BAB I
PENGENALAN TEKNIK KOMPILASI

TUJUAN PRAKTIKUM

1) Memahami penggolongan Bahasa Pemrograman berdasarkan tingkat ketergantungannya dengan mesin.
2) Mengetahui dan memahami mengenai Translator.
3) Mengetahui definisi Kompilator (*Compililer*).
4) Mengetahui dan mengerti tahap-tahap kompilasi.

TEORI PENUNJANG

1.1 Teori Bahasa

Teori Otomata dan bahasa formal, berkaitan dalam hal pembangkitan kalimat / *generation* yaitu, menghasilkan *semua* kalimat dalam bahasa L berdasarkan aturan yang dimilikinya. Dan pengenalan kalimat / recognition yaitu, menentukan suatu string (kalimat) termasuk sebagai salah satu anggota himpunan L.

Teori bahasa membicarakan bahasa formal (*formal language*), terutama untuk kepentingan perancangan kompilator (*compiler*) dan pemroses naskah (*text processor*). Bahasa formal adalah kumpulan *kalimat*. Semua kalimat dalam sebuah bahasa dibangkitkan oleh sebuah tata bahasa (*grammar*) yang sama. Sebuah bahasa formal bisa dibangkitkan oleh dua atau lebih tata bahasa berbeda. Dikatakan bahasa formal karena grammar diciptakan mendahului pembangkitan setiap kalimatnya. Bahasa manusia bersifat sebaliknya; grammar diciptakan untuk meresmikan kata-kata yang hidup di masyarakat. Dalam pembicaraan selanjutnya ‘bahasa formal’ akan disebut ‘bahasa’ saja.

1.2 Automata

Arti menurut *American Heritage Dictionary* :

1. a robot
2. one that behaves in an automatic or mechanical fashion

Arti dalam dunia matematika

Berkaitan dengan teori mesin abstrak, yaitu mesin sekuensial yang menerima input, dan mengeluarkan output, dalam bentuk diskrit.

Contoh :

- Mesin Jaja / vending machine
- Kunci kombinasi
- Parser/compiler

Jika disimpulkan maka pengertian automata adalah mesin abstrak yang dapat mengenali (*recognize*), menerima (*accept*), atau membangkitkan (*generate*) sebuah kalimat dalam bahasa tertentu.

1.2.1 Beberapa Pengertian Dasar

- Simbol adalah sebuah entitas abstrak (seperti halnya pengertian *titik* dalam geometri). Sebuah huruf atau sebuah angka adalah contoh simbol.
- String adalah deretan terbatas (*finite*) simbol-simbol. Sebagai contoh, jika *a*, *b*, dan *c* adalah tiga buah simbol maka *abcb* adalah sebuah string yang dibangun dari ketiga simbol tersebut.
- Jika *w* adalah sebuah string maka panjang string dinyatakan sebagai $|w|$ dan didefinisikan sebagai cacahan (banyaknya) simbol yang menyusun string tersebut. Sebagai contoh, jika *w* = *abcb* maka $|w| = 4$.
- String hampa adalah sebuah string dengan nol buah simbol. String hampa dinyatakan dengan simbol ε (atau ^) sehingga $|\varepsilon| = 0$. String hampa dapat dipandang sebagai simbol hampa karena keduanya tersusun dari nol buah simbol.
- Alfabet adalah hinpunan hingga (*finite set*) simbol-simbol

1.3 Grammar dan Bahasa

1.3.1 Konsep Dasar

1. Dalam pembicaraan grammar, anggota alfabet dinamakan simbol terminal atau token.
2. Kalimat adalah deretan hingga simbol-simbol terminal.
3. Bahasa adalah himpunan kalimat-kalimat. Anggota bahasa bisa tak hingga kalimat.
4. Simbol-simbol berikut adalah simbol terminal :
   - huruf kecil awal alfabet, misalnya : a, b, c
   - simbol operator, misalnya : +, –, dan ×
   - simbol tanda baca, misalnya : (,), dan ;
   - string yang tercetak tebal, misalnya : **if**, **then**, dan **else**.
5. Simbol-simbol berikut adalah simbol non terminal :
   - huruf besar awal alfabet, misalnya : A, B, C
   - huruf S sebagai simbol awal
   - string yang tercetak miring, misalnya : *expr* dan *stmt*.
6. Huruf besar akhir alfabet melambangkan simbol terminal atau non terminal, misalnya : X, Y, Z.
7. Huruf kecil akhir alfabet melambangkan string yang tersusun atas simbol-simbol terminal, misalnya : x, y, z.
8. Huruf yunani melambangkan string yang tersusun atas simbol-simbol terminal atau simbol-simbol non terminal atau campuran keduanya, misalnya : α, β, dan γ.
9. Sebuah produksi dilambangkan sebagai $\alpha \rightarrow \beta$, artinya : dalam sebuah derivasi dapat dilakukan penggantian simbol α dengan simbol β.
10. Simbol α dalam produksi berbentuk $\alpha \rightarrow \beta$ disebut ruas kiri produksi sedangkan simbol β disebut ruas kanan produksi.
11. Derivasi adalah proses pembentukan sebuah kalimat atau sentensial. Sebuah derivasi dilambangkan sebagai : $\alpha \Rightarrow \beta$.
12. Sentensial adalah string yang tersusun atas simbol-simbol terminal atau simbol-simbol non terminal atau campuran keduanya.

