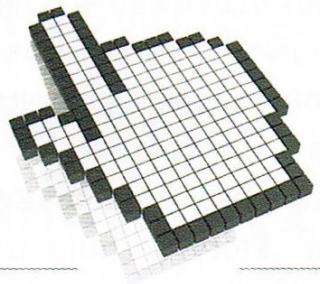
1 Od problemu do wyniku



Stosując podejście algorytmiczne w rozwiązywaniu problemów, zwracamy uwagę na wiele kolejno po sobie występujących **akcji** (działań, instrukcji, operacji), które w skończonym czasie wykonują pewien proces lub obliczenie. Każda z akcji wymaga istnienia **obiektów** (danych), do których się odnosi. Rezultatem przeprowadzenia akcji na obiekcie jest otrzymanie **wyniku**.

> **Algorytm** jest przepisem opisującym krok po kroku rozwiązanie problemu lub osiągnięcie jakiegoś celu.[1]

Algorytm wyrażony w języku określonego typu nazywa się **programem**. Różnica między algorytmem ogólnym a programem polega głównie na tym, że program musi być określony ściśle według reguł językowych, aż do najdrobniejszych szczegółów. W języku literackim pojawienie się błędu ortograficznego w zasadzie nie zmieni sensu zdania. W przypadku komputera będą to błędy uniemożliwiające dalsze obliczenia. Z tego powodu warto wcześniej opracować algorytm, w którym całą uwagę skoncentrujemy na podaniu przepisu rozwiązania postawionego zadania, bez potrzeby odwoływania się do konkretnego języka programowania. Algorytm możemy wyrazić językiem mówionym, w formie listy kroków, za pomocą schematów blokowych lub w każdy inny uzasadniony sposób.

Opracowywanie algorytmów i pisanie programów musi być poprzedzone precyzyjnym zdefiniowaniem **problemu**. Opis problemu (zadania) polega na podaniu jego **specyfikacji**, na którą składają się: **dane wejściowe** i **wyniki**, czyli **cel** do osiągnięcia. Zarówno dla danych, jak i dla wyników trzeba określić warunki, jakie muszą one spełniać.

Oto przykładowe sformułowanie najprostszego problemu:
Oblicz sumę dwóch liczb naturalnych a, b.
Wynik oznacz jako S.
W tym prostym zadaniu danymi wejściowymi są dwie liczby a, b, natomiast celem – obliczenie sumy $S(a, b)$. Warunek dla danych wejściowych jest taki, że a i b są liczbami naturalnymi.

Następnym krokiem jest przeprowadzenie analizy, która prowadzi do wyboru metody rozwiązania problemu, czyli algorytmu, który określa sposób otrzymania wyników na podstawie danych wejściowych. Kolejnym etapem jest opisanie algorytmu (metody) w konkretnym języku programowania.

