PERTEMUAN 1

PERANAN DAN FUNGSI SEBUAH KOMPILER

A. TUJUAN PEMBELAJARAN

Pada pertemuan ini akan dijelaskan mengenai sejarah kompilasi, Pengertian Translator (Compiler & Interpreter), Tahap-tahap Kompilasi, dan Pembuatan Kompilator (*Compiler*). Setelah menyelesaikan materi pada pertemuan ini, mahasiswa mampu:

1. Mendefinisikan pengertian kompilasi.
2. Menjelaskan pengertian dan model dari *translator compiler, interpreter* dan *assembler.*
3. Menjelaskan pembuatan *Compiler.*

B. URAIAN MATERI

1. Pengertian Kompilasi

Sejarah perkembangan suatu kompilator sudah dimulai sejak lama, yaitu pada saat mulai ditemukannya komputer pada awal 1950-an. Sejak waktu tersebut teknik dan cara pembentukan suatu kompilator telah berkembang dengan sangat pesat dan pembentukkan suatu kompilator dapat dilakukan makin mudah. Demikian pula program bantu (*tools*) untuk membuat suatu kompilator sudah dapat diperoleh sehingga pembentukan suatu kompilator dapat dilakukan dengan cepat.

Kompilator pertama yang dibuat adalah kompilator untuk bahasa FORTRAN yang pada saat itu dikembangkan dengan memakan sejumlah tenaga ahli yang setara dengan pekerjaan yang dilakukan oleh 18 orang. Dengan adanya program bantu dan tata cara pembentukan yang sistematis dan tertata dengan baik serta pendefinisian struktur bahasa yang cermat, maka suatu kompilator untuk bahasa yang terstruktur seperti PASCAL atau C dapat dikembangkan.

Proses kompilasi dari suatu kompilator pada dasarnya dapat dibagi ke dalam 2 bagian utama yaitu bagian analisis dan bagian sintesis. Tahap analisis program yang ditulis dalam bahasa sumber dibagi dan dipecah ke dalam beberapa bagian yang kemudian akan dipresentasikan ke dalam suatu bentuk antara dari program sumber.

Operasi-operasi yang dilakukan oleh program sumber ditentukan dan dicatat dalam suatu struktur pohon (*tree*) yang disebut dengan nama pohon sintaks (*sintax tree*) Dalam hal ini setiap nodal pada tree tersebut menyatakan suatu operasi, sedangkan anak dari node (titik) tersebut memberikan argumen yang diperlukan.

2. Pengertian dan model dari translator, compiler, interpreter & assembler

a. Pengertian *Translator (Compiler & Interpreter)*

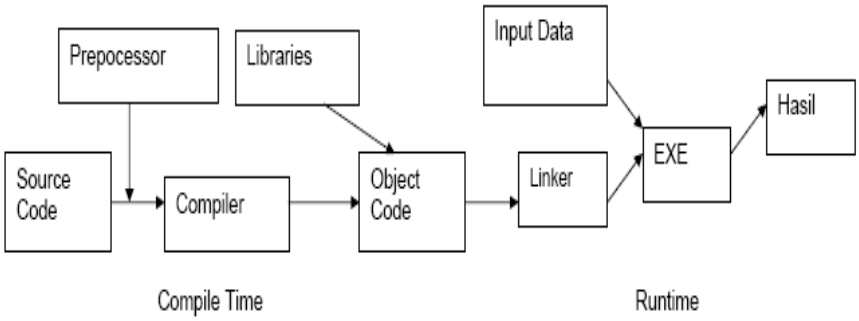
Arti kata teknik kompilasi:

1) Teknik adalah suatu metode atau cara
2) Kompilasi suatu proses menggabungkan serta menterjemahkan sesuatu (*source program*) menjadi bentuk lain
3) *Compile – to translate a program written in a high-level programming language*

b. Pengertian *Compiler*

Kompilator (*compiler*) adalah suatu program yang menerjemahkan bahasa program (*source code*) kedalam bahasa objek (object code). Kompilator menggabungkan keseluruhan bahasa program, mengumpulkannya dan kemudian menyusunnya kembali.

Kompilator memerlukan waktu untuk membuat suatu program dapat di eksekusi oleh komputer, program yang di eksekusi oleh *compiler* dapat berjalan lebih cepat dibanding program yang diproduksi oleh *interpreter*, disamping itu juga bersifat *independent*. Contoh program yang menggunakan *compiler* adalah Visual Basic, dan Pascal.



