BAB II
ANALISA LEKSIKAL (SCANNER)

TUJUAN PRAKTIKUM

1) Memahami bahasa sumber.
2) Memahami dan mengerti tugas analisa leksikal.
3) Memahami dan mengerti membangun alisa leksikal.

TEORI PENUNJANG

2.1 Bahasa Sumber

Bahasa adalah kumpulan kalimat. Kalimat adalah rangkaian kata. Kata adalah unit terkecil komponen bahasa yang tidak bisa dipisah-pisahkan lagi. Kalimat-kalimat : '*Seekor kucing memakan seekor tikus.*' dan '*Budi menendang sebuah bola.*' adalah dua contoh kalimat lengkap Bahasa Indonesia. '*A cat eats a mouse*' dan '*Budi kick a ball.*' adalah dua contoh kalimat lengkap Bahasa Inggeris. '*if a2 < 9.0 then b2 := a2+a3;*' dan '*for i := start to finish do A[i] := B[i]*sin(i*phi/16.0).*' adalah dua contoh kalimat lengkap dalam Bahasa Pemrograman Pascal. Dalam bahasa pemrograman *kalimat* lebih dikenal sebagai *ekspresi* sedangkan *kata* sebagai *token*.

Perancangan sebuah bahasa harus memperhatikan tiga aspek berikut :

1. *spesifikasi leksikal*, misalnya setiap kata harus tersusun atas huruf mati dan huruf hidup yang disusun bergantian, atau setiap token harus dimulai dengan huruf dan selanjutnya boleh diikuti oleh huruf atau angka.
2. *spesifikasi sintaks*, misalnya setiap kalimat mengikuti pola *subyek-predikat-obyek* atau ekspresi *for_do* mengikuti pola *for*-identifier-*:=*-identifier-*to*-identifier-*do*-ekspresi.
3. *aturan-aturan semantik*, misalnya kata yang mendahului kata kerja haruslah kata benda yang menggambarkan sesuatu yang hidup dan berkaki, atau operasi perkalian hanya bisa dilakukan antara dua operan dengan tipe yang sama.

Berdasarkan rancangan bahasa di atas, perhatikan hal-hal berikut. Kita tidak bisa mengganti kata *Budi* dengan *2udi* sebagaimana kita tidak bisa mengganti token *start* dengan *?tart*. Kita juga tidak bisa merubah susunan kata-kata menjadi *Budi sebuah menendang bola* sebagaimana kita tidak boleh merubah susunan token-token menjadi *9.0 if < a2 then b2:= a2*. Demikian pula kita tidak boleh mengganti kata *Budi* dengan *lemari* sebagaimana kita tidak boleh mengganti *B[i]*sin(i*pi/16.0)* dengan *B*sin(i*pi/16.0)*.

Dalam spesifikasi leksikal biasanya digunakan *grammar regular* (GR) dalam bentuk *ekspresi regular* (ER). Sebagai contoh pola token *identifier* ditentukan oleh *grammar regular* berikut :

$I \rightarrow aA \mid bA \mid ... \mid zA \mid a \mid b \mid ... \mid z$, $A \rightarrow aA \mid bA \mid ... \mid zA \mid 0A \mid 1A \mid ... \mid 9A \mid a \mid b \mid ... \mid z \mid 0 \mid 1 \mid ... \mid 9$

yang ekuivalen dengan *ekspresi regular* berikut :

$I = (a \mid b \mid ... \mid z)(a \mid b \mid ... \mid z \mid 0 \mid 1 \mid ... \mid 9)^* = huruf(huruf \mid angka)^*$

Dalam spesifikasi sintaks biasanya digunakan *context free grammar* (CFG). Sebagai contoh ekspresi *if-then* E adalah :

$E \rightarrow$ *if* L *then*, $L \rightarrow IOA$, $I = huruf(huruf \mid angka)^*$, $O \rightarrow < \mid = \mid > \mid <= \mid >=$, $A \rightarrow 0 \mid 1 \mid ... \mid 9$.

