## Precyzja i zakres

#### Precyzja (Precision)

**Precyzja** odnosi się do **liczby cyfr znaczących**, które typ danych może dokładnie reprezentować. Mówiąc prościej, to miara tego, jak **szczegółowo i dokładnie** liczba może być przechowywana.

Wyobraź sobie linijkę. Precyzja mówi o tym, jak drobne są podziałki na tej linijce – czy możesz zmierzyć coś z dokładnością do milimetra, dziesiątej części milimetra, czy tylko do centymetra.

* **W kontekście liczb zmiennoprzecinkowych (float, double):** Precyzja jest często mierzona w liczbie **cyfr dziesiętnych**, które mogą być reprezentowane bez utraty dokładności. Na przykład:
  + Typ float (pojedyncza precyzja) ma zazwyczaj około **6-9 cyfr dziesiętnych precyzji**. Oznacza to, że jeśli masz liczbę 123.456789, to pierwsze 6-7 cyfr jest przechowywanych dokładnie, a dalsze cyfry mogą być zaokrąglane lub tracone.
  + Typ double (podwójna precyzja) ma około **15-17 cyfr dziesiętnych precyzji**, co pozwala na znacznie dokładniejsze obliczenia.
  + Typ decimal (w C#) ma aż **28-29 cyfr dziesiętnych precyzji**, dzięki czemu jest idealny do obliczeń finansowych.
* **Dlaczego to ważne?** Niska precyzja może prowadzić do **błędów zaokrągleń**, które mogą kumulować się w długich ciągach obliczeń, dając niedokładne wyniki.

#### Zakres (Range)

**Zakres** odnosi się do **przedziału wartości**, od najmniejszej do największej, które typ danych może pomieścić. Mówiąc prościej, to miara tego, jak **duże lub jak małe** liczby mogą być przechowywane.

Wracając do analogii z linijką, zakres mówi o tym, **jak długa** jest ta linijka – czy pozwala zmierzyć odległość od 0 do 10 cm, czy od 0 do 100 metrów.

* **W kontekście liczb zmiennoprzecinkowych:** Zakres jest zazwyczaj wyrażany w potęgach liczby 10. Na przykład:
  + Typ float może przechowywać wartości od około ±1.5×10−45 do ±3.4×1038.
  + Typ double ma znacznie większy zakres, od około ±5.0×10−324 do ±1.7×10308.
  + Typ decimal w C# ma zakres od ±1.0×10−28 do ±7.9228×1028.
* **Dlaczego to ważne?** Jeśli liczba, którą próbujesz przechowywać, wykracza poza zakres danego typu danych, może dojść do **przepełnienia (overflow)** (liczba jest za duża) lub **niedopełnienia (underflow)** (liczba jest za mała, bliska zeru, by być reprezentowana), co skutkuje nieprawidłowymi wynikami lub błędami programu.

#### Kluczowa różnica

Możesz mieć typ danych o **dużej precyzji**, ale **małym zakresie** (np. decimal jest bardzo precyzyjny, ale jego zakres jest mniejszy niż double). Z drugiej strony, możesz mieć typ danych o **dużym zakresie**, ale **niższej precyzji** (np. float ma spory zakres, ale gorszą precyzję niż double).

Wybór odpowiedniego typu danych zależy od tego, co jest ważniejsze dla Twoich obliczeń: czy potrzebujesz bardzo dokładnie reprezentować liczby, czy raczej przechowywać bardzo duże/małe wartości.

## Liczba stałoprzecinkowa

Liczba stałoprzecinkowa (ang. **fixed-point number**) to sposób reprezentowania liczb rzeczywistych w komputerze, gdzie **liczba cyfr po przecinku jest zawsze stała i z góry określona**. W przeciwieństwie do liczb zmiennoprzecinkowych (jak float czy double), które mają zmienną precyzję i zakres, liczby stałoprzecinkowe oferują stałą precyzję i często są używane, gdy potrzebna jest bardzo dokładna kontrola nad precyzją, np. w aplikacjach finansowych.

Wyobraź sobie, że masz liczbę, powiedzmy, 123.45. W reprezentacji stałoprzecinkowej, jeśli zdecydujesz, że zawsze masz dwie cyfry po przecinku, ta liczba zawsze będzie miała dwie cyfry po przecinku.

