5. Reprezentacja liczb rzeczywistych w komputerze

Rozwiązaliśmy już wiele problemów i poznaliśmy różne techniki algorytmiczne. Dotychczas posługiwaliśmy się jednak tylko liczbami całkowitymi oraz znakami, które w komputerze są reprezentowane przez liczby całkowite. W tym temacie dowiesz się, jak komputery radzą sobie z liczbami rzeczywistymi.

Cele lekcji

- Znajdziesz rozwinięcie binarne ułamka właściwego.
- Poznasz reprezentację stało- i zmiennoprzecinkową liczb rzeczywistych.
- Dowiesz się, czym jest normalizacja reprezentacji zmiennoprzecinkowej.
- Wykonasz działania arytmetyczne na liczbach zmiennoprzecinkowych.

5.1. Rozwinięcie binarne ułamka właściwego

System dwójkowy, podręcznik *Informatyka na czasie 2. Zakres rozszerzony*, s. 28–29

W części drugiej podręcznika, przy okazji omawiania **systemu dwójkowego**, znajdowaliśmy reprezentację binarną liczby całkowitej dodatniej zapisanej w systemie dziesiętnym. Teraz wyznaczymy rozwinięcie binarne ułamka zapisanego w systemie dziesiętnym.

Załóżmy, że mamy dodatni ułamek właściwy. Na początku mnożymy wyjściowy ułamek przez 2. Część całkowita wyniku (może nią być tylko 1 lub 0) odpowiada pierwszej cyfrze szukanego rozwinięcia binarnego. Część ułamkową wyniku wykorzystamy do wyznaczenia kolejnej cyfry rozwinięcia binarnego – postępujemy z nią w ten sam sposób co z wyjściowym ułamkiem. Aby znaleźć następne cyfry, czynności powtarzamy. Działanie algorytmu na przykładzie ułamka $\frac{5}{8}$ przedstawia tabela 5.1. Dla tego ułamka algorytm kończy działanie, gdy licznik części ułamkowej jest równy 0.

Dobra rada

Pamiętaj, że w algorytmie zamiany liczby całkowitej dodatniej z postaci dziesiętnej na binarną wykorzystuje się dzielenie przez 2. Reszta z dzielenia danej liczby przez 2 jest jedną z szukanych cyfr binarnych, a część całkowita stanowi podstawę wyznaczania w ten sam sposób kolejnej cyfry binarnej. Algorytm wyznacza cyfry binarne od ostatniej do pierwszej, a kończy działanie, gdy część całkowita jest równa zero.

Wykonywane działanie	Część całkowita wyniku	Część ułamkowa wyniku	Zapis binarny
–	0	$\frac{5}{8}$	0
$\frac{5}{8} \cdot 2$	1	$\frac{2}{8}$	0,1
$\frac{2}{8} \cdot 2$	0	$\frac{4}{8}$	0,10
$\frac{4}{8} \cdot 2$	1	$\frac{0}{8}$	0,101

Tabela 5.1. Kolejne etapy wyznaczania rozwinięcia binarnego ułamka $\frac{5}{8}$

Jednak nie każdy ułamek właściwy, który ma skończone rozwinięcie dziesiętne, ma też skończone rozwinięcie binarne. Przykładem jest ułamek $\frac{1}{10}$. Tabela 5.2 ilustruje wyznaczanie jego rozwinięcia binarnego.

Dobra rada

Pamiętaj, że ułamek właściwy (jako liczba wymierna) w systemie dziesiętnym ma rozwinięcie dziesiętne skończone albo nieskończone okresowe.

Wykonywane działanie	Część całkowita wyniku	Część ułamkowa wyniku	Zapis binarny
–	0	$\frac{1}{10}$	0
$\frac{1}{10} \cdot 2$	0	$\frac{2}{10}$	0,0
$\frac{2}{10} \cdot 2$	0	$\frac{4}{10}$	0,00
$\frac{4}{10} \cdot 2$	0	$\frac{8}{10}$	0,000
$\frac{8}{10} \cdot 2$	1	$\frac{6}{10}$	0,0001
$\frac{6}{10} \cdot 2$	1	$\frac{2}{10}$	0,00011
$\frac{2}{10} \cdot 2$	0	$\frac{4}{10}$	0,000110
$\frac{4}{10} \cdot 2$	0	$\frac{8}{10}$	0,0001100
$\frac{8}{10} \cdot 2$	1	$\frac{6}{10}$	0,00011001
$\frac{6}{10} \cdot 2$	1	$\frac{2}{10}$	0,000110011
$\frac{2}{10} \cdot 2$	0	$\frac{4}{10}$	0,0001100110
...	...	...	0,0(0011)

Tabela 5.2. Poszukiwanie rozwinięcia binarnego ułamka $\frac{1}{10}$

Jak widać, licznik części ułamkowej nigdy nie będzie równy 0. Otrzymywane części całkowite i ułamkowe od pewnego miejsca się powtarzają – zaznaczone są w tabeli 5.2 różnymi odcieniami zieleni (po 4 kolejne wiersze). Wynikiem jest zatem ułamek okresowy.