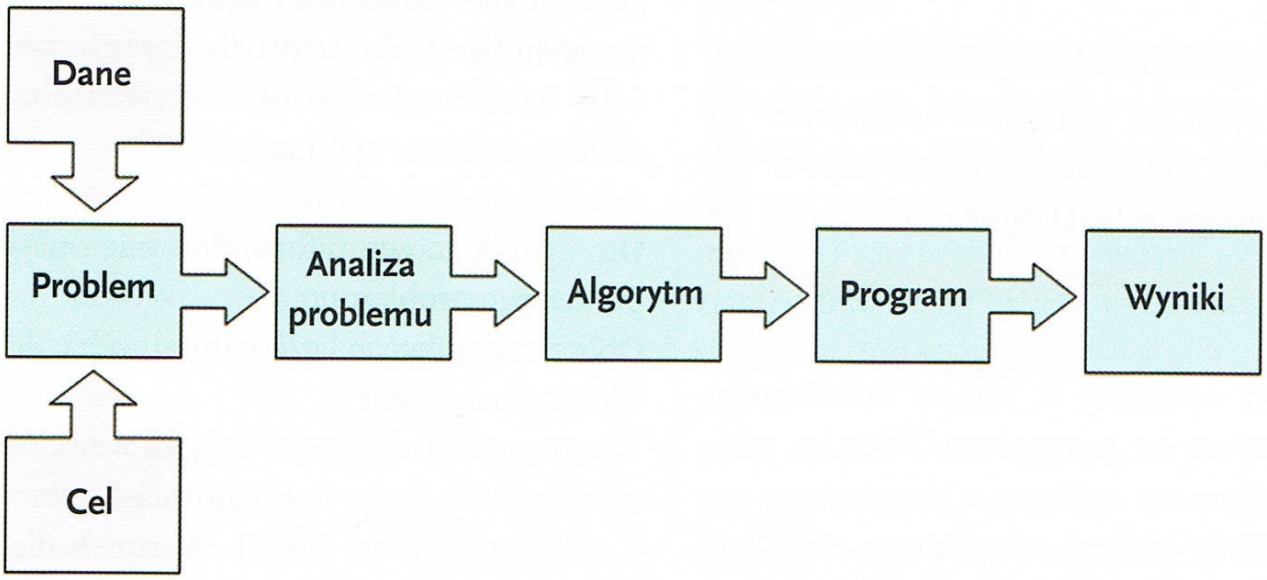
Dla algorytmu i programu wspólne są następujące trzy podstawowe własności:

- **Poprawność** – wykonanie algorytmu (programu) dla dowolnych danych wejściowych daje zawsze poprawne wyniki.
- **Skończoność** – algorytm wykonuje skończoną liczbę kroków.
- **Sprawność** – ten algorytm (program) jest lepszy, który cechuje mniejsza **złożoność czasowa** (liczy szybciej) oraz mniejsza **złożoność pamięciowa** (zajmuje mniej miejsca w pamięci).

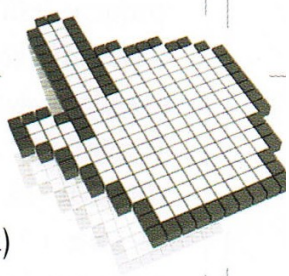
Każdy algorytm można podzielić na **moduły**, które stanowią opis wyróżnionego, dobrze określonego działania (zadania) na równie dobrze określonych obiektach. Z tych szczegółowych zadań można następnie zbudować ogólny plan (algorytm) całego problemu. Wydzielenie w algorytmie modułów nazywa się **modularyzacją algorytmu**.

Tematem tego rozdziału będzie zastosowanie **umownego strukturalnego języka programowania** (aby wyrazić w nim algorytm) i języka **C++** (aby zapisać w nim program). Budowanie programów z procedur jest istotą programowania strukturalnego. Procedury można traktować jak klocki (bloki), które (po uprzednim przetestowaniu), odpowiednio poukładane, będą rozwiązaniem problemu.

Powyższe rozważania można podsumować prostym schematem[2] przedstawionym na rys. 1.



2 Projektowanie rozwiązania problemu za pomocą umownego strukturalnego języka programowania



W programowaniu strukturalnym korzysta się z prostych konstrukcji, z których można zbudować poprawny program. Zestaw podstawowych instrukcji umownego języka programowania (zwanego także pseudojęzykiem) umożliwi nam sformułowanie kilku przykładowych algorytmów.

Instrukcja przypisania – przedstawiana w następujący sposób:
`z := w`
gdzie: `z` – zmienna, `w` – wyrażenie.

Przykłady
`x := 5` – nadanie zmiennej *x* wartości 5.
`I := I + 1` – interpretacja tego zapisu jest następująca: *Zmiennej I przypisz nową wartość równą poprzedniej wartości I powiększonej o 1.* Przykładowo, jeśli *I* wynosiło 4, to nową wartością *I* będzie 5. Zwróć uwagę, że znakiem przypisania jest „:=", a nie „=".

