ALGORITMA PEMROGRAMAN DAN STRUKTUR DATA

Haisyam Maulana, S.T., M.Kom.

ALGORITMA PEMROGRAMAN DAN STRUKTUR DATA

ALGORITMA PEMROGRAMAN DAN STRUKTUR DATA

Haisyam Maulana, S.T., M.Kom.

RC
Rumah Cemerlang

ALGORITMA PEMROGRAMAN DAN STRUKTUR DATA

Penulis:
Haisyam Maulana, S.T., M.Kom.

Editor: Rusli

Cetakan Pertama: Juli 2025

Cover: Tim Kreatif PRCI

Tata Letak: Tim Kreatif PRCI

Perkumpulan Rumah Cemerlang Indonesia
ANGGOTA IKAPI JAWA BARAT
Pondok Karisma Residence Jalan Raflesia VI D.151
Panglayungan, Cipedes Tasikmalaya – 085223186009

Website: www.rcipress.rcipublisher.org
E-mail: rumahcemerlangindonesia@gmail.com

- Cet. I –: Perkumpulan Rumah Cemerlang Indonesia, 2025
Dimensi : 21 x 29,7 cm

ISBN 978-634-239-125-9

Undang-undang No.19 Tahun 2002 Tentang Hak Cipta
Pasal 72

Barang siapa dengan sengaja melanggar dan tanpa hak melakukan perbuatan sebagaimana dimaksud dalam pasal ayat (1) atau pasal 49 ayat (1) dan ayat (2) dipidana dengan pidana penjara masing-masing paling sedikit 1 (satu) bulan dan/atau denda paling sedikit Rp.1.000.000,00 (satu juta rupiah), atau pidana penjara paling lama 7 (tujuh) tahun dan/atau denda paling banyak Rp.5.000.000.000,00 (lima miliar rupiah).

Barang siapa dengan sengaja menyiarkan, memamerkan, mengedarkan, atau menjual kepada umum suatu ciptaan atau barang hasil pelanggaran hak cipta terkait sebagai dimaksud pada ayat (1) dipidana dengan pidana penjara paling lama 5 (lima) tahun dan/atau denda paling banyak Rp.500.000.000,00 (lima ratus juta rupiah).

KATA PENGANTAR

Puji syukur kami panjatkan ke hadirat Tuhan Yang Maha Esa atas terbitnya buku **"Algoritma Pemrograman dan Struktur Data"** ini. Buku ini disusun untuk memberikan pemahaman yang sistematis, terstruktur, dan aplikatif mengenai dasar-dasar algoritma serta penerapannya dalam pemrograman dan pengelolaan struktur data, yang merupakan fondasi penting dalam dunia ilmu komputer dan rekayasa perangkat lunak.

Sebagai penerbit, kami menyadari pentingnya sumber belajar yang tidak hanya mengupas teori, tetapi juga menghadirkan pendekatan praktis, studi kasus, dan latihan soal yang relevan dengan kebutuhan pembelajaran di era digital saat ini. Oleh karena itu, kami mendukung penuh hadirnya buku ini sebagai referensi utama bagi mahasiswa, dosen, maupun praktisi yang ingin memperkuat keterampilan logika dan pemrograman secara mendalam.

Kami berharap buku ini dapat menjadi kontribusi nyata dalam pengembangan literasi digital dan penguatan kompetensi teknologi informasi di lingkungan akademik maupun industri. Ucapan terima kasih kami sampaikan kepada penulis atas dedikasi dan kerja kerasnya dalam menyusun buku ini, serta kepada semua pihak yang telah berperan dalam proses penerbitan.

Akhir kata, semoga buku ini dapat memberikan manfaat yang luas dan menjadi inspirasi bagi para pembaca untuk terus belajar, bereksperimen, dan berinovasi di bidang teknologi.

Penerbit

Rumah Cemerlang Indonesia

DAFTAR ISI

BAB 1

PENDAHULUAN ALGORITMA - APA ITU ALGORITMA DAN MENGAPA PENTING?

Selamat bergabung di dunia algoritma dan pemrograman! Mungkin Anda sempat bertanya-tanya, "Algoritma? Bukankah itu sesuatu yang rumit dan hanya dipahami oleh para ahli komputer?" Tenang saja, Anda tidak sendirian dalam pikiran itu. Banyak orang merasa demikian pada awalnya. Namun sebenarnya, algoritma bukanlah sesuatu yang eksklusif bagi para programmer. Ia justru sangat dekat dengan kehidupan kita sehari-hari. Mulai dari cara Anda membuat secangkir kopi, menyusun jadwal harian, hingga menentukan rute tercepat ke kampus—semuanya melibatkan algoritma secara tidak langsung.

Bayangkan algoritma sebagai resep masakan untuk komputer. Sama seperti resep yang memandu kita langkah demi langkah agar menghasilkan hidangan yang lezat, algoritma memberikan instruksi yang jelas dan terstruktur agar komputer dapat menyelesaikan tugas tertentu secara efisien. Tanpa algoritma, komputer hanyalah mesin kosong tanpa arah. Dengan algoritma, komputer bisa menjadi alat yang sangat bermanfaat untuk memecahkan masalah, membuat keputusan, dan menyederhanakan banyak aspek kehidupan kita.

1.1 Definisi Algoritma

Mari kita bayangkan sejenak. Ketika Anda ingin membuat secangkir kopi, apa saja langkah-langkah yang Anda lakukan?

1. Ambil cangkir.
2. Ambil bubuk kopi.
3. Masukkan bubuk kopi ke cangkir.
4. Panaskan air.
5. Tuangkan air panas ke cangkir.
6. Aduk.
7. Kopi siap diminum.

Tahukah Anda? Tanpa disadari, Anda baru saja menjalankan sebuah algoritma! Ya, algoritma bukanlah sesuatu yang hanya terjadi di dalam komputer—ia juga hadir dalam rutinitas harian kita. Dalam contoh di atas, setiap langkah disusun secara berurutan, logis, dan memiliki tujuan yang jelas: menghasilkan secangkir kopi yang nikmat.

Dalam dunia pemrograman, algoritma memiliki peran yang sangat penting. Algoritma merupakan **kumpulan instruksi yang terstruktur dan logis untuk menyelesaikan suatu masalah, memproses data, atau menghasilkan suatu informasi**. Agar dapat dieksekusi oleh komputer, langkah-langkah ini harus tepat, tidak ambigu, dan dapat diterapkan secara sistematis.

Tanpa algoritma, komputer hanyalah benda mati tanpa arah. Tidak peduli seberapa canggih perangkat kerasnya, ia tidak akan mampu melakukan apa pun tanpa petunjuk yang jelas. Algoritma-lah yang menjadi "otak" di balik kecerdasan mesin—memungkinkan komputer untuk berpikir, mengambil keputusan, dan menyelesaikan tugas dengan efisiensi tinggi.