13. Kalimat adalah string yang tersusun atas simbol-simbol terminal. Jelaslah bahwa kalimat adalah kasus khusus dari sentensial.
14. Pengertian terminal berasal dari kata *terminate* (berakhir), maksudnya derivasi berakhir jika sentensial yang dihasilkan adalah sebuah kalimat (yang tersusun atas simbol-simbol terminal itu).
15. Pengertian non terminal berasal dari kata *not terminate* (belum/tidak berakhir), maksudnya derivasi belum/tidak berakhir jika sentensial yang dihasilkan mengandung simbol non terminal.

1.3.2 Grammar dan Klasifikasi Chomsky

Grammar G didefinisikan sebagai pasangan 4 tuple : V_T, V_N, S, dan Q, dan dituliskan sebagai G(V_T, V_N, S, Q), dimana :

V_T : himpunan simbol-simbol terminal (atau himpunan token -token, atau alfabet)

V_N : himpunan simbol-simbol non terminal

$S \in V_N$: simbol awal (atau simbol start)

Q : himpunan produksi

Berdasarkan komposisi bentuk ruas kiri dan ruas kanan produksinya ($\alpha \rightarrow \beta$), Noam Chomsky mengklasifikasikan 4 tipe grammar :

1. Grammar tipe ke-0 : Unrestricted Grammar (UG)
   Ciri : $\alpha, \beta \in (V_T \mid V_N)^*, |\alpha| > 0$
2. Grammar tipe ke-1 : Context Sensitive Grammar (CSG)
   Ciri : $\alpha, \beta \in (V_T \mid V_N)^*, 0 < |\alpha| \leq |\beta|$
3. Grammar tipe ke-2 : Context Free Grammar (CFG)
   Ciri : $\alpha \in V_N, \beta \in (V_T \mid V_N)^*$
4. Grammar tipe ke-3 : Regular Grammar (RG)
   Ciri : $\alpha \in V_N, \beta \in \{V_T, V_T V_N\}$ **atau** $\alpha \in V_N, \beta \in \{V_T, V_N V_T\}$

Mengingat ketentuan simbol-simbol (hal. 3 no. 4 dan 5), ciri-ciri RG sering dituliskan sebagai : $\alpha \in V_N$, $\beta \in \{a, bC\}$ *atau* $\alpha \in V_N$, $\beta \in \{a, Bc\}$

Atau disederhanakan seperti tabel dibawah ini:

Tabel 1.1 Tabel Grammar Chomsky

Kelas	Ruas kiri	Ruas Kanan	Contoh
Regular	$\alpha \in N$	$\leq$ 1 non terminal (paling kiri/kanan)	$P \rightarrow aR$ $Q \rightarrow ab$ $R \rightarrow cc$
Context Free	$\alpha \in N$	-	$P \rightarrow aQb$ $Q \rightarrow abPRS$
Context Sensitive	$\alpha \in (T \cup N)^+$	$\lvert\alpha\rvert \leq \lvert\beta\rvert$	$aD \rightarrow Da$ $AD \rightarrow aCD$
Unrestricted	$\alpha \in (T \cup N)^+$	-	$CB \rightarrow DB$ $ADc \rightarrow \varepsilon$

Tipe sebuah grammar (atau bahasa) ditentukan dengan aturan sebagai berikut :

A language is said to be type-i (i = 0, 1, 2, 3) language if it can be specified by a type-i grammar but can't be specified any type-(i+1) grammar.

1.3.3 Mesin Pengenal Bahasa

Untuk setiap kelas bahasa Chomsky, terdapat sebuah mesin pengenal bahasa. Masing-masing mesin tersebut adalah :

Tabel 1.2 Tabel Kelas Bahasa dan Mesin Pengenal Bahasa

Kelas Bahasa	Mesin Pengenal Bahasa
Regular Grammar, RG	Automata Hingga (*Finite Automata*), FA

Context Free Gammar (CFG)	Automata Pushdown (*Pushdown Automata*), PDA
Context Sensitive Grammar (CSG)	*Linear Bounded Automaton*, LBA
Unrestricted Grammar (UG)	Mesin Turing (*Turing Machine*), TM

Catatan :