Instrukcje wejścia / wyjścia – służą do wprowadzania danych i wyprowadzania wyników.
`Podaj (x)` – wprowadzenie do programu wartości, która zostanie przypisana zmiennej *x*.
`Pisz (x)` – wyprowadzenie z programu wyniku, który został przypisany zmiennej *x*.
Należy przyjąć ponadto, że jeśli argumentem instrukcji wyjścia jest ciąg znaków ujęty w apostrofy (`'...'`), to z programu zostaną wyprowadzone dokładnie te znaki, np.
`Pisz ('Wynik')` – wyprowadzenie napisu Wynik.
`Pisz ('x')` – wyprowadzenie napisu x, a nie wartości zmiennej *x*.

Instrukcja złożona – jeśli dowolny zbiór (ciąg) instrukcji zostanie poprzedzony słowem **POCZĄTEK**, a zakończony słowem **KONIEC**, to otrzymamy instrukcję złożoną traktowaną jak pojedyncza instrukcja. Słowa te pełnią funkcje podobne do nawiasów i służą do określenia granic danej instrukcji złożonej.

```
POCZĄTEK
 Instrukcja_1
 Instrukcja_2  => pojedyncza instrukcja
 Instrukcja_3
 itd.
KONIEC
```

Instrukcja warunkowa (decyzyjna) – służy do wyboru dalszej akcji. Wyróżnia się dwa przypadki:

- **instrukcję warunkową prostą:**

```
JEŚLI warunek TO akcja_1
```

- **instrukcję warunkową z alternatywą:**

```
JEŚLI warunek TO akcja_1 W PRZECIWNYM RAZIE akcja_2
```

Warunek jest wyrażeniem logicznym, któremu przypisuje się wartość logiczną – **prawda (Tak)** lub **fałsz (Nie)**. Interpretacja instrukcji warunkowej z alternatywą jest następująca: *Jeśli jest spełniony warunek logiczny (Tak), **to** wykonaj akcję 1, **w przeciwnym razie** (Nie) wykonaj akcję 2.* Dla instrukcji warunkowej prostej, jeśli warunek logiczny nie jest spełniony – wykonywana jest kolejna instrukcja programu.

Przykłady

```
JEŚLI liczba > 0 TO
 Pisz ('liczba dodatnia')
W PRZECIWNYM RAZIE
 Pisz ('liczba ujemna lub równa zeru')

JEŚLI a <> 0 TO x := -b / a

JEŚLI a = b TO Pisz ('obie liczby są równe')
```

W wyrażeniach logicznych do porównywania wyrażeń liczbowych używa się następujących operatorów relacji:

`=` – równe (nie mylić ze znakiem przypisania :=),
`<>` – nierówne (różne),
`<` – mniejsze,
`>` – większe,
`<=` – mniejsze lub równe,
`>=` – większe lub równe.

Mogą w nich również występować operatory logiczne:

NIE (ang. NOT) – negacja,
I (ang. AND) – iloczyn logiczny,
LUB (ang. OR) – suma logiczna.

Przy obliczaniu wartości wyrażeń logicznych kolejność wykonywania operatorów zależy od ich priorytetów – pierwszeństwo mają operatory o najwyższym priorytecie, a zatem kolejność będzie następująca: negacja, iloczyn, suma, operatory relacji. Można ją zmieniać, stosując nawiasy okrągłe spełniające tę samą funkcję co w wyrażeniach arytmetycznych.

Przykłady

```
JEŚLI (a = 0) I (b <> 0) TO Pisz ('równanie sprzeczne')
JEŚLI NIE a <> 0 TO Pisz ('liczba równa zeru')
```

Instrukcje iteracyjne – służą do deklarowania wielokrotnego wykonywania w programie tych samych operacji. Taki zapis nazywa się pętlą programową, którą można wykonać według dwóch metod:

- **instrukcja powtarzaj:**

```
POWTARZAJ akcję AŻ warunek
```

Interpretacja tej instrukcji jest następująca: ***Powtarzaj*** *czynności opisane przez akcję,* ***aż*** *stwierdzisz, że warunek logiczny jest prawdziwy.* Aby nie było żadnych wątpliwości, należy dodać, że powtarzanie akcji trwa dopóty, dopóki warunek logiczny jest fałszywy.

