Journal of Computer Science and Computational Intelligence
November 2025 | Vol. 1 | No. 2 | Page 61-68
E-ISSN: 3109-7383 P-ISSN: 3110-8393
DOI: https://doi.org/10.30598/algorithm.v1i2.61-68

Programmer Logis: Analisis Peran Logika Informatika dalam Menulis Kode Program yang Valid, Efisien, dan Solutif

(Analysis of the Role of Logic in Computer Science in Writing Valid, Efficient, and Solution-Oriented Program Code)

Rindy Marsya Mataheruila[1], Citra Fathia Palembang[2], Dedy Ricardo Serumena[3*]
[1,2]Program Studi Ilmu Komputer, Fakultas Sains dan Teknologi, Universitas Pattimura
Jl. Ir. M. Putuhena, Ambon, 97233, Indonesia
[*]Corresponding author's e-mail: *** rserumena@gmail.com**

Manuscript submitted: ***18th November 2025***
Manuscript revision: ***22th November 2025***
Accepted for publication: ***25th November 2025***

Abstract

The uncompromising pace of technological advancement demands that program code be written not merely to function, but to be truly precise, efficient, and free from even the smallest errors. A single logical mistake can trigger catastrophic system failures, compromise data integrity, and potentially endanger user safety. Through the application of propositional logic, predicate logic, and the disciplined use of data structures and algorithms, programmers are able to formalize programming languages and construct a code foundation that is resilient to elementary errors. This journal critically highlights how propositional logic in informatics and control structures such as if and else serve as decisive factors in determining code quality, enabling developers to eliminate ambiguity and neutralize potential logical errors before they escalate into major issues. This study employs a comparative case study approach by examining 50 code samples drawn from the Codeforces, ManyBugs, and Bugs.jar datasets. The analysis is conducted by juxtaposing logically valid code with erroneous (buggy) code using Truth Tables and Predicate Logic to validate algorithmic flow. The results demonstrate that disciplined application of informatics logic significantly enhances program accuracy, efficiency, and stability. Robust logical structures also substantially reduce debugging time. In conclusion, informatics logic is not merely an academic concept, but an absolute foundation that determines whether software is fit for use or instead becomes a threat to the system itself.

Keywords: *Informatics Logic; Code, Writing; Programming; If-Else*

How to cite this article:
R. M. Mataheruila, C. F. Palembang and D. R. Serumena, "Programmer Logis: Analisis Peran Logika Informatika dalam Menulis Kode Program yang Valid, Efisien, dan Solutif", *algorithm*, vol. 1, no. 2, pp. 61-68, November 2025.

Journal homepage: https://ojs3.unpatti.ac.id/index.php/algorithm
Research Article Open Access

1. PENDAHULUAN

Dalam era digital saat ini, Menulis kode yang benar dan efisien adalah langkah utama dalam memastikan kualitas perangkat lunak yang dikembangkan kode yang ditulis secara tepat tidak hanya memastikan program berjalan sesuai dengan yang diharapkan, tetapi juga mempermudah proses pemeliharaan dan pengembangan di masa depan. Selain itu, hal ini juga mengurangi risiko kesalahan yang bisa berpengaruh buruk terhadap keamanan dan kestabilan sistem. Kesalahan dalam penulisan kode seperti logika yang keliru dapat menyebabkan berbagai masalah serius, mulai dari performa yang lambat, celah keamanan, hingga kegagalan sistem secara menyeluruh. Penulisan kode perlu didasari oleh proses berpikir yang sistematis dan logis, agar hasilnya lebih efektif dan terstruktur dengan baik [1] . Pada kenyataannya, banyak pengembang perangkat lunak lebih fokus pada menguasai sintaksis bahasa pemrograman daripada memahami secara mendalam logika informatika. Padahal, logika informatika adalah dasar dari cara berpikir sistematis yang sangat membantu *programmer* dalam menyusun alur algoritma yang benar dan efisien. Logika dan informatika menjadi peran utama dalam membangun pola pikir yang sistematis dan terstruktur [2]. Dengan menerapkan prinsip-prinsip logika informatika, para *programmer* dapat menulis kode yang tidak hanya efektif dan efisien, tetapi juga menghindari kesalahan logika yang sering menjadi sumber utama *bug* dan kegagalan dalam sebuah program. Penggunaan logika informatika juga berperan penting dalam perancangan algoritma yang tepat, memastikan setiap langkah di dalam kode memiliki alasan dan tujuan yang jelas, sehingga proses pengembangan perangkat lunak menjadi lebih terukur dan dapat diandalkan. Penguasaan logika, seperti logika proposisional dan penerapan struktur kontrol seperti pernyataan *if-else*, memungkinkan *programmer* dalam membuat keputusan yang benar dalam alur eksekusi program, serta mencegah terjadinya kesalahan dalam penulisan kode [3].

