code basicstyle=, frame=single, rulecolor=, breaklines=true, showstringspaces=false, columns=fullflexible, keepspaces=true, framesep=6pt, aboveskip=1em, belowskip=1em, xleftmargin=0pt

out basicstyle=, frame=tb, rulecolor=, breaklines=true, showstringspaces=false, columns=fullflexible, keepspaces=true, framesep=4pt, aboveskip=1em, belowskip=1em

Anatomi Tipe Data:
Dari Representasi Bit Hingga Sistem Tipe Tingkat Lanjut

Muhammad Riduwan Khafidi
Program Studi Teknik Rekayasa Perangkat Lunak
Politeknik Negeri Batam
4342611034@students.polibatam.ac.id

November 5, 2026

Abstrak

Tipe data adalah fondasi fundamental dalam ilmu komputer yang mendefinisikan bagaimana nilai direpresentasikan, dioperasikan, dan dibatasi di dalam memori mesin. Makalah ini menyajikan kajian komprehensif mengenai tipologi data, mulai dari representasi biner tingkat mesin (machine types) hingga struktur abstrak tingkat tinggi seperti Dependent Types dan Refinement Types. Pembahasan mencakup definisi tipe data menurut literatur klasik, klasifikasi (atomik vs komposit), anatomi tipe primitif (Boolean, Numerik, String), tipe komposit (Array, Record), Abstract Data Types (ADT), hingga Algebraic Data Types yang populer di bahasa fungsional dan modern. Melalui sepuluh visualisasi diagram, makalah ini memetakan perjalanan data dari sekadar kumpulan bit hingga menjadi entitas yang bermakna dan aman secara logika. Makalah ini juga dilengkapi studi kasus perancangan sistem tipe, tabel kesalahan umum, dan lembar rujukan cepat.

1 Pendahuluan

1.1 Analogi Pembuka

Bayangkan sebuah gudang logistik yang sangat besar. Jika semua barang—mulai dari jarum jahit, televisi, hingga cairan kimia—dilempar begitu saja ke dalam satu ruangan tanpa rak atau kotak, gudang itu akan menjadi kekacauan. Barang-barang akan rusak, sulit ditemukan, dan berbahaya. Untuk mengelolanya, kita membutuhkan wadah yang spesifik: kotak kecil untuk jarum, peti kayu untuk televisi, dan drum anti-bocor untuk bahan kimia.

Dalam dunia perangkat lunak, tipe data adalah sistem pengarsipan canggih tersebut. Komputer pada dasarnya bodoh; ia hanya mengenal tegangan listrik yang diterjemahkan menjadi angka 0 dan 1. Tipe data adalah instruksi kepada kompilator tentang bagaimana urutan 0 dan 1 tersebut harus ditafsirkan, seberapa besar ruang memori yang harus dicadangkan, dan operasi logika atau matematika apa yang legal untuk dilakukan.

1.2 Mengapa Topik Ini Penting

Pemilihan tipe data yang salah ibarat meminjamkan palu besi kepada tukang kaca. Program mungkin berjalan di permukaan, namun akan hancur berantakan saat menghadapi data dunia nyata. Memahami anatomi tipe data memisahkan antara penulis kode pemula dengan insinyur perangkat lunak yang mampu membangun sistem terdistribusi yang aman, efisien, dan bebas dari anomali.

1.3 Peta Pembahasan

Makalah ini disusun secara sistematis. Bagian kedua membedah definisi tipe data secara filosofis. Bagian ketiga mengklasifikasikan tipe data. Bagian keempat membahas tipe mesin dan primitif. Bagian kelima mengeksplorasi tipe komposit dan abstrak. Bagian keenam menyelami tipe data tingkat lanjut (Sistem Tipe). Bagian ketujuh menyajikan perbandingan, dilanjutkan dengan studi kasus, kesalahan umum, lembar rujukan cepat, dan kesimpulan.

2 Konsep dan Definisi Filosofis

Dalam literatur ilmu komputer, definisi "tipe" seringkali bersifat implisit. Parnas, Shore, dan Weiss (1976) mengidentifikasi lima cara mendefinisikan tipe data:

Pertama, definisi **Sintaksis**, di mana tipe hanyalah label murni yang menempel pada variabel saat dideklarasikan. Kedua, definisi **Representasi**, di mana tipe didefinisikan berdasarkan komposisi tipe primitif di tingkat mesin. Ketiga, **Representasi dan Perilaku**, yang menambahkan sekumpulan operator (fungsi) yang memanipulasi representasi tersebut.

