seguita da ":" e dal numero di biglietti staccati (es. (O: 12288; M:1130; J:1118; S:323; D: 52; A:89)).

La fine delle informazioni di una tappa è segnata da una linea di 20 asterischi:

```
********************
```

La fine della sezione delle tappe è indicata da tre simboli di dollaro:

```
$$$
```

Sezione 2: Prezzi dei Biglietti La seconda sezione contiene i prezzi dei biglietti di tipo A, B, C, D nel formato:

```
PrezzoA --> prezzo.
PrezzoB --> prezzo.
PrezzoC --> prezzo.
PrezzoD --> prezzo.
++++++++++++++++++++
```

Il prezzo è un numero reale con due cifre decimali seguito dalla parola "euro". La sezione termina con una linea di 20 simboli +:

```
++++++++++++++++++++
```

Sezione 3: Sconti Applicabili La terza sezione indica gli sconti per categoria:

```
Nome_categoria: perc_sconto %
```

Le categorie di riduzione sono: Ordinario, Musicisti, Junior, Senior, Disabili, Autorità.

1.1.1 Formato del File di Output

Il programma deve produrre un output che riporta per ogni tappa l'importo guadagnato dalla vendita dei biglietti per ciascuna categoria, tenendo conto degli sconti applicati, e l'importo destinato alla beneficenza. Inoltre, deve calcolare il totale ottenuto dalla vendita dei biglietti e il totale dato in beneficenza, considerando che il gruppo destina in beneficenza:

- l'1% delle vendite per tappe con guadagno $\leq$ 3.700.000 Euro,
- l'1,2% per tappe con guadagno tra 3.700.000 e 4.500.000 Euro,
- l'1,8% per tappe con guadagni superiori a 4.500.000 Euro.

Esempio di Output

```
Roma Stadio Olimpico
Vendita biglietti --> euro 3.636.180,95
Beneficienza --> euro 36.361,81
---
Milano Stadio San Siro
Vendita biglietti --> 3.798.247
Beneficienza --> 45.578,96
---
Reggio Emilia Campovolo
Vendita biglietti --> 4.797.635
Beneficienza --> 86.357,43
%%%
Totale vendita biglietti: 12.232.062,95
Totale Beneficienza: 168298,2
```

L'output riporta le stesse informazioni per ciascuna tappa e un riepilogo finale con il totale delle vendite e della beneficenza.

Capitolo 2

Progettazione della Grammatica

La progettazione della grammatica è stata eseguita per garantire che il parser possa riconoscere e processare correttamente i dati degli eventi, i prezzi dei biglietti e gli sconti applicabili. La grammatica è stata progettata con i seguenti simboli terminali e non terminali, ottimizzati per le grammatiche LALR(1) e verificati per assenza di conflitti di tipo Shift-Reduce e Reduce-Reduce.

2.1 Simboli Terminali

I simboli terminali della grammatica includono:

- `%token citta location dataTappa separatoreTappa`
- `%token ordinariA musicistiA juniorA seniorA disabiliA autoritaA`
- `%token ordinariB musicistiB juniorB seniorB disabiliB autoritaB`
- `%token ordinariC musicistiC juniorC seniorC disabiliC autoritaC`
- `%token ordinariD musicistiD juniorD seniorD disabiliD autoritaD`
- `%token prezzoA prezzoB prezzoC prezzoD`
- `%token scontoA scontoO scontoM scontoS scontoJ scontoD`

2.2 Simboli Non Terminali

I simboli non terminali della grammatica includono:

- `input`

- `tappa`
- `fine`
- `bigliettoA`
- `bigliettoB`
- `bigliettoC`
- `bigliettoD`
- `prezzi`
- `sconti`

