Tajemnice liczb zmiennoprzecinkowych

W tym artykule omówię liczby zmiennoprzecinkowe zgodne ze standardem IEEE 754, które są dostępne w większości języków programowania, a także opiszę ich budowę, możliwości i ograniczenia. Odniosę się również do powszechnie panujących przekonań, że liczby te są niedokładne lub że są niedeterministyczne. Pokażę ponadto wiele nieoczywistych pułapek, które czyhają na używających ich programistów. Ta wiedza może się przydać każdemu, obojętnie, w jakim języku i na jakie platformy programujesz!

Liczby zmiennoprzecinkowe to świetny wynalazek. Dostępne w językach programowania typy danych tego rodzaju pozwalają na niewielkiej liczbie bitów zapisywać liczby dodatnie, jak i ujemne, całkowite, jak i ułamkowe, bardzo małe (bliskie zeru), jak i bardzo duże. Ponieważ komputery na początku służyły głównie naukowcom do wykonywania obliczeń, historia różnych sposobów kodowania liczb jest tak stara, jak historia samej informatyki. Standard IEEE 754, na którym opierają się używane współcześnie liczby zmiennoprzecinkowe, pojawił się w roku 1985. To jego używają wszystkie typy danych, o których będziemy mówić w tym artykule.

Podczas nauki programowania i w miarę nabierania doświadczenia w używaniu wybranego języka programowania zrozumienie dla liczb zmiennoprzecinkowych można podzielić na trzy etapy. Najpierw programista używa takich liczb bez zastanowienia, przyjmując, że reprezentują one dowolne liczby rzeczywiste. Szybko jednak może się natknąć na trudności z tym związane i zaobserwować różne wynikające z tego błędy. Wówczas przychodzi moment na refleksję nad ograniczeniami tego typu liczb. Stosowanie prostych „zasad ograniczonego zaufania" do liczb zmiennoprzecinkowych pozwala uniknąć wielu błędów. Są to zasady takie jak:

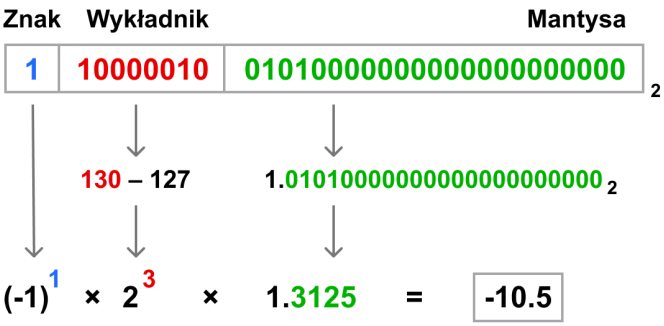
- „Liczby zmiennoprzecinkowe są niedokładne."
- „Liczby zmiennoprzecinkowe są niedeterministyczne."
- „Liczb zmiennoprzecinkowych nie wolno porównywać operatorem ==."

Warto jednak zagłębić się bardziej w szczegóły budowy i działania tych typów danych. Wówczas, dzięki ich lepszemu zrozumieniu, można ich używać świadomie, unikać błędów bądź minimalizować szanse ich wystąpienia, a równocześnie wykorzystywać pełnię ich możliwości. Ten artykuł ma za zadanie wprowadzić czytelnika w tajniki liczb zmiennoprzecinkowych, pokazać różne pułapki i nieoczywiste zjawiska z nimi związane.

PODSTAWY

Zasada działania liczb zmiennoprzecinkowych przypomina notację naukową używaną na lekcjach fizyki w szkole. O ile liczby, którymi posługujemy się na co dzień, np. licząc pieniądze, można zapisywać zwyczajnie, pisząc np. 12300, o tyle w fizyce często mamy do czynienia z liczbami bardzo dużymi lub bardzo małymi. Na przykład odległość Ziemi od Słońca wynosi około 150 milionów kilometrów, co zamiast 150 000 000 możemy wygodniej zapisać jako 1.5×10^8. Tak zapisana liczba jest niejako „znormalizowana" do postaci, w której przed przecinkiem została tylko pierwsza cyfra, natomiast wykładnik potęgi służy do tego, aby przesunąć przecinek o wybraną liczbę miejsc dziesiętnych w prawo (zwiększając wartość liczbową), kiedy jest wykładnik dodatni, lub w lewo (zmniejszając wartość), kiedy jest ujemny.