Tapi dari mana sebenarnya istilah *algoritma* berasal? Untuk menjawabnya, kita perlu menengok ke belakang, ke abad ke-9 Masehi, ketika seorang cendekiawan Muslim asal Persia bernama **Muhammad ibn Musa al-Khwarizmi** menulis karya penting yang berjudul *Kitab al-Jabr wal-Muqabala*. Buku ini tidak hanya menjadi tonggak utama dalam perkembangan aljabar, tetapi juga menjadi dasar lahirnya konsep algoritma yang kita kenal sekarang.

Nama *al-Khwarizmi* kemudian diadaptasi ke dalam bahasa Latin menjadi *Algoritmi*. Dari sinilah kata *algorithm* muncul dan berkembang dalam literatur matematika dan komputasi di dunia Barat. Meski saat itu belum ada komputer, pemikiran al-Khwarizmi mengenai prosedur sistematis untuk menyelesaikan persoalan matematika sudah sangat mirip dengan konsep algoritma modern: sebuah proses langkah demi langkah yang pasti dan dapat diulangi untuk mencapai hasil tertentu.

Kontribusi al-Khwarizmi tidak hanya berpengaruh pada matematika, tetapi juga menjadi jembatan penting menuju lahirnya ilmu komputer. Gagasan tentang pemrosesan informasi secara sistematis inilah yang kelak menjadi pondasi bagi para ilmuwan komputer dalam merancang sistem, perangkat lunak, hingga kecerdasan buatan seperti yang kita kenal sekarang. Maka, bisa dikatakan bahwa setiap baris kode yang ditulis hari ini memiliki akar sejarah yang mengakar hingga ke zaman keemasan peradaban Islam.

Dengan memahami akar sejarah algoritma, kita tidak hanya belajar cara memprogram komputer, tetapi juga menghargai warisan intelektual yang telah membentuk dunia teknologi saat ini. Maka, saat Anda mulai menulis algoritma pertama Anda, ingatlah bahwa Anda sedang melanjutkan estafet pengetahuan yang telah berjalan selama berabad-abad—dan mungkin, seperti al-Khwarizmi, Anda pun bisa menciptakan sesuatu yang mengubah dunia.

Kini, setelah kita mengenal asal-usul dan sejarah algoritma, pertanyaannya adalah: apa peran algoritma dalam kehidupan kita sehari-hari saat ini? Ternyata, algoritma tidak hanya berperan dalam dunia pemrograman atau teknologi tinggi. Ia hadir hampir di setiap aspek kehidupan modern—mulai dari aplikasi yang merekomendasikan lagu favorit, sistem navigasi yang menunjukkan rute tercepat, hingga media sosial yang menampilkan konten yang relevan dengan minat Anda. Tanpa kita sadari, algoritma bekerja di balik layar, membantu kita membuat keputusan, menyederhanakan proses, dan meningkatkan efisiensi dalam berbagai aktivitas harian.

1.2 Peran Algoritma dalam Pemrograman, Kehidupan Sehari-hari, Konservasi dan Budaya

Anda mungkin bertanya, "Oke, saya paham apa itu algoritma. Tapi kenapa ini penting, apalagi jika dikaitkan dengan konservasi dan budaya?" Jawabannya sederhana: **Algoritma adalah tulang punggung setiap solusi digital.** Baik itu aplikasi di ponsel Anda, situs web yang Anda kunjungi, bahkan sistem pintar yang memantau hutan, semuanya bekerja berdasarkan algoritma.

Dalam era digital ini, masalah yang kita hadapi semakin kompleks. Dari mengelola data populasi hewan langka yang tersebar di berbagai lokasi, mendigitalisasi jutaan naskah kuno, hingga memprediksi pola cuaca untuk mitigasi bencana alam dan budaya, semuanya membutuhkan pendekatan sistematis. Di sinilah algoritma berperan sangat krusial.

Dalam Kehidupan Sehari-hari:

- Saat Anda mencari rute terbaik di aplikasi peta, ada algoritma yang menghitung jalur tercepat.
- Ketika Anda memilih film rekomendasi di layanan *streaming*, ada algoritma yang mempelajari preferensi Anda.
- Bahkan saat Anda mencuci baju, mesin cuci mengikuti algoritma yang telah diprogram (isi air, putar, buang air, bilas, putar lagi).

Dalam Pemrograman:

Algoritma adalah "otak" di balik setiap program. Seorang programmer merancang algoritma untuk:

- **Memproses Data:** Seperti menghitung total karbon yang terserap oleh hutan, atau mengidentifikasi jumlah artefak yang ditemukan di suatu situs.
- **Mengambil Keputusan Otomatis:** Misalnya, sistem otomatis yang memberi peringatan jika suhu di ruang penyimpanan koleksi museum terlalu tinggi.
- **Mengurutkan dan Mencari Informasi:** Memudahkan pencarian nama spesies dalam database konservasi atau mencari resep makanan tradisional.
- **Melakukan Simulasi:** Seperti mensimulasikan pertumbuhan populasi hewan atau dampak perubahan iklim pada suatu ekosistem.

Peran Algoritma dalam Konservasi dan Budaya:

1. **Efisiensi dalam Pengelolaan Data:** Bayangkan jika kita memiliki data ribuan spesies tumbuhan endemik atau ratusan ribu motif batik tradisional. Tanpa algoritma yang baik, mencari informasi spesifik akan sangat sulit dan memakan waktu. Algoritma membantu kita menyimpan, mencari, dan mengolah data ini secara efisien.
2. **Pemantauan dan Analisis Cerdas:** Algoritma memungkinkan sistem untuk secara otomatis **menganalisis citra satelit untuk mendeteksi deforestasi**, memantau **suhu dan kelembapan di museum untuk menjaga artefak**, atau bahkan **mengenali spesies hewan dari rekaman suara**. Ini semua dilakukan oleh algoritma yang diprogram untuk mengidentifikasi pola dan anomali.
3. **Digitalisasi dan Akses Informasi:** Algoritma adalah kunci dalam proses **digitalisasi manuskrip kuno, rekaman cerita rakyat, atau musik tradisional**. Dengan algoritma, data ini bisa diatur, diindeks, dan disajikan dengan cara yang mudah diakses oleh siapa pun di seluruh dunia, memastikan warisan ini tidak punah.