**Jak to działa?**

W systemach komputerowych liczby stałoprzecinkowe są często przechowywane jako liczby całkowite. Przecinek jest **niejawny** – to znaczy, że nie jest fizycznie przechowywany, ale programista wie, gdzie on się znajduje.

Na przykład, jeśli mamy liczbę całkowitą 12345 i wiemy, że nasz "punkt stały" znajduje się dwie pozycje od prawej strony, to interpretujemy tę liczbę jako 123.45.

## Liczba zmiennoprzecinkowa

Liczba zmiennoprzecinkowa (ang. **floating-point number**) to sposób reprezentowania liczb rzeczywistych w komputerze, który pozwala na przechowywanie bardzo dużych i bardzo małych wartości, a także wartości ułamkowych. W przeciwieństwie do liczb stałoprzecinkowych (które mają stałą liczbę cyfr po przecinku), w liczbach zmiennoprzecinkowych **przecinek może "pływać"** (czyli zmieniać swoją pozycję), co pozwala na dynamiczne dostosowanie precyzji w zależności od wielkości liczby.

**Jak to działa?**

Liczby zmiennoprzecinkowe są przechowywane w komputerze w formie przypominającej notację naukową. Składają się zazwyczaj z trzech części:

1. **Mantysa (lub znacząca część)**: To są cyfry, które tworzą właściwą wartość liczby.
2. **Podstawa**: Zazwyczaj jest to 2 (ponieważ komputery działają w systemie binarnym).
3. **Wykładnik**: Określa, gdzie "pływa" przecinek, czyli o ile pozycji mantysa ma być przesunięta.

Na przykład, liczbę 123.45 można by zapisać w notacji naukowej jako 1.2345×102. W komputerze używana jest podobna koncepcja, ale z podstawą 2.

## Liczba zmiennoprzecinkowa pojedynczej precyzji

Liczba zmiennoprzecinkowa pojedynczej precyzji to typ danych używany do reprezentowania liczb rzeczywistych (czyli takich z częścią ułamkową) w pamięci komputera. Jest to standardowy sposób kodowania liczb z zmiennym przecinkiem, co oznacza, że przecinek dziesiętny może "pływać" (zmieniać swoje położenie) w zależności od wartości liczby.

**Jak działa liczba zmiennoprzecinkowa?**

Liczby zmiennoprzecinkowe (float, double) są zazwyczaj reprezentowane w formacie, który przypomina notację naukową. Składają się z trzech głównych części:

1. **Bit znaku (Sign Bit):** Określa, czy liczba jest dodatnia (0) czy ujemna (1).
2. **Wykładnik (Exponent):** Przechowuje informację o skali liczby, czyli o tym, gdzie znajduje się przecinek. Działa jak potęga w notacji naukowej (np. 103).
3. **Mantysa (Mantissa/Significand):** Zawiera znaczące cyfry liczby, czyli jej precyzję.

**Pojedyncza precyzja (Single-Precision)**

Liczba zmiennoprzecinkowa pojedynczej precyzji zwykle zajmuje **32 bity (4 bajty)** w pamięci komputera. Jest to zgodne ze standardem **IEEE 754**, który jest powszechnie stosowany.

**W C#** taką liczbą jest typ **float**.

Oto, jak 32 bity są zazwyczaj podzielone:

* **1 bit** na znak.
* **8 bitów** na wykładnik.
* **23 bity** na mantysę.

To daje około **6-9 cyfr dziesiętnych precyzji**. Oznacza to, że po tych 6-9 cyfrach, dokładność może zacząć spadać z powodu zaokrągleń.

**Kiedy używać float (pojedynczej precyzji)?**

* **Oszczędność pamięci:** Gdy potrzebujesz przechowywać wiele liczb zmiennoprzecinkowych, a pamięć jest ograniczona.
* **Gry i grafika komputerowa:** Wiele obliczeń graficznych (pozycje wierzchołków, kolory, wektory) używa float ze względu na wydajność i wystarczającą precyzję.
* **Obliczenia, gdzie wysoka precyzja nie jest krytyczna:** Na przykład, jeśli obliczasz odległości, prędkości, czy inne pomiary, gdzie niewielkie zaokrąglenia nie wpłyną znacząco na wynik.

