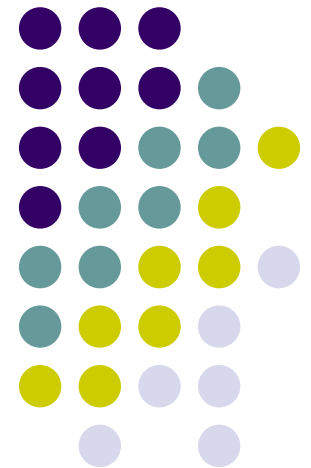


Teknik Kompiler 0

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



Teknik Kompiler (3 sks)



- **Tujuan :**

- Mempelajari teori-teori kompilasi, struktur mesin kompiler, dan pada akhirnya mahasiswa mampu menerapkan teori tersebut untuk membuat aplikasi suatu kompiler sederhana

- **Hari dan Waktu :**

- Setiap hari 8.00-10.00 (Blok 2)
- TTS: 14 juli dan TAS: 24 juli



Pengajar

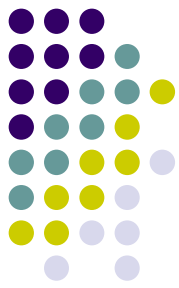
- **Dosen:** Antonius Rachmat C, S.Kom, M.Kom
- **Email:** anton@ukdw.ac.id
- **Website:** <http://lecturer.ukdw.ac.id/anton>
- **Blog:** <http://antonie.wordpress.com>
- **YM:** antonie_oo

Silabus



- Silabus & Pengantar Kompiler secara umum
- Teori Kompilasi
 - Kategori Bahasa Pemrograman, Translator, Model Kompilator, dan Mutu Kompilator
 - Struktur dan Fase Kompiler
- Perancangan Bahasa Pemrograman
- Regular Expression 1
 - Sintaks Regex
- Regular Expression 2
 - Sintaks Regex (lanjutan)
 - Regex dalam penggunaan (PHP&.NET) dan Contoh kasus

Silabus (2)



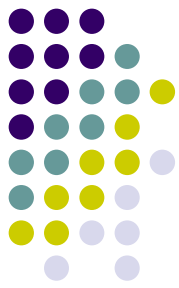
- Notasi Bahasa & Analisis Leksikal
 - Grammar
 - Hirarki Chomsky
 - Automata - Finite State Automata
 - DFA dan NFA
 - Token, dan Lexem
 - Diagram Transisi
- Analisis Sintaks
 - Sintaks
 - Tata Bahasa Bebas Konteks
 - NFA ke TBBK
 - Parsing : Top-down dan Bottom-Up
 - TBBK Rekursif kiri dan kanan dan solusinya

Silabus (3)



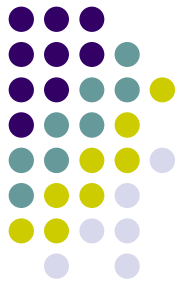
- Transformasi TBBK:
 - Penghilangan TBBK useless, produksi unit, dan produksi epsilon
 - Chomsky Normal Form (CNF) dan Algoritma Chocke, Youger, Kasami (CYK)
- Analisis Semantik
 - LL(1) dan Push Down Automata
 - Recursive Descent Parsing
 - Translasi Berdasarkan Sintaks
 - Tabel Simbol & Hashing

Silabus (4)



- Pengecekan Tipe & Intermediate Code
 - Type Checking
 - Tupple
 - Translasi Intermediate Code
 - Linking & Loading
- Memory Allocations & Runtime Environments
 - Storage
 - Runtime Environment
 - Activation Record
 - Procedure & Function Call / Return

Silabus (5)



- Code Optimization
 - Optimasi Lokal dan Global
- Code Generation
 - Result
 - Error Recovery