Liczby zmiennoprzecinkowe działają na podobnej zasadzie jak taka notacja naukowa w fizyce, ale na liczbach binarnych, a nie na dziesiętnych. W pamięci komputera mają budowę nieco bardziej skomplikowaną od liczb całkowitych. Liczby całkowite (ang. *integer*) kolejne kombinacje bitów przeznaczają do reprezentowania kolejnych wartości: 0, 1, 2, … Liczby zmiennoprzecinkowe natomiast dzielą cały ciąg bitów na trzy części. Przykład pokazano na Rysunku 1. Format danych, którego tutaj używamy dla przykładu, to liczba zmiennoprzecinkowa 32-bitowa nazywana liczbą „pojedynczej" precyzji (ang. *single precision*), która przeznacza 1 bit na znak, 8 bitów na wykładnik i 23 bity na mantysę.



Rysunek 1. Dekodowanie liczby zmiennoprzecinkowej

Zaczynając od najmłodszych bitów, **mantysa** (ang. *mantissa*, nazywana też *significand*, pokazana tu na zielono) koduje cyfry po przecinku. Liczba jest znormalizowana tak, aby przed przecinkiem znajdowała się pojedyncza jedynka. Toteż możemy tę jedynkę pozostawić w domyśle i nie zapisywać jej w pamięci, zyskując tym samym jeden bit precyzji. (Wyjątkiem są liczby zdenormalizowane, w których ta domyślna jedynka nie występuje. Wspomnimy o nich jeszcze, omawiając wartości specjalne.)

Tak zakodowana liczba leży w zakresie od 1 do 2. Aby ją zwiększyć lub zmniejszyć, przesuwając przecinek, kolejna część koduje **wykładnik** (ang. *exponent*, pokazany tu na czerwono) dla potęgi dwójki, przez którą liczba zostaje pomnożona. Mając do dyspozycji 8 bitów,

GROOVEBOX: PROGRAMOWANIE SPOTYKA MUZYKĘ • STREFA CTF • Z ARCHIWUM CVE

INDEX: 285358

www.programistamag.pl

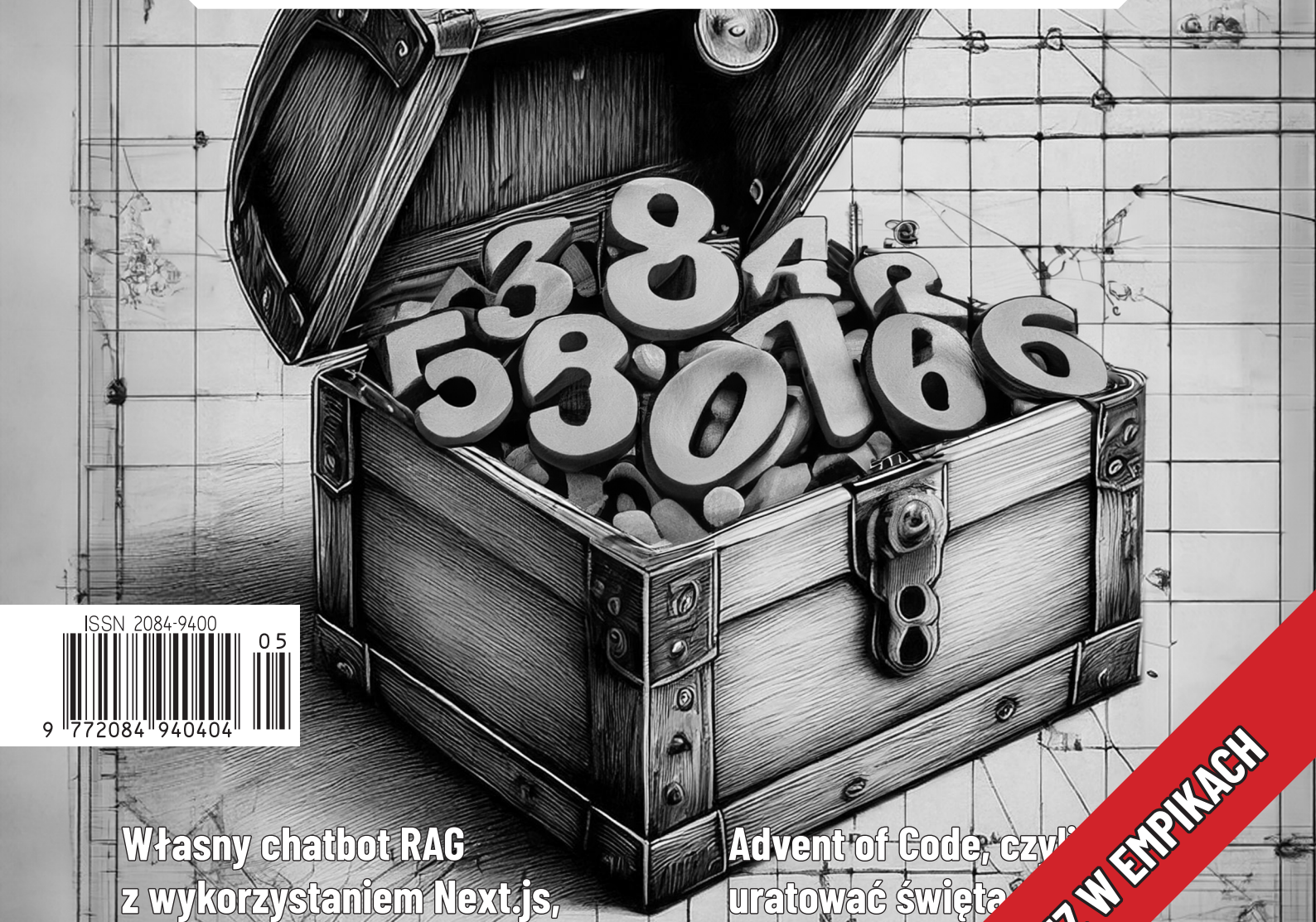
Magazyn programistów i liderów zespołów IT

