PODSTAWY ALGORYTMIKI

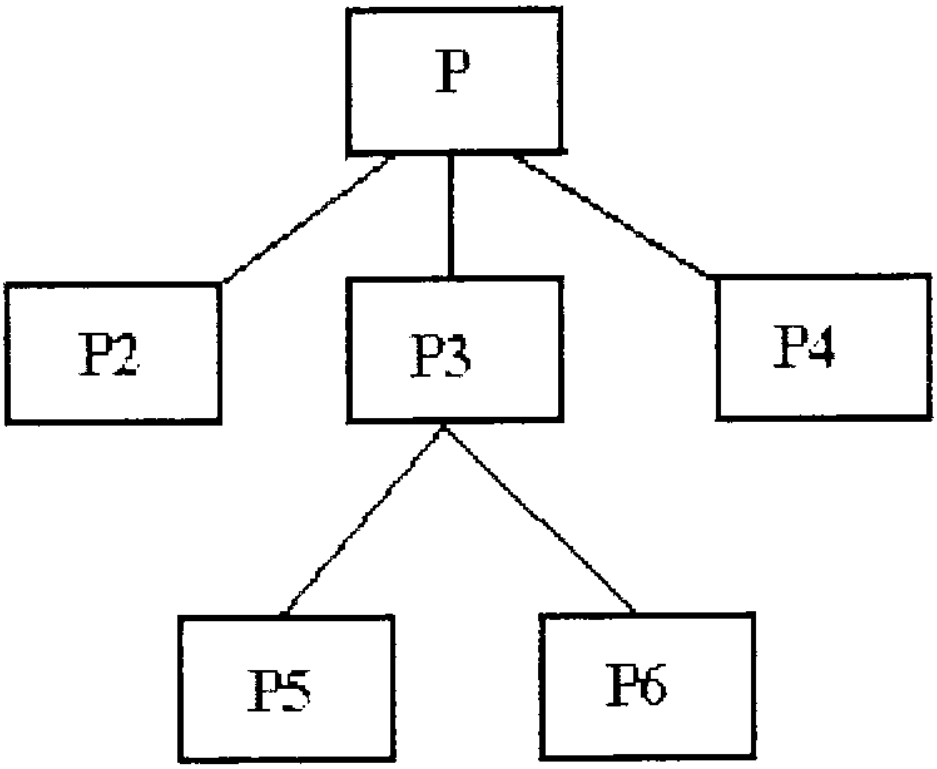
Algorytm – sposób rozwiązania danego problemu
Może być opisany słownie, może być przedstawiony wzorem matematycznym lub za pomocą schematów blokowych.

Programowanie jest to metoda rozwiązania problemu, poprzez podanie w określonej postaci algorytmu rozwiązania.

W ćwiczeniach, realizowanych w ramach niniejszego kursu, algorytm wyrażony będzie w języku MathLAB i będzie nazywany **programem**.

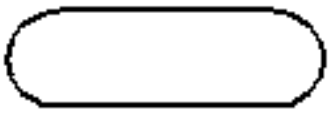
Programy składają się z ciągu instrukcji zapisanych w określonym języku programowania, których wykonanie prowadzi do zrealizowania przez komputer postawionego mu zadania.

Przygotowując program komputerowy należy go podzielić na mniejsze części (**moduły**) i zająć się osobno każdym poszczególnym zadaniem.
Zadany do wykonania problem należy sformułować w sposób jednoznaczny. Dany problem dzieli się na kolejne **cząstkowe** problemy. Następuje podzielenie problemu na problemy cząstkowe, które łącznie wpływają na rozwiązanie problemu.



