BAB 2
LANDASAN TEORI

2.1 Algoritma

Algoritma adalah urutan langkah-langkah untuk memecahkan suatu masalah [8]. Di bawah ini beberapa notasi yang digunakan untuk menulis algoritma.

1. Menyatakan langkah-langkah algoritma dengan untaian kalimat deskriptif

   Dengan menggunakan untaian kalimat ini, dapat dijabarkan secara gamblang pada setiap proses langkah-langkahnya. Notasi ini baik digunakan untuk orang awam, namun sulit diterjemahkan ke *source code* [8].

2. Menggunakan bagan-alir (*flowchart*)

   Bagan-alir (*flowchart*) adalah notasi algoritmik yang menggunakan sekumpulan bentuk-bentuk geometri untuk menggambarkan proses, misalnya bentuk persegi panjang menyatakan proses, mentuk intan menggambarkan pernyataan kondisi[8]. Notasi algoritmik dengan diagram alir lebih cocok digunakan untuk kasus yang kecil, tidak cocok digunakan untuk kasus yang besar karena akan memakai berlembar lembar kertas. Selain itu, bagan alir sulit untuk diterjemahkan ke *source-code*.

3. Menggunakan *pseudo-code*

   Pseudo-code merupakan format penulisan algoritma yang menggabungkan bahasa alami dan bahasa pemrograman. Keuntungan dari menggunakan *pseudocode* adalah kemudahan dalam menerjemahkan notasi algoritmik ke *source code*[8].

2.1.1 Struktur Dasar Algoritma

Algoritma berisi langkah-langkah penyelesaian suatu masalah. Langkah-langkah tersebut dapat berupa runtunan, pemilihan, dan pengulangan. Ketiga jenis konstruksi tersebut membentuk konstruksi suatu algoritma.

Runtunan adalah struktur algoritma paling dasar yang berisi rangkaian instruksi yang diproses secara sekuensial, satu per satu, mulai dari instruksi pertama sampai terakhir. Sedangkan pemilihan adalah konstruksi yang digunakan untuk memilih beberapa aksi yang akan dijalankan. Selanjutnya adalah pengulangan,

konstruksi ini dapat dilakukan sejumlah kali, atau sampai kondisi berhenti pengulangan tercapai[8].

2.1.2 Pemilihan

Pernyataan pemilihan adalah sebuah instruksi yang dikerjakan jika kondisi tertentu dipenuhi. Contohnya, jika air di dalam panci mendidih, maka matikan api kompor. Pernyataan tersebut dapat ditulis dalam pernyataan-pemilihan (*selection-statement*), atau disebut juga pernyataan kondisional[8].

Pernyataan pemilihan memiliki struktur *if* kondisi *then* aksi. Dalam bahasa Indonesia, *if* berarti 'jika' dan *then* artinya 'maka', kondisi adalah persyaratan yang dapat bernilai benar atau salah, aksi sesudah kata *then* hanya dilaksanakan apabila kondisi bernilai benar.

Struktur *if-then* hanya memberikan satu pilihan aksi bila kondisi (persyaratan) dipenuhi (bernilai benar), dan tidak ada pilihan aksi lain bila kondisi bernilai salah. Bentuk pemilihan yang lain ialah memilih satu dari dua buah aksi bergantung pada nilai kondisinya, yaitu *if-then-else*. Struktur dari *if-then-else* yaitu '*if* kondisi *then* aksi 1 *else* aksi 2'. Pada struktur ini, jika kondisi tidak terpenuhi maka akan melaksanakan aksi 2. Gambar 2.1 berikut menunjukkan struktur *if-then-else*.

```
if ekspresi = nilai1 then
   aksi1
else
   if ekspresi = nilai2 then
    aksi2
          .
          .
          .
   else
    if ekspresi = nilai3 then
         aksi3
    else
         aksi3
     endif
   endif
endif
```

