DO NOWEJ PODSTAWY PROGRAMOWEJ

Część 2

PODRĘCZNIK dla szkół ponadpodstawowych

Informatyka Europejczyka

Zakres podstawowy

Danuta Korman,
Grażyna Szabłowicz-Zawadzka

Helion EDUKACJA

Redaktor prowadzący: Joanna Zaręba

Ilustracja na okładce została wykorzystana za zgodą Shutterstock.

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie?ieppp2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-5897-3

Printed in Poland.

Spis treści

Rozdział 1.

ROZUMIENIE I ANALIZOWANIE PROBLEMÓW. WPROWADZENIE DO PROGRAMOWANIA W JĘZYKU PYTHON

Co już potrafisz?

Podajesz i wykorzystujesz **etapy rozwiązywania problemów za pomocą komputera**.

Potrafisz przedstawiać algorytmy w postaci **schematów blokowych i list kroków**.

Konstruujesz proste **algorytmy iteracyjne i rekurencyjne**.

Znasz podstawy **tekstowego języka programowania Python** oraz korzystasz ze **środowiska programowania** dla tego języka.

Sprawdzasz poprawność algorytmów poprzez testowanie dla przykładowych danych.

Czego się nauczysz?

Poznasz i będziesz stosować proste **algorytmy na liczbach, ciągach liczbowych i tekstach**.

Nauczysz się wykorzystywać **sekwencyjne typy danych w języku Python**: listy i napisy.

Dowiesz się, czym jest **kryptografia i kryptoanaliza**. Poznasz proste **metody przestawieniowe i podstawieniowe** stosowane w kryptografii.

Co pojawi się na poziomie rozszerzonym?

Poszerzysz znajomość **tekstowego języka programowania dopuszczonego na egzaminie maturalnym** z informatyki rozszerzonej. Może to być również język Python, którego podstawy poznajesz na lekcjach informatyki na poziomie podstawowym.

Będziesz omawiać oraz stosować **zaawansowane algorytmy iteracyjne** i **rekurencyjne** na liczbach, ciągach liczbowych i tekstach.

Zajmiesz się **analizą efektywności omawianych algorytmów**.

Zapoznasz się z **zadaniami z egzaminu maturalnego z informatyki** i będziesz je rozwiązywać.

Temat 1. Badanie, czy dana liczba jest liczbą pierwszą

Definicja

Liczbę naturalną n większą od 1 nazywamy **liczbą pierwszą**, jeśli ma tylko dwa dzielniki: 1 i n. Liczbę naturalną większą od 1, która nie jest liczbą pierwszą, nazywamy **liczbą złożoną**. Natomiast liczby 0 i 1 nie są ani liczbami złożonymi, ani pierwszymi.

Najprostszym algorytmem określającym, czy liczba n to liczba pierwsza, jest sprawdzenie, czy ma ona więcej niż dwa dzielniki. Należy więc zbadać, czy w przedziale $[2, n-1]$ znajduje się co najmniej jedna wartość całkowita, przez którą dzieli się liczba n.

Ćwiczenie 1.1.

Przeanalizuj podany kod programu i odpowiedz na pytania:

a) Jaki komunikat pojawi się na ekranie po uruchomieniu przedstawionego programu dla wpisanej z klawiatury liczby n o wartości 12, a jaki dla liczby n o wartości 16?

b) Co jest efektem działania podanego algorytmu dla dowolnej liczby naturalnej n?

```
n = int(input("podaj liczbę: "))
i = 1
while i * i < n:
    if n % i == 0:
        print(i, n // i)
    i += 1
if i * i == n:
    print(i)
```

Skonstruujmy **algorytm sprawdzający, czy liczba naturalna *n* jest liczbą pierwszą**. W tym celu sprawdzimy, czy w przedziale $[2, n-1]$ znajduje się co najmniej jedna wartość całkowita, przez którą dzieli się liczba n.

Specyfikacja:

Dane: liczba naturalna: $n > 1$.

Wynik: komunikat informujący, czy liczba n jest liczbą pierwszą.

Program w języku Python:

```
def liczba_pierwsza(n):
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

print(liczba_pierwsza(61))
```

Wynik:

True

Przedstawiona funkcja jest typu logicznego. Jeśli po jej wykonaniu otrzymamy wartość `True`, liczba n jest liczbą pierwszą. W przeciwnym razie n jest liczbą złożoną.

Spróbujmy zmodyfikować powyższy algorytm w taki sposób, aby poprawić jego złożoność obliczeniową. W obecnej postaci złożoność tej metody jest liniowa. Wynika to stąd, że liczba operacji dominujących, czyli porównań, wynosi $n-2$.

Zauważ, że nie ma konieczności sprawdzania wszystkich liczb z przedziału $[2, n-1]$. Załóżmy, że istnieje liczba x większa niż $\sqrt{n}$, która jest dzielnikiem liczby n. Wynika stąd, że musi istnieć liczba y, będąca również dzielnikiem liczby n, taka, że $n = x \cdot y$. Liczba y musiałaby jednak być mniejsza od $\sqrt{n}$, a to oznacza, że zostałaby znaleziona już w zakresie $[2, \sqrt{n}]$. W rzeczywistości wystarczy więc sprawdzenie, czy w zakresie $[2, \sqrt{n}]$ znajduje się wartość, która jest dzielnikiem liczby n.

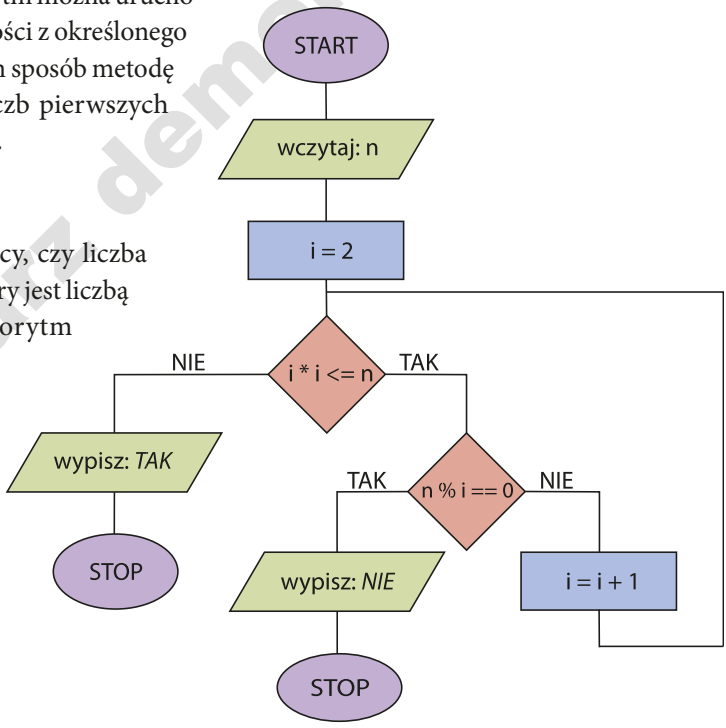
Na rysunku 1.1 został przedstawiony **schemat blokowy zmodyfikowanego algorytmu sprawdzającego, czy dana liczba jest liczbą pierwszą**.

Liczba operacji dominujących, czyli porównań, w tym algorytmie jest tym większa, im później zostaje znaleziona liczba będąca dzielnikiem n. Najwięcej porównań zostanie wykonanych w sytuacji, gdy n będzie liczbą pierwszą.

Funkcję realizującą ten algorytm można uruchomić w pętli generującej wartości z określonego przedziału. Uzyskujemy w ten sposób metodę wyznaczania wszystkich liczb pierwszych z podanego przedziału liczb.

Ćwiczenie 1.2.

Napisz program sprawdzający, czy liczba naturalna wpisana z klawiatury jest liczbą pierwszą. Skonstruuj algorytm zgodny ze schematem blokowym przedstawionym na rysunku 1.1.



Rysunek 1.1. Schemat blokowy algorytmu sprawdzającego, czy dana liczba jest liczbą pierwszą

Zadanie 1.1.

Napisz program, który dla liczby naturalnej n wpisanej z klawiatury wykonuje następujące operacje:

a) wypisuje wszystkie dzielniki naturalne liczby n,

b) wypisuje wszystkie dzielniki pierwsze liczby n.

Zadanie 1.2.

Napisz program sprawdzający, czy wpisana z klawiatury liczba naturalna jest podzielna przez 3. Wykonaj to zadanie bez wykorzystania operatora reszty z dzielenia. Zauważ, że gdy zsumujesz cyfry wczytanej liczby, potem cyfry powstałej sumy i będziesz powtarzać to sumowanie, aż uzyskasz jedną cyfrę, otrzymasz odpowiedź. Jeśli cyfrą, którą uzyskasz, będzie 3, 6 lub 9, to wczytana liczba jest podzielna przez 3.

Na przykład dla liczby 32 415 należy obliczyć sumę 3 + 2 + 4 + 1 + 5 = 15, a następnie 1 + 5 = 6. Wynikiem jest cyfra 6, więc liczba 32 415 jest podzielna przez 3.

Zadanie 1.3.

Parą **liczb bliźniaczych** nazywamy dwie liczby pierwsze różniące się o 2. Przykładem liczb bliźniaczych są liczby 11 i 13, ponieważ obie są liczbami pierwszymi i różnica pomiędzy nimi wynosi 2. Przykładem liczb, które nie są bliźniacze, jest para liczb 15 i 17, ponieważ 15 jest liczbą złożoną.

a) Podaj specyfikację zadania i napisz program, który sprawdza, czy dwie liczby naturalne wpisane z klawiatury są liczbami bliźniaczymi.

b) Napisz program generujący wszystkie pary liczb bliźniaczych, które są nie większe od wpisanej z klawiatury liczby k.

Temat 2. Sekwencyjne typy danych — listy

Definicja

Lista to typ sekwencyjny zmienny, a więc możliwe jest **przypisywanie wartości pojedynczym elementom** tego typu. Do zapisu listy wykorzystujemy **nawiasy kwadratowe**, a poszczególne elementy rozdzielamy przecinkami. Listy **mogą zawierać wartości różnego typu**. Każdy element listy ma przyporządkowany **indeks**. Elementy listy **są numerowane od zera**.

Deklaracja listy ma następującą składnię:

```
lista = [wartości listy]
```

Przykład 2.1.

Przyjrzyjmy się przykładowej deklaracji listy:

```
T = [1.0, 2, "trzy", 4.0, 5]
```

Zadeklarowano 5-elementową listę zawierającą wartości różnego typu, które są ponumerowane od 0 do 4. Do elementów listy odwołujemy się poprzez ich indeksy. Uzyskaliśmy więc dostęp do następujących zmiennych: `T[0]`, `T[1]`, `T[2]`, `T[3]`, `T[4]`. Na przykład `T[0]` to odwołanie do pierwszego elementu listy o wartości 1,0, a `T[2]` to odwołanie do elementu listy o wartości „trzy".

