PERTEMUAN 5

NOTASI BNF (BACKUS NAUR FORM)

A. TUJUAN PEMBELAJARAN

Setelah mempelajari materi ini diharapkan mahasiswa mampu mengaplikasikan notasi BNF.

B. URAIAN MATERI

1. Sejarah Backus Naur Form

Ide menggambarkan struktur bahasa menggunakan aturan penulisan ulang dapat ditelusuri kembali setidaknya kepada karya *Pāṇini*, seorang ahli tata bahasa Sanskerta India kuno dan seorang sarjana agama Hindu yang dihormati yang hidup antara abad ke-6 dan ke-4 SM. Notasinya untuk mendeskripsikan struktur kata Sansekerta memiliki kekuatan yang setara dengan Backus dan memiliki banyak sifat yang serupa.

Dalam masyarakat Barat, tata bahasa telah lama dianggap sebagai subjek pengajaran (bukan studi ilmiah), deskripsi bersifat informal dan ditargetkan pada penggunaan praktis. Pada paruh pertama abad ke-20, ahli bahasa seperti Leonard Bloomfield dan Zellig Harris mulai mencoba memformalkan deskripsi bahasa, termasuk struktur frasa.

Sementara itu, aturan penulisan ulang string sebagai sistem logika formal diperkenalkan dan dipelajari oleh matematikawan seperti Axel Thue (tahun 1914), Emil Post (1920s-40s) dan Alan Turing (1936). Noam Chomsky, mengajar linguistik kepada mahasiswa teori informasi di MIT, menggabungkan linguistik dan matematika dengan mengambil apa yang pada dasarnya adalah formalisme Thue sebagai dasar untuk mendeskripsikan sintaks bahasa alami. Dia juga memperkenalkan perbedaan yang jelas antara aturan generatif (aturan tata bahasa bebas konteks) dan aturan transformasi (1956).

BNF diusulkan pada sekitar tahun 1959 untuk menggambarkan sintaks *Algol 60* oleh John Backus, salah satu dari tiga belas orang-orang di komite *Algol 60*. (John Backus dari IBM juga salah satunya dari tokoh-tokoh utama

yang bertanggung jawab untuk FORTRAN.) Karena modifikasinya dan menggunakan BNF secara ekstensif sebagai editor laporan *Algol 60*, Peter Naur dari Universitas Kopenhagen juga terkait dengannya. Ide-ide ditemukan secara independen sebelumnya oleh Noam Chomsky, seorang ahli bahasa di tahun 1956. BNF dan ekstensinya telah menjadi alat standar untuk mendeskripsikan sintaksis notasi pemrograman, dan dalam banyak kasus bagian dari kompilers dibuat secara otomatis dari deskripsi BNF.

2. Pengertian Backus Naur Form

BNF (Backus Naur Form) adalah *bahasa meta* yang dapat digunakan untuk mendefinisikan struktur leksikal dan sintaksis bahasa lain. Pada umumnya mempelajari bahasa BNF bertujuan untuk mendefinisikan bahasa baru, hal pertama sebelum menggunakan notasi BNF untuk mendefinisikan bahasa Inggris sederhana, mari kita pelajari terlebih dahulu definisi BNF itu sendiri. Pertama-tama kami akan menggunakan BNF untuk mendefinisikan bahasa Inggris sederhana yang kami kenal, dan kemudian kami akan mempelajari BNF dari definisi itu sendiri.

Backus-Naur Form, yang juga dikenal dengan istilah Backus Normal Form, merupakan sebuah *metasyntax* untuk menjelaskan context-free grammars. Istilah BNF diambil dari dua nama, yaitu John Backus dan Peter Naur. BNF banyak digunakan sebagai notasi untuk mendeskripsikan grammar dari bahasa pemrograman.

BNF menggunakan seperangkat aturan *derivasi* untuk mendeskripsikan sintaks suatu bahasa. Mari kita mulai dengan sebuah contoh. Berikut deskripsi BNF dari bahasa ekspresi kecil yang hanya menyertakan bilangan bulat dan penambahan:

e ::= i | e + e

i ::= <integers>

Aturan ini mengatakan bahwa ekspresi e adalah bilangan bulat i, atau dua ekspresi dengan simbol + yang muncul di antara keduanya. Sintaks "integers" dibiarkan tidak ditentukan oleh aturan ini.