1.3 Konsep Pemecahan Masalah dengan Algoritma

Pada dasarnya, pemrograman adalah seni memecahkan masalah. Algoritma adalah alat kita untuk melakukan itu. Proses pemecahan masalah dengan algoritma biasanya melibatkan langkah-langkah berikut:

1. **Memahami Masalah:** Apa sebenarnya yang ingin kita selesaikan? Dalam konteks konservasi, masalahnya bisa jadi "Bagaimana cara mendeteksi lokasi kebakaran hutan secara cepat?" atau dalam budaya "Bagaimana kita bisa membuat kamus digital untuk bahasa daerah yang hampir punah?". Memahami masalah dengan jelas adalah langkah pertama dan terpenting.
2. **Merancang Solusi (Algoritma):** Setelah masalah dipahami, kita mulai berpikir tentang langkah-langkah logis untuk menyelesaikannya. Ini adalah inti dari perancangan algoritma. Misalnya, untuk masalah kebakaran hutan, solusinya mungkin melibatkan:
    - Mengumpulkan data citra satelit.
    - Menganalisis perbedaan suhu atau anomali asap.
    - Jika terdeteksi anomali, periksa kembali dengan data sensor lain.
    - Jika terkonfirmasi, kirimkan notifikasi ke pihak terkait dengan koordinat lokasi.
3. **Mengekspresikan Algoritma:** Algoritma dapat diekspresikan dalam berbagai bentuk:
    - **Bahasa Natural (seperti yang kita gunakan sehari-hari):** Cocok untuk penjelasan awal, tapi seringkali ambigu.
    - **Pseudocode:** Mirip bahasa pemrograman tapi tidak terikat pada aturan sintaksis tertentu, sangat bagus untuk merencanakan langkah-langkah secara semi-formal.
    - **Flowchart:** Diagram visual yang menggunakan simbol-simbol standar untuk merepresentasikan langkah-langkah dan alur logika.
    - **Bahasa Pemrograman:** Implementasi nyata algoritma dalam kode yang bisa dijalankan komputer (misalnya Python, Java, C++).
4. **Mengimplementasikan dan Menguji:** Setelah algoritma dirancang, kita akan menulisnya dalam bahasa pemrograman dan mengujinya. Apakah hasilnya sesuai harapan? Apakah ada kesalahan?
5. **Mengevaluasi dan Menyempurnakan:** Algoritma yang baik harus efisien. Bisakah kita membuat prosesnya lebih cepat atau menggunakan lebih sedikit sumber daya? Misalnya, apakah algoritma untuk memantau populasi burung migran bekerja dengan cepat ataukah terlalu lambat saat data yang diproses sangat besar?

Singkatnya, **algoritma adalah jembatan antara masalah yang kita hadapi dan solusi yang bisa disediakan oleh teknologi.** Dengan mempelajari algoritma, Anda tidak hanya belajar "cara memecahkan masalah", tetapi juga "cara berpikir secara sistematis" yang sangat berharga tidak hanya dalam pemrograman, tetapi juga dalam kehidupan nyata, termasuk dalam upaya mulia melestarikan alam dan budaya kita.

1.4 Implementasi Bahasa Pemrograman (Java & Python)

Untuk memberikan gambaran nyata bagaimana algoritma bekerja dalam kode, mari kita lihat implementasi algoritma "Membuat Secangkir Kopi" yang kita bahas di awal bab ini dengan bahasa pemrograman. Meskipun ini sederhana, ia menunjukkan urutan langkah-langkah yang jelas.

1.4.1 Java

Buat file dengan nama file: BuatKopi.java

```java
public class BuatKopi {
    public static void main(String[] args) {
        // Ini adalah algoritma untuk membuat secangkir kopi
        System.out.println("Memulai proses pembuatan kopi...");
        // Langkah 1: Ambil cangkir
        System.out.println("1. Mengambil cangkir.");
        // Langkah 2: Ambil bubuk kopi
        System.out.println("2. Mengambil bubuk kopi.");
        // Langkah 3: Masukkan bubuk kopi ke cangkir
        System.out.println("3. Memasukkan bubuk kopi ke cangkir.");
        // Langkah 4: Panaskan air
        System.out.println("4. Memanaskan air.");
        // Langkah 5: Tuangkan air panas ke cangkir
        System.out.println("5. Menuangkan air panas ke cangkir.");
        // Langkah 6: Aduk
        System.out.println("6. Mengaduk kopi.");
        // Langkah 7: Kopi siap diminum
        System.out.println("7. Kopi siap dinikmati!");
        // Langkah 8: Selesai
        System.out.println("Proses pembuatan kopi selesai.");
    }
}

```

Penjelasan Kode Java:

- `public class BuatKopi { ... }`: Ini mendefinisikan sebuah **kelas** di Java bernama `BuatKopi`. Dalam Java, semua kode biasanya berada di dalam kelas.
- `public static void main(String[] args) { ... }`: Ini adalah **metode utama** (main method) yang menjadi titik awal eksekusi program Java. Ketika Anda menjalankan program `BuatKopi`, kode di dalam metode inilah yang akan dieksekusi pertama kali.
    - `public static void`: Kata kunci ini menunjukkan bahwa metode ini bisa diakses secara publik, statis (bisa dipanggil tanpa membuat objek), dan tidak mengembalikan nilai (`void`).

    - String[] args: Ini adalah parameter untuk menerima argumen baris perintah, yang tidak kita gunakan di contoh ini.
- System.out.println("...");: Ini adalah perintah di Java untuk **menampilkan teks** ke konsol (layar). Setiap baris System.out.println menjalankan satu instruksi dalam algoritma kita.
- **Urutan Eksekusi:** Perhatikan bahwa setiap baris kode dieksekusi secara **berurutan** dari atas ke bawah, persis seperti langkah-langkah resep kopi. Ini adalah contoh paling dasar dari alur sekuensial dalam algoritma.