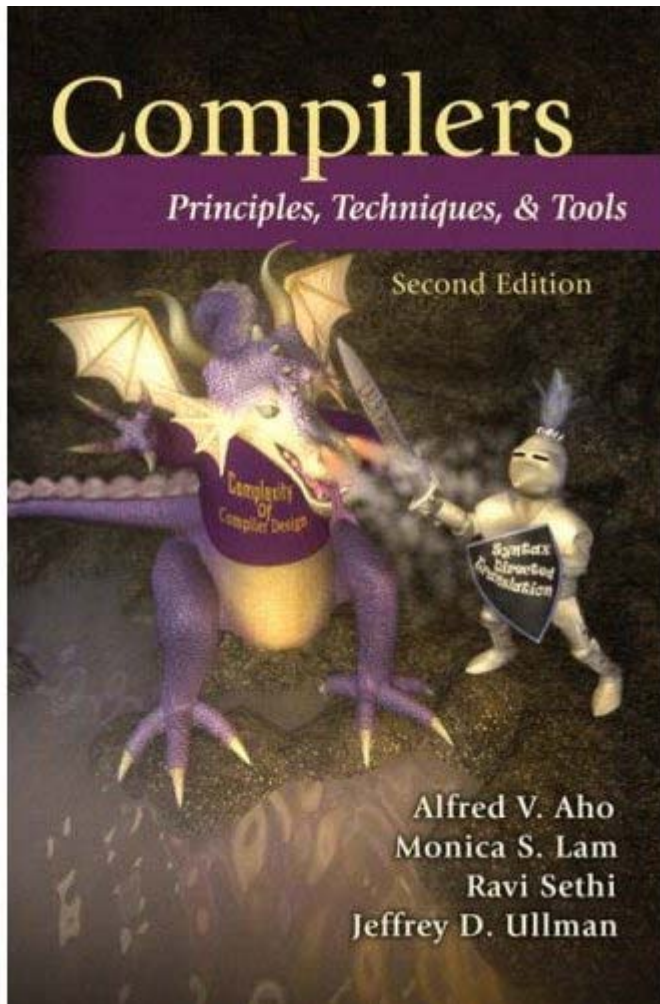
Daftar Pustaka

- Alfred V.Aho cs, Compilers Principles, Techniques, and Tools, 2003, Prentice-Hall
- Firrar Utadirartatmo, Teknik Kompilasi, 2001, Yogyakarta : J&J Learnings
- Firrar Utadirartatmo, Teori Bahasa dan Otomata, 2001, Yogyakarta : J & J Learnings
- Steven Haryanto, Regex Kumpulan Resep Pemrograman, 2004, Jakarta : Dian Rakyat



Daftar Pustaka (2)

- Thomas Pittman & James Peters, The Art of Compiler Design Theory and Practice, 1992, New Jersey : Prentice-Hall International Editions
- Jeffrey E.F. Friedl, Mastering Regular Expressions Powerful Techniques for Perl and Other Tools, 1997, USA : O'REILLY(TM)
- Donald A. Lewine Data General Corporation, POSIX Programmer's Guide Writing Portable UNIX Programs with the POSIX.1 Standard, 1991, USA : O'REILLY(TM)



Penilaian



- **Penilaian :**

● 85.0 - 100	A	4.0
● 80.0 - 84.9	A-	3.7
● 75.0 - 79.9	B+	3.3
● 70.0 - 74.9	B	3.0
● 65.0 - 69.9	B-	2.7
● 60.0 - 64.9	C+	2.3
● 55.0 - 59.9	C	2.0
● 45.0 - 54.9	D	1.0
● 0 - 44.9	E	0.0



Distribusi Nilai

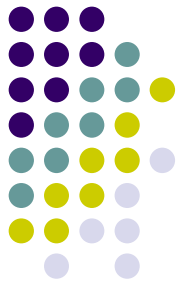
- Tes Tengah Semester : 25
- Tes Akhir Semester : 30
- Intiasari Jurnal 2 org : 15
- Tugas Program : 15
- Presensi @ 1 : 12
- Tes Kecil 2x : 10
- Jumlah : **107**