2.2 Analisa Leksikal (Scanner)

Dalam kaitan ini aliran karakter yang membentuk program sumber dibaca dari kiri ke kanan dan dikelompokkan dalam apa yang disebut token yaitu barisan dari karakter yang dalam suatu kesatuan mempunyai suatu arti tersendiri. Analisa ini melakukan penerjemahan masukan menjadi bentuk yang lebih berguna untuk tahap-tahap kompilasi berikutnya.

Analisa Leksikal merupakan antarmuka antara kode program sumber dan analisa sintaktik (parser). Scanner melakukan pemeriksaan karakter per karakter pada teks masukan, memecah sumber program menjadi bagian-bagian disebut Token. Analisa Leksikal mengerjakan pengelompokkan urutan-urutan karakter ke dalam komponen pokok: identifier, delimeter, simbol-simbol operator, angka, keyword, noise word, blank, komentar, dan seterusnya menghasilkan suatu Token Leksikal yang akan digunakan pada Analisa Sintaktik. Model dasar untuk membentuk suatu Analisa Leksikal adalah Finite-State Automata.

Dua aspek penting pembuatan Analisa Leksikal adalah :

- Menentukan token-token bahasa.
- Mengenali token-token bahasa dari program sumber.

Token-token dihasilkan dengan cara memisahkan program sumber tersebut dilewatkan ke parser. Analisa Leksikal harus mengirim token ke parser. Untuk mengirim token, scanner harus mengisolasi barisan karakter pada teks sumber yang merupakan 1 token valid. Scanner juga menyingkirkan informasi seperti komentar, blank, batas-batas baris dan lain-lain yang tidak penting (tidak mempunyai arti) bagi parsing dan Code Generator.

Scanner juga harus dapat mengidentifikasi token secara lengkap dan membedakan keyword dan identifier. Untuk itu scanner memerlukan tabel simbol. Scanner memasukkan identifier ke tabel simbol, memasukkan konstanta literal dan numerik ke tabel simbol sendiri setelah konversi menjadi bentuk internal.

Analisa Leksikal merupakan komponen kompilasi independen yang berkomunikasi dengan parser lewat antarmuka yang terdefinisi bagus dan sederhana sehingga pemeliharaan analisa leksikal menjadi lebih mudah dimana perubahan-perubahan terhadap analisa leksikal tidak berdampak pada pengubahan kompilator secara keseluruhan. Agar dapat memperoleh fitur ini, maka antarmuka harus tidak berubah. Kebanyakan kode yang menyusun analisa leksikal adalah sama untuk seluruh kompilator, tidak peduli bahasa.

Pada analisa leksikal yang dituntun tabel (table-driven lexical analyzer), maka satu-satunya yang berubah adalah tabel itu sendiri. Kadang diperlukan interaksi analisa leksikal dan analisa sintaktik yang lebih kompleks. Sehingga analisa leksikal harus dapat menganggap string sebagai token bertipe, bukan identifier. Untuk itu perlu komunikasi tingkat lebih tinggi yang biasanya dilakukan suatu struktur data dipakai bersama seperti tabel simbol.

Analisa Sintaktik dapat memasukkan string ke tabel simbol, mengidentifikasi sebagai **Type** atau **typedef**, sehingga analisa leksikal dapat memeriksa tabel simbol untuk menentukan apakah lexeme adalah tipe token atau identifier.

2.3 Tugas Analisa leksikal

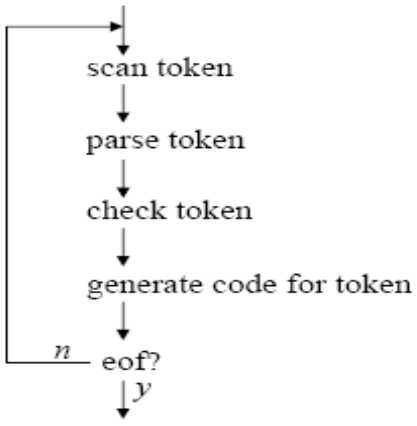
Tugas – tugas analisa leksikal antara lain :

a. Melakukan pembacaan kode sumber dengan merunut karakter demi karakter.
b. Mengenali besaran leksik (identifier, keywords, dan konstanta).
c. Mentransformasi menjadi sebuah token dan menentukan jenis tokennya.
d. Mengirimkan token.
e. Membuang atau mengabaikan *white-space* dan komentar dalam program.
f. Menangani kesalahan.
g. Menangani tabel simbol.

