Informatyka Europejczyka. Informatyka. Podręcznik dla szkół ponadgimnazjalnych. Część I

Autor: Grażyna Zawadzka
ISBN: 978-83-246-0925-3
Format: 170×240, stron: 288

Z komputerami stykamy się dziś niemal każdego dnia. Wykorzystujemy je do pracy i rozrywki, wyszukiwania informacji w sieci, komunikowania się ze znajomymi i wielu innych zadań. Jednak komputer to nie tylko gry, edytory tekstu, poczta elektroniczna, portale społecznościowe czy komunikatory – to także wiele przydatnych narzędzi, które stają się niezbędne do codziennego funkcjonowania we współczesnym świecie.

„Informatyka Europejczyka. Informatyka. Podręcznik dla szkół ponadgimnazjalnych. Część I" przedstawia zagadnienia związane z algorytmiką i programowaniem. Dowiesz się z tego podręcznika, jakimi prawami rządzą się algorytmy, i nauczysz się rozpoznawać ich typy. Zyskasz możliwość samodzielnego analizowania algorytmów określających najczęstsze metody numerycznego rozwiązywania problemów obliczeniowych. W dalszej części podręcznika znajdziesz informacje dotyczące programowania w języku C++ – typy danych i instrukcji, strukturę programu, sposoby realizacji typowych zadań programistycznych oraz podstawy programowania obiektowego.

Na płycie CD-ROM dołączonej do książki znajdziesz przykłady programów napisanych w językach C++ i Pascal, uzupełniający materiał dotyczący programowania obiektowego, dane do zadań, pliki potrzebne do wykonywania ćwiczeń oraz wybrane zadania z egzaminów maturalnych.

- Pojęcie algorytmu
- Sposoby przedstawiania algorytmów
- Metody programowania: liniowe, warunkowe, iteracja, rekurencja, „dziel i zwyciężaj", zachłanna
- Analiza i realizacja algorytmów
- Podstawy kryptografii i wybrane algorytmy szyfrujące
- Podstawy języka C++: struktura programu, operacje wejścia i wyjścia, typy instrukcji, proste i złożone typy danych, strukturalizacja programu, dynamiczne struktury danych, plikowe operacje wejścia i wyjścia oraz programowanie obiektowe

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

2.2. Wyznaczanie wartości wielomianu, pozycyjne systemy liczbowe i reprezentacja danych liczbowych w komputerze

2.2.1. Systemy liczbowe

Systemem liczbowym nazywamy zbiór zasad określających sposób zapisywania i nazywania liczb.

Pozycyjny system liczbowy to system, w którym wartość cyfry zależy od miejsca, w jakim znajduje się ona w danej liczbie. Miejsce to nazywamy pozycją.

Do najważniejszych pozycyjnych systemów liczbowych wykorzystywanych w informatyce należą:

- system dwójkowy, czyli binarny;
- system ósemkowy, czyli oktalny;
- system szesnastkowy, czyli heksadecymalny.

Podstawą **systemu binarnego**, określającą liczbę cyfr, jest dwa. System ten korzysta więc z dwóch cyfr, którymi są 0 i 1.

System oktalny ma podstawę osiem, stąd cyframi są tutaj 0, 1, 2, 3, 4, 5, 6, 7.

Podstawą **systemu heksadecymalnego** jest szesnaście, a więc w systemie tym korzystamy z szesnastu cyfr. Cyframi tego systemu są: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *A*, *B*, *C*, *D*, *E*, *F*. Wykorzystanie liter w zapisie cyfr podyktowane jest koniecznością jednoznacznej notacji liczby w tym systemie. Litery odpowiadają cyfrom, których wartości zapisane w układzie dziesiętnym są liczbami dwucyfrowymi:

$$A_{16} = 10_{10},$$
$$B_{16} = 11_{10},$$
$$C_{16} = 12_{10},$$
$$D_{16} = 13_{10},$$
$$E_{16} = 14_{10},$$
$$F_{16} = 15_{10}.$$

Gdybyśmy nie korzystali z liter, zapis liczby 112_{16} mógłby oznaczać 112_{16} lub $B2_{16}$ lub $1C_{16}$.

Przy realizacji konwersji i działań arytmetycznych w różnych systemach liczbowych można zastosować udostępnioną w systemie Windows **aplikację Kalkulator**. Program ten umożliwia realizację obliczeń w następujących systemach: decymalnym (czyli dziesiętnym), binarnym, oktalnym i heksadecymalnym. Wykonywać można zarówno konwersję pomiędzy wymienionymi systemami, jak i operacje arytmetyczne. Aby uzyskać dostęp do tych systemów, należy po uruchomieniu aplikacji Kalkulator wybrać w menu polecenie **Widok-Naukowy**.

Najbardziej znanym systemem liczbowym, który nie jest pozycyjny, jest **system rzymski**. Zaliczany jest on do systemów zwanych **addytywnymi**. Charakteryzują się one tym, że mają symbole dla kilku małych liczb oraz ich wielokrotności.

W przypadku systemu rzymskiego dotyczy to wielokrotności liczb 5 i 10. Dostępnych jest razem siedem znaków:

$I = 1$,

$V = 5$,

$X = 10$,

$L = 50$,

$C = 100$,

$D = 500$,

$M = 1000$.

Zapisywanie liczby w tym systemie polega na składaniu jej przez dodawanie lub odejmowanie kolejnych symboli o określonej wartości. Liczba reprezentująca dany symbol odejmowana jest wówczas, gdy następny symbol ma większą od niej wartość. W przeciwnym wypadku wykonywane jest dodawanie.

Na przykład wartość liczby *MCCXCIX* wyznacza się następująco:

$$MCCXCIX = 1000_{10} + 100_{10} + 100_{10} - 10_{10} + 100_{10} - 1_{10} + 10_{10} = 1299_{10}.$$

Zadanie 2.3.

Zamień liczby podane w systemie rzymskim na system dziesiętny:

a) *MXLVIII*,
b) *MCMLXXXIV*,
c) *CMXLVII*,
d) *DXLIX*,
e) *MMMCDI*.

Zadanie 2.4.

