BAB 2
LANDASAN TEORI

2.1. Algoritma

Algoritma adalah runtunan langkah-langkah untuk menyelesaikan suatu masalah [1]. Notasi algoritmik dapat ditulis dengan berbagai notasi sebagai berikut [1].

1. Menyatakan langkah-langkah dengan untaian kalimat deskriptif.

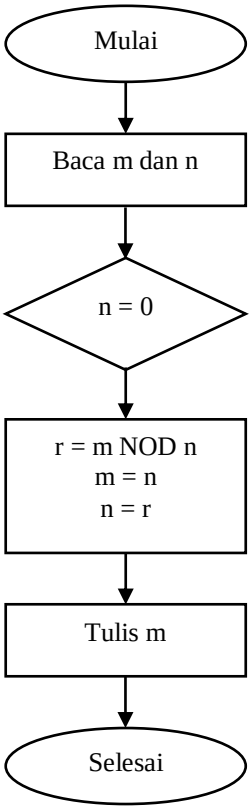
   Dengan menggunakan untaian kalimat ini, dapat dijabarkan secara gamblang pada setiap proses langkah-langkahnya [1]. Notasi ini baik digunakan untuk orang awam, namun sulit diterjemahkan ke *source code*.

```
Jika n > 60 maka
  Tampilkan ‘lulus’.
Tetapi jika tidak
  Tampilkan ‘tidak lulus’.
```

Gambar 2.1 Notasi Algoritmik Dengan Untaian Kalimat Deskriptif

2. Bagan Alur (*flowchart*).

   Bagan alur (*flowchart*) adalah notasi algoritmik yang menggunakan sekumpulan bentuk-bentuk geometri untuk menggambarkan proses, misalnya bentuk persegi panjang menyatakan proses, bentuk intan menggambarkan pernyataan kondisi [1]. 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 alur sulit untuk diterjemahkan ke *source code*.



Gambar 2.2 Notasi Algoritmik Dengan Bagan Alur [1]

3. *Pseudocode*

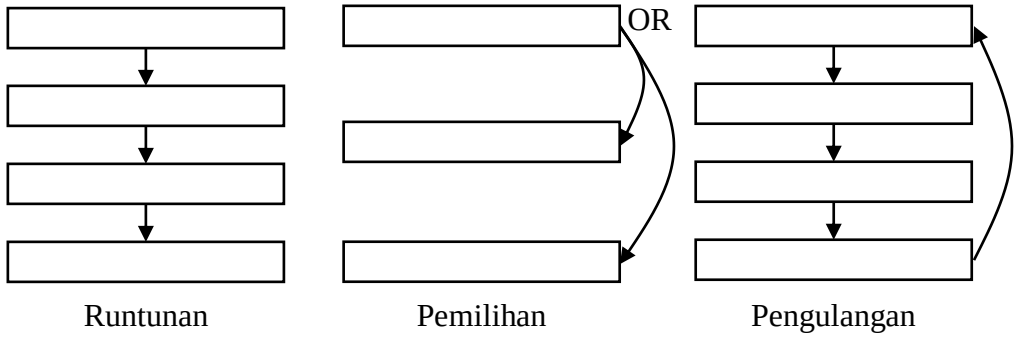
   Pseudocode merupakan format penulisan algoritma yang menggabungkan bahasa alami dan bahasa pemrograman [1]. Keuntungan dari menggunakan *pseudocode* adalah kemudahan dalam menerjemahkan notasi algoritmik ke *source code*.

```
PROGRAM Euclidean

DEKLARASI
   m, n : integer
   r : integer

ALGORITMA
   read(m,n)
   while n ≠ 0 do
      r ← m MOD n
      m ← n
      n ← r
   endwhile
   write(m)
```

Gambar 2.3 Notasi Algoritmik Dengan *Pseudocode* [1]

2.1.1. Struktur Dasar Algoritma

Algoritma berisi langkah-langkah penyelesaian suatu masalah. Dalam penyelesaian masalah algoritma berisi tiga konstruksi dasar, yaitu runtunan, pemilihan atau percabangan, dan pengulangan [1].



