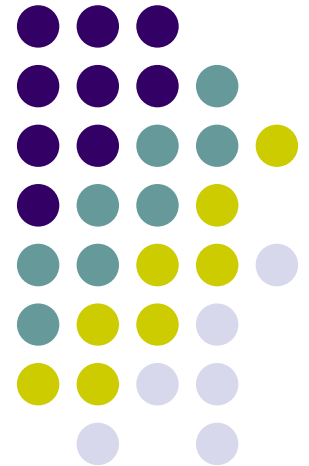


Teknik Kompiler 8

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



Bentuk Normal Greibach disebut juga LL(1)



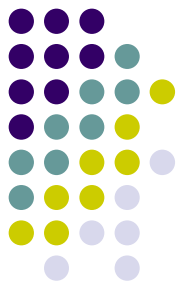
- Syarat GNF adalah:
 - Hasil produksinya selalu diawali oleh satu simbol terminal dan selanjutnya bisa diikuti oleh rangkaian simbol variabel ($A \Rightarrow aV$)
 - Untuk setiap terminal yang memulai GNF harus **berbeda-beda**
 - Tidak memiliki produksi ϵ
 - Tidak bersifat rekursif kiri
 - Sudah dalam bentuk CNF
- *Contoh:*
 $S \Rightarrow aS \mid bT$
 $T \Rightarrow aS \mid b$

LL1 (2)



- Cara pengubahanan TBBK yang belum GNF
S \Rightarrow CA
A \Rightarrow a | d
B \Rightarrow b
C \Rightarrow DD
D \Rightarrow AB
- Tentukan urutan simbol variabel, misalnya S < A < B < C < D. Pengurutan ini terserah, yang penting memudahkan proses berikutnya
- Periksa aturan produksi simbol pertama yang berupa simbol variabel, apakah sudah sesuai dengan aturan urutan simbol variabel yang dibuat diatas.
S \Rightarrow CA (sudah karena S < C)
C \Rightarrow DD (sudah karena C < D)
D \Rightarrow AB (belum karena D > A)
- Maka lakukan substitusi pada simbol variabel A sehingga:
D \Rightarrow aB | dB

LL1 (3)



- Setelah semua memenuhi aturan urutan variabel, maka kita lakukan substitusi mundur pada TBBK yang belum GNF:
C \Rightarrow DD menjadi C \Rightarrow aBD | dBD
S \Rightarrow CA menjadi S \Rightarrow aBDA | dBDA
- Substitusi mundur dilakukan mulai dari aturan produksi yang memiliki urutan variabel yang paling akhir ($S < A < B < C < D$) sehingga C disubstitusi lebih dahulu daripada S
- Hasil akhir GNF:
S \Rightarrow aBDA | dBDA
A \Rightarrow a | d
B \Rightarrow b
C \Rightarrow aBD | dBD
D \Rightarrow aB | dB

PDA (Push Down Automata)



- Adalah mesin otomata dari TBBK yang diimplementasikan dengan stack sehingga hanya terdapat operasi “push” dan “pop”
- Stack (tumpukan) adalah suatu struktur data yang menggunakan prinsip LIFO (*Last In First Out*).
- Sebuah stack selalu memiliki top of stack dan elemen-elemen stack itu yang akan masuk ke dalam stack dengan method “push” dan akan keluar dari stack dengan method “pop”.



PDA (2)

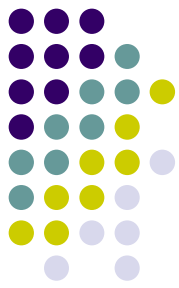
- Sebuah PDA dinyatakan dengan :
 - Q = himpunan state
 - Σ = himpunan simbol input
 - T = simbol stack
 - Δ = fungsi transisi
 - S = state awal
 - F = state akhir
 - Z = top of stack

PDA (3)



- PDA memiliki 2 jenis transisi, yaitu Δ yang menerima simbol input, simbol top of stack, dan state.
 - Setiap pilihan terdiri dari state berikutnya dan simbol-simbol.
 - Penggantian isi stack dilakukan dengan operasi push dan pop.
- Jenis transisi yang kedua adalah transisi ε .
 - Transisi ε tidak melakukan pembacaan input namun hanya menerima simbol top of stack dan state.
 - Transisi ini memungkinkan PDA untuk memanipulasi isi stack dan berpindah antar state tanpa membaca input.