Setiap aturan memiliki bentuknya

metavariable ::= symbols | ... | symbols

Metavariabel adalah variabel yang digunakan dalam aturan BNF, bukan variabel dalam bahasa yang dijelaskan. Simbol (::=) dan (|) yang muncul dalam aturan adalah metasyntax: Sintaks BNF digunakan untuk mendeskripsikan sintaks bahasa. Simbol adalah urutan yang dapat menyertakan metavariabel (seperti i dan e) serta token bahasa (seperti +). Spasi kosong tidak relevan dalam aturan ini.

Tabel 5.1 Simbol-simbol yang dipakai dalam notasi BNF:

::=	Identik dengan simbol"→" dalam aturan produksi
|	Sama dengan atau
< >	Mengapit simbol variabel/non-terminal
{ }	Pengulangan 0 s/d n kali

Tabel 5.2 Aturan Produksi dan Notasi BNF-nya

Aturan produksi	Notasi BNF
S → A | A + B | A – B A → B	S ::= <A> | <A> + <B> | <A> – <B> A ::= B

Contoh dari notasi BNF

<kalimat>	:: = <subyek><kata kerja><obyek>
<subyek>	:: = <kata benda> | <artikel><kata benda> | <kata sifat> <kata benda> | <artikel><kata sifat><kata benda>
<kata sifat>	:: = < kata sifat > | < kata sifat >< kata sifat >
<obyek>	:: = < subyek >
< kata benda >	:: = meja | kuda | komputer
<artikel>	:: = **itu** | **sebuah**
<kata sifat>	:: = **besar** | **cepat** | **bagus** | **tinggi**
<kata kerja>	:: = **adalah** | **membuat**

Dalam definisi, simbol ":: =" berarti nama di sisi kiri ditentukan oleh ekspresi di sisi kanan. Nama dalam sepasang tanda kurung siku "< >" adalah non-terminal, yang berarti bahwa nama tersebut perlu didefinisikan lebih lanjut. Bilah garis vertical "|" bisa disebut dengan "atau". Nama yang dicetak tebal adalah terminal, yang berarti bahwa nama tersebut tidak perlu didefinisikan lebih lanjut. Karena mereka membentuk kosakata bahasa.

BNF digunakan untuk menggambarkan angka, konstanta integer, dan ekspresi aritmatika yang disederhanakan. Kami akan memperkenalkan BNF dengan menggunakannya untuk menggambarkan angka, konstanta integer, dan ekspresi aritmatika yang disederhanakan.

Dalam BNF, fakta bahwa 1 adalah digit yang dinyatakan dengan

<digit> ::= 1

Istilah <digit> dipisahkan oleh tanda kurung siku untuk membantu menunjukkan bahwa itu tidak dapat muncul dalam "kalimat" dari bahasa yang sedang dijelaskan, tetapi hanya digunakan untuk membantu menjelaskan kalimat. Ini adalah "entitas sintaksis", seperti "kata kerja" atau "frase kata benda" dalam bahasa Inggris. Ini biasanya disebut simbol non-terminal, atau non-terminal. Simbol sebelah kanan atau "1", di sisi lain dapat muncul dalam kalimat dari bahasa yang dijelaskan, dan disebut terminal.

Dua aturan dapat digunakan untuk menunjukkan <digit> bisa jadi 0 atau 1:

<digit> :: = 0 (<digit> terdiri dari 0)

<digit> :: = 1 (<digit> terdiri dari 1)

Kedua aturan ini, yang mengekspresikan bentuk berbeda untuk non-terminal yang sama, dapat disingkat menggunakan simbol vertical " | ", dibaca sebagai "atau",

<digit> :: = 0 | 1 (<digit> dapat terdiri dari 0 atau 1)

Singkatan ini dapat digunakan untuk menentukan semua angka:

<digit> :: = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Dimana <digit> dapat terdiri dati angka 0 sampai 9

Konstanta integer adalah urutan (terbatas) dari satu atau lebih digit. Menggunakan <constant> non-terminal untuk mewakili kelas konstanta integer, konstanta integer didefinisikan secara rekursif sebagai berikut:

<constant> ::= <digit>

<constant> ::= <constant> <digit>

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Aturan pertama dibaca sebagai berikut: <constant> dapat terdiri dari <digit>. Aturan kedua dibaca sebagai berikut: <constant> dapat terdiri dari <constant> lain yang diikuti oleh <digit>.