Keempat, definisi **Ruang Nilai** (*Value Space*), di mana tipe adalah himpunan nilai yang mungkin dimiliki oleh sebuah variabel. Kelima, **Ruang Nilai dan Perilaku**, yang menggabungkan himpunan nilai dengan himpunan fungsi yang dapat diterapkan padanya. Pendekatan ruang nilai dan perilaku inilah yang paling dekat dengan pemodelan *Object-Oriented Programming* (OOP) modern.

Perbedaan semantik juga sangat krusial. Tipe `int` di Python merepresentasikan bilangan bulat dengan presisi arbitrer (sebesar memori yang tersedia). Namun, tipe `int` di Java adalah bilangan bulat 32-bit yang kaku (berkisar $-2.147.483.648$ hingga $2.147.483.647$). Jika melebihi batas, Java akan mengalami *integer overflow* (nilai berputar kembali ke negatif). Di Rust, tipe 32-bit ini dinamakan `i32`, dan alih-alih berputar, program akan sengaja dihentikan (*panic*) saat mode *debug* untuk mencegah *bug* logika.

3 Klasifikasi Tipe Data

Tipe data dapat dikategorikan berdasarkan beberapa faktor. Berdasarkan asalnya, ada **Tipe Primitif** (bawaan bahasa) dan **Tipe Buatan Pengguna** (*User-defined*). Berdasarkan strukturnya,

ada **Tipe Atomik** (satu item yang tak terpecahkan, seperti integer) dan **Tipe Komposit** (kumpulan item yang bisa diakses terpisah, seperti array).

Selain itu, ada **Tipe Dasar** (*Basic*) yang didefinisikan secara aksiomatik (misal: integer dalam matematika), dan **Tipe Turunan** (*Generated/Derived*) yang dibangun dari tipe lain (misal: array dari integer).

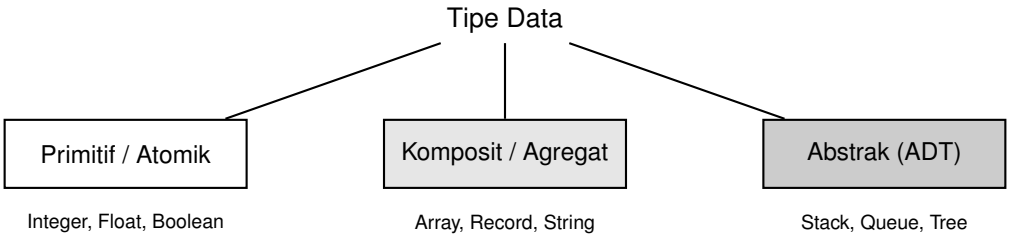


Figure 1. Klasifikasi hierarkis tipe data.

4 Tipe Mesin dan Primitif

Di tingkat terendah, semua data direpresentasikan sebagai *bit* (0 dan 1). Unit terkecil yang dapat dialamatkan adalah *byte* (8 bit). Unit yang diproses oleh instruksi mesin disebut *word* (umumnya 32 atau 64 bit).

Tipe mesin memberikan kendali presisi atas perangkat keras, namun mengorbankan portabilitas. Bahasa tingkat tinggi seperti C menyediakan tipe Boolean, integer, dan float, namun representasi bit pastinya ditentukan oleh implementasi kompilator. Hanya tipe `char` di C yang dijamin berukuran tepat satu byte.

4.1 Tipe Boolean

Tipe Boolean hanya memiliki dua nilai: Benar (*true*) dan Salah (*false*). Secara logis, ini hanya butuh 1 bit. Namun, karena arsitektur CPU modern membaca data dalam blok memori, Boolean seringkali tetap memakan 1 byte penuh. Banyak bahasa (seperti C versi lama) tidak memiliki Boolean eksplisit, melainkan menggunakan integer dengan konvensi 0 untuk Salah dan bukan-nol untuk Benar.

4.2 Tipe Numerik

Integer adalah tipe dasar bilangan bulat. Jika tipe 32-bit dipaksakan pada platform 16-bit, kompilator akan memecahnya menjadi dua integer 16-bit.

