Relazione Compilatori – Progetto “Banca”

L’obiettivo del progetto “Banca” è realizzare un’applicazione, con l’ausilio di Flex e Bison, capace di aiutare l’impiegato di una banca analizzare le transazioni monetarie dei clienti ed individuare quelle “sospette”.

Descrizione del programma

L’applicazione “Banca” è realizzata utilizzando Flex e Bison, generatori automatici di analizzatori lessicali e sintattici rispettivamente; lo scopo del programma è analizzare le transazioni monetarie di una banca e aiutare gli impiegati ad individuare quelle identificate come “sospette”, ed eventualmente associare ad esse l’identità di un cliente della banca stessa. Definiamo un insieme di transazioni come “sospette” se nel mese in analisi esse ammontano ad almeno 8000€ sia entrata e/o in uscita, ovvero uno stesso cliente può essere segnalato sia per attivi che per passivi monetari.

Si fornisce al programma un file di testo in input contenente la data del mese e anno in analisi; una sezione con il codice fiscale delle persone che hanno effettuato transazioni bancarie in tale data, con specificati il giorno e l’ammontare della transazione, che può essere sia in entrata che in uscita; infine, una sezione finale che contiene l’elenco dei clienti della banca e informazioni sulla loro identità.

In caso di corretto formato del file in input, il programma restituisce in output la data del file in analisi, seguita da un messaggio che informa della presenza o meno di transazioni sospette, e l’elenco di tali transazioni associate alle credenziali dei clienti che hanno effettuato i movimenti, presentati all’utente in ordine alfabetico.
Se le transazioni sospette non appartengono a un cliente della banca, esse non vengono inserite in elenco, ma viene segnalato con un ulteriore messaggio il numero di clienti sospetti non identificati.

Input del programma

Ogni file fornito come input al programma deve rispettare un formato particolare. Individuiamo tre sezioni fondamentali: *data in analisi*, *sezione delle transazioni* e *sezione dei clienti*. Le tre sezioni sono inoltre divise da separatori simbolici.

Data in analisi

La prima sezione contiene l’indicazione sul mese che si sta prendendo in considerazione nel formato “mm/aaaa” come nell’esempio seguente:

```
DATA: 01/2015
```

La sezione della data e quella delle transazioni sono divise tra loro dal separatore simbolico “!!”.

Sezione delle transazioni

La sezione delle transazioni contiene il codice fiscale e le singole transazioni che un cliente ha effettuato nell’arco del mese considerato, nel formato seguente:

```
RSSMRA60B01F205S
12: + 3000 ;
21: - 1000 ;
19: + 4000 ;
08: + 3000 ;
%%
```

Il separatore "%%" individua le sottosezioni riguardanti le transazioni di uno stesso cliente. Inoltre, la sezione delle transazioni può essere plausibilmente vuota (nessuna transazione avvenuta nel mese corrente). Il divisore tra questa sezione e quella dei clienti è il separatore "??".

Sezione dei clienti

L'ultima sezione contiene un elenco dei clienti affiliati alla banca, con nome e cognome, e codice fiscale del cliente; il formato dati è il seguente:

```
Rossi Mario, RSSMRA60B01F205S ;
```

Questa sezione non può essere vuota (che banca sarebbe senza clienti?).

In tutte e tre le precedenti sezioni non è previsto alcun controllo sulla coerenza e consistenza delle informazioni, ad esempio la validità di una data o la corrispondenza tra nomi e codici fiscali, purché tali informazioni rispecchino il formato richiesto (data in formato "mm/aaaa", codice fiscale come stringa alfanumerica maiuscola e di lunghezza 16 simboli).

Analizzatore lessicale

L'analizzatore lessicale del programma "Banca" è realizzato in Flex, e prevede la tokenizzazione dei seguenti elementi:

Regex	Token
"DATA:"	DATE
[0-9]{2}[/][0-9]{4}	**DATESTR**
"!!"	SEP1
[A-Z0-9]{16}	**CF**
[0-9][0-9][:]	DAY
"+"	+
"-"	-
0|[1-9][0-9]*	**NUMVAL**
";"	;
"%%"	SEP_CM
"??"	SEP2
[a-zA-Z]+/,	**CTM_NAME**
","	,
[\t\n]+	-

Ad ogni espressione regolare corrisponde un token, ed un attributo addizionale per alcuni di questi.

- Il token DATE segnala la presenza della data, la cui stringa è effettivamente contenuta nel token seguente DATESTR.
- Dopo il token SEP1 iniziano le informazioni sulle transazioni; ogni agente è identificato tramite il codice fiscale CF, e ogni sua transazione è identificata da un giorno del mese

DAY, il tipo di transazione (in entrata '+', in uscita '-') e il suo ammontare NUMVAL; le transazioni di ogni cliente sono separate tra loro da ';'; le sottosezioni di ogni cliente sono terminate da SEP_CM.