PDA (4)



- *Contoh:*
 - $TBBK = D \Rightarrow aDa \mid bDb \mid c$
 - $Q = \{q_1, q_2, q_3\}$
 - $\Sigma = \{a, b, c\}$
 - $T = \{D, A, C, Z\}$
 - $S = q_1$
 - $F = \{q_3\}$
- Memiliki fungsi transisi sebagai berikut:
 - $\Delta(q_1, \varepsilon, Z) = \{(q_2, DZ)\}$
 - $\Delta(q_2, \varepsilon, D) = \{(q_2, aDa), (q_2, bDb), (q_2, c)\}$
 - $\Delta(q_2, a, a) = \{(q_2, \varepsilon)\}$
 - $\Delta(q_2, b, b) = \{(q_2, \varepsilon)\}$
 - $\Delta(q_2, c, c) = \{(q_2, \varepsilon)\}$
 - $\Delta(q_2, \varepsilon, Z) = \{(q_3, Z)\}$

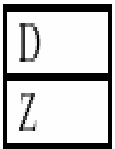


PDA (5)

- Misalkan inputan string: “aca”.
- Kita tahu hal itu dihasilkan oleh
 - $D \Rightarrow aDa \Rightarrow aca$
- Maka PDA yang bisa kita buat:

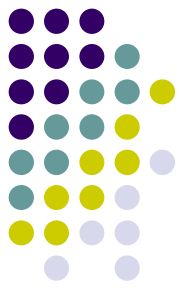


- Kondisi awal : state q_1 , top of stack Z



- Tanpa menerima input (ϵ)
- Fungsi transisinya : $\Delta(q_1, \epsilon, Z) = \{(q_2, DZ)\}$
- Stack menjadi : state q_2 dan D di push

PDA (6)



a
D
a
Z

- Tanpa menerima input (ϵ)
- Fungsi transisinya : $\Delta(q_2, \epsilon, D) = \{(q_2, aDa)\}$
- Stack menjadi : state q_2 , pop top of stack, push 'aDa'

PDA (7)

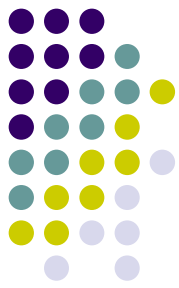


D
a
Z

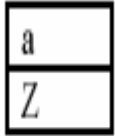
- Menerima input 'a'
- Fungsi transisinya : $\Delta(q_2, a, a) = \{(q_2, \varepsilon)\}$
- Stack menjadi : state q_2 , pop top of stack

c
a
Z

- Tanpa menerima input (ε)
- Fungsi transisinya : $\Delta(q_2, \varepsilon, D) = \{(q_2, c)\}$
- Stack menjadi : state q_2 , pop top of stack, push 'c'



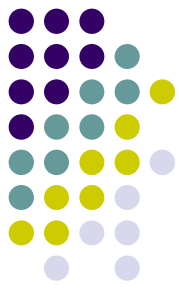
PDA (8)



- Menerima input 'c'
- Fungsi transisinya : $\Delta(q_2, c, c) = \{(q_2, \epsilon)\}$
- Stack menjadi : state q_2 , pop top of stack

- Menerima input 'a'
- Fungsi transisinya : $\Delta(q_2, a, a) = \{(q_2, \epsilon)\}$
- Stack menjadi : state q_2 , pop top of stack





PDA (9)