Przykład

```
ile := 1
POWTARZAJ
  Pisz ('Będę się uczył informatyki')
     ile := ile + 1
  AŻ ile >100
```

Interpretacja tego krótkiego programu jest bardzo prosta: *Pisz sto razy tekst: Będę się uczył informatyki. Gdy zmienna* ***ile*** *przekroczy wartość 100, zakończ... naukę.*

■ **instrukcja dopóki:**

```
DOPÓKI warunek WYKONUJ akcję
```

Interpretacja tej instrukcji jest następująca: *Dopóki warunek logiczny jest prawdziwy, wykonuj czynności opisane przez akcję.*

Należy zwrócić uwagę na różnice między instrukcjami **POWTARZAJ** i **DOPÓKI.** W pierwszym przypadku jest wykonywana akcja, a dopiero potem sprawdzany warunek. Jeśli jest fałszywy, następuje powrót. W drugim przypadku jest sprawdzany warunek i jeśli jest prawdziwy, dopiero wtedy akcja jest wykonywana. Jeśli w instrukcji **DOPÓKI** trzeba wykonać nie jedną, lecz kilka akcji, to należy je ująć klamrą **POCZĄTEK ... KONIEC**. Nie ma takiej potrzeby w przypadku instrukcji **POWTARZAJ**, ponieważ naturalną klamrą jest **POWTARZAJ ... AŻ.**

Przykład

```
ile := 1
DOPÓKI ile <=100 WYKONUJ
 POCZĄTEK
   Pisz ('Będę się uczył informatyki')
   ile := ile + 1
   KONIEC
```

Interpretacja tego programu jest taka sama jak poprzednio.

Sposób zapisu każdej z wymienionych wyżej instrukcji jest odmienny w różnych językach programowania. Ich znaczenie będzie jednak takie samo. Przykładowo, instrukcja przypisania przedstawiona w języku Pascal ma postać: **ile := ile + 1,** w C++ : **ile = ile + 1,** a w polskim Logo: **PRZYPISZ "ile :ile + 1.**

ĆWICZENIE 1

Rozwiąż równanie liniowe $ax + b = 0$.

Analiza problemu

Aby wyznaczyć pierwiastek równania liniowego, należy rozpatrzyć wszystkie możliwe przypadki:

$a = 0$ *i* $b = 0$ – dowolna liczba rzeczywista,

$a = 0$ i $b <> 0$ – równanie sprzeczne,

$a <> 0$ – $x = -b/a$.

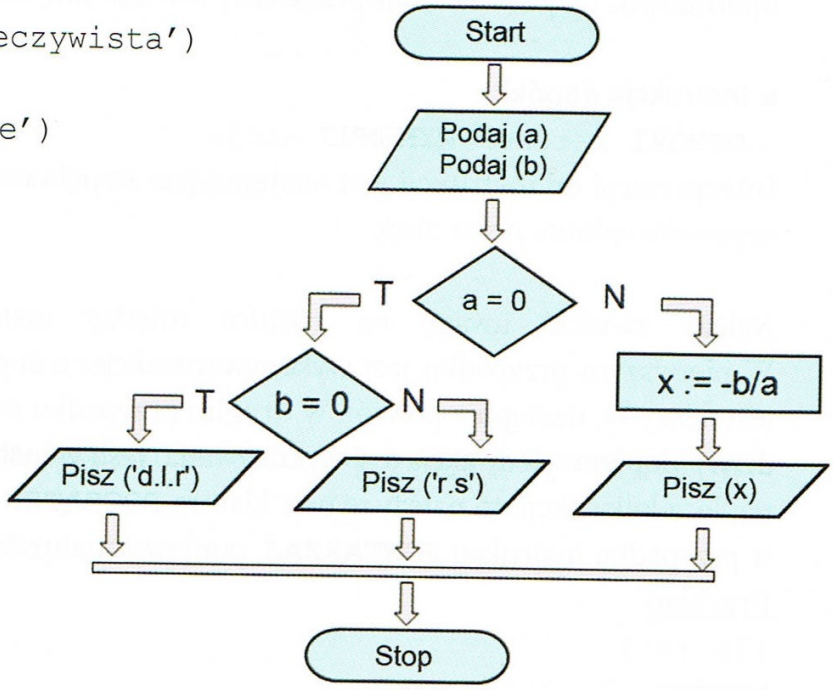
Tego typu analiza sugeruje trzykrotne zastosowanie w algorytmie instrukcji warunkowej. Nie ma to jednak sensu, ponieważ wykluczenie dwóch pierwszych przypadków w sposób naturalny prowadzi do wyznaczenia pierwiastka. Dlatego lepszą propozycją będzie:

