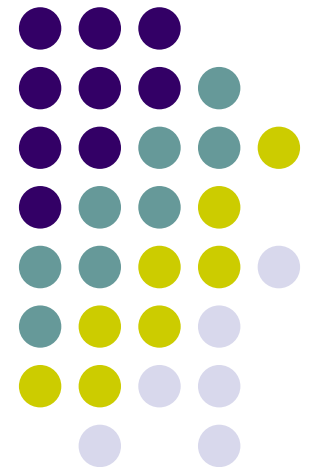


Teknik Kompiler 9

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



Pengecekan Tipe



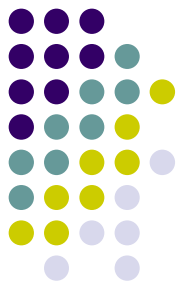
- Apakah tipe data dari variabel atau konstanta yang dibuat?
- Variabel dapat berasal dari pendeklarasian ataupun pengisian nilai pada saat eksekusi program.
- Tipe data dapat diprediksi nilainya dengan pendeklarasian tipe data, seperti integer, float, dan lain-lain.
- Type Errors: jika terjadi ketidaksesuaian operasi dalam eksekusi program. Mungkin hal ini karena inputan dari user.

Pengecekan Tipe (2)



- *Type Binding:*
 - Mendefinisikan sesuai dengan yang dideklarasikan di dalam program, Variabel bisa secara eksplisit (Ex: `int x`) ataupun implisit (Ex: `x = 1`).
 - Kesalahan penulisan dapat diminimisasi dengan cara membuat IDE yang dilengkapi dengan code completion, code suggestion, atau intellisense editor. Lihat contohnya pada VB, Delphi, Netbeans, .NET. Atau bisa menggunakan tool tambahan seperti VisualAssistX, CodeRush, atau CnPackWizard.

Pengecekan Tipe (3)



- *Type Checking:*
 - Apakah program memiliki tipe data yang sesuai dengan definisinya?
 - Gunakan analisis semantik untuk mengatasinya.
 - Contoh error:
 - Unary and binary operators (Ex: +, ==, []) harus menerima operand yang sesuai tipenya
 - Fungsi harus menghasilkan kembalian dan mendapat parameter yang sesuai dengan tipenya
 - Dalam assignment, tipe harus sesuai dengan definisinya
 - Anggota class harus memiliki hak akses yang benar



Intermediate Code

- Notasi Postfix

Biasanya kita menggunakan notasi infix dimana operator diletakkan ditengah-tengah antara operand-operand. Notasi Postfix adalah notasi dimana operator-operator yang ada diletakkan di paling belakang. Notasi postfix disebut juga notasi *Reverse Polish*.

`<operan><operan><operator>`

Contoh:

Infix : $(a*b) + (c-d)$

Postfix : $ab*cd-+$

BR: Branch

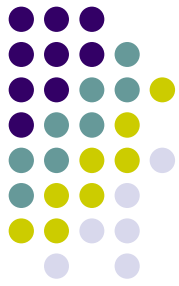
BZ: Branch if Zero



Intermediate Code (2)

- Contoh:
`if a>b then`
 `c:=d`
`else`
 `c:=e;`
- Maka implementasi postfix kode antara:
 - 1 a
 - 2 b
 - 3 >
 - 4 goto 11
 - 5 BZ
 - 6 c
 - 7 d
 - 8 :=
 - 9 goto 14
 - 10 BR
 - 11 c
 - 12 e
 - 13 :=
 - 14

Intermediate Code (3)



- Contoh:
a:=1
while a<5 do
 a := a+1

Dalam notasi postfix:

- 1 a
- 2 1
- 3 :=
- 4 a
- 5 5
- 6 <
- 7 goto 16
- 8 BZ
- 9 a
- 10 a
- 11 1
- 12 +
- 13 :=
- 14 goto 4
- 15 BR
- 16



N-Tuple (1)

- Notasi ini terdiri dari beberapa tupel.
Format umumnya:
- **operator ... N-1 operan**
- Notasi 3-Tuples (*3 of representation code*)
Contoh instruksi: $a := d * c + b / e$
- 3-tupel:
 1. $d, *, c$ atau $*, d, c$
 2. $b, /, e$ atau $/, b, e$
 3. (1), $+$, (2) atau $+$, (1), (2)
 4. $a, :=,$ (3) atau $:=, a,$ (3)



N-Tuple (2)

- Contoh :
If $x > y$ then
 $x := a - b$
else
 $x := a + b$
- 3-tupel:
 1. $>$, x , y
 2. BZ, (1), (6)
 3. $-$, a , b
 4. $:=$, x , (3)
 5. BR, , (8)
 6. $+$, a , b
 7. $:=$, x , (6)



N-Tuple (3)

- Contoh:

$A := B + C * D / E$

$F := C * D$

- 3-Tuples

- *, C, D

- /, (1), E

- +, B, (2)

- :=, A, (3)

- :=, F, (1)



N-Tuple (4)