2.4 Tahap Pelaksanaan Analisa Leksikal

- Pada single one pass

  Terjadi interaksi antara scanner dan parser. Scanner dipanggil saat parser memerlukan token berikutnya. Pendekatan ini lebih baik karena bentuk internal program sumber yang lengkap tidak perlu dibangun dan disimpan di memori sebelum parsing dimulai.

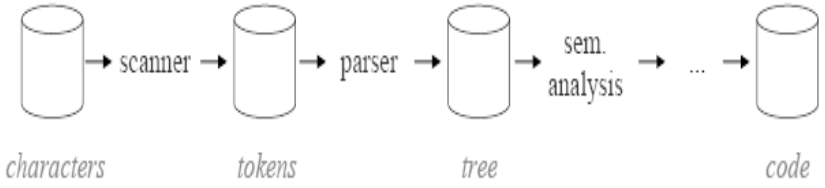


Gambar 2.1. Skema Single One Pass

- Pada separate pass / multi pass

  Scanner memproses secara terpisah, dilakukan sebelum parsing. Hasil scanner disimpan dalam file. Dari file tersebut, parsing melakukan kegiatannya. Scanner mengirim nilai-nilai integer yang mempresentasikan bentuk internal token, bukan nilai-nilai string. Keunggulan cara ini adalah ukurannya kecil dan tetap. Parser sangat lebih

efisien bekerja dengan nilai integer yang mempresentasikan simbol daripada string nyata dengan panjang variabel.



Gambar 2.2. Skema Separate Pass

2.5 Implementasi Analisa Leksikal

a. Pengenalan Token
   - Scanner harus dapat mengenali token
   - Terlebih dahulu dideskripsikan token-token yang harus dikenali

b. Pendeskripsian Token
   - Menggunakan reguler grammar. Menspesifikasikan aturan-aturan pembangkit token-token dengan kelemahan reguler grammar menspesifikasikan token berbentuk pembangkit, sedang scanner perlu bentuk pengenalan.
   - Menggunakan ekspresi grammar. Menspesifikasikan token-token dengan ekspresi reguler.
   - Model matematis yang dapat memodelkan pengenalan adalah finite-state acceptor (FSA) atau finite automata.

c. Implementasi Analisa Leksikal sebagai Finite Automata

   Pada pemodelan analisa leksikal sebagai pengenal yang menerapkan finite automata, analisa leksikal tidak cuma hanya melakukan mengatakan YA atau TIDAK. Dengan demikian selain pengenal, maka analisa leksikal juga melakukan aksi-aksi tambahan yang diasosiasikan dengan string yangsedang diolah. Analisa leksikal dapat dibangun dengan menumpangkan pada konsep pengenal yang berupa finite automata dengan cara menspesifikasikan rutin-rutin (aksi-aksi) tertentu terhadap string yang sedang dikenali.

d. Penanganan Kesalahan di Analisa Leksikal

Hanya sedikit kesalahan yang diidentifikasi di analisa leksikal secara mandiri karena analisa leksikal benar-benar merupakan pandangan sangat lokal terhadap program sumber. Bila ditemui situasi dimana analisa leksikal tidak mampu melanjutkan proses karena tidak ada pola token yang cocok, maka terdapat beragam alternatif pemulihan, yaitu:

- "Panic mode" dengan menghapus karakter-karakter berikutnya sampai analisa leksikal menemukan token yang terdefinisi bagus
- Menyisipkan karakter yang hilang
- Mengganti karakter yang salah dengan karakter yang benar
- Mentransposisikan 2 karakter yang bersebelahan.