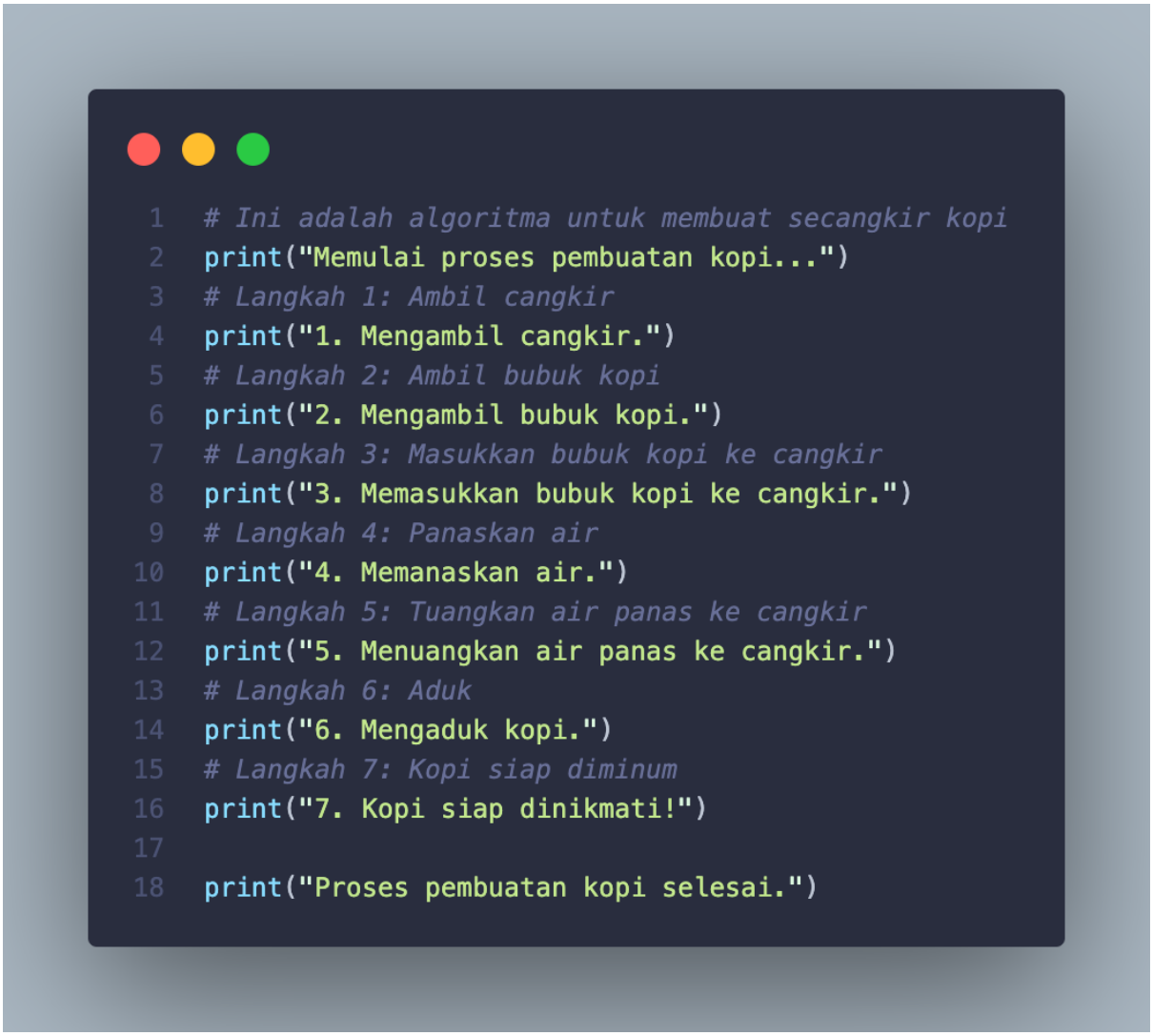
Output yang diharapkan (Java):

Output:

```
Memulai proses pembuatan kopi...
1. Mengambil cangkir.
2. Mengambil bubuk kopi.
3. Memasukkan bubuk kopi ke cangkir.
4. Memanaskan air.
5. Menuangkan air panas ke cangkir.
6. Mengaduk kopi.
7. Kopi siap dinikmati!
Proses pembuatan kopi selesai.
```

1.4.2 Python

Buat file dengan nama file: buat_kopi.py



```python
# Ini adalah algoritma untuk membuat secangkir kopi
print("Memulai proses pembuatan kopi...")
# Langkah 1: Ambil cangkir
print("1. Mengambil cangkir.")
# Langkah 2: Ambil bubuk kopi
print("2. Mengambil bubuk kopi.")
# Langkah 3: Masukkan bubuk kopi ke cangkir
print("3. Memasukkan bubuk kopi ke cangkir.")
# Langkah 4: Panaskan air
print("4. Memanaskan air.")
# Langkah 5: Tuangkan air panas ke cangkir
print("5. Menuangkan air panas ke cangkir.")
# Langkah 6: Aduk
print("6. Mengaduk kopi.")
# Langkah 7: Kopi siap diminum
print("7. Kopi siap dinikmati!")

print("Proses pembuatan kopi selesai.")
```

Penjelasan Kode Python:

- Nama file: buat_kopi.py: Baris yang diawali dengan # adalah **komentar**. Komentar digunakan untuk memberikan penjelasan dalam kode dan diabaikan oleh interpreter Python.
- print("..."): Ini adalah fungsi bawaan di Python yang digunakan untuk **menampilkan teks** ke konsol (layar). Sama seperti System.out.println di Java.
- **Tidak Ada main method atau class:** Berbeda dengan Java, Python tidak mengharuskan semua kode berada di dalam kelas atau metode main untuk program sederhana. Anda bisa langsung menulis instruksi dan Python akan menjalankannya dari atas ke bawah.
- **Urutan Eksekusi:** Setiap baris print dieksekusi secara **berurutan** dari atas ke bawah, menunjukkan alur sekuensial dari algoritma.

Output yang diharapkan (Python):

```
Memulai proses pembuatan kopi...
1. Mengambil cangkir.
2. Mengambil bubuk kopi.
3. Memasukkan bubuk kopi ke cangkir.
4. Memanaskan air.
5. Menuangkan air panas ke cangkir.
6. Mengaduk kopi.
7. Kopi siap dinikmati!
Proses pembuatan kopi selesai.
```

Melalui contoh sederhana tadi, Anda telah mulai memahami bagaimana sebuah "resep" atau algoritma dasar dapat diterjemahkan menjadi instruksi yang logis dan sistematis—yang pada akhirnya bisa dipahami dan dijalankan oleh komputer melalui bahasa pemrograman. Inilah inti dari pemrograman: menerjemahkan pemikiran manusia ke dalam bentuk instruksi yang bisa dimengerti oleh mesin.

Namun, itu baru permulaan. Di bab-bab selanjutnya, Anda akan dibimbing untuk mengeksplorasi algoritma yang lebih kompleks dan aplikatif, termasuk contoh-contoh nyata yang berkaitan dengan konservasi budaya dan pelestarian nilai-nilai lokal. Kita akan melihat bagaimana teknologi dan algoritma bisa dimanfaatkan untuk mendukung pelestarian warisan budaya, mulai dari pengelolaan data cagar budaya hingga sistem pendukung keputusan untuk pelestarian situs sejarah.

Jangan khawatir jika semuanya tampak baru dan menantang. Anda tidak sendiri—materi ini dirancang secara bertahap, dengan pendekatan yang mudah dipahami dan dilengkapi contoh-contoh praktis. Tetap ikuti alur pembelajaran, dan luangkan waktu untuk memahami setiap konsep sebelum melanjutkan ke topik berikutnya.

Dengan semangat belajar dan rasa ingin tahu, Anda akan menyadari bahwa algoritma bukanlah sesuatu yang sulit atau menakutkan, melainkan alat yang luar biasa kuat—terutama saat digunakan untuk sesuatu yang bermakna, seperti melestarikan kebudayaan bangsa. Mari lanjutkan perjalanan kita di bab berikutnya!