Beberapa penelitian sebelumnya telah menyoroti pentingnya kualitas penulisan kode dan menunjukkan bahwa teknik *debugging* sangat penting dalam meningkatkan keandalan perangkat lunak [5]. Meskipun banyak penelitian yang telah dilakukan, sebagian besar lebih menekankan aspek teknis seperti standar *coding* dan metode *debugging*, tanpa secara khusus membahas peran logika informatika dalam proses penulisan kode [4]. Menyoroti pentingnya pemahaman algoritma, belum menggali secara mendalam bagaimana prinsip logika proposisional dan struktur kontrol diterapkan dalam praktik *coding* sehari-hari. Oleh karena itu, masih ada kekosongan penelitian yang secara khusus mengkaji bagaimana penerapan logika informatika dapat membantu dalam mengidentifikasi pola kesalahan menulis kode[6].

2. METODE PENELITIAN

2.1. Jenis dan Pendekatan Penelitian

Penelitian ini menggunakan jenis penelitian kualitatif dengan pendekatan studi kasus untuk mengkaji peran logika informatika dalam penulisan kode yang benar dan efektif. Pendekatan Studi kasus dipilih karena metode ini memungkinkan analisis mendalam terhadap fenomena yang terjadi dalam konteks nyata, yaitu praktik penulisan kode oleh *programmer* pada berbagai sumber yang tersedia secara *online*. Penelitian akan berfokus pada kode program (struktur sintaksis dan semantik), logika algoritma yang mendasari kode, kesalahan logika dalam kode yang sering terjadi, dan penerapan struktur logika (misalnya, kondisi, perulangan, fungsi) dalam penulisan kode yang efektif.

2.2. Objek Penelitian

Penelitian ini akan berfokus pada beberapa elemen kunci agar dapat memahami secara mendalam peran logika informatika dan kualitas kode program. Berikut merupakan objek-objek yang diteliti yaitu:

1. Kode program (struktur sintaksis dan semantik), Ini merupakan bagian utama dari penelitian ini, yang mencakup baris-baris instruksi langsung yang ditulis dalam bahasa

pemrograman tertentu. Analisisnya akan meliputi struktur sintaksis dan semantik dari kode tersebut, serta bagaimana kode tersebut mengimplementasikan solusi tertentu. Kami juga akan meninjau bagaimana elemen-elemen seperti variabel, operator, fungsi, dan kelas tersusun dan saling berinteraksi[7].

2. Logika algoritma yang mendasari kode, Objek ini mengacu pada cara berpikir serta langkah-langkah yang disusun untuk menemukan solusi atas suatu masalah. Kami akan menelusuri bagaimana prinsip dasar seperti logika proposisional dan predikat, induksi matematika, rekursi, serta struktur data abstrak diubah menjadi algoritma yang efektif dan akurat dalam kode program[8].
3. Kesalahan logika dalam kode yang sering terjadi, penting juga untuk mengenali dan mengelompokkan kesalahan yang muncul bukan karena *syntax error*, tetapi lebih disebabkan oleh pola pikir yang kurang tepat atau implementasi logika yang salah. Kesalahan ini bisa berupa *bug* yang menyebabkan hasil tidak akurat, *infinite loops*, kondisi yang tidak pernah terpenuhi, atau pengabaian terhadap *edge cases* tertentu[9].
4. Penerapan struktur logika, Objek ini membahas bagaimana konsep-konsep logika informatika diimplementasikan ke dalam struktur kode. Ini mencakup penggunaan elemen *control* seperti kondisi (seperti *if-else* dan *switch-case*), perulangan (seperti *for*, *while*, dan *do-while*), fungsi, serta modularitas. Kami akan melihat bagaimana penerapan struktur-struktur ini memengaruhi keakuratan, kenyamanan membaca, dan efisiensi kode secara keseluruhan[8].