Setelah kita memiliki pemahaman dasar tentang BNF, kita dapat menggunakannya untuk mendefinisikan suatu bahasa pemrograman kecil. Lima baris pertama mendefinisikan struktur leksikal, dan sisanya mendefinisikan struktur sintaksis bahasa.

<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
<digit>	::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<symbol>	::= _ | @ | . | ~ | ? | # | $
<char>	::= <letter> | <digit> | <symbol>
<operator>	::= + | - | * | / | % | < | > | == | <= | >= | and | or | not

```
<identifier>   ::= <letter> | <identifier><char>
<number>    ::= <digit> | <number><digit>
<item>   ::= <identifier> | <number>
<expression>::=  <item>  |  (<expression>)  |  <expression>  <operator>
               <expression>
<branch>    :: = if <expr>then {<block>} | if <expr>then {<block>} else
               {<block>}
<switch>      :: = switch<expr>{<sbody>}
<sbody>:: = <cases> | <cases>; default :<block>
<cases>       ::= case <value>:<block> | <cases> ; case <value>:<block>
<loop>   ::= while <expr> do {<block>}
<assignment> ::= <identifier>=<expression>;
<statement>  ::= <assignment> | <branch> | <loop>
<block>  ::= <statement> | <block>;<statement>
```

Dengan definisi diatas kita dapat memeriksa mana dari pernyataan berikut ini yang benar secara sintaksis. Sekarang kita menggunakan definisi untuk memeriksa mana dari pernyataan berikut yang benar secara sintaksis.

```
1     suml = 0;
2     while suml <= 100 do {
3     suml = suml + (al + a2) * (3b % 4*b); }
4     if suml == 120 then 2sum – sum1 else sum2 + suml;
5     p4#rd_2 = ((la + a2) * (b3 % b4)) / (c7 - c8);
6     _foo.bar = (al + a2 - b3 - b4);
7     (a1/ a2) = (c3 - c4);
```

Penjelasan dari kotak diatas :

Menurut definisi bahasa BNF, pernyataan 1 dan 2 benar. Pernyataan 3 dan 4 salah karena 3b dan 2sum bukanlah pengidentifikasi yang dapat diterima

atau ekspresi yang dapat diterima. Pernyataan 5 salah. Pernyataan 6 tidak benar karena pengenal harus dimulai dengan huruf. Pernyataan 7 tidak benar karena sisi kiri pernyataan penugasan harus menjadi pengenal.

3. Tata bahasa untuk ekspresi aritmatika (yang disederhanakan)

Perhatikan bagaimana menulis tata bahasa untuk ekspresi aritmatika yang menggunakan penjumlahan, pengurangan biner, perkalian, ekspresi dalam tanda kurung, dan konstanta integer sebagai operan. Ini cukup mudah dilakukan:

<expr> :: = <expr> + <expr>

<expr> :: = <expr> - <expr>

<expr> :: = <expr> * <expr>

<expr> :: = (<expr>)

<expr> :: = <constant>

Berikut adalah turunan dari ekspresi (1 + 3) * 4 menurut tata bahasa ini:

<expr> => <expr> * <expr>

=> (<expr>) * <expr>

=> (<expr> + <expr>) * <expr>

=> (<constant> + <expr>) * <expr>

=> (<constant> + <constant>) * <expr>

=> (<constant> + <constant>) * <constant>

=> (<digit> + <constant>) * <constant>

=> (<digit> + <digit>) * <constant>

=> (<digit> + <digit>) * <digit>

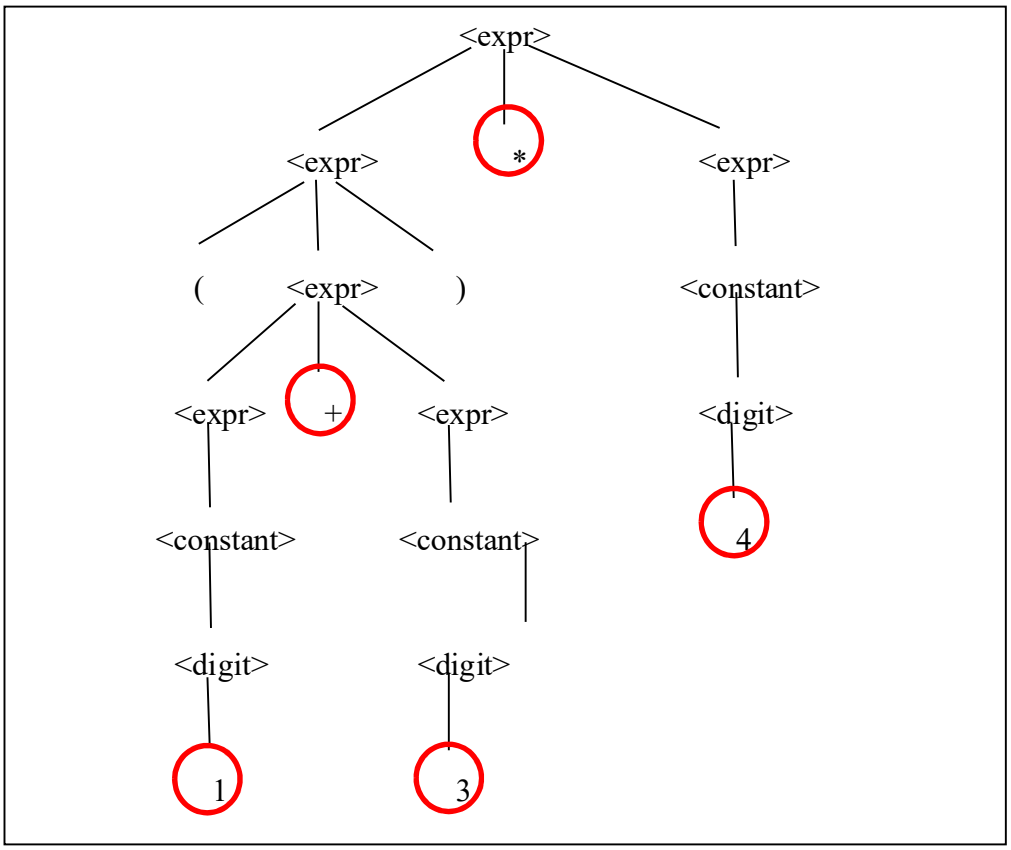
=> (1 + <digit>) * <digit>

=> (1 + 3) * <digit>

=> (1 + 3) * 4

4. Pohon Sintaks dan Ambiguitas

Pohon sintak yaitu metode urutan penurunan dengan cara seperti pohon dimana sebuah akar menjalar ke tangkai hingga sampai pada pucuknya dan menghasilkan buah. Urutan penurunannya dapat dijelaskan sebagai berikut,



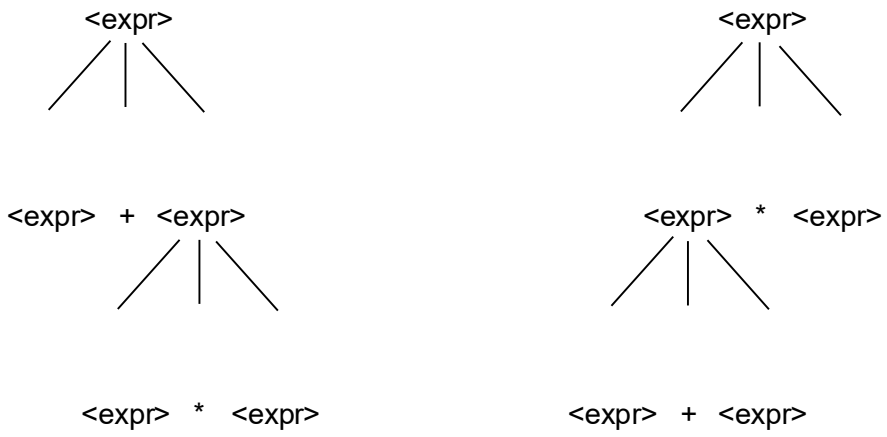
misalnya pohon sintaks untuk devirasi No.3 diatas adalah