programista

5/2024 (115)

Cena 29.90 zł (w tym VAT 8%)

TAJEMNICE LICZB ZMIENNOPRZECINKOWYCH



ISSN 2084-9400
05
9 772084 940404

Własny chatbot RAG z wykorzystaniem Next.js, OpenAI i MongoDB

Advent of Code, czyli uratować święta pomocy progr a

Generative AI: technologiczna bańka spekulacyjna?

Szcześ eloper to pro y developer

NOWY NUMER JUŻ W EMPIKACH

interpretowanych jako liczba całkowita, wykładnik byłby w zakresie 0…255. Jednakże stosuje się do niego przesunięcie (ang. *bias*), które w przypadku tego typu 32-bitowego wynosi 127. Przesunięcie to jest odejmowane od wartości wykładnika, aby móc zakodować zarówno wartości dodatnie, jak i ujemne. Wartość wykładnika zawierająca same zera lub same jedynki jest zarezerwowana na wartości specjalne, więc pozostaje nam zakres wykładnika [1…254] – 127 = [–126…127].

Wreszcie, najstarszy bit to **bit znaku** (ang. *sign bit*, oznaczony na niebiesko), który pozwala na zapisywanie liczb ujemnych. Pokazana na Rysunku 1 liczba -1 podniesiona do potęgi s to nic innego, jak mądry sposób na zapisanie prostej zasady, że bit ten odpowiada na pytanie: „Czy jest minus?" Kiedy wartością bitu jest 0, minusa nie ma, więc liczba jest dodatnia. Kiedy bit jest ustawiony na 1, wówczas dopisujemy minus i liczba jest ujemna.

Zauważmy, że mając osobny bit reprezentujący znak, liczby zmiennoprzecinkowe są symetryczne względem zera. Aby zanegować liczbę, wystarczy zanegować jej najstarszy bit. Aby obliczyć wartość bezwzględną, wystarczy wyzerować ten bit. Działa to inaczej, niż w przypadku liczb całkowitych ze znakiem, które stosują notację „uzupełnienia do dwóch" (U2). W liczbach całkowitych ze znakiem najstarszy bit również jest ustawiony wtedy, kiedy liczba jest ujemna, jednak ich zanegowanie nie jest już tak trywialne.

Z takiej budowy liczb zmiennoprzecinkowych wynika jeszcze jedna ich ciekawa właściwość: zmieniając tylko bit znaku, możemy uzyskać dwie różne reprezentacje zera, nazywane +0 i -0. Takie wartości faktycznie istnieją i mogą nieść pewną informację, np. czy liczba, która po obliczeniach stała się zbyt mała i musiała zostać zapisana jako 0, była dodatnia, czy ujemna. Ponieważ jednak te dwie wartości przy porównaniu są sobie równe, to praktycznie nigdy nie musimy się tym przejmować.

WSPARCIE SPRZĘTOWE

Liczby zmiennoprzecinkowe, przy całej swojej skomplikowanej budowie, nie miałyby większego sensu w programowaniu, gdyby obliczenia na nich nie były wspierane w sposób sprzętowy, aby mogły być wykonywane szybko. Takiego wsparcia w dawnych czasach dostarczał osobny, instalowany opcjonalnie koprocesor matematyczny: *Floating Point Unit* (FPU), jak układ 8087 w dawnych pecetach. Obecnie jednak procesory (CPU), jak i układy graficzne (GPU) w naszych komputerach mają wbudowaną obsługę takich liczb.