2.3. Sumber Dataset

Penggunaan dataset untuk *code analysis* menjadi bagian penting dalam mengungkap bagaimana logika informatika berpengaruh langsung terhadap kualitas suatu program. Dataset seperti *Codeforces*, *LeetCode*, *ManyBugs*, dan Bugs.jar menyediakan kumpulan kode nyata yang mengandung berbagai bentuk kesalahan logika, mulai dari kondisi *if-else* yang tidak tepat, perulangan yang salah, hingga struktur kontrol yang tidak konsisten. Melalui analisis terhadap kesalahan-kesalahan tersebut, penelitian ini menunjukkan bahwa ketidakmampuan memahami dan menerapkan logika informatika secara benar selalu berujung pada program yang tidak valid, inefisiensi algoritma, serta kegagalan sistem[7].

1. ***Codeforces* atau *LeetCode submissions*.**
   Merupakan kumpulan kode nyata dari *programmer* seluruh dunia yang bisa dianalisis untuk melihat kesalahan logika, *Codeforces* atau *LeetCode submissions* (menganalisis kesalahan logika) adalah *website* tempat programmer dari seluruh dunia mengerjakan soal-soal pemrograman kemudian di submit, Dari situ, dapat di ketahui Mana kode yang berhasil, Mana kode yang gagal (*wrong answer*, *time limit exceeded*, *runtime error*) kesalahan-kesalahan ini di jadikan sebagai data penelitian[4].

2. ***ManyBugs* Dataset.**
   Melalui dataset *bug real-world* pada program C, peneliti dapat mengamati langsung berbagai bentuk kesalahan logika, seperti kondisi yang tidak tepat, perulangan yang berpotensi *infinite loop*, alur kontrol yang salah. Analisis terhadap *bug* yang tercatat di *ManyBugs* memperlihatkan bahwa sebagian besar *error* muncul akibat tidak diterapkannya prinsip-prinsip dasar logika informatika. Dengan membandingkan versi kode sebelum perbaikan dengan versi setelah perbaikan, penelitian ini memberikan bukti empiris bahwa penerapan logika yang benar dapat mengembalikan validitas program, meningkatkan efisiensi eksekusi, dan menghasilkan solusi yang tepat[7].

3. **Bugs.jar Dataset.**

Bugs.jar merupakan dataset *bug* berbasis Java yang berisi kumpulan kesalahan logika nyata pada berbagai proyek perangkat lunak. Dalam konteks penelitian ini, Bugs.jar menjadi bukti empiris bahwa kegagalan menerapkan logika informatika secara benar selalu berujung pada ketidakstabilan sistem. Melalui analisis terhadap pola *bug* yang terdapat dalam dataset, penelitian ini menegaskan bahwa logika informatika memiliki peran sentral dalam menghasilkan kode yang valid, efisien, dan solutif[8].

2.4. Teknik Pengumpulan Data

Dalam penelitian ini, teknik pengumpulan data dilakukan dengan beberapa cara untuk memperoleh data yang relevan dan mendalam mengenai peran logika informatika dalam penulisan kode yang benar dan efektif [9]. Total sampel yang dianalisis dalam penelitian ini berjumlah 50 kode, yang terdiri dari 20 submission algoritma (Codeforces), 15 kasus kesalahan memori (ManyBugs), dan 15 kasus logika bisnis (Bugs.jar). Teknik-teknik yang digunakan yaitu:

Tabel 1. *Codeforces* (Analisis Kesalahan Logika)