Untuk bilangan pecahan, terdapat **Floating-point** (IEEE 754) yang menyimpan angka dalam notasi ilmiah biner ($a \times 2^b$). Tipe ini memiliki jangkauan luas, namun rentan terhadap galat pembulatan (misalnya $0.1 + 0.2 \neq 0.3$ secara eksak). Sebaliknya, **Fixed-point** menyimpan angka desimal dengan mengalikannya menjadi integer (misal 12.34 disimpan sebagai 1234), sehingga presisi eksak terjaga, sangat cocok untuk aplikasi finansial.

Untuk angka yang sangat besar, tersedia **Bignum** (*Arbitrary-precision arithmetic*), yang batasannya hanya memori komputer, meskipun operasi matematikanya jauh lebih lambat.

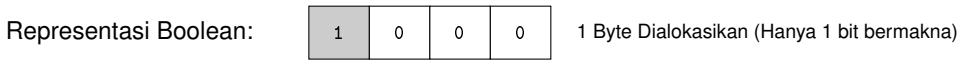


Figure 2. Representasi memori tipe Boolean.

5 Tipe Teks, Komposit, dan Abstrak

5.1 String dan Karakter

String adalah urutan karakter yang diambil dari set karakter seperti ASCII atau Unicode (UTF-8). Tipe string dapat berupa panjang tetap (*fixed-length*) yang selalu memakan memori sama, atau panjang variabel (*variable-length*) yang menyesuaikan panjang teks. String numerik seperti `"1234"` secara semantik berbeda dengan integer `1234`, meskipun beberapa bahasa memfasilitasi konversi otomatis.

5.2 Union Types

Tipe Union mendefinisikan bahwa suatu variabel dapat menampung salah satu dari beberapa subtipe yang diizinkan (misal: `float` ATAU `long integer`). Berbeda dengan *Record* yang menampung keduanya sekaligus, Union hanya menyimpan satu nilai pada satu waktu. *Tagged Union* menambahkan field penanda tipe untuk keamanan ekstrak data.

5.3 Algebraic Data Types (ADT)

ADT adalah gabungan dari *Sum Type* (tipe penjumlahan, "ATAU") dan *Product Type* (tipe perkalian, "DAN"). Nilai ADT terdiri dari tag konstruktor dan field nilai. Contoh klasiknya adalah tipe `Option` (Haskell: `data Maybe a = Nothing | Just a`), yang digunakan untuk menangani nilai kosong secara aman tanpa *null pointer exception*.

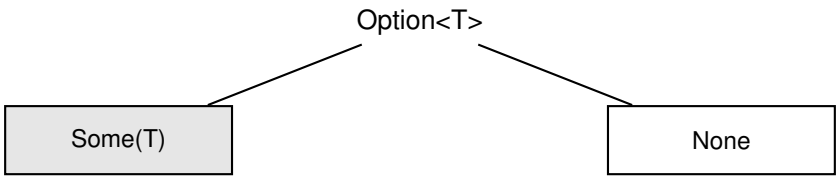


Figure 3. Sum Type pada tipe Option (Mungkin berisi nilai, mungkin kosong).

5.4 Struktur Data dan Abstract Data Types (ADT)

Struktur data adalah tipe komposit yang digunakan untuk menyimpan dan mengambil data. Contohnya adalah **Array** (akses acak via indeks), **Record** (kumpulan field bernama), **Linked List** (node yang saling terhubung), dan **Binary Tree** (pohon pencarian cepat).

Di atas struktur data, terdapat *Abstract Data Type* (ADT). ADT tidak mendefinisikan representasi konkret, melainkan hanya spesifikasi operasi (seperti `push` dan `pop` pada Stack). Implementasi di baliknya bisa menggunakan Array atau Linked List, selama aturan LIFO (Last-In-First-Out) terpenuhi.

6 Tipe Tingkat Lanjut (Sistem Tipe)

Ilmu komputer teoretis mengembangkan sistem tipe yang jauh lebih canggih untuk menjamin kebenaran program.