BAB 2

STRUKTUR DASAR ALGORITMA DAN NOTASI ALGORITMA – MEMBANGUN FONDASI LOGIKA

Setelah memahami apa itu algoritma dan mengapa ia sangat penting, terutama dalam konteks pelestarian alam dan budaya, kini saatnya kita belajar bagaimana "merancang" algoritma itu sendiri. Sama seperti arsitek yang membuat denah bangunan sebelum membangunnya, seorang *programmer* perlu merancang struktur algoritmanya sebelum menulis kode program. Bab ini akan membahas struktur dasar yang menjadi fondasi setiap algoritma dan berbagai cara untuk menuliskannya agar mudah dipahami, baik oleh manusia maupun komputer.

2.1 Struktur Algoritma Dasar

Setiap algoritma, tidak peduli seberapa kompleksnya, dibangun dari tiga struktur dasar utama. Ketiga struktur ini bagaikan "balok-balok bangunan" yang dapat kita susun untuk menyelesaikan berbagai masalah.

2.1.1. Struktur Sekuensial (Urutan)

Ini adalah struktur paling sederhana, di mana instruksi-instruksi dieksekusi secara berurutan, satu demi satu, dari awal hingga akhir. Tidak ada lompatan atau pengulangan; setiap langkah dijalankan tepat sekali.

Analogi Konservasi/Budaya:

Bayangkan langkah-langkah untuk mengumpulkan data spesies langka di lapangan.

1. Siapkan peralatan pengamatan.
2. Pergi ke lokasi pengamatan.
3. Cari jejak atau tanda keberadaan spesies.
4. Catat temuan (jumlah, lokasi, kondisi).
5. Kembali ke basecamp.
   Ini adalah urutan langkah yang harus dilakukan secara sekuensial.

Contoh Umum: Resep kopi yang kita bahas di Bab 1 adalah contoh klasik dari struktur sekuensial. Setiap langkah harus diikuti berurutan untuk mendapatkan hasil akhir.

2.1.2. Struktur Pemilihan (Percabangan/Seleksi)

Struktur ini memungkinkan algoritma untuk **membuat keputusan** berdasarkan suatu kondisi. Jika kondisi terpenuhi, satu blok instruksi akan dijalankan; jika tidak, blok instruksi lain (atau tidak ada sama sekali) yang akan dijalankan. Ini adalah inti dari "jika-maka-lainnya" (*if-then-else*) dalam logika berpikir kita.

Analogi Konservasi/Budaya:

Pertimbangkan sistem peringatan dini untuk situs budaya yang rentan bencana.

- **Jika** (kondisi) **tinggi muka air sungai melebihi batas aman**,
    - **Maka** (aksi jika benar) **kirim notifikasi darurat ke tim penyelamat dan evakuasi artefak penting**.
- **Lainnya** (aksi jika salah),
    - **Maka** (aksi jika salah) **terus pantau kondisi air sungai**.

Dalam kasus ini, algoritma "memilih" jalur eksekusi berdasarkan data yang diterima dari sensor.

2.1.3. Struktur Pengulangan (Perulangan/Iterasi)

Struktur pengulangan memungkinkan algoritma untuk **mengulang satu blok instruksi berulang kali** selama suatu kondisi terpenuhi atau sejumlah kali tertentu. Ini sangat efisien untuk tugas-tugas yang repetitif.

2.1.4. Analogi Konservasi/Budaya

Bayangkan Anda perlu mendigitalisasi seluruh koleksi naskah kuno di sebuah perpustakaan kerajaan.

- **Untuk setiap** (perulangan) **naskah dalam koleksi**,
    - **Lakukan:**
        1. Pindai naskah.
        2. Lakukan OCR (Optical Character Recognition) untuk mengubah gambar teks menjadi teks digital.
        3. Simpan hasil digitalisasi ke *database*.
- **Lanjutkan** sampai semua naskah selesai diproses.

Contoh lain: **memantau suhu harian di hutan konservasi selama setahun penuh**. Algoritma akan "mengulang" proses pembacaan sensor suhu dan pencatatan data setiap hari selama 365 hari.

Ketiga struktur dasar ini adalah fondasi dari setiap algoritma yang akan Anda temui. Kemampuan untuk mengkombinasikan dan menyusunnya dengan benar adalah kunci untuk memecahkan masalah komputasi yang kompleks.

2.2 Notasi Algoritma

Setelah memahami struktur dasarnya, bagaimana kita menuliskannya agar orang lain (dan kita sendiri di kemudian hari) dapat memahami algoritma yang telah kita rancang? Ada beberapa cara atau **notasi** yang umum digunakan:

2.2.1. Pseudocode

Pseudocode (dibaca: "sudo-kode") adalah cara menulis algoritma yang **mirip dengan bahasa pemrograman** tetapi **tidak terikat pada aturan sintaksis (tata bahasa) yang ketat** dari bahasa pemrograman tertentu. Kata "pseudo" berarti "semu" atau "mirip", jadi pseudocode adalah "kode semu". Tujuannya adalah untuk menggambarkan logika algoritma dengan jelas dan ringkas, sehingga mudah dibaca oleh manusia.

Kelebihan:

- Mudah ditulis dan dipahami.
- Tidak terikat pada bahasa pemrograman spesifik.
- Fokus pada logika, bukan detail sintaksis.

Contoh (untuk Analogi Konservasi): Prosedur Penanaman Bibit Mangrove

```
ALGORITMA PenanamanBibitMangrove
  INPUT: JumlahBibit, LokasiPenanaman

  MULAI
    UNTUK setiap Bibit dari 1 sampai JumlahBibit LAKUKAN
      CariAreaCocokDi(LokasiPenanaman)
      GaliLubang
      MasukkanBibit
      TutupLubangDenganTanah
      SiramBibit
    SELESAI UNTUK

    JIKA SemuaBibitTertanamDengnBaik MAKA
      CatatDataPenanamanKeDatabase("Sukses")
      KIRIM_PESAN("Penanaman mangrove berhasil!")
    LAINNYA
      CatatDataPenanamanKeDatabase("Gagal sebagian")
      KIRIM_PESAN("Ada masalah dalam penanaman. Mohon cek ulang.")
    SELESAI JIKA
  AKHIR
```

Dalam contoh ini, kita bisa melihat kata kunci seperti `UNTUK...LAKUKAN` (perulangan) dan `JIKA...MAKA...LAINNYA` (pemilihan), yang mirip dengan perintah dalam bahasa pemrograman, namun lebih fleksibel.

