Progetto di Compilatori

“Corsa ciclistica”

6 Giugno 2024

Andrea Fazio 0729956
Giuseppe Rosa 0730875

Commissionato dalla professoressa: Sabrina Mantaci

Università degli Studi di Palermo

Sommario

Consegna

Traccia

Si costruisca, utilizzando la coppia di programmi FLEX e BISON, un traduttore guidato dalla sintassi in grado di riconoscere un linguaggio che descrive i risultati e le classifiche di una corsa ciclistica a tappe e che sia in grado di gestire le classifiche parziali e generali. Il file di ingresso è suddiviso in tre sezioni separate, nell'ordine, dalle stringhe "%%" e "$$".

La prima sezione contiene il giorno in cui viene calcolata la classifica. Il giorno

è espresso nel formato Data: *GG-MM-AA*.

Nella seconda parte è elencata la lista dei ciclisti partecipanti. Questi ciclisti vengono rappresentati mediante il loro codice di iscrizione alla federazione, il loro nome e cognome, la squadra e il numero di pettorale con cui è iscritto alla gara. Il formato della sezione è costituito da blocchi del seguente tipo:

codice: *codice_atleta*
Nome Cognome: *Nome_e_cognome_atleta*
Squadra *Squadra_atleta*
Pettorale:*n_pettorale*
&&&

dove:

- il *codice_atleta* è formato da due caratteri alfabetici maiuscoli e quattro cifre;
- *Nome_e_cognome atleta* iniziano per lettera maiuscola e ha le altre lettere minuscole. Sono separati da uno spazio e sia il nome che il cognome possono essere formati da più parole;
- La *Squadra_atleta* è una frase di più parole di lettere minuscole separate da spazi;
- Il *n_pettorale* è un intero tra 1 e 999;
- Le informazioni di ogni atleta sono separate da quelle dell'atleta successivo con i simboli &&&

Nella terza parte vengono rappresentate le varie tappe, con:

data > città_partenza > città arrivo > classifica.

Dove:

- la *data* è nel formato GG/MM.
- *Città partenza* (*citta arrivo*) è costituita da una o più parole che iniziano per lettera maiuscola e che hanno gli altri caratteri minuscoli.
- Una *classifica* è una lista di ciclisti, ordinata per ordine di arrivo alla tappa, in cui ogni elemento è rappresentato come segue:

  (pettorale, tempo tappa)

  Con tempo tappa nel formato hh:mm:ss

Si richiede di fornire in output:

1. La stampa della classifica relativa a quella tappa (oss: conviene mantenere la lista ordinata)
2. La stampa della classifica parziale fino a quella tappa

La classifica parziale (così come quella finale) viene così calcolata: si sommano i tempi di ogni ciclista nelle singole tappe effettuate (ovviamente somma modulo 60), ma attribuisce in ogni tappa un bonus di -60 secondi al 1° classificato, -30 secondi al 2° classificato, -10 secondi al 3° classificato. Le classifiche sono stampate nel formato

Nome cognome *tempo*

Dove tra Nome Cognome e Tempo c'è un tab

Esempio input

```
Data: 23-05-24
%%
Codice: PQ3274
Cognome e nome: Giovanni Amato
Squadra: libertas ct
Pettorale: 215
&&&
Codice: PM4285
Cognome e nome: Pietro Magri
Squadra: ASD Ciclismo PA
Pettorale: 357
&&&
Codice: AP8242
Cognome e nome: Marco Bonanno
Squadra: libertas ct
Pettorale: 818
&&&
Codice: AZ6813
Cognome e nome: Massimo Bonelli
Squadra: palermo cicismo
Pettorale: 203
&&&
Codice: BY9532
Cognome e nome: Antonio Cusimano
Squadra: palermo ciclismo
Pettorale: 613
&&&
Codice: CA4794
Cognome e nome: Paolo Di Blasi
Squadra: sporting club
Pettorale: 418
&&&
Codice: PN1137
Cognome e nome: Manfredi Terranova
Squadra: asd cefau
Pettorale: 206
&&&
$$
22/05 - Palermo - Cefalu - (203: 2:42:30) -> (215: 2:42:53) -> (206:
2:43:15) -> (613: 2:43:16) -> (418:2:43:21) -> (357: 2:44:02) -> (818:
2:44:50)
23/05 - Cefalu- Messina - (215: 3:55:34) -> (613: 3:55:35) -> (206:
3:55:50) -> (418: 3:55:52) -> (203:3:56:48) -> (818: 3:57:10) -> (357:
3:58:30)
```

Esempio output

```
 CLASSIFICA TAPPA 23 Maggio
    1. Giovanni Amato              3:55:34
    2. Antonio Cusimano            3:55:35
    3. Manfredi Terranova          3:55:50
    4. Paolo Di Blasi              3:55:52
    5. Massimo Bonelli             3:56:48
    6. Marco Bonanno               3:57:10
    7. Pietro Magri                3:58:30


CLASSIFICA PARZIALE
    1. Giovanni Amato              6:36:57
    2. Massimo Bonelli             6:38:18
    3. Antonio Cusimano            6:38:21
    5. Manfredi Terranova          6.38:45
    4. Paolo Di Blasi              6:39:13
    6. Marco Bonanno               6:42:00
    7. Pietro Magri                6:42:32
```

Soluzione proposta

Analisi lessicale

L'analizzatore lessicale, realizzato in Flex, ha il compito di riconoscere i lessemi tramite le espressioni regolari, e passare il token corrispondente all'analizzatore sintattico.