- Il token SEP2 dà inizio all'ultima sezione, in cui ogni cliente è individuato da un nome CTM_NAME, separato dal proprio codice fiscale dal token ','; in questo caso i vari clienti sono separati tra di loro dal token ';'.

L'analizzatore lessicale inoltre può individuare altri pattern descritti da altre due espressioni regolari: [\t\n]+, che serve ad individuare i caratteri *blank* ed ignorarli, e la regola generica da applicare in caso non si trovi una corrispondenza con le espressioni regolari illustrate in precedenza, la quale prevede l'esecuzione del comando yyterminate() che immediatamente arresta l'analisi lessicale, ed eventualmente portando l'analizzatore sintattico a mostrare un messaggio di errore come previsto in caso di cattivo formato dell'input.

Per esempio, con un file input ben formato con una sola transazione e un solo cliente memorizzato, la lista di token che l'analizzatore lessicale genera potrebbe essere la seguente:

DATE **DATESTR** SEP1 **CF** DAY + **NUMVAL** **;** SEP_CM SEP2 **CTM_NAME** **,** **CF** **;**

Ogni token è separato solo da uno spazio, e i token in grassetto possiedono un attributo passato all'analizzatore sintattico tramite la variabile di Flex yylval.

Analizzatore sintattico

L'analizzatore sintattico del programma "Banca" è realizzato in Bison, e prevede il riconoscimento di una struttura grammaticale con regole prestabilite a partire dai token forniti dall'analizzatore lessicale.
Avendo necessità di trattare sia stringhe che numeri interi, definiamo un tipo di dato *union* come puntatore a caratteri char *string e intero a 32 bit int value; definiamo quindi i tipi di token in lettura:

Token	Tipo di attributo
DATE	-
DATESTR	<string>
SEP1	-
CF	<string>
DAY	-
+	-
-	-
NUMVAL	<value>
;	-
SEP_CM	-
SEP2	-
CTM_NAME	<string>
,	-
-	-

Specifichiamo che vogliamo visualizzare un messaggio d'errore da noi definito con la direttiva %error-verbose e che l'analisi sintattica deve iniziare con il simbolo non terminale Input:

```
Input: Date SEP1 Sect1 SEP2 Sect2 ;
```

La regola iniziale ha la stessa struttura del file in input: sezione della data, sezione delle transazioni, e sezione dei clienti, con i relativi separatori. Il non terminale `Date` si trasforma nei terminali `DATE` e `DATESTR` con le informazioni sulla data.
La prima sezione `Sect1` può essere vuota, in caso contrario contiene una lista di clienti:

```
Sect1: /*no transactions*/ | CtmList ;
```

La lista CtmList conterrà una o più liste sulle transazioni di un singolo cliente:

```
CtmList: CtmMovs | CtmList CtmMovs ;
```

`CtmMovs` contiene l'identificativo di un singolo cliente e la lista di movimenti bancari effettuati:

```
CtmMovs: CF MovList SEP_CM ;
```

La lista dei movimenti di ogni cliente contiene almeno un elemento; ogni transazione viene individuata come un valore e un segno che indica se il denaro è in entrata o in uscita dal conto:

```
MovList: Mov | MovList Mov ;

Mov: DAY '+' NUMVAL ';' | DAY '-' NUMVAL ';' ;
```

La seconda sezione `Sect2` contiene le informazioni di almeno un cliente:

```
Sect2: Customer | Sect2 Customer ;

Customer: CTM_NAME ',' CF ';' ;
```

In corrispondenza di una struttura sintattica valida, verranno intraprese le azioni semantiche corrispondenti alle varie produzioni della grammatica.

Azioni semantiche e tabella dei simboli

Nella sezione delle definizioni dell'analizzatore sintattico abbiamo dichiarato alcune variabili che utilizziamo per gestire i dati di nostro interesse.
Il puntatore a caratteri `*input_date` viene utilizzato per tener traccia della data del file in input che stiamo analizzando. Utilizziamo gli interi `c` e `k` come contatori per misurare rispettivamente il numero di clienti che effettuano transazioni, e il numero di clienti registrati e di cui si conoscono le credenziali. Gli interi `tmp_sum` e `tmp_sub` sono registri temporanei in cui memorizziamo per ogni cliente l'ammontare delle transazioni in entrata e uscita, in modo da analizzarle separatamente.
Inoltre definiamo la costante `MAX` che indica la dimensione di alcuni array che utilizziamo per memorizzare le informazioni di nostro interesse.
Il vettore `*cf_list[]` viene utilizzato per memorizzare il codice fiscale di un cliente che effettua transazioni sospette, le quali vengono memorizzate in `*sum_list[]` (transazioni in entrata) e in `*sub_list[]` (transazioni in uscita). I vettori `*ctm_name[]` e `*ctm_cfs[]` vengono utilizzati per salvare le informazioni dei clienti registrati.