W języku Python możemy stosować również indeksy ujemne, które umożliwiają odwoływanie się do poszczególnych elementów począwszy od ostatniego. Na przykład `T[-1]` to odwołanie do elementu o wartości 5, a `T[-4]` to odwołanie do elementu o wartości 2.

Przykład 2.2.

Przeanalizuj odwołania do elementów przykładowej listy podanej poniżej.

```
T = [1, 4, 0, 3, 2]
```

Uzyskujemy 5-elementową listę zawierającą liczby całkowite o wartościach:

`T[0]` = 1; `T[1]` = 4; `T[2]` = 0; `T[3]` = 3; `T[4]` = 2.

Dostęp do elementów listy można uzyskać również w sposób następujący:

`T[-1]` = 2; `T[-2]` = 3; `T[-3]` = 0; `T[-4]` = 4; `T[-5]` = 1.

Możesz wypisać na ekranie zarówno całą listę, `print(T)`, jak i pojedyncze elementy tej listy, na przykład `print(T[2])`.

Ćwiczenie 2.1.

Napisz program wypisujący na ekranie trzy elementy listy, których numery wprowadzane są z klawiatury.

Przykład 2.3.

W języku Python mamy dostęp do funkcji `len()`, której wartością jest liczba elementów listy. Na przykład dla listy `T = [1, 4, 0, 3, 2]` funkcja `len(T)` zwraca wartość 5, co jest równe liczbie elementów tej listy. Przeanalizuj przykład zastosowania tej metody do wypisywania wszystkich elementów listy.

```
T = [1, 4, 0, 3, 2]
print(T)
for i in range(len(T)):
    print(T[i], end='  ')
```

Po uruchomieniu tego programu pojawiają się następujące wyniki:

[1, 4, 0, 3, 2]

1 4 0 3 2

Przykład 2.4.

W języku Python elementy listy możemy wprowadzać z klawiatury. Wykorzystujemy do tego metodę `append()`, co przedstawiono poniżej:

```
T = []
n = int(input("podaj liczbę elementów listy: "))
for i in range(n):
    T.append(int(input()))
print("lista:", T)
```

Po uruchomieniu tego programu pojawiają się następujące wyniki:

podaj liczbę elementów listy: 6
9
7
8
6
7
5
lista: [9, 7, 8, 6, 7, 5]

Ćwiczenie 2.2.

Napisz program, który wypisze na ekranie wszystkie elementy listy wprowadzonej z klawiatury w odwrotnej kolejności.

Przykład 2.5.

Przeanalizuj kody programów wykonujących następujące operacje:

a) obliczenie sumy wszystkich elementów listy,

b) obliczenie iloczynu tych elementów listy, które są mniejsze od 6,

c) obliczenie liczby tych elementów listy, których numer nie jest podzielny przez 5,

d) wyzerowanie tych elementów listy, które mają nieparzysty numer zawarty w przedziale [3, 7], i wyświetlenie zmodyfikowanej listy.

Rozwiązanie 1.:

```
T = [3, 4, 5, 5, 7, 9, 4, 2, 1]

# zad_a
s = 0
for i in range(len(T)):
    s += T[i]
print("suma =", s)
```

```
# zad_b
s = 1
for i in range(len(T)):
    if T[i] < 6:
        s *= T[i]
print("iloczyn =", s)

# zad_c
s = 0
for i in range(len(T)):
    if i % 5 != 0:
        s += 1
print("liczba elementów =", s)

# zad_d
for i in range(3, 8, 2):
    T[i] = 0
print("wyzerowana lista =", T)
```

Wyniki:

suma = 40

iloczyn = 2400

liczba elementów = 7

wyzerowana lista = [3, 4, 5, 0, 7, 0, 4, 0, 1]

Rozwiązanie 2.:

```
T = [3, 4, 5, 5, 7, 9, 4, 2, 1]

# zad_a
s = 0
for k in T:
    s += k
print("suma =", s)

# zad_b
s = 1
for k in T:
    if k < 6:
        s *= k
print("iloczyn =", s)
```

Wyniki:

suma = 40

iloczyn = 2400

Zwróć uwagę na dwie propozycje rozwiązania dla podpunktów a) i b) tego zadania. Dlaczego nie można w ten sposób rozwiązać pozostałej części zadania?

Zadanie 2.1.

Napisz program wykonujący następujące operacje na liście zawierającej liczby całkowite:

a) wypisanie wszystkich elementów listy,

b) obliczenie liczby tych elementów listy, których numer jest parzysty,

c) zwiększenie o 2 wartości tych elementów listy, których wartość jest mniejsza od 5,

d) obliczenie iloczynu tych elementów listy, których wartość jest równa 3,

e) obliczenie sumy tych elementów listy, których numer jest podzielny przez 3.

Zadanie 2.2.

Napisz program wykonujący następujące operacje na liście zawierającej liczby całkowite:

a) obliczenie liczby elementów listy równych wartości wczytanej z klawiatury,

b) obliczenie średniej arytmetycznej wszystkich elementów listy,

c) obliczenie średniej arytmetycznej tych elementów tablicy, których wartość jest nieparzysta.

Przykład 2.6.

W tabeli 2.1 podano dodatkowe informacje na temat list i ich indeksowania. Na podstawie przykładów pokazanych w tej tabeli przeanalizuj sposób indeksowania list i porównaj go z zasadami generowania ciągu liczbowego za pomocą metody `range()`. Czy zauważasz podobieństwa? Co oznaczają kolejne wartości oddzielone dwukropkami w indeksie listy?

Tabela 2.1. Indeksowanie list

Polecenie	Wyniki
`lista = [9, 2, 3, 4, 5, 6, 7, 8, 1, 10]`	
`print(lista)`	*[9, 2, 3, 4, 5, 6, 7, 8, 1, 10]*
`print(lista[2:])`	*[3, 4, 5, 6, 7, 8, 1, 10]*
`print(lista[3::2])`	*[4, 6, 8, 10]*
`print(lista[::-1])`	*[10, 1, 8, 7, 6, 5, 4, 3, 2, 9]*
`print(lista[::-2])`	*[10, 8, 6, 4, 2]*
`print(lista[::3])`	*[9, 4, 7, 10]*
`print(lista[:4:-2])`	*[10, 8, 6]*
`print(lista[:4:2])`	*[9, 3]*
`print(lista[0:5:2])`	*[9, 3, 5]*
`print(lista[3:9:2])`	*[4, 6, 8]*

Ćwiczenie 2.3.

Na podstawie przykładu 2.6 napisz program, w którym przetestujesz różne możliwości indeksowania listy wpisanej z klawiatury.

W tabeli 2.2 podano wybrane dodatkowe metody stosowane na listach, które możesz wykorzystywać w programach.

Tabela 2.2. Wybrane metody stosowane na listach

Polecenie	Opis polecenia i wyniki
`lista = [9, 2, 3, 4, 5, 6, 7, 8, 1, 10]`	
`lista.append(11)`	dołączanie elementu do listy, wynik: *[9, 2, 3, 4, 5, 6, 7, 8, 1, 10, 11]*
`lista.extend([12, 13])`	dołączanie listy *[12, 13]* do listy *[9, 2, 3, 4, 5, 6, 7, 8, 1, 10]*, wynik: *[9, 2, 3, 4, 5, 6, 7, 8, 1, 10, 12, 13]*
`lista.count(9)`	obliczanie, ile razy podana wartość 9 występuje na liście, wynik: *1*
`lista.index(8)`	wyznaczanie pozycji (licząc od 0) pierwszego wystąpienia podanej wartości 8, wynik: *7*
`lista.insert(3, 33)`	wstawianie liczby 33 do listy na podaną pozycję 3., wynik: *[9, 2, 3, 33, 4, 5, 6, 7, 8, 1, 10]*
`lista.pop(4)`	zwracanie wartości z podanej pozycji 4. i usunięcie tego elementu z listy, wynik: *[9, 2, 3, 4, 6, 7, 8, 1, 10]*
`lista.remove(2)`	usunięcie z listy pierwszej znalezionej wartości 2, wynik: *[9, 3, 4, 5, 6, 7, 8, 1, 10]*
`lista.reverse()`	odwracanie kolejności elementów listy, wynik: *[10, 1, 8, 7, 6, 5, 4, 3, 2, 9]*
`lista.sort()`	porządkowanie listy w porządku niemalejącym, wynik: *[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]*

Ćwiczenie 2.4.

Napisz program, który wykonuje następujące operacje na wczytanej z klawiatury liście zawierającej liczby całkowite. Zastosuj na listach metody podane w tabeli 2.2:

a) obliczenie, ile razy liczba 6 występuje w liście,

b) wyznaczenie pozycji pierwszego wystąpienia liczby 8 w liście,

c) wstawienie liczby 15 do listy na pozycję 0,

d) usunięcie z listy pierwszej znalezionej liczby 10,

e) odwrócenie kolejności elementów listy,

f) uporządkowanie listy w porządku niemalejącym.

Zadanie 2.3.

Napisz program wykonujący następujące operacje na 12-elementowej liście zawierającej liczby całkowite, której wartości wpisywane są z klawiatury:

a) wypisanie wszystkich elementów listy,

b) obliczenie liczby tych elementów listy, których numer nie jest podzielny przez 5,

c) zwiększenie o 2 wartości tych elementów listy, które mają nieparzysty numer zawarty w przedziale [3, 9],

d) obliczenie iloczynu tych elementów listy, których wartość jest równa 5,

e) obliczenie liczby tych elementów listy, których wartość nie zawiera się w przedziale [5, 8].

Zadanie 2.4.

Napisz program wykonujący następujące operacje na 10-elementowej liście zawierającej liczby całkowite, której wartości wpisywane są z klawiatury:

a) wypisanie wszystkich elementów listy,

b) obliczenie sumy tych elementów listy, których wartość jest większa od 6,

c) wypisanie tych elementów listy, których indeks jest zawarty w przedziale [3, 7],

d) wyzerowanie tych elementów listy, których indeks jest podzielny przez 3,

e) obliczenie średniej arytmetycznej wszystkich elementów listy i wypisanie tych elementów, których wartość jest mniejsza od wyznaczonej średniej.

Zadanie 2.5.

Napisz program wykonujący następujące operacje na 9-elementowej liście zawierającej liczby rzeczywiste, której wartości wprowadzane są w programie:

a) wypisanie wszystkich elementów listy,

b) obliczenie sumy tych elementów listy, których indeks jest podzielny przez 4,

c) zmniejszenie o 5 wartości tych elementów listy, które mają wartość większą od 0,

d) wypisanie tych elementów listy, których indeks jest nieparzysty i zawiera się w przedziale [1, 5],

e) obliczenie liczby tych elementów listy, których wartość jest zawarta w przedziale [5, 21].

Temat 3. Pozycyjne systemy liczbowe

3.1. Systemy liczbowe

Definicja

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

Definicja

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

W życiu codziennym korzystamy z **systemu dziesiętnego**, zwanego decymalnym, którego podstawą jest dziesięć. 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 112_{16} mógłby oznaczać liczbę 112_{16} lub $B2_{16}$, lub $1C_{16}$.