Gambar 1.1. Alur kerja Kompilator

Tahap Kompilasi:

1) Pertama *source code* (program yang ditulis) dibaca ke memori komputer.
2) *Source code* tersebut diubah menjadi object code (bahasa *assembly*).
3) *Object code* dihubungkan dengan library yang dibutuhkan untuk membentuk file yang bisa di eksekusi.

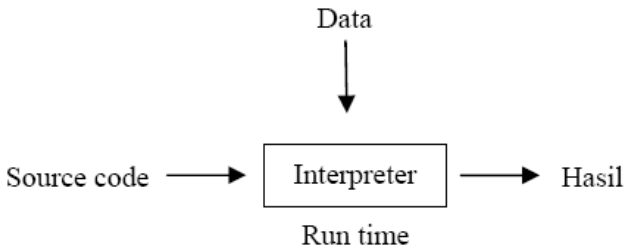
c. Pengertian *Interpreter*

Interpreter adalah perangkat lunak yang mampu mengeksekusi kode program (yang ditulis oleh programmer) lalu menterjemahkannya ke dalam bahasa mesin, sehingga mesin melakukan intruksi yang diminta oleh programmer tersebut. Perintah-perintah yang dibuat oleh programmer tersebut di eksekusi baris demi baris, sambil mengikuti logika yang terdapat di dalam kode tersebut.

Proses ini sangat berbeda dengan *compiler*, dimana pada *compiler* hasilnya sudah langsung berupa satu kesatuan perintah dalam bentuk bahasa mesin, dimana proses penterjemahan dilaksanakan sebelum program tersebut di eksekusi.

Interpreter atau dalam Bahasa Indonesia dikenal sebagai (Juru Bahasa) berbeda dengan *Translator* atau penterjemah dalam segi media yang dipakai untuk menerjemahkan. *Interpreter* akan menterjemahkan bahasa sumber ke dalam bahasa sasaran secara langsung atau orally sementara *translator* akan menerjemahkan bahasa sumber ke bahasa sasaran secara tertulis.

Java dijalankan menggunakan *interpreter* yaitu Java Virtual Machine (JVM). Hal ini menyebabkan *source code* java yang telah di kompilasi menjadi java byte codes dapat dijalankan pada platform yang berbeda-beda.



Gambar 1.2. Alur kerja Intepreter

d. Pengertian Asemmbler

Bahasa *assembly* adalah sebuah program yang terdiri dari intruksi-intruksi yang menggantikan kode-kode biner dari bahasa dengan "mnemonic" yang mudah di ingat. Misalnya sebuah intruksi penambahan dalam bahasa mesin dengan kode "10110011" yang dalam bahasa *assembly* dapat dibuat dalam intruksi mnemonic ADD, sehingga mudah di ingat dibandingkan dengan angka 0 dan 1, dalam setiap intruksi membutuhkan suatu operand baik berupa data langsung maupun suatu lokasi memori yang menyimpan data bersangkutan.

Bahasa *assembly* sering juga disebut kode sumber atau kode simbolik yang dapat dijalankan oleh prosesor, sedangkan *assembler* adalah suatu program yang dapat menerjemahkan program bahasa *assembly* ke program bahasa mesin. Bahasa mesin adalah kumpulan kode biner yang merupakan intruksi yang bisa dijalankan oleh komputer. Program bahasa mesin sering disebut sebagai kode objek.

e. Pengertian Linker

Linker adalah suatu program yang menterjemahkan program objek (berekstension OBJ) ke bentuk program eksekusi (berekstension .EXE atau .COM). Sedangkan untuk membuat file object ke bentuk file yang dapat di eksekusi (berekstension .EXE atau .COM) bisa gunakan file TLINK.EXE.

3. Tahap–tahap Kompilasi

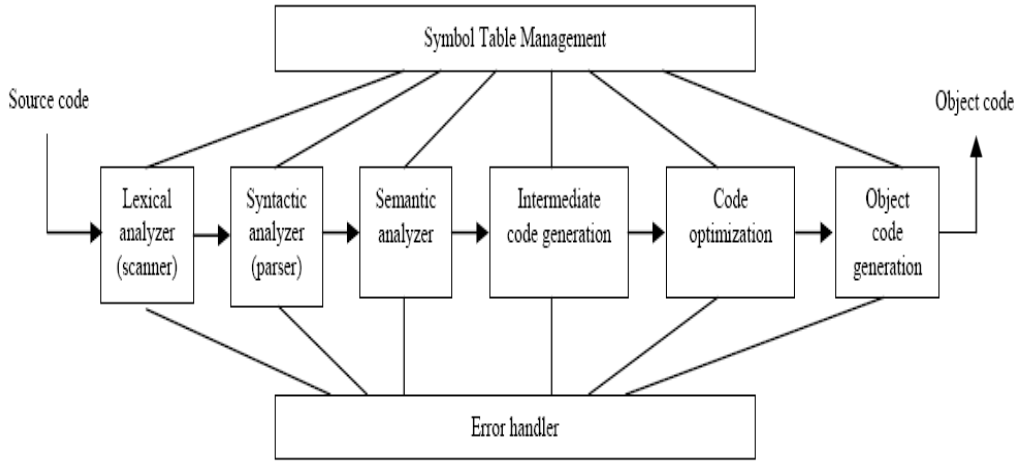
Proses kompilasi dikelompokan ke dalam dua kelompok besar:

a. Tahap Analisa (*Front-end*): Menganalisis *source code* dan memecahnya menjadi bagian-bagian dasarnya. Menghasilkan kode level menengah dari *source code* input yang ada.

b. 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 — ***Tahap analisa (front-end)***
4) Pembangkit Kode Antara
5) *Code optimization*
6) *Object code generation* — ***Tahap sintesa (back-end)***



Gambar 1.3. Skema blok kompilator

Keterangan :

a) *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.

b) *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.

c) *Analisa Semantik*

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

d) *Pembangkit Kode Antara*

Berfungsi membangkitkan kode antara.

e) *Code optimation*

Berfungsi mengefisienkan kode antara yang dibentuk.

f) *Code generator*

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

g) *Symbol Tabel management*

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

h) *Penangan Kesalahan (Error handler)*

Berfungsi menangani kesalahan yang berlangsung selama proses kompilasi.

Contoh :

pernyataan pemberian nilai (assignment) :

*position := initial + rate * 60*

a) **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.

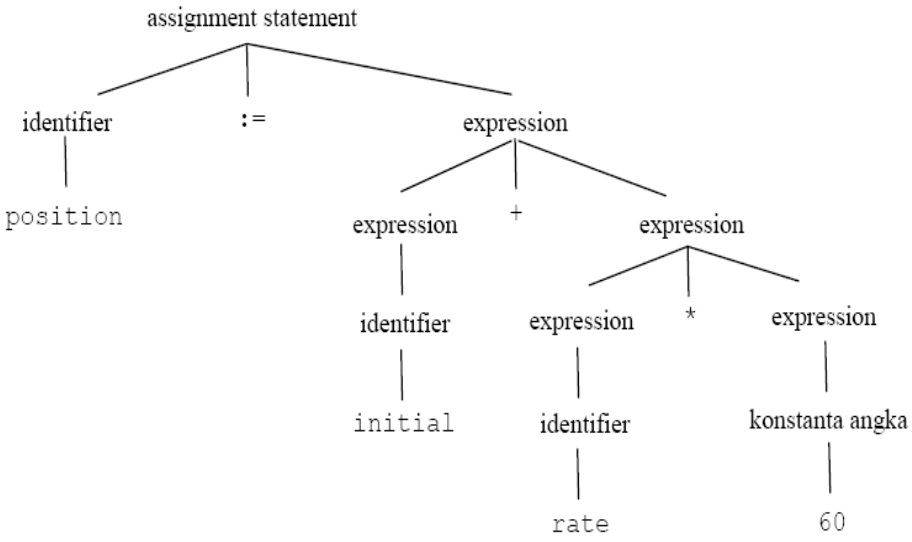
position := initial + rate * 60

diubah menjadi

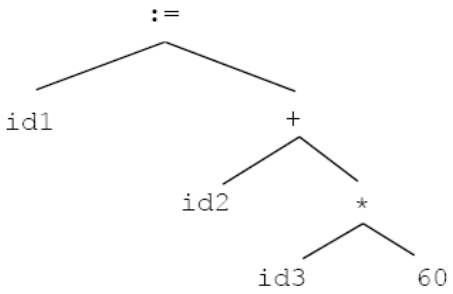
id1 := id2 + id3 * 60

b) **Syntax analysis**

Memparsing atau membentuk pohon sintaks pernyataan, yaitu :



Gambar 1.4 Bentuk Pohon Sintaks Pernyataan



Gambar 1.5 Bentuk Perubahan Pernyataan

c) **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.