Z

- Tanpa menerima input (ε)
- Fungsi transisinya : $\Delta(q_2, \varepsilon, Z) = \{(q_3, Z)\}$
- Stack menjadi : state q_3
- Tidak ada transisi lagi dari q_3 , dan karena q_3 adalah state akhir dan semua input sudah dibaca semua, maka 'aca' diterima oleh PDA tersebut

Pseudocode TBBK dengan stack



```
while(masih ada input){
    if(top_of_stack adalah terminal){
        if(input == karakter_dalam_top_of_stack){
            pop top of stack;
            baca karakter input selanjutnya
        } else ERROR;
    }
    else
    {
        Cari produks yang cocok dengan karakter inputan;
        if(ada produksi) ganti stack dengan
produksi ruas kanan;
        else ERROR;
    }
}
if(stack tidak kosong) ERROR;
```



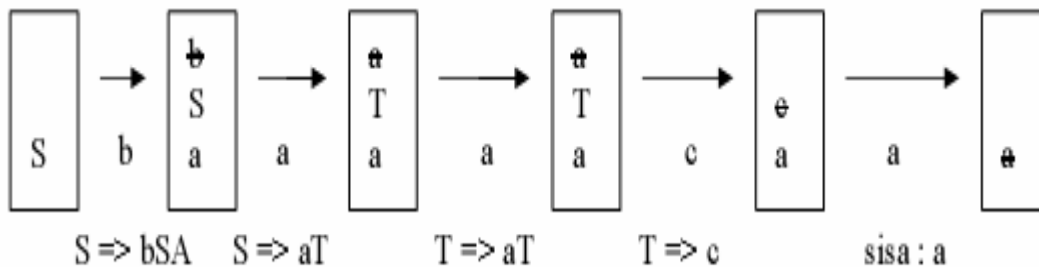
Contoh

- $S \Rightarrow aT \mid bSa$
- $T \Rightarrow aT \mid bS \mid c$

Array 2D :

	A	B	c
S	aT	bSa	ERR
T	aT	bS	C

Inputan : "baaca"

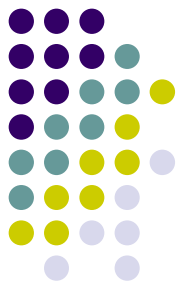


Recursive Descent Parsing



- Melakukan penelusuran grammar di ruas kanan sesuai dengan masukkannya, sampai ditemukan token yang cocok.
- Jika ditemukan non terminal maka akan dipanggil fungsi lain yang bersesuaian dengan non terminal tersebut.
- Setiap simbol non terminal memiliki fungsi sendiri-sendiri
- Contoh:
 $S \Rightarrow bA|c$
 $A \Rightarrow dSd|e$
Bagaimana source code nya?

Source Code



```
function S:boolean;
begin
  S:=true;
  if token_is( 'b' ) then
    if A then
      writeln( 'S->bA' );
    else
      S:=false
    end
  else
    if token_is( 'c' ) then
      writeln( 'S->c' )
    else
      begin
        error( 'S' );
        S:=false;
      end
    end
  end ;
end ;
```

```
function A:boolean;
begin
  A:=true;
  if token_is( 'd' ) then
    begin
      if S then
        if token_is( 'd' ) then
          writeln( 'A->dSd' );
        else
          begin
            error( 'A' );
            A:=false;
          end ;
        else
          A:=false;
        end
      end
    else
      if token_is( 'e' ) then
        writeln( 'A->e' );
      else
        begin
          error( 'A' );
          A:=false;
        end ;
      end
    end ;
end ;
```



