Corso di Laurea Magistrale in Ingegneria Informatica
A.A. 2013-2014

Linguaggi Formali e Compilatori

Backus Naur Form (BNF)

Giacomo PISCITELLI

Descrizione compatta delle grammatiche: la BNF

Per descrivere in maniera compatta le grammatiche si usa la BNF (*Backus–Naur form*), che fornisce le regole della grammatica.

I terminali e i nonterminali vengono descritti tramite parole che ne indicano il significato; per poterli distinguere, i terminali vengono scritti in **grassetto**.

Una regola BNF, quindi, è data da un **corpo della regola** (che descrive in chiaro la composizione della sequenza di terminali e nonterminali) e da un **costrutto (o testa) della regola** (descritto tramite parole che indicano il significato del corpo stesso).

Al posto di '→' scriviamo '::='.

La regola spiega, come una produzione, il modo di riscrivere il costrutto della regola, ma ha una sintassi molto più ricca per descrivere tale costrutto.

- ✓ Ogni terminale e ogni nonterminale è un'**espressione BNF**; una regola contenente solo un terminale o un nonterminale corrisponde in modo ovvio a una produzione.
- ✓ La giustapposizione di espressioni BNF è ancora un'espressione BNF.

Descrizione compatta delle grammatiche: la BNF

- La notazione $A ::= e_0 \mid e_1 \mid \cdots \mid e_{n-1}$ significa che la grammatica contiene le produzioni $A \rightarrow e_0$, $A \rightarrow e_1$,. . . , $A \rightarrow e_{n-1}$, e cioè che per riscrivere A bisogna scegliere, alternativamente, uno dei costrutti e_i.
- La notazione $[e]$ nel membro destro di una produzione significa che l'espressione e può essere usata o meno, cioè è opzionale.
- La notazione $\{e\}$ nel membro destro di una produzione significa che l'espressione e può comparire zero o più volte.
- Le parentesi tonde possono essere utilizzate per raggruppare: per esempio, $(a \mid b)\ [c]$ è diverso da $a \mid (b\ [c])$, che è uguale a $a \mid b\ [c]$.

Descrizione compatta delle grammatiche: la BNF

Esempi

- parola ::= {**a**}
- parola ::= {**ab**} **a** | **b** {**ab**} **a** | ε
- ◦ cifra ::= **0** | **1** | **2** | · · · | **9**
  - ◦ cifrapari ::= **0** | **2** | **4** | **6** | **8**
  - ◦ numeropari ::= {cifra} cifrapari
- ◦ underscore ::= **_**
  - ◦ lettera ::= **A** | **B** | · · · | **Z** | **a** | **b** | · · · | **z**
  - ◦ cifra ::= **0** | **1** | **2** | · · · | **9**
  - ◦ identificatore ::= (lettera | underscore){lettera | underscore | cifra}
- istr-if ::= **if (** espr **)** istr [**else** istr]
- ciclo-for ::= **for** ([espr] ; [espr] ; [espr]) istr
- ciclo-while ::= **while** (espr) istr
- ciclo-do ::= **do** istr **while** (espr);

Analisi lessicale

Dato che un file sorgente è una sequenza di caratteri, per poter stabilire se la sequenza è un programma C corretto, occorre tokenizzarlo, cioè assegnare sequenze di caratteri ASCII a token del linguaggio C.

Per farlo, **il compilatore utilizza una logica di massimizzazione**: durante l'analisi lessicale, il massimo numero di caratteri consecutivi viene accumulato in un token.

Per esempio, se scrivete return0 invece di return 0 l'analizzatore lessicale decide che avete voluto specificare l'identificatore return0 invece di return seguito dal numero 0.

Al contrario, scrivere return(0) o return (0) è equivalente, perchè quando il compilatore arriva alla parentesi aperta si accorge che il token corrente è terminato, ed è return.

La BNF del linguaggio C: elementi lessicali

Nelle slide che seguono si riporta in sintesi la BNF del linguaggio C.