1. Pengenal bahasa adalah salah satu kemampuan mesin turing.
2. LBA adalah variasi dari Mesin Turing Nondeterministik.

1.3.4 Contoh Analisa Penentuan Tipe Grammar

1. Grammar G_1 dengan Q_1 = {S $\rightarrow$ aB, B $\rightarrow$ bB, B $\rightarrow$ b}. Ruas kiri semua produksinya terdiri dari sebuah V_N maka G_1 kemungkinan tipe CFG atau RG. Selanjutnya karena semua ruas kanannya terdiri dari sebuah V_T atau string $V_T V_N$ maka G_1 adalah RG.
2. Grammar G_2 dengan Q_2 = {S $\rightarrow$ Ba, B $\rightarrow$ Bb, B $\rightarrow$ b}. Ruas kiri semua produksinya terdiri dari sebuah V_N maka G_2 kemungkinan tipe CFG atau RG. Selanjutnya karena semua ruas kanannya terdiri dari sebuah V_T atau string $V_N V_T$ maka G_2 adalah RG.
3. Grammar G_3 dengan Q_3 = {S $\rightarrow$ Ba, B $\rightarrow$ bB, B $\rightarrow$ b}. Ruas kiri semua produksinya terdiri dari sebuah V_N maka G_3 kemungkinan tipe CFG atau RG. Selanjutnya karena ruas kanannya mengandung string $V_T V_N$ (yaitu bB) dan juga string $V_N V_T$ (Ba) maka G_3 bukan RG, dengan kata lain G_3 adalah CFG.

1.3.5 Derivasi Kalimat dan Penentuan Bahasa

Tentukan bahasa dari masing-masing gramar berikut :

1. G_1 dengan Q_1 = {1. S $\rightarrow$ aAa, 2. A $\rightarrow$ aAa, 3. A $\rightarrow$ b}.

   Jawab :

   Derivasi kalimat terpendek :

   S $\Rightarrow$ aAa (1)

   $\Rightarrow$ aba (3)

   Derivasi kalimat umum :

   S $\Rightarrow$ aAa (1)

   $\Rightarrow$ aaAaa (2)

…

$\Rightarrow a^n Aa^n$ (2)

$\Rightarrow a^n ba^n$ (3)

Dari pola kedua kalimat disimpulkan : $L_1(G_1) = \{ a^n ba^n \mid n \geq 1\}$

2. G_2 dengan $Q_2 = \{1.\ S \rightarrow aS,\ 2.\ S \rightarrow aB,\ 3.\ B \rightarrow bC,\ 4.\ C \rightarrow aC,\ 5.\ C \rightarrow a\}$.

Jawab :

Derivasi kalimat terpendek :

$S \Rightarrow aB$ (2)

$\Rightarrow abC$ (3)

$\Rightarrow aba$ (5)

Derivasi kalimat umum :

$S \Rightarrow aS$ (1)

…

$\Rightarrow a^{n-1}S$ (1)

$\Rightarrow a^n B$ (2)

$\Rightarrow a^n bC$ (3)

$\Rightarrow a^n baC$ (4)

…

$\Rightarrow a^n ba^{m-1}C$ (4)

$\Rightarrow a^n ba^m$ (5)

Dari pola kedua kalimat disimpulkan : $L_2(G_2) = \{ a^n ba^m \mid n \geq 1, m \geq 1\}$

1.4 Bahasa Pemrograman

Bahasa pemrograman adalah bahasa yang menjadi sarana manusia untuk berkomunikasi dengan komputer. Pikiran manusia yang tidak terstruktur harus dibuat terstruktur agar bisa berkomunikasi dengan komputer. Komputer memerlukan kepastian dan logika yang benar untuk dapat melakukan suatu instruksi tertentu. Untuk itu diperlukan algoritma yg baik dan benar.

Penggolongan bahasa pemrograman berdasarkan tingkat ketergantungannya dengan mesin :

a. Bahasa Mesin

Bahasa mesin adalah bahasa yang berisi kode-kode mesin yang hanya dapat diinterpretasikan langsung oleh mesin komputer. Bahasa mesin sering juga disebut native code (sangat tergantung pada mesin tertentu). Bahasa ini merupakan bahasa level terendah dan berupa kode biner 0 dan 1. Sekumpulan instruksi dalam bahasa mesin dapat membentuk *microcode* (semacam prosedur dalam bahasa mesin).

Contoh:

untuk mesin IBM/370

0001100000110101 = 1835

yang berarti mengkopikan isi register 5 ke register 3

Keuntungan : Eksekusi cepat

Kerugian : Sangat sulit dipelajari manusia

b. Bahasa Assembly (Mnemonic Code)

Merupakan bentuk simbolik dari bahasa mesin, dianggap sebagai bahasa pemrograman yang pertama kali berbentuk string dan lebih mudah dimengerti manusia. Setiap kode bahasa mesin memiliki simbol sendiri dalam bahasa assembly.