Zamień liczby podane w systemie dziesiętnym na system rzymski:

a) 1999_{10},
b) 184_{10},
c) 2876_{10},
d) 3012_{10},
e) 488_{10}.

Zadanie 2.5.

Podaj specyfikację zadania i skonstruuj algorytm w postaci schematu blokowego i programu realizujący konwersję liczb z systemu rzymskiego na dziesiętny.

Zadanie 2.6.

Podaj specyfikację zadania i skonstruuj algorytm w postaci programu realizujący konwersję liczb z systemu dziesiętnego na rzymski.

2.2.2. Konwersje pozycyjnych systemów liczbowych

Konwersja systemu dziesiętnego na inny pozycyjny system liczbowy

Aby **zamienić liczbę nieujemną zapisaną w systemie decymalnym na wartość w systemie binarnym**, należy powtarzać dzielenie z resztą tej liczby przez podstawę sytemu dwójkowego, dopóki w wyniku dzielenia nie uzyskamy 0. Wówczas otrzymane reszty z dzielenia stanowią rozwiązanie.

Przykład 2.3.

Przeanalizujmy konwersję systemu dziesiętnego na dwójkowy na przykładzie liczbowym. Zapiszmy liczbę 125_{10} w systemie binarnym:

125	:	2	=	62	reszta **1**
62	:	2	=	31	reszta **0**
31	:	2	=	15	reszta **1**
15	:	2	=	7	reszta **1**
7	:	2	=	3	reszta **1**
3	:	2	=	1	reszta **1**
1	:	2	=	**0**	reszta **1**

W wyniku dzielenia uzyskaliśmy zero, więc obliczenia zostały zakończone. Rozwiązanie odczytujemy, rozpoczynając od reszty uzyskanej na końcu, stąd $125_{10} = 1111101_2$.

Wygodniejszy jest następujący zapis konwersji tych liczb:

125	1
62	0
31	1
15	1
7	1
3	1
1	1
0	

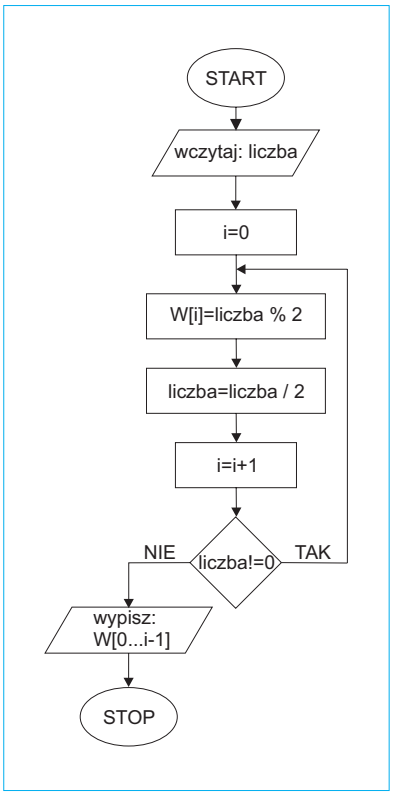
Opracujmy **algorytm wykonujący zamianę liczb zapisanych w systemie dziesiętnym na liczby binarne** w postaci schematu blokowego (patrz rysunek 2.2) oraz programów w językach C++ (patrz punkt 3.6.1, „Tablice") i Pascal.

Specyfikacja:

Dane: Liczba całkowita: *liczba* ≥ 0 (liczba w systemie dziesiętnym).

Wynik: Liczba całkowita: $i > 0$ (liczba cyfr wartości otrzymanej po zamianie z systemu dziesiętnego na dwójkowy).

i-elementowa tablica jednowymiarowa zawierająca liczby całkowite: $W[0...i-1]$ (liczba zapisana w systemie dwójkowym uzyskana po zamianie z systemu dziesiętnego, której cyfry należy odczytać w kolejności $W[i-1], W[i-2], \ldots, W[0]$).



Rysunek 2.2.
Schemat blokowy algorytmu realizującego konwersję liczb z systemu dziesiętnego na dwójkowy

Funkcja w języku C++ (*prog2_6.cpp*):

```
void oblicz (long liczba, int &i, int W[])
{
 i=0;
 do
 {
  W[i]=liczba % 2;
  liczba=liczba/2;
  i++;
 }
 while (liczba!=0);
}
```

Procedura w języku Pascal (*prog2_6.pas*):

```
procedure oblicz (liczba: longint; var i: integer; var W: tablica);
begin
 i:=0;
 repeat
  W[i]:=liczba mod 2;
  liczba:=liczba div 2;
  i:=i+1
 until liczba=0
end;
```

Omówioną metodę konwersji liczb z systemu decymalnego na binarny można zastosować również przy **zamianie systemu dziesiętnego na inne systemy liczbowe**. Należy jednak pamiętać, że każdy z tych systemów ma inną podstawę. Na przykład, zamieniając liczby systemu decymalnego na system oktalny, będziemy dzielić przez osiem, na system szesnastkowy — przez szesnaście itd.

Przykład 2.4.

Zapiszmy liczbę 459_{10} w systemie szesnastkowym. Zwróć uwagę na cyfry, których wartość jest większa niż 9.

459	:	16	=	28	reszta **11** = *B*
28	:	16	=	1	reszta **12** = *C*
1	:	16	=	0	reszta **1**

Poniżej przedstawiono skrócony zapis konwersji tych liczb:

459	11 = *B*
28	12 = *C*
1	1
0	

Uzyskaliśmy następujący wynik: $459_{10} = 1CB_{16}$.

Zadanie 2.7.

Przekonwertuj podane liczby całkowite z systemu dziesiętnego na systemy o podstawach 2, 4, 8, 9, 16:

a) 1234_{10},
b) 999_{10},
c) 1380_{10},
d) 49_{10},
e) 2135_{10}.

Zadanie 2.8.

Podaj specyfikację zadania i skonstruuj algorytm w postaci listy kroków i programu realizujący konwersję liczb zapisanych w systemie dziesiętnym na liczby w systemie o podstawie z zakresu [2; 9].

Zadanie 2.9.