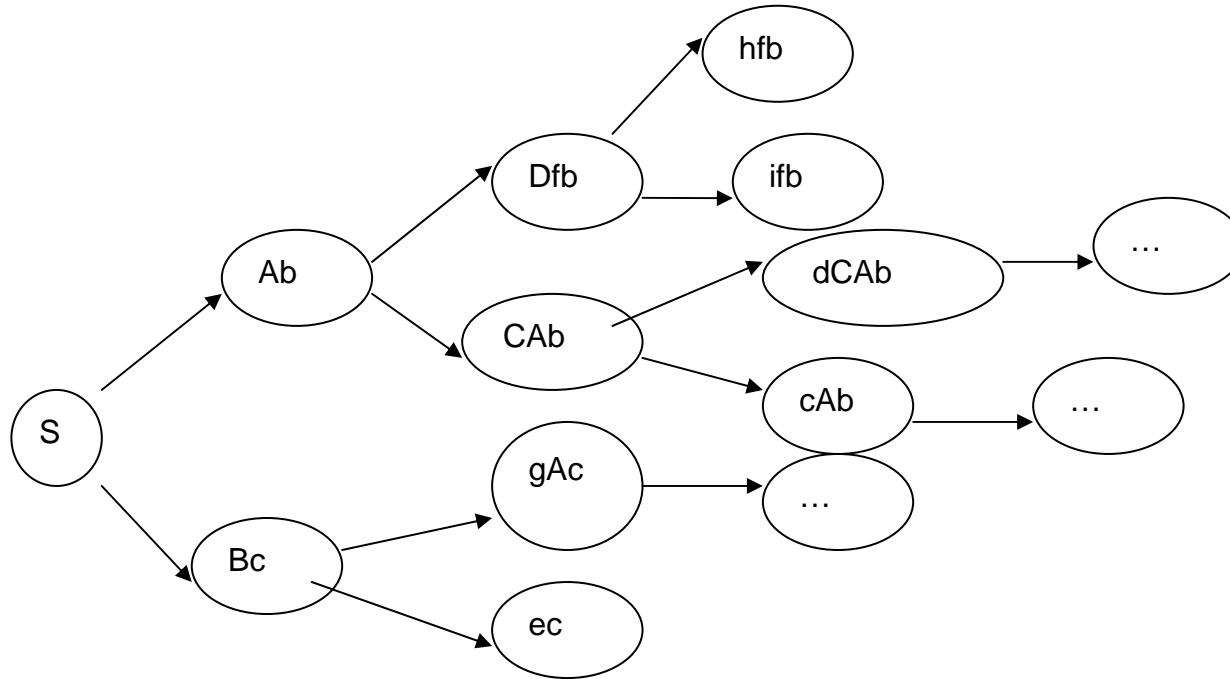
Predictive Parsing

- Untuk TBBK yang rekursif kiri
- Dengan melakukan prediksi awal, membentuk semua kemungkinan parse tree nya
- Contoh:
- $S \Rightarrow Ab \mid Bc$
- $A \Rightarrow Df \mid CA$
- $B \Rightarrow gA \mid e$
- $C \Rightarrow dC \mid c$
- $D \Rightarrow h \mid l$

Dengan recursif descent “gchfc”:

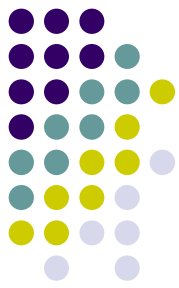
$A \Rightarrow Bc \Rightarrow gAc \Rightarrow gCAc \Rightarrow gcAc \Rightarrow gcDfc \Rightarrow gchfc$

Predictive Parsing

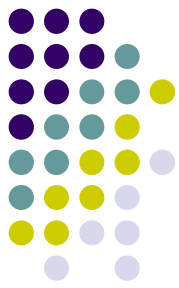


- Bila token dimulai dengan h atau i atau c atau d, maka fungsi $S \Rightarrow Ab$ yang dipilih
- Bila token dimulai dengan g atau e, maka fungsi $S \Rightarrow Bc$, selain itu error!