Gambar 2.1 Struktur *if-then-else*

Selain *if-then* dan *if-the-else*, ada struktur lain dalam pemilihan, yaitu konstruksi *case*. Konstruksi *case* ini lebih sederhana bila digunakan pada kasus

yang memiliki masalah lebih dari dua. Konstruksi *case* dapat dilihat pada Gambar 2.2.

```
case ekspresi
      nilai 1 : aksi 1
      nilai 2 : aksi 2
      nilai 3 : aksi 3
      .
      .
      .
      Nilai n : aksi n
      Otherwise : aksi x
endcase
```

Gambar 2.2 Struktur *case*

Ekspresi pada struktur *case* adalah sembarang ekpresi (aritmatika atau *boolean*) yang menghasilkan suatu nilai (konstanta). Konstruksi case memeriksa apakah nilai dari ekspresi tersebut sama dengan salah satu nilai 1, nilai 2, nilai 3,... nilai n. Jika ekspresi sama dengan salah satu nilai tersebut maka aksi akan dilaksanakan. Aksi yang bersesuaian dengan nilai dapat lebih dari satu, karena itu dia berupa runtunan . Jika tidak ada satupun nilai ekspresi yang cocok, maka aksi sesudah *otherwise* dikerjakan. *Otherwise* bersifat *optional*, artinya ia boleh ditulis atau tidak di dalam konstruksi *case*.

2.2 Bahasa Pemrograman *Pascal*

Bahasa *Pascal* dikembangkan oleh Niklaus Wirth, seorang profesor di bidang komputer di Technical University Zurich, Swiss, sekitar tahun 1970 [10]. Nama Pascal diambil dari nama seorang ahli matematika, yaitu Blaise Pascal. Bahasa Pascal merupakan pengembangan dari bahasa ALGOL. Pada awalnya bahasa Pascal dikembangkan sebagai bahasa untuk pengajaran tentang pemrograman.

Dalam *Pascal*, pengenal ialah nama yang dapat diberikan pada suatu elemen program, dapat berupa konstanta, variabel, fungsi, suatu prosedur, maupun suatu program. Pengenal dapat disusun dari karakter huruf maupun karakter bilangan, dengan beberapa aturan yang harus dipenuhi, yaitu:

1. Nama pengenal harus diawali dengan karakter huruf.
2. Karakter kedua dan selanjutnya dapat berupa kombinasi angka dan huruf, tetapi tidak boleh menggunakan karakter khusus seperti ?, #, dan

sebagainya. Namun ada beberapa versi Pascal yang menerima garis bawah '_' sebagai karakter penyusun nama pengenal.

3. Panjang karakter yang digunakan sebagai pengenal bisa sembarang, tapi dalam beberapa versi Pascal hanya mengenal delapan karakter awal, karakter sembilan dan seterusnya diabaikan.
4. Beberapa karakter sudah digunakan oleh Pascal untuk tujuan tertentu, sehingga tidak dapat dipakai sebagai nama pengenal. Nama-nama ini disebut kata khusus(reserved word).
5. Beberapa nama yang disebut pengenal standar juga telah mempunyai arti khusus, tetapi jika didefinisikan ulang maka dapat menjadi nama pengenal. Jika pengenal standar digunakan sebagai pengenal biasa maka arti khususnya tidak akan dipergunakan.

2.3 *Natural Language Processing*

Bahasa alami merupakan bahasa yang secara umum digunakan oleh manusia untuk berkomunikasi dalam kehidupan sehari-hari. *Natural Language Processing* (Pemrosesan bahasa alami) sendiri merupakan salah satu bidang dari ilmu kecerdasan buatan (*Artificial Intelligence*). *Natural Language Processing* mengkaji tentang hubungan interaksi antara manusia dan komputer dengan menggunakan bahasa alami manusia [11]. Ada beberapa alasan mengapa sebuah komputer agen harus dapat memroses bahasa alami, yang pertama adalah untuk berkomunikasi dengan manusia. Kedua, untuk mendapatkan informasi dari bahasa yang ditulis.

Ada lebih dari satu triliun halaman dari informasi pada web, hampir seluruhnya ditulis dalam bahasa alami. Seorang agen yang ingin melakukan akuisisi pengetahuan perlu memahami (setidaknya sebagian) ambiguitas, bahasa yang tidak teratur yang digunakan manusia.