Ćwiczenie 1

Znajdź rozwinięcie binarne ułamków $\frac{13}{16}$ i $\frac{2}{5}$.

Zastanówmy się, dla jakich ułamków otrzymamy rozwinięcie binarne skończone, a dla jakich – nieskończone okresowe. Weźmy liczbę binarną o skończonym rozwinięciu, np. $0{,}1011_2$, i znajdźmy jej reprezentację w systemie dziesiętnym. Aby to zrobić, postępujemy podobnie jak w przypadku liczb całkowitych – mnożymy cyfry binarne części ułamkowej liczby przez odpowiednie potęgi podstawy systemu, czyli potęgi liczby 2. W przypadku zamiany części ułamkowej wykładniki potęg będą ujemne:

$$0{,}1011_2 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = \frac{1}{2} + \frac{0}{4} + \frac{1}{8} + \frac{1}{16} = \frac{11}{16}$$

Warto wiedzieć

W systemie dziesiętnym skończone rozwinięcie dziesiętne mają te ułamki, w których w postaci nieskracalnej mianownik jest iloczynem potęgi liczby 2 i potęgi liczby 5.

Na podstawie tego przykładu można zauważyć, że skończone rozwinięcie binarne mają ułamki właściwe nieskracalne, których mianownikiem jest potęga liczby 2. Pozostałe ułamki zwykłe, także te o skończonym rozwinięciu dziesiętnym, mają okresowe rozwinięcie binarne.

Ćwiczenie 2

Znajdź reprezentację liczby $0{,}00011_2$ w systemie dziesiętnym.

Dobra rada

Jeśli chcesz znaleźć reprezentację binarną liczby, która w systemie dziesiętnym składa się z części całkowitej różnej od zera oraz części ułamkowej (np. liczby 1,25), musisz zamieniać oddzielnie część całkowitą i część ułamkową. Do zamiany części całkowitej wykorzystaj sposób omówiony w podręczniku *Informatyka na czasie 2. Zakres rozszerzony* (s. 28–29, 33–35).

Omówimy algorytmy wyznaczające rozwinięcia binarne ułamków właściwych nieskracalnych zapisanych w systemie dziesiętnym. Najpierw rozważymy przypadek, gdy mianownik ułamka jest potęgą liczby 2, a więc rozwinięcie binarne ułamka będzie skończone. Oto specyfikacja problemu oraz zapis algorytmu w pseudokodzie:

Specyfikacja

Dane: `licz`, `mian` – liczby całkowite dodatnie reprezentujące licznik i mianownik ułamka, `licz` < `mian`, `mian` = 2^n, gdzie n – liczba całkowita dodatnia, NWD(`licz`, `mian`) = 1.

Wynik: `s` – napis reprezentujący rozwinięcie binarne ułamka.

```
s ← "0,"
dopóki licz > 0 wykonuj
    licz ← licz * 2
    jeśli licz div mian = 1 to s ← s + "1"
    w przeciwnym przypadku s ← s + "0"
    licz ← licz mod mian
```

Wartością początkową wyniku `s` jest napis „0,". Mnożymy licznik przez 2. Jeśli część całkowita z dzielenia otrzymanego wyniku przez mianownik jest równa 1, to na końcu wyniku dopisujemy cyfrę 1. W przeciwnym przypadku dopisujemy cyfrę 0. Nowa wartość licznika to reszta z dzielenia licznika przez mianownik. Działania powtarzamy, dopóki licznik jest większy od 0.

Ćwiczenie 3

Napisz program znajdujący rozwinięcie binarne nieskracalnego ułamka właściwego, którego mianownik jest potęgą liczby 2.

Trudniejszym problemem jest wyznaczenie okresu rozwinięcia binarnego ułamków nieskracalnych, których mianownik nie jest potęgą liczby 2. Należy wówczas pamiętać znajdowane wartości części całkowitej i licznika części ułamkowej (mianownika nie musimy pamiętać, ponieważ jest cały czas taki sam – ułamka nie skracamy).

Ponieważ nie wiemy, jak długi będzie okres, powinniśmy użyć dynamicznej struktury danych, np. **zmiennej typu `vector`**. Po każdym mnożeniu przez 2 należy sprawdzić, czy część całkowita i licznik już wystąpiły. Jeśli tak, kończymy obliczenia. Okres rozpoczyna się od miejsca, w którym aktualnie policzone wartości części całkowitej i licznika wystąpiły po raz pierwszy. Jeśli obliczone część całkowita i licznik jeszcze nie wystąpiły, to ich wartości dopisujemy na końcu struktury danych przechowującej te wartości.

Deklaracja zmiennej typu `vector`, s. 66

Dobra rada

Zamiast typu **`vector`** możesz użyć typu **`list`**.

Oto specyfikacja problemu znajdowania rozwinięcia binarnego ułamka właściwego nieskracalnego, którego mianownik nie jest potęgą liczby 2:

Specyfikacja

Dane: `licz`, `mian` – liczby całkowite dodatnie reprezentujące licznik i mianownik ułamka, `licz` < `mian`, `mian` ≠ 2^n, gdzie n – liczba całkowita dodatnia, NWD(`licz`, `mian`) = 1.