2.3 Osservazione sulla Distinzione dei Token

Pur quanto i simboli non terminali relativi ai biglietti delle varie categorie sembrino produrre una cosa similare quasi uguale, è necessario distinguere i token relativi alle tipologie dei biglietti per ciascuna categoria. Questo è necessario poiché nel file di input, per ciascuna tappa, possono anzi devono comparire solamente quattro categorie di biglietto, in ordine dalla A alla D. Per analizzare sintatticamente questo ordine, è necessario tenere separati i token delle tipologie per ciascuna categoria e anche i simboli non terminali stessi di ciascuna categoria. Se così non fosse, potremmo solamente garantire che in ciascuna tappa vi siano 4 biglietti ma perderemmo l'informazione sulla categoria. Questa informazione è cruciale per il progetto in quanto il prezzo di ciascun biglietto varia al variare della categoria e questa informazione non può essere persa.

2.4 Regole Grammaticali

La grammatica è definita come segue:

```
input: tappa prezzi sconti;

tappa: citta location dataTappa bigliettoA bigliettoB bigliettoC bigliettoD fine {
tuple<string,string,string> tappa(*$1,*$2,*$3);
stackTappe.push(tappa);
};
```

```
fine: separatoreTappa tappa | separatoreTappa;

biglettoA: ordinariA musicistiA juniorA seniorA disabiliA autoritaA {
codaOrdinari.push($1);
codaMusicisti.push($2);
codaJunior.push($3);
codaSenior.push($4);
codaDisabili.push($5);
codaAutorita.push($6);
};
biglettoB: ordinariB musicistiB juniorB seniorB disabiliB autoritaB {
codaOrdinari.push($1);
codaMusicisti.push($2);
codaJunior.push($3);
codaSenior.push($4);
codaDisabili.push($5);
codaAutorita.push($6);
};
biglettoC: ordinariC musicistiC juniorC seniorC disabiliC autoritaC {
codaOrdinari.push($1);
codaMusicisti.push($2);
codaJunior.push($3);
codaSenior.push($4);
codaDisabili.push($5);
codaAutorita.push($6);
};
biglettoD: ordinariD musicistiD juniorD seniorD disabiliD autoritaD {
codaOrdinari.push($1);
codaMusicisti.push($2);
codaJunior.push($3);
codaSenior.push($4);
codaDisabili.push($5);
codaAutorita.push($6);
};

prezzi: prezzoA prezzoB prezzoC prezzoD {
prezziBiglietti.at(A)=$1;
prezziBiglietti.at(B)=$2;
prezziBiglietti.at(C)=$3;
prezziBiglietti.at(D)=$4;
```

```
};
sconti: scontoO scontoM scontoJ scontoS scontoD scontoA {
scontiBiglietti.at(ORDINARI) = $1;
scontiBiglietti.at(MUSICISTI) = $2;
scontiBiglietti.at(JUNIOR) = $3;
scontiBiglietti.at(SENIOR) = $4;
scontiBiglietti.at(DISABILI) = $5;
scontiBiglietti.at(AUTORITA) = $6;
};
```

2.5 Verifica della Grammatica

Dato che Bison è ottimizzato per le grammatiche LALR(1), è stato necessario verificare che la grammatica progettata fosse LALR(1). Ottenuto un esito positivo, è stata effettuata un'ulteriore verifica per assicurarsi che Bison non producesse errori di conflitto né di tipo Shift-Reduce né di tipo Reduce-Reduce. La grammatica è risultata priva di conflitti, garantendo così una corretta parsificazione degli input forniti.

Capitolo 3

Analisi Lessicale

Per l'analisi lessicale, è stato progettato un lexer in grado di riconoscere vari elementi del file di input, tra cui i nomi delle città, delle location, le date, il numero di biglietti per ciascuna tipologia di categoria, i separatori delle tappe, i prezzi dei biglietti di ciascuna categoria e gli sconti applicabili a ciascuna tipologia di biglietto.