Dalam pohon sintaks, turunan tunggal menggunakan aturan U ::= u diekspresikan oleh simpul U dengan garis-garis yang turun ke simbol-simbol deret u. Jadi, untuk setiap derivasi tunggal dalam deretan derivasi terdapat non-terminal dalam pohon, dengan simbol yang menggantikannya di bawahnya. Misalnya, turunan pertama adalah <expr> => <expr> * <expr>, jadi di bagian atas diagram di atas adalah node <expr> dan node ini memiliki garis yang memancar ke bawah dari ke <expr>, * dan <expr>. Juga, ada turunan menggunakan aturan <digit> :: = 1, jadi ada cabang yang sesuai dari <digit> ke 1 di pohon.

Perbedaan utama antara derivasi dan pohon sintaks adalah bahwa pohon sintaks tidak menentukan urutan pembuatan beberapa devirasi. Misalnya, di pohon yang diberikan di atas, tidak dapat ditentukan apakah aturan <digit> :: =

1 digunakan sebelum atau sesudah aturan <digit> :: = 3. Untuk setiap turunan ada pohon sintaks, tetapi lebih dari satu derivasi bisa sesuai dengan pohon yang sama. Derivasi ini dianggap setara.

Sekarang pertimbangkan himpunan turunan yang diekspresikan oleh <expr> => <expr> + <expr> * <expr>. Sebenarnya ada dua pohon derivasi berbeda untuk dua derivasi dari <expr> + <expr> * <expr>:



Tata bahasa yang memungkinkan lebih dari satu pohon sintaks untuk beberapa kalimat yang dibuat disebut ambigu. Karena keberadaan dua pohon sintaks yang memungkinkan kita untuk "mengurai" kalimat dalam dua cara berbeda, dan karena itu mungkin memberikan dua arti padanya. Dalam kasus ini, ambiguitas menunjukkan bahwa tata bahasa tidak menunjukkan apakah simbol + harus dilakukan sebelum atau sesudah simbol *. Pohon sintaks di sebelah kiri (atas) menunjukkan bahwa simbol + harus dilakukan terlebih dahulu, karena <expr> dari mana ia diturunkan karena merupakan operan dari operator penjumlahan (+). Di sisi lain, pohon sintaks di sebelah kanan menunjukkan bahwa simbol * harus dilakukan terlebih dahulu.

Seseorang dapat menulis tata bahasa yang tidak ambigu yang menunjukkan bahwa dalam algoritma perkalian lebih diutamakan daripada penjumlahan (kecuali jika tanda kurung digunakan untuk mengganti yang diutamakan). Untuk melakukan hal ini dibutuhkan pengenalan simbol non-terminal baru yaitu <term> dan <factor>, dengan contoh sebagai berikut:

<expr> ::= <term> | <expr> + <term> | <expr> - <term>

<term> ::= <factor> | <term> * <factor>

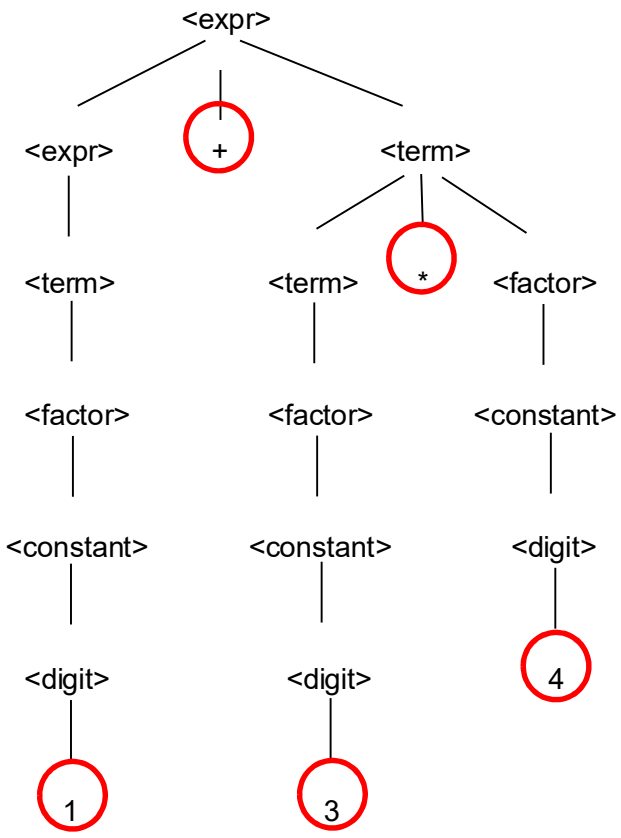
<factor> ::= <constant> | (<expr>)

<constant> ::= <digit>

<constant> ::= <constant><digit>

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Dalam tata bahasa ini, setiap kalimat memiliki satu pohon sintaks, sehingga tidak ada ambiguitas. Dengan contoh dibawah ini saya membuat kalimat 1 + 3 * 4 yang memiliki satu pohon sintaks:



Pohon sintaks diatas menunjukkan bahwa perkalian harus dilakukan terlebih dahulu, penulisan tata bahasa ini secara umum * diutamakan daripada + kecuali jika prioritas diganti dengan menggunakan tanda kurung.

5. Ekstensi ke BNF

Beberapa ekstensi BNF digunakan untuk membuatnya lebih mudah dibaca dan dipahami. Salah satu hal terpenting untuk menunjukkan pengulangan adalah dengan penggunaan tanda kurung kurawal: {x} menunjukkan nol atau lebih kemunculan dari urutan simbol x. Dengan ekstensi ini, kita bisa mendeskripsikan <constant> menggunakan satu aturan sebagai

<constant> :: = <digit> {<digit>}

Bahkan, tata bahasa untuk ekspresi aritmatika dapat ditulis ulang sebagai

<expr> ::= <term> {+ <term> | - <term>}

<term> ::= <factor> {* <factor>}

<factor> ::= <constant> | (<expr>)

<constant> ::= <digit> {<digit>}

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

6. Bentuk lain BNF

Ada banyak bentuk lain dan ekstensi dari BNF, baik demi kesederhanaan dan kekompakan, atau untuk beradaptasi ke aplikasi tertentu. Salah satu fitur umum dari banyak bantuk lain adalah penggunaan operator pengulangan ekspresi reguler seperti * dan +. Yang paling umum digunakan adalah extended Backus-Naur form (EBNF) atau dalam bahasa indonesia adala bentuk Backus-Naur yang diperluas.

Ekstensi lain yang umum adalah penggunaan tanda kurung persegi di sekitar item opsional. Meskipun tidak ada dalam laporan *ALGOL 60* asli (namun ekstensi ini diperkenalkan beberapa tahun kemudian dalam definisi PL / I IBM), notasi tersebut sekarang diakui secara universal.

Augmented Backus – Naur form (ABNF) dan Routing Backus – Naur form (RBNF) adalah ekstensi yang biasa digunakan untuk menjelaskan protokol Internet Engineering Task Force (IETF).

Banyak spesifikasi BNF yang ditemukan online saat ini dimaksudkan agar dapat dibaca oleh manusia dan tidak terlalu formal. Hal ini sering kali menyertakan banyak aturan dan ekstensi sintaks seperti berikut:

a. Item opsional yang diapit oleh tanda kurung siku: [<item-x>].
b. Item yang ada 0 kali atau lebih diapit tanda kurung kurawal atau diakhiri dengan tanda bintang (*) seperti <word> :: = <letter> {<letter>} atau <word> :: = <letter> <letter> * masing-masing .
c. Item yang ada 1 kali atau lebih diberi akhiran dengan simbol tambahan (+).
d. Terminal mungkin muncul dalam huruf tebal daripada miring, dan non-terminal dalam teks biasa daripada tanda kurung sudut.
e. Saat item dikelompokkan, item tersebut diapit dalam tanda kurung sederhana.

