BAB 2
LANDASAN TEORI

2.1 Algoritma

Algoritma merupakan susunan langkah-langkah penyelesaian dalam suatu permasalahan secara sistematis dan logis [10]. Menurut Donald E. Knuth algoritma yang baik memiliki kriteria sebagai berikut [10]:

1. Input

   Dalam pembuatan algoritma harus memiliki 0 (nol) atau lebih masukan (input). Dalam hal ini dapat diartikan bahwa algoritma memungkinkan tidak memiliki data masukan sama sekali atau memiliki beberapa data masukan. Biasanya algoritma yang tidak memiliki data masukan secara langsung oleh pengguna, semua data pada algoritma dapat didefinisikan atau dibangkitkan dalam algoritma tersebut.
2. *Output*

   Dalam algoritma harus memiliki sedikitnya satu keluaran (*output*). Suatu algoritma yang tidak memiliki keluaran merupakan algoritma yang sia-sia, dan dalam hal ini algoritma ini tidak perlu dilakukan. Hal ini disebabkan algoritma dibuat agar dapat menghasilkan sesuatu yang diinginkan dalam bentuk hasil keluaran (*output*).
3. *Finiteness*

   Dalam menjalankan algoritma pasti akan berhenti. Oleh karena itu dalam membuat algoritma harus dapat menjamin bahwa algoritma tersebut dapat berhenti setelah menjalankan berbagai macam proses.
4. *Definitenes*

   Dalam membuat algoritma tidak menimbulkan makna ganda (ambigu). Setiap langkah dalam algoritma harus pasti agar tidak menimbulkan kesalahan penafsiran bagi yang membaca algoritma tersebut. Sehingga algoritma dapat memberikan outpu yang sesuai.

5. *Effectiveness*

   Langkah-langkah yang ada dalam algoritma dikerjakan dalam waktu yang wajar, artinya langkah-langkah tersebut tidak terdapat suatu aksi yang tidak perlu dilakukan karena setiap menjalankan langkah dalam algoritma memerlukan waktu eksekusi agar.

2.1.1 **Notasi Algoritmik**

Suatu algoritma dapat disajikan dalam tiga cara [10], yaitu:

1. Naratif

   Langkah-langkah penyelesaian masalah ditulis dalam bentuk kalimat seperti halnya menggunakan bahasa sehari-hari. Biasanya notasi ini dilakukan oleh orang-orang awam namun akan jika dibandingkan dengan notasi algoritmik yang lain, notasi algoritmik ini lebih sulit diterjemahkan kedalam bentuk *source code.*

2. *Flowchart*

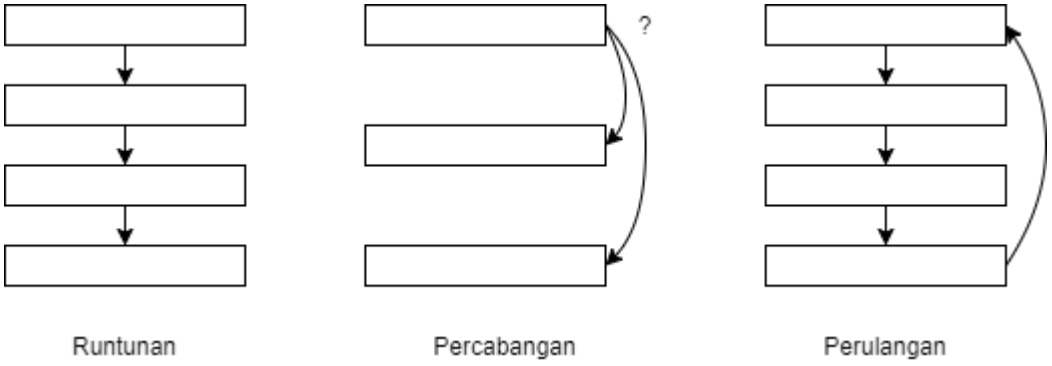
   Langkah-langkah penyelesaian masalah menggunakan *flowchart* setiap urutan langkah-langkahnya digambarkan dalam bentuk urutan simbol-simbol khusus. Urutan simbol-simbol tersebut dihubungkan menggunakan anak panah. Notasi algoritmik menggunakan *flowchart* lebih cocok digunakan untuk kasus yang kecil, karena jika digunakan dalam kasus yang besar akan menghabiskan banyak lembar kertas. Selain itu notasi algoritmik *flowchart* cukup sulit diterjemakah kedalam bentuk *source code.*

3. *Pseudocode*

   Langkah-langkah penyelesaian masalah menggunakan *pseudcode* dituliskan menyerupai pemrograman. *Pseucode* tidak spesifik terhadap bahasa pemrograman tertentu. Pada notasi algoritmik menggunakan *pseudcode* memiliki keuntungan yaitu mudah dalam menerjemahkan ke dalam *source code*.

