Automatic Machine Translation

Bahasan Materi Kuliah

- Pendahuluan: arti dari Kompilasi
- Translator: Compiler dan interpreter
- Bahasa Pemrograman
- Pembuatan Compiler
- Konsep bahasa dan Notasi
- Hirarki Comsky
- Aturan Produksi
- Diagram state
- Notasi BNF
- Diagram Syntax
- Kualitas Compiler

Bahasan Materi Kuliah

- Beberapa translator
- Struktur Compiler
- Lexical Analysis
- Analysis Syntax
- Analysis Semantics
- Error Handling
- Optimation
- Tabel informasi

ARTI KATA TEKNIK KOMPILASI

- **Teknik** :
  - Metode atau Cara
- **Kompilasi** :
  - Proses mengabungkan serta menterjermahkan sesuatu (source program) menjadi bentuk lain
- **Compile** :
  - To translate a program written in a high-level programming language into machine language.

Translator : Compiler & Interpreter

Translator :

- Adalah suatu program dimana mengambil input sebuah program yang ditulis pada satu bahasa program (source language) ke bahasa lain (The object on target language)

- Jika source language adalah high level language, seperti cobol, pascal, fortran maka object language adalah low-level language atau mesin language. Translator seperti ini disebut COMPILER

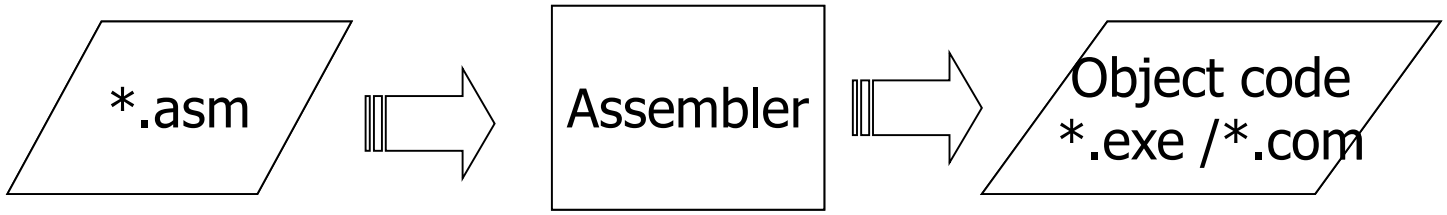
Kenapa perlu Translator ?

- Dengan bahasa mesin adalah bahasa bentuk bahasa terendah komputer, berhubungan langsung dengan bagian bagian komputer seperti bits, register & sangat primitive
- Jawaban atas pertanyaan ini akan membingungkan bagi programmer yang membuat program dengan bahasa mesin.
- Bahasa mesin adalah tidak lebih dari urutan 0 dan 1
- Instruksi dalam bahasa mesin bisa saja dibentuk menjadi *micro-code*, semacam prosedur dalam bahasa mesin
- Bagaimana dengan orang tidak mengerti bahasa mesin

Ada Beberapa Translator

1. Assembler

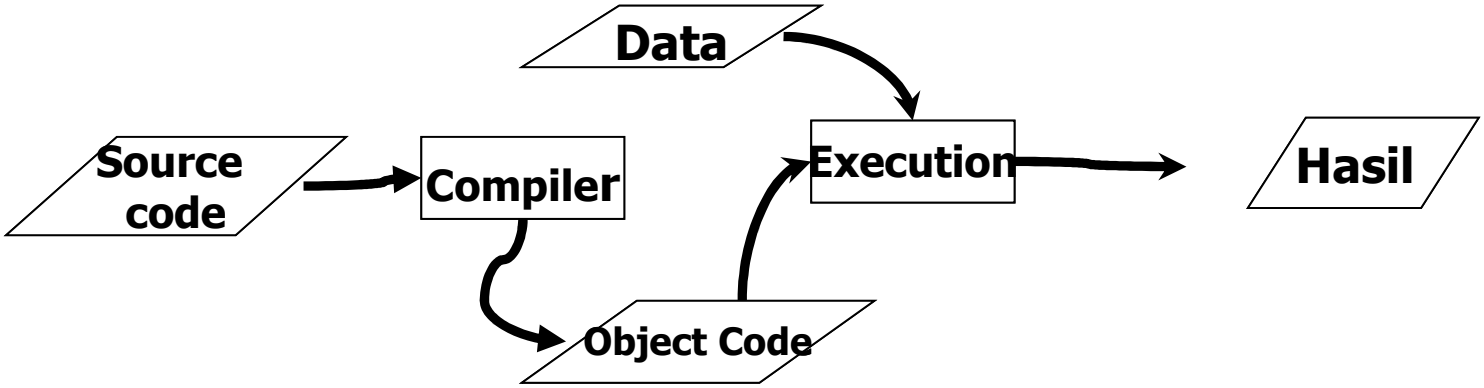
Source code adalah bahasa assembly, Object code adalah bahasa mesin