7. Backus Naur Form yang diperluas

Dalam bahasa inggris disebut sebagai Extended Backus-Naur Form (EBNF), adalah keluarga notasi metasyntax yang salah satunya dapat digunakan untuk mengekspresikan tata bahasa bebas konteks . EBNF digunakan untuk membuat deskripsi formal dari bahasa formal seperti bahasa pemrograman komputer. Mereka adalah ekstensi dari notasi metasyntax bentuk Backus-Naur (BNF) dasar.

EBNF dikembangkan oleh Niklaus Wirth yang menggabungkan beberapa konsep antara sintaks dan notasi yang berbeda dari notasi sintaks Wirth. Karena telalu banyak varian dari EBNF yang digunakan, maka organisasi Internasional untuk Standardisasi mengadopsi standar EBNF (ISO / IEC 14977) pada tahun 1996. Namun, menurut Zaytsev standar ini hanya menambahkan tiga dialek lagi ke dalam kekacauan dan, setelah mencatat kurangnya keberhasilan, juga mencatat bahwa ISO EBNF bahkan tidak digunakan di semua standar ISO. Wheeler menentang penggunaan standar ISO saat menggunakan EBNF, dan merekomendasikan untuk mempertimbangkan notasi EBNF alternatif seperti yang dari W3C Extensible Markup Language (XML) 1.0 (Fifth Edition).

EBNF adalah kode yang mengungkapkan tata bahasa formal. Sebuah EBNF terdiri dari simbol terminal dan aturan produksi non-terminal yang merupakan batasan yang mengatur bagaimana simbol terminal dapat

digabungkan menjadi urutan yang legal. Contoh simbol terminal termasuk karakter alfanumerik , tanda baca , dan karakter spasi.

EBNF mendefinisikan aturan produksi di mana urutan simbol masing-masing ditetapkan ke non-terminal :

digit tidak termasuk nol = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;

digit = "0" | digit tidak termasuk nol ;

Aturan produksi ini mendefinisikan digit non-terminal yang ada di sisi kiri tugas. Simbol vertikal (|) mewakili alternatif dan simbol terminal diapit dengan tanda kutip (“ ”) diikuti oleh titik koma (;) sebagai karakter pengakhir. Oleh karena itu digit adalah 0 atau digit tidak termasuk nol yang bisa 1 atau 2 atau 3 dan seterusnya sampai 9 .

Aturan produksi yang juga dapat mencakup urutan terminal atau non-terminal yaitu masing-masing dipisahkan oleh koma, seperti dibawah ini:

dua belas = "1" , "2" ;

dua ratus satu = "2" , "0" , "1" ;

tiga ratus dua belas = "3" , dua belas ;

dua belas ribu dua ratus satu = dua belas , dua ratus satu ;

Ekspresi yang mungkin dihilangkan atau diulang dapat direpresentasikan melalui tanda kurung kurawal {...}:

bilangan asli = digit tidak termasuk nol , { digit } ;

Dalam hal ini, string *1* , *2* , ..., *10* , ..., *10000* , ... adalah ekspresi yang benar. Untuk mewakili ini, semua yang diatur dalam kurung kurawal dapat diulangi sesering mungkin, dan juga bisa tidak sama sekali.