No	Teknik Pengumpulan Dataset	Observasi dan Sampling Code
1	Observasi langsung terhadap submission	Submission yang gagal (Wrong Answer, Runtime Error, Time Limit Exceeded) Submission yang berhasil untuk perbandingan penjelasan error dari sistem online judge
2	Pengambilan sampel (purposive sampling)	Kesalahan logika (logical error) Struktur kontrol yang salah (if-else, loop, operator logika) Alur logika yang tidak konsisten
3	Dokumentasi kode dan Analisis komparatif	*Buggy code* vs *correct code* Logika sebelum diperbaiki vs sesudah diperbaiki

Tabel 2. *ManyBugs* Dataset (Pemograman C)

No	Teknik Pengumpulan Dataset	Observasi dan *Sampling Code*
1	*Download* dataset dari repositori resmi.	Versi kode sebelum perbaikan (*buggy version*) Versi kode sesudah perbaikan (*fixed version*)
2	*Extract* dan klasifikasi data.	Dokumentasi lokasi *bug* *Test case* terkait *bug* Jenis *bug* *File* yang terdampak Kondisi *if* yang salah *Loop* yang tidak tepat Fungsi atau logika yang keliru Pointer/variabel yang tidak diperbarui
3	Analisis kode berbasis logika.	Alur kontrol yang tidak logis

Tabel 3. Bugs.jar: Dataset Mining & Code Examination

No	Teknik Pengumpulan Dataset	Observasi dan *Sampling Code*
1	*Download* dataset	Dataset diunduh dalam bentuk: • *buggy-class* • *fixed-class* Setiap *bug* memiliki: • laporan *bug* • *patch* perbaikan
2	Penelusuran *bug*	Peneliti menelusuri logika yang penyebabkan *error*: • kesalahan operator logika • logika percabangan tidak tepat • kondisi null tidak ditangani • loop salah batas
3	Pemeriksaan kode *buggy* secara manual	• full project • *test case* untuk verifikasi *bug* • *commit history*
4	Analisis perbaikan	Membandingkan kode *buggy* dengan kode *fixed* untuk melihat: • bagian logika apa yang diubah • bagaimana perubahan tersebut memperbaiki *error*

2.5. Teknik Analisis Data

Setelah data dikumpulkan melalui berbagai metode seperti studi dokumentasi, observasi langsung, analisis konten, dan studi perbandingan [10]. langkah berikutnya adalah melakukan analisis data. Teknik analisis yang dipilih dalam penelitian ini meliputi:

Tabel 4. Analisis Dataset

No	Analisis Data	Langkah-langkah	Tujuan
1	*Static Code Analysis*	▪ Mengeliminasi data yang tidak berkaitan dengan penerapan logika dalam penulisan kode program. ▪ Memilah contoh kode, kasus, dan konten yang sesuai dengan rumusan masalah.	Untuk memperjelas dan memfokuskan data yang relevan.
2	*Comparative Logic Analysis*	▪ Menyajikan ringkasan hasil studi dokumentasi, observasi studi kasus, analisis konten, dan studi komparatif. ▪ Menampilkan visualisasi seperti	Untuk Mempermudah pemahaman terhadap pola dan hubungan antar data.

		tabel dan diagram untuk membandingkan kualitas kode dan penerapan logika. ▪ Mengidentifikasi pola penerapan logika yang benar dan salah. ▪ Mengevaluasi efektivitas metode yang ditemukan dalam berbagai sumber. ▪ Memberikan saran perbaikan untuk meningkatkan kualitas kode.	
3	Penarikan Kesimpulan		Untuk Memberikan hasil akhir dan rekomendasi yang sesuai dengan tujuan penelitian.

Teknik pengumpulan data ini bertujuan untuk memberikan gambaran yang terstruktur mengenai teknik analisis data dalam pengumpulan data yang sesuai dengan penelitian. Teknik pengumpulan data ini, diharapkan hasil penelitian yang dapat memberikan pemahaman tentang bagaimana logika informatika diterapkan secara efektif dalam penulisan kode program yang benar, efisien, dan minim kesalahan.