2. Compiler

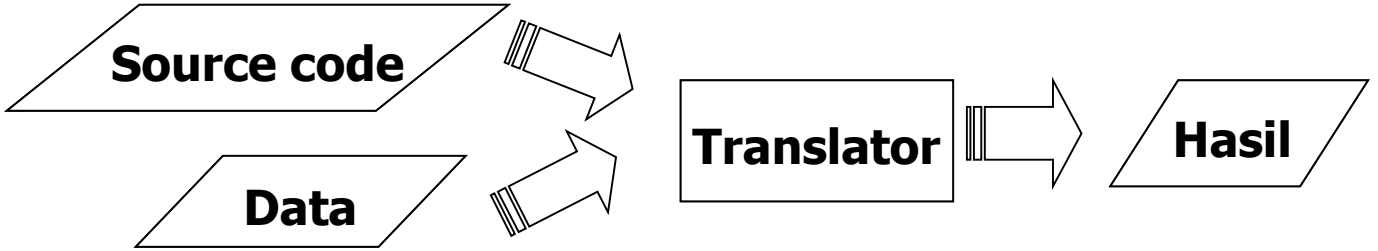
Source code adalah bahasa tingkat tinggi, object code adalah bahasa mesin atau bahasa assembly. Source code dan data diproses berbeda

compiler

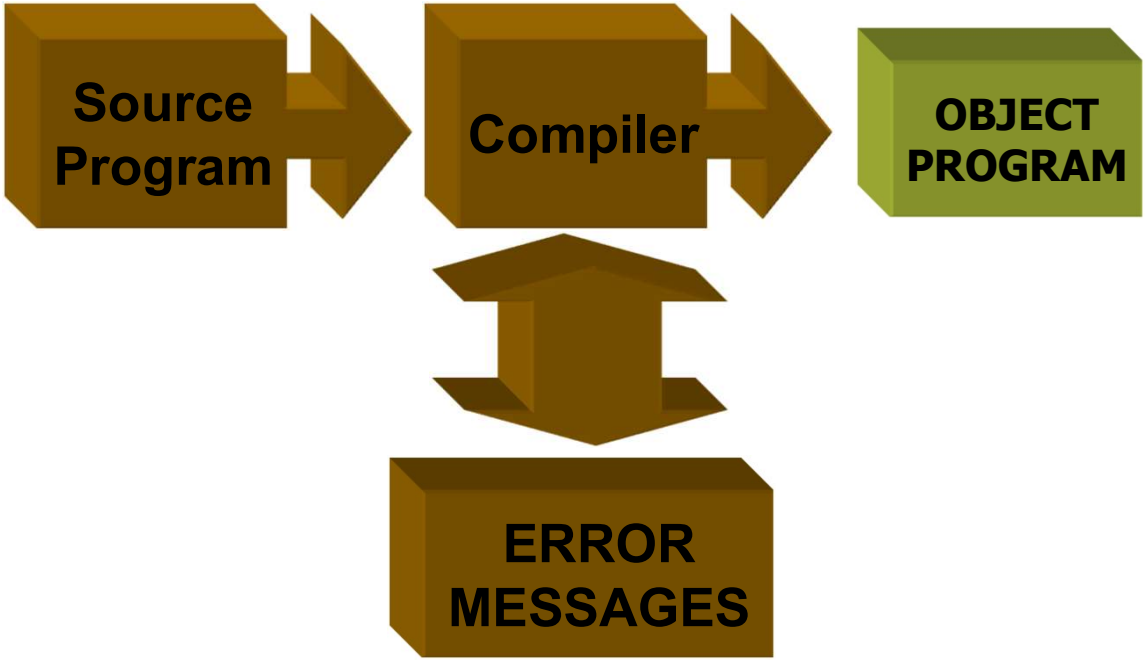


3. Interpreter

Interpreter tidak menghasilkan bentuk object code, tetapi hasil translasinya hanya dalam bentuk internal, dimana program induk harus selalu ada-berbeda dengan compiler



Translator : Compiler & Interpreter



COMPILER vs INTERPRETER

- Compiler bisa menangkap berbagai kesalahan dalam 1 program kode sumber secara sekaligus. Kalau Interpreter cuma bisa menangkap beberapa kesalahan pada 1 baris kode sumber pada suatu saat
- Biasanya program yang dihasilkan compiler lebih cepat dari waktu pelaksanaan program dengan interpreter.
- Kalau compiler menghasilkan kode antara (misal object code) dan harus digabungkan / dilink menjadi bentuk yang dapat dijalankan mesin / komputer (executable). Kalau Interpreter biasanya tidak menghasilkan kode antara.
- Kalau hendak menjalankan program hasil kompilasi bisa dilakukan tanpa kode sumber. Kalau interpreter butuh kode sumber.

COMPILER vs INTERPRETER

- Kalau dengan kompiler, maka pembuatan kode yang bisa dijalankan mesin dilakukan dalam 2 tahap terpisah, yaitu parsing / pembuatan kode objek dan linking / penggabungan kode objek dengan library. Kalau interpreter tidak ada proses terpisah.

- Kalau compiler membutuhkan linker untuk menggabungkan kode objek dengan berbagai macam library demi menghasilkan suatu kode yang bisa dijalankan oleh mesin. Kalau interpreter tidak butuh linker.