Wynik: `s` – napis reprezentujący rozwinięcie binarne ułamka.

Przyjmijmy, że elementami zmiennej `Tab` są pary liczb całkowitych reprezentujących odpowiednio część całkowitą i licznik części ułamkowej. Niech `m` oznacza aktualną liczbę elementów zmiennej `Tab`. W zmiennej `calk` będzie pamiętana część całkowita. Oto zapis algorytmu w pseudokodzie:

```
s ← "0,"
jest_okres ← fałsz
dopóki nie jest_okres wykonuj
    licz ← licz * 2
    calk ← licz div mian
    licz ← licz mod mian
    i ← 0
    dopóki i < m oraz nie jest_okres wykonuj
        jeśli Tab[i].calk=calk oraz Tab[i].licz=licz to
            jest_okres ← prawda
        w przeciwnym wypadku
            i ← i + 1
    jeśli nie jest_okres to
        dodaj nowy element na końcu Tab
        Tab[m-1].calk ← calk
        Tab[m-1].licz ← licz
dla j ← 0, 1, ..., m - 1 wykonuj
    jeśli j = i to s ← s + "("
    jeśli Tab[j].calk = 1 to s ← s + "1"
    w przeciwnym przypadku s ← s + "0"
s ← s + ")"
```

Znalezienie okresu sygnalizuje zmienna `jest_okres`. Gdy okres zostanie znaleziony (za jego poszukiwanie odpowiada wewnętrzna pętla `dopóki`), wówczas zmienna `i` zapamięta miejsce rozpoczęcia okresu.

Budując napis reprezentujący rozwinięcie binarne ułamka, przeglądamy wszystkie elementy zmiennej Tab. Bezpośrednio przed elementem o indeksie i dopisujemy nawias otwierający, który wskazuje rozpoczęcie okresu. Na końcu dopisujemy nawias zamykający.

W programie realizującym omówiony algorytm części całkowite i liczniki będziemy pamiętać w zmiennej typu **vector**. Każdy element tej zmiennej będzie przechowywał parę liczb: część całkowitą i licznik wyliczone w jednym kroku algorytmu. Do przechowania takiej pary wykorzystamy strukturę o nazwie wymierna. Oto jej definicja:

```
struct wymierna
{
    int calk;
    int licz;
};
```

Kod źródłowy funkcji main może być następujący:

Fragment kodu źródłowego programu znajdującego rozwinięcie binarne ułamka właściwego nieskracalnego, którego mianownik nie jest potęgą liczby 2 – funkcja main

```
1.  int main()
2.  {
3.      int licz, mian, i, j;
4.      cout<<"Licznik: "; cin>>licz;
5.      cout<<"Mianownik: "; cin>>mian;
6.      string s="0,";
7.      bool jest_okres=false;
8.      wymierna w;
9.      vector<wymierna> Tab;
10.     while (!jest_okres)
11.     {
12.         licz=licz*2;
13.         w.calk=licz/mian;
14.         licz=licz%mian;
15.         w.licz=licz;
16.         i=0;
17.         while (i<Tab.size() && !jest_okres)
18.             if (Tab[i].calk==w.calk && Tab[i].licz==w.licz)
19.                 jest_okres=true;
20.             else i++;
21.         if (!jest_okres) Tab.push_back(w);
22.     }
23.     for (j=0;j<Tab.size();j++)
24.     {
25.         if (j==i) s=s+'(';
26.         if (Tab[j].calk==1) s=s+'1';
27.         else s=s+'0';
28.     }
29.     s=s+')';
30.     cout<<"Rozwiniecie binarne: "<<s;
31.     return 0;
32. }
```


W linii 9 zadeklarowana jest zmienna Tab typu **vector**, składająca się z elementów typu wymierna. Pętla w liniach 10–22 poszukuje okresu rozwinięcia binarnego. W zmiennej w typu wymierna pamiętane są policzone ostatnio część całkowita i licznik części ułamkowej (linie 13–15). Pętla w liniach 17–20 sprawdza, czy ta para wartości już wystąpiła. Jeśli tak, to w zmiennej i zapamiętywany jest indeks elementu wektora, w którym zapisane jest pierwsze wystąpienie tej pary. Jeśli para jeszcze nie wystąpiła, to wartość zmiennej w dodawana jest jako nowy element na końcu wektora (linia 21). Pętla w liniach 23–28 przegląda wektor i zapisuje w zmiennej s rozwinięcie binarne ułamka. Dla elementu o indeksie i (początek okresu) dodaje nawias otwierający (linia 25). W linii 29 dodawany jest nawias zamykający (kończący okres).

Rysunek 5.1 przedstawia efekt wykonania programu dla ułamka $\frac{1}{10}$.

```
Licznik: 1
Mianownik: 10
Rozwiniecie binarne: 0,0(0011)
```

Rys. 5.1. Efekt wykonania programu dla ułamka $\frac{1}{10}$

Dobra rada

Dynamiczne struktury danych stosuj wtedy, gdy nie znasz liczby elementów, które program ma pamiętać.

