BAB V
ANALISA SEMANTIK

TUJUAN PRAKTIKUM

1) Memahami dan mengerti tugas analisa semantik.
2) Memahami dan mengerti notasi posfix dan kode antara.
3) Memahami dan mengerti pembangkit kode.

TEORI PENUNJANG

5.1 Analisa Semantik

Analisa semantik memanfaatkan pohon sintaks yang dihasilkan pada proses parsing (analisa sintaks). Fungsi dari analisa semantik adalah untuk menentukan makna dari serangkaian instruksi yang terdapat dalam program sumber. Untuk mengetahui makna, maka rutin analisa semantik akan memeriksa :

- Apakah variabel yang ada telah didefinisikan sebelumnya,
- Apakah variabel – variabel tersebut tipenya sama,
- Apakah operan yang akan dioperasikan tersebut ada nilainya dan seterusnya.

Untuk dapat menjalankan fungsi tersebut dengan baik, *semantic analyze*r seringkali menggunakan tabel simbol. Pemeriksaan bisa dilakukan pada tabel *identifier*, tabel *display* dan tabel blok, misal pada *field link.*

Pengecekan yang dilakukan oleh analisis semantik adalah :

- Memeriksa keberlakuan nama – nama meliputi pemeriksaan :
  - Duplikasi

    Pengecekan apakah sebuah nama terjadi pendefinisian lebih dari dua kali. Pengecekan dilakukan pada bagian pengelola blok.

- ♦ Terdefinisi

  Pengecekan apakah sebuah nama yang dipakai pada tubuh program sudah terdefinisi atau belum. Pengecekan dilakukan pada semua tempat kecuali blok.

- Memeriksa tipe

  Melakukan pemeriksaan terhadap kesesuaian tipe dalam statement – statement yang ada. Misal : Bila ada operasi antara dua operan, maka tipe operan pertama harus bisa dioperasikan dengan operan kedua.

5.2 Kode Antara.

Kode antara/*Intermediate Code* merupakan hasil dari tahapan analisis, yang dibuat oleh kompilator pada saat mentranslasikan program dari bahasa tingkat tinggi. Kegunaan dari Kode Antara / *intermediate code* :

- Untuk memperkecil usaha dalam membangun kompilator dari sejumlah bahasa ke sejumlah mesin
- Proses optimasi lebih mudah. (dibandingkan pada program sumber atau kode *assembly* dan kode mesin)
- Bisa melihat program internal yang gampang dimengerti.

5.3 Notasi Postfix.

Pada notasi postfix operator diletakkan paling akhir, maka disebut juga dengan notasi *Sufix* atau Reverse Polish. Sintaks notasi postfix :

<operand> <operand> <operator>

Contoh ekspresi:

(a+b)*(c+d)

Dinyatakan dengan notasi postfix :

ab+cd+*

Kontrol program yang ada dapat diubah ke dalam notasi postfix. Misal :

IF <exp> THEN <stmt1> ELSE <stmt2>

Diubah ke dalam postfix :

<exp> <label1> BZ <stmt1> <label2> BR <stmt2>

↑ ↑

label1 label2

Keterangan :

BZ : branch if zero (zero = salah) {bercabang/meloncat jika kondisi yang dites salah}

BR : branch {bercabang/meloncat tanpa ada kondisi yang dites}

Arti dari notasi postfix diatas adalah :

" *Jika kondisi ekspresi salah, maka instruksi akan meloncat ke label1 dan menjalankan statement2. Bila kondisi ekspresi benar, maka statement1 akan dijalankan lalu meloncat ke label2. Label1 dan label2 sendiri menunjukkan posisi tujuan loncatan, untuk label1 posisinya tepat sebelum statement2, dan label2 adalah statement2*."

5.4 Notasi N–*Tuple*.

Bila pada postfix setiap baris instruksi hanya terdiri dari satu *tuple*, pada notasi N–*tuple* setiap baris bisa terdiri dari beberapa *tuple*. Format umum notasi N-*tuple* adalah :

operator.....N-1 operan

Notasi N-*Tuple* yang biasa digunakan adalah notasi 3 *tupel* dan 4 *tupel*.

5.4.1 Triples Notation

Notasi ini memiliki format sebagai berikut :

<operator> <operan> <operan>

Contoh instruksi :

A := D * C + B / E

Kode antara tripel :

1. *, D, C
2. /, B, E
3. +, (1), (2)
4. :=, A, (3)

Kekurangan dari notasi *tripel* adalah sulit pada saat melakukan optimasi, maka dikembangkan *Indirect Triples* yang memiliki dua list, yaitu list instruksi dan list eksekusi. List instruksi berisi notasi tripel, sedang list eksekusi mengatur urutan eksekusinya.

5.4.2 Quadruples Notation

Format notasi quadruples :

<operator> <operan> <operan> <hasil>

Hasil adalah *temporary variabel* yang bisa ditempatkan pada memory atau register. Masalah yang ada bagaimana mengelola *temporary variabel* (hasil) seminimal mungkin.

Contoh instruksi :

A := D * C + B / E

Dibuat dalam kode antara :

1. *, D, C, T1
2. /, B, E, T2
3. +, T1, T2, A

5.5 Pembangkitan Kode.

Hasil dari tahapan analisis akan diterima oleh bagian pembangkit kode (*code generation*). Di sini kode antara dari program biasanya ditranslasikan ke bahasa *assembly* atau bahasa mesin.

Contoh :

(A+B) * (C+D)

Kode antara dalam bentuk quadruples :

1. +, A, B, T1
2. +, C, D, T2
3. *, T1, T2, T3

Dapat ditranslasikan ke dalam bahasa *assembly* dengan akumulator tunggal :

```
LDA     A       { muat isi A ke akumulator }
ADD     B       { tambahkan isi akumulator dengan B }
```

```
STO     T1      { simpan isi akumulator ke T1 }
LDA     C
ADD     D
STO     T2
LDA     T1
MUL     T2
STO     T3
```

LAPORAN PENDAHULUAN

1. Sebutkan kegunaan kode antara ?
2. Buatlah notasi postfix dari ekspresi (a+b)*(c+d)

LAPORAN AKHIR

1. Jelaskan tugas analisa semantik !
2. Ubah statement berikut ke menjadi kode antara dalam bentuk notasi postfix dan triple !

   Instruksi CASE

   ```
   CASE A OF
        1 :  B := 10;
        2 :  B := 20;
        3 :  B := 30;
   END;
   ```