2.1.2 **Struktur Dasar Algoritma**

Secara umum, algoritma terdiri dari tiga struktur dasar yaitu runtunan, percabangan dan perulangan [10].



Gambar 2.1 Struktur dasar algoritma

1. Runtunan

   Runtunan merupakan struktur yang paling dasar dalam algoritma. Dalam runtunan langkah-langkah dilakukan secara berurutan sesuai dengan urutan penulisannya dari awal sampai akhir.

2. Percabangan

   Percabangan adalah langkah pada algoritma yang akan dikerjakan atau tidak akan dikerjakan tergantung kondisi tertentu. Dalam percabangan ini hanya akan mengerjakan satu dari dua atau lebih pilihan yang tersedia.

3. Perulangan

   Perulangan adalah mengerjakan satu atau beberapa langkah secara berulanga sesuai dengan kebutuhan atau kondisi.

2.2 Bahasa Pemrograman *Pascal*

Pascal merupakan salah satu bahasa pemrograman tingkat tinggi (high level language) yang dibangun pada tahun 1968-1969 oleh Niklaus Wirth [11]. Bahasa Pemrograman *Pascal* pada masanya merupakan bahasa proseduran yang berpengaruh karena bahasa pemrograman *Pascal* relatif mudah dibaca, selain itu

notasinya juga mirip bahasa inggris standar. Bahkan bahasa pemrogramman *Pascal* hingga saat ini masih digunakan sebagai latihan pemrograman strukturan dan struktur data.

Bahasa pemrograman *Pascal* dibuat karena kesadaran Nuklaus Wirth terhadap mahasiswanya yang sulit beradaptasi dengan teknik pemrograman terstruktur dengan menggunakan Bahasa yang sudah ada sebelumnya seperti FOTRAN dan ALGOL. Bahasa pemrograman *Pascal* ini didesain secara sederhana dan efisien, berbeda dengan bahasa pemrograman ALGOL 68 yang juga dibangun pada tahun yang kuran lebih sama dengan *Pascal* [11].

Dalam bahasa pemrograman *Pascal*, pengenal ialah nama yang dapat diberikan pada suatu elemen program, dapat berupa konstanta, variabel, fungsi, 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

Natural language processing (pemrosesan bahasa alami) merupakan bagian dari 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* ini 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.

2.4 Grammar

Tata bahasa (*grammar*) merupakan sekumpulan dari himpunan-himpunan variabel, simbol-simbol terminal, simbol awal, yang dibatasi oleh aturan-aturan produksi [12]. Noam Chomsky Pada tahun 1959 telah melakukan penggolongan tingkatan bahasa menjadi empat tangka yang dapat dilihat pada Tabel 2.1.

Tabel 2.1 Penggolongan Tingkat Bahasa

Bahasa		Mesin Otomata	Batasan Aturan produksi
Reguler / Tipe 3		Finite State Automata (FSA) meliputi Deterministic Finite Automata (DFA) & Nondeterministic 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		Mesin Turing	Tidak ada batasan

Language / Tipe 0			

Aturan produksi merupakan pusat dari tata bahasa yang mengatur tata bahasa dalam melakukan perubahan suatu string ke bentuk lainnya, dan dengan 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). 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 kebalikannya yaitu simbol yang dapat diturunkan.

Pada penelitian ini, penulis melakukan penerjemahan dari bahasa alami ke bahasa bebas konteks (*contex free*). Bahasa alami atau bahasa manusia termasuk pada *grammar* tipe 0 (*unrestricted*), dimana tidak ada batasan untuk 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) [12]. 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

2.5 Case Folding

Case folding merupakan proses menyeragamkan huruf pada teks masukan menjadi huruf kecil (*lowercase*) atau huruf kapital (*uppercase*). Dalam penelitian ini penggunaan *case folding* bertujuan untuk mengubah semua huruf pada teks

masukan dalam Bahasa Indonesia menjadi huruf kecil. Contoh penerapan *case folding* dapat dilihat pada Tabel 2.3. Pada Tabel 2.3 terdapat data masukan dan hasil *case folding*, dimana data masukan adalah kalimat yang masih mengandung huruf kapital dan hasil *case folding* adalah kalimat yang sudah tidak mengandung huruf kapital.