Semantic Analysis

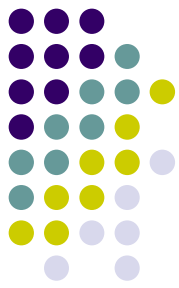


- Analisis semantik menganalisis kebenaran source program.
- Analisis semantik akan memanfaatkan pohon sintaks yang dihasilkan oleh proses parsing.
- Bagian ini berfungsi menentukan makna dari serangkaian instruksi dari source code.
- Tujuan: menentukan makna dari serangkaian instruksi yang terdapat pada source code.



Semantic Analysis (2)

- Yang dilakukan oleh analisis semantik:
 - Type Checking
 - Dilakukan pengecekan tipe ekspresi dan variabel.
 - Static Checking: pengecekan dilakukan oleh kompiler
Contoh: pengecekan operator dan operand sesuai dengan tipe, flow of control check, uniqueness check (apakah ada duplikasi), name-related check (apakah sudah terdefinisi)
 - Dynamic Checking: pengecekan dilakukan oleh target program.
 - Type Conversion
 - Implicit, dilakukan oleh kompiler
 - Explicit, dilakukan oleh programmer



Semantic Analysis (3)

- Contoh:
- $A := (A+B) * (C+D)$
- Pada proses parsing, parser akan menjumpai ekspresi-ekspresi diatas seperti atas, seperti simbol ':=', '+', dan '*'. Namun parser tidak tahu makna yang tersimpan di dalam simbol-simbol tersebut.
- Oleh karena itu Analisis Semantik akan melakukan:
 - Apakah variabel yang ada telah didefinisikan sebelumnya.
 - Apakah variabel tersebut tipenya sama dan benar.
 - Apakah operan yang akan dioperasikan ada nilainya.
 - Menentukan derajat operator
- Untuk dapat menjalankan aksinya, analisis semantik akan membutuhkan tabel simbol

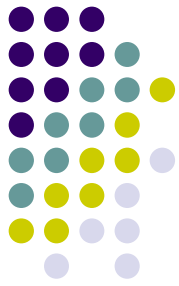


Tabel Simbol

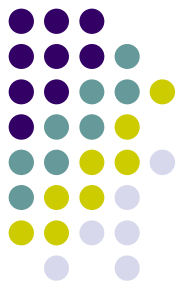
Tabel Simbol berfungsi untuk:

- Menyimpan informasi tentang:
 - Nama variabel dan tipe datanya
 - Informasi detail untuk record dan array
 - Nama prosedur dan fungsi yang ada
 - Jumlah, nama, tipe data dan paramter fungsi/prosedur
 - Nama label
 - Konstanta dan String
- Membantu pemeriksaan kebenaran semantik dari source code
- Membantu mempermudah dalam pembuatan intermediate code dan code generation