Misalnya ADD untuk penjumlahan, MUL untuk perkalian, SUB untuk pengurangan, dan lain-lain.

Sekumpulan kode - kode bahasa assembly dapat membentuk *makroinstruksi*. Bahasa assembly juga memiliki program pendebug-nya, tidak seperti bahasa mesin.

Misalnya: Turbo Assembler dan debug pada DOS.

Assembler akan mencocokkan token simbol dari awal s/d akhir, kemudian dikodekan menjadi bahasa mesin.

Kelebihan : Eksekusi cepat, masih bisa dipelajari daripada bahasa mesin, file hasil sangat kecil.

Kekurangan : Tetap sulit dipelajari, program sangat panjang.

c. Bahasa Tingkat Tinggi (High Level Language) / user oriented

Bahasa ini lebih dekat dengan bahasa manusia. Bahasa ini juga memberikan banyak sekali fasilitas kemudahan pembuatan program, misalnya: variabel, tipe data, konstanta, struktur kontrol, loop, fungsi, prosedur dan lain-lain. Contoh: Pascal, Basic, C++, dan Java. Mendukung information hiding, enkapsulasi, dan abstract data type.

Bahasa Tingkat tinggi memiliki generasi, misalnya generasi ke-3 (Pascal, C/C++) dan generasi ke-4 (Delphi, VB, VB.NET, Visual Foxpro)

Keuntungan : - Mudah dipelajari
- Mendekati permasalahan yang akan dipecahkan
- Kode program pendek

Kerugian : Eksekusi lambat

d. Bahasa yang berorientasi pada masalah spesifik (*specific problem oriented*).

Bahasa ini adalah bahasa yang digunakan langsung untuk memecahkan suatu masalah tertentu.

Contoh : SQL untuk aplikasi database, COGO untuk aplikasi teknik sipil, Regex untuk mencocokkan pola pada string tertentu, MatLab untuk matematika, dll.

Bahasa problem oriented kadang digolongkan ke dalam bahasa tingkat tinggi.

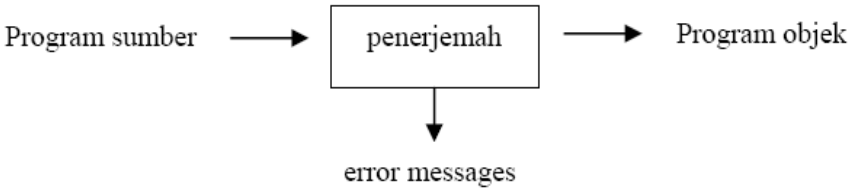
1.5 Translator

Translator (penerjemah) melakukan pengubahan *source code / source program* (program sumber) ke dalam *target code / object code / object program* (program objek).

Source code ditulis dalam bahasa sumber, object code berupa bahasa pemrograman lain / bahasa mesin pada suatu komputer.

Jadi penerjemah membaca suatu program yang ditulis dalam bahasa sumber dan menerjemahkan bahasa sumber ke dalam suatu bahasa lain.

Saat melakukan proses penerjemahan, penerjemah akan melaporkan adanya keanehan/kesalahan yang mungkin diketemukan.



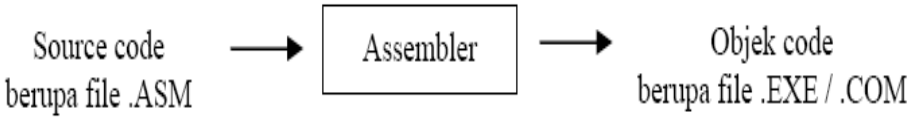
Gambar 1.1. Alur kerja Translator

Ada beberapa macam translator, yaitu :

a. Assembler

Source code adalah bahasa assembly, object code adalah bahasa mesin

contoh : Turbo Assembler, Macro Assembler



Gambar 1.2. Alur kerja Assembler

b. Interpreter

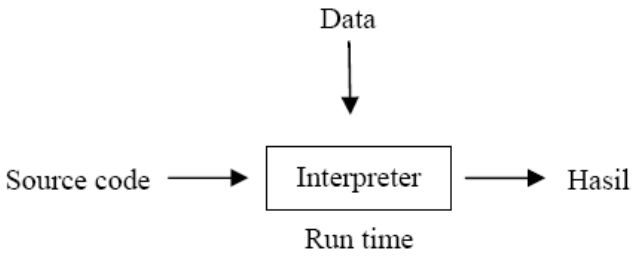
Input berupa source code yaitu bahasa scripting seperti PHP, Basic, Perl, Javascript, ASP, Java bytecode, Basic, Matlab, Matematica, Ruby.