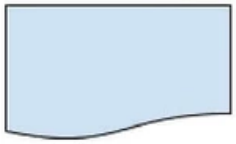
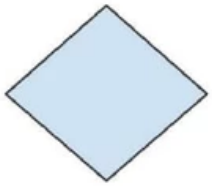
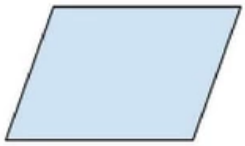
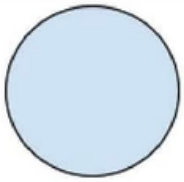
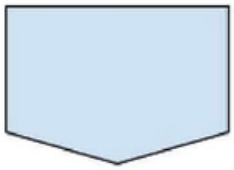
2.2.2. Flowchart (Diagram Alir)

Flowchart adalah **representasi visual (gambar)** dari sebuah algoritma. Ia menggunakan simbol-simbol standar untuk merepresentasikan berbagai jenis langkah dan garis penghubung untuk menunjukkan alur eksekusi algoritma. Flowchart sangat baik untuk menggambarkan alur logika yang kompleks secara intuitif.

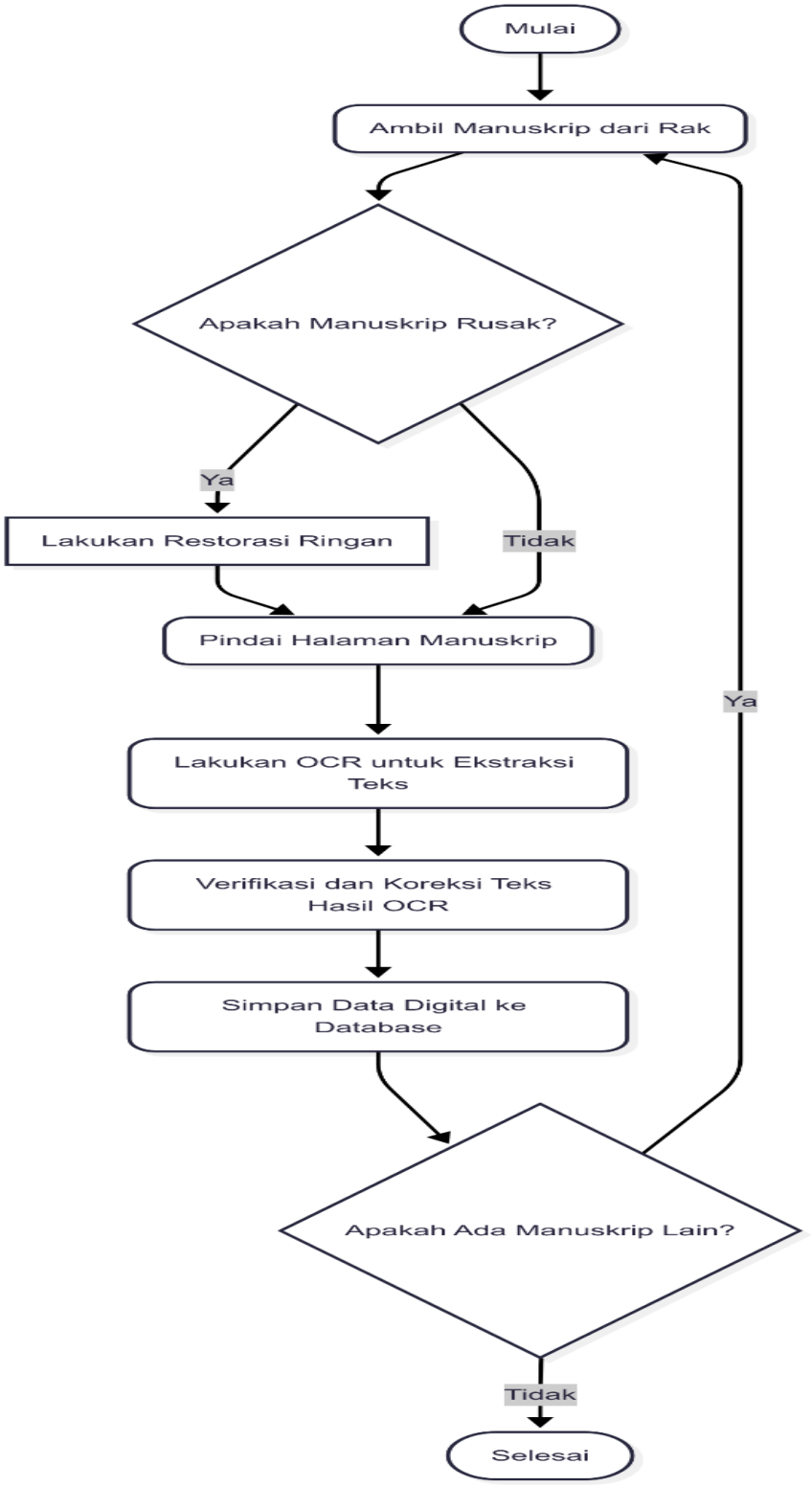
Kelebihan:

- Visual dan mudah dipahami alurnya.
- Baik untuk melacak alur program dan keputusan.
- Baku dan standar (ada simbol-simbol khusus).

Simbol-simbol Dasar Flowchart:

No.	Simbol Flowchart	Nama	Arti Simbol Flowchart
1		*Terminator*	Awal atau akhir konsep (prosedur)
2		*Process*	Proses operasional
3		*Document*	Dokumen atau laporan berupa *print out*
4		*Decision*	Keputusan atau sub-point. Garis yang terhubung dengan bentuk *decision* merujuk pada situasi-situasi yang berbeda sesuai dengan keputusan yang digambarkan
5		Data	Input dan Output (Contohnya, Input: feedback dari pelanggan. Output: desain produk baru)
6		*On-Page Reference/ Connector*	Penghubung alur dalam halaman yang sama
7		*Off-Page Reference/ Off-Page Connector*	Penghubung alur dalam halaman yang berbeda
8		*Flow*	Arah alur dalam konsep (prosedur)

Contoh Flowchart (untuk Analogi Budaya): Proses Digitalisasi Manuskrip Kuno Sederhana



Flowchart di atas secara visual menunjukkan langkah-langkah, keputusan (apakah manuskrip rusak), dan perulangan (proses terus berlanjut jika masih ada manuskrip lain).

2.3 Implementasi Bahasa Pemrograman(Java & Python)

Mari kita wujudkan struktur dasar algoritma ke dalam kode Java dan Python. Kita akan membuat program sederhana yang mensimulasikan proses **penanaman bibit mangrove**, yang akan mencakup struktur **sekuensial**, **percabangan**, dan **perulangan**.