Operasi Tabel Simbol



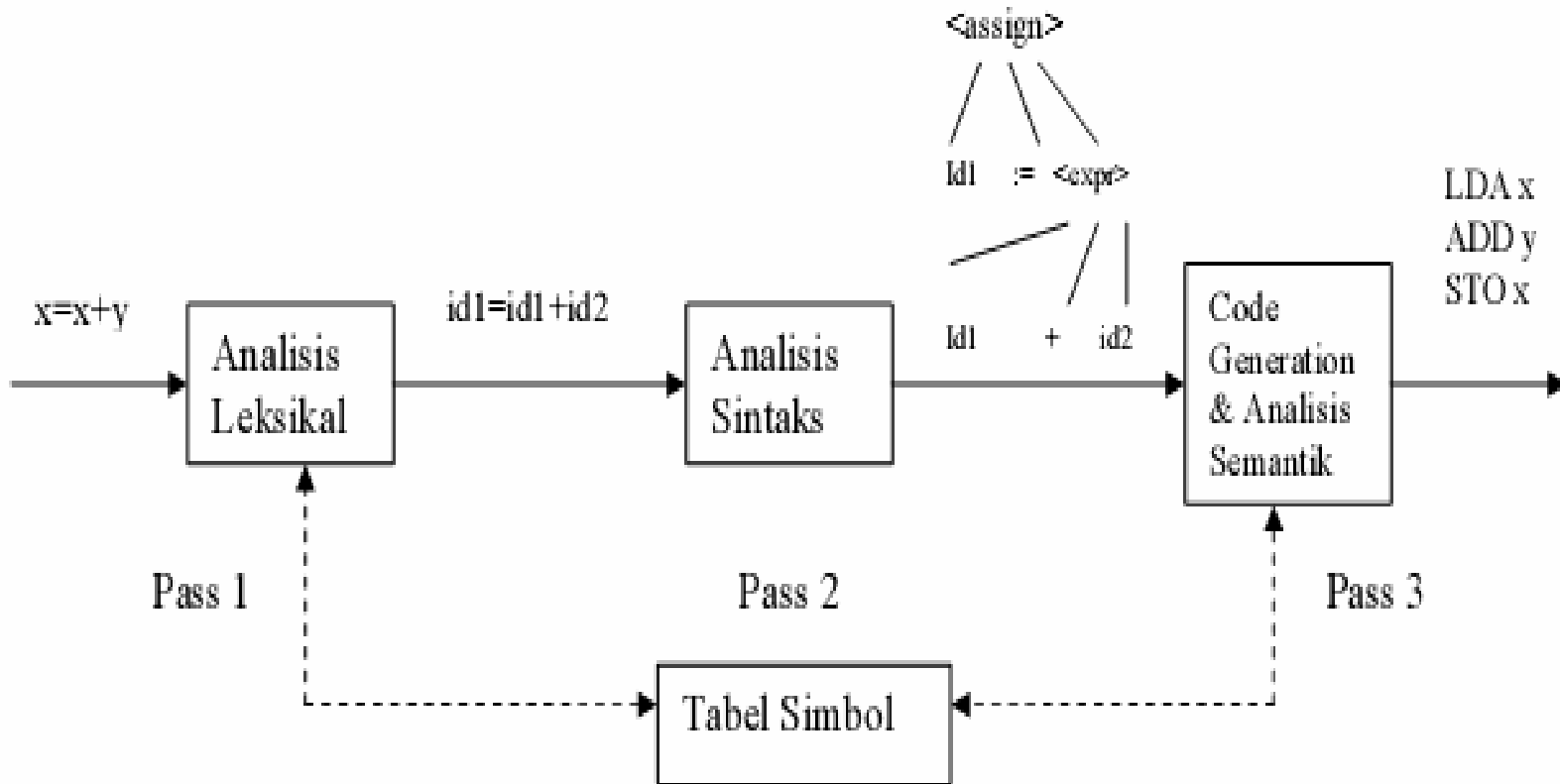
- Jenis operasi yang dilakukan dalam tabel simbol adalah
 - Operasi insert (append/add)
 - Operasi search (dengan hashing)
 - Operasi delete



Tabel Simbol (2)

- Biasanya tabel simbol dibuat pada tahap analisis lexical dan masing-masing data di dalam tabel simbol diberi indeks tertentu yang bersifat unik.
- Oleh analisis sintaks, tabel simbol digunakan untuk memeriksa kebenaran sintaks dan membangkitkan pohon sintaks untuk proses parsing.
- Hasilnya akan dianalisa kebenaran semantiksnya dan digunakan pada tahapan code generation untuk menghasilkan sekumpulan instruksi object code.