Podaj specyfikację zadania i skonstruuj algorytm w postaci programu realizujący konwersję liczb zapisanych w systemie dziesiętnym na system szesnastkowy.

Konwersja innych pozycyjnych systemów liczbowych na system dziesiętny

Aby **zamienić liczbę zapisaną w systemie binarnym na decymalny**, należy wyznaczyć wartość sumy cyfr tej liczby pomnożonych przez kolejne potęgi podstawy systemu, czyli 2.

Przykład 2.5.

Przeanalizujmy przebieg działania tej metody na przykładzie liczbowym. Wykonajmy konwersję z systemu binarnego na decymalny liczby 1011011_2. Najpierw należy do każdej cyfry tej liczby dopasować odpowiednie potęgi liczby 2. Wartość mnożnika będącego potęgą liczby 2 zależy tutaj od pozycji cyfry w danej liczbie.

1	0	1	1	0	1	1
2^6	2^5	2^4	2^3	2^2	2^1	2^0

Następnie wyznaczamy wartość sumy iloczynów:

$$1\cdot 2^6+0\cdot 2^5+1\cdot 2^4+1\cdot 2^3+0\cdot 2^2+1\cdot 2^1+1\cdot 2^0=91_{10}.$$

Uzyskana wartość 91_{10} to liczba dziesiętna będąca wynikiem zamiany systemów. Mamy więc $1011011_2=91_{10}$.

W przypadku, gdy chcemy **zamieniać liczby z innych systemów pozycyjnych na decymalny**, postępujemy podobnie. Musimy jednak pamiętać o tym, by kolejne cyfry konwertowanej liczby mnożyć przez potęgi podstawy systemu, w którym jest zapisana.

Przykład 2.6.

Zapiszmy liczbę $1A0B_{12}$ w systemie decymalnym:

1	A	0	B
12^3	12^2	12^1	12^0

$1\cdot 12^3+10\cdot 12^2+0\cdot 12^1+11\cdot 12^0=3179_{10}$

Otrzymaliśmy wynik: $1A0B_{12}=3179_{10}$.

Zadanie 2.10.

Zapisz podane liczby całkowite w systemie dziesiętnym:

a) 1011101_2,
b) 10011111_2,
c) 1000001_2,
d) 2120_3,
e) 430_5,
f) 145_6,
g) 264_8,
h) 7777_8,
i) 10007_8,
j) $ABCDE_{16}$,
k) $FFFF_{16}$,
l) $1A17B0_{16}$.

Konwersje między systemami niedziesiętnymi

Najczęściej stosowanymi systemami liczbowymi w informatyce są systemy: binarny, oktalny i heksadecymalny. Wykorzystanie systemu dwójkowego wynika ze sposobu zapisu liczb w pamięci komputera za pomocą bitów. Z kolei systemy ósemkowy i szesnastkowy to systemy, których podstawy są potęgami liczby 2. Wynika stąd możliwość wykonywania bezpośredniej konwersji między tymi systemami a systemem binarnym.

Liczba binarna zapisana na trzech miejscach ma wartości w zakresie [0; 111], co w systemie dziesiętnym wynosi [0; 7]. Liczby zawarte w tym zakresie to wszystkie cyfry systemu ósemkowego. Wykorzystajmy tę własność przy konwersji liczb między systemami binarnym i oktalnym.

Przykład 2.7.

Wykonajmy konwersję liczby 1011101111_2 na system oktalny. Najpierw należy zamienianą liczbę pogrupować po trzy cyfry, rozpoczynając od prawej strony:

1 011 101 111

Następnie każdą z uzyskanych grup traktujemy jak cyfrę liczby, którą chcemy uzyskać w systemie oktalnym. Wykonujemy więc następujące obliczenia:

$$1_2 = 1 \cdot 2^0 = 1$$

$$11_2 = 1 \cdot 2^1 + 1 \cdot 2^0 = 3$$

$$101_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5$$

$$111_2 = 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 7$$

Uzyskujemy wynik: $1011101111_2 = 1357_8$.

Zamianę liczby zapisanej w systemie oktalnym na binarny realizujemy podobnie. Tym razem jednak każdą kolejną cyfrę liczby oktalnej konwertujemy na system binarny.

W przypadku konwersji między systemem binarnym i heksadecymalnym tok myślenia jest podobny. Należy jednak uwzględnić grupowanie po cztery cyfry. Wynika to stąd, że liczba binarna zapisana na czterech miejscach ma wartości w zakresie [0; 1111], co w systemie dziesiętnym daje [0; 15]. Tym razem są to wszystkie cyfry systemu szesnastkowego.

Przykład 2.8.

Przekonwertujmy liczbę 110011011_2 na system szesnastkowy:

1 1001 1011

$1_2 = 1 \cdot 2^0 = 1$

$1001_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 9$

$1011_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 11 = B$

Po wykonaniu konwersji otrzymujemy: $110011011_2 = 19B_{16}$.

Zadanie 2.11.

Zamień podane liczby całkowite z systemu dziesiętnego na ósemkowy i szesnastkowy z wykorzystaniem systemu dwójkowego:

a) 523_{10},
b) 458_{10},
c) 399_{10},
d) 878_{10},
e) 1001_{10},
f) 1112_{10},
g) 2056_{10}.

2.2.3. Operacje arytmetyczne wykonywane w różnych systemach liczbowych

Wykonując operacje arytmetyczne w różnych systemach liczbowych, należy pamiętać przede wszystkim o podstawie tych systemów. W przypadku systemu dziesiętnego wiemy, że zarówno dodawanie, odejmowanie, jak i mnożenie wykonuje się w oparciu o podstawę systemu, którą jest liczba dziesięć. Gdy realizujemy operację dodawania, nadmiar dziesiątek przenosimy w lewo, natomiast odejmowanie wymaga pożyczania dziesiątek z lewej strony.

Wykonajmy **podstawowe operacje arytmetyczne w systemie binarnym**.

Rozpocznijmy od działania **dodawania**. W tym przypadku, gdy w wyniku dodawania otrzymamy wartość równą lub większą od dwóch, w rozwiązaniu wpisujemy resztę z dzielenia tej wartości przez 2, natomiast w lewo przenosimy wynik dzielenia całkowitego tej liczby przez 2.

