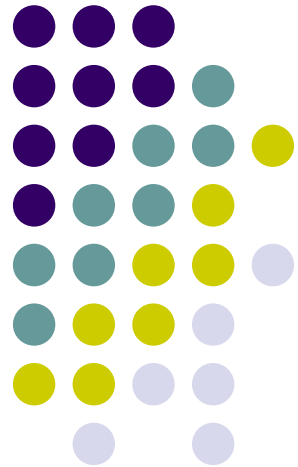
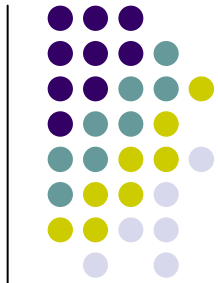
Teknik Kompiler 1

oleh: **antonius rachmat c, s.kom**

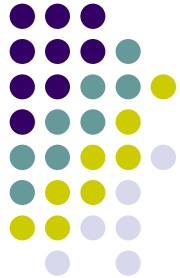


Bahasa Pemrograman



- 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**.

Jenis Bahasa Pemrograman



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:

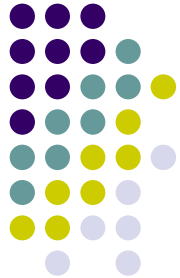
  untuk mesin IBM/370

  0001100000110101 = 1835 yang berarti mengkopikan isi register 5 ke register 3

Keuntungan: Eksekusi cepat

Kerugian: Sangat sulit dipelajari manusia

Jenis Bahasa Pemrograman (2)

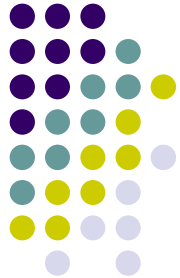


Bahasa Assembly (mnemonic code)

- Bahasa assembly adalah bahasa simbol dari bahasa mesin.
- 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 pen-debug-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.

Jenis Bahasa Pemrograman (3)



Bahasa Tingkat Tinggi (*High Level Language*)

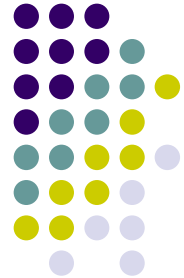
- Bahasa ini lebih dekat dengan bahasa manusia. Bahasa inilah yang akan dibahas pada matakuliah ini.
- Bahasa ini juga memberikan banyak sekali fasilitas kemudahan pembuatan program, misalnya: variabel, tipe data, konstanta, struktur kontrol, loop, fungsi, prosedur dan lain-lain. Contoh: Pascal, Basic, 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

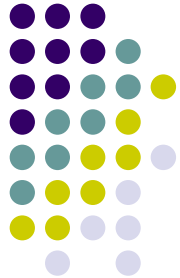
Jenis Bahasa Pemrograman (4)



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

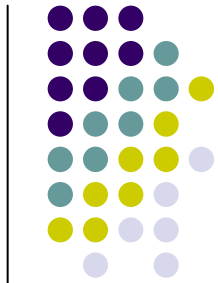
- Bahasa ini adalah bahasa yang digunakan langsung untuk memecahkan suatu masalah tertentu. Misalnya **SQL** untuk database, **Regex** untuk mencocokkan pola pada string tertentu.
- Jenis bahasa ini juga masuk ke bahasa tingkat tinggi.

Istilah-istilah



- Source language : jenis bahasa yang menjadi sumber.
- Source code: kode program yang akan dikompilasi/diinterpret.
- Object code: program hasil kompilasi/interpretasi.
- Object file: file hasil kompilasi, biasanya berekstensi .OBJ atau .O
- Target Machine: komputer yang digunakan untuk menjalankan program hasil interpretasi/kompilasi

Translator

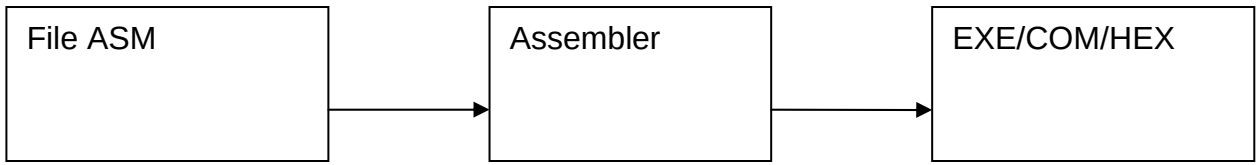


- Translator melakukan **pengubahan** source code ke dalam target code/object code/object program.
- Source code ditulis dengan suatu bahasa pemrograman tertentu sedangkan object code bisa bermacam-macam outputnya tergantung pada translatornya.

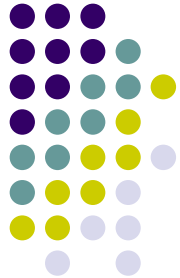
Jenis Translator

Assembler

- Source code adalah bahasa assembly, object code adalah bahasa mesin.