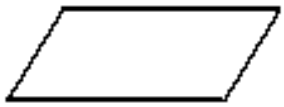
Często używanym sposobem przedstawiania algorytmu rozwiązania określonego zadania jest utworzenie **schematu blokowego**. Schemat blokowy przedstawia w sposób graficzny zbiór operacji i wzajemnych powiązań między nimi, które określają kolejność wykonywanych operacji.
Schematy blokowe są tzw. **metajęzykiem**. Oznacza to, że jest to język bardzo ogólny, służy do opisywania algorytmów w taki sposób, by na jego podstawie można było je zaimplementować w każdym języku.
Częściami składowymi schematów blokowych są proste figury geometryczne, np. prostokąt, romb, koło, równoległobok itd... W tych figurach umieszczamy warunki oraz proste instrukcje, przy czym mogą być one związane z jakimś konkretnym językiem (np. symbolem instrukcji przypisania może być ":=" tak, jak w Pascalu lub "=" tak, jak w C) Jeśli tworząc schemat nie jesteśmy jeszcze zdecydowani w jakim języku będziemy pisali nasz program lub tworzymy schemat dla kogoś, to lepiej jest stosować notację bardziej symboliczną, np. instrukcję przypisania zapisywać jako strzałkę skierowaną od wartości przypisywanej do zmiennej.
Do graficznego przedstawiania i analizy schematów blokowych można użyć programu **Magiczne Bloczki (**http://www.algorytm.org/downloads/kurs/mb.zip


Poszczególne elementy schematu łączy się za pomocą strzałek. W większości przypadków blok ma jedną strzałkę wchodzącą i jedną wychodzącą, lecz są także wyjątki (omówię je poniżej).


Ta figura oznacza początek lub koniec algorytmu. W każdym algorytmie musi się znaleźć dokładnie jedna taka figura z napisem "Start" oznaczająca początek algorytmu oraz dokładnie jedna figura z napisem "Stop" oznaczająca koniec algorytmu. Najczęściej popełnianym błędem w schematach blokowych jest umieszczanie kilku stanów końcowych, zależnych od sposobu zakończenia programu. Jest to niedopuszczalne, w programie mamy przecież dokładnie jedną instrukcję "end." Blok symbolizujący początek algorytmu ma dokładnie jedną strzałkę wychodzącą a blok symbolizujący koniec ma co najmniej jedną strzałkę wchodzącą.

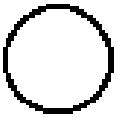

Jest to figura oznaczająca proces. W jej obrębie umieszczamy wszelkie obliczenia lub podstawienia. Proces ma dokładnie jedną strzałkę wchodzącą i dokładnie jedną strzałkę wychodzącą.

Romb symbolizuje blok decyzyjny. Umieszcza się w nim jakiś warunek (np. "x>2"). Z dwóch wybranych wierzchołków rombu wyprowadzamy dwie możliwe drogi: gdy warunek jest spełniony (strzałkę wychodzącą z tego wierzchołka należy opatrzyć etykietą "Tak") oraz gdy warunek nie jest spełniony. Każdy romb ma dokładnie jedną strzałkę wchodzącą oraz dokładnie dwie strzałki wychodzące.

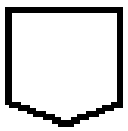


Równoległobok jest stosowany do odczytu lub zapisu danych. W jego obrębie należy umieścić stosowną instrukcję np. Read(x) lub Write(x) (można też stosować opis słowny np. "Drukuj x na ekran"). Figura ta ma dokładnie jedną strzałkę wchodzącą i jedną wychodzącą.

Ta figura symbolizuje proces, który został już kiedyś zdefiniowany. Można ją porównać do procedury, którą definiuje się raz w programie, by następnie móc ją wielokrotnie wywoływać. Warunkiem użycia jest więc wcześniejsze zdefiniowanie procesu. Podobnie jak w przypadku zwykłego procesu i tu mamy jedno wejście i jedno wyjście.