Wskazówka

W języku Python można skorzystać z **wbudowanych funkcji, które wykonują konwersję** liczb całkowitych z systemu dziesiętnego na liczby w innym systemie pozycyjnym.

```
x = 209
print(oct(x), bin(x), hex(x))
```

Wynikiem działania tego programu jest poniższy komunikat:

0o321 0b11010001 0xd1

Oznacza on, że liczba 209 zapisana w systemie decymalnym jest równa liczbie 321 w systemie oktalnym, 11010001 w systemie binarnym oraz *D*1 w systemie heksadecymalnym.

Wskazówka

Przy wykonywaniu konwersji i działań arytmetycznych w różnych systemach liczbowych można zastosować udostępnioną w systemie Windows **aplikację Kalkulator**. Program ten umożliwia przeprowadzanie obliczeń w następujących systemach: decymalnym (czyli dziesiętnym), binarnym, oktalnym i heksadecymalnym. Wykonywać można zarówno konwersję pomiędzy wymienionymi systemami, jak i operacje arytmetyczne.

Ćwiczenie 3.1.

Napisz program, który wykona konwersję wpisanej z klawiatury liczby całkowitej podanej w systemie dziesiętnym na liczbę we wskazanym systemie:

a) ósemkowym,

b) dwójkowym,

c) szesnastkowym.

Skorzystaj z funkcji wbudowanych języka Python.

Zadanie 3.1.

Napisz program, który dla wpisanej z klawiatury liczby naturalnej *n* wykonuje następujące operacje:

a) wypisuje cyfrę jedności liczby *n*,

b) wypisuje cyfrę setek liczby *n*,

c) wypisuje kolejne cyfry liczby *n*, rozpoczynając od cyfry jedności,

d) oblicza sumę cyfr liczby *n*.

Zadanie 3.2.

Liczba pseudobinarna to liczba, która w systemie dziesiętnym jest zapisana tylko za pomocą cyfr 1 lub 0. Przykładami takich liczb są 110011, 10110, natomiast liczbami pseudobinarnymi nie są liczby 345, 2091. Podaj specyfikację zadania i napisz program, który sprawdza, czy liczba wpisana z klawiatury jest liczbą pseudobinarną.

Temat 4. Zamiany reprezentacji liczb pomiędzy pozycyjnymi systemami liczbowymi

4.1. Konwersja liczb z systemu dziesiętnego na liczby w innym pozycyjnym systemie liczbowym

Definicja

Aby **zamienić liczbę nieujemną zapisaną w systemie decymalnym na liczbę w systemie binarnym**, należy powtarzać dzielenie z resztą tej liczby przez podstawę systemu dwójkowego, dopóki w wyniku dzielenia nie uzyska się 0. Wówczas otrzymane reszty z dzielenia czytane od końca stanowią rozwiązanie.

Przykład 4.1.

Przeanalizujmy konwersję z 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 otrzymaliśmy 0, 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	

Przeanalizuj podany kod programu, który realizuje **zamianę liczby z systemu decymalnego na liczbę w systemie binarnym i wypisuje kolejno cyfry wyniku na ekranie**. Jaką metodę programowania zastosowano w tym algorytmie? Zauważ, że funkcja `oblicz()` uruchamia sama siebie.

```
def oblicz(n):
    if n > 0:
        oblicz(n // 2)
        print(n % 2, end='')

liczba = int(input("podaj liczbę naturalną: "))
oblicz(liczba)
```

Zadanie 4.1.

Skonstruuj **algorytm iteracyjny** w postaci programu, który wykonuje zamianę liczb z systemu decymalnego na liczby w systemie binarnym i wypisuje kolejno cyfry wyniku na ekranie.

Przeanalizuj **iteracyjny algorytm** w postaci schematu blokowego **wykonujący zamianę liczb zapisanych w systemie decymalnym na liczby binarne** (rysunek 4.1), w którym uzyskany **wynik jest zapisywany jako lista**.

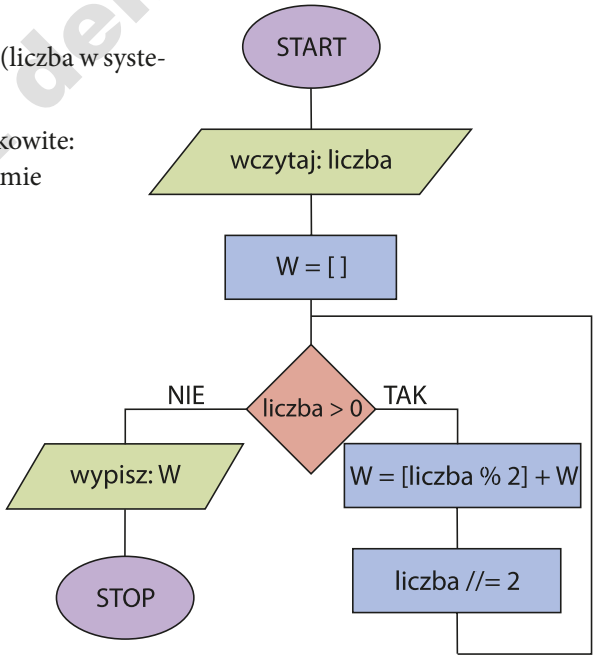
Specyfikacja:

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

Wynik: lista zawierająca liczby całkowite: *W* (liczba zapisana w systemie dwójkowym uzyskana po zamianie z systemu dziesiętnego).

Ćwiczenie 4.1.

Napisz program na podstawie schematu blokowego (rysunek 4.1) wykonujący **iteracyjny algorytm** zamiany liczb z systemu decymalnego na liczby w systemie binarnym, w którym uzyskany wynik jest zapisywany jako lista.



Rysunek 4.1. Schemat blokowy algorytmu realizującego konwersję liczb z systemu dziesiętnego na liczby w systemie dwójkowym

Zadanie 4.2.

Skonstruuj **algorytm rekurencyjny** w postaci programu realizujący zamianę liczb z systemu decymalnego na liczby w systemie binarnym, w którym uzyskany wynik jest zapisywany jako lista.

Zadanie 4.3.

Napisz programy, które wykonają następujące polecenia:

a) obliczanie liczby cyfr w rozwinięciu binarnym liczby naturalnej wprowadzonej z klawiatury,

b) zamianę n liczb wprowadzonych z klawiatury z systemu dziesiętnego na liczby w systemie binarnym.

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

Przykład 4.2.

Zapiszmy liczbę 459_{10} w systemie szesnastkowym. Zwróć uwagę na cyfry, których wartość jest większa niż 9, a które w systemie szesnastkowym zapisujemy literami od A do F.

$$\begin{array}{rcrcr|l} 459 & : & 16 & = & 28 & \text{reszta } \mathbf{11} = B \\ 28 & : & 16 & = & 1 & \text{reszta } \mathbf{12} = C \\ 1 & : & 16 & = & 0 & \text{reszta } \mathbf{1} \end{array}$$

Poniżej przedstawiono skrócony zapis konwersji tych liczb:

$$\begin{array}{r|l} 459 & 11 = B \\ 28 & 12 = C \\ 1 & 1 \\ 0 & \end{array}$$

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

Zadanie 4.4.

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

a) 1234_{10},

b) 999_{10},

c) 1380_{10},

d) 49_{10},

e) 2135_{10}.

Zadanie 4.5.

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

Zadanie 4.6.

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

4.2. Konwersja liczb w innych pozycyjnych systemach liczbowych na liczby w systemie dziesiętnym

Definicja

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

Przykład 4.3.

Przeanalizuj przebieg działania tej metody na przykładzie liczbowym. Wykonaj konwersję liczby 1011011_2 z systemu binarnego na liczbę w systemie decymalnym. 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 wyznacz 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 liczby w systemie decymalnym**, postępujemy podobnie. Musimy jednak pamiętać o tym, by kolejne cyfry konwertowanej liczby mnożyć przez potęgi podstawy systemu, w którym jest zapisana.

Przykład 4.4.

Zapisz liczbę $1A0B_{12}$ w systemie decymalnym. Najpierw dopasuj odpowiednie potęgi podstawy systemu do cyfr tej liczby:

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

Następnie wykonaj obliczenia:

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

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

Zadanie 4.7.

Oblicz wartości podanych liczb 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) $1A17B0_{16}$.

Zadanie 4.8.

Podaj specyfikację zadania i skonstruuj algorytm w postaci programu realizujący konwersję liczb binarnych na liczby w systemie dziesiętnym.

Zadanie 4.9.

Podaj specyfikację zadania i skonstruuj algorytm w postaci programu realizujący konwersję liczb zapisanych w systemie o podstawie zawartej w przedziale [2, 9] na liczby w systemie dziesiętnym.

Temat 5. Liniowe porządkowanie ciągu liczbowego

Sortowanie ciągu liczbowego polega na uporządkowaniu jego wyrazów rosnąco, malejąco, nierosnąco lub niemalejąco, a więc tak, aby ciąg niemonotoniczny został przekształcony w monotoniczny. W przedstawionych algorytmach sortowania będziemy przekształcać ciąg liczbowy $T[0 \ldots n-1]$ w ciąg monotoniczny, którego każda para kolejnych wyrazów będzie spełniać warunek $T[i] <= T[i+1]$, dla $i = 0, 1, \ldots, n-2$. Naszym celem będzie więc uzyskanie ciągu niemalejącego, ewentualnie rosnącego.

Istnieją różne typy metod sortujących. Najpopularniejszą grupę algorytmów stanowią **metody oparte na porównywaniu wyrazów ciągu** i ich zamianie. Należą do nich m.in. porządkowanie bąbelkowe, przez wybór, przez wstawianie. W szkole podstawowej pojawiła się już metoda sortowania przez wybór. W tym temacie zajmiemy się pozostałymi metodami.

Algorytm sortowania wybieramy w zależności od rodzaju danych, które chcemy uporządkować.

Poniżej podane zostały przykłady odpowiednio uporządkowanych ciągów liczbowych:

- porządek rosnący: 1, 3, 5, 36, 672, 7892;
- porządek malejący: 6789, 437, 45, 3, 2;
- porządek niemalejący: 3, 4, 4, 7, 9, 9, 12, 16;
- porządek nierosnący: 17, 17, 16, 8, 8, 8, 3, 1.

Ćwiczenie 5.1.

Podczas porządkowania zapisanego w liście ciągu liczbowego metodą opartą na porównywaniu elementów trzeba wykonać **zamianę wybranych elementów**. W języku Python można to zapisać w następujący sposób:

```
a, b = b, a
```

W ten sposób zmiennej `a` przypisano wartość zmiennej `b`, a zmiennej `b` — wartość zmiennej `a`.

Napisz program, w którym wykonasz zamianę elementów dla zmiennych `a` i `b` wprowadzonych z klawiatury, a następnie dla elementów listy o indeksach podanych z klawiatury.