Jenis Translator (2)

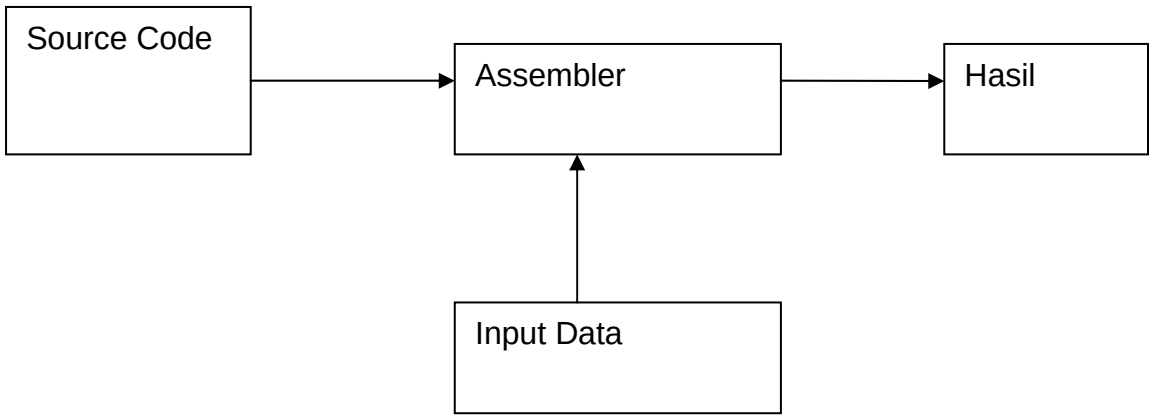


Interpreter

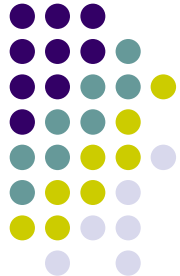
- Input berupa source code yaitu bahasa scripting seperti PHP, Perl, Javascript, ASP, Java bytecode, Basic.
- Interpreter tidak menghasilkan object code. Hanya menghasilkan **translasi internal**.
- Input dapat berasal dari source code maupun dari inputan program dari user.
- Source code dan inputan data user diproses pada saat yang bersamaan.
- Pada interpreter: program tidak harus dianalisis seluruhnya dulu, tapi **bersamaan** dengan jalannya program.

Jenis Translator (3)

- **Interpreter (2)**
- Keuntungan: mudah bagi user, debuging cepat
- Kekurangan: eksekusi program lambat, tidak langsung menjadi program *executable*.



Jenis Translator (4)

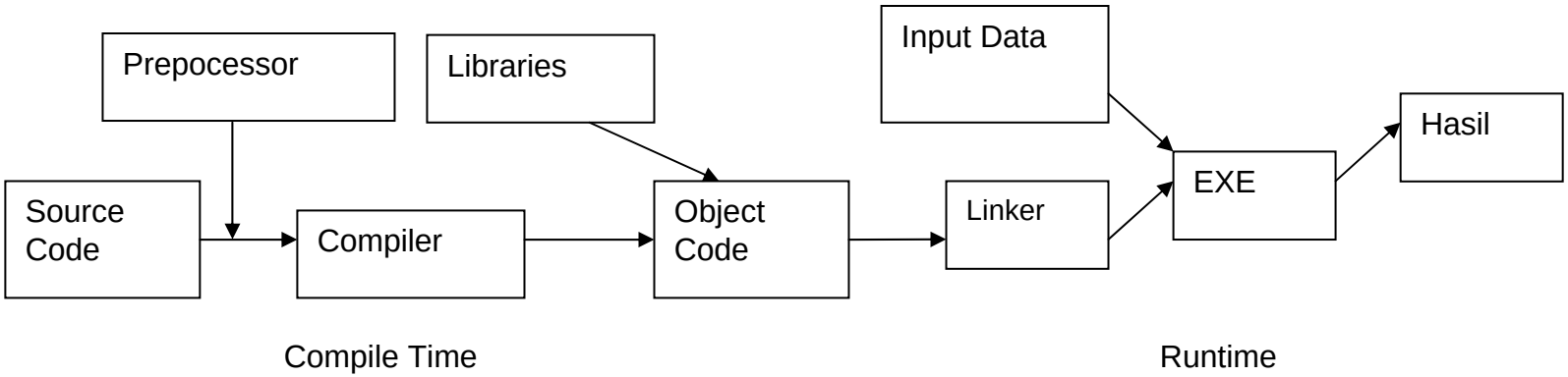


Kompiler

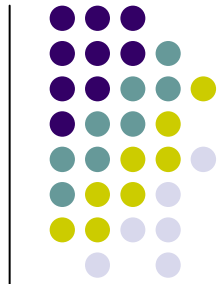
- Istilah kompiler muncul karena dulu ada program yang menggunakan subrutin-subrutin atau pustaka-pustaka untuk keperluan yang sangat khusus yang dikumpulkan menjadi satu sehingga diistilahkan *compiled.*
- Input berupa source code program seperti Pascal, C, C++.
- Object code adalah bahasa assembly.
- 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

Jenis Translator (5)

- **Kompiler (2)**
- Kekurangan: debugging lebih lambat
- Keuntungan: eksekusi program lebih cepat, menghasilkan file *executable* yang berdiri sendiri.