Gambar 2.4 Struktur Dasar Algoritma [1]

Runtunan merupakan struktur yang dikerjakan secara sekuensial, dari perintah 1, sampai perintah n dilakukan secara berurut, tidak boleh ada yang dilewat atau ditukar posisinya. Sedangkan pemilihan adalah konstruksi yang digunakan untuk memilih beberapa aksi yang akan dijalankan. Yang terakhir adalah konstruksi pengulangan, konstruksi ini memungkinkan untuk mengulang perintah sesuai dengan kondisi yang dibutuhkan.

2.1.1.1. Runtunan

Runtunan merupakan struktur paling dasar yang berisi rangkaian instruksi yang diproses secara berurut (sekuensial), satu per satu, mulai dari instruksi pertama sampai instruksi terakhir [1]. Runtunan memiliki satu atau lebih pernyataan, jika pernyataan dikerjakan tidak disesuaikan dengan urutannya maka dapat menghasilkan hasil akhir yang berbeda.

Misalkan terdapat masalah untuk mencari luas dari persegi panjang kemudian menampilkan hasilnya ke monitor, kasus ini dapat diselesaikan dengan runtunan. Penyelesaiannya dapat dilihat pada Gambar 2.5.

```
PROGRAM hitung_luas

KAMUS
    panjang : integer
    lebar : integer
    luas : integer

ALGORITMA
    panjang ← 150
    lebar ← 80
    luas ← panjang * lebar
    write(luas)
```

Gambar 2.5 Contoh Runtunan

2.1.1.2. Pemilihan

Pernyataan pemilihan adalah cara yang digunakan untuk pengambilan keputusan dari kondisi yang ada. Contohnya, jika sepeda motor mogok, maka gunakan angkot untuk ke kampus. Pernyataan tersebut dapat ditulis dalam pernyataan pemilihan (*selection-statement*), atau disebut juga pernyataan-kondisional [1].

Pemilihan memiliki struktur '***if*** kondisi ***then*** aksi'. Dalam bahasa Indonesia, *if* berarti 'jika' dan *then* berarti 'maka', kondisi adalah persyaratan yang dapat bernilai benar atau salah, dan aksi hanya dilakukan jika kondisi bernilai benar. Contoh struktur pemilihan *if-then* dapat dilihat pada Gambar 2.6.

```
PROGRAM keterangan_indeks

KAMUS
    indeks : char

ALGORITMA
    read(indeks)
    if indeks = 'E' then
        write('Tidak Lulus')
    endif
```

Gambar 2.6 Contoh Pernyataan Pemilihan Satu Aksi

Pada Gambar 2.7, kondisi yang dimiliki adalah 'indeks = E', dan aksi yang dimiliki adalah 'write ('Tidak Lulus')'. Jika kondisi terpenuhi, yaitu indeks memiliki nilai E, maka aksi baru akan dijalankan. Akan tetapi bila kondisi tidak terpenuhi atau indeks tidak memiliki nilai E, maka aksi tidak akan dijalankan.

Struktur *if-then* hanya memberikan satu aksi, jika kondisi bernilai benar, aksi baru akan dijalankan, dan jika kondisi bernilai salah maka tidak ada aksi yang dijalankan. Untuk itu terdapat bentuk pemilihan lain, yaitu *if-then-else*. Bentuk ini memilih satu dari dua buah aksi berdasarkan kondisinya [1]. Struktur dari i*f-then-else* yaitu '***if*** kondisi ***then*** aksi 1 ***else*** aksi 2'. Pada struktur ini, jika kondisi tidak terpenuhi maka akan melaksanakan aksi 2.

```
PROGRAM keterangan_indeks

KAMUS
    indeks : char

ALGORITMA
    read(indeks)
    if indeks = 'E' then
        write('Tidak Lulus')
    else
        write('Lulus')
    endif
```

Gambar 2.7 Contoh Pernyataan Pemilihan Dua Aksi

Pada Gambar 2.7, jika kondisi tidak terpenuhi atau indeks bernilai selain E, maka akan melakukan aksi 2, yaitu write('lulus').

