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

Si progettino e realizzino alcune componenti di un compilatore per un semplice linguaggio di programmazione che tratta due tipi di dati (i numeri interi e i numeri reali) che possono essere sia costanti sia variabili con tipo. Il linguaggio ha come unica istruzione l'assegnamento.
I compilatori sono formati da quattro componenti:

1. un analizzatore lessicale (*scanner* o *lexer*) che prende in ingresso il programma (inteso come stringa di simboli), estrae le parole del linguaggio e le classifica in una delle categorie lessicali del linguaggio;
2. un analizzatore sintattico (*parser*) che a partire dalle parole estratte dal *lexer* controlla che le frasi del programma appartengano al linguaggio;
3. un analizzatore semantico che effettua i controlli di tipo e rende gli operatori non ambigui (in caso di overloading);
4. un generatore di codice per la creazione del programma oggetto.

Il candidato dovrà progettare e realizzare (utilizzando un qualunque linguaggio di programmazione) parte del punto 1 e i punti 2 e 3 interamente secondo le spiegazioni sotto riportate.

DOMANDA 1

Del blocco 1 si supponga di avere a disposizione una funzione `nextString` che, quando invocata, restituisce la stringa successiva nel programma ignorando caratteri bianchi, fine linea, e commenti. Si supponga inoltre di avere a disposizione una tabella (TABELLA 1) che stabilisce le corrispondenze tra le parole nel programma sorgente e le categorie lessicali del linguaggio. Se un simbolo non compare in questa tabella può essere o una costante numerica (reale o intera) oppure un identificatore di variabile. Quindi, se il simbolo non appartiene alla tabella viene passato a un riconoscitore di numeri, invocabile attraverso una funzione data `isNum`, che nel caso la stringa sia un numero intero restituisce il simbolo `numInt`, mentre nel caso la stringa rappresenti un numero reale restituisce il simbolo `numReal`. Se la stringa non compare nella tabella, e non è riconosciuta come numero allora è considerata come un identificatore di variabile e associato al simbolo `identifier`. Si realizzi una primitiva `getCategory` che accedendo alla tabella (si decida anche una organizzazione della tabella appropriata, ad esempio tabella hash oppure tabella ordinata) e sfruttando la primitiva `isNum`, restituisca il simbolo corrispondente (terminale per la grammatica usata dal secondo blocco sotto specificata).

Simboli nel programma	Categorie lessicali
+	piu
-	meno
*	mul
/	div
begin	beginSymbol
end	endSymbol
:=	assignSym
;	;
:	:

TABELLA 1

Utilizzando la primitiva `getCategory` si trasformi il programma sorgente in un programma "astratto" in cui i simboli sono sostituiti con le categorie lessicali del linguaggio (ossia tutte le categorie in tabella più `numInt`, `numReal` e `identifier`).

DOMANDA 2

Si progetti e realizzi il parser del compilatore che, a partire dalla rappresentazione del programma estratta al punto precedente, controlli la consistenza sintattica del programma stesso. In caso contrario, restituisca un errore. Il linguaggio è definito dalla seguente grammatica che ha come simboli non terminali quelli racchiusi tra parentesi angolari < >, come simboli terminali `beginSymbol`, `endSymbol`, `int`, `real`, `identifier`, `:`, `;`, `assignSym`, `piu`, `meno`, `mul`, `div`. Lo scopo della grammatica è `<programma>`. Si noti che il simbolo | rappresenta un "oppure" e le parentesi graffe rappresentano la ripetizione di 0 o più volte del loro contenuto.

```
<programma> ::= beginSymbol <definizioni> <istruzioni> endSymbol
<definizioni>::= <defVariabile> {<defVariabile>}
<defVariabile>::= identifier: int; | identifier : real;
<istruzioni>::= <assegnamento>{<assegnamento>}
<assegnamento>::= identifier assignSym <espressione>;
<espressione>::= <espressione> <operatore> <espressione> | <numero> | identifier
<operatore>::= piu | meno | mul | div
<numero>::= numInt | numReal
```

Si noti che le espressioni non contemplano l'uso di parentesi. Quindi le operazioni seguono le classiche regole di priorità (moltiplicazione e divisione sono più prioritarie della somma e sottrazione) e sono associative a destra. Il candidato dovrà realizzare l'analizzatore sintattico.
In caso di successo dell'analisi sintattica, l'analizzatore dovrà produrre in uscita, per ogni variabile che si trova nella parte sinistra di un assegnamento un albero binario che rappresenta l'espressione alla destra dell'assegnamento (ricordando le priorità tra operatori e l'associatività). Un esempio è riportato in FIGURA 1

FACOLTATIVO: Si discuta come verrebbe complicata la grammatica e il progetto dell'analizzatore nel caso in cui la grammatica per le espressioni preveda l'uso di parentesi per modificare la priorità degli operatori.

DOMANDA 3
Si passi alla progettazione e realizzazione dell'analizzatore semantico. Esplorando ogni albero binario generato al punto precedente controlli i tipi degli identificatori e dei numeri in esse contenute e renda gli operatori non ambigui. Ogni operatore ha un unico simbolo per rappresentare 2 operazioni: quella tra interi e quella tra reali, ma per generare il codice oggetto sarà necessario risolvere questa ambiguità. Caso particolarmente significativo è la divisione. Quindi, se una espressione è omogenea (ossia contenente numeri e identificatori dello stesso tipo) l'operatore generico ambiguo sarà sostituito con l'operatore relativo a quel tipo. Se invece e' eterogenea, è necessario convertire tramite una primitiva data `intToReal` gli interi in reali (e sostituire i nodi dell'albero binario) e usare l'operazione tra reali. Quindi devono essere sostituiti tutti i nodi dell'albero che rappresentano operatori con l'operatore non ambiguo.
Si supponga che per ciascuna delle quattro operazioni siano date due realizzazioni:

- `+` associato alla categoria lessicale `piu` ha le realizzazioni `piuInt` e `piuReal`
- `-` associato alla categoria lessicale `meno` ha le realizzazioni `menoInt` e `menoReal`
- `*` associato alla categoria lessicale `mul` ha le realizzazioni `mulInt` e `mulReal`
- `/` associato alla categoria lessicale `div` ha le realizzazioni `divInt` e `divReal`

L'albero relativo alla FIGURA 1 in cui gli operatori sono stati resi non ambigui è riportato in FIGURA 2

Esempio: se nel programma originale abbiamo:

`X:= 3 - 5 * Y + 4`
produce l'albero binario

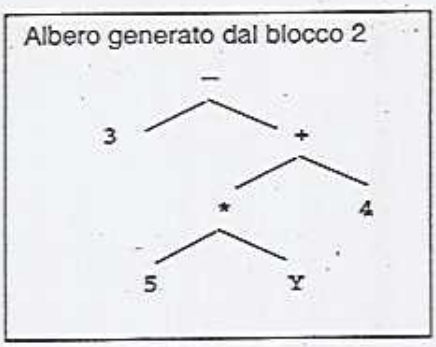

FIGURA 1

FIGURA 2

Infine per ogni istruzione si proceda a una valutazione. Esplorando l'albero sintattico modificato (FIGURA 2), si valuti ogni espressione e si fornisca un risultato. Poiché l'unica istruzione del linguaggio è l'assegnamento, al termine di questa fase, si controlli che ogni variabile che si incontra alla destra dell'assegnamento sia associata a un valore dello stesso tipo definito nella parte iniziale del programma.