Ćwiczenie 4

Napisz program znajdujący rozwinięcie binarne ułamka właściwego nieskracalnego, którego mianownik nie jest potęgą liczby 2.

Zapamiętaj

Skończone rozwinięcie binarne mają tylko ułamki właściwe nieskracalne, których mianownikiem jest potęga liczby 2. Pozostałe ułamki zwykłe, także te o skończonym rozwinięciu dziesiętnym, mają okresowe rozwinięcie binarne.

5.2. Reprezentacja stało- i zmiennoprzecinkowa liczb rzeczywistych

Liczby rzeczywiste, podobnie jak liczby całkowite, są pamiętane w komputerze w postaci binarnej. Na ich reprezentację przeznacza się skończoną liczbę bajtów. Ponieważ wiele liczb wymiernych oraz liczby niewymierne mają nieskończone rozwinięcie binarne części ułamkowej, są pamiętane przez komputer w przybliżeniu. Konsekwencje tego można zaobserwować po uruchomieniu programu ze s. 104, który dodaje sto tysięcy liczb 0,1 (liczba 0,1 ma nieskończone rozwinięcie binarne). Wykorzystany w programie typ **float** poznasz w dalszej części tematu. Rysunek 5.2 na s. 104 przedstawia efekt wykonania programu.

Kod źródłowy programu dodającego 100 000 liczb 0,1 oraz wskazującego różnicę między otrzymanym a poprawnym wynikiem

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  const int N=100000;
5.
6.  int main()
7.  {
8.      float liczba=0.1, suma=0;
9.      for (int i=0;i<N;i++) suma+=liczba;
10.     cout<<"Wynik dodawania: "<<suma<<endl;
11.     cout<<"Poprawny wynik: "<<N*liczba<<endl;
12.     cout<<"Blad: "<<N*liczba-suma<<endl;
13.     return 0;
14. }
```