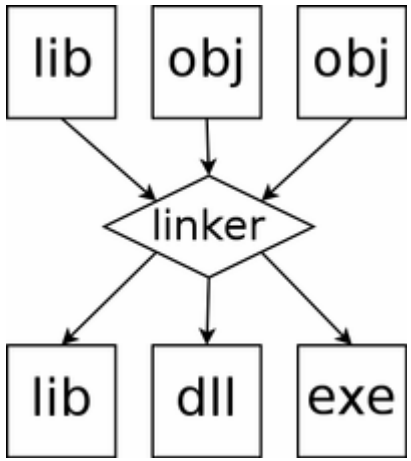
OBJ File



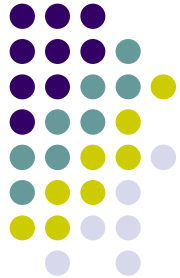
- DOS
  - .COM
  - DOS executable (MZ)
  - Relocatable Object Module Format (commonly known as "OBJ file" or "OMF"; also used in Microsoft Windows by some tool vendors)
- Macintosh
  - PEF/CFM
  - Mach-O (NEXTSTEP, Mac OS X)
- Unix
  - .out
  - Mach-O (NeXT, Mac OS X)
- Microsoft Windows
  - Portable Executable (PE)

Linker

- a **linker** or **link editor** is a program that takes one or more objects generated by compilers and assembles them into a single executable program
  - Dynamic Linking
  - Static Linking

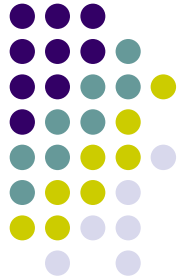


Tugas pokok kompiler



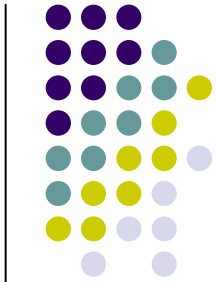
- **Analisis** di bagian *Front End*: menganalisis source code dan memecahnya menjadi bagian-bagian dasarnya. Menghasilkan kode level menengah dari source code input yang ada.
- **Sintesis** di bagian *Back End* : untuk *intermediate code optimization* dan *code generation*.

Alasan Back End & Front End



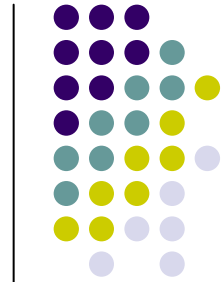
- Menyimpan hal-hal yang belum dapat terselesaikan pada bagian sebelumnya.
- Karena keterbatasan memori.

Yang harus dimiliki kompiler



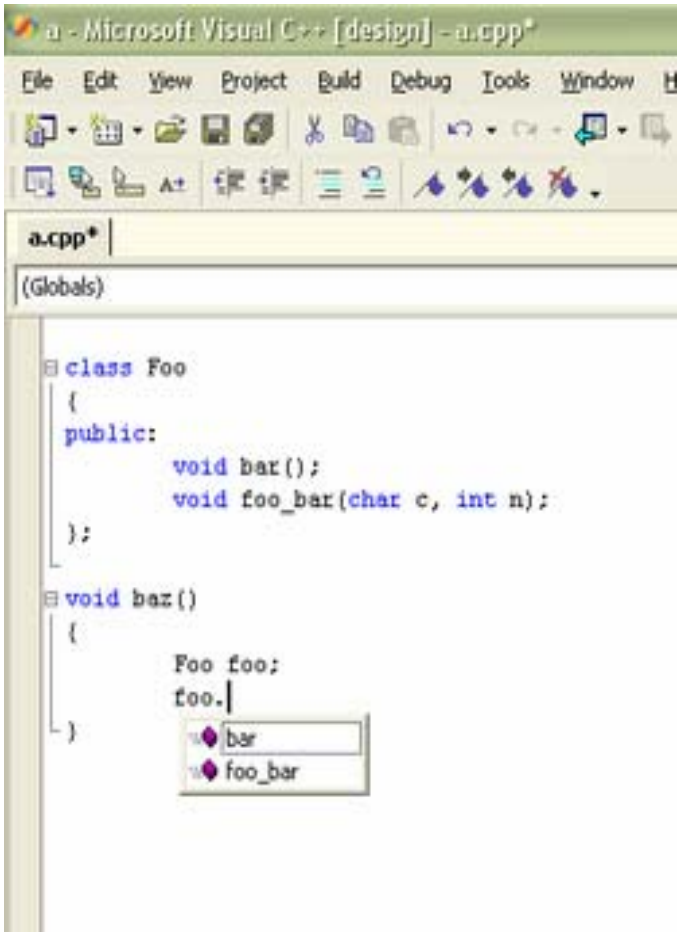
- ***Structure editor*** : menganalisis hirarki struktur source code. Misalnya memvalidasi apakah struktur source code benar atau salah. Contoh: *text auto completion/ auto correct*.
- ***Pretty printers*** : Fungsi ini akan menghasilkan source code yang ‘bersih’ dan ‘rapi’. Misalnya komentar bahasa pemrograman ditampilkan dengan font yang berbeda atau warna font yang berbeda dan termasuk bagaimana cara mengindentasi (pengaturan tab).
- ***Static checkers*** : Membaca program dan mengecek kemungkinan adanya bug dari program jika dieksekusi Fungsi ini sudah termasuk bagian dari tahap optimisasi kode, misalnya dapat mengetahui bagian kode mana yang tidak pernah dieksekusi, variabel belum didefinisikan, tipe data tidak cocok dan lain-lain.
- ***Interpreters*** : interpreter bertanggung jawab mentraslasikan source code secara internal dan disimpan ke dalam memori.