- sprawdzenie warunku na wartość *a*,
- gdy *a* będzie równe zeru, sprawdzenie warunku na wartość *b*.

Algorytm

```
Podaj (a)
Podaj (b)
JEŚLI a = 0 TO
  JEŚLI b = 0 TO
    Pisz ('dowolna liczba rzeczywista')
  W PRZECIWNYM RAZIE
    Pisz ('równanie sprzeczne')
W PRZECIWNYM RAZIE
  POCZĄTEK
    x := -b/a
    Pisz (x)
  KONIEC
```

Zapis tego algorytmu w formie schematu blokowego przedstawiono na rys. 2.



Rysunek 2. Rozwiązanie równania liniowego $ax + b = 0$

Zastosowanie wcięć w zapisie algorytmu poprawia czytelność programu – w tym przykładzie wyraźnie widać, że dla pierwszej (zewnętrznej) instrukcji warunkowej: akcja_1 jest kolejną (wewnętrzną) instrukcją warunkową, piszącą *dowolna liczba rzeczywista* albo *równanie sprzeczne*; akcja_2 jest instrukcją złożoną, obliczającą pierwiastek równania liniowego oraz wyprowadzającą z programu (algorytmu) wynik. W algorytmie wystąpiła tzw. **selekcja**, czyli wybór akcji oparty na warunkach logicznych.

ZADANIE 2.1. Przedstaw za pomocą schematu blokowego przebieg rozwiązania równania liniowego $ax + b = 0$ i napisz algorytm w pseudojęzyku, przedstawiając go za pomocą trzech instrukcji warunkowych:

```
a = 0 i b = 0
a = 0 i b <> 0
a <> 0
```

Dla ułatwienia podajemy poniżej zapis pierwszego warunku w pseudojęzyku:

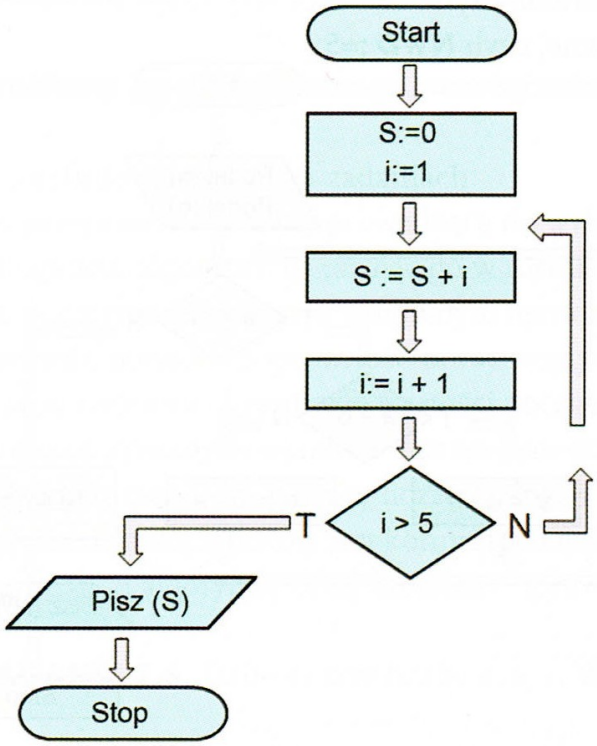
```
JEŚLI (a = 0) I (b = 0) TO Pisz ('dowolna liczba rzeczywista')
```

ĆWICZENIE 2

Oblicz sumę pięciu kolejnych liczb naturalnych rozpoczynających się liczbą 1. Wynik oznacz jako *S*.

