Università degli Studi di Bologna - Facoltà di Ingegneria
Esame di Stato per l'abilitazione alla professione di Ingegnere
15 Maggio 2001
Tema di Informatica N. 4

Un'applicazione di generazione automatica di script produce file comandi (un comando per riga) per un terminale di raccolta dati. Il linguaggio script è un semplicissimo linguaggio che gestisce solo dati di tipo carattere (minuscolo) e che conta sei comandi (identificati univocamente da stringhe di 3 caratteri): un comando di lettura da tastiera `INK`, un comando di scrittura su terminale `OUT`, un comando di salto `SUB`, un comando label `LAB`, un comando `RET` di ritorno, e un comando `END` di fine programma. Il linguaggio può gestire variabili di tipo carattere o costanti carattere. Le variabili sono identificate da un singolo carattere maiuscolo, mentre le costanti sono un singolo carattere minuscolo o numerico. Ad esempio X, V, A, B sono nomi di variabili, mentre c, a, x, v, 1, 5 sono costanti.

Tutti i comandi tranne `RET` e `END` ammettono un parametro:
`INK Parametro` legge da tastiera un carattere e lo memorizza nel `Parametro` (variabile)
`OUT Parametro` scrive a terminale il carattere `Parametro` (variabile o costante)
`SUB Label` salta ad eseguire la riga identificata dalla label `Label` (costante)
`LAB Label` questo comando non effettua azioni ma semplicemente etichetta con la label `Label` (costante) la riga del programma in cui compare.

Formalmente la grammatica del linguaggio è la seguente

```
<riga-file> ::=   <comando>
<comando>   ::=   INK <nome-variabile>| OUT <nome-variabile>| OUT <costante> |
                  SUB <costante> | LAB <costante> | RET | END
<nome-variabile> ::= A | B | ... | Z
<costante>  ::= a | b | ... | z | 0 | 1 | ... | 9
```

L'esecuzione dello script deve essere sequenziale (i comandi vengono eseguiti uno dopo l'altro nell'ordine in cui compaiono nel file) tranne nel caso in cui si incontrino comandi di salto che modificano tale schema sequenziale. Vediamo come si comportano i vari comandi: `INK Parametro` e `OUT Parametro` effettuano rispettivamente la lettura da testiera e la scrittura su terminale poi il programma prosegue eseguendo l'istruzione collocata nella riga successiva. Il comando `LAB Label` non effettua azioni, ma semplicemente etichetta la riga in cui si trova con una label, poi il programma prosegue eseguendo l'istruzione collocata nella riga successiva. Il comando `SUB Label` non effettua azioni ma fa sì che il programma prosegua eseguendo come istruzione successiva quella collocata alla riga etichettata dalla label `Label` A questo punto l'esecuzione torna ad essere sequenziale, fino a quando o si incontra il comando `END` (fine programma), oppure si incontra il comando `RET`. `RET` non effettua azioni ma fa sì che il programma prosegua eseguendo l'istruzione successiva all'ultima `SUB` chiamata. Quindi una `SUB` equivale a una sorta di "chiamata di procedura". Il codice della procedura è compreso tra la label (parametro della `SUB`) e il `RET` che indica la fine della procedura. Se si incontra un comando `RET` quando non è attiva (non ancora terminata) nessuna procedura, il comando non ha effetto e si procede sequenzialmente.
Esempio 1: vediamo un semplice programma con la corrispondente spiegazione

```
INK A
LAB 1
OUT c
OUT i
OUT a
OUT o
RET
INK B
SUB 1
OUT f
OUT i
OUT n
OUT e
END
```

Viene eseguita la prima lettura di un carattere da tastiera, che viene memorizzato nella variabile A. Viene poi etichettata la seconda riga con la label '1', vengono eseguite 4 `OUT` che scrivono i 4 caratteri 'c' 'i' 'a' 'o' a terminale. Poi viene eseguita la `RET` che non ha effetti perché non è stata chiamata nessuna `SUB` prima quindi nessuna procedura è "attiva".