d) ***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
```

e) ***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
```

f) ***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
```

4. Pembuatan *compiler*

Kompiler yang bagus adalah yang dapat bekerja dengan baik pada mesin-mesin komputer dan tidak membutuhkan proses yang lama. Ada berapa cara dalam membuat suatu penterjemah dalam hal ini ,misalnya *compiler* akan bahasa yang digunakan untuk membuat *compiler* pun beragam misalnya:

a. Bahasa mesin

   1) Sangat sukar dan sangat sedikit kemungkinannya untuk membuat *compiler* dengan bahasa ini, karena manusia susah mempelajari bahasa mesin
   2) Sangat tergantung pada mesin
   3) Bahasa Mesin kemungkinan digunakan pada saat pembuatan *Assembler*

b. *Assembly*

   1) Hasil dari program mempunyai Ukuran yang relatif kecil
   2) Sulit dimengerti karena statement/perintahnya singkat-singkat, butuh usaha yang besar untuk membuat
   3) Fasilitas yang dimiliki terbatas

c. Bahasa Tingkat Tinggi (high level language)

   1) Lebih mudah dipelajari
   2) Fasilitas yang dimiliki lebih baik (banyak)
   3) Memiliki ukuran yang relatif besar, misal membuat *compiler* pascal dengan menggunakan bahasa C
   4) Untuk mesin yang berbeda perlu dikembangkan tahapan-tahapan tambahan.
   5) Misal membuat *compiler* C pada Dos bedasarkan *compiler* C pada unix

d. Bahasa Tingkat Tinggi (pemrograman)

   1) Bahasa yang lebih dikenal oleh manusia, maksudnya adalah *statement* yang digunakan menggunakan bahasa yang dipakai oleh manusia (Inggris)
   2) Programmudah untuk dikoreksi dan diperbaiki (*debug*)

3) Tidak tergantung pada salah satu jenis mesin komputer

4) Bahasa tingkat tinggi biasanya masih membutuhkan *translator*

5) Memberikan fasilitas yang lebih banyak, seperti struktur kontrol program yang terstruktur, memiliki blok-blok, serta prosedur dan fungsi-fungsi:

   a) Struktur kontrol seperti:

   b) kondisi (If ...Then...Else)

   c) perulangan (For, While)

   d) struktur blok (begin...End, {...})

6) Oleh karena itu dari bahasa tingkat tinggi ke dalam bahasa mesin dibutuhkan sesuatu untuk menterjemahkan agar mesin (komputer) mengerti apa yang diinginkan manusia. Menterjemahkan statement bahasa tingkat tinggi ke dalam bahasa tingkat rendah dapat dibedakan menjadi dua; melalui *interpreter* atau *compiler* yang fungsinya sama yaitu menerjemahkan.

e. BootStrap

1) Untuk membangun sesuatu yang besar, dibangun/dibuat dulu bagian intinya (niklaus Wirth - saat membuat pascal *compiler*).

2) PO dibuat dengan *assembly*, P1 dibuat dari P0, dan P2 dibuat dari P1, jadi *compiler* untuk bahasa P dapat dibuat tidak harus dengan menggunakan *assembly* secara keseluruhan.

C. SOAL LATIHAN/TUGAS

1. Carilah definisi Teknik kompilasi menurut para ahli selain yang telah dijelaskan dalam modul. Buatlah kesimpulan berdasarkan definisi para ahli tersebut dengan bahasa Anda sendiri!
2. Jelaskan tahap-tahap kompilasi dan berikan contoh!
3. Berikan contoh pembuatan *compiler* dengan satu bahasa *compiler* yang ada di modul!

D. REFERENSI

Alfred V. Aho, Monica S. Compilers “Priciples, Techniques, & Tools” , Second EditionLam, Boston San Francisco New York , 2007

Kenneth C. Louden “Compiler Contruction, Principles and Practice”,

P.D. Terry, “ Compilers and compiler Generator ", Rhodes University, 1996

Jean-Paul Tremblay, Paul G. Sorenson “ The Theory and Practice of Compiler Writing”, McGraw-Hill Book Company, United State of America, International Edition , 1995

Thomas W. Parsons, “ Introduction to Compiler Construction”, Hostfra University, Computer Science Press, New York , 1992

GLOSARIUM

Kompilator (compiler) adalah suatu program yang menerjemahkan bahasa program kedalam bahasa objek.

Interpreter adalah perangkat lunak yang mampu mengeksekusi kode program (yang ditulis oleh programmer) lalu menterjemahkannya ke dalam bahasa mesin.

Bahasa assembly adalah sebuah program yang terdiri dari intruksi-intruksi yang menggantikan kode-kode biner dari bahasa dengan “mnemonic” yang mudah di ingat.
Linker adalah suatu program yang menterjemahkan program objek ke bentuk program eksekusi