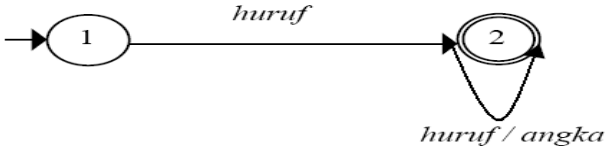
Salah satu cara untuk menemukan kesalahan-kesalahan di program adalah menghitung jumlah transformasi kesalahan minimum yang diperlukan untuk mentransformasikan program yang salah menjadi program yag secara sintaks benar.

2.6 Input Buffering

Perancangan analisa leksikal seharusnya dapat membuat buffering masukkan yang membantu mempercepat proses pembacaan dari file serta mempunyai fleksibelitas yang tinggi agar analisa leksikal tidak bergantung platform sehingga mempunyai portabilitas yang tinggi.

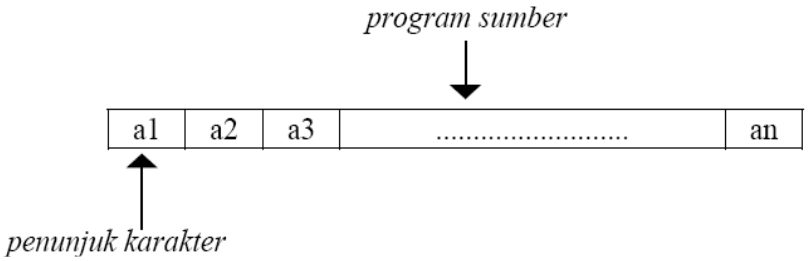
2.7 Membangun Analisa Leksikal

Scanner diimplementasikan dengan *Automata Hingga Deterministik* (AHD). Pada kuliah *Teori Bahasa dan Automata* (atau *Pengantar Automata, Bahasa Formal, dan Kompilasi*) telah dipelajari siklus transformasi : GR $\rightarrow$ ER $\rightarrow$ AHN $\rightarrow$ AHD $\rightarrow$ GR. Sebagai contoh, scanner (yaitu AHD) untuk mengenali identifier adalah :



Gambar 2.3. Skema Scanner

2.7.1 Membaca Program Sumber



2.7.2 Aturan Translasi

Berikut ini adalah contoh aturan translasi (*translation rule*) untuk beberapa *ekspresi regular* (ER) atau token.

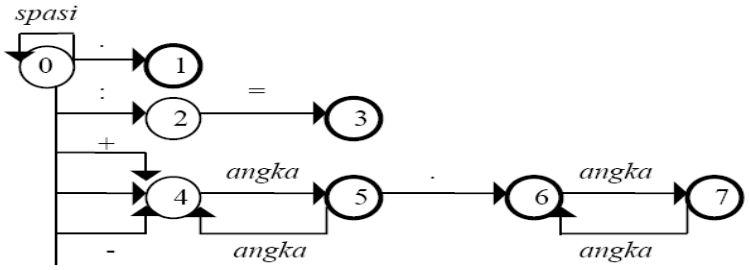
Tabel 2.1 Tabel Aturan Translasi

Token	Aturan Translasi	Token	Aturan Translasi
.	{Current_Token(1,1)}	=	{Current_Token(15,1)}
,	{Current_Token(2,1)}	< >	{Current_Token(15,2)}
;	{Current_Token(3,1)}	<	{Current_Token(15,3)}
:	{Current_Token(4,1)}	< =	{Current_Token(15,4)}
: =	{Current_Token(12,1)}	>	{Current_Token(15,5)}
+	{Current_Token(13,1)}	> =	{Current_Token(15,6)}
-	{Current_Token(13,2)}	identifier	{Current_Token(27,Id)}
*	{Current_Token(14,1)}	(+|-|Ɛ)angka$^+$	{Current_Token(28,IN)}
/	{Current_Token(14,2)}	(+|-|Ɛ)angka$^+$ .angka$^+$	{Current_Token(29,RN)}

Current_Token(tipe,nilai) adalah *procedure* yang memberikan spesifikasi kepada sebuah token yang baru saja ditemukan. Argumen *tipe* adalah *kelompok token* sedangkan sedangkan argumen *nilai* merupakan nilai dari token tersebut. Tipe = 0 ditetapkan bagi *token yang tidak dikenal*.