Przykład 2.9.

Obliczmy sumę liczb w systemie binarnym: $11011_2+111110_2$. Pogrubieniem wyróżnione są wartości przenoszone w lewo.

1	**1**	**1**	**1**	**1**		
		1	1	0	1	1
+	1	1	1	1	1	0
1	0	1	1	0	0	1

Otrzymany wynik to: $11011_2+111110_2 = 1011001_2$.

Przykład 2.10.

Wyznaczmy sumę czterech liczb zapisanych w systemie binarnym: $111111_2+1110_2+10111_2+110111_2$. Ten przykład wydaje się trudniejszy, jednak jego realizacja opiera się na dokładnie tych samych zasadach.

1	**2**	**2**	**2**	**3**	**2**	**1**
		1	1	1	1	1	1
				1	1	1	0
			1	0	1	1	1
+		1	1	0	1	1	1
1	0	0	1	1	0	1	1

Po wykonaniu obliczeń uzyskujemy następujący wynik: $111111_2+1110_2+10111_2+110111_2 = 10011011_2$.

Kolejnym działaniem, które wykonamy w systemie binarnym, będzie **odejmowanie**. W tym przypadku problem pojawia się w sytuacji, gdy chcemy wykonać operację odejmowania, a liczba, od której odejmujemy, jest zbyt mała. Wówczas należy pobrać wartość z lewej strony. Każda jedynka pobrana bezpośrednio z lewej strony zamieniana jest na podstawę systemu, czyli dwa, a następnie wykonywane jest odejmowanie.

Przykład 2.11.

Obliczmy różnicę liczb zapisanych w systemie binarnym: 110110_2-1010_2. Pogrubieniem wyróżnione są wartości uzyskane po pobraniu z lewej strony.

	0	**2**			
1	~~1~~	~~0~~	1	1	0
–		1	0	1	0
1	0	1	1	0	0

Wynikiem wykonanego działania jest: $110110_2-1010_2 = 101100_2$.

Przykład 2.12.

Wyznaczmy różnicę liczb binarnych: $100000_2 - 111_2$. Ten przykład jest trudniejszy, ale zasady identyczne. Dokładnie przeanalizuj wykonane działanie.

$$\begin{array}{cccccc} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \\ \mathbf{0} & \mathbf{\cancel{2}} & \mathbf{\cancel{2}} & \mathbf{\cancel{2}} & \mathbf{\cancel{2}} & \mathbf{2} \\ \cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0} & 0 \\ - & & & 1 & 1 & 1 \\ \hline & 1 & 1 & 0 & 0 & 1 \end{array}$$

Uzyskaliśmy następujący wynik: $100000_2 - 111_2 = 11001_2$.

Mnożenie jest działaniem, które łączy w sobie operacje mnożenia i dodawania. Najpierw wykonujemy mnożenie kolejnych cyfr jednej liczby przez drugą, a następnie uzyskane wyniki w odpowiedni sposób dodajemy.

Przykład 2.13.

Obliczmy iloczyn dwóch liczb w systemie binarnym: $110111_2 \cdot 1011_2$. Porównaj wykonanie tego działania w systemie dwójkowym z mnożeniem w systemie dziesiętnym.

$$\begin{array}{cccccccccc} & & & 1 & 1 & 0 & 1 & 1 & 1 & \\ & & \times & & & 1 & 0 & 1 & 1 & \\ \hline & & & 1 & 1 & 0 & 1 & 1 & 1 & \\ & & 1 & 1 & 0 & 1 & 1 & 1 & & \\ + & 1 & 1 & 0 & 1 & 1 & 1 & & & \\ \hline 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{array}$$

Po wykonaniu obliczeń otrzymujemy: $110111_2 \cdot 1011_2 = 1001011101_2$.

Zadanie 2.12.

Wykonaj następujące działania arytmetyczne w systemie binarnym:

a) $10110_2 + 1101_2 \cdot 111_2$,

b) $111000_2 - 11_2 + 101100_2 \cdot 110_2$,

c) $101_2 \cdot 1100_2 - 11011_2 + 10_2$,

d) $101110_2 - 10011_2 - 111_2$,

e) $110_2 \cdot 11001_2 + 110111_2 - 11_2$,

f) $11111111_2 + 1_2$.

Zadanie 2.13.

Podaj specyfikację zadania i skonstruuj algorytm w postaci programu wykonujący dodawanie dwóch wprowadzonych z klawiatury nieujemnych liczb całkowitych zapisanych w systemie binarnym.

Przeanalizowaliśmy dokładnie realizację podstawowych działań arytmetycznych w systemie binarnym. Znając zasady wykonywania tych operacji, można przenieść je na płaszczyznę **innych pozycyjnych systemów liczbowych**. Należy jednak pamiętać, aby w tych systemach stosować właściwą im podstawę. Na przykład w oktalnym — 8, w heksadecymalnym — 16.

Przykład 2.14.

Wyznaczmy wartość wyrażenia: $323_4 \cdot 32_4 - 213_4$.

$$\begin{array}{rrrrr} & & 3 & 2 & 3 \\ & \times & & 3 & 2 \\ \hline & 1 & 3 & 1 & 2 \\ + & 2 & 3 & 0 & 1 \\ \hline & 3 & 0 & 3 & 2 & 2 \end{array}$$

$$\begin{array}{rrrrrr} & & & & \mathbf{1} & \mathbf{6} \\ & 3 & 0 & 3 & \cancel{2} & \cancel{2} \\ - & & & 2 & 1 & 3 \\ \hline & 3 & 0 & 1 & 0 & 3 \end{array}$$

Uzyskaliśmy następujący wynik: $323_4 \cdot 32_4 - 213_4 = 30103_4$.

Zadanie 2.14.

Wykonaj następujące działania arytmetyczne w podanych systemach liczbowych:

a) $112_3 \cdot 222_3 - 1100_3$,

b) $4430_5 + 302_5 \cdot 31_5$,

c) $707_8 + 1066_8 \cdot 45_8 - 12_8$,

d) $AB10_{16} - FF_{16} \cdot C_{16} + 1_{16}$.