Interpreter tidak menghasilkan object code. Hanya menghasilkan translasi internal. Input dapat berasal dari source code maupun dari inputan program dari user. Source code dan inputan data user diproses pada saat yang **bersamaan.**

Pada interpreter, program tidak harus dianalisis seluruhnya dulu, tapi bersamaan dengan jalannya program.

Keuntungan : mudah bagi user, debuging cepat

Kekurangan : eksekusi program lambat, tidak langsung menjadi program *executable*.



Gambar 1.3. Alur kerja Intepreter

c. Kompilator (Compiler)

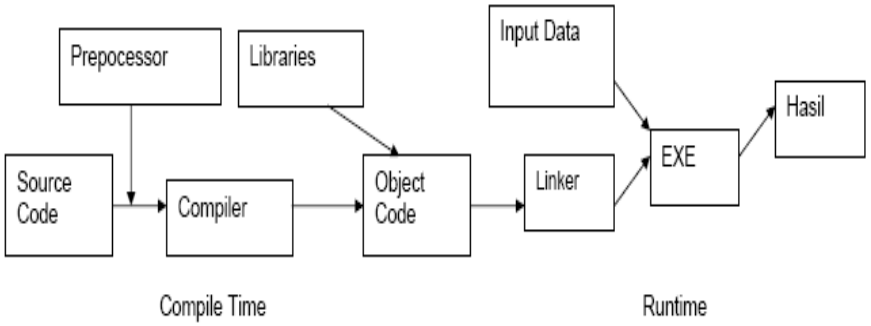
Istilah *compiler* muncul karena dulu ada program yang menggunakan subrutin-subrutin atau pustaka-pustaka untuk keperluan yang sangat khusus yang dikumpulkan menjadi satu sehingga diistilahkan *compiled.*

Input berupa source code program seperti Pascal, C, C++. Object code adalah bahasa assembly. Source code dan data input diproses pada saat yang berbeda.

Compile time adalah saat pengubahan dari source code menjadi object code. *Runtime* adalah saat eksekusi object code dan mungkin menerima input data dari user. Output : bahasa assembly. Kemudian oleh linker dihasilkan file EXE.

Kekurangan : debugging lebih lambat

Keuntungan : eksekusi program lebih cepat, menghasilkan file *executable* yang berdiri sendiri.



Gambar 1.4. Alur kerja Kompilator

1.6 Tahap–tahap Kompilasi

Kompilator (*compiler*) adalah sebuah *program* yang membaca suatu program yang ditulis dalam suatu *bahasa sumber* (*source language*) dan menterjemah-kannya ke dalam suatu *bahasa sasaran* (*target language*).

Proses kompilasi dikelompokan ke dalam dua kelompok besar:

1. Tahap Analisa (*Front-end*)

   Menganalisis source code dan memecahnya menjadi bagian-bagian dasarnya. Menghasilkan kode level menengah dari source code input yang ada.

2. Tahap Sintesa (*Back-end*)

   Membangun program sasaran yang diinginkan dari bentuk antara.

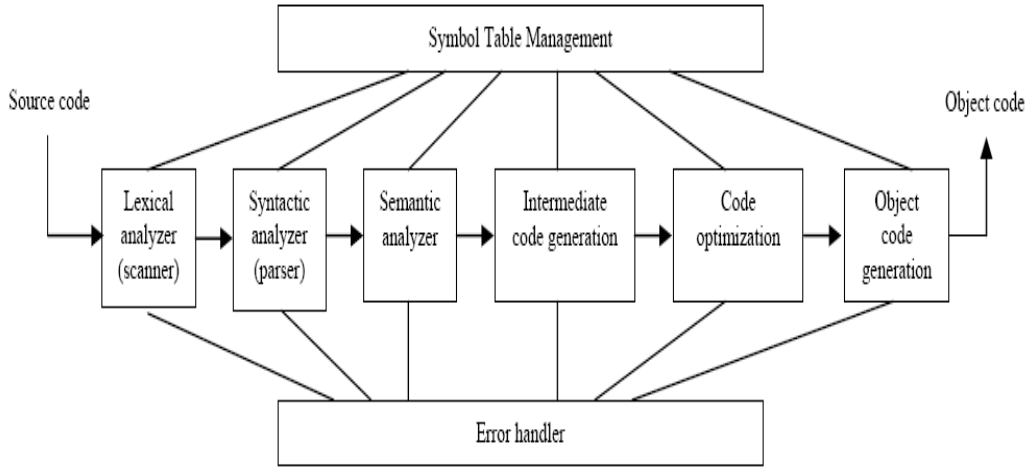
Tahap-tahap yang harus dilalui pada saat mengkompilasi program, yaitu:

1. Analisa Leksikal
2. Analisa Sintaks
3. Analisa Semantik
4. Pembangkit Kode Antara

Tahap analisa (front-end)

5. Code optimization
6. Object code generation

Tahap sintesa (back-end)



Gambar 1.5. Skema blok kompilator

Keterangan :

- *Analisa Leksikal (scanner)*

  Berfungsi memecah teks program sumber menjadi bagian-bagian kecil yang mempunyai satu arti yang disebut token, seperti : konstanta, nama variabel, keyword, operator.