Selain *if-then* dan *if-the-else*, ada struktur lain dalam percabangan, yaitu konstruksi *case*. Konstruksi *case* ini lebih sederhana bila digunakan pada kasus yang memiliki masalah lebih dari dua. Konstruksi *case* dapat dilihat pada Gambar 2.8.

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

Gambar 2.8 Konstruksi *Case*

Ekspresi adalah sembarang aritmetika atau *boolean* yang menghasilkan suatu nilai (konstanta) [1]. Konstruksi *case* akan memeriksa apakah nilai dari ekspresi sama dengan salah satu nilai yang ada pada bandan *case*, jika nilai ekspresi sama dengan nilai 2, maka aksi 2 akan dijalankan. Namun jika nilai pada ekspresi tidak ada yang sama, maka aksi pada *otherwise* akan dijalankan. Contoh dari pemilihan *case* ini dapat dilihat pada Gambar 2.9.

```
case indeks
    'A' : write('Sangat Baik')
    'B' : write('Baik')
    'C' : write('Cukup')
    'D' : write('Kurang')
    'E' : write('Sangat Kurang')
    otherwise : write('Indeks Salah')
endcase
```

Gambar 2.9 Contoh Pernyataan Pemilihan *Case*

2.1.1.3. Pengulangan

Pengulangan adalah proses untuk melaksanakan perintah sebanyak n kali atau sampai kondisi terpenuhi. Struktur pengulangan secara umum terdiri atas dua bagian, yaitu [1]:

1. Kondisi pengulangan, yaitu, ekspresi *bolean* yang harus dipenuhi untuk masuk ke badan pengulangan. Kondisi ini ada yang dinyatakan oleh pemrogram, atau dikelola oleh sendiri komputer.
2. Badan pengulangan, yaitu, bagian perintah yang diulang.

Dalam algoritma terdapat beberapa macam konstruksi pengulangan, namun dalam buku [1] hanya memberikan tiga macam konstruksi pengulangan, yaitu, pernyataan *for*, pernyataan *while*, pernyataan *repeat*. Penjelasan dari masing-masing konstruksi sebagai berikut.

1. Pernyataan *for*

   Pernyataan *for* digunakan untuk menghasilkan pengulangan sebanyak n kali, nilai n telah ditentukan oleh pembuat program. Untuk mencacah berapa kali pengulangan dibutuhkan sebuah variabel pencacah (variabel *counter*). Nilai variabel ini akan bertambah atau berkurang setiap kali pengulangan berlangsung. Ketika nilai pencacah sudah mencapai jumlah yang ditentukan maka pengulangan akan berhenti. Pengulangan *for* terdapat dua macam, yaitu, pengulangan *for* menaik, dan menurun. Contoh pernyataan pengulangan *for* dapat dilihat pada *pseudocode* Gambar 2.10.

```
PROGRAM cetak_HaloDunia

KAMUS
   i : integer

ALGORITMA
   for i ← 1 to 10 do
     write ('Halo Dunia')
   endfor
```

Pengulangan For Menaik

```
PROGRAM cetak_HaloDunia

KAMUS
   i : integer

ALGORITMA
   for i ← 10 downto 0 do
write ('Halo Dunia')
   endfor
```

Pengulangan For Menurun

Gambar 2.10 Contoh Pernyataan *For*

Algoritma pada Gambar 2.10, terdapat beberapa hal yang perlu diperhatikan, yaitu:

