PCS-3566 / PCS-3866
Linguagens e Compiladores

Prof. João José Neto

Aula 02 – Especificação Gramatical da linguagem Dartmouth BASIC

Extraída do livro de J.A.N.Lee "*The anatomy of a compiler*", Van Nostrand Reinhold, 1967, p. 250, e adaptada para a Notação de Wirth

Introdução

- Em maio de 1964 foi disponibilizada a linguagem **Dartmouth BASIC**, historicamente uma das pioneiras.
- Esta apresentação expõe essa linguagem com certo grau de detalhe, para que possa ser usada como meta do projeto de um compilador ou interpretador a ser desenvolvido como parte prática desta disciplina.
- Para simplificar o trabalho de implementação a ser desenvolvido, a gramática adotada para a descrição da sintaxe da linguagem, originalmente denotada em BNF (Backus-Naur Form), foi convertida para a **Notação de Wirth**.
- Isso traz grandes vantagens práticas, por ser a Notação de Wirth muito mais **amigável** que a BNF para as manipulações necessárias aos processos de compilação adotados.

Generalidades

- Não vamos nesta disciplina projetar a linguagem-alvo do nosso estudo, e sim **analisá-la** a partir de uma especificação gramatical fornecida.
- Essa especificação foi adaptada para a Notação de Wirth a partir de uma gramática equivalente, originalmente publicada em BNF.
- Isso pode ser feito com base em informações encontradas em livros sobre a teoria e análise de linguagens de programação.
- Para isso, convém ter familiaridade com o conteúdo da obra ***Concepts of Programming Languages*****,** de **R.W.Sebesta**, disponível [setembro/2018] em:

`https://cs444pnu1.files.wordpress.com/2014/02/concepts-of-programming-languages-10th-sebesta.pdf`

A Notação de Wirth

- Metalinguagens são notações usadas para descrever linguagens.
- A Notação de Wirth é a metalinguagem que foi adotada nesta disciplina para redigir gramáticas de linguagens de programação
- No slide a seguir, ela foi utilizada para descrever a sintaxe da linguagem Dartmouth BASIC.
- Esta notação utiliza as seguintes convenções:
  - **()** parênteses – agrupam opções sintáticas alternativas
  - **[]** colchetes – agrupam opções sintáticas opcionais (0 ou 1 vez)
  - **{}** chaves – agrupam opções sintáticas que se podem repetir indefinidamente (0 a infinitas vezes)
  - **|** barra vertical – separa entre si opções sintáticas
  - **α β** – dois elementos α e β concatenados representam a justaposição de instâncias das sintaxes por eles representadas
  - **ε** – a letra épsilon denota a cadeia vazia na Notação de Wirth
  - **“α”** – cadeias α entre aspas denotam terminais da gramática
  - **α** – cadeias α sem aspas denotam não-terminais da gramática

Gramática Original, em Notação de Wirth

1. Program= BStatement { BStatement } int “**END**” .
2. BStatement= int (Assign | Read | Data | Print | Goto | If | For | Next | Dim | Def | Gosub | Return | Remark) .
3. Assign= “**LET**” Var “**=**” Exp .
4. Var= letter digit | letter [“**(**” Exp { “**,**” Exp } “**)**”] .
5. Exp= { “**+**” | “**-**” } Eb { (“**+**” | “**-**” | “*****” | “**/**” | “**↑**”) Eb } .
6. Eb= “**(**” Exp “**)**” | Num | Var | (“**FN**” letter | Predef) “**(**” Exp “**)**” .
7. Predef= “**SIN**” | “**COS**” | “**TAN**” | “**ATN**” | “**EXP**” | “**ABS**” | “**LOG**” | “**SQR**” | “**INT**” | “**RND**” .
8. Read= “**READ**” Var { “**,**” Var } .
9. Data= “**DATA**” Snum { “**,**” Snum } .
10. Print= “**PRINT**” [Pitem { “**,**” Pitem } [“**,**”]].
11. Pitem= Exp | “**“**” Character { Character } “**”**” [Exp] .
12. Goto= (“**GOTO**” | “**GO**” “**TO**”) int .
13. If= “**IF**” Exp (“**>=**” | “**>**” | “**<>**” | “**<**” | “**<=**” | “**=**”) Exp “**THEN**” int .
14. For= “**FOR**” letter [digit] “**=**” Exp “**TO**” Exp [“**STEP**” Exp] .
15. Next= “**NEXT**” letter [digit] .
16. Dim= “**DIM**” letter “**(**” int { “**,**” int } “**)**” { “**,**” letter “**(**” int { “**,**” int } “**)**” } .
17. Def= “**DEF FN**” letter “**(**” letter [digit] “**)**” “**=**” Exp .
18. Gosub= “**GOSUB**” int .
19. Return= “**RETURN**” .
20. Remark= “**REM**” { Character } .
21. Int= digit { digit } .
22. Num= (Int [“**.**” { digit }] | “**.**” Int) [“**E**” [“**+**” | “**-**”] Int] .
23. Snum= [“**+**” | “**-**”] Num .
24. Character= letter | digit | special .