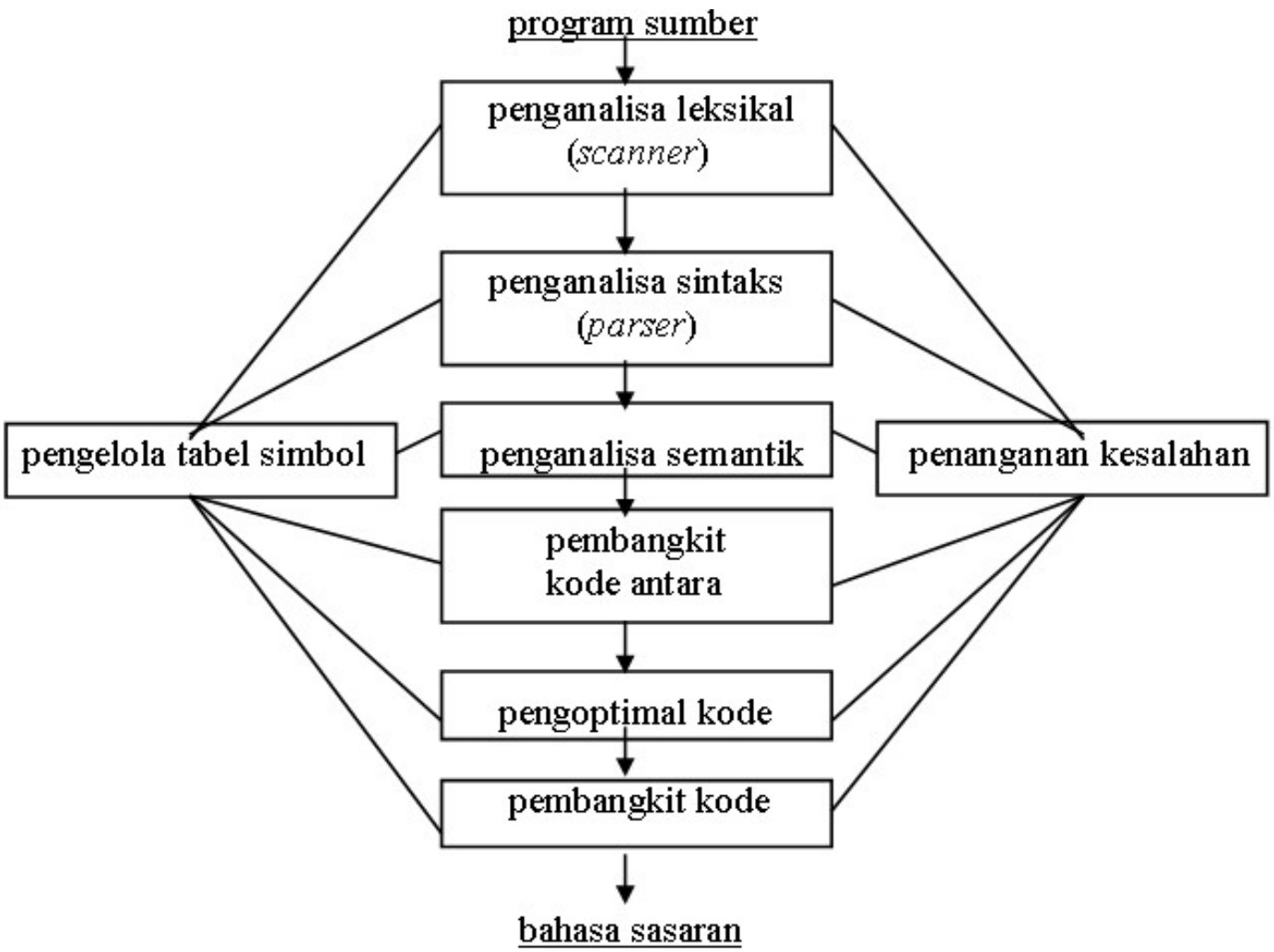
- Interpreter cocok untuk membuat / menguji coba modul / sub-routine / program-program kecil. Kalau compiler agak repot karena untuk mengubah suatu modul / kode objek kecil, maka harus dilakukan proses linking / penggabungan kembali semua objek dengan library yang diperlukan.

- Pada kompiler bisa dilakukan optimisasi / peningkatan kwalitas kode yang bisa dijalankan. Ada yang dioptimasi supaya lebih cepat, ada yang supaya lebih kecil, ada yang dioptimasi untuk sistem dengan banyak processor. Kalau interpreter susah / tidak bisa dioptimasikan.

Proses kompilasi dikelompokkan ke dalam dua kelompok besar :

1. *analisa* : program sumber dipecah-pecah dan dibentuk menjadi bentuk antara (*inter-mediate representation*)
2. *sintesa* : membangun program sasaran yang diinginkan dari bentuk antara

Fase-fase proses sebuah kompilasi



Penganalisa Leksikal

- membaca program sumber, karakter demi karakter. Sederetan (satu atau lebih) karakter dikelompokkan menjadi satu kesatuan mengacu kepada *pola kesatuan kelompok karakter* (*token*) yang ditentukan dalam *bahasa sumber*. Kelompok karakter yang membentuk sebuah token dinamakan *lexeme* untuk token tersebut. Setiap token yang dihasilkan disimpan di dalam *tabel simbol*. Sederetan karakter yang tidak mengikuti pola token akan dilaporkan sebagai *token tak dikenal* (*unidentified token*)

Penganalisa Sintaks

- memeriksa kesesuaian *pola deretan token* dengan aturan sintaks yang ditentukan dalam *bahasa sumber*. Sederetan token yang tidak mengikuti aturan sintaks akan dilaporkan sebagai *kesalahan sintaks* (*sintax error*). Secara logika deretan token yang bersesuaian dengan sintaks tertentu akan dinyatakan sebagai pohon parsing (*parse tree*)

Penganalisa Semantik

- memeriksa token dan ekspresi dari batasan-batasan yang ditetapkan. Batasan-batasan tersebut misalnya :

a. panjang maksimum token *identifier* adalah 8 karakter,

b. panjang maksimum ekspresi tunggal adalah 80 karakter,

c. nilai bilangan bulat adalah -32768 s/d 32767,

d. operasi aritmatika harus melibatkan operan-operan yang bertipe sama

Pembangkit Kode Antara

- membangkitkan kode antara (*intermediate code*) berdasarkan pohon parsing. Pohon parse selanjutnya diterjemahkan oleh suatu penerjemah yang dinamakan *penerjemah berdasarkan sintak* (*syntax-directed translator*). Hasil penerjemahan ini biasanya merupakan *perintah tiga alamat* (*three-address code*) yang merupakan representasi program untuk suatu *mesin abstrak*. Perintah tiga alamat bisa berbentuk *quadruples* (*op, arg1, arg2, result*), *tripels* (*op, arg1, arg2*). Ekspresi dengan satu argumen dinyatakan dengan menetapkan *arg2* dengan - (strip, *dash*)