Per una dettagliata descrizione della BNF del C si rimanda al seguente sito:

http://www.cs.man.ac.uk/~pjj/bnf/c_syntax.bnf

Le **parole chiave** sono token a cui sono assegnati significati particolari, e che non possono essere utilizzate dal programmatore come identificatori:

auto	**do**	**goto**	**signed**	**unsigned**
break	**double**	**if**	**sizeof**	**void**
case	**else**	**int**	**static**	**volatile**
char	**enum**	**long**	**struct**	**while**
const	**extern**	**register**	**switch**	
continue	**float**	**return**	**typedef**	
default	**for**	**short**	**union**	

Gli **identificatori** sono un altro tipo di token, definiti in BNF come segue:

- lettera ::= **A** | **B** | · · · | **Z** | **a** | **b** | · · · | **z**
- cifra ::= **0** | **1** | **2** | · · · | **9**
- ident ::= (_ | lettera){_ | lettera | cifra}

La BNF del linguaggio C: elementi lessicali

Costanti

Le costanti (caratteri o numeri) rappresentano ancora un altro tipo di token.

costante ::= cost-car | cost-int | cost-float

cost-car ::= 'c' | 'seq-escape' dove c è un qualunque carattere ASCII tranne l'apostrofo, il backslash e il newline

seq-escape ::= **\'** | **** | **** | **\n** | **\t** | **\x**cifra-esa{cifra-esa}

cifra-esa ::= cifra | **A** | **B** | **C** | **D** | **E** | **F** | **a** | **b** | **c** | **d** | **e** | **f**

cost-int ::= (cost-dec | cost-esa) [su_-int]

nonzero ::= **1** | **2** | · · · | **9**

cost-dec ::= **0** | nonzero{cifra}

cost-esa ::= (**0x** | **0X**)cifra-esa{cifra-esa}

su_-int ::= [**U** | **u**][**L** | **l**]

cifre ::= cifra {cifra}

cost-float ::= cost-fraz [esponente] [su_-float] |

cifre esponente [su_-float]

cost-fraz ::= [cifre] . cifre | cifre .

esponente ::= [**e** | **E**][**+** | **-**] cifre

su_-float ::= **f** | **F** | **l** | **L**

Un programma C

programma ::= file {file}
file ::= {dichiarazione | def-fun}
def-fun := tipo dichiaratore**(**dich-param**)** istr-comp
istr-comp ::= **{** {dichiarazione} {istr} **}**
istr ::= istr-comp | istr-espr | istr-salto | istr-if | istr-iter | istr-switch | istr-vuota
istr-vuota ::= ;
istr-espr ::= espr;
istr-salto ::= **break**; | **continue**; | **return** [espr];
istr-if ::= **if (**espr**)** istr [**else** istr]
istr-switch ::= **switch**(espr)
{ {**case** espr-cost: {istr} } [**default:** {istr}] **}**
istr-iter ::= ciclo-for | ciclo-while | ciclo-do
ciclo-for ::= **for (**[espr] ; [espr] ; [espr]**)** istr
ciclo-while ::= **while (**espr**)** istr ciclo-do ::= **do** istr **while (**espr**)**;

Dichiarazioni in C

Le dichiarazioni C servono a due scopi: definire le variabili, e dare i prototipi delle funzioni non ancora definite, che servono al compilatore per conoscere il tipo dei loro argomenti e del loro risultato prima della loro definizione.

La sintassi delle dichiarazioni è una delle parti più complesse del linguaggio.

dichiarazione ::= tipo dichiaratore {, dichiaratore} ;
tipo dichiaratore ::= ([**unsigned**] [**char** | **int** | **long** | **short**]) | **void** | **float** | **double**
dichiaratore ::= {*****} dich-dir
dich-dir ::= ident | **(**dichiaratore**)** |
dich-dir**[** [espr-cost] **]** | dich-dir**(**dich-param**)**
dich-param ::= void | (tipo dichiaratore { , tipo dichiaratore })

Il preprocessore di C

I programmi C, prima di essere compilati, vengono elaborati da un preprocessore. Il preprocessore legge i caratteri che compongono il programma, effettua delle modifiche e quindi dà il risultato in pasto al compilatore.

Ci sono due usi fondamentali del preprocessore: includere nel testo altri file, e alterare la struttura lessicale del programma.

Il preprocessore è controllato da direttive, che devono stare interamente su una riga, e sono introdotte da un carattere # che deve essere necessariamente il primo carattere della riga.

preproc-dir ::= (dir-include | dir-define)

dir-include ::= **#include** (”nome-file” | <nome-file>)

dir-define ::= **#define** ident [(ident{, ident})] *qualsiasi cosa*