2.3.1. Skenario: Simulasi Penanaman Bibit Mangrove

Program akan:

1. Meminta pengguna memasukkan jumlah bibit yang akan ditanam. (Input - Sekuensial)
2. Mengulang proses penanaman untuk setiap bibit. (Perulangan)
3. Setelah semua bibit ditanam, program akan mengecek apakah jumlah bibit yang ditanam lebih dari batas minimum untuk dianggap sukses. (Percabangan)
4. Menampilkan pesan keberhasilan atau kegagalan. (Output - Sekuensial)

2.3.2. Java

```java
// Nama file: PenanamanMangrove.java
import java.util.Scanner; // Digunakan untuk membaca input dari pengguna

public class PenanamanMangrove {

    public static void main(String[] args) {
        // Objek Scanner untuk membaca input
        Scanner inputPengguna = new Scanner(System.in);

        // --- Struktur Sekuensial: Meminta Input ---
        System.out.println("=== Program Simulasi Penanaman Mangrove ===");
        System.out.print("Masukkan jumlah bibit mangrove yang akan ditanam: ");
        int jumlahBibit = inputPengguna.nextInt(); // Membaca jumlah bibit

        int bibitTertanamSukses = 0; // Inisialisasi counter bibit sukses

        // --- Struktur Perulangan: Proses Penanaman Setiap Bibit ---
        System.out.println("\nMemulai proses penanaman...");
        for (int i = 1; i <= jumlahBibit; i++) {
            System.out.println("  Bibit ke-" + i + ":");
            System.out.println("    - Menggali lubang.");
            System.out.println("    - Memasukkan bibit.");
            System.out.println("    - Menutup lubang dengan tanah.");
            System.out.println("    - Menyiram bibit.");
            bibitTertanamSukses++; // Menambah jumlah bibit yang berhasil ditanam
            // Kita asumsikan setiap penanaman dalam loop ini sukses untuk kesederhanaan
        }
        System.out.println("Proses penanaman " + jumlahBibit + " bibit selesai.");

        // --- Struktur Percabangan: Mengecek Hasil Penanaman ---
        int batasMinimumSukses = 5; // Contoh batas minimum bibit agar dianggap sukses

        System.out.println("\n--- Evaluasi Hasil Penanaman ---");
        if (bibitTertanamSukses >= batasMinimumSukses) {
            System.out.println("Selamat! " + bibitTertanamSukses + " bibit berhasil ditanam.");
            System.out.println("Upaya konservasi mangrove Anda sangat baik!");
        } else {
            System.out.println("Perhatian: Hanya " + bibitTertanamSukses + " bibit yang berhasil ditanam.");
            System.out.println("Mohon periksa kembali metode penanaman atau lokasi.");
        }

        // Menutup objek Scanner
        inputPengguna.close();
    }
}
```

Penjelasan Kode Java:

- import java.util.Scanner;: Baris ini mengimpor kelas Scanner yang dibutuhkan untuk membaca input dari pengguna (seperti angka atau teks).
- Scanner inputPengguna = new Scanner(System.in);: Membuat objek Scanner baru yang akan membaca input dari konsol (System.in).
- System.out.print(...) dan System.out.println(...): Digunakan untuk menampilkan pesan ke konsol. print tidak menambahkan baris baru setelahnya, sedangkan println menambahkannya.
- int jumlahBibit = inputPengguna.nextInt();: Ini adalah contoh **input** sekuensial. Program menunggu pengguna mengetikkan angka dan menekan Enter, lalu nilai tersebut disimpan dalam variabel jumlahBibit.
- for (int i = 1; i <= jumlahBibit; i++) { ... }: Ini adalah contoh **struktur perulangan (for loop)**.
    - int i = 1: Mendeklarasikan dan menginisialisasi variabel i (sering disebut *counter* atau indeks) dengan nilai 1.
    - i <= jumlahBibit: Ini adalah **kondisi** perulangan. Selama i kurang dari atau sama dengan jumlahBibit, blok kode di dalam *loop* akan dieksekusi.
    - i++: Setelah setiap iterasi, nilai i akan bertambah 1.
    - Setiap baris System.out.println di dalam *loop* merupakan instruksi sekuensial yang diulang.
- if (bibitTertanamSukses >= batasMinimumSukses) { ... } else { ... }: Ini adalah contoh **struktur pemilihan (if-else)**.
    - Program mengevaluasi kondisi bibitTertanamSukses >= batasMinimumSukses.
    - Jika kondisi BENAR (jumlah bibit sukses lebih dari atau sama dengan batas minimum), blok kode di dalam if akan dieksekusi.
    - Jika kondisi SALAH, blok kode di dalam else yang akan dieksekusi.
- inputPengguna.close();: Penting untuk menutup objek Scanner setelah selesai menggunakannya untuk mencegah *resource leak*.

Output yang Diharapkan (Java - Contoh Input 7):

```
=== Program Simulasi Penanaman Mangrove ===
Masukkan jumlah bibit mangrove yang akan ditanam: 3

Memulai proses penanaman...
  Bibit ke-1:
    - Menggali lubang.
    - Memasukkan bibit.
    - Menutup lubang dengan tanah.
    - Menyiram bibit.
  Bibit ke-2:
    - Menggali lubang.
    - Memasukkan bibit.
    - Menutup lubang dengan tanah.
    - Menyiram bibit.
  Bibit ke-3:
    - Menggali lubang.
    - Memasukkan bibit.
    - Menutup lubang dengan tanah.
    - Menyiram bibit.
Proses penanaman 3 bibit selesai.

--- Evaluasi Hasil Penanaman ---
Perhatian: Hanya 3 bibit yang berhasil ditanam.
Mohon periksa kembali metode penanaman atau lokasi.
```

2.3.3. Python

```python
# Nama file: penanaman_mangrove.py

# --- Struktur Sekuensial: Meminta Input ---
print("=== Program Simulasi Penanaman Mangrove ===")
try: # Menggunakan try-except untuk menangani error jika input bukan angka
    jumlah_bibit = int(input("Masukkan jumlah bibit mangrove yang akan ditanam: "))
except ValueError:
    print("Input tidak valid. Harap masukkan angka.")
    exit() # Keluar dari program jika input tidak valid

bibit_tertanam_sukses = 0 # Inisialisasi counter bibit sukses

# --- Struktur Perulangan: Proses Penanaman Setiap Bibit ---
print("\nMemulai proses penanaman...")
for i in range(1, jumlah_bibit + 1): # Perulangan dari 1 hingga jumlah_bibit
    print(f"  Bibit ke-{i}:")
    print("    - Menggali lubang.")
    print("    - Memasukkan bibit.")
    print("    - Menutup lubang dengan tanah.")
    print("    - Menyiram bibit.")
    bibit_tertanam_sukses += 1 # Menambah jumlah bibit yang berhasil ditanam
    # Kita asumsikan setiap penanaman dalam loop ini sukses untuk kesederhanaan
print(f"Proses penanaman {jumlah_bibit} bibit selesai.")

# --- Struktur Percabangan: Mengecek Hasil Penanaman ---
batas_minimum_sukses = 5 # Contoh batas minimum bibit agar dianggap sukses

print("\n--- Evaluasi Hasil Penanaman ---")
if bibit_tertanam_sukses >= batas_minimum_sukses:
    print(f"Selamat! {bibit_tertanam_sukses} bibit berhasil ditanam.")
    print("Upaya konservasi mangrove Anda sangat baik!")
else:
    print(f"Perhatian: Hanya {bibit_tertanam_sukses} bibit yang berhasil ditanam.")
    print("Mohon periksa kembali metode penanaman atau lokasi.")
```