Zadanie 5.1.

Prześledź działanie algorytmu na przykładzie liczbowym podanym poniżej, który pokazuje kolejne kroki sortowania ciągu liczbowego (8, 2, 0, 1, 6).

	8	2	0	1	6
Krok 1.	**0**	2	**8**	1	6
Krok 2.	0	**1**	8	**2**	6
Krok 3.	0	1	**2**	**8**	6
Krok 4.	0	1	2	**6**	**8**

Szarym tłem zaznaczono wyrazy ciągu, które są już posortowane. Pogrubieniem wyróżniono wyrazy zamienione miejscami w danym kroku. Czy rozpoznajesz przedstawioną metodę sortowania? Podaj nazwę tej metody oraz skonstruuj algorytm w dowolnej postaci — jako program, schemat blokowy lub listę kroków — porządkujący tą metodą ciąg liczb całkowitych zapisanych w liście.

5.1. Porządkowanie bąbelkowe

Sortowanie bąbelkowe (ang. *bubble sort*) polega na porównywaniu wszystkich kolejnych par wyrazów ciągu w celu znalezienia maksimum. W liście $T[0 \ldots n-1]$ są więc sprawdzane pary elementów $T[i]$ z $T[i+1]$, dla $i = 0, 1, \ldots, n-2$. Po każdym przejściu maksymalny wyraz przeglądanego ciągu jest przesuwany na koniec. Wyraz ten jest już posortowany, więc w kolejnej fazie przeglądamy krótszy ciąg. W pierwszym kroku sprawdzamy ciąg n-wyrazowy, wykonujemy więc $n-1$ porównań. W drugim kroku ciąg zawiera $n-1$ wyrazów, porównań będzie więc tylko $n-2$. Na końcu mamy jedynie dwa wyrazy i jedno porównanie.

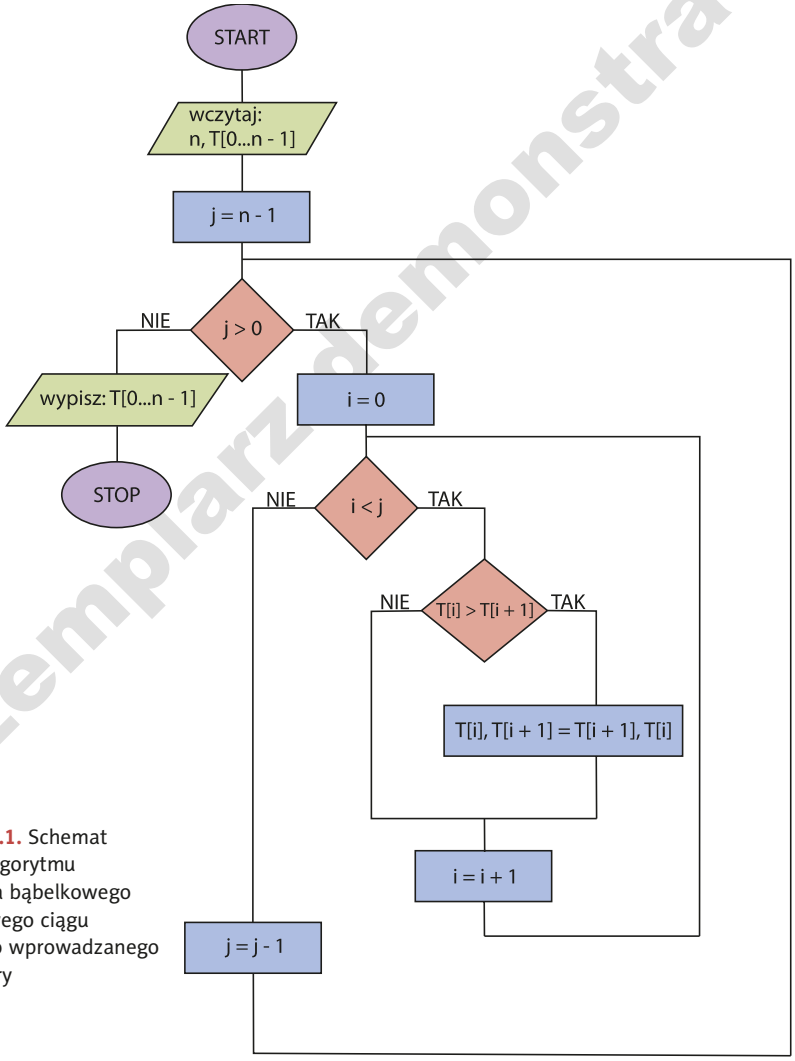
Przykład 5.1.

Prześledźmy kolejne kroki sortowania na przykładzie liczbowym. Uporządkujemy rosnąco ciąg liczbowy (7, 3, 2, 9, 1) z wykorzystaniem algorytmu bąbelkowego.

Każdy krok algorytmu to kolejne przejście przez ciąg, który jest coraz krótszy. Szarym tłem wyróżniono posortowane wyrazy ciągu liczbowego. Pogrubieniem zaznaczono pary tych wyrazów ciągu, które zostały zamienione miejscami.

Krok 1.	7	3	2	9	1
	3	**7**	2	9	1
	3	**2**	**7**	9	1
	3	2	7	**1**	**9**
Krok 2.	**2**	**3**	7	1	9
	2	3	**1**	**7**	9
Krok 3.	2	**1**	**3**	7	9
Krok 4.	**1**	**2**	3	7	9

Skonstruujmy **algorytm** w postaci listy kroków, schematu blokowego (rysunek 5.1) oraz programu w języku Python **realizujący sortowanie bąbelkowe**. Wyrazy ciągu liczbowego mają być uporządkowane niemalejąco.



Rysunek 5.1. Schemat blokowy algorytmu sortowania bąbelkowego n-wyrazowego ciągu liczbowego wprowadzanego z klawiatury

Specyfikacja:

Dane: liczba naturalna: $n > 0$ (liczba elementów listy T);

n-elementowa lista zawierająca liczby całkowite: $T[0\ldots n-1]$.

Wynik: posortowana niemalejąco n-elementowa lista zawierająca liczby całkowite: $T[0\ldots n-1]$.

Lista kroków:

Krok 1. Dla kolejnych wartości zmiennej j: $n-1, n-2, \ldots, 1$, wykonuj krok 2.

Krok 2. Dla kolejnych wartości zmiennej i: $0, 1, \ldots, j-1$, wykonuj krok 3.

Krok 3. Jeśli $T[i] > T[i+1]$, zamień miejscami te elementy listy.

Krok 4. Wypisz elementy listy $T[0\ldots n-1]$. Zakończ algorytm.

Program w języku Python:

```
def sortuj(T):
    n = len(T)
    for j in range(n - 1, 0, -1):
        for i in range(j):
            if T[i] > T[i + 1]:
                T[i], T[i + 1] = T[i + 1], T[i]
    return T


print(sortuj([9, 8, 7, 0, 5, 4, 3, 2]))
```

Wyniki:

[0, 2, 3, 4, 5, 7, 8, 9]

Operacją dominującą w algorytmie sortowania bąbelkowego jest porównanie. Wyznaczmy więc złożoność czasową tej metody dla n-wyrazowego ciągu. Pętla zewnętrzna przedstawionego algorytmu jest powtarzana $n-1$ razy. W pętli wewnętrznej, realizującej przeglądanie ciągu w celu znalezienia wyrazu maksymalnego i przesunięcia go na koniec ciągu, jest wykonywanych najpierw $n-1$ porównań, w kolejnym kroku $n-2$, a w ostatnim przebiegu przeprowadzone zostaje tylko jedno porównanie. Łącznie otrzymujemy:

$$(n-1)+(n-2)+\ldots+1=\frac{n(n-1)}{2}$$

operacji dominujących. Wadą tego algorytmu jest to, że liczba porównań nie zależy od wartości wyrazów ciągu liczbowego. Złożoność omówionej metody jest więc kwadratowa: n^2. Liczba wykonywanych zamian wyrazów sortowanego ciągu może być jednak różna:

$$\text{od } 0 \text{ do } \frac{n(n-1)}{2}.$$

Algorytm sortowania bąbelkowego można zmodyfikować, tak by zmniejszyć liczbę porównań. Jedną z takich możliwości jest zakończenie działania algorytmu w przypadku stwierdzenia, że ciąg jest już posortowany. Warunek ten będzie spełniony, jeśli w kolejnym kroku nie zostanie wykonana żadna zamiana.

Ćwiczenie 5.2.

Napisz program, który porządkuje nierosnąco ciąg liczb rzeczywistych wprowadzonych z klawiatury. Zastosuj algorytm sortowania bąbelkowego przedstawiony na schemacie blokowym (rysunek 5.1). Dokonaj w nim odpowiednich zmian, aby było realizowane sortowanie w porządku nierosnącym.

Zadanie 5.2.

W algorytmie sortowania bąbelkowego omówionym w tym podrozdziale wprowadź zmiany prowadzące do poprawienia jego złożoności czasowej. Algorytm powinien zakończyć działanie w przypadku, gdy ciąg jest już posortowany. Przedstaw algorytm w wybranej przez siebie postaci: jako listę kroków, schemat blokowy lub program.

5.2. Porządkowanie przez wstawianie

Sortowanie przez wstawianie (ang. *insertion sort*) przypomina porządkowanie kart. Po rozdaniu talii każdy z graczy układa swoje karty, biorąc po jednej i wstawiając od razu we właściwe miejsce. Właśnie na tym polega metoda sortowania przez wstawianie. Każdy wyraz ciągu jest kolejno pobierany i wstawiany we właściwym miejscu. W metodzie tej porównanie jest wykonywane w momencie wstawiania wyrazu do ciągu już uporządkowanego w celu znalezienia dla tego wyrazu właściwego miejsca.

Przykład 5.2.

Przeanalizujmy przebieg sortowania przez wstawianie na przykładzie liczbowym. Z użyciem sortowania uporządkujmy rosnąco ciąg liczbowy (7, 3, 0, 1, 5).

	Stąd pobieramy wyrazy ciągu					**Tutaj wstawiamy wyrazy, jednocześnie sortując ciąg**				
	7	3	0	1	5					
Krok 1.		3	0	1	5	**7**				
Krok 2.			0	1	5	**3**	7			
Krok 3.				1	5	**0**	3	7		
Krok 4.					5	0	**1**	3	7	
Krok 5.						0	1	3	**5**	7

Szarym tłem wyróżniono ciąg posortowany, a pogrubieniem zaznaczono wstawiane elementy.

Przykład 5.3.

Przyjrzyjmy się kolejnym krokom algorytmu sortowania przez wstawianie realizowanym w liście jednowymiarowej dla analizowanego wcześniej ciągu (7, 3, 0, 1, 5).