3. HASIL DAN PEMBAHASAN

3.1. Analisis Kegagalan Logika Deteksi Status (*State Detection Logic*)

Studi kasus utama diambil dari permasalahan algoritma *Encode dan Decode* (mencakup varian mudah dan sulit). Tantangan fundamental pada kasus ini adalah ketiadaan persistensi memori antar-eksekusi (*stateless execution*), di mana program dijalankan dua kali secara terpisah namun harus mampu mengenali konteks eksekusi ("Putaran 1" atau "Putaran 2").

Analisis terhadap sampel kode yang gagal menunjukkan pola kesalahan fatal di mana programmer menerapkan logika naif dengan asumsi bahwa nilai variabel global akan tersimpan di memori. Sebaliknya, solusi yang valid memanfaatkan logika proposisional berbasis observasi input.

Tabel 5. Komparasi Logika: Deteksi Status Eksekusi

Komponen	Kode Salah (Logika Naif)	Kode Formal (Logika Proposisional)
Cuplikan Kode	if (run_count == 1) { ... }	if (input_string == "") { ... }
Dasar Logika	Mengandalkan *Variable State* (Internal). Berasumsi variabel penghitung bertahan antar sesi.	Mengandalkan *Fact Observation* (Eksternal). Memeriksa keberadaan data input.
Analisis Formal	Premis Salah: $P(t_1) \rightarrow P(t_2)$ (Nilai variabel pada waktu t_1 terbawa ke t_2) Fakta: Sistem melakukan *reset memory*, sehingga $P(t_2) = \emptyset$	Implikasi Valid: Misalkan p adalah proposisi "Input Kosong". Jika $p \equiv True \rightarrow State = Run\_1$. Jika $p \equiv False \rightarrow State = Run\_2$.

Kegagalan pada kolom "Kode Salah" menunjukkan ketidakmampuan programmer dalam membedakan *scope* variabel statis dan dinamis. Dalam Logika Informatika, ini adalah pelanggaran

terhadap *Logika Temporal* (logika yang melibatkan waktu). Programmer menganggap program memiliki ingatan (*stateful*), padahal lingkungan eksekusi bersifat *stateless*.

Sebaliknya, "Kode Benar" menerapkan prinsip logika proposisional yang deterministik. Keputusan percabangan (if-else) tidak didasarkan pada asumsi variabel yang rapuh, melainkan pada bukti fakta input yang diberikan sistem. Penerapan logika ini menghilangkan ambiguitas dan menjamin program berjalan valid pada setiap putaran eksekusi tanpa risiko kesalahan memori.

3.2. Studi Kasus *ManyBugs*: Kegagalan Logika Sekuensial & Memori

Analisis terhadap dataset *ManyBugs* (berbasis bahasa C) mengungkap pola kesalahan fatal pada manajemen memori, khususnya terkait urutan logika (*sequence logic*) dalam pengaksesan *pointer*. Kesalahan ini terjadi karena programmer melanggar hukum implikasi logika: mengakses data sebelum memvalidasi keberadaannya.

Tabel 6. Komparasi Logika: Penanganan Pointer (*Null Safety*)

Komponen	Kode Salah	Kode Formal

Cuplikan Kode	int val = list->value;
	
	if (list != NULL) { ... }
		if (list != NULL) {
		
		int val = list->value; ... }

Urutan Logika	1. Akses Memori (Action) 2. Validasi (Condition)	1. Validasi (Condition) 2. Akses Memori (Action)
Analisis Formal	Logika Terbalik: $Action \rightarrow Condition$ Jika $list = NULL$, baris 1 langsung memicu *Segmentation Fault* (*Crash*).	Implikasi Valid: $(list \neq NULL) \rightarrow Action$ Premis $list \neq NULL$ harus bernilai *TRUE* terlebih dahulu sebelum konsekuen (Action) dieksekusi.

Contoh di atas memperlihatkan pelanggaran terhadap Logika Predikat. Dalam logika informatika yang benar, sebuah pre-kondisi (syarat) harus selalu mendahului eksekusi (akibat). Kode buggy mencoba menarik data (list->value) dari alamat memori yang belum dipastikan eksistensinya. Ini membuktikan bahwa kesalahan sistem level rendah (*low-level*) sering kali bukan disebabkan oleh ketidaktahuan sintaksis, melainkan kekacauan dalam menyusun alur premis logika. Perbaikan dilakukan dengan memindahkan deklarasi ke dalam scope logika yang terlindungi (*Guard Clause*).