**Ograniczenia i ryzyka**

* **Błędy zaokrągleń:** Ponieważ liczby zmiennoprzecinkowe (zwłaszcza binarne, jak float i double) nie mogą dokładnie reprezentować wszystkich ułamków dziesiętnych (np. 0.1 w systemie binarnym jest liczbą okresową), mogą wystąpić subtelne błędy zaokrągleń.
* **Utrata precyzji:** Przy bardzo dużych lub bardzo małych liczbach, a także przy skomplikowanych obliczeniach, możesz stracić część precyzji.
* **Porównywanie:** Bezpośrednie porównywanie liczb float (np. a == b) jest ryzykowne ze względu na błędy zaokrągleń. Zamiast tego porównuje się je z pewną tolerancją (np. Math.Abs(a - b) < epsilon).

Dla większej precyzji (około 15-17 cyfr dziesiętnych) używa się **liczby zmiennoprzecinkowej podwójnej precyzji** (double-precision), która w C# to typ **double** i zajmuje 64 bity.

## Liczba zmiennoprzecinkowa podwójnej precyzji

Liczba zmiennoprzecinkowa podwójnej precyzji to rozszerzona wersja liczby zmiennoprzecinkowej pojedynczej precyzji. Podobnie jak jej "mniejszy brat", służy do reprezentowania liczb rzeczywistych z częścią ułamkową, ale oferuje znacznie większą dokładność i zakres.

**Jak działa podwójna precyzja?**

Standardowo, liczba zmiennoprzecinkowa podwójnej precyzji zajmuje **64 bity (8 bajtów)** w pamięci komputera. Jest to również zgodne ze standardem **IEEE 754**.

**W C#** taką liczbą jest typ **double**.

Podział 64 bitów wygląda zazwyczaj tak:

* **1 bit** na znak (czy liczba jest dodatnia czy ujemna).
* **11 bitów** na wykładnik (określa skalę liczby, czyli położenie przecinka).
* **52 bity** na mantysę (znaczące cyfry liczby).

Dzięki większej liczbie bitów na mantysę, typ double zapewnia około **15-17 cyfr dziesiętnych precyzji**. To ponad dwukrotnie więcej niż typ float, co znacznie zmniejsza ryzyko błędów zaokrągleń w skomplikowanych obliczeniach. Większa liczba bitów na wykładnik przekłada się również na znacznie większy zakres wartości, które można reprezentować.

**Kiedy używać double (podwójnej precyzji)?**

Double jest domyślnym typem zmiennoprzecinkowym w wielu językach programowania, w tym w C#, ponieważ oferuje dobry balans między precyzją, zakresem a wydajnością. Używaj go, gdy:

* **Potrzebujesz wysokiej precyzji:** W obliczeniach naukowych, inżynierskich, symulacjach fizycznych czy wszelkich zadaniach, gdzie dokładność jest kluczowa.
* **Zakres wartości jest duży:** Gdy pracujesz z bardzo dużymi lub bardzo małymi liczbami.
* **Standardowe obliczenia:** Jeśli nie masz specyficznych wymagań dotyczących oszczędności pamięci lub ekstremalnej wydajności (która czasami jest wymagana w grafice 3D), double jest zazwyczaj bezpiecznym i dobrym wyborem.

**Porównanie z float i decimal**

* **float (pojedyncza precyzja):** Mniejsza precyzja (około 6-9 cyfr), mniejszy zakres, szybszy, zajmuje mniej pamięci. Dobry do gier i grafiki.
* **double (podwójna precyzja):** Wyższa precyzja (około 15-17 cyfr), większy zakres, nieco wolniejszy niż float, ale nadal wydajny (często wspierany sprzętowo). Jest to typ ogólnego przeznaczenia dla większości obliczeń zmiennoprzecinkowych.
* **decimal:** Najwyższa precyzja dla liczb dziesiętnych (28-29 cyfr), brak błędów reprezentacji dla ułamków dziesiętnych (bo jest binarnie kodowaną liczbą dziesiętną, a nie binarną), najmniejszy zakres, najwolniejszy (obliczenia software'owe). Idealny do zastosowań finansowych.

