BAB II

LANDASAN TEORI

2.1 Aksara Lampung

Bila berbicara mengenai aksara Lampung, maka tidak akan terlepas dari tata bahasa yang dimiliki oleh masyarakat Lampung pada umumnya. Tata bahasa ini sangat mempengaruhi penulisan dalam aksara Lampung. Seperti diketahui bahwa dalam masyarakat Lampung terdapat dua ragam dialek bahasa, yaitu dialek Nyo (O) dan dialek Api (A). Berikut ini akan dijelaskan mengenai kebudayaan masyarakat Lampung dalam hal tata bahasa dan tata penulisan aksara Lampung.

2.1.1 Orang Lampung

Orang Lampung asli adalah semua orang yang asal-usul keturunannya dari sekala berak Lampung Barat, yang berbahasa dan beradat budaya Lampung. Dikarenakan bahasa yang dipakai penduduk asli Lampung dapat dibedakan dalam dua dialek bahsa yaitu dialek Nyo (O) dan dialek Api (A), maka Lampung dapat disebut ”Ruwa Jurai” yang artinya dua kelompok atau dua macam.

Begitu pula dengan adat istiadatnya dapat dibedakan dalam dua golongan adat yang terdiri dari adat Pepadun dan adat Saibatin. Pemakai adat ini, untuk adat Pepadun dari semua orang Lampung yang berdialek Nyo bahkan ada beberapa daerah yang berdialek A ikut menggunakan adat Pepadun.

Akan tetapi, untuk adat Saibatin/Sebatin hanya orang Lampung yang berdialek Api (A) lah yang menggunakannya, penggunaan adat istiadat ini

dilaksanakan masyarakat Lampung hingga sekarang. Oleh karena itu, maka lambang daerah propinsi Lampung bertuliskan ”Sang Bumi Ruwa Jurai” yang artinya adalah bumi yang terdiri dari dua macam atau dua kelompok.

Menurut Dr. J. Van Royen orang Belanda itulah yang membagi bahasa Lampung ke dalam dua dialek yaitu sebagai berikut :

a. Dialek Api (A) terdiri dari
    1. Logat Melinting Maringgai (Labuhan Maringgai, Jabung)
    2. Logat Melinting Raja Basa (Kalianda, Bakauheni)
    3. Logat Peminggir Teluk (Bandar Lampung, Teluk Betung)
    4. Logat Peminggir Pemanggilan (Kota Agung, Talang Padang)
    5. Logat Peminggir Pemanggilan Jelma Daya (Kayu Agung, Umpu)
    6. Logat Pubiyan (Natar, Kali Rejo)
b. Dialek Nyo (O) terdiri dari
    1. Logat Abung (Kota Bumi, Gunung Sugih, Sukadana)
    2. Logat Tulang Bawang (Menggala, Tulang Bawang)

Sedangkan logat Pegagan di Mesuji digunakan oleh orang Lampung di Cikoneng, Jawa Barat dimana berlogat seperti logat orang Kalianda.

2.1.2 Induk Aksara Lampung

Induk aksara Lampung yang disebut dengan kelabai surat terdiri dari 20 induk huruf. Tabel berikut ini berisi 20 induk aksara Lampung.

Tabel 2.1 Induk Aksara Lampung

 ka	 ga	 nga	 pa	 ba
 ma	 ta	 da	 na	 ca
 ja	 nya	 ya	 a	 la
 ra	 sa	 wa	 ha	 ghra

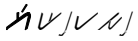
2.1.3 Anak Aksara Lampung

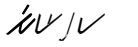
Ditinjau dari namanya, anak aksara Lampung atau anak huruf terdiri dari sembilan nama. Perbandingan penulisan anak huruf Lampung adalah 4:1, dimana 4 untuk induk aksara Lampung dan 1 untuk anak huruf. Dari sembilan nama anak huruf tersebut, mempunyai sebanyak sebelas bunyi anak huruf yang terdiri dari :

1. Bicek ()

   Anak huruf bicek () berfungsi untuk menghasilkan bunyi ’E’ pada induk aksara Lampung. Bicek ditulis di atas induk aksara Lampung dengan perbandingan penulisan 4:1.

   Contoh :

   Sempat ditulis 

   Tempa ditulis 

   Sela ditulis 

2. Ulan (ᨿ)

   Anak huruf ulan (ᨿ) berfungsi untuk menghasilkan bunyi ’i’ pada induk aksara Lampung. Seperti halnya bicek, Ulan pun ditulis di atas induk aksara Lampung dengan perbandingan penulisan 4:1.

   Contoh :

   Sari ditulis 

   Kali ditulis 

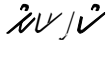
   Pati ditulis 

3. Ulan (ᨾ)

   Anak huruf ulan (ᨾ) berfungsi untuk menghasilkan bunyi ’ė’ pada induk aksara Lampung. Ulan (ᨾ) ditulis di atas induk aksara Lampung dengan perbandingan penulisan 4:1.

   Contoh :

   Peta ditulis 

   Tempe ditulis 

   Bemo ditulis

4. Bitan (ᩩ)

   Anak huruf bitan (ᩩ) berfungsi untuk menghasilkan bunyi ’u’ pada induk aksara Lampung. Bitan (ᩩ) ditulis di bawah induk aksara Lampung dengan perbandingan penulisan 4:1.

   Contoh :

Babu ditulis 

Palu ditulis 

Maju ditulis 

5. Bitan (◌ꦼ)

Anak huruf bitan (◌ꦼ) berfungsi untuk menghasilkan bunyi ’o’ pada induk aksara Lampung. Bitan (◌ꦼ) ditulis di bawah induk aksara Lampung dengan perbandingan penulisan 4:1.

Contoh :

Roda ditulis 

Coba ditulis 

Dona ditulis

6. Datasan (=)

Anak huruf datasan (=) berfungsi untuk menghasilkan tambahan atau akhiran ’n’ pada induk aksara Lampung. Datasan (=) ditulis di atas induk aksara Lampung dengan perbandingan penulisan 4:1.

Contoh :

Batasan ditulis 

Jalan ditulis

Paman ditulis

7. Rejenjung (ᨿ)

   Anak huruf rejenjung (ᨿ) berfungsi untuk memberikan akhiran ’r’ pada induk aksara Lampung. Rejenjung (ᨿ) ditulis di atas induk aksara Lampung dengan perbandingan penulisan 4:1.

   Contoh :

   Bakar ditulis 

   Pasar ditulis 

   Hajar ditulis 

8. Keleniah (ᨾ)

   Anak huruf keleniah (ᨾ) berfungsi untuk menghasilkan tambahan atau akhiran ’h’ pada induk aksara Lampung. Keleniah (ᨾ) ditulis di sebelah kanan atau disamping induk aksara Lampung dengan perbandingan penulisan 4:1.

   Contoh :

   Salah ditulis 

   Parah ditulis 

   Basah ditulis 

9. Tekelungau (ᨻ)

   Anak huruf tekelungau (ᨻ) berfungsi untuk menghasilkan akhiran ’u’ atau ’w’ pada induk aksara Lampung. Tekelungau (ᨻ) ditulis di bawah induk aksara Lampung dengan perbandingan penulisan 4:1.

   Contoh :

   Parau ditulis 

Kalau ditulis 

Mau ditulis 

10. Tekelingai (ꤰ)

Anak huruf tekelingai (ꤰ) berfungsi untuk menghasilkan tambahan atau akhiran ’i’ pada induk aksara Lampung. Tekelingai (ꤰ) ditulis di sebelah kanan atau di samping induk aksara Lampung dengan perbandingan penulisan 4:1.

Contoh :

Rantai ditulis 

Balai ditulis 

Damai ditulis 

11. Tekelubang (–)

Anak huruf tekelubang (–) berfungsi untuk menghasilkan tambahan atau akhiran ’ng’ pada induk aksara Lampung. Tekelubang (–) ditulis di atas induk aksara Lampung dengan perbandingan penulisan 4:1

Contoh :

Barang ditulis 

Pasang ditulis 

Malang ditulis

Dari ke sebelas anak huruf tersebut, ditambah dengan ”nengen” yang gunanya untuk mematikan bunyi ’a’ pada induk huruf. Nengen berbentuk garis

tegak miring namun pada bagian bawahnya diberi tambahan berupa kait untuk membedakan dengan *slash* (garis miring). Bentuk dari ”nengen” adalah (ᨈ)

2.1.4 Bunyi Aksara Lampung

Bunyi aksara Lampung terbentuk dari penggabungan induk aksara Lampung dengan Anak Aksara Lampung sehingga menghasilkan bunyi baru dalam pembacaan aksara Lampung.

1. Bunyi Aksara ᨀ Ka

Tabel 2.2 Bunyi Aksara ᨀ Ka

Ka ᨀ	Ke ᨀᨗ	Kė ᨀᨘ	Ki ᨀᨙ	Ko ᨀᨚ	Ku ᨀᨛ	Kan ᨀ᨞
Kang ᨀ᨟	Kar ᨀᨠ	Kah ᨀᨡ	Kau ᨀᨢ	Kai ᨀᨣ	Ken ᨀᨤ	Keng ᨀᨥ
Ker ᨀᨦ	Keh ᨀᨧ	Keu ᨀᨨ	Kei ᨀᨩ	Kėn ᨀᨪ	Kėng ᨀᨫ	Kėr ᨀᨬ
Kėh ᨀᨭ	Kėu ᨀᨮ	Kėi ᨀᨯ	Kin ᨀᨰ	King ᨀᨱ	Kir ᨀᨲ	Kih ᨀᨳ
Kiu ᨀᨴ	Kii ᨀᨵ	Kon ᨀᨶ	Kong ᨀᨷ	Kor ᨀᨸ	Koh ᨀᨹ	Kou ᨀᨺ
Koi ᨀᨻ	Kun ᨀᨼ	Kung ᨀᨽ	Kur ᨀᨾ	Kuh ᨀᨿ	Kuu ᨀᩀ	Kui ᨀᩁ

2. Bunyi aksara ᨁ Ga

Tabel 2.3 Bunyi Aksara ᨁ Ga

Ga ᨁ	Ge ᨁᨗ	Gė ᨁᨘ	Gi ᨁᨙ	Go ᨁᨚ	Gu ᨁᨛ	Gan ᨁ᨞
Gang ᨁ᨟	Gar ᨁᨠ	Gah ᨁᨡ	Gau ᨁᨢ	Gai ᨁᨣ	Gen ᨁᨤ	Geng ᨁᨥ
Ger ᨁᨦ	Geh ᨁᨧ	Geu ᨁᨨ	Gei ᨁᨩ	Gėn ᨁᨪ	Gėng ᨁᨫ	Gėr ᨁᨬ

Gėh	Gėu	Gėi	Gin	Ging	Gir	Gih
Giu	Gii	Gon	Gong	Gor	Goh	Gou
Goi	Gun	Gung	Gur	Guh	Guu	Gui

3. Bunyi aksara Nga

Tabel 2.4 Bunyi Aksara Nga

Nga	Nge	Ngė	Ngi	Ngo	Ngu	Ngan
Ngang	Ngar	Ngah	Ngau	Ngai	Ngen	Ngeng
Nger	Ngeh	Ngeu	Ngei	Ngėn	Ngėng	Ngėr
Ngėh	Ngėu	Ngėi	Ngin	Nging	Ngir	Ngih
Ngiu	Ngii	Ngon	Ngong	Ngor	Ngoh	Ngou
Ngoi	Ngun	Ngung	Ngur	Nguh	Nguu	Ngui

4. Bunyi aksara Pa

Tabel 2.5 Bunyi Aksara Pa

Pa	Pe	Pė	Pi	Po	Pu	Pan
Pang	Par	Pah	Pau	Pai	Pen	Peng
Per	Peh	Peu	Pei	Pėn	Pėng	Pėr
Pėh	Pėu	Pėi	Pin	Ping	Pir	Pih