Kompiler dalam ilmu komputer



artificial intelligence	greedy algorithms learning algorithms
algorithms	graph algorithms union-find dynamic programming
theory	DFAs for scanning parser generators lattice theory for analysis
systems	allocation and naming locality synchronization
architecture	pipeline management hierarchy management instruction set use

Short History of Compiler Construction



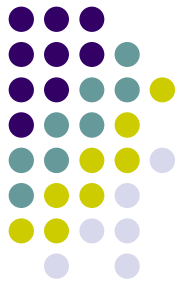
Formerly "a mystery", today one of the best-known areas of computing

- | | | |
|-------------|----------------|---|
| 1957 | Fortran | first compilers
(arithmetic expressions, statements, procedures) |
| 1960 | Algol | first formal language definition
(grammars in Backus-Naur form, block structure, recursion, ...) |
| 1970 | Pascal | user-defined types, virtual machines (P-code) |
| 1985 | C++ | object-orientation, exceptions, templates |
| 1995 | Java | just-in-time compilation |

We only look at **imperative languages**

Functional languages (e.g. Lisp) and **logical languages** (e.g. Prolog) require different techniques.

Why should I learn about compilers?



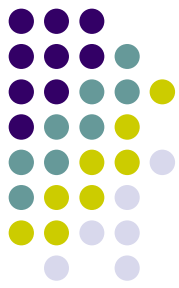
It's part of the general background of a software engineer

- How do compilers work?
- How do computers work?
(instruction set, registers, addressing modes, run-time data structures, ...)
- What machine code is generated for certain language constructs?
(efficiency considerations)
- What is good language design?

Also useful for general software development

- Reading syntactically structured command-line arguments
- Reading structured data (e.g. XML files, part lists, image files, ...)
- Searching in hierarchical namespaces
- Interpretation of command codes
- ... etc

Compiler



- Adalah sebuah program yang menerima input berupa source program (source language), kemudian ditranslasikan menjadi bentuk bahasa lain (target language) .
- Source language biasanya merupakan bahasa pemrograman tingkat menengah atau tinggi
- Target language biasanya berupa bahasa level rendah atau bahasa mesin
- Melaporkan error dan warning untuk membantu programmer
- Contoh kompiler selain untuk programming:
 - Mentranslasikan javadoc ke html
 - Mendapatkan hasil dari SQL query (query optimization)
 - Printer parsing PostScript file
 - Screen Scrapping
 - Konversi LaTeX ke PDF



Fase Kompiler

- Analisis (front-end)
 - Memecah source code dan menciptakan intermediate representation
 - language dependent
- Sintesis (back-end)
 - Membuat target program dari intermediate representation yang dihasilkan oleh tahap analisis
 - language independent

The compilation process

```

if (a < b * 3) {
    a *= 2;
}

```

Source Program

analysis
Front End

↓
Lexical analysis

↓
Syntactic analysis

↓
Semantic analysis

↓
Intermediate code generation

Error

```

x = 506;
y = "hello

```

```

foo(int,);
5 + * 6;

```

```

Bogus x;
f(int a, int a) {}

```

Ok

```

foo(int,)
5 + * 6;

```

```

Bogus x;
f(int a, int a)
{}

```

(valid program)

```

_t1 = b * 3
_t2 = a < _t1
Ifz _t2 Goto _L0
_t3 = a * 2
L0: ...

```

Intermediate Representation (IR)