3.1 Compiti dell'Analisi Lessicale

L'analisi lessicale ha diversi compiti fondamentali:

- **Riconoscimento dei Token**: Identificare e classificare le stringhe di caratteri in token, che rappresentano le unità sintattiche di base del linguaggio.
- **Gestione degli Stati**: Utilizzare le *start conditions* per gestire il passaggio tra le diverse sezioni del file di input, garantendo che il lexer operi nel contesto corretto.
- **Eliminazione di Spazi e Commenti**: Rimuovere gli spazi bianchi e i commenti non significativi per semplificare l'analisi sintattica. Questo passaggio non è stato eseguito nel nostro caso specifico, come verrà dettagliato di seguito.
- **Gestione degli Errori**: Riconoscere e segnalare eventuali errori lessicali, come stringhe di caratteri non riconosciute o formati errati.

3.2 Gestione delle Sezioni

I marcatori che indicano la fine di una sezione e l'inizio di un'altra sono riconosciuti dal lexer per gestire il cambiamento dello stato del lexer attraverso le *start conditions*.

Per garantire il corretto funzionamento del lexer, sono state utilizzate *start conditions* di tipo esclusivo anziché inclusivo. Questa scelta è stata motivata dalla necessità di assicurare una chiara separazione tra le diverse sezioni del file di input, evitando ambiguità nell'analisi.

3.3 Token Riconosciuti

Il lexer è stato configurato per restituire un token specifico che indica la fine di una tappa e il possibile inizio di un'altra tappa. Questo è cruciale per la corretta costruzione dell'albero sintattico durante la fase di parsing. Tuttavia, i marcatori che segnalano le sezioni di fine del blocco delle tappe (`$$$`) e delle linee dei prezzi (`++++++++++++++++++++`) sono utilizzati esclusivamente per cambiare lo stato del lexer tramite le *start conditions*. Questi marcatori non sono rilevanti per il parsing e non contribuiscono alla costruzione dell'albero sintattico.

3.4 Eliminazione di Spazi e Tabulazioni

Uno dei compiti tipici dello scanner (o lexer) è quello di eliminare eventuali spazi bianchi e tabulazioni dal codice sorgente per facilitare il processo di tokenizzazione. Tuttavia, nel nostro caso specifico, questo passaggio non è stato eseguito. Gli spazi possono indicare nomi composti di città e location (ad esempio, "SAN BENEDETTO DEL TRONTO" o "Stadio San Siro"), che sono significativi per l'analisi sintattica. Pertanto, è stato necessario preservare questi spazi per garantire che i nomi composti vengano riconosciuti correttamente come singole entità durante il processo di parsing.

Questa strategia consente una gestione efficiente e logica del flusso di input, garantendo che il lexer e il parser lavorino insieme senza conflitti o ambiguità.

Capitolo 4

Analisi Sintattica

L'analisi sintattica è una fase cruciale nella compilazione di un programma, in cui l'input, sotto forma di una sequenza di token generati dall'analisi lessicale, viene organizzato in una struttura ad albero sintattico. Questo processo verifica che l'ordine e la combinazione dei token seguano le regole grammaticali definite, garantendo che il codice sorgente abbia una sintassi corretta.

4.1 Compiti dell'Analisi Sintattica

I principali compiti dell'analisi sintattica includono:

- **Verifica della Sintassi**: Assicurarsi che la sequenza di token generata dall'analisi lessicale segua la grammatica del linguaggio.
- **Costruzione dell'Albero Sintattico**: Creare una struttura ad albero che rappresenti la gerarchia delle operazioni e delle costruzioni sintattiche del codice sorgente.
- **Gestione degli Errori Sintattici**: Rilevare e segnalare eventuali errori di sintassi nel codice sorgente.
- **Preparazione per l'Analisi Semantica**: Fornire una struttura organizzata che possa essere utilizzata nella successiva fase di analisi semantica.

4.2 L'Analisi Sintattica nel Nostro Caso

Nel nostro programma, l'analisi sintattica ha il compito di organizzare e verificare le informazioni relative alle tappe della tournée, ai prezzi dei biglietti e agli sconti

applicabili. La grammatica utilizzata è di tipo LALR(1), ottimale per garantire un parsing efficiente e privo di ambiguità.