	7	3	0	1	5
Krok 1.	**3**	7	0	1	5
Krok 2.	**0**	3	7	1	5
Krok 3.	0	**1**	3	7	5
Krok 4.	0	1	3	**5**	7

Pogrubieniem zaznaczono wyrazy ciągu, które zostały w danym kroku wstawione do posortowanego już podciągu. Szarym tłem wyróżniono wyrazy ciągu, które zostały posortowane.

Załóżmy, że mamy daną listę $T[0\ldots n-1]$, w której zapisany jest ciąg do posortowania. Z powyższej analizy wynika, że sortowanie jest wykonywane w $n-1$ krokach. Na każdym z tych etapów zwiększa się fragment ciągu już posortowanego. Kolejne wyrazy o numerach z przedziału $[1, n-1]$ są wybierane i przestawiane we właściwe miejsce w tym fragmencie ciągu, który jest już posortowany. Zaczynamy od elementu $T[1]$, który zapisujemy do zmiennej pomocniczej. Następnie, jeśli $T[1]$ jest mniejszy od elementu $T[0]$, wyraz ciągu $T[0]$ przesuwamy w prawo na miejsce o indeksie 1, natomiast wstawiany element $T[1]$, zapisany w zmiennej pomocniczej, umieszczamy w miejscu $T[0]$. W kolejnym kroku zapamiętujemy element $T[2]$ i wstawiamy go do posortowanego fragmentu ciągu, czyli $T[0\ldots1]$. Jeśli wstawiany element $T[2]$ jest mniejszy od kolejnych wartości posortowanego ciągu $T[1]$ lub $T[0]$, wyraz o większej wartości jest przesuwany w prawo. Natomiast element $T[2]$ jest wstawiany w wolne miejsce, czyli pomiędzy element mniejszy i większy od niego lub na początek ciągu. Jeżeli element $T[2]$ nie jest mniejszy od $T[1]$, to pozostawiamy go na dotychczasowym miejscu, które staje się ostatnią pozycją uporządkowanej części ciągu. Kolejne kroki algorytmu są powtarzane tak długo, aż dojdziemy do elementu ostatniego $T[n-1]$, który będziemy wstawiać do ciągu $T[0\ldots n-2]$.

Skonstruujmy **algorytm** w postaci listy kroków i schematu blokowego (rysunek 5.2) **realizujący sortowanie przez wstawianie**.

Specyfikacja:

Dane: liczba naturalna: $n > 0$ (liczba elementów listy T);

n-elementowa lista zawierająca liczby całkowite: $T[0\ldots n-1]$.

Wynik: posortowana niemalejąco n-elementowa lista zawierająca liczby całkowite: $T[0\ldots n-1]$.

Lista kroków:

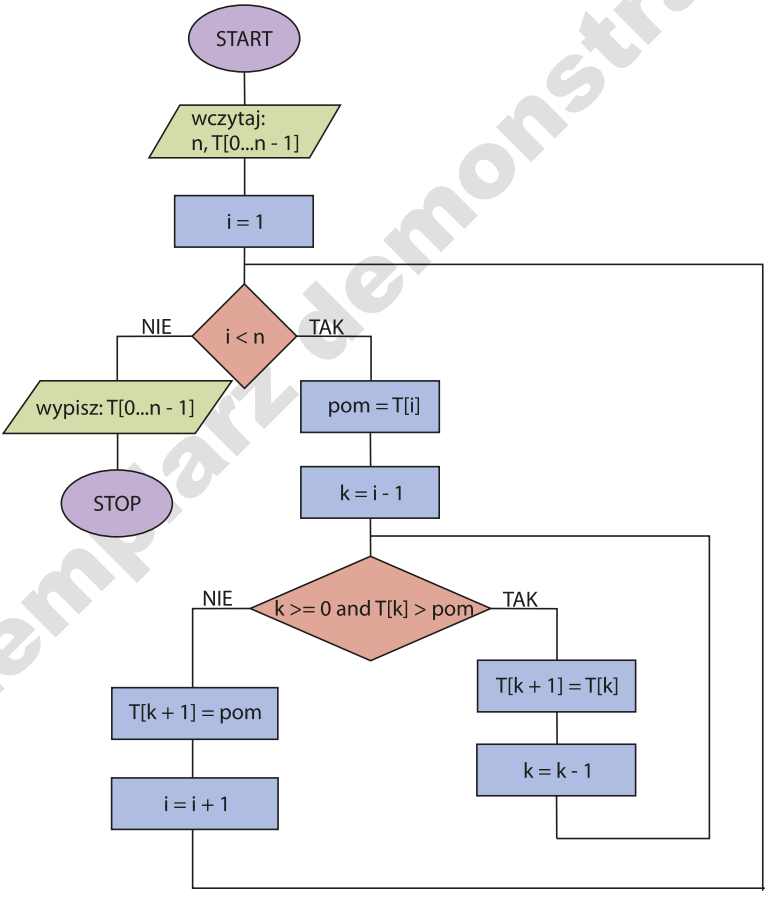
Krok 1. Dla kolejnych wartości zmiennej i: 1, 2, …, n – 1, wykonuj kroki 2. – 3., a następnie przejdź do kroku 4.

Krok 2. Przypisz $pom = T[i]$ (pobranie elementu do wstawienia).

Krok 3. Porównuj element pom z kolejnymi wyrazami uporządkowanego podciągu $T[k]$, dla $k = i - 1, i - 2, \ldots, 0$, przesuń sprawdzone elementy, zwiększając im indeks o 1 w celu zrobienia miejsca dla pom, i umieść element pom przed pierwszym elementem $T[k]$, który będzie spełniał warunek $T[k] \leq pom$. Przyimek „przed" należy tu rozumieć jako „na pozycji o numerze o jeden większym".

(Wstawienie elementu $pom = T[i]$ do posortowanego podciągu $T[0\ldots i-1]$ w taki sposób, aby po dodaniu tego elementu podciąg nadal był uporządkowany).

Krok 4. Wypisz elementy listy $T[0\ldots n-1]$. Zakończ algorytm.



Rysunek 5.2. Schemat blokowy algorytmu sortowania przez wstawianie n-wyrazowego ciągu liczbowego wprowadzanego z klawiatury

Wykonajmy analizę złożoności czasowej algorytmu. Operacją dominującą jest porównywanie. W tym przypadku dane wejściowe mają wpływ na złożoność algorytmu. Wyznaczmy złożoność pesymistyczną, czyli największą możliwą liczbę porównań, jaka w tym algorytmie może zostać wykonana. Taka sytuacja ma miejsce, gdy sortowany jest ciąg uporządkowany odwrotnie, wymagający największej liczby zamian wyrazów. Wówczas liczba porównań wynosi:

$$1+2+\ldots+n-1=\frac{n(n-1)}{2}.$$

Najmniejsza liczba porównań zostanie wykonana dla ciągu uporządkowanego, dokładnie n – 1 razy. Ponadto w tej sytuacji nie zostanie zrealizowana żadna zamiana miejscami wyrazów ciągu. Z przeprowadzonej analizy wynika, że złożoność tego algorytmu jest kwadratowa: n^2. Jednak w przypadku sortowania ciągu prawie uporządkowanego liczba porównań maleje, przez co uzyskujemy, w najlepszym przypadku, dla ciągu posortowanego złożoność liniową n.

Ćwiczenie 5.3.

Napisz program realizujący sortowanie niemalejące ciągu liczbowego przez wstawianie zgodny ze schematem blokowym przedstawionym na rysunku 5.2.

Zadanie 5.3.

Skonstruuj algorytm realizujący sortowanie nierosnące ciągu liczbowego przez wstawianie. Zapisz algorytm w wybranej przez siebie postaci: jako listę kroków, schemat blokowy lub program.

Temat 6. Sekwencyjne typy danych — napisy

Definicja

Napis (łańcuch znaków) to typ sekwencyjny niezmienny, a więc **nie jest możliwe przypisywanie wartości pojedynczym elementom** napisu.

Wartości napisów podajemy **w apostrofach** lub **cudzysłowach**. Różnica między tymi sposobami zapisu polega na tym, że dzięki zastosowaniu cudzysłowu można bez przeszkód używać apostrofów. Poniżej przykład:

```
print("Napisy to 'sekwencje' niezmienne.")
```

Po wykonaniu polecenia wyświetli się napis:

Napisy to 'sekwencje' niezmienne.

W innych przypadkach konieczne jest użycie znaku `\` przed apostrofem lub cudzysłowem, który ma zostać wyświetlony. Przykłady pokazują różne sytuacje, w których trzeba zastosować `\`:

```
print('Napisy to \'sekwencje\' niezmienne.')
print("Napisy to \"sekwencje\" niezmienne.")
```

Po wykonaniu tych poleceń zostaną wypisane napisy:

Napisy to 'sekwencje' niezmienne.

Napisy to "sekwencje" niezmienne.

Przykład 6.1.

Przyjrzyjmy się przykładowym deklaracjom napisu:

```
S = "NAPIS"
S = 'NAPIS'
```

Te deklaracje napisów są równoważne. Możemy stosować apostrofy lub cudzysłowy. Uzyskaliśmy napis `S` z przypisanym tekstem *NAPIS*. Podobnie jak w przypadku listy mamy tutaj dostęp zarówno do całego napisu `S`, jak i do poszczególnych znaków tego napisu: `S[0]`, `S[1]`, `S[2]`, `S[3]`, `S[4]`. Na przykład `S[0]` to odwołanie do pierwszego elementu napisu, czyli do znaku *N*, a `S[2]` to litera *P*.

W języku Python możemy stosować również indeksy ujemne, które umożliwiają odwołanie się do elementów, rozpoczynając od ostatniego. Na przykład `S[-1]` to znak *S*, a `T[-4]` to *A*.

Przykład 6.2.

Przeanalizuj odwołania do elementów przykładowego napisu podanego poniżej.

```
S = "ŁAŃCUCH"
```

Otrzymujemy 7-znakowy napis:

`S[0]` = *Ł*; `S[1]` = *A*; `S[2]` = *Ń*; `S[3]` = *C*; `S[4]` = *U*; `S[5]` = *C*; `S[6]` = *H*.

Dostęp do elementów napisu można również uzyskać w następujący sposób:

`S[-1]` = *H*; `S[-2]` = *C*; `S[-3]` = *U*; `S[-4]` = *C*; `S[-5]` = *Ń*; `S[-6]` = *A*; `S[-7]` = *Ł*.

Możesz wypisać na ekranie zarówno cały napis `print(S)`, jak i pojedyncze elementy tego napisu, na przykład `print(S[2])`.

Ćwiczenie 6.1.

Napisz program, w którym wprowadzisz z klawiatury nazwę aktualnego miesiąca, a następnie wypiszesz kolejno nazwę podanego miesiąca oraz pierwszą i trzecią literę tej nazwy.

Przykład 6.3.