Pamiętaj, że nawet double ma swoje ograniczenia co do dokładnej reprezentacji wszystkich ułamków dziesiętnych (ponieważ nadal bazuje na systemie binarnym). Dlatego, jeśli precyzyjne obliczenia dziesiętne (np. walutowe) są kluczowe, nadal preferowanym typem w C# będzie decimal.

## Decimal

To bardzo specyficzny i niezwykle ważny typ danych w .NET, szczególnie w kontekście finansów i wszelkich obliczeń wymagających absolutnej precyzji dziesiętnej.

**Typ decimal w C#**

Typ decimal w C# jest **128-bitową (16-bajtową) dziesiętną liczbą zmiennoprzecinkową**, która oferuje znacznie wyższą precyzję i mniejsze błędy zaokrągleń dla ułamków dziesiętnych w porównaniu do typów float i double.

**Główny cel:** decimal został zaprojektowany, aby eliminować problemy z dokładnością, które pojawiają się w obliczeniach finansowych i walutowych przy użyciu liczb zmiennoprzecinkowych binarnych (float i double). Te binarne typy mogą nie być w stanie dokładnie reprezentować niektórych ułamków dziesiętnych (np. 0.1), co prowadzi do subtelnych błędów zaokrągleń. decimal rozwiązuje ten problem, operując na podstawie dziesiętnej.

**Kluczowe cechy typu decimal:**

1. **Reprezentacja dziesiętna (nie binarna):**
   * W przeciwieństwie do float i double, które reprezentują liczby w systemie binarnym (podstawa 2), decimal reprezentuje liczby w systemie dziesiętnym (podstawa 10).
   * Oznacza to, że ułamki dziesiętne, takie jak 0.1, 0.25, czy 0.01, są przechowywane **dokładnie**, bez błędów wynikających z konwersji między systemami liczbowymi. To jest najważniejsza cecha decimal i powód jego istnienia.
2. **Wysoka precyzja:**
   * Oferuje **28-29 cyfr znaczących precyzji**. To znacznie więcej niż float (6-9 cyfr) czy double (15-17 cyfr).
   * Ta wysoka precyzja jest kluczowa w zastosowaniach finansowych, gdzie nawet najmniejsze zaokrąglenia mogą mieć poważne konsekwencje.
3. **Zakres:**
   * Zakres wartości decimal wynosi od około ±1.0×10−28 do ±7.9228×1028.
   * Choć jest to mniejszy zakres niż dla double, jest on w zupełności wystarczający dla większości zastosowań finansowych i biznesowych.
4. **Wewnętrzna struktura:**
   * Wewnętrznie decimal przechowuje liczbę jako **liczbę całkowitą 96-bitową** oraz **współczynnik skali** (wykładnik), który określa, ile cyfr znajduje się po przecinku dziesiętnym (od 0 do 28). To właśnie ten współczynnik skali pozwala mu działać jak zmiennoprzecinkowa liczba dziesiętna.
5. **Wydajność:**
   * Operacje na typie decimal są **wolniejsze** niż na float i double. Wynika to z faktu, że obliczenia decimal są zazwyczaj wykonywane programowo (przez oprogramowanie .NET Runtime), podczas gdy float i double mogą korzystać z bezpośredniego wsparcia sprzętowego procesora (jednostka FPU).
   * W większości aplikacji biznesowych i finansowych różnica w wydajności jest niezauważalna, a dokładność jest priorytetem.

**Kiedy używać decimal?**

* **Obliczenia finansowe:** Waluty, ceny, odsetki, podatki, bilanse. To podstawowe zastosowanie.
* **Aplikacje biznesowe:** Wszędzie tam, gdzie dokładne kwoty są przetwarzane.
* **Obliczenia wymagające dokładnych ułamków dziesiętnych:** Np. gdy musisz precyzyjnie reprezentować 0.1 czy 0.001.
* **Unikanie błędów zaokrągleń:** Gdy nawet minimalne błędy zaokrągleń są niedopuszczalne.

## Wewnętrzna struktura float i double

Zrozumienie wewnętrznej struktury float i double jest kluczowe do pojęcia, dlaczego pojawiają się błędy zaokrągleń w obliczeniach zmiennoprzecinkowych. Oba te typy są zgodne ze standardem **IEEE 754**, który definiuje sposób reprezentacji liczb zmiennoprzecinkowych binarnych.