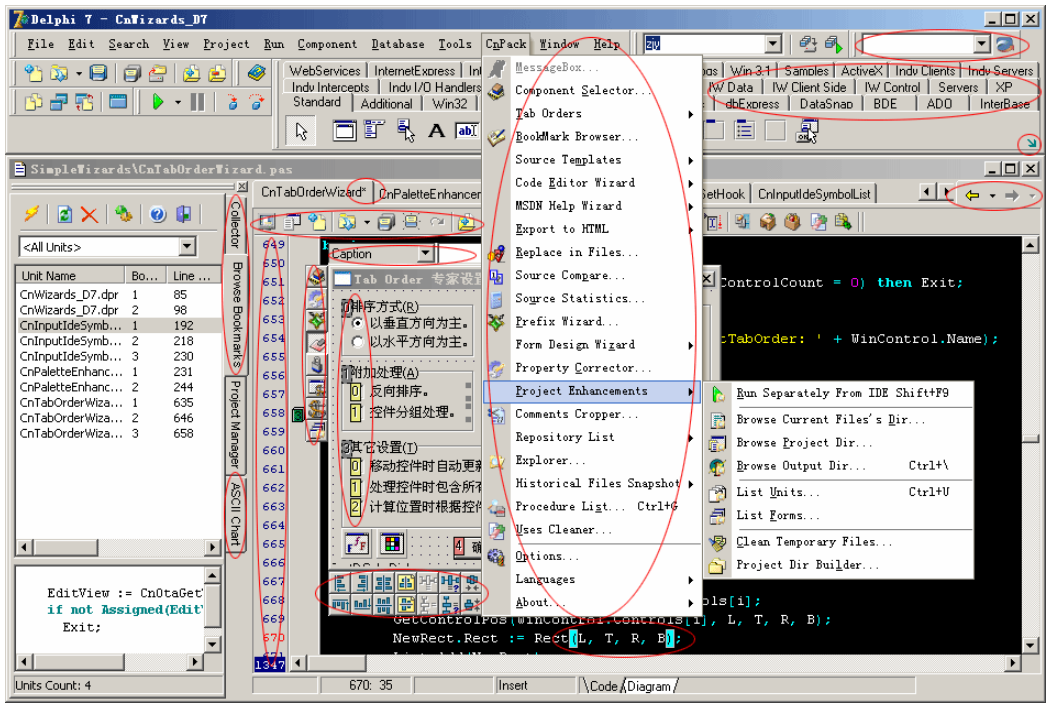
Intellisense, VisualAsistX, CnWizard



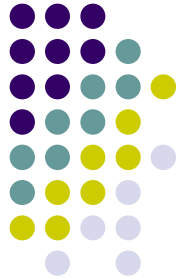
```
if (m_MouseMode == MOUSE_DRAGGING)
    m
```