Piu	Pii	Pon	Pong	Por	Poh	Pou
Poi	Pun	Pung	Pur	Puh	Puu	Pui

5. Bunyi aksara Ba

Tabel 2.6 Bunyi Aksara Ba

Ba	Be	Bė	Bi	Bo	Bu	Ban
Bang	Bar	Bah	Bau	Bai	Ben	Beng
Ber	Beh	Beu	Bei	Bėn	Bėng	Bėr
Bėh	Bėu	Bėi	Bin	Bing	Bir	Bih
Biu	Bii	Bon	Bong	Bor	Boh	Bou
Boi	Bun	Bung	Bur	Buh	Buu	Bui

6. Bunyi aksara Ma

Tabel 2.7 Bunyi Aksara Ma

Ma	Me	Mė	Mi	Mo	Mu	Man
Mang	Mar	Mah	Mau	Mai	Men	Meng
Mer	Meh	Meu	Mei	Mėn	Mėng	Mėr
Mėh	Mėu	Mėi	Min	Ming	Mir	Mih
Miu	Mii	Mon	Mong	Mor	Moh	Mou
Moi	Mun	Mung	Mur	Muh	Muu	Mui

7. Bunyi aksara ᯕ Ta

Tabel 2.8 Bunyi Aksara ᯕ Ta

Ta ᯕ	Te ᯕᮦ	Tė ᯕᮩ	Ti ᯕᮤ	To ᯕᮥ	Tu ᯕᮨ	Tan ᯕᮀ
Tang ᯕᮁ	Tar ᯕᮂ	Tah ᯕᮃ	Tau ᯕᮄ	Tai ᯕᮅ	Ten ᯕᮆ	Teng ᯕᮇ
Ter ᯕᮈ	Teh ᯕᮉ	Teu ᯕᮊ	Tei ᯕᮋ	Tėn ᯕᮌ	Tėng ᯕᮍ	Tėr ᯕᮎ
Tėh ᯕᮏ	Tėu ᯕᮐ	Tėi ᯕᮑ	Tin ᯕᮒ	Ting ᯕᮓ	Tir ᯕᮔ	Tih ᯕᮕ
Tiu ᯕᮖ	Tii ᯕᮗ	Ton ᯕᮘ	Tong ᯕᮙ	Tor ᯕᮚ	Toh ᯕᮛ	Tou ᯕᮜ
Toi ᯕᮝ	Tun ᯕᮞ	Tung ᯕᮟ	Tur ᯕᮠ	Tuh ᯕᮡ	Tuu ᯕᮢ	Tui ᯕᮣ

8. Bunyi aksara ᯗ Da

Tabel 2.9 Bunyi Aksara ᯗ Da

Da ᯗ	De ᯗᮦ	Dė ᯗᮩ	Di ᯗᮤ	Do ᯗᮥ	Du ᯗᮨ	Dan ᯗᮀ
Dang ᯗᮁ	Dar ᯗᮂ	Dah ᯗᮃ	Dau ᯗᮄ	Dai ᯗᮅ	Den ᯗᮆ	Deng ᯗᮇ
Der ᯗᮈ	Deh ᯗᮉ	Deu ᯗᮊ	Dei ᯗᮋ	Dėn ᯗᮌ	Dėng ᯗᮍ	Dėr ᯗᮎ
Dėh ᯗᮏ	Dėu ᯗᮐ	Dėi ᯗᮑ	Din ᯗᮒ	Ding ᯗᮓ	Dir ᯗᮔ	Dih ᯗᮕ
Diu ᯗᮖ	Dii ᯗᮗ	Don ᯗᮘ	Dong ᯗᮙ	Dor ᯗᮚ	Doh ᯗᮛ	Dou ᯗᮜ
Doi ᯗᮝ	Dun ᯗᮞ	Dung ᯗᮟ	Dur ᯗᮠ	Duh ᯗᮡ	Duu ᯗᮢ	Dui ᯗᮣ

9. Bunyi aksara ᨿ Na

Tabel 2.10 Bunyi Aksara ᨿ Na

Na ᨿ	Ne ᨿᩮ	Nė ᨿᩮᩫ	Ni ᨿᩤ	No ᨿᩯ	Nu ᨿᩩ	Nan ᨿᩣᩴ
Nang ᨿᩴ	Nar ᨿᩣᩬ	Nah ᨿᩉ	Nau ᨿᩮᩣ	Nai ᨿᩱ	Nen ᨿᩮᩣᩴ	Neng ᨿᩮᩴ
Ner ᨿᩮᩬ	Neh ᨿᩮᩉ	Neu ᨿᩮᩩ	Nei ᨿᩮᩱ	Nėn ᨿᩮᩫᩣᩴ	Nėng ᨿᩮᩫᩴ	Nėr ᨿᩮᩫᩬ
Nėh ᨿᩮᩫᩉ	Nėu ᨿᩮᩫᩩ	Nėi ᨿᩮᩫᩱ	Nin ᨿᩤᩣᩴ	Ning ᨿᩤᩴ	Nir ᨿᩤᩬ	Nih ᨿᩤᩉ
Niu ᨿᩤᩩ	Nii ᨿᩤᩱ	Non ᨿᩯᩣᩴ	Nong ᨿᩯᩴ	Nor ᨿᩯᩬ	Noh ᨿᩯᩉ	Nou ᨿᩯᩩ
Noi ᨿᩯᩱ	Nun ᨿᩣᩴ	Nung ᨿᩴ	Nur ᨿᩬ	Nuh ᨿᩉ	Nuu ᨿᩩ	Nui ᨿᩱ

10. Bunyi aksara ᨧ Ca

Tabel 2.11 Bunyi Aksara ᨧ Ca

Ca ᨧ	Ce ᨧᩮ	Cė ᨧᩮᩫ	Ci ᨧᩤ	Co ᨧᩯ	Cu ᨧᩩ	Can ᨧᩣᩴ
Cang ᨧᩴ	Car ᨧᩣᩬ	Cah ᨧᩉ	Cau ᨧᩮᩣ	Cai ᨧᩱ	Cen ᨧᩮᩣᩴ	Ceng ᨧᩮᩴ
Cer ᨧᩮᩬ	Ceh ᨧᩮᩉ	Ceu ᨧᩮᩩ	Cei ᨧᩮᩱ	Cėn ᨧᩮᩫᩣᩴ	Cėng ᨧᩮᩫᩴ	Cėr ᨧᩮᩫᩬ
Cėh ᨧᩮᩫᩉ	Cėu ᨧᩮᩫᩩ	Cėi ᨧᩮᩫᩱ	Cin ᨧᩤᩣᩴ	Cing ᨧᩤᩴ	Cir ᨧᩤᩬ	Cih ᨧᩤᩉ
Ciu ᨧᩤᩩ	Cii ᨧᩤᩱ	Con ᨧᩯᩣᩴ	Cong ᨧᩯᩴ	Cor ᨧᩯᩬ	Coh ᨧᩯᩉ	Cou ᨧᩯᩩ
Coi ᨧᩯᩱ	Cun ᨧᩣᩴ	Cung ᨧᩴ	Cur ᨧᩬ	Cuh ᨧᩉ	Cuu ᨧᩩ	Cui ᨧᩱ

11. Bunyi aksara ᠵ Ja

Tabel 2.12 Bunyi Aksara ᠵ Ja

Ja ᠵ	Je ᠵᡝ	Jė ᠵᡞ	Ji ᠵᡳ	Jo ᠵᡭ	Ju ᠵᡠ	Jan ᠵᡣ
Jang ᠵᡤ	Jar ᠵᡨ	Jah ᠵᡦ	Jau ᠵᡮ	Jai ᠵᡯ	Jen ᠵᡰ	Jeng ᠵᡱ
Jer ᠵᡲ	Jeh ᠵᡴᡝ	Jeu ᠵᡵ	Jei ᠵᡶ	Jėn ᠵᡷ	Jėng ᠵᡸ	Jėr ᠵ᡹
Jėh ᠵ᡺ᡝ	Jėu ᠵ᡻	Jėi ᠵ᡼	Jin ᠵ᡽	Jing ᠵ᡾	Jir ᠵ᡿	Jih ᠵᢀᡝ
Jiu ᠵᢁ	Jii ᠵᢂ	Jon ᠵᢃ	Jong ᠵᢄ	Jor ᠵᢅ	Joh ᠵᢆᡦ	Jou ᠵᢇ
Joi ᠵᢈ	Jun ᠵᢉ	Jung ᠵᢊ	Jur ᠵᢋ	Juh ᠵᢌᡦ	Juu ᠵᢍ	Jui ᠵᢎ

12. Bunyi aksara ᠨᡞ Nya

Tabel 2.13 Bunyi Aksara ᠨᡞ Nya

Nya ᠨᡞ	Nye ᠨᡞᡝ	Nyė ᠨᡞᡞ	Nyi ᠨᡞᡳ	Nyo ᠨᡞᡭ	Nyu ᠨᡞᡠ	Nyan ᠨᡞᡣ
Nyang ᠨᡞᡤ	Nyar ᠨᡞᡨ	Nyah ᠨᡞᡦ	Nyau ᠨᡞᡮ	Nyai ᠨᡞᡯ	Nyen ᠨᡞᡰ	Nyeng ᠨᡞᡱ
Nyer ᠨᡞᡲ	Nyeh ᠨᡞᡴᡝ	Nyeu ᠨᡞᡵ	Nyei ᠨᡞᡶ	Nyėn ᠨᡞᡷ	Nyėng ᠨᡞᡸ	Nyėr ᠨᡞ᡹
Nyėh ᠨᡞ᡺ᡝ	Nyėu ᠨᡞ᡻	Nyėi ᠨᡞ᡼	Nyin ᠨᡞ᡽	Nying ᠨᡞ᡾	Nyir ᠨᡞ᡿	Nyih ᠨᡞᢀᡝ
Nyiu ᠨᡞᢁ	Nyii ᠨᡞᢂ	Nyon ᠨᡞᢃ	Nyong ᠨᡞᢄ	Nyor ᠨᡞᢅ	Nyoh ᠨᡞᢆᡦ	Nyou ᠨᡞᢇ
Nyoi ᠨᡞᢈ	Nyun ᠨᡞᢉ	Nyung ᠨᡞᢊ	Nyur ᠨᡞᢋ	Nyuh ᠨᡞᢌᡦ	Nyuu ᠨᡞᢍ	Nyui ᠨᡞᢎ

13. Bunyi aksara ꤟ Ya

Tabel 2.14 Bunyi Aksara ꤟ Ya

Ya ꤟ	Ye ꤟꤰ	Yė ꤟꤱ	Yi ꤟꤲ	Yo ꤟꤳ	Yu ꤟꤴ	Yan ꤟꤵ
Yang ꤟꤶ	Yar ꤟꤷ	Yah ꤟꤸ	Yau ꤟꤹ	Yai ꤟꤺ	Yen ꤟꤻ	Yeng ꤟꤼ
Yer ꤟꤽ	Yeh ꤟꤾ	Yeu ꤟꤿ	Yei ꤟꥀ	Yėn ꤟꥁ	Yėng ꤟꥂ	Yėr ꤟꥃ
Yėh ꤟꥄ	Yėu ꤟꥅ	Yėi ꤟꥆ	Yin ꤟꥇ	Ying ꤟꥈ	Yir ꤟꥉ	Yih ꤟꥊ
Yiu ꤟꥋ	Yii ꤟꥌ	Yon ꤟꥍ	Yong ꤟꥎ	Yor ꤟꥏ	Yoh ꤟꥐ	You ꤟꥑ
Yoi ꤟꥒ	Yun ꤟ꥓	Yung ꤟ꥔	Yur ꤟ꥕	Yuh ꤟ꥖	Yuu ꤟ꥗	Yui ꤟ꥘