2.7.3 DFA dari ER dan Tabel Transisi

Contoh DFA (Deterministic Finite Automata) untuk beberapa ER (Expresion Regular) di atas adalah :



DFA di atas mempunyai *tabel transisi* T(stata,karakter) sebagai berikut :

Tabel 2.2. Tabel Transisi

stata	k a r a k t e r						
	spasi	.	:	=	+	-	angka
0	0	1	x	x	4	4	x
1	x	x	x	x	x	x	x
2	x	x	x	3	x	x	x
3	x	x	x	x	x	x	x
4	x	x	x	x	x	x	5
5	x	6	x	x	x	x	4
6	x	x	x	x	x	x	7
7	x	x	x	x	x	x	6

Formula tabel tersebut adalah : $T(i,a) = \begin{cases} j, \text{ jika ada transisi berlabel a dari i ke j} \\ x, \text{ jika tidak ada transisi berlabel a dari i} \end{cases}$

Jika T(i,a) = x, maka ada 2 kemungkinan :

- jika stata i merupakan stata akhir berarti pada stata i telah ditemukan sebuah token
- jika stata i bukan stata akhir berarti pada stata i telah terdeteksi *token tidak dikenal*

2.7.4 Simulasi Analisa Leksikal

Simulasi DFA dimaksudkan untuk mengenali token.

```
type Token_Kind = record
            tipe : byte;
            nilai : byte;
end;
var     Token : array[0..Max_State] of Token_Kind;
        Found_Token : Token_Kind; {token yang ditemukan}
        Tok_Pos : Text_Pos; {posisi token dalam program sumber}
procedure Next_Token(var Ft : text); {digunakan untuk mengenali
sebuah token}
var state1, state2 : shortint;
begin
   state1 := 0;
   Tok_Pos := Now_Pos;
   repeat
        state2 := Next_State(state1, character);
        if state2 <> -1 then {-1 bersesuaian dengan x pada
tabel transisi}
        begin
             state1 := state2;
             Next_Character(Ft);     {baca karakter berikut
pada program_sumber}
             {di antaranya menghasilkan nilai baru untuk
Now_Pos}
        end;
   until state2 = -1;
   Act_for_Token(state1);
end;
procedure Act_for_Token(state : shortint);
var Tok_Length : byte;
      Err : integer;
begin
   Current_Token(Token[state].tipe, Token[state].nilai);
   Tok_Length := Now_Pos.Char_Numb - Tok_Pos.Char_Numb;
   case Token[state].tipe of
        0     : Error('Token tidak dikenal!', Tok_Pos);
        27    : Id := copy(Line, Tok_Pos.Char_Num, Tok_Length);
```

```
            28    :  val(copy(Line,  Tok_Pos.Char_Num,  Tok_Length),
        IN, Err);
            29    :  val(copy(Line,  Tok_Pos.Char_Num,  Tok_Length),
        RN, Err);
          end
        end;
```

catatan :

- copy(string, start, length) mengembalikan substring
- val(string_value, number_variable, error_flag) :

  jika string_value = '137' maka number_variable = 137 dan error_flag = 0

  jika string_value = 'string' maka number_variable = 137 dan error_flag ≠ 0
- Token.tipe ∈ {1, 2, 3, ..., 26} dimisalkan bernilai pasti, sehingga tidak perlu penangan-an lebih lanjut

```
procedure Current_Token(tipe, nilai : byte);
begin
   Found_Token.tipe := tipe;
   Found_Token.nilai := nilai;
end;
```

LAPORAN PENDAHULUAN

1. Jelaskan apa yang anda ketahui mengenai bahasa sumber dalam teknik kompilasi!
2. Sebutkan dan jelaskan tugas dari analisa leksikal (Scanner)!
3. Sebutkan dan jelaskan tahapan dan teknik membangun analisa leksikal (Scanner)!

LAPORAN AKHIR

1. Buat program sederhana mengenai analisa leksikal beserta penjelasannya!