Zadanie 2.15.

Podaj, w jakich systemach pozycyjnych zostały wykonane następujące działania:

a) 1001+10–1010 = 1,

b) 244·2–14 = 474,

c) (10–2)·2 = 2,

d) *A*1–3·10+*A* = 80.

Które z podanych działań można wykonać w różnych systemach liczbowych?

Zadanie 2.16.

Podaj specyfikację zadania i skonstruuj algorytm w postaci programu wykonujący dodawanie dwóch wprowadzonych z klawiatury nieujemnych liczb całkowitych zapisanych w systemie liczbowym o podstawie z zakresu [2; 9], również wprowadzonej z klawiatury. Wynik niech będzie wypisany w tym samym systemie.

2.2.4. Wyznaczanie wartości wielomianu za pomocą schematu Hornera

Schemat Hornera jest najszybszym sposobem obliczania wartości wielomianu. Przeanalizujmy działanie tej metody, przekształcając ogólny wzór na wartość wielomianu stopnia *n*.

Dany mamy wielomian stopnia *n*, gdzie $n \geq 0$:

$$w_n(x) = a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n. \tag{2.4}$$

W omawianym algorytmie należy stosować grupowanie wyrazów tak długo, aż pozostanie jednomian.

$$\begin{aligned} w_n(x) &= a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n = \\ &= \left(a_0x^{n-1} + a_1x^{n-2} + \ldots + a_{n-1}\right)x + a_n = \\ &= \left(\left(a_0x^{n-2} + a_1x^{n-3} + \ldots + a_{n-2}\right)x + a_{n-1}\right)x + a_n = \\ &= \ldots = \\ &= \left(\left(\ldots\left(\left(a_0x + a_1\right)x + a_2\right)x + \ldots + a_{n-2}\right)x + a_{n-1}\right)x + a_n \end{aligned} \tag{2.5}$$

Schemat Hornera ma więc następującą postać:

$$w_n(x) = \left(\left(\ldots\left(\left(a_0x + a_1\right)x + a_2\right)x + \ldots + a_{n-2}\right)x + a_{n-1}\right)x + a_n. \tag{2.6}$$

Porównując wzory 2.4 i 2.6 na obliczanie wartości wielomianu, łatwo zauważyć, że w schemacie Hornera wykonywana jest mniejsza liczba mnożeń.

Przykład 2.15.

Obliczmy wartość wielomianu $w(x) = 2x^3+4x^2-3x+7$ dla $x = 3$, wykorzystując schemat Hornera. Współczynnikami wielomianu są tutaj $a_0 = 2$, $a_1 = 4$, $a_2 = -3$, $a_3 = 7$, a stopień wielomianu n wynosi 3.

$$w(x) = 2x^3 + 4x^2 - 3x + 7 =$$
$$= ((2x+4)x-3)x+7$$

Stąd dla $x = 3$ mamy:

$$w(3) = 2\cdot 3^3 + 4\cdot 3^2 - 3\cdot 3 + 7 =$$
$$= ((2\cdot 3+4)\cdot 3-3)\cdot 3+7 =$$
$$= (10\cdot 3-3)\cdot 3+7 =$$
$$= 27\cdot 3+7 =$$
$$= 88$$

Porównajmy liczbę działań wykonywanych przy obliczaniu wartości wielomianu po wybraniu każdego ze wzorów:

$$w(x) = 2\cdot x\cdot x\cdot x+4\cdot x\cdot x+(-3)\cdot x+7.$$

W powyższym przykładzie wykonano sześć operacji mnożenia oraz trzy operacje dodawania.

W przypadku schematu Hornera wzór można przedstawić następująco:

$$w(x) = ((2\cdot x+4)\cdot x+(-3))\cdot x+7.$$

Liczba wykonanych działań jest tutaj znacznie mniejsza: trzy mnożenia i trzy dodawania.

Zadanie 2.17.

Opierając się na powyższej analizie, wyznacz ogólną liczbę operacji mnożenia i dodawania przy obliczaniu wartości wielomianu stopnia n. Na podstawie uzyskanych wyników podaj złożoność czasową obydwu algorytmów.

Wyznaczając wartość wielomianu schematem Hornera

$$w_n(x) = (\ldots((a_0x+a_1)x+a_2)x+\ldots+a_{n-2})x+a_{n-1})x+a_n,$$

należy wykonać następujące operacje:

$$
\begin{aligned}
w &= a_0 \\
w &= wx + a_1 \\
w &= wx + a_2 \\
w &= wx + a_3 \\
&\dots \\
w &= wx + a_{n-1} \\
w &= wx + a_n
\end{aligned}
\tag{2.7}
$$

Możemy więc zdefiniować **wzór iteracyjny** tego algorytmu:

$$
\begin{cases}
w = a_0 \\
w = wx + a_i \quad dla \quad i = 1, 2, \dots, n
\end{cases}
\tag{2.8}
$$

Na podstawie otrzymanego wzoru 2.8 konstruujemy **algorytm iteracyjny** w postaci listy kroków, schematu blokowego (patrz rysunek 2.3) oraz programów w językach C++ i Pascal.

Specyfikacja:

Dane: Liczba całkowita: $n \geq 0$ (stopień wielomianu).

n+1-elementowa tablica liczb rzeczywistych: $A[0...n]$ (współczynniki wielomianu).

Liczba rzeczywista: x (wartość argumentu).

Wynik: Wartość rzeczywista wielomianu stopnia n dla wartości argumentu x.