14. Bunyi aksara ꤀ A

Tabel 2.15 Bunyi Aksara ꤀ A

A ꤀	E ꤀ꤰ	Ė ꤀ꤱ	I ꤀ꤲ	O ꤀ꤳ	U ꤀ꤴ	An ꤀ꤵ
Ang ꤀ꤶ	Ar ꤀ꤷ	Ah ꤀ꤸ	Au ꤀ꤹ	Ai ꤀ꤺ	En ꤀ꤻ	Eng ꤀ꤼ
Er ꤀ꤽ	Eh ꤀ꤾ	Eu ꤀ꤿ	Ei ꥀ	Ėn ꥁ	Ėng ꥂ	Ėr ꥃ
Ėh ꥄ	Ėu ꥅ	Ėi ꥆ	In ꥇ	Ing ꥈ	Ir ꥉ	Ih ꥊ
Iu ꥋ	Ii ꥌ	On ꥍ	Ong ꥎ	Or ꥏ	Oh ꥐ	Ou ꥑ
Oi ꥒ	Un ꥓	Ung ꥔	Ur ꥕	Uh ꥖	Uu ꥗	Ui ꥘

15. Bunyi aksara ᯜ La

Tabel 2.16 Bunyi Aksara ᯜ La

La ᯜ	Le ᯜᯩ	Lė ᯜᯩ᯲	Li ᯜ᯲	Lo ᯜ᯦	Lu ᯜᯨ	Lan ᯜᯩ᯳
Lang ᯜᯪ	Lar ᯜᯬ	Lah ᯜᯖ	Lau ᯜᯮ	Lai ᯜᯰ	Len ᯜᯩ᯳	Leng ᯜᯪ
Ler ᯜᯬ	Leh ᯜᯩᯖ	Leu ᯜᯮ	Lei ᯜᯰ	Lėn ᯜᯩ᯳	Lėng ᯜᯪ	Lėr ᯜᯬ
Lėh ᯜᯩᯖ	Lėu ᯜᯮ	Lėi ᯜᯰ	Lin ᯜ᯲᯳	Ling ᯜ᯲ᯪ	Lir ᯜ᯲ᯬ	Lih ᯜ᯲ᯖ
Liu ᯜ᯲ᯮ	Lii ᯜ᯲ᯰ	Lon ᯜ᯦᯳	Long ᯜ᯦ᯪ	Lor ᯜ᯦ᯬ	Loh ᯜ᯦ᯖ	Lou ᯜ᯦ᯮ
Loi ᯜ᯦ᯰ	Lun ᯜᯨ᯳	Lung ᯜᯨ᯲	Lur ᯜᯨᯬ	Luh ᯜᯨᯖ	Luu ᯜᯨᯮ	Lui ᯜᯨᯰ

16. Bunyi aksara ᯞ Ra

Tabel 2.17 Bunyi Aksara ᯞ Ra

Ra ᯞ	Re ᯞᯩ	Rė ᯞᯩ᯲	Ri ᯞ᯲	Ro ᯞ᯦	Ru ᯞᯨ	Ran ᯞᯩ᯳
Rang ᯞᯪ	Rar ᯞᯬ	Rah ᯞᯖ	Rau ᯞᯮ	Rai ᯞᯰ	Ren ᯞᯩ᯳	Reng ᯞᯪ
Rer ᯞᯬ	Reh ᯞᯩᯖ	Reu ᯞᯮ	Rei ᯞᯰ	Rėn ᯞᯩ᯳	Rėng ᯞᯪ	Rėr ᯞᯬ
Rėh ᯞᯩᯖ	Rėu ᯞᯮ	Rėi ᯞᯰ	Rin ᯞ᯲᯳	Ring ᯞ᯲ᯪ	Rir ᯞ᯲ᯬ	Rih ᯞ᯲ᯖ
Riu ᯞ᯲ᯮ	Rii ᯞ᯲ᯰ	Ron ᯞ᯦᯳	Rong ᯞ᯦ᯪ	Ror ᯞ᯦ᯬ	Roh ᯞ᯦ᯖ	Rou ᯞ᯦ᯮ
Roi ᯞ᯦ᯰ	Run ᯞᯨ᯳	Rung ᯞᯨ᯲	Rur ᯞᯨᯬ	Ruh ᯞᯨᯖ	Ruu ᯞᯨᯮ	Rui ᯞᯨᯰ

17. Bunyi aksara ᤔ Sa

Tabel 2.18 Bunyi Aksara ᤔ Sa

Sa ᤔ	Se ᤔᤧ	Sė ᤔᤨ	Si ᤔᤩ	So ᤔᤪ	Su ᤔᤫ	San ᤔ᤬
Sang ᤔ᤭	Sar ᤔ᤮	Sah ᤔ᤯	Sau ᤔᤰ	Sai ᤔᤱ	Sen ᤔᤲ	Seng ᤔᤳ
Ser ᤔᤴ	Seh ᤔᤵ	Seu ᤔᤶ	Sei ᤔᤷ	Sėn ᤔᤸ	Sėng ᤔ᤹	Sėr ᤔ᤺
Sėh ᤔ᤻	Sėu ᤔ᤼	Sėi ᤔ᤽	Sin ᤔ᤾	Sing ᤔ᤿	Sir ᤔ᥀	Sih ᤔ᥁
Siu ᤔ᥂	Sii ᤔ᥃	Son ᤔ᥄	Song ᤔ᥅	Sor ᤔ᥆	Soh ᤔ᥇	Sou ᤔ᥈
Soi ᤔ᥉	Sun ᤔ᥊	Sung ᤔ᥋	Sur ᤔ᥌	Suh ᤔ᥍	Suu ᤔ᥎	Sui ᤔ᥏

18. Bunyi aksara ᤕ Wa

Tabel 2.19 Bunyi Aksara ᤕ Wa

Wa ᤕ	We ᤕᤧ	Wė ᤕᤨ	Wi ᤕᤩ	Wo ᤕᤪ	Wu ᤕᤫ	Wan ᤕ᤬
Wang ᤕ᤭	War ᤕ᤮	Wah ᤕ᤯	Wau ᤕᤰ	Wai ᤕᤱ	Wen ᤕᤲ	Weng ᤕᤳ
Wer ᤕᤴ	Weh ᤕᤵ	Weu ᤕᤶ	Wei ᤕᤷ	Wėn ᤕᤸ	Wėng ᤕ᤹	Wėr ᤕ᤺
Wėh ᤕ᤻	Wėu ᤕ᤼	Wėi ᤕ᤽	Win ᤕ᤾	Wing ᤕ᤿	Wir ᤕ᥀	Wih ᤕ᥁
Wiu ᤕ᥂	Wii ᤕ᥃	Won ᤕ᥄	Wong ᤕ᥅	Wor ᤕ᥆	Woh ᤕ᥇	Wou ᤕ᥈
Woi ᤕ᥉	Wun ᤕ᥊	Wung ᤕ᥋	Wur ᤕ᥌	Wuh ᤕ᥍	Wuu ᤕ᥎	Wui ᤕ᥏

19. Bunyi aksara ᦲ Ha

Tabel 2.20 Bunyi Aksara ᦲ Ha

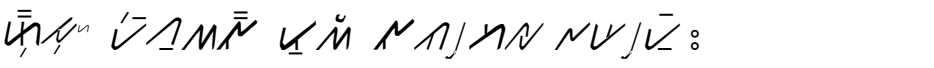
Ha ᦲ	He ᦲᦵ	Hė ᦲᦶ	Hi ᦲᦷ	Ho ᦲᦸ	Hu ᦲᦹ	Han ᦲᦻ
Hang ᦲᦼ	Har ᦲᦽ	Hah ᦲᦾ	Hau ᦲᦿ	Hai ᦲᧀ	Hen ᦲᧁ	Heng ᦲᧂ
Her ᦲᧃ	Heh ᦲᧄ	Heu ᦲᧅ	Hei ᦲᧆ	Hėn ᦲᧇ	Hėng ᦲᧈ	Hėr ᦲᧉ
Hėh ᦲ᧊	Hėu ᦲ᧋	Hėi ᦲ᧌	Hin ᦲ᧍	Hing ᦲ᧎	Hir ᦲ᧏	Hih ᦲ᧐
Hiu ᦲ᧑	Hii ᦲ᧒	Hon ᦲ᧓	Hong ᦲ᧔	Hor ᦲ᧕	Hoh ᦲ᧖	Hou ᦲ᧗
Hoi ᦲ᧘	Hun ᦲ᧙	Hung ᦲ᧚	Hur ᦲ᧛	Huh ᦲ᧜	Huu ᦲ᧝	Hui ᦲ᧞

20. Bunyi aksara ᦴ Gha

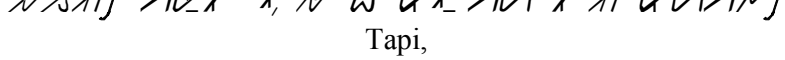
Tabel 2.21 Bunyi Aksara ᦴ Ghra

Ghra ᦴ	Ghre ᦴᦵ	Ghrė ᦴᦶ	Ghri ᦴᦷ	Ghro ᦴᦸ	Ghru ᦴᦹ	Ghran ᦴᦻ
Ghrang ᦴᦼ	Ghrar ᦴᦽ	Ghrah ᦴᦾ	Ghrau ᦴᦿ	Ghrai ᦴᧀ	Ghren ᦴᧁ	Ghreng ᦴᧂ
Ghrer ᦴᧃ	Ghreh ᦴᧄ	Ghreu ᦴᧅ	Ghrei ᦴᧆ	Ghrėn ᦴᧇ	Ghrėng ᦴᧈ	Ghrėr ᦴᧉ
Ghrėh ᦴ᧊	Ghrėu ᦴ᧋	Ghrėi ᦴ᧌	Ghrin ᦴ᧍	Ghring ᦴ᧎	Ghrir ᦴ᧏	Ghrih ᦴ᧐
Ghriu ᦴ᧑	Ghrii ᦴ᧒	Ghron ᦴ᧓	Ghrong ᦴ᧔	Ghror ᦴ᧕	Ghroh ᦴ᧖	Ghrou ᦴ᧗
Ghroi ᦴ᧘	Ghrun ᦴ᧙	Ghrung ᦴ᧚	Ghrur ᦴ᧛	Ghruh ᦴ᧜	Ghruu ᦴ᧝	Ghrui ᦴ᧞

Contoh penggunaan bunyi aksara Lampung :



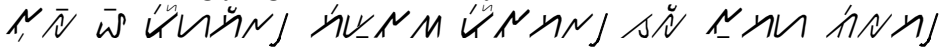
Tidak semua orang yang berusaha akan berhasil



Tapi,



Orang yang berhasil semuanya berasal dari usaha keras



2.1.5 Tanda Baca Aksara Lampung

Penggunaan tanda baca dalam penulisan aksara Lampung pada dasarnya sama dengan penggunaan tanda baca pada bahasa Indonesia. Hal yang membedakan adalah bentuk dari tanda baca tersebut. Tabel berikut ini menjelaskan mengenai tanda baca dalam aksara Lampung :

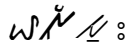
Tabel 2.22 Tanda Baca Aksara Lampung

	Tanda Seru (!)
	Tanda Tanya (?)
	Tanda Titik (.)
	Tanda Koma (,)
	Tanda Petik (’)
	Tanda Titik dua (:)
	Tanda Titik Koma (;)

Contoh penggunaan tanda baca :

Aduh ! ditulis 

Apa ? ditulis

Yaitu : ditulis 

2.1.6 Tata Cara Penulisan Aksara Lampung

Dalam penulisan aksara Lampung, hal-hal yang harus diperhatikan adalah sebagai berikut :

1. Penulisan huruf induk dalam bentuk tegak miring tidak tegak berdiri, misalnya aksara  tidak boleh ditulis .
2. Penulisan huruf melengkung bukan huruf patah, misalnya seperti  tidak boleh ditulis .
3. Penulisan anak huruf ganda dalam satu induk huruf, dahulukan anak huruf yang berbunyi vokal.