La struttura del file di input è suddivisa in tre sezioni principali: tappe, prezzi e sconti. Ogni sezione segue un formato specifico che deve essere riconosciuto e validato dal parser. Di seguito, una breve descrizione delle produzioni della grammatica:

In particolare, il parser deve riconoscere e validare:

- **Tappe del Tour**: Informazioni sulla città, location, data, e numero di biglietti per ciascuna categoria.
- **Prezzi dei Biglietti**: Prezzi per le categorie A, B, C, e D.
- **Sconti Applicabili**: Percentuali di sconto per le diverse categorie di utenti.

Grazie alla struttura definita dalla nostra grammatica, l'analisi sintattica è in grado di organizzare correttamente queste informazioni, preparandole per la successiva fase di analisi semantica e calcolo dei guadagni e delle somme da devolvere in beneficienza.

Capitolo 5

Tabella dei Simboli

La tabella dei simboli è una componente essenziale nella gestione delle informazioni durante il processo di compilazione. Essa tiene traccia delle varie entità presenti nel programma, come variabili, funzioni, e costanti, permettendo di gestire correttamente le informazioni necessarie durante le fasi di analisi sintattica e semantica.

5.1 Strutture Dati Utilizzate

Per implementare la tabella dei simboli nel nostro progetto, abbiamo fatto uso delle seguenti strutture dati:

- **Vettori**: Utilizzati per memorizzare i prezzi dei biglietti e gli sconti applicabili alle diverse categorie di biglietti. I vettori permettono un accesso rapido e diretto agli elementi, facilitando il calcolo dei totali e l'applicazione degli sconti.
- **Code**: Utilizzate per gestire i numeri dei biglietti delle varie tipologie per ciascuna categoria. Per ciascuna tipologia di biglietto (Ordinari, Musicisti, Junior, Senior, Disabili, Autorità), è stata creata una coda specifica. Questa scelta consente di incolonnare, in base all'ordine di sviluppo del parser, i numeri dei biglietti per ciascuna tipologia di ciascuna categoria per ogni tappa.
- **Tuple**: Utilizzate per rappresentare ciascuna tappa del tour. Ogni tupla contiene le informazioni relative alla città, alla location e alla data della tappa.
- **Stack di Tuple**: Utilizzato per memorizzare tutte le tappe. Data la natura bottom-up dell'analisi sintattica del parser, le tappe sono lette in ordine inverso rispetto a come compaiono nel file di input. Lo stack permette di estrarre le tappe con tecnica LIFO (Last-In, First-Out), garantendo così un ordine di elaborazione che rispecchia quello del file di input.

5.2 Motivazioni delle Scelte delle Strutture Dati

La combinazione delle strutture dati sopra descritte è stata scelta per garantire un'efficiente gestione e corretta sequenza delle operazioni necessarie per il nostro programma.

- **Code per i Biglietti**: L'utilizzo delle code per i numeri dei biglietti di ciascuna tipologia di ciascuna categoria è stato determinante. Dato che il parser legge i biglietti in sequenza come appaiono nel file di input, l'estrazione FIFO (First-In, First-Out) delle code assicura che i biglietti siano processati nell'ordine corretto.
- **Stack per le Tappe**: L'analisi sintattica bottom-up implica che le tappe siano lette in ordine inverso. Utilizzando uno stack, è possibile estrarre le tappe in ordine LIFO, ripristinando l'ordine originale del file di input. Questo è fondamentale non solo per l'ordine estetico nel file di output, ma soprattutto per mantenere la corretta corrispondenza tra tappe e numeri dei biglietti di ciascuna tipologia di ciascuna categoria.