**Podstawowa idea reprezentacji zmiennoprzecinkowej**

Liczba zmiennoprzecinkowa jest przechowywana w formie przypominającej **notację naukową** (np. 1.23×105), ale w systemie binarnym (podstawa 2). Ogólna forma to:

A black text with black text

AI-generated content may be incorrect.

Każda liczba float (32 bity) i double (64 bity) jest podzielona na trzy główne pola bitowe:

1. **Bit znaku (Sign Bit - S):**
   * Zawsze 1 bit.
   * 0 oznacza liczbę dodatnią.
   * 1 oznacza liczbę ujemną.
2. **Wykładnik (Exponent - E):**
   * Przechowuje informację o potędze podstawy 2, która skaluje mantysę.
   * Nie jest to "prawdziwy" wykładnik, lecz **przesunięty (biased)** wykładnik. Oznacza to, że do rzeczywistego wykładnika dodawana jest stała wartość (tzw. "bias"), aby umożliwić reprezentowanie zarówno dodatnich, jak i ujemnych wykładników bez użycia dodatkowego bitu znaku dla samego wykładnika.
   * Wartość biasu zależy od typu (float/double).
   * Pola wykładnika złożone wyłącznie z zer lub wyłącznie z jedynek są zarezerwowane dla wartości specjalnych (zero, nieskończoność, NaN).
3. **Mantysa (Mantissa / Significand / Fraction - M):**
   * Przechowuje znaczące cyfry liczby.
   * W formacie znormalizowanym (dla większości liczb), przyjmuje się, że pierwsza cyfra mantysy (bit przed przecinkiem binarnym) to zawsze 1. Ponieważ jest ona zawsze 1, nie jest faktycznie przechowywana w bitach mantysy. Jest to tzw. **ukryty bit (implied leading 1)**. Dzięki temu zyskuje się jeden bit precyzji za darmo.

**Struktura float (pojedyncza precyzja - IEEE 754 Single-Precision)**

* **Rozmiar:** 32 bity (4 bajty)
* **Podział bitów:**
  + Bit 31: **Znak (S)** (1 bit)
  + Bity 30-23: **Wykładnik (E)** (8 bitów)
    - Bias dla float wynosi **127**.
    - Rzeczywisty wykładnik = wartość\_wykładnika\_z\_bitów - 127.
  + Bity 22-0: **Mantysa (M)** (23 bity)
    - Reprezentuje ułamkową część mantysy (po ukrytej 1).
* **Wartość liczby:**



**Struktura double (podwójna precyzja - IEEE 754 Double-Precision)**

* **Rozmiar:** 64 bity (8 bajtów)
* **Podział bitów:**
  + Bit 63: **Znak (S)** (1 bit)
  + Bity 62-52: **Wykładnik (E)** (11 bitów)
    - Bias dla double wynosi **1023**.
    - Rzeczywisty wykładnik = wartość\_wykładnika\_z\_bitów - 1023.
  + Bity 51-0: **Mantysa (M)** (52 bity)
    - Reprezentuje ułamkową część mantysy (po ukrytej 1).
* **Wartość liczby:**



**Przykład (uproszczony)**

Weźmy prosty przykład liczby 6.25 jako float:

1. **Znak:** Liczba jest dodatnia, więc S = 0.
2. **Konwersja do binarnego:**



1. **Normalizacja (zmienna forma naukowa):**



1. **Wykładnik:**
   * Prawdziwy wykładnik to 2.
   * Przesunięty wykładnik (dla float, bias = 127) to 2 + 127 = 129.
   * 129 w systemie binarnym to 10000001.
2. **Mantysa:**
   * Po normalizacji mamy 1.10012​. Ukryty bit to 1.
   * Część ułamkowa mantysy to 1001.
   * Musimy wypełnić 23 bity, więc dodajemy zera na końcu: 10010000000000000000000.

**Końcowa reprezentacja 6.25 jako float (32 bity):**

|  |  |  |
| --- | --- | --- |
| **S (1 bit)** | **Wykładnik (8 bitów)** | **Mantysa (23 bity)** |
| 0 | 10000001 | 10010000000000000000000 |