Tabel 2.3 Contoh *case folding*

Data masukan	Hasil *case folding*
BUAT program uji2!. Buat variabel i dengan tipe data integer. Untuk iterasi 1 sampai 10 pada i maka tampilkan "benar"!!.	buat program uji2!. buat variabel i dengan tipe data integer. untuk iterasi 1 sampai 10 pada i maka tampilkan "benar"!!.

2.6 Filtering

Filtering merupakan proses penghapusan karakter yang tidak diperlukan dari data masukan [13]. *Filtering* biasanya dilakukan pada dokumen untuk menghapus beberapa stop word. Pada penelitian ini karakter yang diperbolehkan dalam penelitian ini yaitu ‘a’ sampai ‘z’, ‘0’ sampai ‘9’, koma (‘,’), titik (‘.’), garis bawah(‘_’), tambah(‘+’), kurang(‘-’), kali(‘*’), bagi(‘/’), dan spasi. Contoh *filtering* dapat dilihat pada Tabel 2.4. Pada Tabel 2.4 terdapat data masukan yang berupa kalimat tanpa huruf kapital dan hasil *filtering*. Pada kalimat data masukan masih terdapat karakter yang tidak diperbolehkan yaitu ‘!’, lalu proses *filtering* bertujuan untuk menghilangkan karakter tersebut sehingga hasil akhir berupa kalimat yang yang sudah tidak mengandung karakter yang tidak diperbolehkan.

Tabel 2.4 Contoh *Filtering*

Data masukan	Hasil *filtering*
buat program uji2!. buat variabel i dengan tipe data integer. untuk iterasi 1 sampai 10 pada i maka tampilkan "benar"!!.	buat program uji2. buat variabel i dengan tipe data integer. untuk iterasi 1 sampai 10 pada i maka tampilkan "benar".

2.7 Scanning

Scanning merupakan tahap pemecahan teks masukan menjadi token-token yang selanjutnya ditententukan kelasnya [12]. Pada penelitian ini, tahap *scanning* akan menerima kalimat masukan dalam Bahasa Indonesia yang selanjutnya dipencah kemudian memilahnya menjadi satuan leksik atau token. Token-token hasi dari *scanning* ini selanjutnya akan menjadi data masukan pada tahap *parsing* dan *translasi*. Besaran pembangunan leksik atau token meliputi hal-hal sebagai berikut [12].

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 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.

2.8 Parsing

Parsing merupakan tahapan pemeriksaan kesesuaian pola deretan token-token dengan aturan sintak yang telah ditentukan baik dalam bahasa alami maupun bahasa pemrograman *Pascal*. Sederet token yang tidak mengikuti aturan sintak yang telah ditentukan akan menghasislkan sintak *error* (kesalahan sintak). Secara logika deretan token yang bersesuaian dengan sintak tertentu akan dinyatakan sebagai pohon parsing (*parse tree*) [14]. 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. *Backtack/backup : Brute Force,*
   b. *No. backtrack : Recursive Descent Parser*

2. *Bottom Up*

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

2.9 Hypertext Prepocessor (PHP)

PHP (PHP *hypertext preprocessor*) merupakan bahasa pemrograman berbasis web berupa *script* yang dapat diintegrasikan dengan HTML (*Hypertext Markup Language).* PHP adalah *script* yang digunakan agar dapat membuat halaman website yang dinamis, yaitu website yang akan ditampilkan dibuat saat halaman itu diminta oleh *client.* Sehingga informasi yang diterima oleh *client* akan selalu mendapatkan data yang terbaru. Setiap *script* php akan dijalankan deserver lalu dikirimkan ke *client* menggunakan *web browser*. Pada penelitian ini bahasa pemrograman PHP digunakan kerena PHP merupakan sebuah perangkat lunak *open source* yang dapat dijalankan dibeberapa sistem operasi serta memiliki banyak dukungan [15]. PHP dalam penelitian ini digunakan untuk melakukan proses perubahan bahasa alami ke *source code.*

2.10 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 pada pohon sintaks 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 yang terletak paling kiri 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 → aAS | a

   A → SbA | ba

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

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

2.11 Unified Modeling Language (UML)

Unified Modeling Language (UML) merupakan salah satu standar bahasa yang banyak digunakan untuk mendefinisikan requirement, membuat analisis & desain, serta menggambarkan arsitektur dalam pemrograman berorientasi objek [16]. Dalam pengembangan perangkat lunak, UML digunakan untuk membuat model suatu sistem yang menggunakan konsep berorientasi objek agar lebih bisa dipahami oleh banyak pihak yang terlibat dalam pengembangan.