Lista kroków:

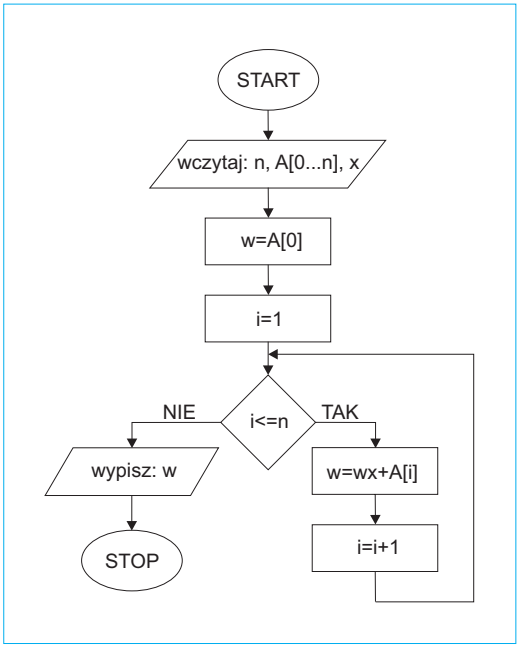
Krok 0. Wczytaj wartości danych n, $A[0...n]$, x.

Krok 1. Przypisz $w = A[0]$.

Krok 2. Dla kolejnych wartości i: 1, 2, …, n, wykonuj krok 3.

Krok 3. Przypisz $w = wx+A[i]$.

Krok 4. Wypisz wartość wielomianu: w. Zakończ algorytm.



Rysunek 2.3. Schemat blokowy algorytmu iteracyjnego wyznaczającego wartość wielomianu schematem Hornera

Funkcja w języku C++ (*prog2_7.cpp*):

```
double oblicz (double A[], int n, double x)
{
 double w=A[0];
 for (int i=1;i<=n;i++) w=w*x+A[i];
 return w;
}
```

Funkcja w języku Pascal (*prog2_7.pas*):

```
function oblicz (A: tablica; n: integer; x: real): real;
var w: real;
    i: integer;
begin
 w:=A[0];
 for i:=1 to n do w:=w*x+A[i];
 oblicz:=w
end;
```

Przedstawiony algorytm można wykonać również rekurencyjnie. Nie jest trudne zauważenie zależności rekurencyjnej, na podstawie której obliczana jest wartość wielomianu stopnia *n*.

$$w_n(x) = a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n = \underbrace{\left(a_0x^{n-1} + a_1x^{n-2} + \ldots + a_{n-1}\right)}_{w_{n-1}(x)}x + a_n = w_{n-1}(x)x + a_n \tag{2.9}$$

Na podstawie wzoru 2.9 tworzymy **definicję rekurencyjną**, która wygląda następująco:

$$w_n(x) = \begin{cases} a_0 & dla \quad n = 0 \\ w_{n-1}(x) \cdot x + a_n & dla \quad n > 0 \end{cases} \tag{2.10}$$

Zastosowanie schematu Hornera nie ogranicza się do wyznaczania wartości wielomianu stopnia *n*. Algorytm ten wykorzystywany jest również do:

- konwersji liczb z dowolnego pozycyjnego systemu liczbowego na system dziesiętny;
- szybkiego obliczania wartości potęgi;
- jednoczesnego obliczania wartości wielomianu i jego pochodnej.

Zadanie 2.18.

Napisz program obliczający rekurencyjnie wartość wielomianu stopnia *n* z wykorzystaniem schematu Hornera zgodny z podaną powyżej specyfikacją algorytmu iteracyjnego.

2.2.5. Zamiana liczb z dowolnego pozycyjnego systemu liczbowego na system dziesiętny z zastosowaniem schematu Hornera

Schemat Hornera można zastosować do konwersji liczb zapisanych w różnych systemach liczbowych na system dziesiętny. Przypomnijmy, w jaki sposób dokonujemy takiej zamiany w systemie binarnym, co zostało omówione w punkcie 2.2.2, „Konwersje pozycyjnych systemów liczbowych". Zamieńmy liczbę 1011101_2 na wartość w systemie decymalnym:

$$1011101_2 = 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 93_{10}.$$

Łatwo zauważyć, że zapis liczby podczas obliczeń przypomina wielomian. Cyfry zamienianej liczby można więc potraktować jak współczynniki wielomianu, a podstawę systemu jak wartość argumentu *x*. W tym przypadku mamy następującą sytuację:

$n = 6$,
$a_0 = 1$,
$a_1 = 0$,

$a_2 = 1$,
$a_3 = 1$,
$a_4 = 1$,
$a_5 = 0$,
$a_6 = 1$,
$x = 2$.

Taka interpretacja konwersji liczb na system dziesiętny umożliwia zastosowanie do jej realizacji schematu Hornera. Skonstruujmy więc **algorytm wykonujący zamianę liczb z systemu dwójkowego na dziesiętny**.

Specyfikacja:

Dane: Liczba całkowita: $n \geq 0$ (stopień wielomianu).

n+1-elementowa tablica liczb rzeczywistych: $A[0...n]$ (współczynniki wielomianu, czyli cyfry liczby zapisanej w systemie binarnym).

Wynik: Wartość wielomianu stopnia n dla argumentu 2 (liczba w systemie dziesiętnym).

Funkcja w języku C++ (*prog2_8.cpp*):

```
long oblicz (int A[], int n)
{
 long w=A[0];
 for (int i=1;i<=n;i++) w=w*2+A[i];
 return w;
}
```

Funkcja w języku Pascal (*prog2_8.pas*):

```
function oblicz (A: tablica; n: integer): longint;
var w: longint;
    i: integer;
begin
 w:=A[0];
 for i:=1 to n do w:=w*2+A[i];
 oblicz:=w
end;
```

Zadanie 2.19.

Podaj specyfikację zadania i skonstruuj rekurencyjny algorytm w postaci programu realizujący konwersję liczb zapisanych w systemie o podstawie zawartej w zakresie [2; 9] na system dziesiętny z zastosowaniem schematu Hornera.

Zadanie 2.20.

Podaj specyfikację zadania i skonstruuj iteracyjny algorytm w postaci programu realizujący konwersję liczb zapisanych w systemie szesnastkowym na system dziesiętny z zastosowaniem schematu Hornera.