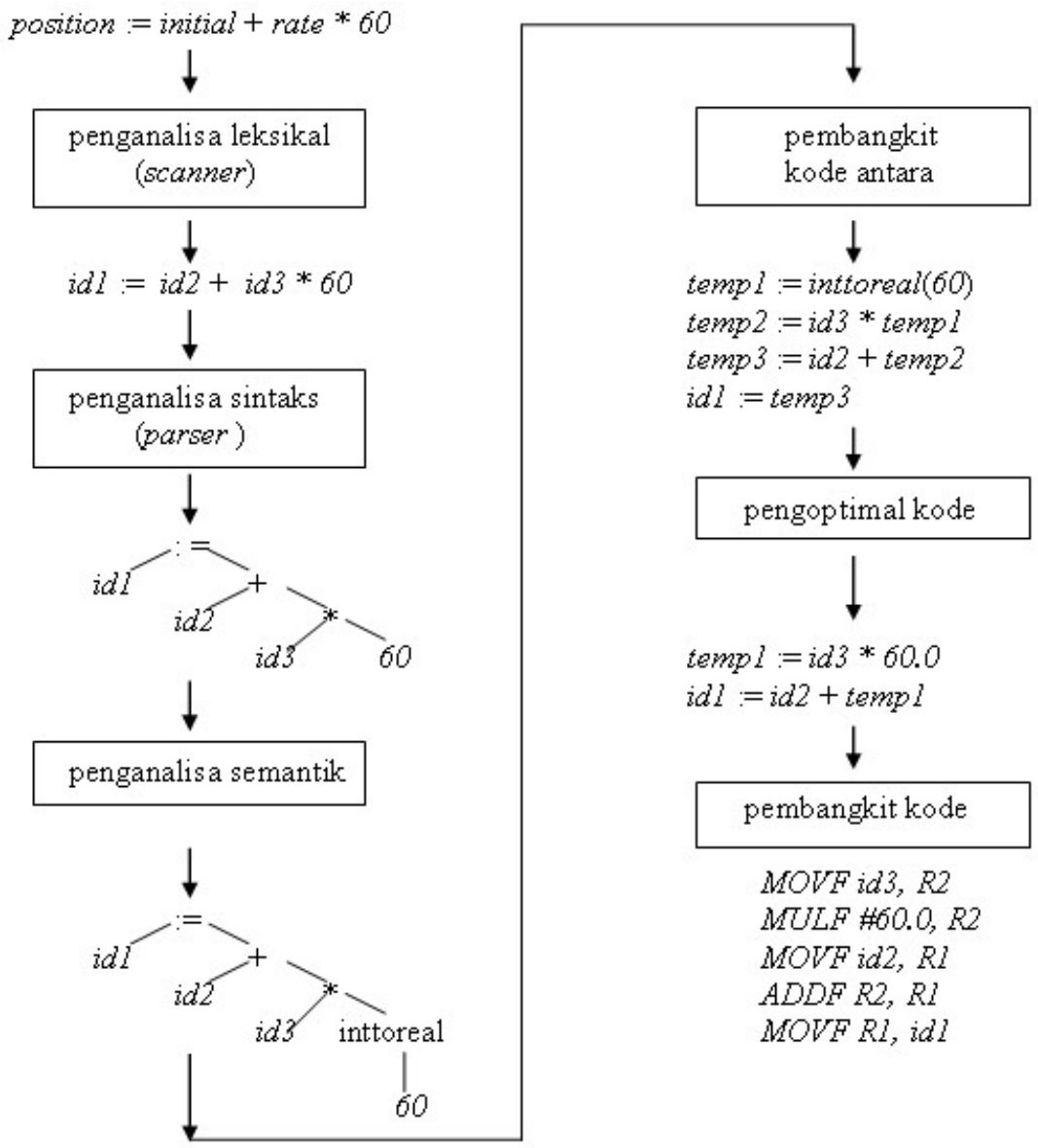
Pengoptimal kode

- melakukan optimasi (penghematan *space* dan *waktu komputasi*), jika mungkin, terhadap kode antara