1. Variabel i merupakan pencacah pada pengulangan *for*.

2. Pada pengulangan *for* menaik, variabel pencacah akan diberi nilai awal. Sedangkan pada pengulangan *for* menurun, variabel pencacah diberi nilai akhir.
3. Pada pengulangan *for* menaik nilai variabel pencacah akan bertambah satu setiap pengulangan dilaksanakan. Namun pada pengulangan *for* menurun, variabel pencacah akan berkurang satu setiap pengulangan.
4. Pada pengulangan *for* menaik, 10 merupakan nilai akhir dari pengulangan *for*, jika nilai pada variabel pencacah sudah mencapai nilai akhir maka pengulangan berhenti. Pada pengulangan *for* menurun, 0 merupakan nilai awal dari pengulangan *for*.
5. Perintah 'write ('Halo Dunia')' merupakan badan pengulangan *for*, dan akan dijalankan sebanyak nilai akhir yang ditentukan.

2. Pernyataan *while*

Pengulangan *while* memiliki bentuk umum '***while*** kondisi ***do*** aksi'. Aksi akan dilaksanakan berulang kali selama kondisi bernilai *true* [1]. Pengulangan akan berhenti bila kondisi bernilai *false*. Dalam pengulangan *while* berhenti maka diperlukan sebuah aksi yang dapat mengubah kondisi bernilai *false*. Contoh pengulangan *while* dapat dilihat pada Gambar 2.11.

```
PROGRAM cetak_HaloDunia

KAMUS
   i : integer

ALGORITMA
   i ← 0
   while i < 10 do
      write ('Halo Dunia')
      i ← i + 1
   endwhile
```

Gambar 2.11 Contoh Pengulangan *While*

Yang perlu diperhatikan pada Gambar 2.11 yaitu:

1. Nilai variabel i harus didefinisikan di awal, sebelum pengulangan dilaksanakan.

2. Dalam badan pengulangan, variabel i ditambah 1 agar suatu ketika menghasilkan kondisi *false*.

3. Pernyataan *repeat*

   Pernyataan *repeat* memiliki bentuk umum '***repeat*** aksi ***until*** kondisi'. Berbeda dengan dua pernyataan pengulangan sebelumnya, pada pengulangan sebelumnya aksi dijalankan setelah kondisi selesai diperiksa, namun pada pengulangan *repeat* aksi terlebih dahulu dijalankan sebelum kondisi diperiksa. Aksi akan dijalankan sampai kondisi bernilai *true*, jika kondisi masih bernilai *false* pengulangan akan terus dilakukan. Contoh pengulangan *repeat* dapat dilihat pada Gambar 2.12.

```
PROGRAM cetak_HaloDunia

KAMUS
   i : integer

ALGORITMA
   i ← 0
   repeat
      write ('Halo Dunia')
      i ← i + 1
   until i > 10
```

Gambar 2.12 Contoh Pengulangan *Repeat*

Yang perlu diperhatikan pada Gambar 2.12 yaitu:

1. Aksi akan dijalankan terlebih dahulu, atau dengan kaya lain aksi dijalankan baru kemudian kondisi diperiksa.
2. Jika kondisi masih bernilai *false* maka pengulangan bernilai *true*.
3. Jika kondisi bernilai *true*, pengulangan akan berhenti.

2.2. Bahasa Pemrograman *Pascal*

Bahasa *Pascal* dikembangkan oleh Niklaus Wirth, seorang profesor di bidang komputer di Technical University Zurich, Swiss, sekitar tahun 1970 [8]. 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 [8]. Pengenal dapat disusun dari karakter huruf maupun karakter bilangan, dengan beberapa aturan yang harus dipenuhi [8], 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*

Natural language processing (pemrosesan bahasa alami) merupakan salah satu bidang dari ilmu kecerdasan buatan (*Artificial Intelligence*). Bahasa natural sendiri adalah bahasa yang secara umum digunakan oleh manusia untuk berkomunikasi dalam kehidupan sehari-hari. *Natural language processing* mengkaji tentang hubungan interaksi antara manusia dan komputer dengan menggunakan bahasa alami manusia. Dalam melakukan pemrosesan bahasa alami, bahasa yang diterima oleh komputer harus diproses dan dipahami terlebih dahulu agar maksud dari pengguna dipahami oleh komputer [9]. Salah satu

tantangan dalam melakukan pemrosesan bahasa alami adalah pemilihan arti yang tepat dari suatu kata bermakna ganda, seperti kata ‘bisa’, kata ‘bisa’ dapat berarti ‘racun’, bisa juga bermakna ‘dapat’ sesuai dengan kalimatnya [10]. Ada beberapa area penelitian utama NLP, yaitu [9]:

1. *Question Answering System* (QAS)

   QAS merupakan kemampuan komputer untuk menjawab pertanyaan dari pengguna. Pengguna dapat langsung bertanya ke komputer dalam bahasa natural yang digunakannya.
2. *Summarization*

   Summarization merupakan kemampuan komputer untuk meringkas sekumpulan dokumen.
3. *Machine Translation*

   Machine Translation merupakan kemampuan komputer untuk memahami bahasa manusia dan menerjemahkannya ke bahasa lain.
4. *Speech Recognition*

   Speech Recognition proses pembangunan mode untuk digunakan komputer untuk mengenali bahasa yang diucapkan oleh pengguna.
5. *Document Classification*

   Document Classification merupakan kemampuan komputer untuk menentukan dimana tempat terbaik dokumen yang baru dimasukkan ke dalam sistem.

Dalam penelitian ini, merupakan implementasi dari *machine translation.* Karena pada penelitian ini menerjemahkan dari bahasa alami dalam bahasa Indonesia ke *source code* dalam bahasa *Pascal*.

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 [11]. Pada tahun 1959 Noam Chomsky melakukan penggolongan tingkatan bahasa menjadi empat, seperti yang ditunjukkan pada tabel berikut [11].

Tabel 2.1 Penggolongan Tingkat Bahasa [11]

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 [11]. 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 [11]. 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 [11].

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*. Bagian *syntax* dalam suatu kompilator umumnya didefinisikan secara formal dengan notasi BNF (*Backus Naur Form* atau *Backus Normal Form)* [11]. Beberapa simbol yang dipakai dalam notasi BNF dapat dilihat pada Tabel 2.2 [11].

Tabel 2.2 Simbol Notasi BNF

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 [9]:

Tabel 2.3 Contoh Aturan Produksi [9]

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*) [12]. Pada penelitian ini, semua huruf pada data masukan akan diseragamkan menjadi huruf kecil. Contoh dari tahap *case folding* dapat dilihat pada Tabel 2.5. Pada Tabel 2.5 data masukan memiliki beberapa huruf kapital, huruf kapital yang dideteksi akan diubah menjadi huruf kecil.

Tabel 2.5 Contoh Tahap *Case Folding*

Data Masukan	Hasil *Case Folding*
Buatkan variabel **X** dengan tipe data **I**nteger.	**b**uat variabel **x** dengan tipe data **i**nteger.

2.6. *Filtering*

Filtering biasanya dilakukan pada dokumen untuk menghapus beberapa *stop word* [13]. Namun pada penelitian ini, *filtering* digunakan untuk menghapus karakter yang dianggap tidak dibutuhkan dalam penelitian ini. Karakter yang diperbolehkan dalam penelitian ini yaitu ‘a’ sampai ‘z’, ‘0’ sampai ‘9’, koma (‘,’), titik (‘.’), ‘_’, dan spasi. Pada Tabel 2.6, dideteksi terdapat karakter ‘!’, karakter tersebut akan dihapus, karena tidak sesuai dengan kriteria yang ditentukan.

Tabel 2.6 Contoh Tahap *Filtering*

Data Masukan	Hasil *Filtering*
buat variabel x dengan tipe data integer**!**.	buat variabel x dengan tipe data integer.

2.7. *Scanning*

Scanning merupakan tahap memilah teks masukan menjadi token-token berdasarkan kelasnya. 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 [11].