4. KESIMPULAN DAN SARAN

Penelitian ini menyimpulkan bahwa logika informatika memiliki peran yang sangat penting dalam memastikan penulisan kode yang benar. Hasil dari studi kasus yang kami lakukan menunjukkan bahwa, meskipun banyak *programmer* fokus pada penguasaan sintaksis bahasa pemrograman, mereka seringkali kurang memiliki pemahaman mendalam tentang logika informatika. Hal ini menjadi penyebab utama munculnya kesalahan logika dalam kode yang mereka buat. Jika kesalahan ini tidak segera ditangani, bisa berakibat serius terhadap kinerja, keamanan, serta kestabilan sistem secara keseluruhan. Dengan pendekatan yang sistematis dan berbasis logika, kesalahan dalam struktur program dapat dikurangi secara signifikan.

Dengan menerapkan prinsip-prinsip logika informatika secara konsisten, seperti logika proposisional dan struktur kontrol (misalnya *if-else*), kita dapat melihat peningkatan yang nyata dalam kualitas kode secara keseluruhan. Ini terlihat dari peningkatan kohesi, pengurangan ketergantungan antar bagian kode, dan penurunan jumlah *bug* yang signifikan setelah dilakukan restrukturisasi berdasarkan prinsip-prinsip logika tersebut. Penerapan prinsip-prinsip logika tidak hanya membantu dalam *debugging*, tetapi juga meningkatkan kualitas dan efisiensi kode secara keseluruhan. Oleh karena itu, penguasaan logika informatika sangat diperlukan oleh setiap

programmer, baik pemula maupun profesional, untuk menciptakan sistem yang andal, efisien, dan bebas dari kesalahan logika.

REFERENSI

[1] A. Maulana et al., *Rekayasa Perangkat Lunak: Konsep, Metode, dan Praktik Terbaik*. Get Press, 2023.

[2] M. N. Ardian dan D. Soyusiawaty, "Multimedia Pembelajaran Logika Informatika pada Mahasiswa Teknik Informatika," *Jurnal Sarjana Teknik Informatika*, vol. 2, no. 1, 2014.

[3] A. Sari, "Aplikasi CAI untuk Pembelajaran Logika Informatika," *Jurnal Untan, Jurnal Aplikasi dan Riset Informatika (JUARA)*, vol. 1, no. 1, 2022.

[4] A. M. Hassan dan L. Wei, "Programming Errors: Syntax vs Logic," *International Journal of Computer Science and Information Security*, vol. 17, no. 4, 2019.

[5] M. A. Smith dan J. D. Lee, "The Impact of Logical Thinking Skills on Programming Performance," *Journal of Software Engineering and Applications*, vol. 13, no. 1, 2020, Art. No. 13001. Doi: 10.4236/jsea.2020.13001.

[6] J. Brownlee, "Machine Learning Mastery: Logic in Algorithms," 2016. [Online]. Available: https://machinelearningmastery.com.

[7] R. Hidayat, "Studi Sintaksis dan Semantik Bahasa Pemrograman dalam Pengembangan Sistem Informasi," *Jurnal Sistem Informasi*, vol. 15, no. 1, pp. 33-40, 2019.

[8] I. G. N. A. Putra dan P. I. Santosa, "Penerapan Struktur Kontrol dan Modularitas dalam Pengembangan Perangkat Lunak Berbasis Java," *Jurnal Ilmiah Teknologi Informasi Terapan*, vol. 7, no. 1, pp. 45-52, 2021.

[9] T. Handayani dan E. Prasetyo, "Penerapan Logika Informatika pada Pengembangan Algoritma untuk Penyelesaian Masalah Kompleks," *Jurnal Informatika*, vol. 12, no. 2, pp. 101-110, 2018.

[10] N. F. M. Noor, "An integrated development environment on problem-solving for C programming fundamentals," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 18, no. 3, pp. 4–17, 2024. doi: 10.3991/ijim.v18i03.46820.