synthesis
Back End

↓
IR optimization

↓
Object code generation

↓
Object code optimization

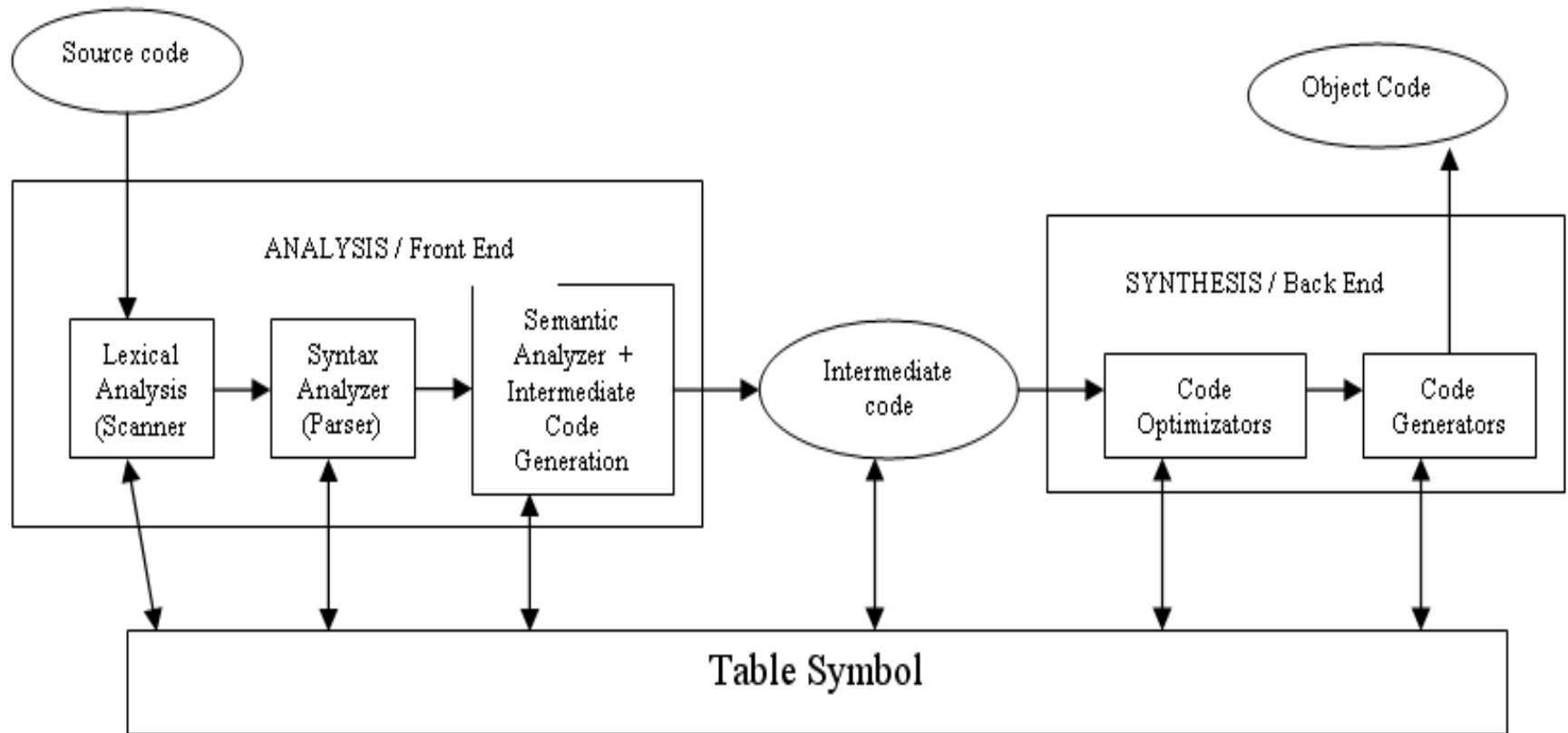
```

li $t0, 3
lw $t1, -12($fp)
mul $t2, $t1, $t0
lw $t3, -8($fp)
slt $t4, $t3, $t2
...

```

Target Program

Fase Kompilasi





Analisis Stage

- First step: recognize letters and words.
 - Words are smallest unit above letters

This is a sentence.