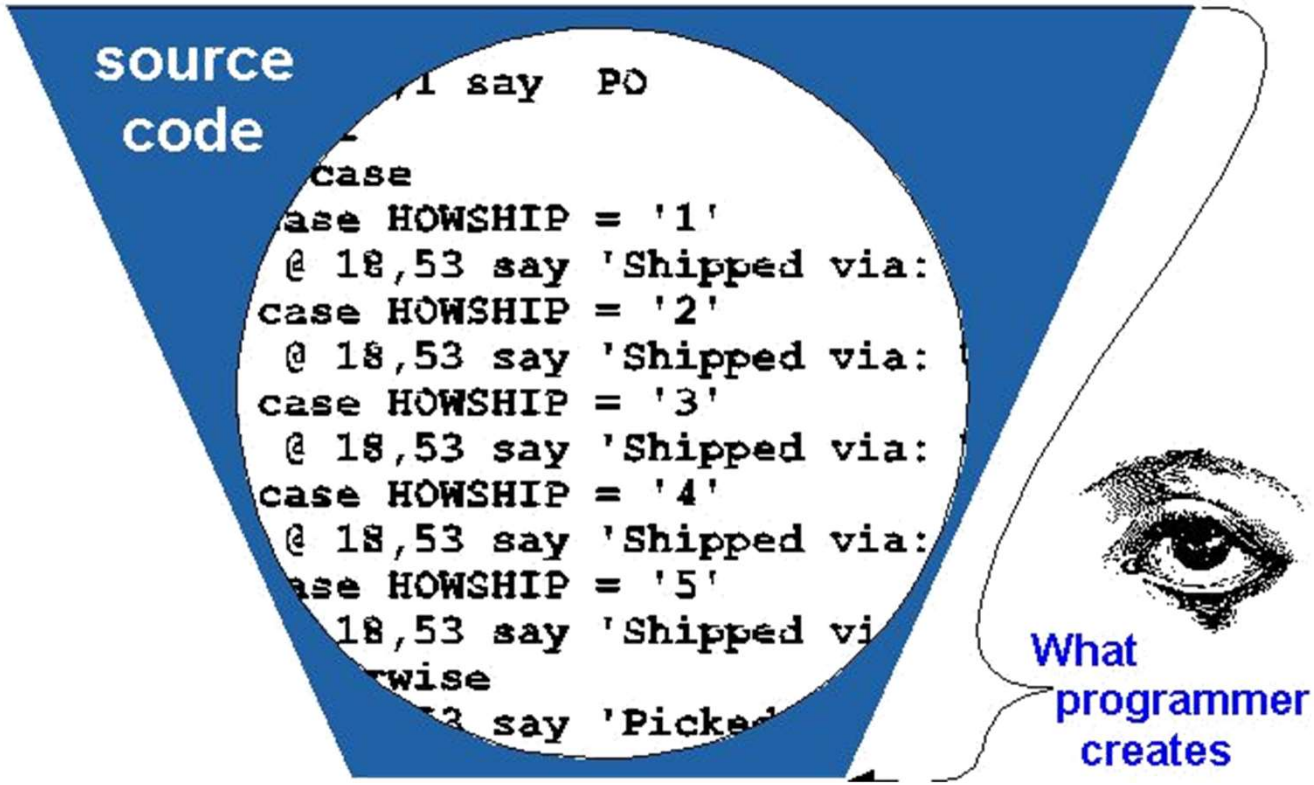
Pembangkit Kode Mesin

- membangkitkan kode dalam bahasa target tertentu (misalnya bahasa mesin)

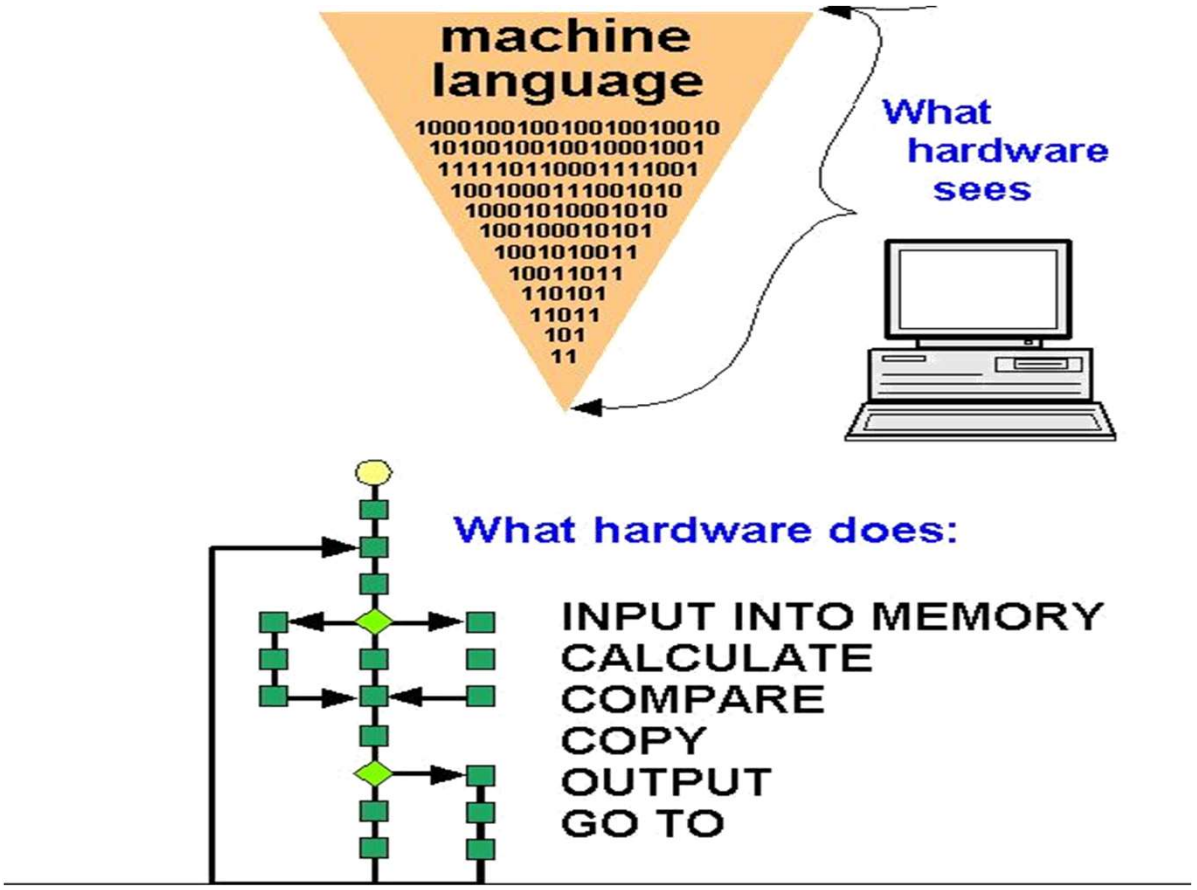
Contoh Kompilasi



View dari programmer



Mesin View


machine language
10001001001001001001010
10100100100100010001
11111011000111100 1
10010001110010 10
10001010001010
10010001010 1
1001010011
10011011
110101
11011
101
11
What hardware sees
What hardware does:
INPUT INTO MEMORY
CALCULATE
COMPARE
COPY
OUTPUT
GO TO