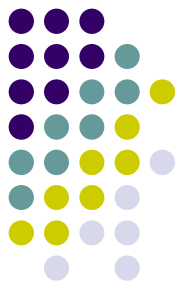
Tabel Simbol (3)





Tabel Simbol (4)

- Pada dasarnya tabel simbol berisi daftar dan informasi indentifier pokok yang terdapat pada source code.
- Tabel ini disebut sebagai tabel pokok.
- Dari tabel pokok ini kemungkinan besar dapat terjadi tidak semua informasi tercover semuanya. Jadi diperlukan tabel lagi yang berfungsi sebagai tabel pembantu.
- Di dalam tabel utama harus terdapat field yang menjembatani indentifier dari tabel utama ke tabel lain yang bersesuaian (analogikan dengan konsep basis data atau senarai pointer)



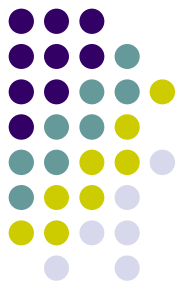
Elemen Tabel Simbol

- Pada umumnya elemen-elemen tabel simbol:
 - No urut identifier (ID unik / auto increment)
 - Nama identifier: berisi nama-nama variabel, prosedur, fungsi, dan lain-lain yang akan digunakan untuk referensi pada analisis semantik, intermediate code, dan code generation.
 - Tipe identifier: berisi keterangan tipe identifier.
 - Object Time Address: berisi address yang mengacu pada alamat tertentu di memori
 - Dimensi (ukuran) dari identifier yang bersangkutan
 - Nomor baris variabel yang dideklarasikan
 - Field link (opsional)

Jenis Tabel Simbol



- Beberapa jenis Tabel Simbol:
 - Tabel identifier: berisi daftar identifier
 - Tabel array: berisi informasi tambahan untuk array
 - Tabel blok: berisi variabel-variabel dalam lingkup blok yang sama (lokal)
 - Tabel real: berisi elemen tabel bernilai real
 - Tabel string: berisi informasi string
 - Tabel display: berisi blok yang aktif
 - Tabel integer: berisi informasi elemen bernilai integer



Tabel Simbol Identifier

- No urut identifier
- Nama identifier
- Jenis identifier : prosedur, fungsi, tipe, variabel, konstanta
- Tipe identifier: integer, real, char, boolean, string, record
- Level : berupa kedalaman identifier (blok program). Misal main program = level 0, prosedur dan fungsi dalam main program = level 1. Field ini digunakan pada saat runtime untuk mengetahui current activation record yang bisa diakses.
- Pada identifier, perlu dicatat juga:
 - Alamat dari identifier
 - Informasi acuan identifier ke tabel identifier lain yang menerangkannya
 - Link: menghubungkan identifier ke identifier lainnya, atau yang dideklarasikan pada level yang sama
 - Normal: digunakan pada pemanggilan parameter by value dan by reference (berupa variabel boolean)

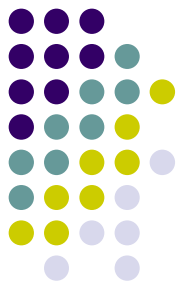


Contoh Tabel Identifier

- Contoh:
Program A;
var B : integer;
 Procedure X(Z:char);
 var C : integer;
 begin

- Pada tabel identifier akan muncul:
- 0 A
- 1 B
- 2 X
- 3 Z
- 4 C

Contoh (2)



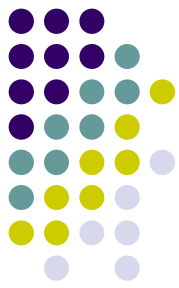
- Contoh implementasi tabel identifier:

```
TabId : array [0..tabmax] of
  Record
    Name      : string;
    Link      : integer;
    Obj       : objek;
    Tipe      : types;
    Ref       : integer;
    Normal    : Boolean;
    Level     : 0..maxlevel;
    Address   : integer;
```

```
End;
```

- Dimana :

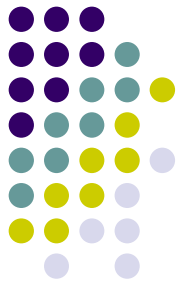
```
Objek = { konstant, variabel, prosedur, fungsi }
Types = { notipe, int, reals, booleans, chars, arrays,
records }
```