? m_crShadow
? m_MouseMode
? m_nTimerID
? MOUSE_DRAGGING



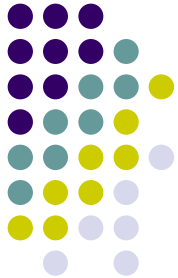


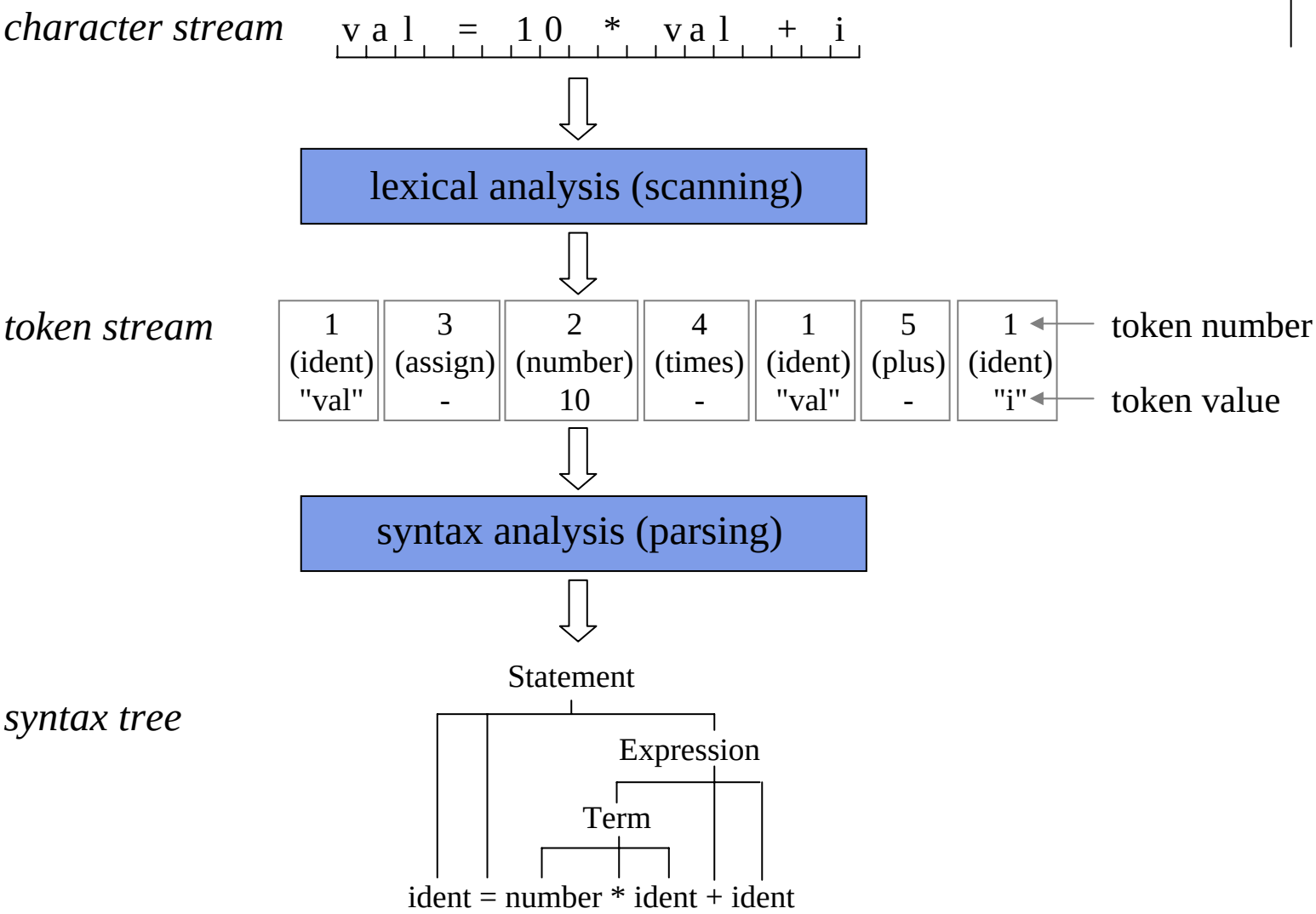
COMPILER PHASE



- ***Scannner/Lexical Analisys*** : memecah source code menjadi token-token, yaitu kumpulan karakter-karakter yang memiliki suatu arti.
- ***Parser/Syntax Analysis*** : memeriksa kebenaran token-token berdasarkan aturan-aturan sintaks, membentuk pohon sintaks.
- ***Semantic Analyzer*** : menganalisis semantik dengan mencocokkan arti secara keseluruhan. Biasanya akan digabungkan dengan *intermediate code generator*.
- ***Intermediate Code Generation:*** bahasa level menengah
- ***Intermediate Code Optimization*** : mengoptimasi kode level menengah
- ***Code Generation*** : menghasilkan bahasa assembly / bahasa mesin.
- ***Table Symbol*** : menyimpan semua informasi selama proses kompilasi.
- ***Object Code:*** target program