Pembuatan compiler

Bahasa mesin

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

Pembuatan compiler

Assembly

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

Pembuatan compiler

Bahasa Tingkat Tinggi (high level language)

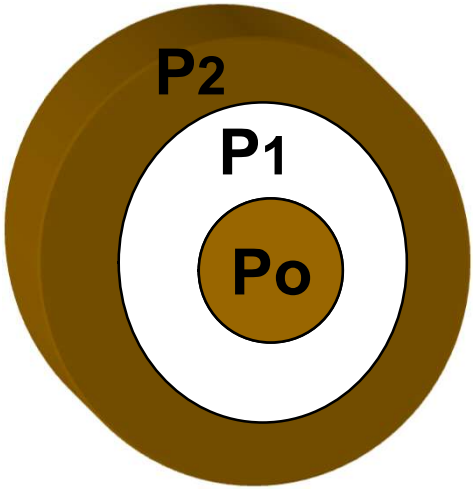
- Lebih mudah dipelajari
- Fasilitas yang dimiliki lebih baik (banyak)
- Memiliki ukuran yang relatif besar, misal membuat compiler pascal dengan menggunakan bahasa C
- Untuk mesin yang berbeda perlu dikembangkan tahapan-tahapan tambahan.
- Misal membuat compiler C pada Dos bedasarkan compiler C pada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

Pembuatan compiler

BootStrap

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

BootStrap



- 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

Contoh dari source program ke dalam kode mesin

IF COUNT =10
GOTO DONE
ELSE
GOTO AGAIN
ENDIF
	Compare A to B
	If equal go to C
	Go to D
		Compare 3477 2883
		If = go to 23883
		Go to 23343

10010101001010001010100
10101010010101001001010
10100101010001010010010

BAHASA SUMBER

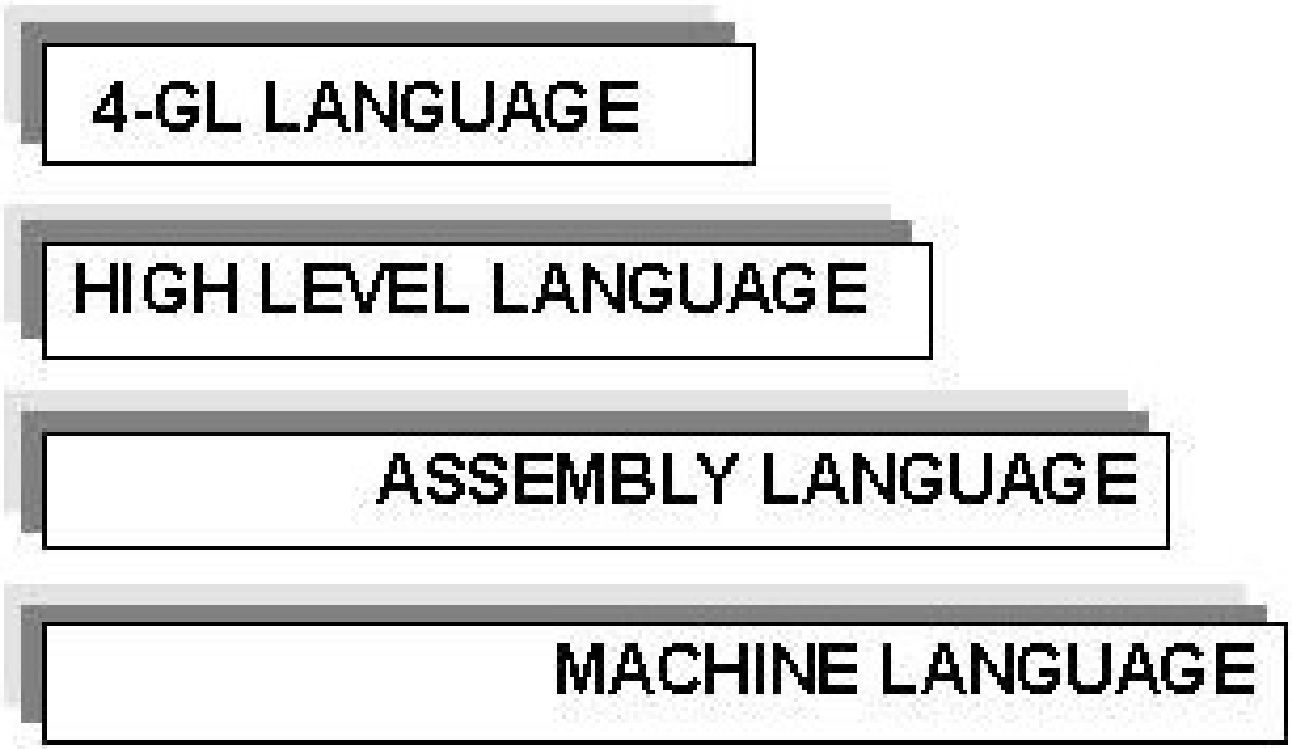
DEFINISI "bahasa sumber"