   Contoh : bunyi ’gen’ ditulis tidak boleh ditulis
4. Penempatan anak huruf pada induk aksara Lampung harus diletakan pada tempat yang tepat untuk menghindari kesalahan bunyi pada saat membacanya.

   Contoh :  dapat dibaca ’gar’ dan juga dapat dibaca ’gah’.
5. Nengen tidak boleh dirangkai atau dipasang pada satu induk aksara Lampung saja, dimana penulisannya harus mengikuti bunyinya.

   Contoh :

   DPR tidak boleh ditulis  seharusnya 
6. Tidak menggunakan tanda nengen/tanda mati pada induk aksara Lampung yang bunyinya sama dengan anak aksara Lampung. Terdapat enam induk aksara Lampung yang tidak boleh diberi tanda nengen, yaitu :

a. **nga** tidak boleh di-nengen karena terdapat anak huruf tekelubang (–)

Contoh :

Penulisan kata ’datang’ ditulis tidak boleh boleh ditulis 

b. **na** tidak boleh di-nengen karena terdapat anak huruf datasan (=)

Contoh :

Penulisan kata ’alasan’ ditulis tidak boleh ditulis

c. **ra** tidak boleh di-nengen karena terdapat anak huruf rejenjung ()

Contoh :

Penulisan kata ’pasar’ ditulis tidak boleh ditulis 

d. **wa** tidak boleh di-nengen karena terdapat anak huruf tekelungau (****)

Contoh :

Penulisan kata ’palau’ ditulis tidak boleh ditulis 

e. **ya** tidak boleh di-nengen karena terdapat anak huruf tekelingai ()

Contoh :

Penulisan kata ’wahai’ ditulis tidak boleh ditulis 

f. **ha** tidak boleh di-nengen karena terdapat anak huruf keleniah ()

Contoh :

Penulisan kata ’parah’ ditulis tidak boleh ditulis 

7. Penggunaan huruf besar atau huruf kapital misalnya pada awal kalimat, nama orang, nama kota dan sebagainya, dengan cara memperbesar ukuran huruf dengan perbandingan 3:2, dimana 3 untuk huruf kapital dan 2 untuk huruf biasa.

2.2 Konsep Pohon / *Tree*

Teori pohon merupakan salah satu teori yang cukup tua karena sudah dikenal sejak tahun 1857, dimana ketika itu matematikawan Inggris Arthur Cayley menggunakan teori pohon ini untuk menghitung jumlah senyawa kimia.

Teori pohon ini sebenarnya adalah suatu mekanisme penyelesaian suatu masalah dengan menganalogikan permasalahan tersebut kedalam struktur pohon untuk memudahkan pencarian solusi masalah tersebut. Teori pohon ini juga merupakan salah satu penerapan konsep graf. Dimana pohon itu dapat didefinisikan sebagai graf tak-berarah terhubung yang tidak mengandung sirkuit.

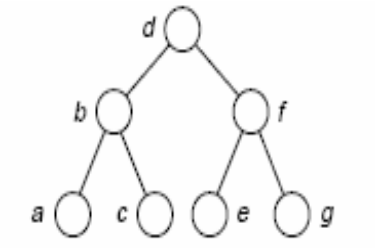
Kajian struktur data merupakan kajian yang sangat penting dalam bidang informatika. Dan di jaman sekarang ini yang teknologinya semakin berkembang, dibutuhkan struktur data yang efisien yang dapat meningkatkan kinerja program. Teori pohon ini merupakan teori yang sangat berguna dalam struktur data dimana aplikasi-aplikasi dari teori pohon ini dapat dijadikan struktur penyimpanan data yang sangat baik dalam kasus tertentu yang mana kasus tersebut sudah umum ditemui sekarang ini.

Implementasi teori pohon merupakan salah satu pilihan struktur data terbaik yang pernah ada dan masing-masing implementasi tersebut tentunya merupakan pilihan terbaik untuk kasus-kasus tertentu. Berikut beberapa contoh implementasi dari teori pohon :

1. *Binary Search Tree*

   Binary Search Tree adalah sebuah pohon biner yang mempunyai properti tambahan. Properti ini adalah :

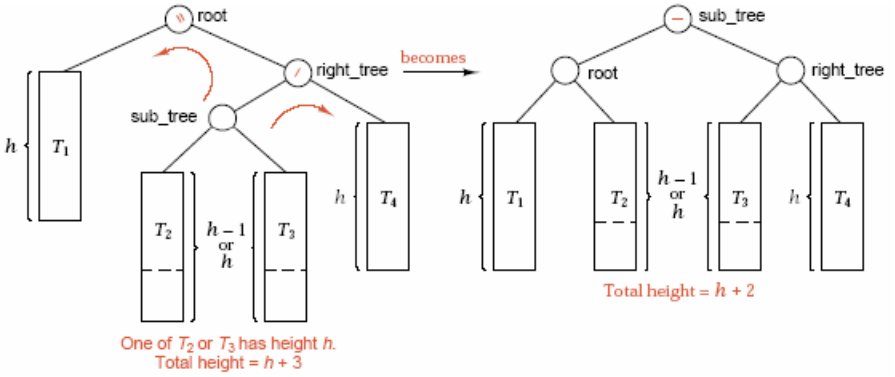
a. Semua elemen pada upapohon (subpohon) kiri nilainya lebih kecil atau sama dengan nilai akar.
b. Semua elemen pada upapohon kanan nilainya lebih besar dari nilai akar.
c. Upapohon kiri dan kanan merupakan *Binary Search Tree*.



Gambar 2.1 Contoh Binary Search Tree

2. *AVL-Tree (The Height Balance Tree)*

Dalam pohon yang benar-benar seimbang, upapohon kiri dan kanan dari setiap simpul mempunyai tinggi yang sama. Walaupun kita tidak dapat mencapai tujuan ini secara sempurna, setidaknya dengan membangun *Binary Search Tree* dengan metode penambahan elemen yang nantinya akan kita bahas, kita dapat meyakinkan bahwa setiap upapohon kiri dan kanan tidak akan pernah berselisih lebih dari 1. Jadi, sebuah *AVL-Tree* merupakan *Binary Search Tree* yang upapohon kiri dan kanan dari akarnya tidak akan berselisih lebih dari 1 dan setiap upapohon dari *AVL-Tree* juga merupakan *AVLTree*. Dan setiap simpul di *AVL-Tree* mempunyai faktor penyeimbang (*balance factor*) yang bernilai *left-higher* (upapohon kiri > kanan), *equal-height* (upapohon kiri = kanan), *righthigher* (upapohon kiri < kanan)

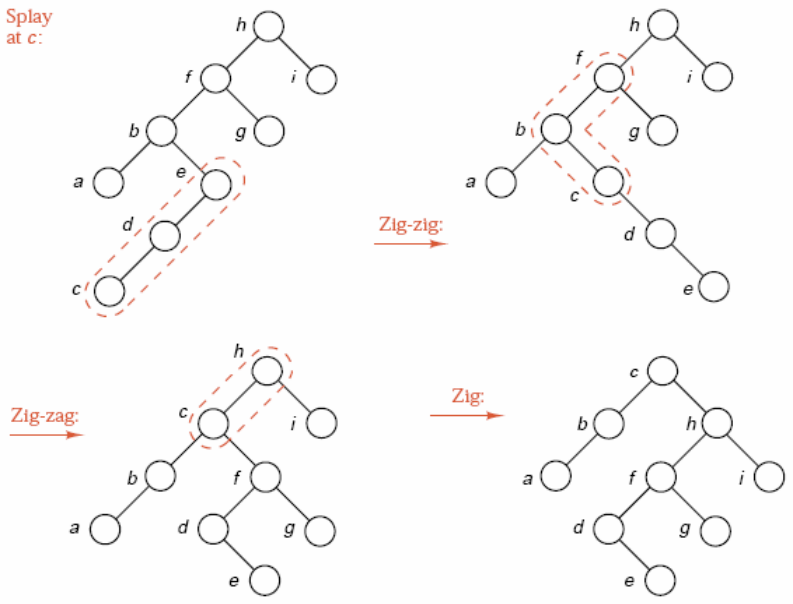


Gambar 2.2 Contoh Penyeimbangan AVL-Tree dengan Merotasi Ganda

3. *Splay-Tree (The Self-Adjusting Tree)*

   Splay-Tree merupakan salah satu modifikasi *binary search tree* dengan tujuan tertentu. Tujuan utama dari konsep *splay-tree* ini adalah untuk memudahkan pencarian dan pengambilan data terutama data yang baru masuk dan yang paling aktif diakses atau dimodifikasi.

   Perbedaan utama *splay-tree* dibandingkan *binary search tree* ataupun *AVL-Tree* adalah data baru atau yang frekuensi aksesnya tinggi berada dekat dengan akar pohon sehingga untuk mengakses data tersebut tidak diperlukan waktu yang lama dibandingkan dengan binary search tree atau AVL-Tree yang membuat kita harus menelusuri pohon sampai ke daun karena data baru akan ditempatkan sebagai daun. Untuk jenis data seperti inilah *splay-tree* sangat dibutuhkan dimana setiap kali adanya operasi penambahan atau juga pengambilan data, struktur pohonnya akan dirombak ulang dengan menaikkan data tersebut (jika berada dibagian bawah pohon / daun) sesuai dengan tingkat frekuensi akses dan keterbaharuannya.



Gambar 2.3 Contoh *Splay-Tree*

2.3 Bagian Penganalisis Leksikal

Analisis leksikal atau disebut juga dengan *scanner* bertugas mengidentifikasikan semua besaran yang membangun suatu program sumber. *Scanner* merupakan bagian dari kompilator yang menerima input berupa *stream* karakter (urutan karakter) kemudian memilih program sumber menjadi satuan leksik yang disebut dengan token yang bertujuan untuk mengelompokkan token ke dalam konstanta, nama variabel, *keyword* dan operator.

Bagian penganalisis leksikal memiliki beberapa fungsi, yaitu :

1. Merupakan fase pertama yang dilakukan oleh suatu kompilator.
2. Membaca karakter input dan memproduksi barisan dari token yang akan digunakan oleh penganalisis sintaks.
3. Membuang komentar dan ruang kosong dari program sumber seperti karakter spasi, tabulasi dan baris baru.

4. Dapat menangani kesalahan, serta menangani tabel simbol.

2.4 Bagian Penganalisis Sintaksis

Pada bagian penganalisis sintaksis terdiri dari beberapa pokok bahasan, diantaranya adalah sebagai berikut.