L'analisi lessicale è partita dalla definizione dei lessemi e dei token:

DESCRIZIONE LESSEMA	TOKEN	REGEX
Prefisso data	DATA_PREFIX	"Data: "
Valore della data	DATA	(0[1-9]|[12][0-9]|3[01])"-"(0[1-9]|1[0-2])"-"[0-9]{2}
Prefisso del codice ciclista	CODICE_PREFIX	"Codice: "
Valore del codice ciclista	CODICE	[a-zA-Z]{2}[0-9]{4}
Prefisso del cognome e nome del ciclista	NOME_PREFIX	"Cognome e nome: "
Cognome e nome del ciclista	NOME	[A-Z][a-z]+([ ][A-Z][a-z]+)*
Prefisso del nome squadra del ciclista	SQUADRA_PREFIX	"Squadra: "
Nome squadra del ciclista	SQUADRA	[a-zA-Z]+
Il prefisso numero pettorale del ciclista	PETTORALE_PREFIX	"Pettorale: "
Numero del pettorale del cliclista	NUMERO_PETTORALE	[0-9]{1,3}
Separatore dei ciclisti	SEPARATORE_CICLISTI	"&&&"
Separatore sezione 1	SEPARATORE_SEZIONE_1	"%%"
Separatore sezione 2	SEPARATORE_SEZIONE_2	"$$"
Data della tappa	DATA_TAPPA	[0-9]{2}"/"[0-9]{2}

Trattino che separa le sezioni delle tappe	TRATTINO	"-"
Città della tappa	CITTA	[A-Z][a-z]+
Parentesi SX	PARENTESI_SX	"("
Parentesi DX	PARENTESI_DX	")"
Seperatore del pettorale e tempo nella tappa	SEP_PETT_TEMPO	“:”|”,”
Tempo della tappa di un atleta	TEMPO	[0-9]{1,2}":"[0-9]{2}”:”[0-9]{2}
Freccia	FRECCIA	"->"

Sono anche presenti regole per ignorare gli spazi vuoti, i tab e le newline

Abbiamo usato delle start condition in Flex per permettere una migliore suddivisione dell’analizzatore, nonché per evitare conflitti di match (ad esempio Città e Nome Squadra).

Analisi sintattica

L'analizzatore sintattico (parser), realizzato in Bison, contiene le regole della grammatica, che fanno uso dei token, e include i file di intestazione usati per la tabella dei simboli.

Definiamo una union per gestire i tipi associati ai token, che possono essere stringhe o interi (nel caso del numero di pettorale).

1. L'assioma è dato dalla regola

```
start:
    intestazione SEPARATORE_SEZIONE_1 ciclisti SEPARATORE_SEZIONE_2 tappe
```

2. L'intestazione si occupa di riconoscere la data e richiamare la funzione `parseData()` per processarla e sapere la data voluta per la stampa della classifica.

```
intestazione:
    DATA_PREFIX DATA { parseData($2); }
```

3. La regola ciclisti permette di riconoscere uno o più ciclisti. Ogni ciclista contiene i prefissi e i dati necessari al riconoscimento, che poi elaboriamo tramite `insert()` per l'inserimento nella tabella dei simboli.

```
ciclisti: ciclista | ciclisti ciclista

ciclista:
    CODICE_PREFIX CODICE NOME_PREFIX NOME SQUADRA_PREFIX nome_o_nome_squadra
    PETTORALE_PREFIX NUMERO_PETTORALE SEPARATORE_CICLISTI
    { insert($8, $4); }
```

4. La regola tappe riconosce una o più tappe. Ogni tappa ha una data, due città, i relativi separatori e una serie di risultati.

```
tappe: tappa | tappe tappa

tappa: DATA_TAPPA TRATTINO CITTA TRATTINO CITTA TRATTINO risultati_tappa
```

5. La regola risultati_tappa permette di leggere più risultati relativi alla stessa tappa. Ogni volta che leggiamo l'ultimo risultato di una tappa resettiamo il contatore della posizione. Per ogni risultato, aggiorniamo il contatore della posizione e aggiungiamo il tempo chiamando la funzione `aggiungi_tempo()`

```
risultati_tappa: risultato {posizione_attuale = 0;}
               | risultato FRECCIA risultati_tappa

risultato:
    PARENTESI_SX NUMERO_PETTORALE SEP_PETT_TEMPO TEMPO PARENTESI_DX {
        posizione_attuale++;
        aggiungi_tempo($2, $4, posizione_attuale);
    }
```

6. Infine, nel `main()`, se `yyparse()` va a buon fine, chiamiamo le funzione di stampa della classifica del giorno e di quella parziale.

Strutture usate per la tabella dei simboli

Abbiamo deciso di implementare la tabella dei simboli usando una lista concatenata, questo per permetterci di tenere in ordine i simboli (i ciclisti). Questa è definita nel file symbol_table.h, incluso dal parser.

```
typedef struct Ciclista {
    int numero_pettorale;
    char *nome;
    int tempo_totale;
    int tempo_tappa;

    struct Ciclista *next;
} Ciclista;

void insert(int numero_pettorale, char *nome);
Ciclista *find(int numero_pettorale);
void aggiungi_tempo(int numero_pettorale, char *tempo, unsigned int posizione);
```

Nel file symbol_table.c sono inoltre presenti tutte le funzioni che usiamo per gestire i dati in ingresso dal parser, e andare a stampare le classifiche necessarie.

Compilazione e Test

Sono presenti 5 file di test:

- input.txt: contiene l'input corretto che l'analizzatore aspetta di ricevere
- errore1.txt: contiene un **errore lessicale** sul nome del ciclista
- errore2.txt: contiene un **errore lessicale** sulla città della prima tappa
- errore3.txt: contiene un **errore sintattico**: nome squadra al posto di nome e cognome
- errore4.txt: contiene un **errore semantico**: nella classifica è presente un numero di pettorale sconosciuto