- 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*pi/16.0).*' adalah dua contoh kalimat lengkap dalam Bahasa Pemrograman Pascal.
- Dalam bahasa pemrograman *kalimat* lebih dikenal sebagai *ekspresi* sedangkan *kata* sebagai *token*

Bahasa Tingkat Tinggi (Pemrograman)

- Bahasa yang lebih dikenal oleh manusia, maksudnya adalah *statement* yang digunakan menggunakan bahasa yang dipakai oleh manusia (inggris),
- Bahasa pemrograman didefinisikan dengan menentukan bentuk programnya (sintak) dan arti programnya (semantik)
- Memberikan fasilitas yang lebih banyak, seperti struktur kontrol program yang terstruktur, blok-blok serta prosedur dan fungsi-fungsi
- Progam mudah untuk di koreksi (debug)
- Tidak tergantung pada salah satu mesin
- Kontrol struktur seperti : kondisi (if .. Then.. Else), perulangan (For, while), Struktur blok (begin.. End { .. })

Tingkatan Bahasa Pemrograman



Sumber perancangan bahasa

- Konstruksi yang diturunkan dari bahasa alami, karena bahasa alami dapat digunakan sebagai panduan untuk perancangan sintaks
- Matematika, misal untuk perancangan operasi aritmatika
- Bahasa pemrograman yang sudah ada.

Tujuan perancangan bhs program

- Komunikasi dengan manusia
- Pencegahan dan deteksi kesalahan
- Usability
- Efektifitas pemrograman
- Compilability (mengurangi kompleksitas,mis:penggunaan bracket)
- Efisiensi dengan meminimalisir ketidakcocokan antara hardware dengan bahasa

Tujuan perancangan bhs program(2)

- Machine independent
- Simplicity :penyederhanaan komponen bahasa program
- Orthogonality : kumpulan primitive yang dikombinasikan dengan berbagai cara dalam membangun kontrol dan struktur data dalam bahasa program

Struktur Ekspresi

Metode pengurutan evaluasi dalam ekspresi :

- Explicit Bracketing
- Operator binding

Binding adalah asosiasi antara atribut dan entity atau antara operasi dan simbol.

Binding time adalah waktu yang dialokasikan untuk menyatukan variable dengan nilainya.

Struktur Data

Empat aspek dalam struktur data

- Deklarasi data
- Tipe data yang tersedia
- Alokasi storage
- Lingkup variabel

Struktur I/O

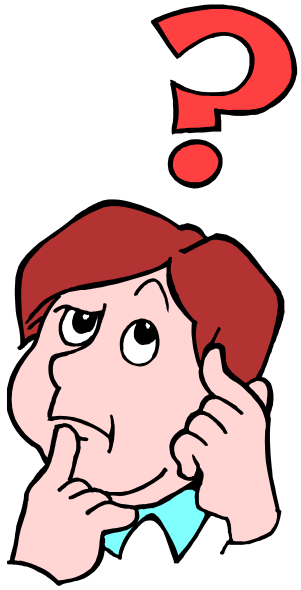
- Format free

  langsung ditampilkan sehingga mudah bagi user untuk memeriksa kebenaran program. Contoh pada VB.

- Formatted

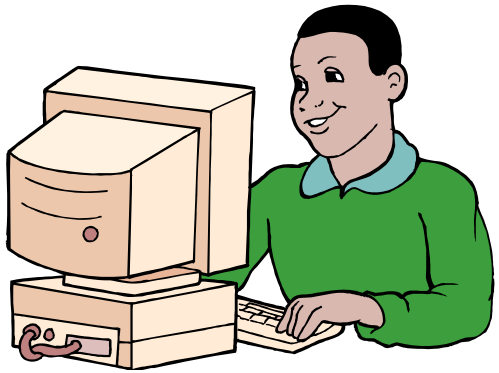
  output ditampilkan secara terformat, seperti di C : printf(), delphi/VB : format()

ANDAIKAN.....



Anda akan menciptakan sebuah bahasa program, coba sebutkan urutan proses yang harus ditentukan/skenario yang dijalani untuk menghasilkan bahasa pemrograman impian Anda tersebut !

SKENARIO PERANCANGAN



1. Tentukan apa yang diinginkan.
2. Tentukan feature yang mungkin
3. Tentukan desain dan sesuaikan dengan featurenya
4. Tentukan rincian, parsing, dan error checking.
5. Tuliskan user manual dan help.
6. Evaluasilah, jika salah mulai lagi dari langkah 3.
7. Jika sudah benar, optimisasilah dan uji segala kemungkinan.
8. Cobakan kepada pengguna, tunggu reaksinya.
9. Perbaiki bug dan mulai versi baru.

Tools Bantu Compiler

- Free Compiler Construction Tools
  http://www.thefreecountry.com/developercity/compiler.html

- TASSKAF. Bahasa TASSKAF ini merupakan subset dari Java. Dapat disusun suatu program ke byte code yang dapat dijalankan di Java Virtual Machine (JVM).
  Pada site tersebut juga tersedia informasi materi kuliah dengan LEX, YACC
  http://rw4.cs.uni-sb.de/~martin/COMP/TK/

- GENTLE. Gentle ini merupakan perangkat bantu (toolkit) modern untuk menulis compiler dan mengimplemntasikannya pada bahasa tertentu. Perangkat bantu ini mendukung semua proses translasi, dari definisi tree sintaks abstrak, pater matching, smart traversal dan lain sebagainya. Toolkit ini telah digunakan secara luas di riest dan industri .**http://www.first.gmd.de/gentle/**