W języku Python mamy dostęp do metody `len()`, która zwraca wartość oznaczającą liczbę elementów napisu. Na przykład dla napisu `S = "PROGRAMOWANIE"` funkcja `len(T)` zwraca wartość 13, co jest równe liczbie znaków w podanym napisie. Przeanalizuj przykład zastosowania tej metody do wypisywania wybranych znaków napisu, które mają numer parzysty.

```
S = "PROGRAMOWANIE"
print(S)
for i in range(len(S)):
    if i % 2 == 0:
        print(S[i], end='')
```

Po uruchomieniu powyższego programu pojawiają się następujące wyniki:

PROGRAMOWANIE

PORMWNE

Ćwiczenie 6.2.

Napisz program, który wypisze na ekranie napis wczytany z klawiatury w odwrotnej kolejności.

Przykład 6.4.

Przeanalizuj kod programu wykonujący następujące operacje na napisach:

a) wypisywanie znaków różnych od *m* i *k*,

b) obliczenie liczby znaków *a*,

c) obliczenie liczby znaków różnych od *f*, które mają numer parzysty,

d) zamiana znaku *o* na *x*,

e) zamiana znaków różnych od *m* i *a* na *W*.

Porównaj zaproponowane rozwiązania dla podpunktów a i b.

W podpunktach d i e zastosowano konwersję napisu na listę w celu wykonania operacji przypisania wartości pojedynczym elementom napisu (tabela 6.3).

Rozwiązanie 1.

```
def zad_a(s):
    n = len(s)
    print("zad_a = ", end='')
    for i in range(n):
        if s[i] != 'm' and s[i] != 'k':
            print(s[i], end='')
```

```
def zad_b(s):
    n = len(s)
    j = 0
    for i in range(n):
        if s[i] == 'a':
            j += 1
    return j

def zad_c(s):
    n = len(s)
    k = 0
    for i in range(0, n, 2):
        if s[i] != 'f':
            k += 1
    return k

def zad_d(s):
    n = len(s)
    s = list(s)
    for i in range(n):
        if s[i] == 'o':
            s[i] = 'x'
    s = ''.join(s)
    return s

def zad_e(s):
    n = len(s)
    s = list(s)
    for i in range(n):
        if s[i] != 'm' and s[i] != 'a':
            s[i] = 'W'
    s=''.join(s)
    return s

zad_a("informatyka")
print()
print("zad_b =", zad_b("informatyka"))
print("zad_c =", zad_c("informatyka"))
print("zad_d =", zad_d("informatyka"))
print("zad_e =", zad_e("informatyka"))
```

Wyniki:

zad_a = inforatya

zad_b = 2

zad_c = 5

zad_d = infxrmatyka

zad_e = WWWWWmaWWWa

Rozwiązanie 2.

```
def zad_a(s):
    w = ""
    for k in s:
        if k != 'm' and k != 'k':
            w += k
    return w

def zad_b(s):
    n = len(s)
    j = 0
    for k in s:
        if k == 'a':
            j += 1
    return j

print("zad_a =", zad_a("informatyka"))
print("zad_b =", zad_b("informatyka"))
```

Wyniki:

zad_a = inforatya

zad_b = 2

Zadanie 6.1.

Napisz program wykonujący następujące polecenia na napisach:

a) obliczenie liczby znaków *d*,

b) obliczenie liczby znaków różnych od *a*,

c) zamiana znaków różnych od *b* i *B* na *a*,

d) wypisanie tylko tych znaków, które są różne od *s*,

e) obliczenie liczby znaków różnych od *d*, które mają numer podzielny przez 3,

f) zamiana znaków *m* na *R*.

Przykład 6.5.

W tabeli 6.1 podano dodatkowe informacje na temat napisów i ich indeksowania. Na podstawie przykładów pokazanych w tej tabeli przeanalizuj sposób indeksowania napisów i porównaj go z zasadami indeksowania list. Czy zauważasz podobieństwa? Co oznaczają kolejne wartości oddzielone średnikami w indeksie napisu?

Tabela 6.1. Indeksowanie napisów

Polecenie	Wyniki
`tekst = 'ABCDEFGH'`	
`print(tekst)`	*ABCDEFGH*
`print(tekst[2:])`	*CDEFGH*
`print(tekst[1::2])`	*BDFH*
`print(tekst[::-1])`	*HGFEDCBA*
`print(tekst[::-2])`	*HFDB*
`print(tekst[::2])`	*ACEG*
`print(tekst[:3:-2])`	*HF*
`print(tekst[:4:2])`	*AC*
`print(tekst[0:4:2])`	*AC*
`print(tekst[2:8:2])`	*CEG*

Ćwiczenie 6.3.

Na podstawie przykładu 6.5 napisz program, w którym przetestujesz różne możliwości indeksowania napisu wprowadzonego z klawiatury.

W tabeli 6.2 podano wybrane dodatkowe metody stosowane na napisach, które możesz wykorzystywać w programach.

Tabela 6.2. Metody stosowane na napisach — nie modyfikują napisów, tylko zwracają nowy napis

Polecenie	Opis polecenia i wyniki
`tekst = "NAPIS to jest NAPIS"`	
`tekst.count('t')`	zwraca liczbę znaków *t* w napisie, wynik: *2*
`tekst.find('NAPIS')`	odnajduje pozycję pierwszego wystąpienia ciągu znaków *NAPIS* w napisie, a jeśli nie znajdzie, zwraca –1, wynik: *0*
`tekst.rfind('NAPIS')`	odnajduje pozycję ostatniego wystąpienia ciągu znaków *NAPIS* w napisie, a jeśli nie znajdzie, zwraca –1, wynik: *14*
`tekst.isdigit()`	sprawdza, czy w napisie są tylko cyfry, wynik: *False*
`tekst.lower()`	zamienia litery na małe, wynik: *napis to jest napis*

Polecenie	Opis polecenia i wyniki
`tekst.upper()`	zamienia litery na wielkie, wynik: *NAPIS TO JEST NAPIS*
`tekst.swapcase()`	odwraca wielkości liter w napisie, wynik: *napis TO JEST napis*
`tekst.capitalize()`	zamienia pierwszą literę na wielką, a pozostałe na małe, wynik: *Napis to jest napis*
`tekst.strip()`	usuwa białe znaki (tabulatory, spacje, znaki nowego wiersza) z lewej i prawej strony napisu
`tekst.lstrip()`	usuwa białe znaki z lewej strony napisu
`tekst.rstrip()`	usuwa białe znaki z prawej strony napisu
`tekst.replace('NAPIS', 'tekst')`	zamienia wszystkie wystąpienia łańcucha *NAPIS* na *tekst*, wynik: *tekst to jest tekst*
`tekst.split()`	konwertuje napis na listę wyrazów, wynik: *['NAPIS', 'to', 'jest', 'NAPIS']*
`chr(65)`	znak o kodzie ASCII równym 65, wynik: *A*
`ord(tekst[1])`	kod ASCII znaku `tekst[1]`, wynik: *65*

Ćwiczenie 6.4.

Napisz program, który wykonuje na tekście wczytanym z klawiatury działania wymienione w podpunktach a – f. Zastosuj metody na napisach podane w tabeli 6.2.

a) obliczenie liczby znaków *m* w napisie,

b) wyznaczenie pozycji pierwszego wystąpienia w napisie ciągu znaków *ka*,

c) wyznaczenie pozycji ostatniego wystąpienia w napisie ciągu znaków wpisanego z klawiatury,

d) sprawdzenie, czy w napisie są tylko cyfry,

e) odwrócenie wielkości liter w napisie,

f) zamiana wszystkich wystąpień tekstu *AB* na *BA*.

Zadanie 6.2.

Napisz program wypisujący wprowadzony z klawiatury tekst jako rozstrzelony w następujący sposób:

a) 1 znak + 1 spacja,

b) 1 znak + 1 spacja (tekst czytany od końca),

c) 2 znaki + 1 spacja,

d) 3 znaki + 2 spacje (tekst czytany od końca).

Zadanie 6.3.

Napisz program wyznaczający w tekście wpisanym z klawiatury:

a) liczbę znaków *a*,

b) liczbę znaków różnych od *B*, które mają numer parzysty.

Zadanie 6.4.

Napisz program wypisujący określone znaki tekstu wprowadzonego z klawiatury:

a) znaki różne od *h*,

b) znaki równe *a* lub *A*, lub *u*, lub *U*.

Zadanie 6.5.

Napisz program wczytujący z klawiatury tekst ze spacjami, obliczający liczbę słów w tekście oraz wypisujący wczytany napis następująco:

a) wszystkie wyrazy wypisane w kolumnie,

b) wszystkie wyrazy, których pierwszy znak jest równy ostatniemu, wypisane w kolumnie.

Zadanie 6.6.

Napisz program wczytujący dany tekst z klawiatury i wypisujący z niego wszystkie wyrazy o nieparzystej liczbie liter zaczynające się na literę *P*.

Zadanie 6.7.

Palindrom to ciąg znaków, który czytany od początku i od końca jest taki sam. Podaj specyfikację zadania i napisz program sprawdzający, czy podany tekst jest palindromem.

Zadanie 6.8.

Napisz programy wykonujące sortowanie alfabetyczne napisu składającego się z samych cyfr, niezawierającego spacji. Zastosuj jedną z metod sortowania opisanych w temacie 5.

Zadanie 6.9.

Anagramami nazywamy ciągi znaków utworzone w wyniku przestawienia znaków innego łańcucha znaków. Przykładami anagramów są pary słów: algorytm — logarytm, tyran — narty, alergia — galeria, arbuz — burza, absurd — brudas. Podaj specyfikację zadania i napisz program sprawdzający, czy wpisane z klawiatury słowa są anagramami. Zastosuj sortowanie tekstu.

Konwersje między typami sekwencyjnymi

Omówione zostały dwa typy sekwencyjne: listy i napisy. Jak już wiesz, typy sekwencyjne mogą być zmienne bądź niezmienne, dlatego może zaistnieć potrzeba konwersji między tymi typami. W tabeli 6.3 zestawiono **podstawowe metody konwersji** między typami sekwencyjnymi, które zostały omówione w tym rozdziale.

Tabela 6.3. Konwersje między typami sekwencyjnymi

Polecenie	Opis polecenia
`liczba = int(napis)`	konwersja napisu na liczbę
`napis = str(liczba)`	konwersja liczby na napis
`lista = list(napis)`	konwersja napisu na listę
`''.join(['x', 'y', 'z'])` `' '.join(['x', 'y', 'z'])`	konwersja listy na napis, wyniki: *‚xyz'* *‚x y z'*

`s = list(s)`
`.`
`.`
`.`
`.`
`s = ''.join(s)`
	konwersja tekstu na listę, a potem listy na tekst,
	stosujemy, gdy w programie chcemy wykonywać przypisywanie wartości elementom listy, ale wczytaliśmy tekst

Temat 7. Proste algorytmy na tekstach

W tym temacie poznasz proste algorytmy, za pomocą których będziesz wykonywać różne operacje na napisach. Przygotujesz się w ten sposób do lektury następnych tematów. Te umiejętności będą niezbędne do zrozumienia poruszanych w nich zagadnień.