Tabel Array

- No urut array dalam tabel
- Tipe dari indeks array yang bersangkutan
- Tipe elemen array
- Alamat Referensi dari elemen array
- Indeks batas atas dan bawah array
- Jumlah elemen array
- Ukuran total array = $(\text{atas} - \text{bawah} + 1) * \text{elemen size}$

Contoh Tabel Array



- Contoh implementasi:

```
TabArray: array [1..tabmax] of  
Record
```

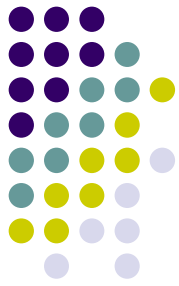
```
    Indextype, elementtype: types;  
    Elemenref, low, high,  
    tabsize: integer;  
End;
```



Tabel Blok

- No urut blok
- Batas awal blok
- Batas akhir blok
- Ukuran parameter
- Ukuran variabel
- Last variabel
- Last parameter

Contoh Tabel Blok

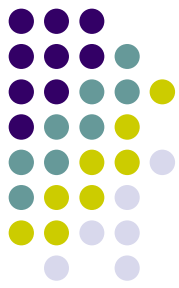


```
TabBlok: array[1..tabmax] of
    Record
        Lastvar, lastpar, parsize,
varsize:integer;
End;
```

Contoh Tabel Blok (2)

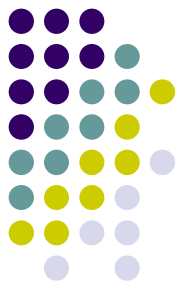
```
Program A;  
var B : integer;  
  Procedure X(Z:char);  
  var C : integer;  
  begin
```

- Dengan contoh program di atas maka untuk program A:
 - Last variabel: 2 (lihat dari tabel identifier, last variable adalah X = 2)
 - Variabel size: 2 (integer = 2 byte)
 - Last parameter: 0 (tanpa parameter)
 - Parameter size: 0
 - Untuk procedure X:
 - Last variabel: 4 (lihat dari tabel identifier, last variable adalah C = 4)
 - Variabel size: 2 (integer = 2 byte)
 - Last parameter: 3 (Z = 3)
 - Parameter size: 1 (char = 1 byte)



Tabel Simbol lain

- **Tabel Real dan Tabel String:**
 - No urut
 - Untuk real: nilai real sedangkan untuk string: karakter-karakter yang ada dalam string
- **Tabel Display:**
 - Berfungsi mencatat blok yang sedang aktif
 - No urut
 - Blok yang sedang aktif
 - Pengisiannya menggunakan konsep stack



Urutan Pemrosesan

- Urutan pengaksesan: Tabel Display – Tabel Blok – Tabel Simbol
- Pertama, tabel display akan mengetahui mana bagian yang aktif, maka akan diketahui identifier-identier yang aktif dalam blok tersebut.
- Informasi identifier yang ada mungkin belum lengkap sehingga diperlukan melihat referensi ke tabel-tabel pelengkap lainnya.



Implementasi Tabel Simbol

- Jelas tidak menggunakan database, Tapi menggunakan:
 - Linked List
 - Tree
 - Hash table

Hash



- Contoh fungsi hash:
 - $\text{maxtabel} = 9$
 - $h(\text{string}) = \sum (\text{ASCII}(C_i)) \bmod (\text{maxtabel}+1)$
 - $h(\text{"ABC"}) = 65+66+67 = 198 \bmod 10 = 8$
 - $h(\text{"AA"}) = 65+65 = 130 \bmod 10 = 0$
 - $h(\text{"BAC"}) = 66+65+67 = 198 \bmod 10 = 8$ terjadi collision
- Maka :
 - 0 AA
 - 1
 - 2
 - 3
 - 4
 - 5
 - 6
 - 7
 - 8 ABC -> BAC
 - 9

NEXT