6.1 Pointers dan References

Pointer adalah tipe data yang nilainya merujuk pada alamat memori nilai lain. Ini adalah bentuk referensi primitif. Jika pointer menunjuk ke alamat memori yang tidak valid (misal: sudah dihapus), upaya untuk melakukan *dereference* akan menyebabkan program *crash*. Oleh karena itu, sistem tipe modern memisahkan pointer dari integer biasa, meskipun representasi bit-nya sama.

6.2 Function Types

Dalam bahasa fungsional (dan JavaScript), fungsi diperlakukan sebagai tipe data kelas satu (*first-class citizen*). Fungsi dapat disimpan dalam variabel dan dioperasikan. Sistem tipe mendefinisikan tipe fungsi berdasarkan argumen dan nilai kembaliannya, misalnya `Int -> Bool`.

6.3 Type Constructors

Konstruktor tipe adalah operator yang membangun tipe baru dari tipe lama. Contohnya adalah `List<T>` yang membangun tipe List dari tipe T, atau `Product` yang menggabungkan dua tipe.

6.4 Quantified Types

Berdasarkan logika predikat, tipe dapat dikuantifikasi. *Universal Quantification* ($\forall x.f(x)$) berarti nilai tersebut bertipe $f(x)$ untuk semua tipe x. *Existential Quantification* ($\exists x.f(x)$) berarti ada tipe x tertentu yang memenuhi.

6.5 Refinement Types

Tipe yang dilengkapi dengan predikat logika. Misalnya, tipe bilangan asli yang lebih besar dari 5 ditulis sebagai $\{n \in \mathbb{N} \mid n > 5\}$. Ini memungkinkan kompilator menolak nilai yang tidak memenuhi syarat matematis sebelum program berjalan.

Dependent Types
Tipe yang definisinya bergantung pada nilai.
Contoh: Fungsi yang mengembalikan array berukuran N, di mana N adalah argumen fungsi.

Intersection Types
Tipe yang memuat nilai yang merupakan anggota dari dua tipe.
Contoh: Objek di Java yang mengimplementasikan `Serializable & Comparable`.

Figure 4. Perbandingan Dependent Types dan Intersection Types.

7 Perbandingan

Table 1. Ringkasan perbandingan seluruh kategori tipe data.

Kategori	Contoh	Karakteristik Utama
Tipe Mesin	Byte, Word	Kontrol hardware langsung, tidak portabel
Primitif	Integer, Boolean, Float	Atomik, batas memori jelas
Teks	String, Char	Berbasis encoding, array karakter
Komposit	Array, Record	Mengelompokkan data, akses per indeks/field
Abstrak (ADT)	Stack, Queue	Spesifikasi operasi, menyembunyikan implementasi
Aljabar (ADT)	Option, Result	Sum dari Product, aman dari null
Referensi	Pointer, Reference	Menunjuk alamat memori, rawan crash
Fungsi	Function Object	Fungsi sebagai nilai variabel
Kuantifikasi	forall, exists	Logika predikat dalam tipe
Refinement	$\{n \mid n > 5\}$	Tipe dengan batasan predikat matematis
Dependent	Dependent Pair	Tipe bergantung pada nilai spesifik
Intersection	`A & B`	Menggabungkan sifat dua tipe

8 Studi Kasus: Perancangan Sistem Tipe E-Commerce

8.1 Langkah 1: Identifikasi Kebutuhan

Kita akan merancang tipe data untuk produk e-commerce yang memiliki nama, harga, stok, dan status (tersedia/habis).

8.2 Langkah 2: Tipe String untuk Nama

Nama produk menggunakan tipe `String` karena panjangnya dinamis.

8.3 Langkah 3: Tipe Numerik untuk Harga

Harga tidak boleh menggunakan `Float` karena akan merusak perhitungan uang. Kita gunakan `Decimal` atau `Integer` (dalam satuan sen).

8.4 Langkah 4: Tipe Integer untuk Stok

Stok barang menggunakan `u32` (unsigned 32-bit integer) karena tidak mungkin bernilai negatif.

8.5 Langkah 5: Enum untuk Status

Status produk adalah Enum (tipe enumerasi): `Tersedia` atau `Habis`.

8.6 Langkah 6: Record untuk Produk

Semua field di atas digabungkan menjadi satu tipe `Record` (Struct).

8.7 Langkah 7: Penanganan Opsional

Apa yang terjadi jika produk memiliki diskon? Kita gunakan `Option<Diskon>` di mana jika tidak ada diskon, nilainya `None`.