Observação

- Apenas para completar a especificação, na gramática do slide anterior estão omitidas as regras que definem os não-terminais **letter, digit e special** (que apenas classificam de forma trivial os caracteres ASCII nessas três categorias):

25. letter = “A” | “B” | “C” | “D” | “E” | “F” | “G” | “H” | “I” | “J” | “K” | “L” | “M” | “N” | “O” | “P” | “Q” | “R” | “S” | “T” | “U” | “V” | “W” | “X” | “Y” | “Z” | “a” | “b” | “c” | “d” | “e” | “f” | “g” | “h” | “i” | “j” | “k” | “l” | “m” | “n” | “o” | “p” | “q” | “r” | “s” | “t” | “u” | “v” | “w” | “x” | “y” | “z” .

26. digit = “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9” .

27. special = “!” | “@” | “#” | “%” | “¨” | “&” | “*” | “(” | “)” | “_” | “+” | “-” | “=” | “§” | “{” | “[” | “ª” | “}” | “]” | “º” | “?” | “/” | “°” | “`” | “´” | “^” | “~” | “<” | “,” | “>” | “.” | “:” | “;” | “|” | “\” | “”” | “”” .

Análise da linguagem

- Vamos analisar a linguagem BASIC, definida pela gramática fornecida como sendo sua especificação sintática.
- Para isso, vamos procurar formular e depois responder a algumas questões naturais sobre a linguagem em estudo.
- A primeira questão pode ser: ***Que estrutura pode apresentar um programa expresso nesta linguagem? Simplesmente uma sucessão de comandos ou pode compreender formações sintáticas mais complexas?***
- Examinando isoladamente a regra da linha 1 da gramática:

1. Program= BStatement { BStatement } int "**END**" .

a resposta aparentemente correta para a pergunta pareceria ser "o programa não apresenta *nenhuma estrutura*".
No entanto, observando com mais cuidado a gramática, identificam-se diversas estruturas sintáticas mais complexas, conforme mostram os próximos slides.

Grupos aninháveis FOR ... NEXT

– O grupo FOR ... NEXT é definido nas linhas 14-15:

14. For= “**FOR**” letter [digit] “**=**” Exp “**TO**” Exp [“**STEP**” Exp] .
15. Next= “**NEXT**” letter [digit] .

Consiste de um par casado de comandos FOR e NEXT, (ou seja, ambos os comandos devem referir-se a uma mesma variável), delimitando uma sequência de comandos;

– A sequência delimitada por um comando FOR e pelo correspondente comando NEXT pode, por sua vez, compreender novas sequências, contendo outros pares casados de comandos FOR e NEXT (que não se refiram a nenhuma variável de controle de nenhum eventual par casado envolvente) com os respectivos comandos circundados, e assim por diante, caracterizando assim FOR ... NEXT como uma estrutura sintática aninhável.

Estruturas aninháveis FOR...NEXT

```
...
150 FOR J=5 TO 40
160 REM COMANDOS CONTROLADOS POR J
...
170 NEXT J
...
175     REM NOTAR QUE NENHUM GRUPO FOR...NEXT ATUA NESTE PONTO DO PROGRAMA
...
250 FOR K=1 TO 50
260 REM COMANDOS CONTROLADOS POR K
265 REM NOTAR QUE ESTE GRUPO ESTÁ NO MESMO NÍVEL DO GRUPO J ACIMA
...
350     FOR L=1 TO 10
360     REM COMANDOS CONTROLADOS POR L
365     REM NOTAR QUE K TAMBÉM CONTROLA ESTE GRUPO
...
370     NEXT L
...
470 NEXT K
...
```