La scelta di utilizzare degli array statici di dimensione prefissata può essere vincolante, poiché è possibile gestire fino a `MAX` clienti che effettuano transazioni e/o fino a `MAX` clienti registrati, ma l'utilizzo di tali strutture dati è sia semplice che sufficiente per lo scopo che il progetto si propone.

Analizziamo adesso in maniera ascendente le azioni semantiche intraprese dall'analizzatore sintattico.

Data in analisi
Al simbolo non terminale `Date` è associato un semplice assegnamento di variabile per memorizzare la data (più correttamente, l'indirizzo di memoria della sua stringa) in `*input_date`.

Informazioni sulle transazioni
Le azioni semantiche di `Mov` vengono applicate per memorizzare temporaneamente l'ammontare delle transazioni in entrata e in uscita di ogni cliente presente nella prima sezione. Il valore numerico è sempre positivo, ma si discrimina il tipo di transizione in base al segno che precede il numero nel file.
Una volta letti tutti i movimenti di un cliente, le azioni riferite a `CtmMovs` controllano se almeno uno dei registri `tmp_sum` e `tmp_sub` risulta contenere quote "sospette", provvedendo ad aggiungere il codice fiscale del cliente in `*cf_list[]` e le somme sospette nei relativi registri. Utilizziamo la variabile `flag` per verificare che il cliente sia sospetto, incrementando così il contatore `c` dei clienti sospetti. Se le transazioni totali non eccedono il limite previsto, il cliente non viene memorizzato nella lista, analogamente per i suoi movimenti, e il contatore non viene incrementato. Inoltre azzeriamo i registri `tmp_sum` e `tmp_sub` per analizzare le transazioni dei clienti successivi.

Informazioni sui clienti registrati
Le azioni semantiche del non terminale `Customer` si limitano ad aggiungere agli array `*ctm_names[]` e `*ctm_cfs[]` i nomi e i codici fiscali dei clienti presenti nella seconda sezione del file, tenendo traccia del loro numero grazie al contatore `k`.

Ordinare le informazioni sulle transazioni
Una volta raccolti i dati sulle transazioni, le azioni legate al simbolo iniziale `Input` si occupano di ordinare i codici fiscali dell'array `*cf_list[]` in ordine alfabetico tramite un algoritmo di ordinamento di tipo Bubble Sort; durante l'ordinamento dei codici fiscali vengono aggiornate anche le liste contenenti le somme delle transazioni sospette, in modo da non perdere il riferimento posizionale tra codici fiscali e quote monetarie.

Programma principale e output

Nella sezione delle definizioni abbiamo definito la funzione `find_name_index()` che accetta come argomento una stringa (puntatore a caratteri) che rappresenta un codice fiscale, e restituisce l'indice in cui essa è presente all'interno dell'array `*ctm_cfs[]`; tale funzione risulterà utile per verificare se un cliente di cui conosciamo il codice fiscale è registrato presso la banca.

Il corpo principale dell'applicazione "Banca" contiene solo le istruzioni necessarie a visualizzare l'output richiesto, le quali vengono eseguite solo in caso di analisi lessicale e sintattica avvenute con successo, altrimenti apparirà un messaggio di errore.
Dopo aver stampato la data presente nel file in input, il programma stamperà un messaggio per segnalare la presenza o meno di transazioni sospette, verificando il valore assunto dalla variabile globale `c`; in presenza di clienti sospetti, ne stamperà le informazioni se avverrà corrispondenza tra il codice fiscale presente in `*cf_list[]` e uno tra i codici fiscali dell'array `*ctm_cfs[]`, in caso contrario incrementerà la variabile `nd` che tiene traccia del numero dei clienti sospetti non identificati, per poi stampare tale valore a schermo con un messaggio aggiuntivo.

File di esempio

Si allegano i seguenti file di esempio da sottoporre all'applicazione "Banca" con descrizione dell'esito di lettura:

- "input.txt" – Input di esempio fornito con la consegna del progetto;
- "input2.txt" – Variante con cliente doppiamente sospetto;
- "input3.txt" – Variante con cliente doppiamente sospetto e cliente non registrato;
- "input4.txt" – Variante senza transazioni;
- "input5.txt" – Variante con transazioni regolari;
- "input6.txt" – Variante con input in formato errato (assenza di separatore "%%");
- "input7.txt" – Variante con dati lessicali non consistenti ma validi.