```
Wynik dodawania: 9998.56
Poprawny wynik: 10000
Blad: 1.44351
```

Rys. 5.2. Efekt wykonania programu dodającego 100 000 liczb 0,1 oraz wskazującego różnicę między otrzymanym a poprawnym wynikiem

Reprezentacja stałoprzecinkowa (stałopozycyjna)

Reprezentacja stałoprzecinkowa (stałopozycyjna) liczb

W **reprezentacji stałoprzecinkowej (stałopozycyjnej) liczb** przeznacza się określoną liczbę bitów na część całkowitą liczby i na jej część ułamkową. Części te traktuje się niezależnie. Położenie przecinka oddzielającego część całkowitą od ułamkowej jest stałe.

Warto wiedzieć

Ponieważ część całkowita zawiera informację o znaku liczby, zapisuje się ją w kodzie U2.

Tabela 5.3 przedstawia zaokrąglenia wartości $0{,}0(0011)_2$, czyli ułamka $\frac{1}{10}$ w systemie dziesiętnym, w zależności od liczby bitów przeznaczonych na reprezentację części ułamkowej.

Liczba bitów przeznaczonych na część ułamkową	Zaokrąglona wartość binarna	Reprezentacja w systemie dziesiętnym zaokrąglonej wartości binarnej
1	0,0	0
2	0,00	0
3	0,001	0,125
4	0,0010	0,125
5	0,00011	0,09375
6	0,000110	0,09375
7	0,0001101	0,1015625
8	0,00011010	0,1015625

Tabela 5.3. Zaokrąglenia liczby $0{,}0(0011)_2$ w zależności od liczby bitów przeznaczonych na część ułamkową oraz wartości dziesiętne zaokrągleń

Zaokrąglenie części ułamkowej odbywa się w następujący sposób: jeśli pierwszą niemieszczącą się cyfrą jest zero, to odrzucane są wszystkie niemieszczące się cyfry bez zmiany wartości zaokrąglenia. W przeciwnym przypadku do ostatniej pozycji zaokrąglenia dodaje się 1.

Ćwiczenie 5

Znajdź reprezentację w systemie dziesiętnym zaokrąglonej wartości binarnej:

a. liczby $\frac{1}{10}$, jeśli na część ułamkową przeznaczono 9 bitów,

b. liczby $\frac{1}{5}$, jeśli na część ułamkową przeznaczono 8 bitów,

c. liczby $\frac{2}{7}$, jeśli na część ułamkową przeznaczono 7 bitów.

Przypuśćmy, że na część całkowitą i ułamkową przeznaczono po 1 bajcie (8 bitów) pamięci. Wówczas liczba 100,1 zapisana w systemie dziesiętnym będzie miała reprezentację stałoprzecinkową taką jak na rysunku 5.3.

Rys. 5.3. Reprezentacja stałoprzecinkowa liczby 100,1 w systemie dwójkowym, jeśli na część całkowitą i ułamkową przeznaczono po 1 bajcie

Reprezentacja stałoprzecinkowa liczb jest mało praktyczna, zwłaszcza w programach, w których będą stosowane liczby z szerokiego zakresu (od bardzo dużych po bardzo małe). Na przykład w astronomii używa się często bardzo dużych liczb, a dokładność części ułamkowej nie ma większego znaczenia. Z kolei w mikrobiologii czy fizyce kwantowej ważna jest duża precyzja części ułamkowej. Użycie reprezentacji stałoprzecinkowej do zapisu liczb z tak dużego zakresu wymaga wielu bajtów pamięci, które często nie byłyby wykorzystane. Dlatego przedstawimy bardziej praktyczny sposób reprezentacji liczb rzeczywistych w komputerze.

Warto wiedzieć

W arytmetyce liczb stałoprzecinkowych obowiązują prawa łączności i przemienności dodawania.

Zapamiętaj

W reprezentacji stałoprzecinkowej (stałopozycyjnej) liczb przeznacza się określoną liczbę bitów na część całkowitą i część ułamkową liczby. Część całkowitą oraz część ułamkową traktuje się niezależnie. Reprezentacja stałoprzecinkowa jest mało praktyczna w przypadku zapisywania liczb z szerokiego zakresu.

Reprezentacja zmiennoprzecinkowa (zmiennopozycyjna)

Reprezentacja zmiennoprzecinkowa (zmiennopozycyjna) liczb

W **reprezentacji zmiennoprzecinkowej** (**zmiennopozycyjnej**) **liczb** przecinek oddzielający część całkowitą od ułamkowej nie ma stałej pozycji. Liczbę x w reprezentacji zmiennoprzecinkowej przedstawia wzór:

Mantysa

$x = m \cdot p^c$, gdzie m – **mantysa**,

p – podstawa systemu, w którym zapisana jest liczba,

Cecha

c – **cecha**, wyrażona liczbą całkowitą.

Stosując powyższy wzór, daną liczbę można przedstawić w różny sposób. Na przykład w systemie dziesiętnym liczbę 123,456 można zapisać m.in. jako $0{,}123456 \cdot 10^3$ lub $1{,}23456 \cdot 10^2$. Aby liczba miała jednoznaczną reprezentację zmiennoprzecinkową, na mantysę narzucane są pewne warunki. Powinna być ona liczbą z przedziału $[1; p)$, gdzie p jest podstawą systemu liczbowego. Mówimy, że liczba zapisana z taką

Postać znormalizowana