Berikut ini adalah tabel simbol usulan standar ISO / IEC 14977, oleh RS Scowen

Pemakaian	Notasi
Definisi	=
Rangkaian	,
Penghentian	;
Alternasi	|
Pilihan	[...]
Pengulangan	{...}
pengelompokan	(...)
string terminal	"..."
string terminal	'...'
Komentar	(* ... *)
urutan khusus	? ...?
Pengecualian	-

8. Keuntungan EBNF dibanding BNF

Tata bahasa apa pun yang didefinisikan dalam EBNF juga dapat direpresentasikan dalam BNF, meskipun representasi dalam EBNF umumnya lebih panjang. Misalnya, opsi dan pengulangan tidak dapat secara langsung diekspresikan dalam BNF dan membutuhkan penggunaan aturan perantara atau produksi alternatif yang didefinisikan sebagai tidak ada atau produksi opsional untuk opsi, atau produksi berulang itu sendiri, secara rekursif, untuk pengulangan. Konstruksi yang sama masih dapat digunakan dalam EBNF.

BNF menggunakan simbol (< , > , | , ::=) untuk dirinya sendiri, tetapi tidak menyertakan tanda kutip di sekitar string terminal. Hal ini mencegah

karakter ini digunakan dalam bahasa, dan memerlukan simbol khusus untuk string kosong. Dalam EBNF, terminal diapit dengan tanda petik (" … " atau ' … '). Tanda kurung sudut (" < … > ") untuk non-terminal dapat dihilangkan.

Sintaks BNF hanya dapat merepresentasikan sebuah rule dalam satu baris, sedangkan pada EBNF sebuah karakter terminating yaitu karakter titik koma “ ; ”Menandai akhir dari suatu aturan.

Lebih lanjut, EBNF mencakup mekanisme untuk penyempurnaan, menentukan jumlah pengulangan, tidak termasuk alternatif, komentar, dll.

C. SOAL LATIHAN/TUGAS

1. Terdapat aturan produksi dibawah ini, lalu buatlah notasi BNF nya

   E → A * B | A / B | C

   T → abc

2. Buatlah pohon sintaks dari aturan berikut 15 + 8 *4 - 21:

   <expr> ::= <term> | <expr> + <term> | <expr> - <term>

   <term> ::= <factor> | <term> * <factor>

   <factor> ::= <constant> | (<expr>)

   <constant> ::= <digit>

   <constant> ::= <constant><digit>

   <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

3. Dari soal nomor 2 apakah pohon sintaks tersebut termasuk ambiguitas atau bukan? Jelaskan!

D. REFERENSI

Yinong Chen. *Introduction to Programming Languages (Programming in C, C++, Scheme, Prolog, C#, and SOA).* Arizona State University. Edisi 5

Edward Cohen. (1990). *Programming in the 1990s: An Introduction to the Calculation of Programs*. Cornell University:USA

Hari Soetanto. (2004). *Teknik Kompilasi.* Universitas Budi Luhur:Indonesia

Michael Sipser. (1997). *Introduction to The theory of computation.* PWS publishing company

Wikiwand. *Backus Naur Form.* (diakses pada 25 Oktober 2020). Tersedia pada http://www.wikiwand.com/en/Backus%E2%80%93Naur_form

Wikipedia. *Backus-Naur form..* (diakses pada 24 Oktober 2020). Tersedia pada https://en.wikipedia.org/wiki/Backus%E2%80%93Naur_form

Wikipedia. *Extended Backus-Naur form..* (diakses pada 26 Oktober 2020). Tersedia pada https://en.m.wikipedia.org/wiki/Extended_Backus–Naur_form

GLOSARIUM

ALGOL 60 (singkatan dari Algorithmic Language 1960) adalah anggota dari keluarga ALGOL bahasa pemrograman komputer yang berdiri pada tahun 1958.

Metasyntax menggambarkan struktur dan komposisi frasa dan kalimat yang diijinkan dari bahasa logam, yang digunakan untuk menggambarkan bahasa alami atau bahasa pemrograman komputer.

Derivasi adalah adalah proses pembentukan kata yang menghasilkan leksem baru (menghasilkan kata-kata yang berbeda dari paradigma yang berbeda).

Bahasa meta atau metabahasa adalah bahasa atau perangkat lambang yang dipakai untuk menguraikan bahasa, yang disebut bahasa objek. Metabahasa dapat berupa istilah atau bahasa apa pun yang digunakan untuk membahas tentang bahasa, misalnya tata bahasa tertulis.

Representatif adalah istilah yang mengacu pada kata perwakilan atau mewakili.