7.1. Ukrywanie tekstu

Przykład 7.1.

Przeanalizuj algorytm ukrywający tekst przez wstawienie do tego tekstu, co drugą literę, znaku wybranego losowo z alfabetu składającego się z samych samogłosek alfabetu łacińskiego (bez znaków diakrytycznych), czyli AEIOUY.

Program w języku Python:

```
from random import *
def ukryj(tekst, alfabet):
    wynik = ''
    for t in tekst:
        wynik += t + choice(alfabet)
    return wynik

print(ukryj('UKRYWANIE', 'AEIOUY'))
```

Przykładowy wynik:

UOKARIYIWYAYNUIOEA

Dla tekstu UKRYWANIE uzyskaliśmy UOKARIYIWYAYNUIOEA. Wynik otrzymujemy przez ustawienie na przemian kolejnych liter tekstu i losowo wybranych samogłosek. Zwróć uwagę na metodę `choice()`, która losowo wybiera jeden znak z tekstu podanego jako parametr. W tym przypadku metoda `choice(alfabet)` zwraca jedną losowo wybraną samogłoskę. Aby skorzystać z tego polecenia, trzeba zaimportować bibliotekę `random`.

Ćwiczenie 7.1.

Napisz program, który ukryje tekst wczytany z klawiatury przez wstawienie pomiędzy litery tego tekstu losowo wybranych znaków alfabetu ABCDEF i dodatkowo oddzielenie wszystkich liter spacjami. Na przykład dla napisu KOMPUTER wynikiem może być K B O F M A P A U C T E E F R B.

Zadanie 7.1.

Napisz programy ukrywające tekst następująco:

a) dwie litery ukrywanego tekstu + znak podkreślenia + dwie losowo wybrane litery alfabetu składającego się ze spółgłosek alfabetu łacińskiego (bez znaków diakrytycznych) + znak podkreślenia, na przykład dla tekstu LICZBA wynikiem może być LI_MN_CZ_ZK_BA_BW_, a dla napisu TEKST wynikiem może być TE_HJ_KS_MN_T,

b) jedna litera tekstu (czytanego od końca) + jedna losowo wybrana litera alfabetu składającego się z samogłosek alfabetu łacińskiego (bez znaków diakrytycznych) + kropka, na przykład dla tekstu SYMBOL wynikiem może być LA.OO.BO.MI.YU.SY,

c) dwie litery tekstu w odwrotnej kolejności + znak gwiazdki + jedna losowo wybrana litera alfabetu łacińskiego (bez znaków diakrytycznych) + znak gwiazdki, na przykład dla tekstu LITERA wynikiem może być IL*H*ET*K*AR*U*, a dla napisu TEKST wynikiem może być ET*H*SK*U*T.

7.2. Porównywanie tekstów

Przykład 7.2.

Przeanalizuj algorytm porównujący dwa teksty zawierające tyle samo znaków i obliczający, ile jest niezgodnych liter w tych tekstach. Na przykład dla słów **IN**FORMA**TYK** i **RE**FORMA**CJA** liczba niezgodnych liter wynosi 5. Oznaczono je w tych słowach pogrubieniem.

Program w języku Python:

```
def porownaj(tekst_1, tekst_2):
    n = len(tekst_1)
    wynik = 0
    for i in range(n):
        if tekst_1[i] != tekst_2[i]:
            wynik += 1
    return wynik

print(porownaj('INFORMATYK', 'REFORMACJA'))
```

Wynik:

5

Porównywane są kolejne znaki obydwu słów i w przypadku, gdy znaki są różne, wartość zmiennej *wynik* jest zwiększana o 1. Wynikiem algorytmu jest liczba niezgodnych liter.

Zadanie 7.2.

Napisz program, który dla dwóch napisów o tej samej liczbie znaków wprowadzonych z klawiatury obliczy, ile jest zgodnych liter, wypisze je na ekranie oraz wyznaczy, jaki pro-

cent wszystkich liter stanowią litery niezgodne. Na przykład dla słów IN**FORMA**TYK i RE**FORMA**CJA liczba zgodnych liter wynosi 5 (oznaczono je w tych słowach pogrubieniem), natomiast litery niezgodne stanowią w tym przypadku 50%.

Zadanie 7.3.

Napisz program porównujący dwa teksty, które nie muszą zawierać tej samej liczby znaków, i obliczający, ile jest zgodnych liter w tych tekstach. Na przykład dla słów IN**FORMA**TYKA i RE**FORMA**CJA liczba zgodnych liter wynosi 5 (oznaczono je w tych słowach pogrubieniem). Dodatkowo program powinien wypisywać wszystkie litery niezgodne w dowolnej kolejności tak, aby się nie powtarzały. W tym przykładzie będą to litery: INTYKARECJ.

Temat 8. Szyfrowanie tekstu metodą przestawieniową

Omawianie tematyki związanej z szyfrowaniem danych rozpoczniemy od wyjaśnienia podstawowych zagadnień. Zasadniczym pojęciem jest **kryptologia**. Obejmuje ona dwa obszary:

- **kryptografię**, czyli szyfrowanie danych;
- **kryptoanalizę**, czyli metody łamania szyfrów.

W opisie metod szyfrowania, którymi będziemy się zajmować, zetkniesz się również z określeniami:

- **tekst jawny** — informacja, która ma zostać utajniona, czyli zaszyfrowana;
- **kryptogram**, **szyfrogram** — zaszyfrowana informacja.

Szyfrowanie treści przekazywanej informacji może być wykonywane na dwa sposoby, stąd dwie metody szyfrowania danych:

- **szyfrowanie przestawieniowe (permutacyjne)** — przez przestawienie znaków utajnianej wiadomości;
- **szyfrowanie podstawieniowe** — przez zastąpienie znaków utajnianej informacji innymi znakami.

Podczas szyfrowania wykorzystywana jest informacja w postaci **klucza**, który stanowi dodatkowe zabezpieczenie kryptogramu. W przypadku omawianych tutaj metod ten sam klucz jest używany do szyfrowania i deszyfrowania — takie algorytmy są nazywane **symetrycznymi**.

Szyfr płotkowy

Algorytm płotkowy jest przykładem szyfrowania przestawieniowego. W tej metodzie znaki tekstu jawnego zapisuje się w taki sposób, aby tworzyły kształt przypominający **płot** zbudowany ze sztachet. Szyfrogram uzyskujemy przez odczytanie kolejnych wierszy tak utworzonej konstrukcji. **Klucz** tego algorytmu to wysokość płotu, czyli liczba wierszy. Trzeba pamiętać, że dla uproszczenia przy szyfrowaniu pomija się spacje.

Przykład 8.1.

Zaszyfrujmy tą metodą słowo KRYPTOANALIZA. Zastosujmy płot o wysokości trzech wierszy, co będzie stanowić klucz tego algorytmu. Po wprowadzeniu tekstu jawnego do tablicy otrzymujemy następujący efekt:

```
K       T       A       A
  R   P   O   N   L   Z
    Y       A       I
```

Odczytanie szyfrogramu wierszami od góry daje wynik: KTAARPONLZYAI.

Skonstruujmy **algorytm realizujący szyfrowanie metodą płotkową**. Zastosujmy **płot o wysokości 3**.

Specyfikacja:

Dane: łańcuch znaków: *tekst* (zawiera tekst jawny).

Wynik: łańcuch znaków: *wynik* (zawiera kryptogram).

Program w języku Python:

```
def szyfruj(tekst):
    wynik = ''
    n = len(tekst)
    for i in range(0, n, 4):
        wynik += tekst[i]
    for i in range(1, n, 2):
        wynik += tekst[i]
    for i in range(2, n, 4):
        wynik += tekst[i]
    return wynik

print(szyfruj('KRYPTOANALIZA'))
```

Wynik:

KTAARPONLZYAI

Zauważ, że każde kolejne powtórzenie pętli `for` to wczytywanie do kryptogramu jednego wiersza płotu. W pierwszym wierszu pobieramy znaki, których indeks jest podzielny przez 4, w drugim — znaki, które mają numery nieparzyste, a w trzecim — znaki o indeksach 2, 6, 10…

Ćwiczenie 8.1.

Zaszyfruj metodą płotkową tekst: WLASNE MALE OGNISKO CENNIEJSZE OD STOSU ZLOTA. Zastosuj płot o wysokości 2. Przy szyfrowaniu pomiń spacje.

Ćwiczenie 8.2.

Odszyfruj kryptogram: BZOIJPYECYNSNEAEDOZNUZNCDOCK zaszyfrowany metodą płotkową. Wykorzystano tutaj płot o wysokości 3. Przy szyfrowaniu pominięto spacje.

Zadanie 8.1.

Podaj specyfikację zadania oraz napisz program deszyfrujący kryptogram wczytywany z klawiatury uzyskany metodą płotkową dla płotu o wysokości 3.

Zadanie 8.2.

Podaj specyfikację zadania oraz napisz program szyfrujący i deszyfrujący wczytywaną z klawiatury wiadomość algorytmem płotkowym dla płotu o wysokości 2.

Zadanie 8.3.

W metodzie szyfrowania płotkowego płot może przybierać różne kształty. Zaprojektuj własny płot o wysokości co najmniej 3 i zaszyfruj tekst jawny KRYPTOGRAFIA. Podaj specyfikację zadania oraz napisz program szyfrujący i deszyfrujący wczytywaną z klawiatury wiadomość metodą płotkową zaprojektowaną przez Ciebie.

Temat 9. Szyfrowanie tekstu metodą podstawieniową — szyfr Cezara

Szyfrowanie przez podstawianie polega na zastępowaniu znaków utajnianej informacji innymi znakami. Klasycznym przykładem algorytmu podstawieniowego jest **szyfr Cezara**. Metoda ta polega na zamianie pojedynczych znaków tekstu jawnego na inne znaki według specjalnego klucza. Załóżmy, że alfabet tekstu jawnego składa się z x znaków i każdy znak ma przypisany numer. Szyfrowanie polega tutaj na zastąpieniu znaku tekstu jawnego o numerze n znakiem o numerze równym reszcie z dzielenia $(n + k)$ przez x. Wartość k jest tutaj liczbą naturalną mniejszą od x i pełni funkcję klucza. W pierwszej wersji tego algorytmu szyfrującego klucz k, określający przesunięcie w alfabecie, wynosił 3. Litery znajdujące się na końcu alfabetu, dzięki zastosowaniu operacji reszty z dzielenia, są w tej metodzie zastępowane znakami z początku alfabetu.

Przyjmijmy, że tekst jawny zawiera wyłącznie następujące litery:

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z.

Zestaw ten nazywamy **alfabetem jawnym**, który służy do zapisywania tekstu jawnego. Znaki alfabetu są uporządkowane oraz każdy ma przypisany numer.

Zestaw symboli, który otrzymujemy po zastosowaniu klucza, nazywamy **alfabetem szyfrowym**. Służy on do zapisywania kryptogramów. Na przykład dla klucza równego 2 uzyskamy następujący alfabet:

C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, A, B.