mantysą jest w **postaci znormalizowanej**. Liczba 123,456 zapisana w tej postaci wygląda następująco: $1{,}23456 \cdot 10^2$. Mantysa jest równa 1,23456, a cecha 2. Zwróć uwagę, że w postaci znormalizowanej mantysa jest zawsze dodatnia, więc oddzielnie trzeba pamiętać znak liczby. Nie da się zapisać w postaci znormalizowanej liczby 0.

Warto wiedzieć

Przyjmuje się, że liczba 0 jest reprezentowana przez mantysę równą 0.

W opisanym przykładzie widać, że przesunięcie przecinka w mantysie wpływa na wartość cechy. Na przykład przesunięcie przecinka w prawo o jedną pozycję skutkuje zmniejszeniem cechy o 1, z kolei przesunięcie przecinka w lewo o jedną pozycję – zwiększeniem cechy o 1.

Warto wiedzieć

Reprezentacja zmiennoprzecinkowa nazywana jest też notacją wykładniczą (naukową). Wykorzystuje się ją często w różnych dziedzinach. Na przykład stałą grawitacji zapisuje się w postaci: $G = 6{,}67259 \cdot 10^{-11}$.

Zajmiemy się teraz reprezentacją zmiennoprzecinkową liczb binarnych w postaci znormalizowanej. W przypadku tych liczb nie trzeba pamiętać części całkowitej mantysy, ponieważ jest ona zawsze równa 1. Liczbę binarną można reprezentować jako znak liczby oraz parę liczb: mantysę i cechę. Liczbę można więc wyrazić wzorem:

$x = (-1)^{znak} \cdot mantysa \cdot 2^{cecha}$, gdzie *znak* – liczba 0 dla liczby dodatniej i 1 dla liczby ujemnej.

A to ciekawe

Jak zaokrąglenie liczby może zmienić skład parlamentu?

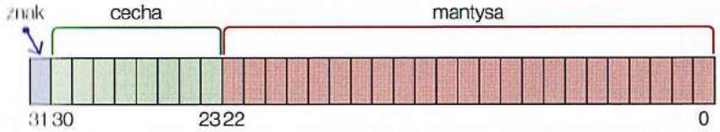
Konsekwencje zaokrąglania liczb mogą mieć duży wpływ na życie obywateli. 5 kwietnia 1992 r. podczas wyborów do parlamentu w Szlezwiku-Holsztynie, jednym z krajów związkowych Niemiec, o mały włos nie doszło do poważnej pomyłki. Aby wejść do parlamentu, partia musiała zdobyć minimum 5% głosów. Po podliczeniu głosów wydawało się, że jedno ze startujących ugrupowań uzyskało dokładnie tyle. Okazało się jednak, że naprawdę było to 4,97% głosów, a program komputerowy zaokrąglił wynik do jednego miejsca po przecinku, czyli 5,0%. Na szczęście błąd wykryto na czas i odpowiednio przeliczono mandaty do parlamentu.

W języku C++ do pamiętania liczb zmiennoprzecinkowych najczęściej używa się dwóch typów zmiennych:

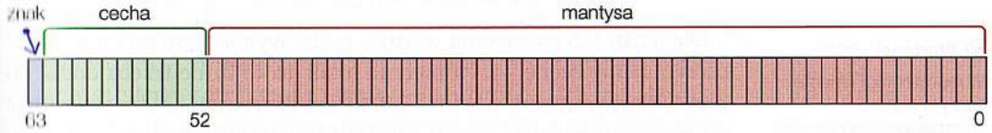
- typu **float**, wykorzystującego 4 bajty (32 bity) pamięci z przeznaczeniem: 1 bitu na znak, 8 bitów na cechę, 23 bitów na mantysę,
- typu **double**, wykorzystującego 8 bajtów (64 bity) pamięci z przeznaczeniem: 1 bitu na znak, 11 bitów na cechę, 52 bitów na mantysę.

Liczby zmiennoprzecinkowe zapisane na 32 bitach nazywane są **liczbami pojedynczej precyzji**, natomiast zapisane na 64 bitach – **liczbami podwójnej precyzji**. Rysunki 5.4 i 5.5 przedstawiają schematy reprezentacji liczb zmiennoprzecinkowych pojedynczej i podwójnej precyzji.

Liczba pojedynczej precyzji, liczba podwójnej precyzji



Rys. 5.4. Reprezentacja liczby zmiennoprzecinkowej pojedynczej precyzji



Rys. 5.5. Reprezentacja liczby zmiennoprzecinkowej podwójnej precyzji

O precyzji, czyli dokładności (liczbie cyfr znaczących), decyduje liczba bitów przeznaczonych na mantysę. O zakresie decyduje liczba bitów przeznaczonych na cechę. Tabela 5.4 przedstawia zakresy oraz precyzję liczb typu **float** oraz **double**.

Warto wiedzieć

Sposób reprezentacji liczb zmiennoprzecinkowych pojedynczej i podwójnej precyzji opisuje standard IEEE 754.

Nazwa typu	Liczba bajtów	Przeznaczenie i zakres
`float`	4	Liczby rzeczywiste z zakresu od $1{,}17549 \cdot 10^{-38}$ do $3{,}40282 \cdot 10^{38}$, w których liczba cyfr znaczących (dokładność) jest nie większa niż 7
`double`	8	Liczby rzeczywiste z zakresu od $2{,}22507 \cdot 10^{-308}$ do $1{,}79769 \cdot 10^{308}$, w których liczba cyfr znaczących (dokładność) jest nie większa niż 15

Tabela 5.4. Zakresy i dokładność typów zmiennoprzecinkowych

Cecha jest pamiętana przez komputer w tzw. **kodzie z nadmiarem**. Dla liczb pojedynczej precyzji nadmiar wynosi 127. Oznacza to, że aby obliczyć, ile jest równa cecha, od pamiętanej wartości cechy należy odjąć 127. Kod z nadmiarem umożliwia dowolne przesunięcie przedziału reprezentowanych wartości. W tym przypadku reprezentowany jest przedział $[-127; 128]$. Skrajne wartości są zarezerwowane, m.in. na reprezentację zera. Dla liczb podwójnej precyzji nadmiar wynosi 1023.

Kod z nadmiarem

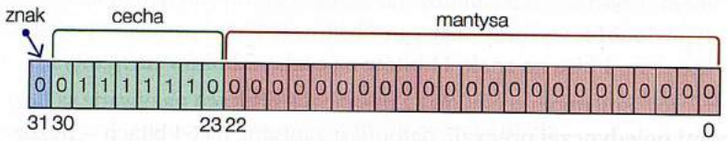
Dobra rada

Pamiętaj, że zero w znormalizowanej postaci nie ma swojej reprezentacji.

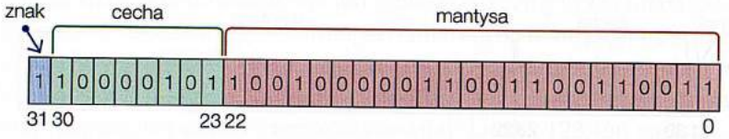
