Linguaggi di programmazione - Principi e paradigmi 2/ed
Maurizio Gabbrielli, Simone Martini
Copyright © 2011 - The McGraw-Hill Companies srl

Approfondimento 6.1

Alcuni problemi di scope

Discuteremo in questo paragrafo alcune questioni relative allo scope statico; le differenze maggiori esistenti fra le regole di scope statico dei vari linguaggi sono relative a dove possono essere introdotte le dichiarazioni e a quale sia l'esatta visibilità delle variabili locali. Le regole di scope appena viste infatti si prestano a qualche ambiguità ed in alcuni casi possono anche essere causa di comportamenti anomali. Discuteremo qui alcuni casi significativi di diverse situazioni che si possono presentare.

Prendiamo innanzitutto il caso di Pascal, nel quale la regola di scope statico già vista è estesa con le seguenti regole aggiuntive:

(i) le dichiarazioni possono comparire solo all'inizio di un blocco;
(ii) lo scope di un nome si estende dall'inizio alla fine del blocco nel quale appare la dichiarazione del nome stesso (escludendo gli eventuali "buchi" di scope) indipendentemente dalla posizione della dichiarazione;
(iii) i nomi non predefiniti nel linguaggio devono essere dichiarati *prima* di essere usati.

È dunque un errore scrivere

```
begin
   const pippo = valore;
   const valore = 0;
...
end
```

perché si utilizza `valore` prima della sua definizione. Si potrebbe supporre che tale frammento divenga corretto se inserito in un blocco che già contiene una definizione per `valore`; anche in questo caso, tuttavia, si producono anomalie. Se infatti abbiamo:

```
begin
   const valore = 1;
   procedure pluto;
       begin
       const pippo = valore;
       const valore = 0;
       ...
       end
...
end
```

Linguaggi di programmazione - Principi e paradigmi 2/ed
Maurizio Gabbrielli, Simone Martini
Copyright © 2011 - The McGraw-Hill Companies srl

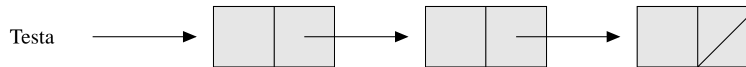

Figura 16.8 Una lista.

le regole viste ci dicono che la dichiarazione della procedura `pluto` introduce un blocco interno, nel quale la dichiarazione locale di `valore` (che inizializza a 0 la costante) copre la dichiarazione esterna (che inizializza a 1 la costante). Quindi il nome `valore` che compare nella dichiarazione

```
const pippo = valore;
```

si dovrebbe riferire alla dichiarazione

```
const valore = 0;
```

del blocco interno. Tuttavia tale dichiarazione arriva dopo l'uso del nome, contravvenendo così alla regola iii). In una tale situazione quindi il comportamento più corretto per un compilatore Pascal è quello di segnalare un errore di semantica statica al momento in cui viene analizzata la dichiarazione di `pippo`. Alcuni compilatori, tuttavia, assegnano a `pippo` il valore 1, il che è evidentemente scorretto in quanto viola la regola di visibilità.

Per evitare questo tipo di problemi alcuni linguaggi con scope statico, quali C e Ada, limitano lo scope della dichiarazione alla porzione di blocco compresa fra il punto in cui la dichiarazione compare e la fine del blocco stesso (escludendo al solito i buchi di scope).

In tali linguaggi dunque non si ha il problema visto in precedenza in quanto il nome `valore` che compare nella dichiarazione

```
const pippo = valore;
```

si riferirebbe alla dichiarazione del blocco esterno, dato che il nome dichiarato internamente non è ancora visibile. Anche in questi linguaggi i nomi devono essere dichiarati prima di essere usati, per cui anche qui le dichiarazioni

```
const pippo = valore;
const valore = 0;
```

producono un errore se `valore` non è dichiarato in un blocco esterno.

La regola iii), che prescrive la dichiarazione *prima* dell'uso, in alcuni casi è particolarmente gravosa: impedisce infatti la definizione di tipi ricorsivi o di procedure mutuamente ricorsive.