- 4-Tuple

- ❖ Example

$a = (b+c) * (-e)$

$t1 = b + c$
 $t2 = -e$
 $a = t1 * t2$

Compiler-generated
temporary variable

- ❖ Assignment instructions

- » $a = b \text{ OP } c$ (binary op)
 - arithmetic: ADD, SUB, MUL, DIV, MOD
 - logic: AND, OR, XOR
 - comparisons: EQ, NEQ, LT, GT, LEQ, GEQ
- » $a = \text{OP } b$ (unary op)
 - arithmetic MINUS, logical NEG
- » $a = b$: copy instruction
- » $a = [b]$: load instruction
- » $[a] = b$: store instruction
- » $a = \text{addr } b$: symbolic address

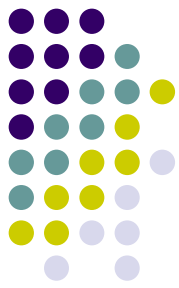
- ❖ Flow of control

- » label L: label instruction
- » jump L: unconditional jump
- » cjump a L : conditional jump

- ❖ Function call

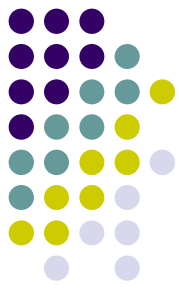
- » call $f(a1, \dots, a_n)$
- » $a = \text{call } f(a1, \dots, a_n)$

- ❖ IR describes the instruction set of an abstract machine



N-Tuple (5)

- Jenis operand:
 - variabel program
 - konstanta
 - variabel temporer
- Temporary Variabel digunakan untuk menyimpan informasi nilai perantara.
- Contoh:
A:= D*C+B/E
*, D, C , T1
/, B, E, T2
+, T1, T2, A



Beberapa Perintah Kompleks

```
IF <boolean expr> THEN <blok1> ELSE <blok2>
```

```
T ← <boolean expr>
```

```
cjump T=false goto label2
```

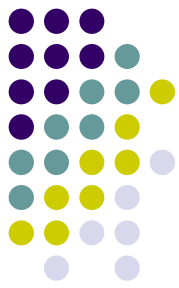
```
label1: kode blok1
```

```
goto label3
```

```
label2: kode untuk blok2
```

```
label3:
```

Perintah Kompleks



```
WHILE (<boolean expr>) DO  
    <blok1>
```

```
label1: T ← kondisi boolean expr  
        ejump T=false goto label2  
        kode untuk blok1  
        goto label1  
label2:
```



Perintah Kompleks

```
REPEAT
```

```
    <blok1>
```

```
UNTIL <bool expr>
```

```
label1 : blok1
```

```
    T ← kondisi boolean expr
```

```
    cjump T = false goto label1
```

Perintah Kompleks



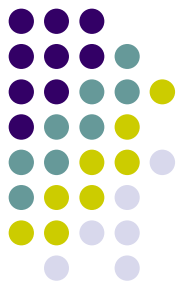
```
FOR x=1 TO n DO <blok1>
```

Translasi pertama:

```
x ← 1  
WHILE (x < 10) DO  
    <blok1>  
    x ← x + 1
```

Translasi kedua:

```
x ← 1  
label1: T ← kondisi boolean expr  
        cjump T=false goto label2  
        kode untuk blok1  
        x ← x + 1  
        goto label1  
label2:
```



Perintah Kompleks

```
SWITCH (a) {  
    CASE 1 : <blok1>  
    CASE 2 : <blok2>  
    CASE 3 : <blok3>  
    ...  
}
```

Translasi pertama:

```
IF (a == 1)  
    <blok1>  
IF (a == 2)  
    <blok2>  
...
```

Translasi kedua:

```
T ← <boolean expr>  
cjump T=false goto label2  
label1: kode blok1  
    goto label3  
label2: kode untuk blok2  
label3:  
...
```

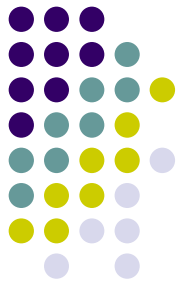
(Diulang sebanyak n kali)



Linking dan Loading

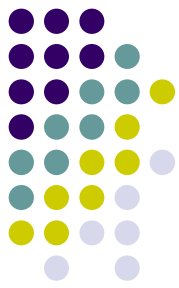
- Pada kenyataannya proses kompilasi biasanya akan menghasilkan obyek file (*.o atau *.obj) kemudian file-file obyek tersebut akan digabung dengan library yang dibutuhkan menjadi **EXE**
- Proses penggabungan antara file obyek dengan file library disebut **Linking**.
- **Library** adalah file-file yang dibutuhkan oleh file obyek untuk melengkapi EXE yang akan dihasilkan agar benar-benar menjadi EXE yang berdiri sendiri.

Library



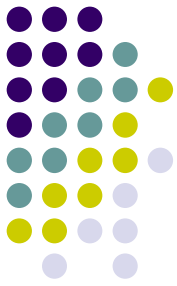
- Jenis Library:
 - Static Library : *.a
 - Dynamic Library : *.dll, *.so

Linking



- Jenis Linking:
 - Static Linking : semua library yang diperlukan dicopykan / digabungkan ke EXE
 - Dynamic Linking: library tidak dikopikan ke EXE, tapi di dalam EXE terdapat pointer informasi mengenai file DLL yang dibutuhkan. Dan jika yang dibutuhkan saja yang dipakai/dipanggil.
- Kerugian:
 - Static Linking: exe besar, tidak mungkin mengubah library dan object file
 - Dynamic Linking: lambat dalam eksekusi program, kalau DLL hilang, program tidak jalan

NEXT



- Memory Allocations dan Runtime Environments