- Note the
 - Capital “T” (start of sentence symbol)
 - Blank “ ” (word separator)
 - Period “.” (end of sentence symbol)



Analisis Stage

- Scanning / Lexical Analysis: memecah source program ke dalam token dan lexem

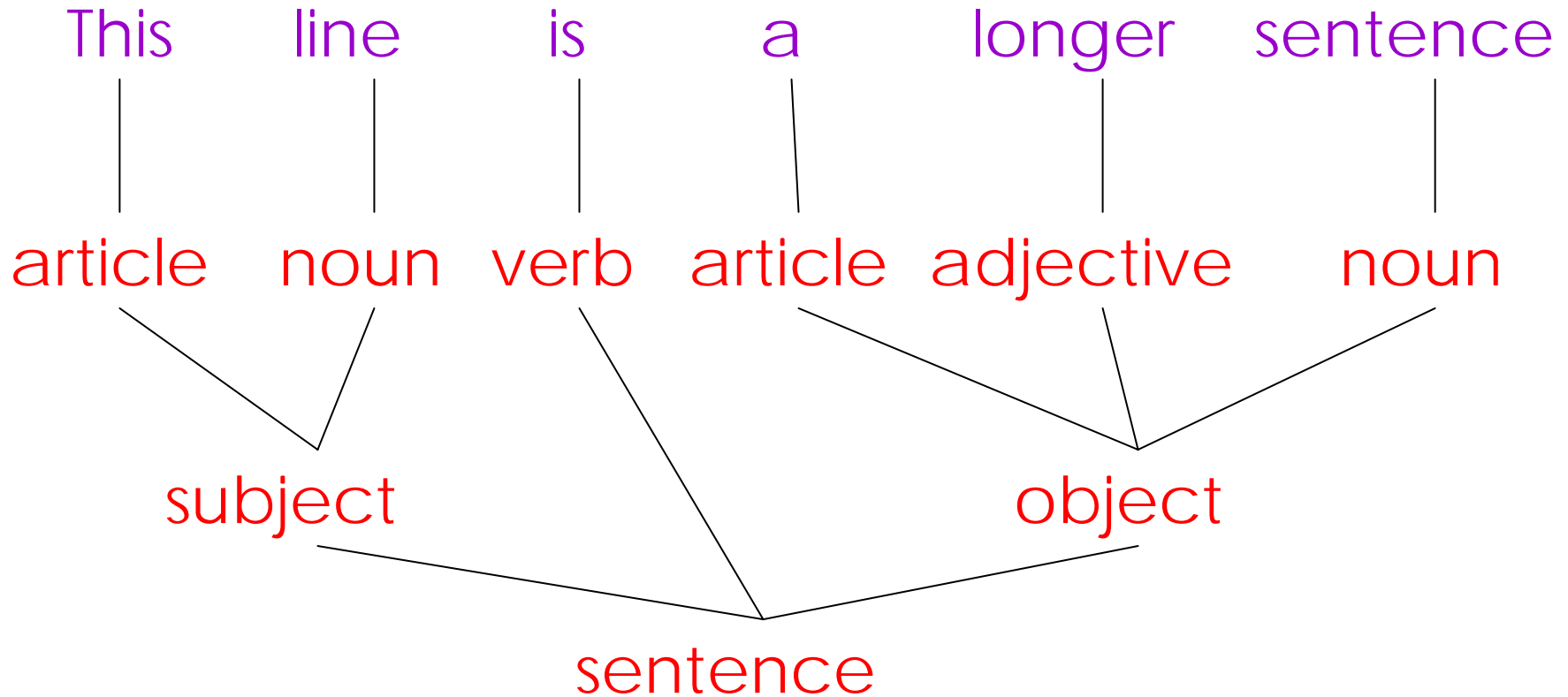
Example of lexical analysis:

```
int a;  
a = a + 2;
```

A lexical analyzer scanning the code fragment above might return:

```
int      T_INT (reserved word)  
a        T_IDENTIFIER (variable name)  
;        T_SPECIAL (special symbol with value of ";")  
a        T_IDENTIFIER (variable name)  
=        T_OP (operator with value of "=")  
a        T_IDENTIFIER (variable name)  
+        T_OP (operator with value of "+")  
2        T_INTCONSTANT (integer constant with value of 2)  
;        T_SPECIAL (special symbol with value of ";")
```