Supponiamo, ad esempio, di voler definire un tipo di dati corrispondente ad una lista che, come noto, è una struttura dati a dimensione variabile costituita da una successione ordinata, eventualmente vuota, di elementi di un qualche tipo, in cui è possibile aggiungere o togliere degli elementi e dove si può accedere direttamente solo al primo elemento (per accedere ad un generico elemento occorre

Linguaggi di programmazione - Principi e paradigmi 2/ed
Maurizio Gabbrielli, Simone Martini
Copyright © 2011 - The McGraw-Hill Companies srl

scandire sequenzialmente la lista). Una lista, come mostrato in Figura 16.8, può essere realizzata usando una successione di elementi, ognuno dei quali è costituito da due campi: il primo conterrà l'informazione che ci interessa memorizzare (ad esempio, un numero intero); il secondo conterrà un puntatore all'elemento successivo della lista, se questo esiste, oppure il valore `nil` se la lista termina. Possiamo accedere alla lista tramite il puntatore al primo elemento della lista stessa, usualmente chiamato *testa*. In Pascal possiamo definire il tipo lista come segue

```
type lista = ^elemento;
type elemento = record
                informazione: intero;
                successivo: lista
                end
```

dove `^T` indica il tipo dei puntatori ad oggetti di tipo `T`. Un valore di tipo `lista` è un puntatore ad un generico elemento della lista; un valore di tipo `elemento` corrisponde ad un elemento della lista, costituito da un intero e da un campo a sua volta di tipo `lista` che permette di collegarsi all'elemento successivo. Tale dichiarazione è scorretta secondo la regola iii): comunque si consideri l'ordine delle dichiarazioni `lista` ed `elemento`, infatti, useremo un nome che non è ancora stato definito. Questo problema è risolto in Pascal rilassando il vincolo iii): per i dati di tipo puntatore, e solo per essi, è permesso un riferimento ad un nome che ancora non è dichiarato. Le dichiarazioni viste sopra per la lista sono quindi corrette in Pascal.

Nel caso di C e Ada, invece, l'analogo delle precedenti dichiarazione non è ammesso e per poter specificare tipi mutuamente ricorsivi si devono usare dichiarazioni di tipo *incomplete*, che introducono un nome che sarà ulteriormente specificato in seguito. Ad esempio, in Ada dovremmo scrivere

```
type elemento;
type lista = access elemento;
type elemento is record
                informazione: intero;
                successivo: lista
                end
```

risolvendo così il problema dell'uso di un nome prima delle sua dichiarazione.

Il problema si presenta in modo analogo per la definizione di procedure mutuamente ricorsive. Pascal usa in questo caso definizioni incomplete: se la procedura `pippo` deve essere definita in termini della procedura `pluto` e viceversa, in Pascal si deve scrivere

```
procedure pippo(A:integer); forward;
procedure pluto(B: integer);
    begin
    ...
    pippo(3);
    ...
    end
procedure pippo;
    begin
    ...
```

Linguaggi di programmazione - Principi e paradigmi 2/ed
Maurizio Gabbrielli, Simone Martini
Copyright © 2011 - The McGraw-Hill Companies srl


```
pluto(4);
...
end
```

È curioso osservare che C permette invece l'uso di un identificatore prima della sua dichiarazione nel caso di nomi di funzioni: la dichiarazione di funzioni mutuamente ricorsive non necessita di alcun accorgimento speciale.

I modi con cui i vincoli iii) sono rilassati sono tanti quanti i linguaggi di programmazione. Java, ad esempio, consente che una dichiarazione possa apparire in un punto qualsiasi di un blocco. Se la dichiarazione corrisponde ad una variabile, lo scope del nome dichiarato si estende dal punto della dichiarazione fino alla fine del blocco (escludendo gli eventuali "buchi" di scope); se la dichiarazione invece si riferisce al membro di una classe (sia esso un campo o un metodo), essa è visibile in tutta la classe nella quale compare, indipendentemente dall'ordine nel quale appaiono le dichiarazioni.