Corso: Fondamenti, Linguaggi e Traduttori

Paola Giannini

Generazione Codice

Il codice oggetto (**dc**)

- **dc** è una macchina a stack
- ha registri che hanno come nome un singolo carattere, ad esempio `a`, `A` , ... che possono essere caricati dallo stack o il cui contenuto può essere memorizzato nello stack. L' istruzione
  - `sa` fa il pop dello stack e lo mette in `a`
  - `la` fa il push sullo stack del registro `a`
- il comando `p` stampa il top dello stack senza fare il pop
- il comando `P` fa il pop dello stack
- per default della precisione di **dc** è `0`, ma per cambiare la precisione (quando c'è una conversione a float) si può usare il comando `k` che setta la precisione al numero sul top dello stack
- per fare delle prove digitate **dc** in un terminale e entrate nell'interprete per cui potete vedere cosa succede (il comando `f` stampa lo stack)
- potete anche mettere il codice in un file (ad esempio `output.txt`) e eseguire il codice da riga di comando con

```
gc output.txt
```

Generazione del codice

- Deve produrre una stringa che è la traduzione del programma sorgente.
- Per generare il codice dobbiamo associare ad ogni identificatore un registro. Il registro associato ad un identificatore sarà rappresentato da un campo della classe `Attributes` (quindi dovete aggiungere alla classe `Attributes` un campo `registro` di tipo **`char`**).
- La generazione dei registri va fatta da un metodo che deve anche tenere conto dei registri che sono già stati generati.
- Per questo definite una classe `Registri` che dovrà contenere il metodo **`char`** `newRegister()`. Usate lo schema più semplice possibile per fare questo. Ad esempio, potete avere un `ArrayList` di caratteri con i caratteri che decidete i usare per i registri (ad esempio tutte le lettere maiuscole e minuscole) e il metodo `newRegister()` rimuove un carattere dall' `ArrayList` e lo ritorna (`ArrayList` e metodo possono essere **`static`**).
  - Quando vengono assegnati i registri alle variabili nella symbol table?
  - Quando processate il `NodeId` non c'è bisogno di cercare nella symbol table (perchè avete il campo `definition`)
- L'unico errore che può succedere durante la generazione del codice è che ci siano più identificatori di quelli rappresentabili con i registri.

UPO
UNIVERSITÀ DEL PIEMONTE ORIENTALE

La precisione

- Per default la precisione di `dc` ha 0 cifre decimali.
- Quando troviamo un nodo di conversione dobbiamo modificare la precisione a 5 cifre decimali emettendo il codice `5 k`.
- Alla fine di una istruzione di assegnamento (nella quale avremo potuto potuto cambiare la precisione) dobbiamo ristabilire la precisione a 0 cifre decimali emettendo il codice `0 k`. Vediamo un esempio

```
int tempA
float tempB
tempB = 1.0 / 6
print tempB
tempA = 1 / 6
print tempA
```

- assumendo che il registro per `tempA` sia `a` e quello per `tempB` sia `b`, la strategia descritta produce la seguente stringa `codice`:

```
1.0 6 5 k / sb 0 k  lb p P 1 6 / sa 0 k  la p P
```

- senza fare il reset della precisione dopo l'assegnamento invece avremo

```
1.0 6 5 k / sb lb p P 1 6 / sa la p P
```

provate a eseguire entrambi!

Implementazione della generazione del codice (1)

- Come per il type checking aggiungeremo un campo a `NodeAST` che chiamiamo `codice` di tipo `String` e che conterrà il codice generato per la parte di programma rappresentata dal nodo specifico.
  - La traduzione delle dichiarazioni farà l'assegnamento del registro (modifica della symbol table) e poi in caso ci sia una inizializzazione genera il codice come per un assegnamento alla variabile dichiarata.
  - La traduzione dell'assegnamento deve essere tale che dopo la sua esecuzione nel registro corrispondente all'identificatore a sinistra è memorizzato il risultato dell'espressione a destra dell'assegnamento.
  - La traduzione di una espressione deve essere tale che dopo la sua esecuzione il risultato dell'espressione si trova sul top dello stack.
  - Per l'istruzione `print` notate che la corrispondente print di dc non fa il pop dello stack.

UPO
UNIVERSITÀ DEL PIEMONTE ORIENTALE

Implementazione della generazione del codice (2)

Come per il type checking possiamo calcolare il campo codice dei nodi aggiungendo un metodo che lo calcola oppure con il pattern visitor.

- Nel primo caso in `NodeAST` aggiungiamo un metodo **`public abstract void`** `calcCodice()` poi ridefinito nei nodi concreti:

```
public abstract class NodeAST {
  private String codice;
    .......
  public abstract void calcCodice(); // assegna a codice il codice generato per il nod
  public String getCodice() {
    return codice;
  }
}


public class NodeBinOp extends NodeExpr {
  private LangOper op; private NodeExpr left; private NodeExpr right;
        ......................
  public void calCodice(){
    left.calcCodice; // codice per la sotto-espressione di sinistra
    right.calcCodice; // codice per la sotto-espressione di destra
    char opereatore = ........
    codice = ....... // assegnamento al campo
  }
}
```

Processamento dei nodi

1. `NodeDecl`
   Genera un nuovo registro da associare all'attributo dell'identificatore dichiarato e lo assegna all'identificatore nella symbol table. Se c'è una inizializzazione deve generare codice come per l'assegnamento (alla variabile dichiarata).
2. `NodeAssign`
   Genera codice per l'espressione a destra dell'assegnamento. Memorizza il top dello stack nel registro associato all' identificatore a sinistra. Riporta la precisione a `0`.
3. `NodePrint`
   Genera il codice per fare il push sullo stack del registro associato al identificatore. Genera il codice per stamparlo e poi rimuoverlo dallo stack.
4. `NodeBinOp`
   Genera codice per l'espressione a sinistra, poi per quella a destra e poi quello dell'operazione.
5. `NodeDeRef`
   Genera il codice per fare il push sullo stack del registro associato all'identificatore.
6. `NodeCost`
   Genera il codice per fare il push sullo stack della costante.
7. `NodeConv`
   Genera il codice per cambiare la precisione a 5 cifre decimali.

UPO
UNIVERSITÀ DEL PIEMONTE ORIENTALE