2.4 *Grammar*

Tata bahasa (*grammar*) dapat diidentifikasikan secara formal sebagai sekumpulan dari himpunan-himpunan variabel, simbol-simbol terminal, simbol awal, yang dibatasi oleh aturan-aturan produksi [12]. Pada tahun 1959 Noam

Chomsky melakukan penggolongan tingkatan bahasa menjadi empat. Tabel 2.1 di bawah ini menunjukkan penggolongan tingkat bahasa.

Tabel 2.1 Penggolongan Tingkat Bahasa

Bahasa	Mesin Otomata	Batasan Aturan Produksi
Reguler / Tipe 3	*Finite State Automata* (FSA) meliputi *Deterministic Finite Automata* (DFA) & *Non-deterministic Finite Automata* (NFA)	α adalah simbol variabel β maksimal memiliki sebuah simbol variabel yang bila ada terletak diposisi paling kanan
Bebas Konteks / *Context Free* / Tipe 2	*Push Down Automata* (PDA)	α merupakan simbol variabel
Context Sensitive / Tipe 1	*Linier Bounded Automata*	$\|\alpha\| \leq \|\beta\|$
Unrestricted / *Phase Structure* / *Natural Language* / Tipe 0	Mesin Turing	Tidak ada batasan

Aturan produksi merupakan pusat dari tata bahasa yang mengatur bagaimana tata bahasa melakukan perubahan suatu *string* ke bentuk lainnya, dan melalui aturan produksi tersebut didefinisikan suatu bahasa yang berhubungan dengan tata bahasa tersebut [12]. Aturan produksi dapat dinyatakan dengan bentuk 'a → b' (a menghasilkan b atau a menurunkan b), dimana a menyatakan simbol-simbol ruas kiri aturan produksi (sebelah kiri tanda '→'), dan b menyatakan simbol-simbol pada ruas kanan dan bisa disebut juga hasil produksi. Simbol-simbol tersebut dapat berupa simbol terminal atau simbol variabel/non terminal. Simbol terminal adalah simbol yang tidak dapat diturunkan lagi, sedangkan simbol variabel/non terminal adalah simbol yang masih bisa diturunkan lagi.

Dengan menerapkan aturan produksi, suatu tata bahasa dapat menghasilkan sejumlah *string*. Himpunan *string* tersebut adalah bahasa yang didefinisikan oleh tata bahasa tersebut.

Contoh aturan produksi:

$$T \rightarrow a$$

Dapat dibaca 'T menghasilkan a'.

$$T \rightarrow T \mid T + E$$

Dapat dibaca 'T menghasilkan T atau T menghasilkan T + E'.

Simbol '|' menyatakan 'atau', simbol tersebut digunakan untuk mempersingkat penulisan aturan produksi yang memiliki ruas kiri yang sama.

$$T \rightarrow T$$

$$T \rightarrow T + E$$

Pada penelitian ini, penulis melakukan penerjemahan dari bahasa alami ke bahasa bebas konteks. Bahasa alami atau bahasa manusia termasuk ke dalam *grammar* tipe 0 atau *unrestricted*, dimana tidak ada batasan dalam aturan produksinya. Sedangkan bahasa bebas konteks termasuk ke dalam *grammar* tipe 2. Bahasa bebas konteks merupakan dasar dalam pembentukan *parser* [12]. Bagian *syntax* dalam suatu kompilator umumnya didefinisikan secara formal dengan notasi BNF (*Backus Naur Form* atau *Backus Normal Form)*. Beberapa simbol yang dipakai dalam notasi BNF dapat dilihat pada Tabel 2.2.

Tabel 2.2 Simbol Notasi BNF [12]

Simbol	**Keterangan**
::=	Identik dengan simbol → pada aturan produksi
|	Identik dengan simbol | pada aturan produksi
<>	Mengapit simbol variabel/non terminal
{}	Mengulang 0 sampai n kali

Contoh :

Terdapat aturan produksi:

Tabel 2.3 Contoh Aturan Produksi

S	→	NP VP
NP	→	the NP1 | PRO | PN | NP1
NP1	→	ADJS N
ADJS	→	Ɛ | ADJ ADJS
VP	→	V | V NP
N	→	file | printer
PN	→	Bill
PRO	→	I
ADJ	→	short | long | fast
V	→	printed | created | want

Notasi BNF dari *grammar* bahasa alami pada yang ada pada aturan produksi Tabel 2.3 dapat dilihat pada Tabel 2.4.