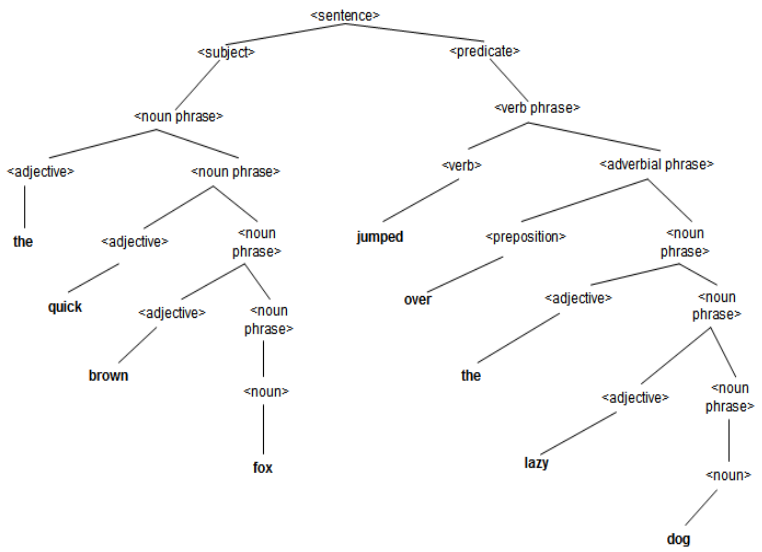
2.4.1 Penganalisis Sintaksis (*Syntax Analyzer*)

Bagian penganalisis sintaksis atau *parser* menerima token dari bagian leksikal untuk diperiksa kemunculan ukuran token. Pemeriksaan urutan ini mengacu pada suatu bahasa yang dibangun dari suatu program dalam teknik kompilasi. Bahasa yang dibangun itu dapat diwujudkan kedalam suatu pohon sintaks atau pohon penurunan.

2.4.2 Pohon Sintaks

Sebuah pohon (*tree*) adalah suatu *graph* terhubung tidak sirkuler, yang memiliki satu simpul (*node* / *vertex*) disebut akar (*root*) dan dari situ memiliki lintasan ke setiap simpul.

Pohon sintaks atau pohon penurunan (*syntax tree* / *derivation tree* / *parse tree*) merupakan pohon yang berguna untuk menggambarkan bagaimana memperoleh suatu untai (string) dengan cara menurunkan simbol-simbol variabel menjadi simbol-simbol terminal. Setiap simbol variabel akan diturunkan menjadi terminal, sampai tidak ada yang belum tergantikan.



Gambar 2.4 Contoh Sebuah Pohon Sintaks

2.4.3 *Parsing*

Proses *parsing* merupakan tahapan analisis sintaksis yang berguna untuk memeriksa urutan kemunculan token. Proses penurunan atau *parsing* dapat dilakukan dengan cara sebagai berikut :

a. Penurunan terkiri (*leftmost derivation*)

   Simbol variabel terkiri yang diperluas terlebih dulu.

b. Penurunan terkanan (*rightmost derivatin*)

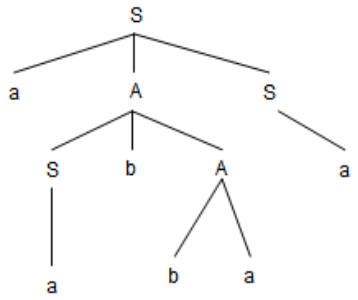
   Simbol variabel terkanan yang diperluas terlebih dulu.

Biasanya persoalan yang diberikan berkaitan dengan pohon penurunan adalah untuk mencari penurunan yang hasilnya menuju kepada suatu untai yang ditentukan. Dalam hal ini, perlu untuk melakukan percobaan pemilihan aturan produksi yang bisa menuju ke solusi.

Misal, terdapat tata bahasa bebas konteks :

$$S \rightarrow aAS \mid a$$
$$A \rightarrow SbA \mid ba$$

Untuk memperoleh untai ‘aabbaa’ dari tata bahasa bebas konteks tersebut, maka pohon penurunannya adalah :



Gambar 2.5 Pohon penurunan untuk untai ‘aabbaa’

Di dalam mengimplementasikan sebuah metode *parsing* ke dalam program perlu diperhatikan tiga hal sebagai berikut :

1. Rentang waktu eksekusi
2. Penanganan kesalahan
3. Penanganan kode

Metode parsing dapat digolongkan sebagai berikut :

1. *Top Down*

Kalau dilihat dari terminologi pohon penurunan, metode ini melakukan penelusuran dari *root*/puncak menuju ke *leaf*/daun (simbol awal sampai simbol terminal). Metode *top down* sendiri meliputi :

a. Backtrack / *backup* : *Brute Force*

Metode *Brute Force* akan memilih aturan produksi mulai dari paling kiri, dan melakukan *expand* semua non terminal pada aturan produksi sampai yang

tertinggal adalah simbol terminal. Kemungkinan pertama *string* masukan sukses di-*parsing*, bisa juga bila terjadi ekspansi yang salah untuk suatu simbol variabel maka akan dilakukan *backtrack*. Algoritma ini membangun pohon parsing yang *top down*, yaitu mencoba segala kemungkinan untuk setiap simbol non terminal.

Kelemahan dari metode *brute force* adalah :

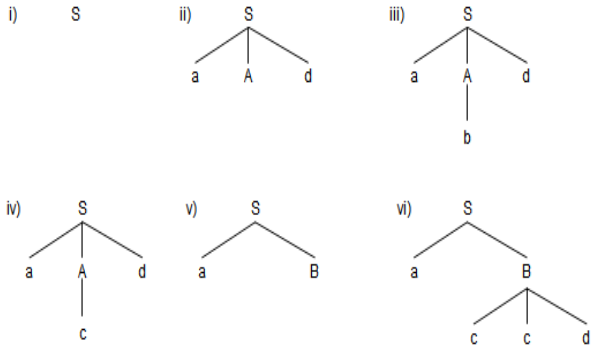
1. Mencoba untuk semua aturan produksi yang ada sehingga menjadi lambat (rentang waktu eksekusi tidak jelas).
2. Menyulitkan untuk melakukan pemulihan kesalahan.
3. Memakan banyak memori karena perlu mencatat (*backup*) lakasi *backtrack*.

Contoh, suatu bahasa dengan aturan produksi :

S → aAd | aB
A → b | c
B → ccd | ddc

(S : simbol awal)

Misal ingin dilakukan parsing untuk string ‘accd’ tahapan yang terjadi adalah sebagai berikut :



Gambar 2.6 Pohon penurunan /*derivation tree*

Terlihat pada penurunan iii) A → b, terjadi kegagalan, maka dilakukan *backtrack* dan pada penurunan iv) yang dilakukan adalah expansi A → b. Karena terjadi kegagalan maka *backtrack* lagi, dan 'naik' ke atas sampai terjadi penurunan v) S → aB. Penurunan vi) B → ccd memberikan hasil akhir bahwa string 'accd' diterima oleh bahasa tersebut.

b. No backtrack : *Recursive Descent Parser*

Recursive Descent Parser adalah salah satu cara untuk mengaplikasikan bahasa bebas konteks untuk melakukan analisis sintaksis suatu *source code*. Dalam metode ini simbol terminal maupun simbol variabelnya sudah bukan sebuah karakter, akan tetapi berupa besaran leksik sebagai simbol terminalnya dan besaran sintaks sebagai simbol variabelnya. Ciri dari *recursive descen parser* yang menonjol adalah secara rekursive menurunkan semua variabel dari awal sampai bertemu terminal dan tidak pernah mengambil token secara mundur (*no backtrack*). Berbeda dengan mesin Turing yang dapat maju dan mundur dalam melakukan *parsing*. Ciri lain dari metode ini adalah sangat bergantung pada algoritma *scan* dalam mengambil token.

Contoh jika memiliki produksi :

stmt → if expr then stmt else stmt | while expr do stmt | begin stmt_list end

Kata kunci if, while dan begin memberitahukan alternatif mana yang merupakan satu-satunya yang memiliki kemungkinan sukses jika kita menemukan non terminal stmt (statement).

2. *Bottom Up*

Metode ini melakukan penelusuran dari *leaf*/daun menuju ke *root*/puncak.

2.4.4 Tata Bahasa Bebas Konteks (*Context free Grammar*)

Tata bahasa bebas konteks berfungsi untuk menentukan sintaks dari suatu bahasa. Tata bahasa bebas konteks secara singkat terdiri dari terminal, non-terminal, simbol awal dan aturan-aturan produksi.

1. Terminal merupakan simbol dasar dari suatu rangkaian yang terbentuk.
2. Non-terminal adalah variabel sintatik yang menyatakan kumpulan dari rangkaian dimana non-terminal mendefinisikan kumpulan dari rangkaian yang dibentuk oleh tata bahasa. Non-terminal juga memberikan struktur hierarki pada suatu bahasa yang sangat bermanfaat dalam proses analisis sintaksis dan translasi.
3. Dalam suatu tata bahasa, non-terminal berfungsi sebagai simbol awal, dan kumpulan rangkaian yang dinyatakan berupa bahasa yang didefinisikan oleh tata bahasa itu.
4. Produksi-produksi dalam suatu tata bahasa menentukan perilaku terminal dan non-terminal, diikuti tanda panah, lalu diikuti oleh rangkaian dari suatu non-terminal dan terminal.

2.5 Bagian Penganalisis Semantik

Bagian terakhir dari tahap analisis dalam kompilasi adalah analisis semantik. Proses pada analisis semantik merupakan kelanjutan dari proses kompilasi sebelumnya, yaitu proses *scanning* (analisis leksikal) dan proses *parsing* (analisis

sintaks). Pada tahap ini dilakukan pengecekan pada struktur akhir yang telah diperoleh dan diperiksa kesesuaiannya dengan komponen program yang ada.

Pada hakikatnya, *semantic analyzer* ini memanfaatkan pohon sintaks yang dihasilkan pada proses *parsing* (analisis sintaks). Secara teori, sebenarnya diinginkan agar proses analisis sintaks dan analisis semantik adalah dua hal yang terpisah. Akan tetapi, pada tingkat praktis, hal ini sulit dilakukan. Proses analisis sintaks dan analisis semantik merupakan dua proses yang sangat erat kaitannya dan sulit sekali untuk benar-benar dipisahkan.

Secara global, analisis semantik digunakan untuk menentukan makna dari serangkaian instruksi yang terdapat dalam program sumber. Maka, penganalisis semantik harus mampu menentukan aksi apa yang akan dilakukan oleh operator-operator tersebut. *Compiler* akan memanggil rutin semantik. Rutin ini akan menentukan aksi khusus yang dilakukan oleh rangkaian simbol tersebut.

Untuk dapat mengetahui maknanya, rutin semantik ini mungkin memeriksa :

1. Apakah variabel yang telah ada telah didefinisikan sebelumnya
2. Apakah variabel-variabel tersebut tipenya sama
3. Apakah *operand* yang akan dioperasikan tersebut ada nilainya, dan seterusnya

Untuk menjalankan fungsinya tersebut dengan baik, *semantic analyzer* seringkali menggunakan tabel simbol. Tabel simbol ini mempunyai dua fungsi penting dalam proses *compiler*, yaitu :

1. Untuk membantu pemeriksaan kebenaran semantik dari program sumber.
2. Untuk membantu dan mempermudah dalam pembuatan *intermediate code* dan proses pembangkitan kode.

Pengecekan yang dilakukan pada analisis semantik adalah :

a) Memeriksa keberlakuan nama-nama meliputi pemeriksaan :

   1. Duplikasi

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

   2. Terdefinisi

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

b) Memeriksa tipe

Melakukan pemeriksaan terhadap kesesuaian tipe dalam *statement-statement* yang ada, misalkan bils terdapat suatu operasi, diperiksa tipe *operand*. Contohnya instruksi IF berarti tipenya *boolean,* akan diperiksa tipe *identifier* dan tipe ekspresi. Bila ada operasi antara dua *operand* maka tipe *operand* pertama harus bisa dioperasikan dengan *operand* kedua.

2.6 Kode Antara

Kode antara merupakan hasil dari tahapan analisis yang telah dibuat oleh kompilator pada saat mentranslasikan program dari bahasa tingkat tinggi. Adapaun kegunaan dari kode antara ini antara lain :

1. Untuk memperkecil usaha dalam membangun kompilator dari sejumlah bahasa ke sejumlah mesin. Dengan adanya kode antara yang lebih *machine*

independent maka kode antara yang dihasilkan dapat digunakan lagi pada mesin lainnya.