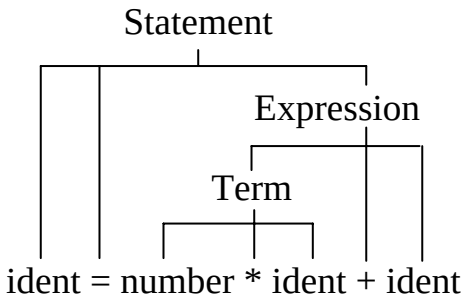
Dynamic Structure of a Compiler

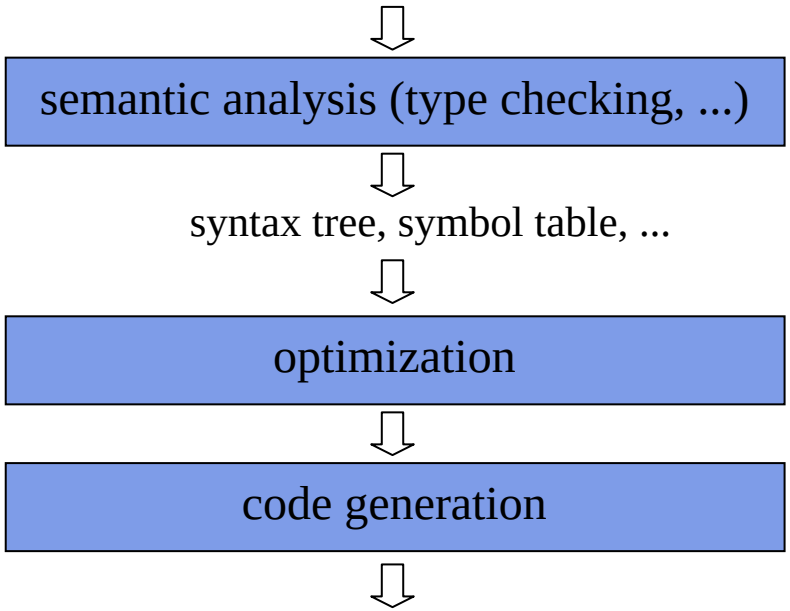




Dynamic Structure of a Compiler

syntax tree





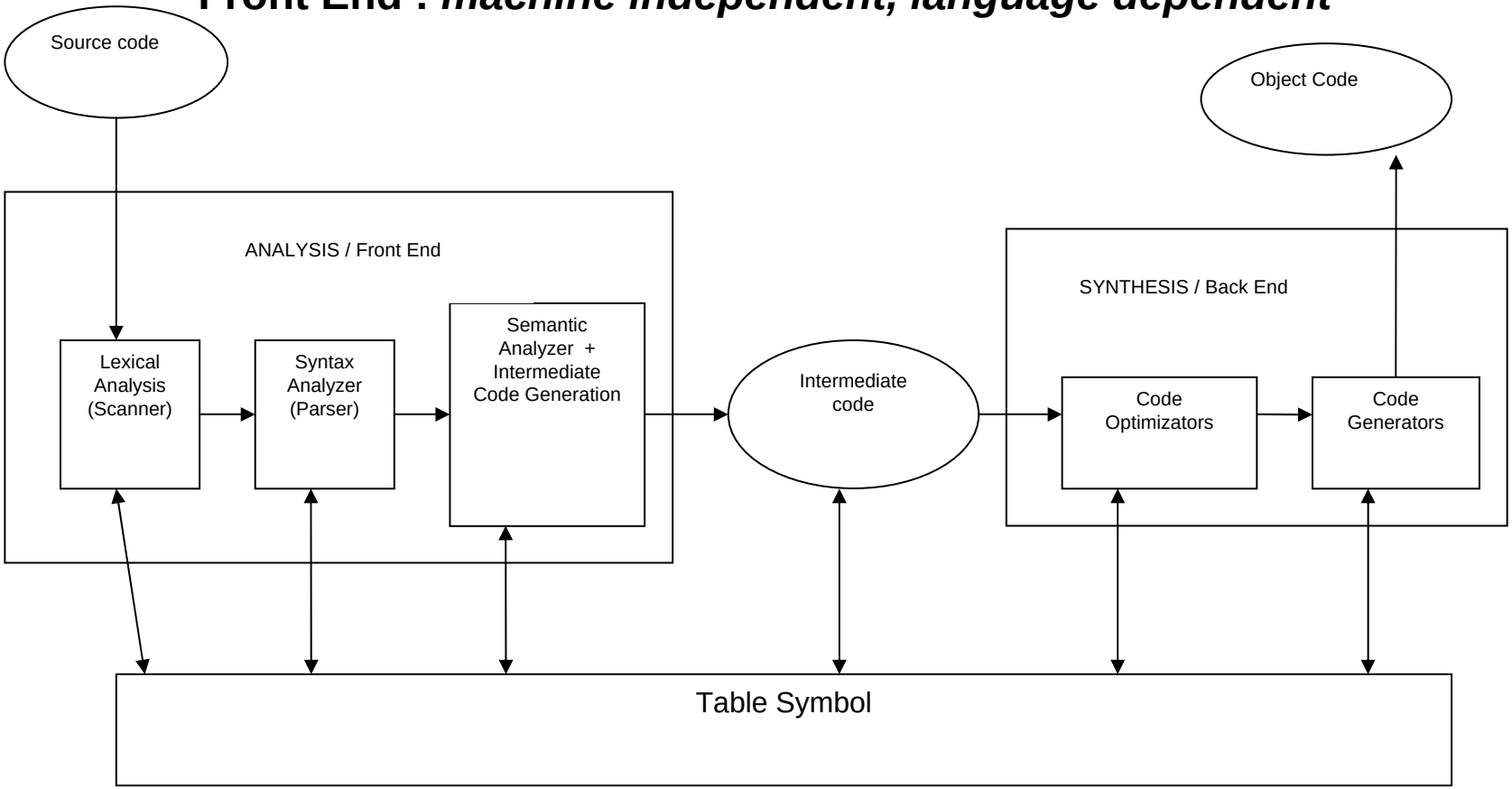
intermediate representation

machine code

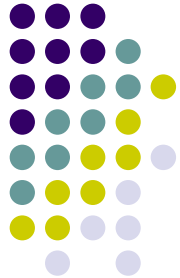
```
ld.i4.s 10
ldloc.1
mul
...
```

Diagram

Back End : *machine dependent, language independence*
Front End : *machine independent, language dependent*

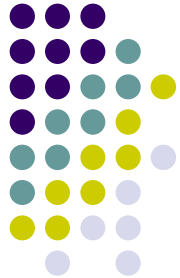


Prepocessor



- Kompiler juga membutuhkan suatu program lain sebelum dapat membuat suatu object code, yaitu *prepocessor*.
- *Prepocessor* adalah program yang mempu menggabungkan semua source code, yaitu antara source code itu sendiri dan modul-modul yang digunakannya, misalnya include file (seperti pada bahasa C : *include <stdio.h>* atau di Pascal : *uses crt*) dan makro yang kita definisikan.

Hal-hal penting *prepocessor*



Pemrosesan Makro

Makro adalah pemendekan dari suatu program. Caranya adalah mendefinisikan makro dan memanggilnya dengan atau tanpa parameter. Parameter yang didefinisikan pada definisi makro disebut parameter formal, sedangkan parameter yang digunakan untuk memanggilnya disebut parameter aktual.