- *Analisa Sintaks(parser)*

  Berfungsi mengambil program sumber (sudah dalam bentuk barisan token) dan menentukan kedudukan masing-masing token berdasarkan aturan sintaksnya dan memeriksa kebenaran dan urutan kemunculan token.

- *Analisa Semantik*

  Berfungsi menentukan validitas semantiks/keberartian program sumber. Biasanya bagian ini digabung dengan Pembangkit kode antara *(intermediate code generator).*

- *Pembangkit Kode Antara*

  Berfungsi membangkitkan kode antara.

- *Code optimation*

  Berfungsi mengefisienkan kode antara yang dibentuk.

- *Code generator*

  Berfungsi membangkitkan kode program target dalam bahasa target yang ekivalen dengan bahasa sumber .

- *Symbol table management*

  Berfungsi mengelola tabel simbol selama proses kompilasi. Tabel simbol adalah struktur data yang memuat record untuk tiap identifier dengan atribut-atribut identifier itu.

- *Penangan Kesalahan (Error handler)*

Berfungsi menangani kesalahan yang berlangsung selama proses kompilasi.

Contoh :

pernyataan pemberian nilai (assignment) :

*position := initial + rate * 60*

Lexical analysis

Mengelompokkan pernyataan tersebut menjadi token-token sebagai berikut :

1. Token *identifier* position
2. Token *simbol assignment* :=
3. Token *identifier* initial
4. Token *tanda plus* +
5. Token *identifier* rate
6. Token *tanda perkalian* *
7. Token *konstanta angka* 60

Ketika identifier pada program sumber ditemukan lexical analyzer, identifier dimasukkan ke tabel simbol.

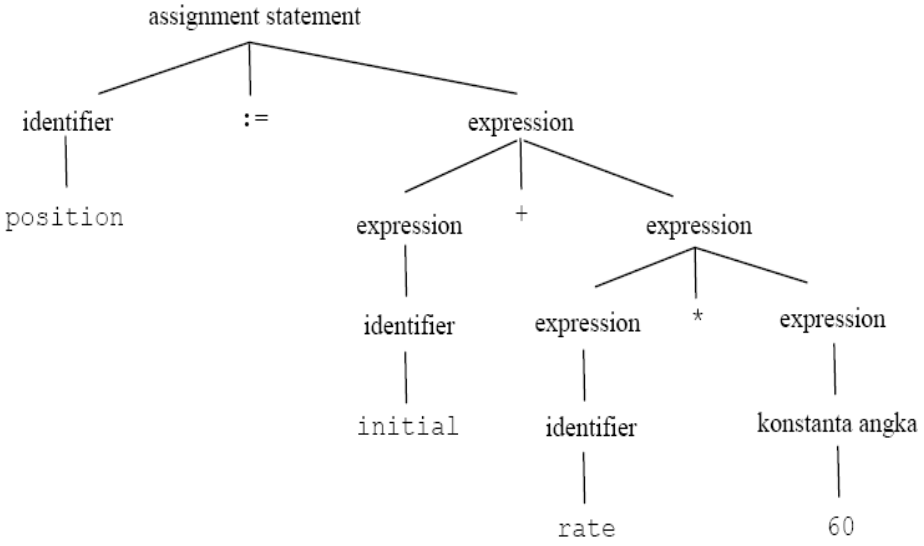
position := initial + rate * 60

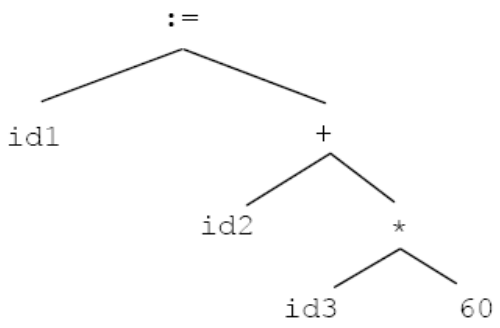
diubah menjadi

id1 := id2 + id3 * 60

Syntax analysis

Memparsing atau membentuk pohon sintaks pernyataan, yaitu :





Semantic analysis

Memeriksa kebenaran arti program sumber, mengumpulkan informasi tipe bagi tahap berikutnya. Tahap ini menggunakan pohon sintaks tahap *syntax analysis* untuk identifikasi operator dan operand suatu ekspresi dan kalimat. Komponen penting analisis semantik adalah pemeriksaan tipe, memeriksa operator yang harus mempunyai operand yang diijinkan oleh spesifikasi bahasa sumber.