Warto wiedzieć

Pierwszym komputerem wykorzystującym binarne liczby zmiennoprzecinkowe był komputer Z1. Skonstruował go w 1938 r. niemiecki inżynier Konrad Zuse.

Rysunki 5.6 i 5.7 przedstawiają reprezentacje liczb 0,5 i –100,1 (w systemie dziesiętnym) jako liczb zmiennoprzecinkowych pojedynczej precyzji (typ **float**).



Rys. 5.6. Reprezentacja binarna liczby 0,5 jako liczby zmiennoprzecinkowej pojedynczej precyzji (typ **float**)



Rys. 5.7. Reprezentacja binarna liczby –100,1 jako liczby zmiennoprzecinkowej pojedynczej precyzji (typ **float**)

Warto wiedzieć

W reprezentacji binarnej liczby w postaci znormalizowanej pierwsza cyfra mantysy (jej część całkowita) jest zawsze równa 1, dlatego nie trzeba jej pamiętać. Dzięki temu na reprezentację mantysy można przeznaczyć jeden bit więcej.

Dla liczby 0,5 pamiętana wartość cechy wynosi 126, po odjęciu 127 otrzymujemy cechę –1. Pamiętana mantysa to 0, ale trzeba dodać do niej 1. Otrzymujemy wartość 1,0, której przecinek należy przesunąć o 1 pozycję w lewo (cecha –1). Liczba $0{,}1_2$ to liczba 0,5 w systemie dziesiętnym.

W przypadku liczby dziesiętnej –100,1 pamiętana cecha to 133, po odjęciu 127 otrzymujemy cechę 6. Po dodaniu 1 do mantysy otrzymujemy mantysę 1,100100000110011..., a po przesunięciu w prawo przecinka o 6 pozycji – liczbę binarną: 1100100,000110011... Część całkowita 1100100 ma wartość dziesiętną 100, a część ułamkowa to zaokrąglone rozwinięcie ułamka dziesiętnego 0,1. Bit znaku jest równy 1, więc liczba jest ujemna.

Warto wiedzieć

Komputer może być wyposażony w koprocesor arytmetyczny, który wspomaga procesor w wykonywaniu działań na liczbach zmiennoprzecinkowych. Ma to znaczenie szczególnie w programach, które wykonują dużo takich obliczeń, np. w grach komputerowych wyznaczających współrzędne obiektów w przestrzeni czy programach do tworzenia modeli 2D i 3D.

Ćwiczenie 6

Podaj reprezentację liczby zapisanej w systemie dziesiętnym jako liczba zmiennoprzecinkowa pojedynczej precyzji.

a. 10,25

b. –0,2

Zapamiętaj

Liczby rzeczywiste są reprezentowane w komputerze najczęściej w postaci znormalizowanej zmiennoprzecinkowej. Pamiętane są: znak liczby, jej mantysa i cecha. Liczbę binarną można wyrazić wzorem:

$x = (-1)^{znak} \cdot mantysa \cdot 2^{cecha}$

5.3. Działania arytmetyczne na liczbach zmiennoprzecinkowych

Wyjaśnimy najpierw, jak wykonać mnożenie i dzielenie liczb zmiennoprzecinkowych. Następnie zajmiemy się dodawaniem i odejmowaniem takich liczb.

Mnożenie i dzielenie liczb zmiennoprzecinkowych

Żeby pomnożyć dwie liczby zmiennoprzecinkowe, należy ustalić znak wyniku, pomnożyć mantysy i dodać cechy. Jeśli w otrzymanym wyniku mantysa nie jest z przedziału $[1; p)$, gdzie p – podstawa systemu liczbowego, należy przekształcić wynik tak, aby był w **postaci znormalizowanej**.

Postać znormalizowana, s. 106

Mnożenie przykładowych liczb $9{,}8765 \cdot 10^3$ i $-4{,}321 \cdot 10^2$ zapisanych w systemie dziesiętnym wykonalibyśmy następująco:

1. Ustalamy znak wyniku: znaki liczb są przeciwne, więc wynik będzie ujemny.
2. Mnożymy mantysy: $9{,}8765 \cdot 4{,}321 = 42{,}6763565$.
3. Dodajemy cechy: $3 + 2 = 5$.
4. Sprawdzamy, czy mantysa jest z przedziału $[1; 10)$. Ponieważ nie jest, przekształcamy otrzymany wynik $(-42{,}6763565 \cdot 10^5)$ tak, aby mantysa należała do wskazanego przedziału. W tym celu trzeba przesunąć przecinek o jedną pozycję w lewo, co skutkuje zwiększeniem cechy o 1. Otrzymujemy zatem wynik: $-4{,}26763565 \cdot 10^6$.

Przy dzieleniu liczb zmiennoprzecinkowych ustala się znak wyniku, dzieli mantysy i odejmuje cechy. Jeśli otrzymany wynik nie jest w postaci znormalizowanej, należy go przekształcić do takiej postaci.

Dodawanie i odejmowanie liczb zmiennoprzecinkowych

Liczby zmiennoprzecinkowe można dodawać i odejmować tylko wtedy, gdy mają takie same cechy. Jeśli dodawane (odejmowane) liczby mają równe cechy, wystarczy dodać (odjąć) mantysy, a następnie w razie potrzeby znormalizować wynik. Jeśli cechy liczb są różne, to liczbę, która ma mniejszą cechę, przekształcamy tak, aby miała cechę taką jak druga liczba.

Na przykład aby dodać liczby $9{,}8765 \cdot 10^3$ oraz $4{,}321 \cdot 10^2$ zapisane w systemie dziesiętnym, przekształcamy drugą liczbę tak, aby miała cechę równą 3. Będzie miała ona wówczas postać $0{,}4321 \cdot 10^3$. Zatem: $9{,}8765 \cdot 10^3 + 0{,}4321 \cdot 10^3 = 10{,}3086 \cdot 10^3$. Po znormalizowaniu otrzymujemy wynik $1{,}03086 \cdot 10^4$.