1. *Identifier*

   Identifier dapat berupa *keyword* atau nama. *Keyword* merupakan kata kunci yang sudah didefinisikan oleh suatu bahasa, 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 teks. Dapat berupa nomor, string, dan sebagainya.

3. Operator dan *Delimiter*

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

Contoh tahap *scanning* pada penelitian ini misalkan terdapat teks masukan ‘tampilkan hello world.’, maka hasil dari tahap *scanning* adalah sebagai berikut.

Tabel 2.7 Contoh Tahap *Scanning*

Sebelum	**Sesudah**	
tampilkan halo dunia.	**Token**	**Kelas**
	tampilkan	*Keyword*
	halo dunia	*String*
	.	.

2.8. *Parsing*

Parsing merupakan tahap analisis sintaksis yang bertujuan untuk memeriksa kemunculan token [11]. Metode parsing dapat digolongkan sebagai berikut [11].

1. *Top Down*

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

   a. *Backtack/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.*

Contoh terdapat aturan produksi:

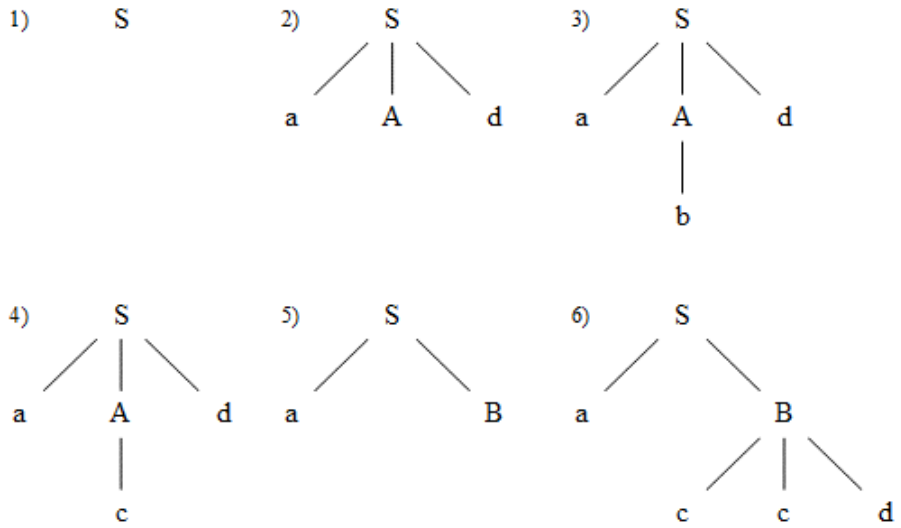
$$S \rightarrow aAd \mid aB$$

$$A \rightarrow b \mid c$$

$$B \rightarrow ccd \mid ddc$$

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

S → aAd | aB
A → b | c
B → ccd | dcc



Gambar 2.13 Pohon Penurunan

Tahap yang ada pada Gambar 2.13, 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 Sintaksis

Pohon sintaksis atau pohon penurunan berguna untuk menggambarkan bagaimana memperoleh suatu *string* dengan cara menurunkan simbol-simbol variabel menjadi simbol-simbol terminal [11]. 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 [11].

1. Penurunan paling kiri (*leftmost derivation*): simbol variabel paling kiri 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 => 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 [14]. 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. DFD merupakan penjabaran dari diagram konteks, dimana setiap proses dijelaskan secara detail. Elemen dari DFD adalah sebagai berikut [14].

1. Kesatuan Luar (*External Entity*)

   Kesatuan luar adalah sesuatu yang berada di luar sistem, tetapi memberikan atau menerima data dari sistem [14]. 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 [14]. Arus data ini mengalir pada kesatuan luar, proses, dan simpanan data.