Pada UML terdapat beberapa diagram yang biasanya digunakan untuk memodelkan analisis fungsional dalam rangka pengembangan perangkat lunak berorientasi objek. Berikut merupakan beberapa diagram yang umum digunakan :

a) *Use Case Diagram*

   Menggambarkan sejumlah aktor eksternal dan hubungannya ke *use case* yang diberikan oleh sistem. *Use case* adalah deskripsi fungsi yang disediakan oleh sistem dalam bentuk teks sebagai dokumentasi dari *use case* symbol. Diagram *Use case* hanya menggambarkan yang dapat dilihat dari luar oleh aktor dan bukan menggambarkan apa yang ada didalam sistem.

b) *Activity Diagram*

   Menggambarkan rangkaian aliran dari aktivitas, digunakan untuk mendeskripsikan aktivitas yang dibentuk dalam suatu operasi. *Activity Diagram* dibuat sebanyak kasus yang digambarkan pada *use case diagram*.

c) *Class Diagram*

   Menggambarkan struktur statis class di dalam sistem. Class merepresentasikan sesuatu yang ditangani oleh sistem. Class dapat berhubungan dengan yang lain melalui berbagai cara: *associated* (terhubung satu sama lain), *dependent* (satu class tergantung/menggunakan class yang lain), *specialed* (satu class merupakan spesialisasi dari class lainnya), atau *package* (grup bersama sebagai satu unit). Didalam sebuah sistem biasanya mempunyai beberapa class diagram.

d) *Sequence Diagram*

   Menggambarkan kolaborasi dinamis antara sejumlah object. Kegunaanya untuk menunjukkan rangkaian pesan yang dikirim antara object juga interaksi antara object, sesuatu yang terjadi pada titik tertentu dalam eksekusi sistem.

2.12 Pengujian Akurasi

Pengujian yang dilakukan dalam penelitian ini menggunakan pengujian akurasi. Pengujian dilakukan untuk mengetahui nilai akurasi dari sistem yang dibangun. Pengujian dilakukan dengan cara membandingkan hasil translasi *source code* dari sistem dengan *source code* yang diharapkan. Pada penelitian ini rumus yang digunakan untuk pengujian akurasi sebagai berikut.

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

2.13 *Software* Pendukung

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

2.13.1 XAMPP

XAMPP adalah sebuah perangkat lunak gratis untuk menjalankan web server. XAMPP ini merupakan gabungan dari beberapa macam program seperti Apache, MySQL, PHP dan Perl. XAMPP sangat mudah digunakan bagi pemula, karena pada proses instalasinya tidak memerlukan konfigurasi Apache, PHP dan MySQL secara manual, melainkan melakukan instalasi dan konfigurasi secara otomatis. XAMPP dapat dipakai pada berbagai sistem operasi. XAMPP memiliki beberapa modul yang dimiliki, yaitu, apache, MySQLl, FileZilla, Mercury, dan Tomcat.

Dalam penelitian ini, modul XAMPP yang digunakan adalah Apache dan MySQL. Apache digunakan untuk menjalankan web server yang didalamnya terdapat proses penerjemahan Bahasa alami ke *source code*, sedangkan MySQL digunakan sebagai tempat penyimpanan data serperti data *grammar Pascal*, *grammar* bahasa alami, dan sebagainya.

2.13.2 **Codeigniter**

CodeIgniter adalah sebuah *Application Development Framework (toolkit)* bagi orang-orang yang ingin membangun website menggunakan PHP. Tujuan dari penggunaan *framework Codeingiter* adalah untuk memungkinkan para pengembang mengembangkan proyek-proyek lebih cepat daripada menulis kode dari awal. Pada *framework* ini tersedia banyak libary yang biasa diperlukan dalam pembuatan website, serta memiliki antarmuka dan struktur yang sederhana untuk mengakses library ini. *CodeIgniter* memungkinkan para pengembang untuk fokus pada proyek dengan meminimalkan jumlah kode yang dibutuhkan untuk tugas yang diberikan php [17].