8.8 Langkah 8: Implementasi Tipe

[style=code] struct Produk id: u64, nama: String, harga: Decimal, stok: u32, status: StatusProduk, diskon: Option<Decimal>,

enum StatusProduk Tersedia, Habis,

9 Kesalahan Umum dan Cara Menghindarinya

Table 2. Kesalahan umum terkait tipe data.

Kesalahan	Akibat	Perbaikan
Integer overflow	Nilai berputar negatif secara acak	Gunakan Bignum atau tipe 64-bit
Float untuk uang	Galat akumulatif pada transaksi	Gunakan Decimal atau Fixed-point
Null Pointer	Program *crash*	Gunakan Option / Nullable Type
Salah encoding teks	Teks menjadi karakter aneh (Mojibake)	Standarkan pada UTF-8
Type casting paksa	Kehilangan data (misal float ke int)	Gunakan konversi aman atau cek rentang
Mengabaikan Signed/Unsigned	*Bug* saat nilai dimasukkan negatif	Gunakan *signed* jika perlu nilai minus
Pointer arithmetic liar	Akses memori ilegal (Segfault)	Gunakan Reference / Smart Pointer
Mengabaikan tipe fungsi	Callback error	Definisikan tipe fungsi secara eksplisit

10 Lembar Rujukan Cepat

Table 3. Rujukan cepat pemilihan tipe data.

Kebutuhan	Tipe Data atau Representasi
Bilangan bulat umum	`int` (32-bit) atau `i32`
Bilangan bulat besar	`i64` atau `BigInt`
Bilangan desimal ilmiah	`float` (32-bit) atau `double` (64-bit)
Mata uang / Finansial	`Decimal` atau `Fixed-Point`
Teks dinamis	`String` atau `VARCHAR`
Teks statis	`CHAR(n)`
Nilai Benar/Salah	`Boolean`
Data kosong (mungkin ada)	`Option<T>` atau `Nullable`
Kumpulan data sejenis	`Array` atau `Vector`
Data berpasangan	`Struct` atau `Tuple`
Kumpulan status tetap	`Enum`
Fungsi sebagai argumen	`Function Type`
Referensi memori aman	`&T` atau `Reference`
Alamat memori mentah	`Pointer` (Hindari jika bisa)
Pembatasan nilai logika	`Refinement Type`

11 Kesimpulan

Tipe data bukan sekadar sintaks deklarasi, melainkan pilar utama dalam menjamin keamanan dan kebenaran perangkat lunak. Tiga prinsip penutup:

1. Tipe adalah Kontrak. Pemilihan tipe data adalah perjanjian antara pengembang dan mesin tentang bagaimana memori dikelola dan operasi apa yang legal.
2. Representasi Menentukan Batasan. Memahami bahwa `Float` memiliki galat atau `Integer` bisa meluap (overflow) adalah kunci untuk menulis algoritma yang tangguh.
3. Abstraksi Memberi Skalabilitas. Dari tipe primitif hingga *Dependent Types*, tipe data memungkinkan manusia memodelkan logika dunia nyata yang paling kompleks ke dalam sistem yang dapat dieksekusi oleh mesin secara aman.

Referensi

References

[1] Parnas, D. L., Shore, J. E., & Weiss, D. (1976). Abstract types defined as classes of variables. *Proceedings of the 1976 conference on Data: Abstraction, definition and structure*, 149–154. https://doi.org/10.1145/800237.807133

[2] Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database System Concepts* (7th ed.). McGraw-Hill Education.

[3] IEEE. (2019). *IEEE Standard for Floating-Point Arithmetic (IEEE 754-2019)*. Institute of Electrical and Electronics Engineers.

[4] Klabnik, S., & Nichols, C. (2019). *The Rust Programming Language*. No Starch Press.

[5] Brookshear, J. G. (2019). *Computer Science: An Overview* (13th ed.). Pearson.

[6] Cardelli, L., & Wegner, P. (1985). On Understanding Types, Data Abstraction, and Polymorphism. *ACM Computing Surveys*, 17(4), 471–523. https://doi.org/10.1145/6041.6042

[7] Wikipedia Contributors. (2026, September 16). *Data type*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Data_type