5.3 Procedura di Calcolo del Totale delle Vendite

Per calcolare il totale delle vendite per ciascuna tappa, è necessario estrarre una tappa dallo stack e, contemporaneamente, estrarre i numeri dei biglietti delle varie tipologie dalle code corrispondenti. Questo approccio assicura che ogni tappa venga associata correttamente ai relativi biglietti, permettendo di calcolare accuratamente il guadagno totale e le somme da devolvere in beneficienza. La combinazione delle tecniche LIFO per le tappe e FIFO per i biglietti si è dimostrata efficace per soddisfare i requisiti del progetto.

In sintesi, la scelta di utilizzare vettori, code, tuple e stack è stata strategica per gestire in modo efficiente e corretto le informazioni richieste dal progetto, garantendo la coerenza e l'accuratezza dei dati elaborati.

Capitolo 6

Analisi Semantica

L'analisi semantica è una fase cruciale nel processo di compilazione, il cui scopo principale è attribuire un significato alle strutture sintattiche riconosciute durante l'analisi sintattica. Questa fase verifica la correttezza semantica del programma, assicurando che le operazioni compiute abbiano senso nel contesto del linguaggio.

Nel nostro caso specifico, l'analisi semantica ha il compito di interpretare correttamente sia le stringhe sia i numeri estratti dal file di input. In particolare, l'analisi semantica si occupa di:

- **Interpretazione delle Stringhe**: Le stringhe relative ai nomi delle città, alle location e alle date delle tappe devono essere interpretate correttamente per essere utilizzate nella generazione dell'output. Questo implica verificare che le stringhe corrispondano ai formati attesi e assegnare loro i significati corretti nel contesto del programma.
- **Interpretazione dei Numeri**: I numeri relativi al numero di biglietti venduti per ciascuna tipologia di ciascuna categoria, i prezzi dei biglietti e gli sconti applicabili devono essere correttamente interpretati per calcolare i totali delle vendite e della beneficienza per ciascuna tappa.

L'analisi semantica nel nostro progetto è fondamentale per eseguire correttamente le seguenti operazioni:

- **Calcolo del Totale delle Vendite**: Per ciascuna tappa, viene calcolato il totale delle vendite dei biglietti, tenendo conto dei prezzi e degli sconti applicabili. Questo richiede che i numeri relativi ai biglietti venduti e ai prezzi siano interpretati e utilizzati correttamente nelle operazioni aritmetiche.
- **Calcolo della Beneficienza**: Per ciascuna tappa, viene calcolato l'importo da devolvere in beneficienza, in base al totale delle vendite. Le regole per la

beneficienza variano a seconda del guadagno totale, richiedendo un'accurata interpretazione dei numeri e l'applicazione delle condizioni specificate.

In sintesi, l'analisi semantica nel nostro progetto si focalizza sulla corretta interpretazione delle stringhe e dei numeri per garantire il calcolo preciso delle vendite e della beneficienza. Questa fase assicura che le informazioni estratte durante l'analisi sintattica siano utilizzate in modo significativo e corretto, rispettando le regole e i requisiti specificati nel progetto.

Capitolo 7

Conclusioni

Il progetto ha raggiunto il suo obiettivo principale: sviluppare un programma in grado di supportare il responsabile delle tappe della tournée di un famoso gruppo musicale nella gestione dei biglietti venduti e nel calcolo delle somme da devolvere in beneficenza. Utilizzando Flex e Bison, sono state implementate le fasi di analisi lessicale e sintattica, garantendo la corretta interpretazione dei dati di input e la generazione di un output accurato e strutturato.

La scelta delle strutture dati e l'organizzazione del parser hanno assicurato l'efficienza e l'accuratezza del programma, permettendo di processare correttamente le informazioni relative a ciascuna tappa del tour. La procedura di calcolo dei guadagni e della beneficenza ha fornito risultati precisi e affidabili, soddisfacendo i requisiti specificati.

In sintesi, il programma realizzato rappresenta uno strumento efficace per la gestione delle vendite di biglietti e la determinazione delle somme destinate alla beneficenza, contribuendo al successo della tournée del gruppo musicale.