Diagramming a Sentence



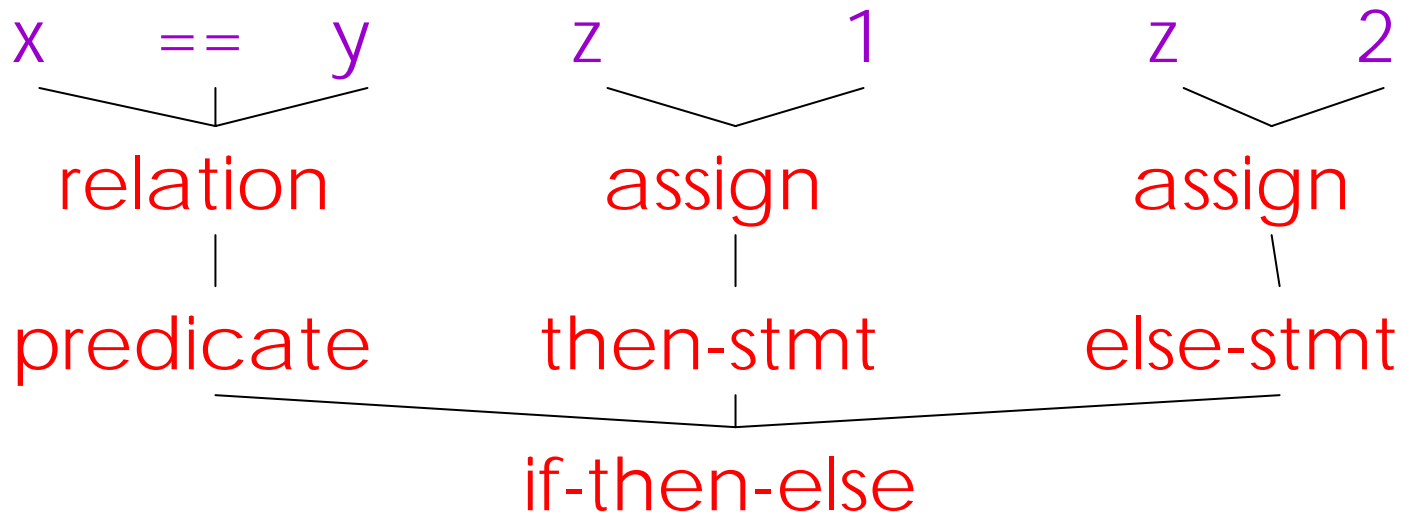


Parsing Programs

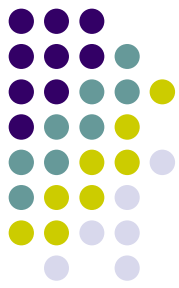
- Parsing program expressions is the same
- Consider:

If $x == y$ then $z = 1$; else $z = 2$;

- Diagrammed:



Analysis Stage



- Parsing / Syntax Analysis

- Mengelompokkan daftar hasil scanning ke dalam aturan grammar Tata Bahasa Bebas Konteks dan divalidasi

```
Expression -> Expression + Expression |  
            Expression - Expression |  
            ...  
            Variable |  
            Constant |  
            ...  
Variable -> T_IDENTIFIER  
Constant -> T_INTCONSTANT | T_DOUBLECONSTANT
```

```
Expression -> Expression + Expression  
            -> Variable + Expression  
            -> T_IDENTIFIER + Expression  
            -> T_IDENTIFIER + Constant  
            -> T_IDENTIFIER + T_INTCONSTANT
```

Semantic Analysis in English



- Example:

Jack said Jerry lost his assignment yesterday.

What does “his” refer to? Jack or Jerry?

- Even worse:

How many Jacks are there?

Which one lost the assignment?