Jeśli chodzi o wspierane sprzętowo typy danych, to najpopularniejsze są liczby zmiennoprzecinkowe 32-bitowe pojedynczej precyzji (ang. *single*) oraz 64-bitowe podwójnej precyzji (ang. *double*). Ich możliwości zaprezentowano w Tabeli 1, której rozszerzoną wersję można też znaleźć online w [1]. Ich dostępność i nazwa bywa różna w różnych językach programowania. Na przykład języki C i C++ definiują typy `float` i `double`. Ich szerokość nie jest co prawda ściśle określona, ale na większości platform odpowiadają one wspomnianym formatom 32b i 64b. Niektóre języki skryptowe natomiast (w tym JavaScript, Lua) oferują tylko jeden typ liczbowy, który implementowany jest za pomocą typu `double` i który może być używany nawet do zapisywania liczb całkowitych. Nazwy typów zmiennoprzecinkowych w różnych językach programowania podsumowuje Tabela 2. Niektóre języki oferują także typy o jeszcze większej precyzji, jak `long double` w C/C++ czy `decimal` w C#, nie pokazane w tabeli.

Nazwa	Pojedynczej precyzji, *single*	Podwójnej precyzji, *double*
Bitów: suma = znak + wykładnik + mantysa	32 = 1 + 8 + 23	64 = 1 + 11 + 52
Wykładnik: przesunięcie, zakres	127, -126…127	1023, -1022…1023
Precyzja – dziesiętne cyfry znaczące	7.22 (6…9)	15.95 (15…17)
Najmniejsza liczba zdenormalizowana	$2^{-149} \approx 1.40 \times 10^{-45}$	$2^{-1074} \approx 4.94 \times 10^{-324}$
Najmniejsza liczba znormalizowana	$2^{-126} \approx 1.18 \times 10^{-38}$	$2^{-1022} \approx 2.23 \times 10^{-308}$
Następna wartość po 1	$1 + 2^{-23} \approx 1.00000012$	$1 + 2^{-52} \approx 1.00000000000000022$
Liczby całkowite reprezentowane dokładnie	$0…2^{24} = 16\ 777\ 216$	$0…2^{53} = 9\ 007\ 199\ 254\ 740\ 992$
Maksimum	$(2 - 2^{-23}) \times 2^{127} \approx 3.40 \times 10^{38}$	$(2 - 2^{-52}) \times 2^{1023} \approx 1.80 \times 10^{308}$

Tabela 1. Możliwości liczb 32- i 64-bitowych

Język	32-bit	64-bit
C, C++*	`float`	`double`
C#	`float`	`double`
Java	`float`	`double`
Delphi / Object Pascal	`Single`	`Double`
Python*		`float`
JavaScript		`Number`
Lua**		`number`

* Zazwyczaj, zależnie od implementacji.
** Jedyny obsługiwany typ liczbowy.

Tabela 2. Typy zmiennoprzecinkowe w językach programowania

CZY TE LICZBY SĄ NIEDOKŁADNE?

Popularny pogląd mówi, że liczby zmiennoprzecinkowe są niedokładne i w związku z tym nie należy ich nigdy porównywać operatorem ==. Stosowanie takiej zasady pomaga uniknąć wielu błędów, ale jest to pewne uproszczenie. Obliczenie 2.0 + 2.0 zawsze zwróci 4.0, a nigdy 3.99999. Przyjrzyjmy się zatem bliżej tematowi precyzji liczb zmiennoprzecinkowych, aby lepiej go zrozumieć.

To oczywiste, że liczby zmiennoprzecinkowe są tylko pewnym przybliżeniem liczb rzeczywistych z matematyki, a zakodowane na określonej liczbie bitów z natury mają pewną skończoną precyzję. Kiedy mówimy o precyzji, mamy na myśli najmniejsze różnice w wartości, które możemy zapisać i rozróżnić w określonym typie danych. W przypadku liczb całkowitych sprawa jest prosta: precyzja zawsze wynosi 1 w całym zakresie, więc po wartości 5 następna możliwa do zapisania wartość to 6, dalej 7 itd.

Można też sobie wyobrazić format obsługujący ułamki, ale stałoprzecinkowy, na przykład używający starszych 8 bitów do kodowania cyfr binarnych przed przecinkiem, a młodszych 8 bitów do kodowania cyfr po przecinku. Takie formaty stałoprzecinkowe nie są obsługiwane sprzętowo przez typowe procesory w komputerach, więc