2. Proses optimasi masih lebih mudah, maksudnya beberapa strategi optimasi lebih mudah dilakukan pada kode antara daripada pada program sumber atau pada kode *assembly* dan kode mesin.
3. Dapat melihat program internal yang mudah dimengerti. Kode antara ini lebih mudah dipahami daripada kode *assembly* maupun kode mesin.

2.7 Pembangkit Kode (*Code Generator*)

Hasil dari tahapan analisis akan diterima oleh bagian pembangkit kode (code generator). Pada kasus bahasa pemrograman, biasanya kode antara yang dihasilkan tahap analisis tersebut ditranslasikan ke bahasa *assembly* atau bahasa mesin.

2.8 Pengoptimasi Kode (*Code Optimizer*)

Tahap optimasi kode bertujuan untuk menghasilkan kode program yang berukuran kecil dan lebih cepat eksekusinya. Proses optimasi kode tidak boleh mengubah makna yang ada pada kode antara yang dihasilkan oleh bagian analisis.

2.9 Pelaksana Kode

Bagian pelaksana kode (*interpreter*) adalah bagian yang akan melakukan aksi-aksi yang harus dilakukan atas kode yang telah dioptimasi oleh pengoptimasi kode. Pelaksana kode akan membaca kode satu persatu secara berurutan dilaksanakan.

2.10 Proses Perangkat Lunak

Cara pengkonversian teks ke aksara Lampung yaitu dengan menggunakan beberapa aturan-aturan penulisan aksara Lampung yang dapat dirangkum, maka akan menjadi bahasa tersendiri yang dapat dipergunakan sebagai alat bantu pengkonversian teks ke aksara Lampung.

Aturan bahasa tersebut dengan *penganalisa leksikal*, *penganalisa sintaksis*, *penganalisa semantik*, *pembangkit kode*. Hasil dari pembangkitan kode ini merupakan suatu kode hasil yang merupakan urutan-urutan penulisan aksara Lampung. Dan langkah terakhir membuat suatu interpreter yang tugasnya menterjemahkan kode hasil serta meng-generate aksi-aksi yang sesuai guna terciptanya hasil berupa tulisan aksara Lampung sesuai dengan teks input *user*.

2.11 Konsep Perangkat Lunak

Pada sub-bab ini akan menjelaskan mengenai pengertian perangkat lunak, model analisis perangkat lunak, perancangan perangkat lunak dan pengujian perangkat lunak.

2.11.1 Pengertian Perangkat Lunak

Berikut ini adalah beberapa pengertian mengenai sebuah perangkat lunak.

1. Perangkat lunak adalah perintah (program komputer) yang bila dieksekusi memberikan fungsi dan unjuk kerja seperti yang diinginkan.
2. Perangkat lunak merupakan struktur data yang memungkinkan program memanipulasi informasi secara proporsional.

3. Perangkat Lunak adalah dokumen yang menggambarkan operasi dan kegunaan program.

Berikut ini merupakan karakteristik perangkat lunak.

a. Perangkat lunak dibangun dan dikembangkan, tidak dibuat dalam bentuk yang klasik.
b. Perangkat lunak tidak pernah usang, artinya berhubungan dengan perangkat keras.
c. Sebagian besar perangkat lunak dibuat secara *custom built,* serta tidak dapat dirakit dari komponen yang sudah ada.
d. Perangkat lunak dibentuk dengan menggunakan bahasa pemrograman.
e. Karakteristik yang paling penting adalah *reusability*, artinya sebuah perangkat lunak harus didesain dan diimplementasi sehingga dapat dipakai lagi pada berbagai program yang berbeda.

2.11.2 Model Analisis Perangkat Lunak

Model analisis merupakan teknik pertama yang merepresentasikan sistem. Model analisis menggunakan kombinasi dari teks dan diagram untuk merepresentasikan kebutuhan perangkat lunak (data, fungsi, dan tingkah laku) yang telah dipahami. Model analisis ini mempermudah memahami kebutuhan yang tidak konsisten dan tidak jelas. Agar model analisis ini terbentuk dengan baik, terdapat beberapa panduan dan aturan dalam model analisis.

1. Panduan Model Analisis
   a. Analisis produk merupakan hal yang harus sangat dirawat, terutama spesifikasi kebutuhan perangkat lunak.

b. Masalah ukuran harus disepakati dan dipartisi dengan menggunakan metode yang efektif.
c. Grafik harus dapat digunakan apabila mungkin.
d. Perbedaan antara logik dan phisik dipertimbangkan.
e. Temukan hal yang dapat membantu dalam partisi kebutuhan dan partisi dokumen sebelum spesifikasi.
f. Merencanakan suatu cara untuk jalur dan evaluasi antarmuka pengguna.
g. Memikirkan alat yang dapat mendeskripsikan logik dan aturan yang terbaik dari teks narasi.

2. Aturan dalam Model Analisis
   a. Sebuah model harus berfokus pada kebutuhan yang terlihat tanpa masalah atau domain bisnis dan ditulis sebagai level yang relatif tinggi.
   b. Setiap elemen dari model analisis harus didasari dengan pengertian dari kebutuhan dan didalamnya menyediakan domain informasi, fungsi dan tingkah laku sistem.
   c. Penundaan pertimbangan dari infrastruktur dan model non fungsional sampai tahap perancangan.
   d. Minimalkan kopling yang keluar dari sistem.
   e. Pastikan model analisis menyediakan nilai untuk semua stakeholders.
   f. Pertahankan model sesederhana mungkin.

Model proses yang digunakan oleh penulis adalah model proses struktural dengan menggunakan *Data Flow Diagram* (DFD). Pendekatan aliran data menekankan logika yang mendasari sistem. Dengan menggunakan kombinasi dari

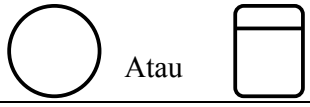
empat simbol, penganalisis sistem dapat menciptakan suatu gambaran proses-proses yang bisa menampilkan dokumentasi sistem yang solid.

DFD memiliki empat kelebihan utama melalui penjelasan naratif mengenai cara data-data berpindah disepanjang sistem. Kelebihan tersebut adalah sebagai berikut.

a. Kebebasan dari menjalankan implementasi teknis sistem yang terlalu dini.
b. Pemahaman lebih jauh mengenai keterkaitan satu sama lain dalam sistem dan subsistem.
c. Mengkomunikasikan pengetahuan sistem yang ada dengan pengguna melalui diagram aliran data.
d. Menganalisis sistem yang diajukan untuk menentukan apakah data-data dan proses yang diperlukan sudah ditetapkan.

Simbol-simbol yang digunakan dalam DFD terlihat seperti pada tabel 2.23 berikut ini.

Tabel 2.23 Simbol-simbol dalam DFD

Simbol	Arti
	Entitas
	Aliran data
 Atau	Proses
	Penyimpanan data

Untuk mengembangkan DFD dapat menggunakan pendekatan atas-bawah. Pendekatan tersebut dapat dilakukan dengan cara sebagai berikut.

1. Membuat sebuah daftar tentang kegiatan-kegiatan bisnis dan digunakan untuk menentukan berbagai macam :
   - Entitas eksternal
   - Aliran data
   - Proses-proses
   - Penyimpanan data
2. Menciptakan sebuah diagram (diagram konteks) yang menunjukkan entitas-entitas eksternal dan aliran data menuju dan dari sistem. Tidak menunjukkan setiap proses atau penyimpanan data yang mendetail.
3. Menggambar Diagram 0, level berikutnya. Menunjukkan proses-proses, namun menjaganya tetap umum. Menunjukkan penyimpanan data-penyimpanan data pada level ini.
4. Menciptakan sebuah diagram anak untuk setiap proses dalam Diagram 0.
5. Mengecek kesalahan dan memastikan label-label yang ditetapkan untuk setiap proses dan aliran data yang mengandung arti.
6. Mengembangkan suatu diagram aliran data fisik dari diagram aliran data logika. Membedakan antara proses manual dengan proses otomatis, menggambarkan file-file aktual dan dilaporkan menurut nama, dan menambahkan kontrol-kontrol untuk menunjukkan kapan proses-proses tersebut selesai atau kapan muncul kesalahan.

7. Membagi diagram aliran data fisik dengan memisahkan atau mengelompokkan bagian-bagian dari diagram agar bisa memfasilitasi pemrograman dan implementasi.

Penulis juga menggunakan *flowchart* untuk menggambarkan alur proses yang terjadi pada perangkat lunak pengkonversi teks menjadi aksara Lampung. *Flowchart* adalah penggambaran secara grafik dari langkah-langkah dan urut-urutan prosedur dari suatu program. *Flowchart* membantu analis dan programmer untuk memecahkan masalah ke dalam segmen-segmen yang lebih kecil dan membantu dalam menganalisis alternatif-alternatif lain dalam pengoperasian.

Kelebihan *flowchart* adalah sebagai berikut.

1. Relationship

   Flowchart dapat memberikan gambaran yang efektif, jelas dan ringkas tentang prosedur logic. Teknik penyajian yang bersifat grafis jelas akan lebih baik daripada uraian-uraian yang bersifat teks khususnya dalam menyajikan logika-logika yang bersifat kompleks.

2. Analisis

   Dengan adanya pengungkapan yang jelas dalam model atau chart, maka para pembaca dapat dengan mudah melihat permasalahan atau memfokuskan perhatian pada area-area tertentu sistem informasi.

3. Communication

   Karena simbol-simbol yang digunakan mengikuti suatu standar tertentu yang sudah diakui secara umum, maka flowchar merupakan alat bantu yang sangat

efektif dalam mengkomunikasikan logika suatu masalah atau dalam mendokumentasikan logika tersebut.

Tabel 2.24 berikut ini akan menjelaskan mengenai simbol-simbol dan fungsi dalam flowchart.

Tabel 2.24 Simbol-simbol dalam Flowchart

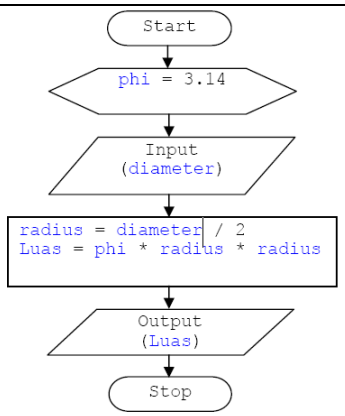
Simbol	Nama	fungsi
	Input/output	Input/output tanpa tergantung peralatannya
	Proses	Operasi yang dilakukan komputer
	connector	Menyatakan sambungan antar proses dalam halaman yang sama
	Offline connector	Menyatakan sambungan antar proses dalam halaman yang berbeda
	flow	Menyatakan jalannya arus suatu proses
	annotation	Digunakan untuk komentar tambahan
	decision	Keputusan dalam program
	Predefined process	Rincian operasi berada ditempat lain
	preparation	Pemberian nilai awal
	terminal	Menyatakan permulaan atau akhir program
	Punched card	Menyatakan input/output berasal dari kartu berlubang
	document	I/O dalam format yang dicetak
	Magnetic tape	I/O yang menggunakan pita magnetik
	Magnetic disk	I/O yang menggunakan disk magnetik
	Magnetic drum	I/O yang menggunakan drum magnetik
	On-line storage	I/O yang menggunakan penyimpanan akses langsung
	Punched tape	I/O yang menggunakan pita kertas berlubang
	Manual input	Input yang dimasukkan secara manual dari keyboard
	display	Output yang ditampilkan pada terminal
	Manual operation	Proses yang tidak dilakukan oleh komputer
	Off-line storage	Penyimpanan yang tidak dapat diakses oleh komputer secara langsung
	Communication link	Transmisi data melalui channel komunikasi.