Analiza problemu

Zanim rozpoczniemy sumowanie kolejnych liczb naturalnych, przyjmijmy dwa oczywiste założenia, które określają warunki początkowe: suma (*S*) jest równa 0, natomiast pierwszą sumowaną liczbą (*i*) będzie 1. W kolejnych pięciu krokach będziemy powiększali sumę o kolejną liczbę naturalną, czyli: 2, 3, 4, 5. Gdy *i* przyjmie wartość 6, kończymy sumowanie i wyprowadzamy wynik, który będzie wynosił: 1 + 2 + 3 + 4 + 5 = 15.



Rysunek 3. Algorytm sumowania pięciu kolejnych liczb naturalnych

Powyższa analiza prowadzi do schematu blokowego przedstawionego na rys. 3.

Algorytm

```
S := 0
i := 1
POWTARZAJ
   S := S + i
   i := i + 1
AŻ i > 5
Pisz (S)
```

Wniosek

W algorytmie wystąpiła tzw. **iteracja**, czyli wielokrotne wykonywanie (powtarzanie) pewnych instrukcji, dopóki nie został spełniony warunek – w tym przykładzie było to sumowanie kolejnych liczb naturalnych.

ĆWICZENIE 3

Wyznacz największy wspólny dzielnik NWD dwóch liczb naturalnych *a*, *b*.

Analiza problemu

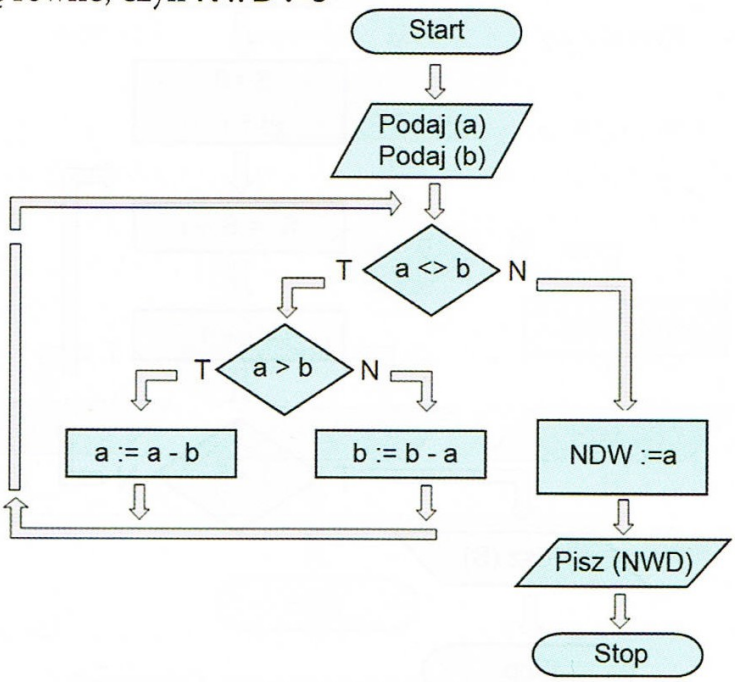
Jedną z metod wyznaczenia NWD dwóch liczb naturalnych jest odejmowanie mniejszej liczby od większej i przypisanie wyniku zmiennej reprezentowanej przez odjemną. Czynność tę wykonujemy dopóty, dopóki nie otrzymamy dwóch równych sobie liczb; będzie to największy wspólny dzielnik. Przykładowo, dla $a = 25$, $b = 15$ odejmowanie będzie przebiegało następująco:

$a > b \rightarrow \quad a := 25 - 15 = 10 \quad \rightarrow b$ pozostaje bez zmian, czyli 15,
$a < b \rightarrow \quad b := 15 - 10 = 5 \quad \rightarrow a$ pozostaje bez zmian, czyli 10,
$a > b \rightarrow \quad a := 10 - 5 = 5 \quad \rightarrow b$ pozostaje bez zmian, czyli 5
$a = b \rightarrow \quad$ NWD $:= 5 \quad \rightarrow$ obie liczby są równe, czyli **NWD :=5**

Algorytm

```
Podaj (a)
Podaj (b)
DOPÓKI a <> b WYKONUJ
  JEŚLI a > b TO
    a := a - b
  W PRZECIWNYM RAZIE
    b := b - a
NWD := a
Pisz (NWD)
```

Zapis tego algorytmu w formie schematu blokowego przedstawiono na rys. 4.



Rysunek 4. Wyznaczenie największego wspólnego dzielnika dwóch liczb naturalnych *a*, *b*

Metodę wyznaczania największego wspólnego dzielnika dwóch liczb naturalnych podał Euklides, żyjący w IV wieku p.n.e. Metoda (nazywana **algorytmem Euklidesa**) polega na dzieleniu całkowitym liczby większej przez mniejszą i przypisaniu reszty zmiennej reprezentowanej przez dzielną. Czynność tę wykonujemy dopóty, dopóki *a* i *b* są większe od zera. Największym wspólnym dzielnikiem będzie ta z liczb *a* i *b*, która pozostała większa od zera. Analiza algorytmu „krok po kroku" dla $a = 25$ i $b = 15$ będzie wyglądała następująco (funkcję *reszta z dzielenia całkowitego* oznaczymy jako MOD):

$a > b \rightarrow a := 25$ MOD $15 = 10 \rightarrow b$ pozostaje bez zmian, czyli 15,
$a < b \rightarrow b := 15$ MOD $10 = 5 \rightarrow a$ pozostaje bez zmian, czyli 10,
$a > b \rightarrow a := 10$ MOD $5 = 0 \rightarrow a = 0$, czyli NWD :=5.

[3] Ciąg znaków ujęty w nawiasy klamrowe jest komentarzem ignorowanym przez algorytm.

Algorytm Euklidesa wyrażony w pseudojęzyku

```
Podaj (a)
Podaj (b)
DOPÓKI (a > 0) I (b > 0) WYKONUJ
  JEŚLI a > b TO a := a MOD b
    W PRZECIWNYM RAZIE b := b MOD a
NWD := a + b {zwróć uwagę, że jedna z liczb a lub b jest równa zeru}[3]
Pisz (NWD)
```

ZADANIE 2.2. Opracuj schemat blokowy dla algorytmu Euklidesa.

Problemy do samodzielnego rozwiązania

W zdefiniowanych niżej zadaniach:

- przeprowadź analizę prowadzącą do wyboru metody rozwiązania;
- opracuj algorytm, wyrażając go w formie schematu blokowego;
- przedstaw algorytm w umownym języku programowania.

Sprawdź ponadto poprawność opracowanych algorytmów tzw. metodą **testowania**. Polega ona na wybraniu dowolnych wartości początkowych, wykonaniu dla nich algorytmu i porównaniu otrzymanych wyników ze znanymi już poprawnymi wynikami. Czynność ta powinna być powtórzona dla pewnej liczby różnych wartości zmiennych. Idealnym narzędziem do wykonania takich testów jest komputer. Dlatego też w kolejnym punkcie wszystkie przedstawione tutaj algorytmy będą zapisane i sprawdzone w języku programowania.

ZADANIE 2.3. Dane są trzy liczby *a, b, c*. Wyznacz największą z nich.

ZADANIE 2.4. Dane są trzy odcinki o długości *a, b, c*. Sprawdź, czy można z nich zbudować trójkąt.