- ELI. Merupakan suatu lingkungan pemrograman yang memungkinkan membuat suatu implementasi bahasa pemrograman secara lengkap dari suatu sepsifikasi. Perangkat bantu ini menangani struktural analisis, analisis nama, type, value dlsb dan akan menghasilkan kode C. **http://www.cs.colorado.edu/~eliuser/**

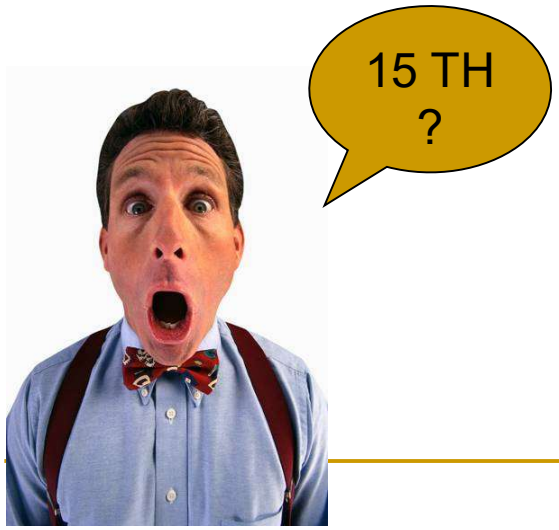
ANTLR, ANother Tool for Language Recognition, is a language tool that provides a framework for constructing recognizers, interpreters, compilers, and translators from grammatical descriptions containing actions in a variety of **target languages**

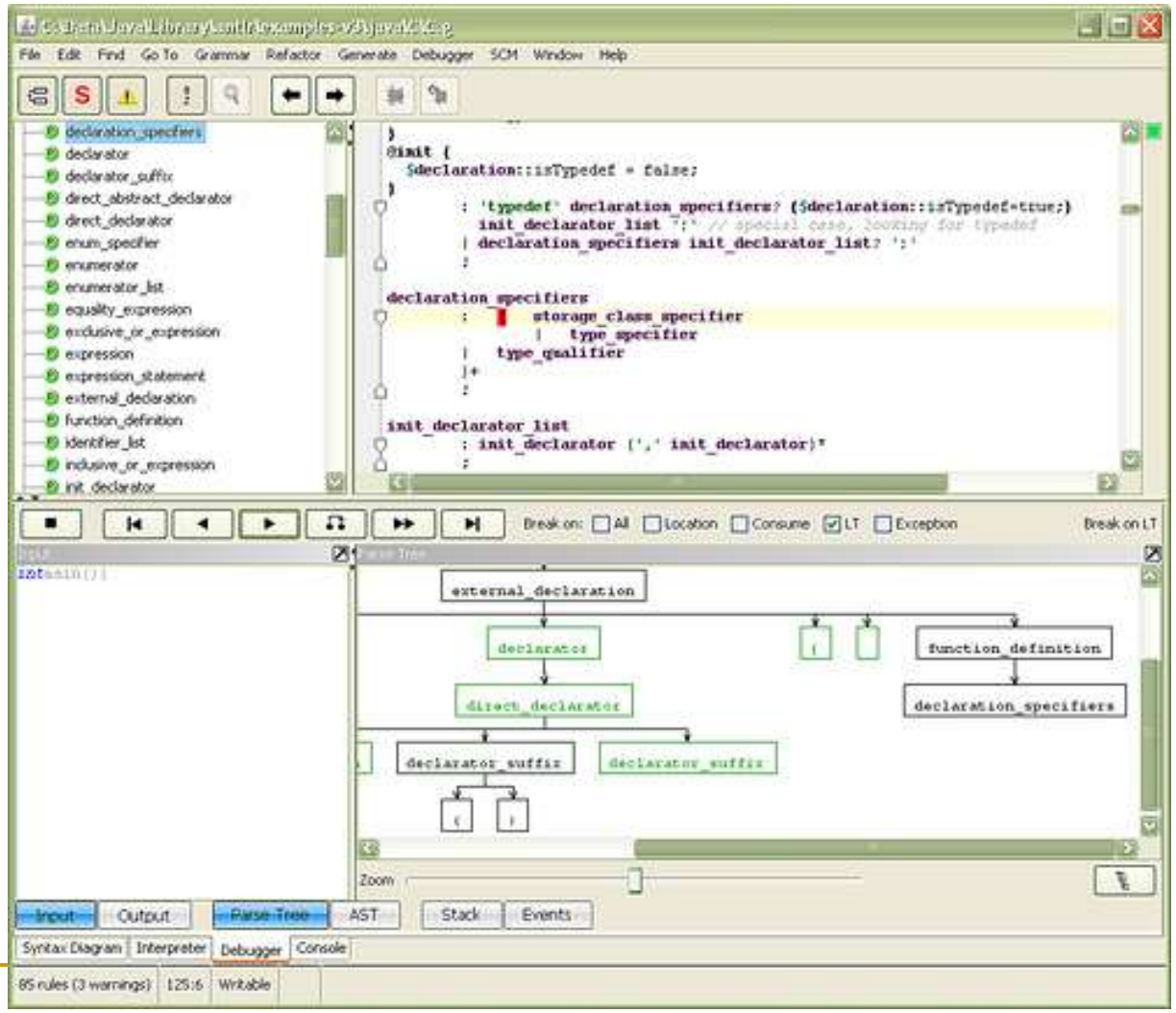
Made by :

Terrence Parr

<almost by himself>

FOR 15 YEARS





IDE SISTEM PEMBELAJARAN CERDAS

What's a Surprise?

Top Topic
Skripsi with
Compiler technique