Semantic Analysis in Programming



- Programming languages define strict rules to avoid such ambiguities
- This C++ code prints “4”; the inner definition is used

```
{  
    int Jack = 3;  
    {  
        int Jack = 4;  
        cout << Jack;  
    }  
}
```



More Semantic Analysis

- Compilers perform many semantic checks besides variable bindings
- Example:

Jack lost her homework yesterday.
- A “type mismatch” between **her** and **Jack**; we know they are different people
 - Presumably Jack is male



Example: Semantic

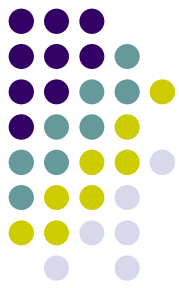
- $X = a + b * c - d / f$
- What are the correct solutions?
 - $X = (a + b) * (c - d) / f$
 - $X = a + ((b * c) - d) / f$
 - $X = a + (b * c) - (d / f)$
 - $X = ((a + b) * c) / f$
- Confused!



Analysis Stage

- Intermediate Code Generation: menghasilkan bahasa level menengah.

```
a = b * c + b * d    _t1 = b * c
                    _t2 = b * d
                    _t3 = _t1 + _t2
                    a = _t3
```



Syntesis Stage

- Intermediate Code Optimization

Example of code optimization:

```
_t1 = b * c          _t1 = b * c
_t2 = _t1 + 0        _t2 = _t1 + _t1
_t3 = b * c          a = _t2
_t4 = _t2 + _t3
a = _t4
```

- Object Code Generation: menjadi bahasa mesin
- Object Code Optimization



Optimization is tricky

```
for (i = 0; i < N; i += 1)  
    A[i] *= D/A[0];
```

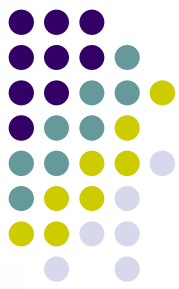
should be

```
tmp1 = D/A[0];  
for (i = 0; i < N; i += 1)  
    A[i] *= tmp1;
```




More Example: Bubble Sort

```
for (i = n-2; i >= 0; i--) {
    for (j = 0; j <= i; j++) {
        if (A[j] > A[j+1]) {
            temp = A[j];
            A[j] = A[j+1];
            A[j+1] = temp;
        }
    }
}
```



Code Generated

```
    i = n-2
S5:if i<0 goto s1
    j = 0
s4:if j>i goto s2
    t1 = 4*j
    t2 = &A
    t3 = t2+t1
    t4 = *t3      ;A[j]
    t5 = j+1
    t6 = 4*t5
    t7 = &A
    t8 = t7+t6
    t9 = *t8      ;A[j+1]
    if t4 <= t9 goto s3
    t10 = 4*j
    t11 = &A
    t12 = t11+t10
    temp = *t12   ;temp=A[j]
    t13 = j+1
    t14 = 4*t13
    t15 = &A
    t16 = t15+t14
    t17 = *t16    ;A[j+1]
    t18 = 4*j
    t19 = &A
    t20 = t19+t18 ;&A[j]
    *t20 = t17    ;A[j]=A[j+1]
    t21 = j+1
    t22 = 4*t21
    t23 = &A
    t24 = t23+t22
    *t24 = temp   ;A[j+1]=temp
s3:j = j+1
    goto S4
S2:i = i-1
    goto s5
s1:
```

(t4=*t3 means read memory at address in t3 and write to t4:
*t20=t17 :store value of t17 into memory at address in t20)

After optimization



```
i = n-2
t27 = 4*i
t28 = &A
t29 = t27+t28
t30 = t28+4
S5:if t29 < t28 goto s1
    t25 = t28
    t26 = t30
s4:if t25 > t29 goto s2
    t4 = *t25      ;A[j]
    t9 = *t26      ;A[j+1]
    if t4 <= t9 goto s3
    temp = *t25    ;temp=A[j]
    t17 = *t26     ;A[j+1]
    *t25 = t17     ;A[j]=A[j+1]
    *t26 = temp    ;A[j+1]=temp
s3:t25 = t25+4
    t26 = t26+4
    goto S4
S2:t29 = t29-4
    goto s5
s1:
```

Hal-hal lain tentang Kompiler



- Tabel Simbol
- Error Handling
- Programming language design
- Programming paradigms

- Compiler today??

NEXT

- Teori Kompiler
- Fase Kompilasi – detail
- One pass & multi pass compilation