Grupos GOSUB ... RETURN

– Nas linhas 18-19 da gramática lê-se:

```
18. Gosub= “GOSUB” int .
19. Return= “RETURN” .
```

No comando GOSUB, de chamada de subrotina, fica implícito que para a particular chamada de subrotina a que se refere, o corpo da subrotina se estende desde o ponto em que é ativada pelo comando GOSUB (no comando cujo número é referenciado por GOSUB), até o ponto do programa em que for executado o primeiro comando RETURN;

– Convém notar que isso caracteriza outra estrutura sintática não trivial, que possibilita ao programa ter um número arbitrário de comandos RETURN para o mesmo ponto de entrada ativado pelo comando GOSUB;

Grupos READ ... DATA

1. Read= “**READ**” Var { “**,**” Var } .
2. Data= “**DATA**” Snum { “**,**” Snum } .

- Este é outro grupo que define uma sintaxe não trivial desta linguagem.
- As variáveis mencionadas no comando READ associam-se aos números da lista que compõe o comando DATA
- A execução do comando READ promove a leitura dos valores contidos na lista de dados, e guarda cada um desses valores, na ordem em que forem lidos, nas variáveis que os referenciaram.

Estruturas básicas da linguagem

- São formas sintáticas de uso geral, que são utilizadas extensivamente pelos comandos da linguagem:
  - As linhas 21-24 definem as construções mais simples, como números inteiros, números reais, com e sem sinal, e os tipos de caracteres (letra, dígito, caractere especial):

```
21.Int= digit { digit } .
22.Num= ( Int [ “.” { digit } ] | “.” Int ) [ “E” [ “+” | “-” ] Int ] .
23.Snum= [ “+” | “-” ] Num .
24.Character= letter | digit | special .
```

  - As linhas 4-7 da gramática definem as expressões aritméticas, utilizadas na maior parte dos comandos:

```
4.  Var= letter digit | letter [ “(” Exp { “,” Exp } “)” ] .
5.  Exp= { “+” | “-” } Eb { ( “+” | “-” | “*” | “/” | “↑” ) Eb } .
6.  Eb= “(” Exp “)” | Num | Var | ( “FN” letter | Predef ) “(” Exp “)” .
7.  Predef= “SIN” | “COS” | “TAN” | “ATN” | “EXP” | “ABS” | “LOG” | “SQR” | “INT” | “RND” .
```

Exemplos dos comandos da linguagem

- Assign - 10 LET I2 = 25+4
- Read - 20 READ I3,J,K1
- Data - 30 DATA 4,-5,0
- Print - 40 PRINT “K1=”, K1, “ J=”, J
- Goto - 50 GO TO 20
- If - 60 IF K1 < I3 THEN 10
- For - 70 FOR I=1 TO K1 STEP J
- Next - 80 NEXT I
- Dim - 90 DIM A(3), B(2,5)
- Def - 100 DEF FNF(X)= X*5+3
- Gosub - 110 GOSUB 20
- Return - 120 RETURN
- Remark - 130 REM isto é um comentário.

Assign

3. Assign= “**LET**” Var “**=**” Exp .

- Calcula o valor da expressão e deposita o resultado na variável referenciada.

Read

8. Read= “**READ**” Var { “,” Var } .

- Lê valores, a partir do conteúdo de comandos DATA, e na ordem em que estiverem listados, e deposita cada um desses valores nas variáveis listadas no comando READ, também pela ordem na lista.

Data

9. Data= “**DATA**” Snum { “**,**” Snum } .

- Este comando por si só nada faz, mas contém uma lista de números relativos, a ser lida utilizando comandos READ.

Print

10. Print= “**PRINT**” [Pitem { “,” Pitem } [“,”]].
11. Pitem= Exp | “”” Character { Character } “”” [Exp] .

- Imprime uma lista de valores, cada qual pode ser:
  - Uma expressão aritmética
  - Uma string de caracteres ASCII entre aspas
  - Uma string seguida por uma expressão aritmética.
- Uma vírgula, ao final da lista de valores a imprimir, indica que não se deseja mudar de linha ao final da execução do comando PRINT.

Goto

9. Goto= (“**GOTO**” | “**GO**” “**TO**”) int .

- Desvia incondicionalmente para o comando cujo número coincide com o inteiro referenciado no comando GOTO.
- É indiferente usar a palavra GOTO ou as palavras GO TO separadas por um espaço.