Tabel 2.4 Contoh Notasi BNF

<S>	::=	<NP> <VP>
<NP>	::=	the <NP1> | <PRO> | <PN> | <NP1>
<NP1>	::=	<ADJS> <N>
<ADJS>	::=	Ɛ | <ADJ> <ADJS>
<VP>	::=	<V> | <V> <NP>
<N>	::=	file | printer
<PN>	::=	Bill
<PRO>	::=	I
<ADJ>	::=	short | long | fast
<V>	::=	printed | created | want

2.5 *Case Folding*

Case folding merupakan tahap penyeragaman semua huruf pada teks ke dalam *case* yang sama, bisa menjadi huruf kecil (*lowercase*) atau huruf kapital (*uppercase*) [13]. Semua huruf pada data masukan dalam penelitian ini akan diseragamkan menjadi huruf kecil. Contoh dari tahap *case folding* dapat dilihat pada Tabel 2.5.

Tabel 2.5 Contoh Tahap *Case Folding*

Data Masukan	Hasil Case Folding
Jika **I**ndex sama dengan **A** maka tampilkan nilai sangat baik.	**j**ika **i**ndex sama dengan **a** maka tampilkan nilai sangat baik.

Pada Tabel 2.5 terdapat beberapa huruf kapital pada kalimatnya. Huruf kapital yang dideteksi akan diubah menjadi huruf kecil.

2.6 *Filtering*

Filtering biasanya dilakukan pada dokumen untuk menghapus beberapa *stop word* [14]. Namun pada penelitian ini, filtering digunakan untuk menghapus karakter yang dianggap tidak perlu. Karakter yang diperbolehkan dalam penelitian ini yaitu ‘a’ sampai ‘z’, ‘0’ sampai 9, koma (‘,’), titik(‘.’), ‘_’, dan spasi. Tabel 2.6 menunjukkan contoh tahap *filtering.*

Tabel 2.6 Contoh Tahap *Filtering*

Data Masukan	Hasil Case Folding
jika index sama dengan a maka tampilkan **“**nilai sangat baik**”.**	jika index sama dengan a maka tampilkan nilai sangat baik.

2.7 Scanning

Scanning merupakan tahap memilah teks masukan menjadi token-token berdasarkan kelasnya [12]. Pada penelitian ini, tahap *scanning* menerima masukan *stream* kata yang kemudian memilah teks masukan menjadi satuan leksik atau token. Token-token ini akan menjadi data masukan pada tahap *parsing*. Besaran pembangun bahasa atau leksik meliputi hal-hal sebagai berikut.

1. *Identifier*

   Identifier dapat berupa *keyword* atau nama. *Keyword* merupakan kata kunci yang sudah didefinisikan oleh suatu bahasa [12], pada kasus dalam penelitian ini contohnya adalah ‘program’, ‘masukkan’, ‘tampilkan’. Sedangkan nama dideklarasikan oleh pemakai, seperti nama sebuah variabel.

2. Nilai Konstanta

   Nilai konstanta merupakan konstanta yang ada pada suatu program. Dapat berupa konstanta nomor, string, dan sebagainya [12]. Misalkan saja dalam Pascal pada program terdapat statement:

   ```
   N := R + 5 * 10
   Kata := kata1 + ‘makan’
   A := 0.333
   Selesai := TRUE
   ```

   Maka, 5, 10, ‘makan’, 0.333, TRUE, termasuk besaran leksik berupa nilai konstan yang terdapat pada *source code* tersebut.

3. Operator dan *Delimiter*

   Operator dapat berupa operator aritmetika atau operator logika. Delimiter adalah pemisah atau pembatas, misalnya titik, koma, dan sebagainya [12].