Karena misal adanya pernyataan deklarasi di awal :

```
var
     position, initial, rate : real
```

Maka konstanta 60 dikonversi menjadi real dengan fungsi **inttoreal(60)** menjadi konstanta bilangan real.

Intermediate Code Generator

Intermediate code adalah representasi perantara antara bentuk bahasa tingkat tinggi dengan bahasa mesin. Karena pada level berikutnya masih akan dilakukan optimasi, maka perlu dibuat representasi yang memudahkan optimasi, yang bukan merupakan bahasa mesin.

```
temp1 := inttoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```

Code Optimization

Tahap code optimization proses identifikasi dan membuang operasi-operasi yang tidak perlu dari intermediate code generation untuk penyederhanaan sehingga nantinya kode mesin hasil menjadi lebih cepat. Kode-kode tersebut dioptimasi menjadi :

```
Temp1 := id3 * 60.0
Id1 := id1 + temp1
```

Code Generator

Tahap akhir kompilator adalah pembangkitan kode target/objek dan biasanya kode mesin atau assembly yang dapat direlokasi. Pembangkitan kode sangat bergantung pada mesin yang dipakai, misal :

```
MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1
```

1.7 Preprocessor

Preprocessor adalah suatu program khusus menanggulangi terjadinya beberapa modul yang terpisah saat melakukan penulisan bahasa sumber menjadi beberapa file ke dalam suatu program baru. Suatu Preprocessor menghasilkan suatu input bagi suatu kompilator. Hal ini mungkin dilakukan oleh suatu kompilator antara lain:

- Pemrosesan Makro

  Makro yang merupakan kependekan dari suatu bagian program yang lebih panjang memungkinkan penulis program untuk memperpendek program yang ditulisnya. Dalam hal ini perlu dilakukan dua hal :

  a. Mendefinisikan makro yang digunakan.

     Parameter yang didefinisikan pada makro disebut dengan parameter formal.

b. Melakukan pemanggilan makro yang mungkin juga mengandung beberapa parameter. Sedangkan parameter yang digunakan untuk memanggil makro disebut dengan paramater actual.

- Pengikutsertaan berkas (File Inclusion)

  Suatu Preprocessor memungkinkan diikutsertakannya beberapa berkas program yang telah ditulis sebelumnya ke dalam program yang sedang ditulis. Biasanya berkas program yang ditulis sebelumnya merupakan segmen program yang sekali digunakan, banyak manfaatnya dan sering terjadi sudah merupakan bagian dari sistem bahasa yang digunakan.

  Misalnya pada bahasa C, isi dari berkas global.h dapat diikutsertakan dalam program yang sedang ditulis dengan menggunakan perintah #include global.h.

- Preprocessor Rasional

  Preprocessor ini memberikan kemampuan baru dari suatu bahasa dengan fasilitas pengendalian aliran (flow-of-control) atau struktur data yang lebih baik. Misalnya dengan menambahkan kemampuan perintah while, if-then-else pada bahasa yang pada mulanya tidak mempunyai fasilitas tersebut. Hal ini biasanya dilakukan dengan menggunakan makro yang sudah ada dalam bahasa tersebut.

- Perluasan Bahasa

  Preprocessor ini memungkinkan suatu bahasa untuk berinteraksi dengan sistem atau bahasa lainnya. Misalnya pada bahasa C yang ditambahkan kemampuannya untk dapat mengakses data dalam suatu database. Untuk itu praprosesor memungkinkan menggunakan tanda ## yang menyatakan bahwa bagian ini bukan merupakan bagian dari bahasa C, tetapi berhubungan dengan sistem suatu paket database lain yang sudah baku. Dengan demikian bagian ini akan diterjemahkan kedalam pemanggilan procedure untuk melakukan akses database.

1.8 Mutu Compiler

a. Kecepatan dan waktu proses kompilasi

Hal ini tergantung dari algoritma untuk menulis kompiler itu dan kompiler pengkompilasi.

b. Mutu program objek

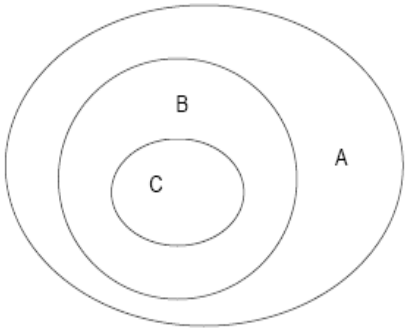
Dilihat dari ukuran dan kecepatan eksekusi program.

c. *Integrated Development Environtment* (IDE)