Szyfrowanie polega tutaj na tym, że w miejsce znaków alfabetu jawnego podstawiamy odpowiadające im symbole alfabetu szyfrowego. W tym przypadku szyfrowanie wygląda następująco: literę B zastępuje D, literę G — I, a Z — B.

Przykład 9.1.

Zaszyfrujmy algorytmem Cezara tekst jawny KRYPTOANALIZA dla $k = 3$. W tym przypadku każdej literze szyfrowanej wiadomości będzie odpowiadać litera z trzeciej pozycji w prawo alfabetu jawnego.

Alfabet jawny: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Alfabet szyfrowy: D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

Otrzymamy więc następujący szyfrogram: NUBSWRDQDOLCD.

Skonstruujmy **algorytm** w postaci schematu blokowego (rysunek 9.1) oraz programu w języku Python **realizujący szyfr Cezara**.

Specyfikacja:

Dane: łańcuch znaków: *alfabet* (zawiera alfabet jawny);
łańcuch znaków: *tekst* (zawiera tekst jawny);
liczba naturalna: *klucz* < liczba znaków w alfabecie jawnym.

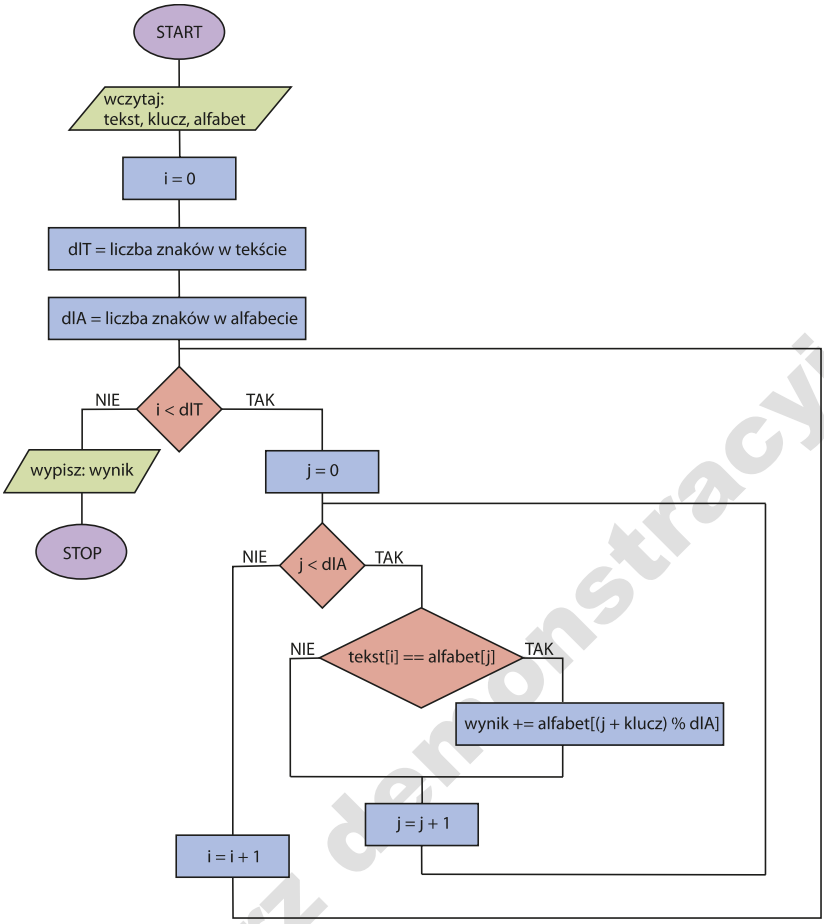
Wynik: łańcuch znaków: *wynik* (zawiera kryptogram).

Program w języku Python:

```
def szyfruj(tekst, alfabet, klucz):
    klucz = 3
    wynik = ''
    for t in tekst:
        if t in alfabet:
            wynik += alfabet[(alfabet.find(t) + klucz) % (len(alfabet))]
    return wynik
print(szyfruj('KRYPTOANALIZA', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ', 3))
```

Wynik:

NUBSWRDQDOLCD



Rysunek 9.1. Schemat blokowy algorytmu realizującego szyfr Cezara

Ćwiczenie 9.1.

Przeanalizuj podany powyżej kod programu oraz schemat blokowy (rysunek 9.1) realizujące szyfrowanie metodą Cezara. Odpowiedz na następujące pytania:

a) Dlaczego przy obliczaniu indeksu znaku szukanego w alfabecie stosujemy operację reszty z dzielenia?

b) Czym różnią się algorytmy zapisane jako program i schemat blokowy?

Napisz program realizujący szyfrowanie metodą Cezara na podstawie schematu blokowego.

Ćwiczenie 9.2.

Kryptogram SF SFZPJ SNLID SNJ OJXY EF UTEST powstał w wyniku szyfrowania metodą Cezara z wykorzystaniem klucza równego 5. Zastosowano alfabet jawny podany w przykładzie 9.1. Dla ułatwienia zachowano odstępy w zaszyfrowanym tekście. Podaj alfabet szyfrowy i odczytaj wiadomość, którą zaszyfrowano.

Ćwiczenie 9.3.

Metodą Cezara zaszyfruj tekst KLAMSTWO MA KROTKIE NOGI. Użyj alfabetu jawnego wykorzystanego w przykładzie 9.1 i klucza równego 9. Podaj alfabet szyfrowy i kryptogram utajnianej wiadomości.

Zadanie 9.1.

Podaj specyfikację zadania i napisz program deszyfrujący wiadomość zaszyfrowaną metodą Cezara. Kryptogram i klucz mają być wprowadzane z klawiatury.

Zadanie 9.2.

Podaj specyfikację zadania i napisz program szyfrujący oraz deszyfrujący tekst jawny metodą Cezara, w której zastosowano dwa klucze:

a) pierwszy klucz dla tych znaków tekstu jawnego, które mają numer podzielny przez 3,

b) drugi klucz dla tych znaków tekstu jawnego, które mają numer niepodzielny przez 3.

Tekst jawny i klucze mają być wprowadzane z klawiatury.

Zadania do rozdziału 1.

Zadanie 1.

Napisz program, który oblicza sumę cyfr liczby naturalnej wprowadzanej z klawiatury. Na przykład dla liczby 125 wynikiem będzie 1 + 2 + 5 = 8.

Zadanie 2.

Liczby lustrzane to para liczb, z których jedna powstaje przez zapisanie cyfr drugiej liczby w odwrotnej kolejności. Przykładami takich liczb są: 12 i 21, 19 i 91, 57 i 75, 208 i 802. Podaj specyfikację zadania i napisz program, który sprawdza, czy wczytane z klawiatury liczby są liczbami lustrzanymi.

Zadanie 3.

Liczba ciekawa cyfrowo to co najmniej dwucyfrowa liczba naturalna, w której każde dwie cyfry stojące bezpośrednio obok siebie są różne. Na przykład liczba 457562 jest ciekawa cyfrowo, a liczba 25583 nie jest ciekawa cyfrowo. Wykonaj następujące polecenia:

a) podaj specyfikację zadania i napisz program, który sprawdza, czy wprowadzona z klawiatury liczba jest ciekawa cyfrowo,

b) podaj specyfikację zadania i napisz program, który wypisze na ekranie wszystkie liczby ciekawe cyfrowo z przedziału $[a, b]$, gdzie a i b są liczbami naturalnymi co najmniej dwucyfrowymi wprowadzonymi z klawiatury.

Zadanie 4.

Liczbą doskonałą I rzędu nazywamy liczbę naturalną, która jest równa **sumie** wszystkich swoich dzielników mniejszych od niej samej. Przykładem takiej liczby jest liczba 28, ponieważ 28 = 1 + 2 + 4 + 7 + 14. **Liczbą doskonałą II rzędu** nazywamy liczbę naturalną, która jest równa **iloczynowi** wszystkich swoich dzielników mniejszych od niej samej. Taką liczbą jest liczba 6, ponieważ 6 = 1 · 2 · 3. Wykonaj następujące polecenia:

a) podaj specyfikację zadania i napisz program, który sprawdza, czy wprowadzona z klawiatury liczba jest liczbą doskonałą I rzędu,

b) podaj specyfikację zadania i napisz program, który sprawdza, czy wprowadzona z klawiatury liczba jest liczbą doskonałą II rzędu.

Zadanie 5.

Liczby *a* i *b* nazywamy **liczbami zaprzyjaźnionymi**, jeżeli suma wszystkich dzielników liczby *a* mniejszych od niej jest równa liczbie *b* i odwrotnie, suma wszystkich dzielników liczby *b* mniejszych od niej jest równa liczbie *a*. Przykładami takich liczb są 220 i 284, ponieważ:

- suma dzielników liczby 220 wynosi 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284, jest więc równa drugiej liczbie;
- suma dzielników liczby 284 wynosi 1 + 2 + 4 + 71 + 142 = 220 i jest równa pierwszej liczbie.

Podaj specyfikację zadania i napisz program, który sprawdza, czy dwie liczby naturalne wczytane z klawiatury są liczbami zaprzyjaźnionymi.

Zadanie 6.

Podany jest pewien ciąg znaków tworzących alfabet: 3, 5, 7, E, U, C, G, H, K, N. Napisz programy wczytujące z klawiatury dziesięć słów, a następnie wykonujące następujące polecenia:

a) Obliczenie, ile słów zawiera tylko znaki należące do podanego alfabetu.

b) Ukrycie tych słów, których pierwsza litera należy do alfabetu, przez wstawienie po każdej literze, która również należy do alfabetu, losowo wybranej z niego litery. Na przykład dla słowa KONTO wynikiem może być K5ONETO.

c) Zaszyfrowanie każdego wczytanego słowa metodą płotkową dla płotu o wysokości 4, w której szyfrogram jest zapisywany począwszy od najniższego wiersza utworzonej konstrukcji. Przykładowo dla słowa KRYPTOGRAM otrzymujemy następujący efekt:

K G
R O R
Y T A
P M

Odczytanie szyfrogramu wierszami od dołu daje wynik: PMYTARORKG.

Zadanie 7.

Zadanie polega na realizacji **projektu edukacyjnego pod tytułem „Przegląd technik kryptograficznych zabezpieczających urządzenia elektroniczne”**.

Celem projektu jest omówienie zagadnienia kryptografii, w tym podstawowych metod szyfrowania i ich zastosowania w zakresie zabezpieczania urządzeń elektronicznych, takich jak komputery, tablety, smartfony.

Pracę nad projektem należy rozpocząć od określenia, czym jest kryptografia. Następnie należy zapoznać się z różnymi metodami szyfrowania i ich rozwojem. Trzeba również poszukać informacji na temat prób łamania szyfrów.

Projekt powinien być realizowany w zespołach. Efekt pracy zespołu można przedstawić w postaci prezentacji multimedialnej lub strony internetowej.