Contoh tahap *scanning* pada penelitian ini misalkan terdapat teks masukan ‘jika x lebih dari 5.’, maka hasil dari tahap *scanning* ditunjukan oleh Tabel 2.7 berikut.

Tabel 2.7 Contoh Tahap *Scanning*

Sebelum	**Sesudah**	
jika x lebih dari 5	**Token**	**Kelas**
	jika	*Keyword*
	x	*VariableIdent*
	Lebih dari	*ComparisonOperator*
	5	*Number*

Tahap *Scanning* Bekerja berdasarkan mesin *Finite State Automata* yang ada pada bahasa Regular [12]. Pada penelitian ini, satuan leksik atau token yang terdeteksi dibandingkan dengan daftar token yang disimpan di penyimpanan data untuk mendapatkan besaran leksik atau kelas tokennya.

2.8 *Parsing*

Parsing merupakan tahap analisis sintaksis yang bertujuan untuk memeriksa kemunculan token. Metode parsing dapat digolongkan sebagai berikut [12].

1. *Top Down*

   Metode ini melakukan penurunan dari *root*/puncak menuju *leaf/*daun. Metode *top down* meliputi:

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

   b. *No. backtrack : Recursive Descent Parser*

2. *Bottom Up*

   Metode ini melakukan penurunan dari *leaf*/daun menuju root.

Dalam penelitian ini, metode parsing yang digunakan adalah *brute force*. Metode ini memilih aturan produksi paling kiri, dan melakukan penurunan pada simbol variabel pada aturan produksi sampai yang tersisa simbol terminal. Pada metode ini, penurunan dilakukan dengan mencoba segala kemungkinan untuk setiap simbol non terminal. Jika terjadi kesalahan pada penurunan, maka akan dilakukan *backtrack* [12].

Contoh terdapat aturan produksi:

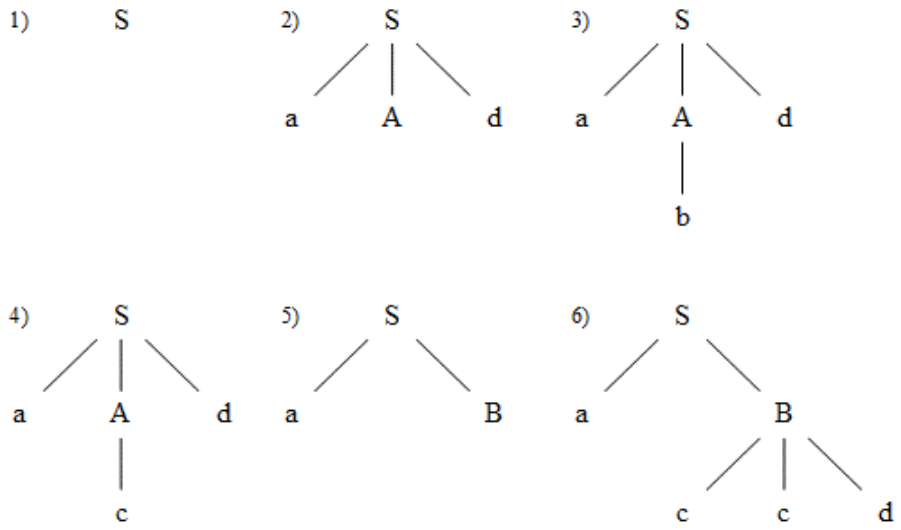
$$S \rightarrow aAd \mid aB$$

$$A \rightarrow b \mid c$$

$$B \rightarrow ccd \mid ddc$$

Misalnya dari aturan produksi diatas dilakukan parsing untuk *string* ‘accd’.

S → aAd | aB
A → b | c
B → ccd | dcc



Gambar 2.3 Pohon Penurunan

Tahap yang ada pada Gambar 2.3, terlihat pada penurunan nomor 3 terdapat kesalahan, maka dilakukan *backtrack* pada penurunan 4. Namun masih terdapat kesalahan, maka akan melakukan *backtrack* lagi sampai semu *string* bisa diturunkan.