Fasilitas-fasilitas terintegrasi yang dimiliki oleh kompiler. Misalnya untuk debugging, editing, dan testing. Contoh : bandingkan antara compiler Pascal dan Clipper.

1.9 Bootstrap

Metode **Bootstrap** dikembangkan oleh Nikolaus Writh, penulis bahasa Pascal. Metode Bootstrap adalah pembuatan kompilator secara bertingkat.



Gambar 1.6. Metode Bootstrap

Metode ini menganggap bahwa C dibangun dengan Assembly, B dibangun dengan C, dan A dibangun dengan B. Jadi compiler dapat dibangan secara keseluruhannya dengan bahasa-bahasa sebelumnya. Metode Bootstrap berarti menulis suatu bahasa dengan kompiler versi sebelumnya.

1.10 Konsep dan Notasi Bahasa

Bahasa adalah himpunan semua string yang dapat dibentuk dari himpunan alphabet. Klasifikasi bahasa menurut hirarki Chomsky :

- Bahasa Regular

  Pada mesin otomata Finite State Automata (NFA dan DFA).
- Bahasa Bebas Konteks (Context Free)

  Pada mesin otomata Push Down Automata.
- Context Sensitive

  Bekerja pada mesin otomata Linier Bounded Automata.
- Unrestricted / Tipe 0 / Phase Structure

  Pada mesin turing, tidak ada batasan.

1.10.2 Token, Lexeme, dan Pattern

Token adalah symbol terminal pada teori bahasa. Token merupakan bagian hasil dari pemecahan sumber program yaitu penerjemahan lexeme pada saat melakukan scanner. Token mereprentasikan identifier (nama variabel, fungsi, tipe atau nama yang didefinisikan pemakai), keyword, literal string, operator, label, simbol tanda (tanda kurung, koma, titik koma), constant, relation, identity, number, variable.

Lexeme adalah string yang merupakan masukan dari analisis Leksikal. Lexeme adalah kelompok karakter yang membentuk sebuah token.

Token tertentu harus memenuhi aturan yang disebut **Pattern.** Token merupakan sekumpulan karakter yang sesuai dengan pattern-nya.

Tabel 1.3. Tabel Penggunaan Token, Lexeme, dan Pattern

Token	**Contoh Lexeme**	**Pattern**
Const	Const	Const
If	If	If
Relation	<, < =, =, < >, > =, >	< or < = or = or < > or > = or >
Id	Phi, count, D2	Letter(letter|digit)*
Num	3.14 , 0.602E23	Digit+(.digit+) ? (E(+ | -) ? Digit +) ?

1.10.2 Diagram Status / State Transition Diagram

Berguna untuk mendapatkan token, yaitu melakukan analisis leksikal. Misal suatu bahasa memiliki himpunan simbol terminal/token berikut : (t_PLUS, t_MIN, t_ID, t_INT).

Maka diagram state-nya :

*t_ID(identifier) bisa berupa nama atau keyword.

Keyword yang sudah didefinisikan oleh suatu bahasa. Misal VAR jumlah : integer, maka VAR, integer adalah keyword, jumlah adalah nama.

1.10.3 Notasi BNF (Backus Naur Form)

Aturan-aturan produksi dapat dinyatakan dalam bentuk BNF.

< > : mengapit non terminal.

{ } : pengulangan 0 sampai n kali.

[ ] : 0 atau 1 kali muncul.

( ) : contoh x(yz) = xy | xz.

Contoh :

Aturan Produksi E -> T | T + E | T - E, T -> a

Notasi BNF E ::= <T> | <T>+<E> | <T>-<E>

1.10.4 Diagram Sintaks

Membantu pembuatan parser/analisis sintaksis.

Kotak = variabel/nonterminal, bulat = terminal.

Catatan Penting : Untuk praktikum materi ini, pelajari program operasi file pada Bahasa C (simpan, baca, ubah, hapus data atau File).

LAPORAN PENDAHULUAN

1. Apa yang kamu ketahui tentang penggolongan bahasa pemrograman berdasarkan tingkat ketergantungannya dengan mesin *?*
2. Jelaskan perbedaan Kompilator *(compiler)* dengan *Intepreter* !
3. Berikan penjelasan dari istilah-istilah berikut :
    a. Kompilator
    b. Translator
    c. Intepreter
    d. Assembler
4. Sebutkan dan jelaskan tahap-tahap kompilasi !
5. Sebutkan dan jelaskan klasifikasi bahasa menurut hirarki Chomsky!

LAPORAN AKHIR

1. Sebutkan dan jelasan beberapa contoh produk yang ada dipasaran untuk :
    a. Kompilator
    b. Intepreter
2. Tuliskan perbandingan komponen-komponen mutu kompilator yang tampak pada beberapa kompilator yang ada di pasaran (misalkan : Turbo Pascal, Turbo C, Microsoft C, dsb)!
3. Buat program sederhana untuk operasi File!