Pseudocode terdiri dari dua kata yaitu *pseudo* yang artinya imitasi atau menyerupai dan *code* menunjukan kode dari sebuah program sehingga *pseudocode* adalah kode yang mirip dengan instruksi kode program yang sebenarnya. *Pseudocode* akan memudahkan *programmer* untuk memahami dan menggunakannya. Berikut ini merupakan aturan dari penulisan *pseudocode*.

1. *Statement* program ditulis dengan bahasa indonesia atau Inggris yang sederhana.
2. Setiap instruksi ditulis pada setiap baris.
3. Pisahkan modul-modul atau kelompok *pseudocode* dengan memberikan spasi beberapa baris untuk mempermudah pembacaan.
4. Bedakan bentuk huruf dalam penulisan *pseudocode*.
5. Berikanlah tabulasi yang berbeda untuk penulisan *pseudocode* pada instruksi-instruksi yang berada dalam *loop* atau struktur kondisi.
6. Lakukan pembatasan terhadap jumlah baris *pseudocode* pada setiap modulnya.

Tabel 2.25 berikut adalah contoh *pseudocode* dan *flowchart*.

Tabel 2.25 Contoh *Pseudocode* dan *Flowchart*

pseudocode	*flowchart*
phi ← 3.14 Input (diameter) radius ← diameter/2 Luas ← phi * radius * radius Output (Luas) end	

2.11.3 Perancangan/Desain Perangkat Lunak

Desain perangkat lunak adalah suatu proses interaktif yang melalui persyaratan diterjemahkan ke dalam ”*blue print*” untuk membangun suatu perangkat lunak. Sepanjang proses desain, kualitas desain yang melengkapi dinilai dengan serangkaian kajian teknis formal atau desain awal. Berikut ini adalah pedoman bagi evaluasi suatu desain yang baik :

a. Desain harus mengimplementasikan keseluruhan persyaratan eksplisit yang dibebankan dalam model analisis, dan harus mengakomodasi semua persyaratan implisit yang diinginkan pelanggan.
b. Desain harus menjadi panduan yang dapat dibaca.
c. Desain harus memberikan suatu gambaran lengkap mengenai perangkat lunak.

Dalam proses desain perangkat lunak terdapat prinsip-prinsip yang harus diperhatikan. Adapun prinsip-prinsip tersebut antara lain :

a. Proses desain tidak boleh menderita *tunnel vision*. Untuk mengatasinya, harus memperhatikan pendekatan-pendekatan alternatif.
b. Desain harus dapat ditelusuri sampai model analisis.
c. Desain tidak boleh berulang, hal ini dapat dihindari dengan menggunakan pola desain.
d. Desain harus ”meminimalkan kesenjangan intelektual” antara perangkat lunak dan masalah yang ada di dunia nyata.
e. Desain harus mengungkapkan keseragaman dan integrasi.
f. Desain harus terstruktur untuk mengakomodasi perubahan.

g. Desain harus terstruktur untuk berdegradasi dengan baik, bahkan pada saat data dan event-event menyimpang atau menghadapi kondisi operasi.
h. Desain bukanlah pengkodean dan pengkodean bukanlah desain.
i. Desain harus dinilai kualitasnya pada saat desain dibuat, bukanlah setelah jadi.
j. Desain harus dikaji untuk meminimalkan kesalahan-kesalahan konseptual (semantik).

2.11.4 Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan mempresentasikan kajian pokok dari spesifikasi, desain dan pengkodean. Sasaran pengujian menurut Glen Myers [MYE79] menyatakan sejumlah aturan :

a. Pengujian adlaah proses eksekusi suatu program dengan maksud menemukan kesalahan.
b. *Test case* yang baik adalah *test case* yang memiliki probabilitas tinggi untuk menemukan kesalahan yang belum pernah ditemukan sebelumnya.
c. Pengujian yang sukses adalah pengujian yang mengungkap semua kesalahan yang belum pernah ditemukan sebelumnya.

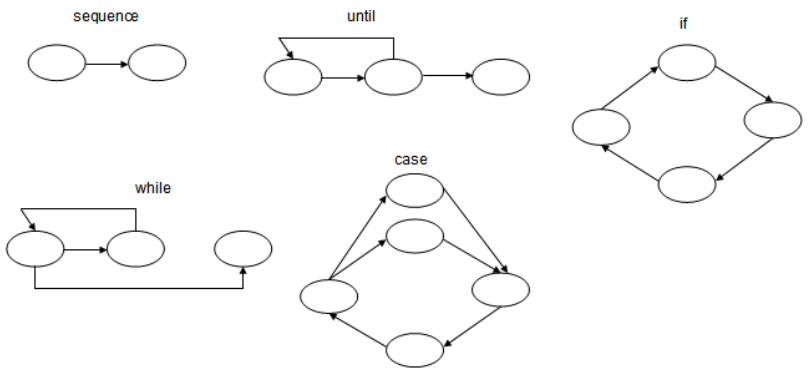
Testabilitas perangkat lunak secara sederhana adalah seberapa mudah sebuah program komputer dapat diuji. Pendekatan desain *test case* terbagi menjadi:

1. Pengujian *White Box*

Pengujian *white box* adalah metode desain *test case* yang menggunakan struktur kontrol desain prosedural untuk memperoleh *test case*. Dengan melakukan pengujian ini perekayasa sistem dapat melakukan *test case* yang :

a. Memberikan jaminan bahwa semua jalur independen pada suatu modul telah digunakan paling tidak satu kali.
b. Menggunakan semua keputusan logis pada sisi *true* dan *false*.
c. Mengeksekusi semua loop pada batasan mereka dan pada batas operasional mereka.
d. Menggunakan struktur data internal untuk menjamin validitasnya.

Dalam pengujian *white box* ini terdapat notasi diagram alir. Grafik alir ini menggambarkan aliran kontrol logika yang menggunakan notasi yang ditunjukan pada gambar berikut :



Gambar 2.7 Aliran kontrol logika pada grafik alir

2. Pengujian *Black Box*

Pengujian *black box* berfokus pada persyaratan fungsional perangkat lunak. Pengujian *black box* berusaha menemukan kesalahan dalam kategori sebagai berikut :

a. Fungsi-fungsi yang tidak benar atau hilang.
b. Kesalahan *interface*.
c. Kesalahan dalam struktur data atau akses database eksternal.
d. Kesalahan kinerja.
e. Inisialisasi dan kesalahan terminasi.

2.12 Perangkat Lunak Pendukung

Untuk membangun perangkat lunak pengkonversi teks menjadi aksara Lampung ini, penulis menggunakan beberapa perangkat lunak pendukung. Perangkat lunak tersebut antara lain sebagai berikut.

2.12.1 Borland Delphi

Delphi merupakan program aplikasi database dengan lingkungan pemrograman visual. Kelebihan Delphi dibandingkan dengan program aplikasi lainnya yaitu produktivitas. Memakai Delphi adalah cara yang paling sederhana untuk membangun aplikasi visual. Produktivitas dari pengembangan perangkat lunak dibagi menjadi lima atribut penting, yaitu :

1. Kualitas dari lingkungan pengembangan visual
2. Kecepatan dari *compiler* dibandingkan dengan kompleksitasnya
3. Kekuatan dari bahasa pemrograman dibandingkan dengan kompleksitasnya
4. Fleksibilitas dari arsitektur basis data
5. Pola desain dan pemakaian yang diwujudkan oleh *framework*

Pada pemrograman visual seperti Delphi, lingkungan pemrograman (*Integrated Development Environment* / IDE) lebih penting dari bahasanya, karena lingkungan memudahkan pemakai dalam pemakaian aplikasinya.

Sejarah singkat Borland Delphi

Turbo pascal, kemudian dikenal dengan Borland Pascal, merupakan sebuah *compiler* yang murah dengan kualitas yang tinggi, dan sebuah revolusi compiler ketika pertama kali diperkenalkan. Borland Pascal mengalami beberapa perubahan dalam sepanjang tahun, dan digunakan terutama untuk memproduksi program yang berjalan berupa display text. Ketika penggunaan *graphical user interface* diwujudkan dengan Microsoft Windows 3, Delphi, dimana Borland Pascal sebagai bahasa dasarnya, diperkenalkan. Delphi telah menjadi satu dan pertama yang menjadi dan kemudian dikenal sebagai *tool Rapid Application Development* (RAD) ketika dirilis pada tahun 1995 untuk Windows 3.1 16-bit. Delphi 2, dirilis setahun kemudian, mendukung lingkungan Windows 32-bit, dan sebuah *variant* menggunakan C++ sebagai bahasa dasarnya, C++ builder, diperkenalkan beberapa tahun kemudian.

Perancang utama delphi adalah Anders Hejlberg, yang telah mengembangkan Turbo Pascal. Dia pindah ke Microsoft pada tahun 1996.

Pada tahun 2001, versi delphi untuk Linux bernama Kylix dirilis. Kylix dikritik karena memiliki kualitas yang buruk dan tidak terjual dengan baik, dan ditinggalkan setelah setelah versi 3. Suport untuk Linux dan Windows pengembangan cross-platform (berupa Kylix dan CLX Component Library) telah ditambahkan pada tahun 2002 dengan dirilisnya Delphi 6.

Delphi 8 dirilis pada Desmber 2003, adalah hanya sebuah rilis .NET yang memungkinkan developer untuk mengkompile kode Objek Pascal Delphi menjadi .NET CIL. Hal ini juga berpengaruh signifikan terhadap IDE untuk pertama kalinya, yaitu dari model IDE *multiple-floating-window-on-desktop* menjadi mirip dan terassa seperti Microsoft Visual Studio .NET. IDE yang baru masih memiliki ”Classic Undocked” layout windows tersedia sebagai suatu option.

Versi berikutnya, yaitu Delphi 2005 (Delphi 9), menyertakan Win32 dan .NET development dalam sebuah IDE tunggal, mengiterasi ulang komitmen Borland untuk developer Win32. Delphi 2005 menyertakan manipulasi desain waktu data langsung dari sebuah database. Delphi 2005 juga menyertakan peningkatan IDE dan menambahkan sebuah pernyataan *for...in* (seperti *foreach*-nya C#) dalam bahasanya. Bagaimanapun juga, hal ini mendapat kritikan karena terdapat beberapa *bugs*; Baik Delphi 8 dan Delphi 2005 memiliki kendala dalam stabilitas ketika dikirimkan, dan masalah ini dapat diatasi hanya dengan *service pack* terpisah.

Pada akhir tahun 2005, delphi 2006 telah dirilis dan mengembangkan kombinasi dari C# dan Delphi.NET, Delphi Win32 dan C++ ke dalam sebuah IDE tunggal. Delphi 2006 jauh lebih stabil dibandingkan dengan Delphi 8 maupun Delphi 2005 kerika dikirimkan dan diujicoba, dan jauh meningkat dengan dirilisnya service pack dan beberapa perbaikan terbarunya.

Pada tanggal 8 Februari 2006, Borland mengumumkan bahwa sedang mencari pembeli dalam jalur produksi untuk IDE dan database, dengan

menyertakan delphi, untuk memusatkannya pada jalur ALM. Berita tersebut mendapat tanggapan yang optimis dari para pengguna Delphi.

Pada tanggal 6 September 2006, *Developer Tools Group* (nama kelompok yang belum diakui perusahaan) dari Borland Software Corporation merilis versi bahasa tunggal dari Borland Developer Studio, dengan menggunakan nama populer *Turbo*. Satu Set Produk Turbo menyediakan Turbo Delphi untuk Win32, Turbo Delphi untuk .NET, Turbo C++, dan Turbo C#.