If

10. If= “**IF**”
Exp (“**>=**” | “**>**” | “**<>**” | “**<**” | “**<=**” | “**=**”) Exp
“**THEN**” int .

- Compara duas expressões aritméticas e desvia para o comando indicado pelo inteiro após a palavra THEN caso a comparação resulte verdadeira.

For

11. For= “**FOR**” letter [digit] “**=**” Exp “**TO**” Exp [“**STEP**” Exp] .

- Este comando inicia e continua um loop iterativo controlado pela variável indicada após a palavra FOR.
- Não pode haver espaço entre a letra e o dígito, se for o caso.
- Para iniciar, o valor da expressão à direita do sinal = é depositado na variável de controle, e o controle passa para o próximo comando.
- Após a execução do comando NEXT que referencia a mesma variável de controle, a execução volta para este comando, e a variável de controle é acrescida do passo indicado pela expressão após a palavra STEP.
- Caso seja omitida essa palavra, a variável é simplesmente incrementada.
- O loop termina quando o valor da variável coincidir ou ultrapassar o valor da expressão à direita da palavra TO.

Next

12. Next= “**NEXT**” letter [digit] .

- Isoladamente este comando não tem sentido.
- Forma par casado com um comando FOR que referencie a mesma variável de controle.
- Serve para delimitar um bloco de comandos a ser iterado.
- Trata-se da sequência de comandos compreendida entre os comandos FOR e NEXT casados.

Dim

13. Dim= “**DIM**” letter “(” int { “,” int } “)”
{ “,” letter “(” int { “,” int } “)” }.

- Este comando serve para declarar e dimensionar vetores e matrizes multidimensionais, a serem utilizadas por algoritmos em expressões aritméticas.
- Vetores e matrizes têm nome com uma única letra.
- Os valores máximos dos vários índices são declarados entre parênteses, separados por vírgulas
- O mesmo comando DIM pode ser usado para declarar e dimensionar diversos vetores e matrizes.

Def

14. Def= “**DEF FN**” letter “(” letter [digit] “)” “=” Exp .

- Define funções do usuário, cujo nome é sempre FNx onde x á uma letra.
- Não é permitido inserir espaço antes dessa letra.
- É permitido definir função com um parâmetro, e esse parâmetro é uma variável simples.
- O corpo da função é a expressão à direita do sinal =.

Gosub

15. Gosub= "**GOSUB**" int .

- Salva o endereço do comando seguinte e desvia incondicionalmente para o comando cujo número seja o referenciado pelo comando GOSUB.
- Este comando é casado ao comando RETURN em tempo de execução.

Return

16. Return= “**RETURN**” .

- Devolve o controle ao comando seguinte ao comando GOSUB que ativou a subrotina.

Remark

17. Remark= “**REM**” { Character } .

- Este comando apenas contém uma cadeia de caracteres que faz o simples papel de um comentário, usado pelo programador como documentação para o programa, e nada executa.

Observação

- Diversas implementações da linguagem BASIC incluem um comando de leitura adicional para ler dados a partir do teclado, um comando INPUT, com uma sintaxe que permite escrever, por exemplo, o comando seguinte:

```
10 INPUT “DIGITE UM NÚMERO ENTRE 1 E 20”, N
```

- Isto apresenta na tela o texto entre aspas (que é opcional) e aguarda a entrada de um dado numérico, cujo valor, depois de digitado pelo usuário, é depositado na variável N.
- Comandos como este, que você eventualmente julgue conveniente incluir na sua linguagem, podem ser a ela acrescentados, para isso modificando-se adequadamente a gramática, com os cuidados necessários para que as eventuais alterações realizadas não deteriorem a especificação da linguagem.

Exercício (Faça-o!)

- Programe em BASIC alguns algoritmos envolvendo números inteiros. Esses algoritmos serão posteriormente usados para alimentar o compilador a ser desenvolvido.
- Evite reaproveitar programas já prontos, porque você precisa exercitar-se para dominar muito bem todos os detalhes da linguagem que deverá implementar.
- Use algum interpretador BASIC que esteja disponível online ou para baixar na internet, e faça os ajustes necessários para testar o funcionamento do programa que você construiu acima.
- Colecione este programa, bem como outros que queira implementar para se familiarizar melhor com a linguagem. Eles serão úteis para testar o funcionamento do compilador que você vai construir.