2.2.6. Reprezentacja danych liczbowych w komputerze

Binarna reprezentacja liczb ujemnych

Dane w komputerze zapisywane są w postaci liczb binarnych. Wynika to stąd, że najmniejsza jednostka pamięci, którą jest bit, służy do zapisu jednej cyfry systemu dwójkowego: 0 lub 1. Dotychczas poznaliśmy reprezentację binarną liczb całkowitych nieujemnych. Wartości ujemne zapisuje się, **używając kodu uzupełniającego do dwóch**, zwanego **kodem U2**. Ogólny zapis liczby w kodzie U2 można przedstawić za pomocą następującego wzoru:

$$y = \begin{cases} x & dla \quad x \geq 0 \\ 2^n + x & dla \quad x < 0 \end{cases} \tag{2.11}$$

gdzie:

x — liczba, którą chcemy zapisać w kodzie U2;

n — liczba bitów przeznaczonych do zapisania kodowanej liczby;

y — liczba x zapisana za pomocą kodu U2.

Liczba y po wykonaniu obliczeń przedstawiana jest w postaci binarnej.

Zakres wartości liczby x, którą konwertujemy za pomocą kodu U2, zależy od liczby bitów przeznaczonych do zapisania tej liczby. Mając do dyspozycji n bitów, pierwszy bit rezerwujemy do oznaczenia znaku liczby (1 — liczba ujemna, 0 — liczba nieujemna), pozostałe n–1 bitów do zapisania liczby. Zauważmy, że rolę pierwszego bitu można rozumieć na dwa równoważne sposoby: reprezentuje on znak liczby x w kodzie U2, a zarazem jest pierwszym (najbardziej znaczącym) bitem nieujemnej liczby y w zwykłym układzie dwójkowym. Wartość kodowanej liczby x zawiera się więc w zakresie $[-2^{n-1}; 2^{n-1})$.

Przykład 2.16.

Załóżmy, że dysponujemy 1 bajtem (czyli 8 bitami) przeznaczonym do zapisania liczby. Stąd $n = 8$, a wartość kodowanej liczby x musi zawierać się

w zakresie $[-2^{n-1}; 2^{n-1}) = [-2^7; 2^7) = [-128_{10}; 128_{10})$. Korzystając z wzoru, wyznaczmy wartość liczby $x = -56$ w kodzie U2. Musimy wykonać następujące obliczenia:

$$y = 2^8 + (-56) = 256 - 56 = 200_{10}.$$

Następnie konwertujemy uzyskaną wartość z systemu dziesiętnego na dwójkowy:

$$200_{10} = 11001000_2.$$

Uzyskana liczba, $y = 11001000_2$, to zakodowana wartość liczby $x = -56_{10}$.

Zadanie 2.21.

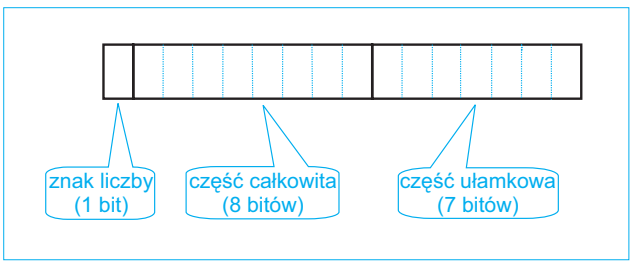
Zapisz podane liczby ujemne dla określonej wartości n za pomocą kodu U2.

a) -108_{10} dla $n = 8$ bitów,
b) -99_{10} dla $n = 8$ bitów,
c) -241_{10} dla $n = 16$ bitów,
d) -189_{10} dla $n = 16$ bitów.

Stałopozycyjna reprezentacja liczb

Stałopozycyjna reprezentacja liczb charakteryzuje się stałym położeniem przecinka, który oddziela część całkowitą od części ułamkowej zapisywanej liczby. Powoduje to, że taki zapis liczby jest dokładny tylko wtedy, gdy dana liczba nie wykracza poza zakres miejsca, jakie zostało przeznaczone do jej zapisu.

Załóżmy, że mamy do dyspozycji 2 bajty (czyli 16 bitów) do zapisania liczby w reprezentacji stałopozycyjnej. Wówczas podział na część całkowitą i ułamkową może przedstawiać się jak na rysunku 2.4.



Rysunek 2.4. Przykładowy podział na część całkowitą i ułamkową w stałopozycyjnej reprezentacji liczb

W reprezentacji stałopozycyjnej liczba zapisywana jest w kodzie uzupełniającym do dwóch.

Potrafimy już wykonywać konwersję liczby całkowitej pomiędzy systemami binarnym i decymalnym. Jednak aby przedstawić liczbę rzeczywistą, wykorzystując reprezentację stałopozycyjną, trzeba uwzględnić również część ułamkową.

Konwertując liczby z systemu binarnego na decymalny, mnożymy kolejne cyfry tej liczby przez potęgi dwójki. W części ułamkowej mnożnikiem są ujemne potęgi liczby 2.

Przykład 2.17.

Zapiszmy liczbę rzeczywistą $101111{,}01101_2$ w systemie dziesiętnym. Zaczynamy od dopasowania potęg liczby 2 do kolejnych cyfr podanej wartości:

1	0	1	1	1	1	,	0	1	1	0	1
2^5	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}

Następnie obliczamy wartość konwertowanej liczby w systemie dziesiętnym:

$$101111{,}01101_2 = 1\cdot 2^5 + 0\cdot 2^4 + 1\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 + 0\cdot 2^{-1} + 1\cdot 2^{-2} + 1\cdot 2^{-3} + 0\cdot 2^{-4} + 1\cdot 2^{-5} = 47\frac{13}{32}_{10}.$$

Zamiana liczby ułamkowej z systemu dziesiętnego na dwójkowy wykonywana jest przez mnożenie części ułamkowej przez 2 tak długo, aż w części ułamkowej uzyskamy zero lub zauważymy, że wynikiem jest ułamek nieskończony. Rozwiązaniem jest liczba utworzona z całkowitych części wyników uzyskiwanych podczas mnożenia liczby przez 2.

Przykład 2.18.

Przekonwertujmy liczbę ułamkową $0{,}1825_{10}$ na system binarny.