Penjelasan Kode Python:

- `print(...)`: Fungsi dasar untuk menampilkan teks ke konsol.
- `input("...")`: Fungsi bawaan untuk membaca input dari pengguna. Teks di dalam tanda kurung adalah prompt yang akan ditampilkan kepada pengguna. Input yang dibaca selalu berupa string.
- `int(input(...))`: Mengubah input string dari pengguna menjadi **bilangan bulat (integer)**. Jika pengguna memasukkan teks non-angka, akan terjadi *error*.
- `try: ... except ValueError: ...`: Ini adalah blok penanganan *error* (disebut *exception handling*). Jika kode di dalam `try` (yaitu `int(input(...))`) menghasilkan `ValueError` (karena input bukan angka), maka blok `except` akan dijalankan, menampilkan pesan *error* yang ramah pengguna.
- `for i in range(1, jumlah_bibit + 1):`: Ini adalah contoh **struktur perulangan (for loop)**.
    - `range(1, jumlah_bibit + 1)`: Fungsi `range` menghasilkan urutan angka. `range(start, stop)` akan menghasilkan angka dari `start` hingga `stop-1`. Jadi, untuk mendapatkan angka dari 1 sampai `jumlah_bibit`, kita perlu `jumlah_bibit + 1`.
    - Variabel `i` akan mengambil setiap nilai dari urutan ini secara berurutan.

- bibit_tertanam_sukses += 1: Ini adalah singkatan dari bibit_tertanam_sukses = bibit_tertanam_sukses + 1.
- if bibit_tertanam_sukses >= batas_minimum_sukses: ... else: ...: Ini adalah contoh **struktur pemilihan (if-else)**.
    - Sama seperti Java, kode di dalam if akan dieksekusi jika kondisi BENAR, dan kode di dalam else jika SALAH.
    - **Indentation (indentasi):** Perhatikan bahwa Python menggunakan **indentasi** (spasi atau tab di awal baris) untuk menandai blok kode di bawah for, if, dan else. Ini sangat penting di Python! Salah indentasi akan menyebabkan *error*.
- f"String {variabel}": Ini adalah *f-string* (formatted string literal), cara modern di Python untuk menyisipkan nilai variabel ke dalam string dengan mudah.

Output yang Diharapkan (Python - Contoh Input 3):

```
=== Program Simulasi Penanaman Mangrove ===
Masukkan jumlah bibit mangrove yang akan ditanam:
3

Memulai proses penanaman...
  Bibit ke-1:
    - Menggali lubang.
    - Memasukkan bibit.
    - Menutup lubang dengan tanah.
    - Menyiram bibit.
  Bibit ke-2:
    - Menggali lubang.
    - Memasukkan bibit.
    - Menutup lubang dengan tanah.
    - Menyiram bibit.
  Bibit ke-3:
    - Menggali lubang.
    - Memasukkan bibit.
    - Menutup lubang dengan tanah.
    - Menyiram bibit.
Proses penanaman 3 bibit selesai.

--- Evaluasi Hasil Penanaman ---
Perhatian: Hanya 3 bibit yang berhasil ditanam.
Mohon periksa kembali metode penanaman atau lokasi.
```

Dengan contoh-contoh ini, Anda sekarang bisa melihat bagaimana tiga struktur dasar algoritma (sekuensial, pemilihan, dan perulangan) diimplementasikan dalam kode nyata. Ini adalah fondasi yang akan kita bangun di bab-bab berikutnya.

BAB 3

TIPE, OPERATOR, EKSPRESI, DAN RUNTUNAN PADA ALGORITMA – MENGENAL BAHAN BAKU ALGORITMA

Di bab sebelumnya, kita sudah belajar bahwa algoritma adalah serangkaian instruksi untuk memecahkan masalah. Instruksi-instruksi ini, pada dasarnya, adalah manipulasi terhadap **data**. Tapi, data itu sendiri tidak selalu sama, bukan? Ada angka, teks, tanggal, dan lain-lain. Bagaimana algoritma tahu cara memperlakukan data-data ini? Di sinilah konsep **tipe data**, **operator**, dan **ekspresi** berperan. Mereka adalah fondasi bagaimana algoritma menyimpan, memproses, dan menghasilkan informasi.

3.1 Tipe Data: Klasifikasi Informasi

Sama seperti di dunia nyata kita membedakan antara angka, huruf, gambar, atau suara, dalam algoritma, **tipe data** adalah klasifikasi yang memberitahu komputer jenis nilai apa yang dapat disimpan oleh suatu variabel dan operasi apa yang dapat dilakukan padanya.

Setiap variabel (ingat, variabel itu seperti "kotak" tempat menyimpan data) harus memiliki tipe data.

Beberapa tipe data dasar yang umum:

- **Bilangan Bulat (Integer / `int`):**
    - Digunakan untuk menyimpan angka bulat tanpa pecahan, baik positif, negatif, maupun nol.
    - **Contoh:** `5`, `-10`, `0`, `1000`.
    - **Relevansi Konservasi/Budaya:** Jumlah spesies, jumlah pengunjung museum, tahun penemuan artefak, jumlah anggota suku adat.
- **Bilangan Desimal / Pecahan (Floating-Point / `float` / `double`):**
    - Digunakan untuk menyimpan angka dengan bagian desimal. `float` (floating-point) dan `double` (double precision floating-point) adalah variasi yang berbeda dalam presisi penyimpanan angka desimal.
    - **Contoh:** `3.14`, `-0.5`, `98.6`, `2.71828`.
    - **Relevansi Konservasi/Budaya:** Suhu rata-rata, persentase kelembaban, luas area konservasi, koordinat geografis.
- **Karakter (Character / `char`):**
    - Digunakan untuk menyimpan satu huruf, angka, simbol, atau spasi.
    - **Contoh:** `'A'`, `'z'`, `'7'`, `'$'`, `' '`.
    - **Relevansi Konservasi/Budaya:** Inisial nama, simbol identifikasi (misal: 'M' untuk Jantan, 'F' untuk Betina), satu huruf awal kode.