Pengikutsertaan berkas (*include*)

Ex: include <stdio.h>

Prepocessor Rasional

Prepocessor ini memberi kemampuan baru dengan adanya *flow of control* (*while,if)* atau struktur data yang lebih baik. Contoh?

Perluasan Bahasa

Prepocessor ini memungkinkan suatu program berkomunikasi dengan bahasa lain. Misal untuk mengkases database.

Mutu Kompiler

- **Kecepatan dan waktu proses kompilasi**

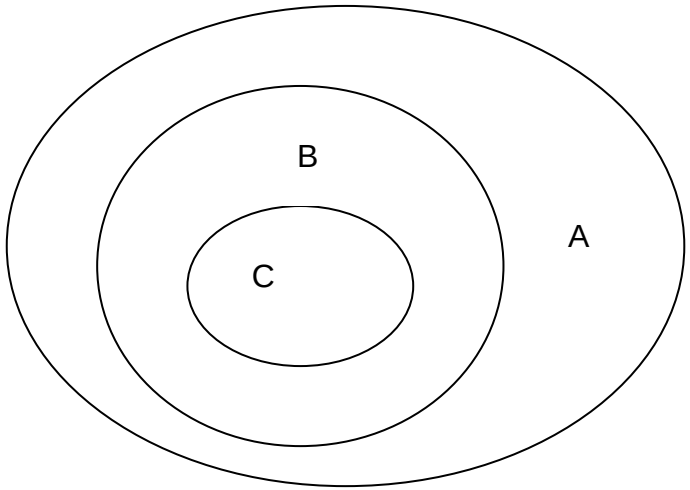
  Hal ini tergantung dari alogiritma untuk menulis kompiler itu dan kompiler pengkompilasi.
- **Mutu program objek**

  Dilihat dari ukuran dan kecepatan eksekusi program.
- ***Integrated Development Environtment* (IDE)**

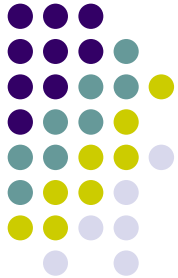
  Adalah fasilitas-fasilitas terintegrasi yang dimiliki oleh kompiler. Misalnya untuk debugging, editing, dan testing. Contoh : bandingkan antara kompiler Pascal dan Clipper.

Bootstrap

- Metode **Bootstrap** dikembangkan oleh Nikolaus Writh, penulis bahasa Pascal



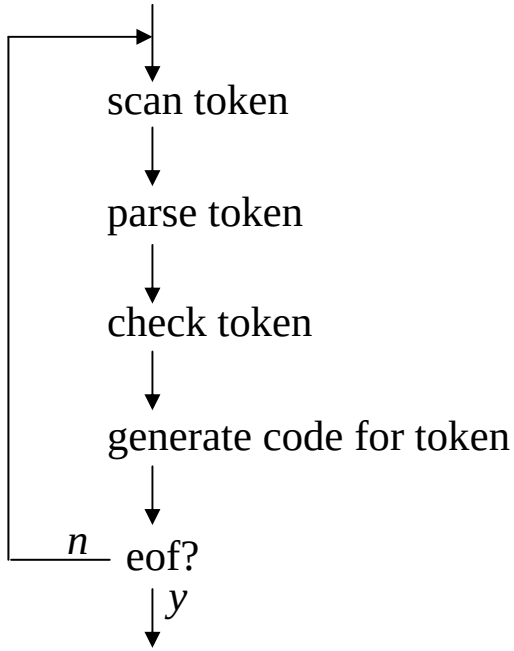
Bootstrap (2)