W tym celu należy wykonać mnożenie części ułamkowej tej liczby przez 2:

0,1825	0,1825·2	=	0,375
0,375	0,375·2	=	0,75
0,75	0,75·2	=	1,5
1,5	0,5·2	=	1,0
1,0			

Rozwiązaniem jest ułamek skończony. Widać to po wartości ostatniego wyniku 1,0, w którym część ułamkowa wynosi zero. Uzyskaliśmy więc następujące rozwiązanie:

$0{,}1825_{10} = 0{,}\mathbf{0011}_2$.

Poniżej pokazany został czytelniejszy zapis konwersji liczb ułamkowych z systemu decymalnego na binarny:

0	1825
0	375
0	75
1	5
1	0

Przykład 2.19.

Przekonwertujmy liczbę $0{,}2_{10}$ na system binarny. Rozwiązaniem będzie ułamek nieskończony.

0	2
0	4
0	8
1	6
1	2
0	4
0	8
1	6
1	2
0	4

…

Łatwo zauważyć, że sekwencja liczb „0011” będzie się powtarzać. Wynikiem jest więc ułamek okresowy $0{,}(0011)_2$.

Zadanie 2.22.

Przekonwertuj podane liczby rzeczywiste na system dziesiętny:

a) $10100{,}11101_2$,
b) $0{,}0111011_2$,
c) $11{,}110001_2$,
d) $10110011{,}11100101_2$,
e) $11011100{,}10010101_2$.

Zadanie 2.23.

Przekonwertuj podane liczby rzeczywiste na system binarny:

a) $852{,}6875_{10}$,
b) $620{,}09375_{10}$,

c) $612{,}03125_{10}$,

d) $1536{,}9921875_{10}$,

e) $2707{,}7734375_{10}$.

Zadanie 2.24.

Wykonaj następujące operacje arytmetyczne w systemie binarnym:

a) $1110{,}011_2 \cdot 10001{,}00111_2$,

b) $1111{,}001_2 + 100000{,}11011_2$,

c) $10011{,}01011_2 - 100{,}1011_2$.

Zmiennopozycyjna reprezentacja liczb

Zmiennopozycyjna reprezentacja liczb charakteryzuje się zmiennym położeniem przecinka, które zależy od zapisywanej liczby. Ogólny wzór na wartości w tej reprezentacji przedstawia się następująco:

$$liczba = m \cdot 2^c, \tag{2.12}$$

gdzie:

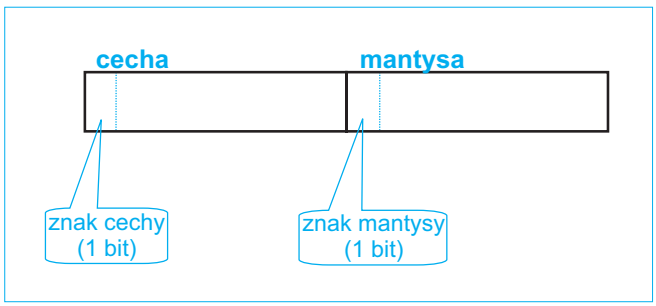
liczba — liczba, którą chcemy zapisać w reprezentacji zmiennopozycyjnej;

m — mantysa (ułamek właściwy);

c — cecha (liczba całkowita).

Ponadto mantysa powinna spełniać warunek $|m| \in [0{,}5;\ 1)$. Wówczas kodowana liczba jest w **postaci znormalizowanej**.

Aby wyznaczyć wartość liczby zapisanej w reprezentacji zmiennopozycyjnej, trzeba znać wartość mantysy i cechy. Na rysunku 2.5 przedstawiono graficznie zapis zmiennopozycyjny, z uwzględnieniem podziału na cechę i mantysę. Cecha i mantysa zapisywane są jako liczby stałopozycyjne w kodzie U2.



Rysunek 2.5. Przykładowy podział na cechę i mantysę w zmiennopozycyjnej reprezentacji liczb

Przykład 2.20.

Załóżmy, że daną mamy liczbę 0000111011100001 zapisaną w reprezentacji zmiennopozycyjnej. Podana liczba zajmuje 2 bajty (czyli 16 bitów), z czego 7 bitów to cecha, a pozostałe 9 bitów to mantysa. Wówczas liczba składa się z następujących elementów:

- 0 — bit znaku cechy,
- 000111 — cecha,
- 0 — bit znaku mantysy,
- 11100001 — mantysa.

Zarówno cecha, jak i mantysa to w tym przypadku liczby nieujemne, o czym świadczą bity znaku równe zero.

Wyznaczmy wartości cechy i mantysy:

$$c = 111_2 = 1\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 = 7_{10},$$

$$m = 0{,}11100001_2 = 1\cdot 2^{-1} + 1\cdot 2^{-2} + 1\cdot 2^{-3} + 0\cdot 2^{-4} + 0\cdot 2^{-5} + 0\cdot 2^{-6} + 0\cdot 2^{-7} + 1\cdot 2^{-8} = \frac{225}{256_{10}} = 0{,}87890625_{10}.$$

Następnie, korzystając z podanego wzoru 2.12, obliczamy wartość zakodowanej liczby:

$$liczba = m\cdot 2^c = \frac{225}{256}\cdot 2^7 = 0{,}87890625\cdot 2^7 = 112{,}5_{10}.$$

W reprezentacji zmiennopozycyjnej nie każdą liczbę rzeczywistą można zapisać dokładnie. Liczby te są najczęściej reprezentowane w sposób przybliżony. Na dokładność ma wpływ liczba cyfr w mantysie, natomiast od liczby cyfr w cesze zależy, jak duże liczby mogą być zapisywane.

Zadanie 2.25.

Wyznacz wartości dziesiętne liczb podanych w reprezentacji zmiennopozycyjnej (cecha i mantysa oddzielone są odstępem):

a) 000000010 0110011,
b) 0001010 010000101,
c) 0000011 010100001.

***MATURA** — egzamin styczeń 2006 r. Arkusz I, zadanie 3.*

MATURA** — Sylabus od 2009 r. Arkusz I poziom podstawowy, zadanie 2.* ***KRAJE

***MATURA** — Sylabus od 2009 r. Arkusz II poziom podstawowy, zadanie 5.*

DODAWANIE LICZB TRÓJKOWYCH