Masing-masing versi terdiri dari dua edisi : *Explorer* – versi yang dapat diunduh secara gratis – dan *Professional* - versi dengan harga murah (US$899 untuk pengguna baru, dan US$399 untuk *update*-nya) yang memiliki akses terbuka untuk ribuan komponen *third-party*. Tidak seperti edisi *Personal* Delphi sebelumnya, edisi *Explorer* yang baru dapat digunakan untuk pengembangan secara komersial.

Pada tanggal 14 November 2006, Borland menyatakan bahwa daripada menjual development tools group, lebih baik memasukkannya kedalam cabang perusahaan independen yang dikenal dengan nama CodeGear.

Delphi 2007, yang merupakan versi pertama CodeGear, dirilis pada tanggal 16 Maret 2007. Delphi 2007 adalah versi Delphi pertama sejak versi 7 yang hanya mengijinkan kompilasi di lingkungan aplikasi Windows 32-bit. Fitur baru yang dimasukan men-*support* MS Build dan peningkatan pada *Visual Component Library* untuk Windows Vista. CodeGear juga memperkenalkan DBX4 sebagai versi pengganti dbExpress. Untuk pertama kalinya Delphi dapat diunduh dari Internet dan diaktivasi menggunakan sebuah kunci lisensi. Pelanggan

internasional yang ingin mendapatkan versi 7 dapat memperolehnya secara langsung di Inggris, Perancis, Jerman dan Jepang. RAD Studio 2007, yang menyertakan .NET Development, dirilis pada tanggal 5 september 2007.

2.12.2 *FontCreator*

FontCreator merupakan program khusus yang berfungsi dalam pembuatan huruf. Fasilitas-fasilitas yang terdapat pada *FontCreator* ini memudahkan kita dalam memilih font yang ingin kita *install* atau kita buat. Hal ini dibuktikan dengan tersedianya seluruh karakter secara lengkap yang mendeskripsikan setiap angka, huruf, tanda baca dan karakter-karakter spesial lainnya.

Sejarah singkat *FontCreator*

Pada tanggal 16 Mei 2003, versi 4.0 dirilis. Perbaikan atau peningkatan dari versi sebelumnya adalah telah mendukung *theme* Windows XP, peningkatan pada kemudahan instalasi font, penambahan *online* ke High-Logic pada menu *Help*, sampel *glyph* yang dapat digunakan pada *glyph* yang kosong, *zoom* pada area yang diseleksi, peningkatan pada fitur *import* gambar, penambahan beberapa tombol *toolbar* baru, *snap to grid* (titik, kontur, dan *glyph*), *snap to user* penambahan garis bantu (titik, kontur, dan *glyph*), memisahkan koleksi *TrueType* atau ekstraksi font *TrueType* dari koleksi *TrueType*, fitur validasi *glyph*, penambahan *import* pasangan *kerning* menjadi fitur *autokern*, dan sebuah item konteks menu *Font Creator Explorer*.

Pada tanggal 1 Desember 2003, versi 4.1 dirilis. Pada versi ini terdapat bebrapa tambahan fitur, diantaranya adalah penambahan pada *snap point* menjadi

garis bantu diagonal, penambahan fungsi pencarian *glyph*, penambahan fitur untuk mengubah ukuran font, penambahan tombol *free hand selection* pada *toolbar*, perubahan laporan validasi menjadi sebuah *wizard*, penambahan *auto fix* yang mengatasi masalah validasi, penambahan fitur pengurutan *glyph*, peningkatan pada fitur *paste special*, dan *minor tweaks*.

Pada tanggal 17 Agustus 2004, versi 4.5 dirilis. Adapun fitur-fitur tambahan yang ditawarkan pada versi ini adalah dapat mengeset batas sisi kiri hingga x=0 (dapat dilihat di menu Tools—Options—Font), dukungan yang lebih baik pada pencarian titikkode dari *Supplementary Planes*, jendela tranformasi *glyph* (skala, rotasi, dan sebagainya untuk semua *glyph*), memilih *glyph* yang tidak lengkap, dalam hal ini dapat menghapus *glyph* yang tidak diinginkan, mendukung *bookmark* dalam jendela *overview*, peningkatan fitur import image, penambahan dukungan *drag* dan *drop* untuk menginpor image baik pada jendela *overview* maupun pada jendela *edit glyph*, penambahan *Launch External* dari menu *Tools* untuk folder fontdan *CharMap*, secara bebas dapat memilih kontur selanjutnya dan sebelumnya (Q dan W), dukungan pada *snap to grid* dan garis bantu ketika membuat kontur baru, mendukung penguncian garis bantu, peningkatan tab penamaan pada jendela *Options*, iniformasi ini ditambahkan ketika membuat sebuah font baru yaitu pada menu File—New, men-generate fitur tambahan penamaan *postscrip* dapat dilihat pada jendela *Postscrip Names*, menghitung range *unicode* yang diperbolehkan, hal ini terlihat pada menu Font Settings—Ranges tab, menghitung codepage yang diperbolehkan pada menu Font Settings—

Ranges tab, dan penambahan pra-definisi preview text (preview.txt) dapat digunakan pada toolbar Preview.

Pada tanggal 12 Januari 2005, versi 5.0 pun dirilis. Perubahan yang dilakukan pada versi ini adalah perubahan nama *Font Creator Program* menjadi *Font Creator* saja, dukungan yang lebih baik pada penambahan karakter, dukungan yang lebih baik pada kalkulasi di jendela Font Settings, peningkatan pada opsi grid, *snap to grid* dan *snap to guidelines* untuk batasannya, dukungan pada CFF yang didasari oleh *OpenType font* (file .otf), kelengkapan komposisi (secara cepat menandai glyph yang tidak lengkap), contour operations (penggabungan, pengirisan, eksklusi dan *knife*), tab bearing pada *toolbar Transform*, peningkatan validasi *glyph*, dan *toolbar Comparison*.

Pada tanggal 30 Mei 2006, versi 5.5 pun dirilis. Perubahan yang paling signifikan yaitu perubahan namanya menjadi *FontCreator,* peningkatan tampilan antarmuka dengan penambahan tombol tambahan untuk perintah-perintah yang sering digunakan, lebih banyak shortcut dan perintah menu dasar, dukungan untuk input unicode (minimal membutuhkan Windows 2000 dan Windows XP), *tools drawing* baru yaitu persegi dan elips, peningkatan pada validasi glyph, peningkatan pengaturan PANOSE, penambahan preview font untuk membuka cara instalasi, peningkatan pada cara pengujian font, penambahan operasi align dan distribusi, peningkatan pada *wizard* transformasi (penambahan penyisipan karakter dan kelengkapan komposisi), dan peningkatan pada “seleksi glyph yang tidak lengkap”.

FontCreator versi 5.6 dirilis pada tanggal 19 Juli 2007. Fitur-fitur yang ditawarkan pada versi 5.6 adalah perbaikan pada kelengkapan komposisi (secara otomatis menghasilkan lebih dari 2000 *glyph*), skrip-skrip transformasi yang baru dan ditingkatkan, *auto center* dan zoom *glyph* ketika jendela di-resize, memunculkan *MainType* dari *Tools*, meluncurkan submenu External atau tombol pada toolbar standard, kini telah mendukung penamaan *glyph* pada jendela overview "Symbol Character" yang digunakan untuk ISO, peningkatan kinerja validasi glyph, sejumlah peningkatan pada fitur transformasi, peningkatan pada dukungan latar belakang gambar, peningkatan jendela *kerning*, opsi untuk memulai dari terakhir kali digunakan, penambahan sebuah tombol kalkulasi pada "Additional Metrics", penambahan indeks *glyph* dengan nilai min/max dengan jendela *Font Properties*, tombol *Save as* dan *Sort Font*, penambahan "Open in New Window" ketika mengklik kanan sebuah anggota *glyph* gabungan, garis *metric* dan referensi dalam jendela *Glyph Edit*, fitur baru yang menjamin validnya nama *PostScript glyph*, penambahan import pasangan *kerning* bersama dengan nilainya, penambahan link terbuka dalam tombol *web browser* ke pada field penamaan spesifik, font baru ditandai dengan garis luar yang berbeda, pengurutan *glyph* oleh titik kode Microsoft, kualitas standar Macintosh, *PostScript name*, atau jenis *glyph*.

2.12.3 Coreldraw

Coreldraw pada dasarnya dikembangkan hanya untuk Microsoft Windows dan pada saat ini berjalan hanya pada Windows 2000 dan versi windows yang

lebih baru. Versi terbaru dari coreldraw adalah versi X4 (versi 14.0.0.567) telah dirilis pada tanggal 23 januari 2008.

Versi untuk Mac OS dan Mac OS X tersedia dalam waktu yang bersamaan, namun karena tingkat penjualan yang buruk maka tidak ada kelanjutan lagi. Versi terakhir untuk Linux adalah versi 9 (dirilis pada tahun 2000, ini benar-benar tidak berjalan sesuai dengan keinginan, meskipun menggunakan versi yang dimodifikasi dari **Wine** agar dapat berjalan) dan versi terakhir untuk OS X adlaah versi 11 (dirilis pada tahun 2001). Juga, sampai dengan versi 5. Coreldraw telah dikembangkan untuk Windows 3.1 dan OS/2.

Sejarah Perkembangan CorelDRAW

Pada tahun 1985, Dr. Michael Cowpland menemukan Corel untuk menjual desktop publishing system berbasis Intel. Pada tahun 1987, Corel mengangkat teknisi-teknisi perangkat lunak yaitu Michel Bouillon dan Pat Beirne untuk mengembangkan sebuah program ilustrasi berbasis vektor untuk dikemas bersama dengan desktop publishing system tersebut. Program tersebut, CorelDRAW, pertama kali dirilis pada tahun 1989. Program tersebut menerima sambutan yang baik, dan Corel dengan sendirinya segera fokus terhadap perangkat lunak tersebut.

CorelDRAW dengan sendirinya semakin dikenal dengan dirilisnya Microsoft Windows 3.1. Disertakannya truetype pada Windows 3.1 mengubah CorelDRAW menjadi program ilustrasi yang benar-benar dapat diandalkan dengan proses instalasi tanpa membutuhkan perangkat lunak dari pihak ketiga seperti Adobe Type Manager.

Berikut ini adalah tanggal perilisan setiap versi dari CorelDRAW :

a. CorelDRAW versi 1.01 dirilis pada bulan Januari 1989.
b. CorelDRAW versi 1.01 dirilis pada bulan Maret 1989.
c. CorelDRAW versi 1.02 dirilis pada bulan April 1989.
d. CorelDRAW versi 1.10 dirilis pada bulan Juli 1989.
e. CorelDRAW versi 1.11 dirilis pada bulan Februari 1990.
f. CorelDRAW versi 2 dirilis pada tahun 1991.
g. CorelDRAW versi 3 dirilis pada bulan Oktober 1992.
h. CorelDRAW versi 4 dirilis pada bulan Mei 1993.
i. CorelDRAW versi 5 dirilis pada bulan Juli 1994.
j. CorelDRAW versi 6 dirilis pada tanggal 24 Agustus 1995.
k. CorelDRAW versi 7 dirilis pada bulan April 1997.
l. CorelDRAW versi 8 dirilis pada bulan April 1998.
m. CorelDRAW versi 9 dirilis pada bulan Desember 1999.
n. CorelDRAW versi 10 dirilis pada bulan Oktober 2000.
o. CorelDRAW Graphics Suite 11 dirilis pada bulan Agustus 2002.
p. CorelDRAW Graphics Suite 12 dirilis pada bulan Januari 2004.
q. CorelDRAW Graphics Suite X3 dirilis pada bulan Februari 2006.