2.9 Pohon Sintaks

Pohon sintaks atau pohon penurunan berguna untuk menggambarkan bagaimana memperoleh suatu *string* dengan cara menurunkan simbol-simbol variabel menjadi simbol-simbol terminal [12]. Setiap simbol variabel akan diturunkan sampai tidak ada lagi simbol variabel atau hanya menyisakan simbol terminal. Proses penurunan atau *parsing* bisa dilakukan dengan cara sebagai berikut [12].

1. Penurunan paling kiri (*leftmost derivation*): simbol variabel paling kiri yang akan diperluas terlebih dahulu.

   Misal, terdapat tata bahasa bebas konteks:

   S → aAS | a
   A → SbA | ba

Untuk memperoleh untaian ‘aabbaa’ dari tata bahasa bebas konteks diatas maka penurunannya sebagai berikut.

S => aAS => aSbAS => aabAS => aabbaS => aabbaa.

2. Penurunan paling kanan (*rightmost derivation*): simbol variabel atau non terminal paling kanan yang akan diperluas terlebih dahulu.

   Misal, terdapat tata bahasa bebas konteks:

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

   Untuk memperoleh untaian ‘aabbaa’ dari tata bahasa bebas konteks diatas maka penurunannya sebagai berikut.

   S=> aAS => aAa => aSbAa => aSbbaa => aabbaa.

2.10 Perancangan Sistem

Perancangan sistem bertujuan untuk menggambarkan, merencanakan, dan pembuatan sketsa dari beberapa elemen terpisah menjadi satu kesatuan sistem yang akan dibangun.

2.10.1 Diagram Konteks

Diagram konteks adalah diagram yang terdiri dari suatu proses dan menggambarkan ruang lingkup suatu sistem [15]. Diagram konteks merupakan level tertinggi dari *Data Flow Diagram* (DFD), yang dipakai untuk menggambarkan aliran data *input* dan *output* sistem. Dalam diagram konteks hanya memiliki satu proses, yaitu sistem secara keseluruhan. Pada diagram konteks tidak memiliki *data store*.

2.10.2 Data Flow Diagram (DFD)

Data Flow Diagram (DFD) digunakan untuk menggambarkan sistem sebagai suatu jaringan proses fungsional yang dihubungkan dengan aliran data [15]. DFD merupakan penjabaran dari diagram konteks, dimana setiap proses dijelaskan secara detail. Elemen dari DFD adalah sebagai berikut.

1. Kesatuan Luar (*External Entity*)

   Kesatuan luar adalah sesuatu yang berada di luar sistem, tetapi memberikan atau menerima data dari sistem [15]. Kesatuan luar dapat berupa manusia, organisasi, atau sistem lain.

2. Arus Data (*Data Flow*)

   Arus data merupakan tempat mengalirnya data yang menghubungkan komponen-komponen pada sistem [15]. Arus data ini mengalir pada kesatuan luar, proses, dan simpanan data.

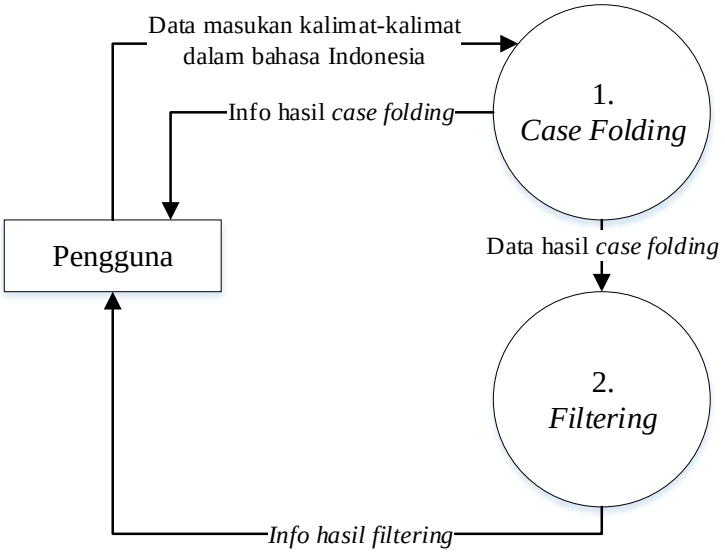
3. Proses (*Process*)

   Proses merupakan apa yang dikerjakan sistem, proses dapat mengelola data masukan menjadi informasi keluaran [15]. Proses berfungsi mengubah satu atau beberapa mata masukan menjadi satu atau beberapa keluaran informasi dengan spesifikasi yang ditentukan.