3. Proses (*Process*)

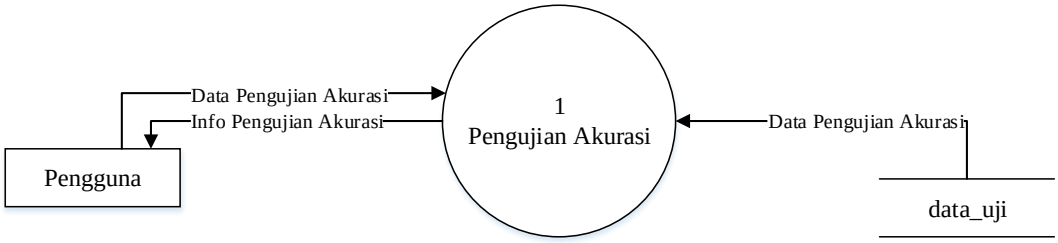
   Proses merupakan apa yang dikerjakan sistem, proses dapat mengelola data masukan menjadi informasi keluaran [14]. Proses berfungsi mengubah satu atau beberapa mata masukan menjadi satu atau beberapa keluaran informasi dengan spesifikasi yang ditentukan.

4. Simpanan Data (*Data Store*)

   Simpanan data merupakan tempat menyimpan data yang dibutuhkan oleh sistem, simpanan data dapat menerima atau mengirimkan data ke proses [14].

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 2.14 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 [14]. Penulisan logika proses dapat menggunakan bahasa deskriptif maupun *pseudocode*.

Tabel 2.8 Tabel Spesifikasi Proses

No.	Proses	Keterangan
1	No. Proses	1
	Nama Proses	Pengujian Akurasi
	Sumber	Pengguna
	Input	Data Pengujian Akurasi
	Output	Info Pengujian Akurasi
	Logika Proses	4. Pengguna menyimpan data hasil translasi beserta status translasinya benar atau salah. 5. Sistem akan menghitung nilai akurasi dari hasil translasi. 6. Sistem menampilkan nilai akurasi.

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 [14]. Contoh kamus data dapat dilihat pada Tabel 2.9.

Tabel 2.9 Kamus Data

1	Nama	Data pengujian akurasi
	Digunakan di	Proses Pengujian Akurasi (masukan)
	Deskripsi	Data pengujian akurasi = data uji + hasil harapan Data uji = teks Hasil harapan = teks

2.11. PHP (*Persinal Home Page*)

Secara umum PHP dikenal sebagai bahasa pemrograman script yang membuat dokumen HTML (*Hypertext Markup Language*) secara *on the fly* yang dieksekusi pada server web [15]. PHP merupakan salah satu aplikasi eksternal yang dijalankan oleh server web, sehingga server tidak hanya memberikan layanan dokumen HTML saja [15]. Sebuah web yang menggunakan PHP dapat menerima data dari luar kemudian memberi keluaran dari *database*.

PHP dikembangkan menjadi bahasa pemrograman *script* yang dijalankan di atas *platform* sistem operasi secara langsung [15]. PHP juga dapat digunakan dan dijalankan dari desktop [15], 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.

2.12. Perangkat Lunak Pendukung

Perangkat lunak pendukung adalah suatu kebutuhan perangkat yang digunakan dalam pembangunan sistem penerjemah algoritma deskriptif ke *source code*. Perangkat lunak pendukung yang digunakan dalam penelitian ini yaitu XAMPP.

2.12.1. XAMPP

XAMPP (X(*Sistem Operasi*) *Apache MySQL* PHP *Perl*) merupakan paket server web PHP dan *database MySQL* yang populer digunakan oleh para pengembang web yang menggunakan PHP dan *MySQL* sebagainya [15]. XAMPP dapat dipakai pada berbagai sistem operasi. XAMPP memiliki beberapa modul yang dimiliki, yaitu, *apache, MySQLl, FileZilla, Mercury,* dan *Tomcat*.

Dalam penelitian ini, modul yang digunakan adalah *Apache* dan *MySQL*. *Apcahe* digunakan sebagai server web penerjemah bahasa alami ke *source code*, dan *MySQL* digunakan sebagai penyimpanan data-data yang dibutuhkan oleh sistem, seperti *grammar*, kata kunci, dan sebagainya.