- Metode ini menganggap bahwa C dibangun dengan Assembly, B dibangun dengan C, dan A dibangun dengan B. Jadi kompiler dapat dibangan secara keseluruhannya dengan bahasa-bahasa sebelumnya.
- Metode Bootstrap berarti menulis suatu bahasa dengan kompiler versi sebelumnya.
- **Cross Compiler** adalah menulis suatu bahasa pada suatu mesin untuk menghasilkan program untuk mesin lain.

Single-Pass Compilers

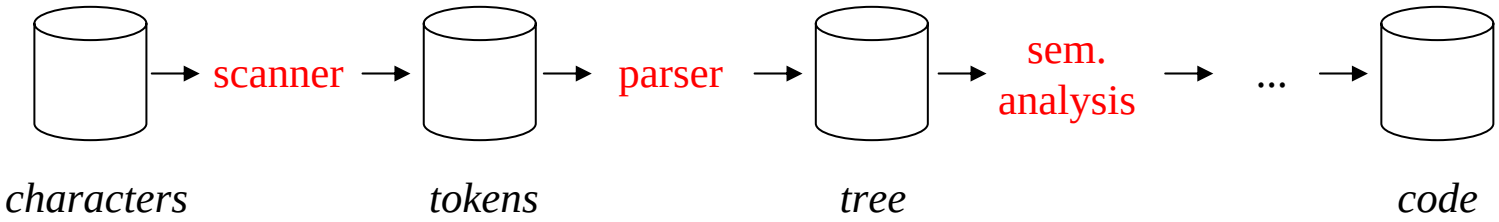
Phases work in an interleaved way



The target program is already generated while the source program is read.

Multi-Pass Compilers

Phases are separate "programs", which run sequentially

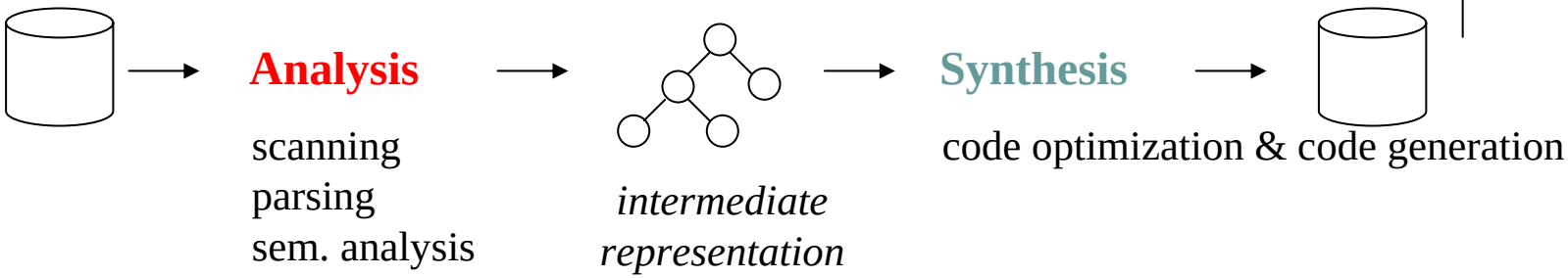


Each phase **reads** from a **file** and writes to a **new file**

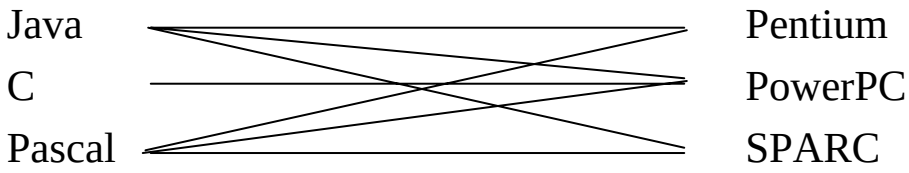
Why multi-pass?

- if memory is scarce (irrelevant today)
- if the language is complex
- if portability is important

Today: Often Two-Pass Compilers



language-dependent **machine-dependent**



any combination possible

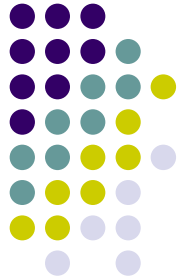
Advantages

- better portability
- many combinations between front ends and back ends possible
- optimizations are easier on the intermediate representation than on source code

Disadvantages

- slower
- needs more memory

NEXT



- Perancangan Bahasa Pemrograman…
- See u on monday! ☺