Poi viene eseguita una `INK` che legge un carattere da tastiera e lo memorizza nella variabile B e una `SUB` che effettua un salto alla seconda riga (etichettata dalla label '1'. (A questo punto la "procedura" corrispondente alla `SUB` è attiva e lo sarà fino a che non si incontrerà il comando `RET` corrispondente.) Quindi, vengono nuovamente eseguite 4 out che scrivono i 4 caratteri 'c' 'i' 'a' 'o' a terminale. Poi si incontra il comando `RET` che mette fine all'attività della procedura invocata dalla `SUB` e l'esecuzione continua dall'istruzione dopo la `SUB`. Le 4 `OUT` finali scrivono i 4 caratteri 'f' 'i' 'n' 'e' a terminale e il programma si conclude con il comando `END`.

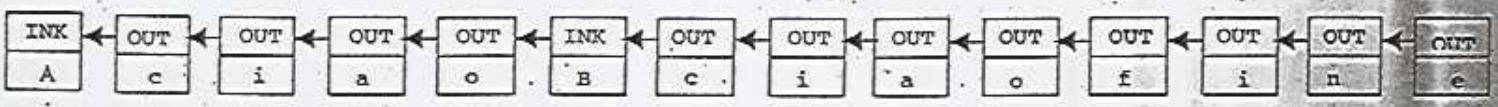
Infine, si noti che è possibile avere più "procedure" annidate.
Esempio 2

```
SUB 1
...
LAB 1
SUB 2
RET
LAB 2
...
RET
```

Si noti che all'interno del "codice" di SUB 1 c'è una chiamata a una seconda procedura.

Si progetti e realizzi, in un qualunque linguaggio di programmazione di alto livello, un sistema con le seguenti componenti

1) Componente in grado di controllare che la sintassi dei comandi sia corretta
   Tale controllore deve basarsi sulla grammatica del linguaggio, accedere al file testo contenente lo script con primitive da definire di accesso al file, e deve fornire in uscita messaggi d'errore nel caso di sintassi scorretta, e una coda di strutture comando (struttura dati di tipo FIFO) in caso di sintassi corretta.
   Ogni elemento della coda è costituito da due campi: il comando e il parametro se esiste. Nella coda devono comparire, nell'ordine in cui verranno eseguiti, i comandi presenti nello script che effettuano azioni di I/O. Si noti che la costruzione di tale coda ordinata prima dell'esecuzione è possibile non essendo presenti operatori di salto condizionato. Si definiscano inoltre le primitive necessarie per l'uso e la manipolazione della coda. Il programma dell'Esempio 1 si traduce nella seguente struttura a coda



NOTA: per svolgere il punto 1) si faccia l'ipotesi semplificativa che il programma non presenti cicli infiniti.

2) Controllore di terminazione che garantisca che il programma non entri in loop. Un possibile caso di loop è il seguente

```
LAB 10
INK A
OUT b
SUB 10
```

Ossia tra `LAB` e `SUB` non c'è l'istruzione `RET`.

Si descrivano anche altri casi (più complessi) di loop e un criterio generale per identificarli.
SUGGERIMENTO: si consideri cosa accade in casi di "mutua ricorsione"

3) Generatore di codice che, a partire dalla coda, produca un file contenente la traduzione in codice C (o qualunque linguaggio di alto livello) del file script. Si precisino le primitive usate per accedere alla coda, e le primitive per generare il codice in linguaggio C. Per affrontare questo punto si definiscano le corrispondenze tra le istruzioni del linguaggio script e le funzioni del linguaggio destinazione. Inoltre, se si utilizza un linguaggio imperativo in cui sono necessarie le definizioni delle variabili a inizio programma, si tenga conto che le variabili usate sono solo di tipo carattere.

Si organizzi il progetto in modo modulare, si motivino le scelte progettuali effettuate e si discutano possibili ottimizzazioni da apportare all'architettura per migliorare l'efficienza del procedimento di traduzione.