Warto wiedzieć

W arytmetyce liczb zmiennoprzecinkowych wynik może zależeć od kolejności wykonywanych działań, np. dla liczb a, b i c: wynik $(a + b) + c$ może być różny od wyniku $a + (b + c)$.

Ćwiczenie 7

Oblicz:

a. $5{,}123 \cdot 10^{-1} \cdot 4{,}987 \cdot 10^2$,

b. $5{,}123 \cdot 10^{-1} + 4{,}987 \cdot 10^2$.

Podsumowanie

- Skończone rozwinięcie binarne mają tylko te ułamki właściwe, których mianownikiem w postaci nieskracalnej jest potęga liczby 2.
- Ułamki właściwe, które w postaci nieskracalnej mają mianownik niebędący potęgą liczby 2, w systemie binarnym są ułamkami okresowymi.
- W algorytmie wyznaczającym rozwinięcie binarne ułamka właściwego mnoży się ten ułamek przez 2: część całkowita wyniku jest pierwszą cyfrą rozwinięcia binarnego, a część ułamkową mnoży się przez 2, aby wyznaczyć kolejną cyfrę rozwinięcia binarnego. Czynności powtarza się do momentu uzyskania licznika części ułamkowej równego 0 lub znalezienia okresu.
- Liczby rzeczywiste można reprezentować w postaci stałoprzecinkowej (stałopozycyjnej) oraz zmiennoprzecinkowej (zmiennopozycyjnej).
- W reprezentacji stałoprzecinkowej przeznacza się stałą liczbę bitów na część całkowitą oraz na część ułamkową. Przecinek oddzielający część całkowitą od ułamkowej ma stałą pozycję.
- W reprezentacji zmiennoprzecinkowej liczbę zapisuje się w postaci $x = m \cdot p^c$, gdzie m – mantysa, p – podstawa systemu, w którym zapisana jest liczba, c – cecha, wyrażona liczbą całkowitą. Przecinek nie ma stałej pozycji w reprezentacji liczby.
- Postać zmiennoprzecinkową liczby nazywamy znormalizowaną, jeśli mantysa jest liczbą z przedziału $[1; p)$, gdzie p – podstawa systemu.
- Zapis zmiennoprzecinkowy jest bardziej uniwersalny od zapisu stałoprzecinkowego, ponieważ pozwala pamiętać liczby z wykorzystaniem mniejszej liczby bajtów.
- Liczby rzeczywiste w komputerze pamiętane są najczęściej jako liczby zmiennoprzecinkowe.
- Aby pomnożyć liczby zmiennoprzecinkowe, ustalamy znak wyniku, mnożymy mantysy i dodajemy cechy. Jeśli otrzymany wynik nie jest w postaci znormalizowanej, trzeba przekształcić go do tej postaci.
- Przy dzieleniu liczb zmiennoprzecinkowych ustala się znak wyniku, dzieli mantysy i odejmuje cechy. Jeśli otrzymany wynik nie jest w postaci znormalizowanej, należy go przekształcić do takiej postaci.
- Żeby dodać (odjąć) liczby zapisane w postaci zmiennoprzecinkowej, wystarczy dodać (odjąć) mantysy, pod warunkiem że cechy są takie same. Jeśli cechy nie są równe, to jedną z liczb przekształca się tak, aby obie cechy były takie same.

Zadania

* **1** Znajdź rozwinięcie binarne ułamka $\frac{1}{3}$ zapisanego w systemie dziesiętnym.

* **2** Znajdź reprezentację w systemie trójkowym ułamka $\frac{1}{3}$ zapisanego w systemie dziesiętnym.

* **3** Znajdź binarną reprezentację stałoprzecinkową liczby 123,375 zapisanej w systemie dziesiętnym. Przeznacz 1 bajt na część całkowitą oraz 1 bajt na część ułamkową.

** **4** Znajdź binarną reprezentację liczby 100,1 zapisanej w systemie dziesiętnym jako liczba zmiennoprzecinkowa podwójnej precyzji (typ **double**).

** **5** Napisz program znajdujący rozwinięcie binarne ułamka właściwego wczytanego w systemie dziesiętnym, który w postaci ułamka zwykłego nieskracalnego nie ma w mianowniku potęgi liczby 2. Wykorzystaj typ `list`.

** **6** Napisz program znajdujący rozwinięcie binarne ułamka właściwego wczytanego w systemie dziesiętnym. Program powinien znajdować zarówno rozwinięcia skończone, jak i okresowe.

** **7** Napisz program znajdujący reprezentację stałoprzecinkową liczby z przedziału [0; 255] zapisanej w systemie dziesiętnym w naturalnym kodzie binarnym. Program powinien wykorzystywać 1 bajt na część całkowitą i 1 bajt na część ułamkową. Wczytaj do programu niezależnie część całkowitą i część ułamkową liczby.

*** **8** Napisz program symulujący dodawanie nieujemnych liczb binarnych w reprezentacji zmiennopozycyjnej, w której na cechę przeznaczony jest 1 bajt i na mantysę także 1 bajt. Cechy zapisane są w kodzie z nadmiarem 127. Dane do programu stanowią dwa napisy złożone z 16 znaków 0 i 1, w każdym napisie co najmniej jeden bit cechy jest równy 0. Wynikiem jest także napis złożony z 16 znaków 0 i 1, reprezentujący obliczoną sumę.