4. Simpanan Data (*Data Store*)

   Simpanan data merupakan tempat menyimpan data yang dibutuhkan oleh sistem, simpanan data dapat menerima atau mengirimkan data ke proses [15].

Berikut adalah contoh dari *data flow diagram* dimana terdapat pengguna yang mengirim data pengujian akurasi ke proses pengujian akurasi, kemudian proses pengujian akurasi mengambil data pengujian akurasi dari *data store* data_uji, dan proses pengujian akurasi mengirim hasil pengujian akurasi ke pengguna.



Gambar 2.4 Contoh *Data Flow Diagram*

2.10.3 Spesifikasi Proses

Spesifikasi proses merupakan keterangan dari semua proses yang ada pada DFD. Dalam spesifikasi proses yang perlu dicantumkan yaitu nomor urut proses, nama proses, sumber, *input*, *output*, tujuan, dan logika proses [15]. Penulisan logika

proses dapat menggunakan bahasa deskriptif maupun *pseudocode*. Tabel 2.8 menunjukkan contoh dari spesifikasi proses.

Tabel 2.8 Tabel Spesifikasi Proses

No.	Proses	Keterangan
1	No. Proses	1
	Nama Proses	*Case Folding*
	Sumber	Pengguna
	Input	Data masukan kalimat-kalimat dalam bahasa Indonesia
	Output	Info Hasil *Case Folding*
	Logika Proses	1. Sistem menerima data teks masukan dari pengguna 2. Sistem mengubah huruf teks masukan menjadi huruf kecil 3. Sistem menampilkan dan mengirimkan hasil *case folding*

2.10.4 Kamus Data

Kamus data berfungsi untuk memudahkan mengartikan aplikasi secara detail dan mengorganisasikan semua elemen data yang digunakan dalam sistem secara presisi, sehingga pengguna dan penganalis sistem mempunyai dasar pengertian yang sama tentang masukan, keluaran, penyimpanan dan proses [15]. Contoh kamus data dapat dilihat pada Tabel 2.9.

Tabel 2.9 Kamus Data

1	Nama	Data masukan kalimat-kalimat dalam bahasa Indonesia
	Digunakan di	Proses penerjemah bahasa alami ke *source code* (masukan) Proses *Preprocessing* (masukan) Proses *Case folding* (masukan)
	Deskripsi	Data masukan kalimat-kalimat dalam bahasa Indonesia = teks

2.11 Akurasi

Perhitungan akurasi digunakan untuk mengetahui tingkat keberhasilan dari proses penerjemahan oleh sistem dengan membandingkan *source code* harapan dengan hasil penerjemahan. Pada penelitian ini, akurasi dari hasil penerjemahan akan dihitung dengan rumus sebagai berikut [16].

$$Akurasi = \frac{Jumlah\ hasil\ source\ code\ sistem\ benar}{Jumlah\ sampel\ data\ uji}\ X\ 100\%$$

2.12 PHP

PHP secara umum dikenal sebagai bahasa pemrograman script yang membuat dokumen HTML secara *on the fly* yang dieksekusi di server web, dokumen HTML

yang dihasilkan dari suatu aplikasi bukan dokumen HTML yang dibuat dengan menggunakan editor teks atau editor HTML, dikenal juga sebagai bahasa pemrograman *server side* [17].

PHP dikembangkan menjadi bahasa pemrograman script yang dijalankan di atas platform sistem operasi secara langsung. PHP juga dapat digunakan dan dijalankan dari desktop, yang disebut PHP CLI (Command Line Interface). Dalam penelitian ini, PHP digunakan untuk pemrosesan bahasa alami ke source code, dari mulai preprocessing sampai proses translasi.