Koło symbolizuje tzw. łącznik stronicowy. Może się zdarzyć, że chcemy "przeskoczyć" z jednego miejsca na kartce na inne (np. by nie krzyżować strzałek). Możemy w takim wypadku posłużyć się łącznikiem. Umieszczamy w jednym miejscu łącznik z określonym symbolem w środku (np. cyfrą, literą) i doprowadzamy do nie go strzałkę. Następnie w innym miejscu kartki umieszczamy drugi łącznik z takim samym symbolem w środku i wyprowadzamy z niego strzałkę. Łącznik jest często porównywany do teleportacji (z jednego miejsca na kartce do drugiego). Łączniki występują więc w parach, jeden ma tylko wejście a drugi wyjście.



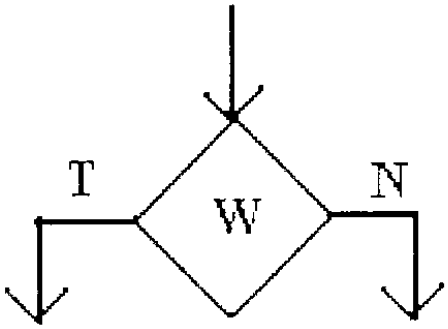
Ten symbol to łącznik międzystronicowy. Działa analogicznie jak pierwszy, lecz nie w obrębie strony. Przydatne w złożonych algorytmach, które nie mieszczą się na jednej kartce. Uwaga: jeśli stosujemy oba typy łączników w schemacie, to najlepiej jest stosować liczby do identyfikowania jednych i litery do drugich. Dzięki temu nie dojdzie do pomyłki.

Podstawowe elementy graficzne, zwane węzłami, które występują na schemacie blokowym

a) Węzeł funkcyjny

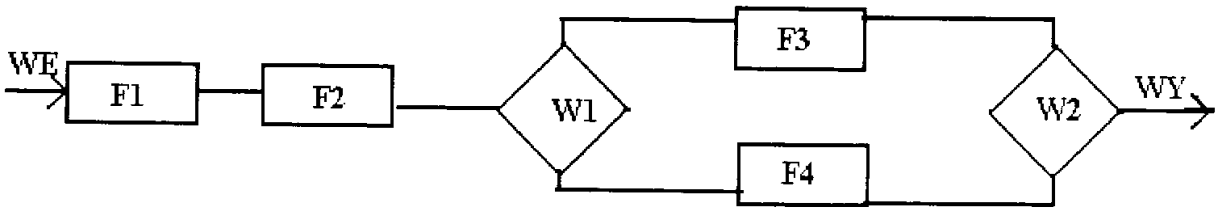
Jest to prostokąt, w którym znajdują się instrukcje. Z węzła funkcyjnego wychodzi tylko jedno połączenie.

b) Węzeł warunkowy



Jest to romb, w którym umieszcza się sprawdzany warunek. Warunek może przyjąć jedną z dwóch wartości: Tak (T) lub Nie (N). Jeżeli warunek logiczny jest spełniony, otrzymujemy wartość logiczną **Tak**, jeśli nie jest spełniony – **Nie**. Z rombu wychodzą dwa połączenia. Jedno z nich reprezentuje spełnienie warunku **Tak**, drugie połączenie jest dla warunku **Nie**.

Program właściwy zbudowany jest z węzłów podstawowych:

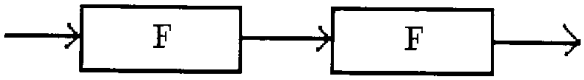


Przez każdy węzeł prowadzi droga prowadząca z wejścia do wyjścia.

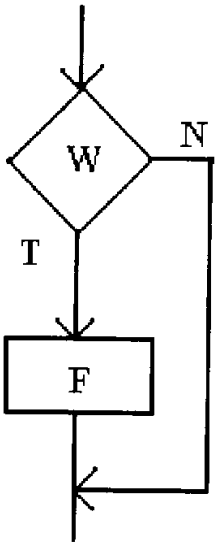
Konstrukcje różnych rodzajów węzłów

1. Węzeł funkcyjny

2. Sekwencja dwóch węzłów funkcyjnych

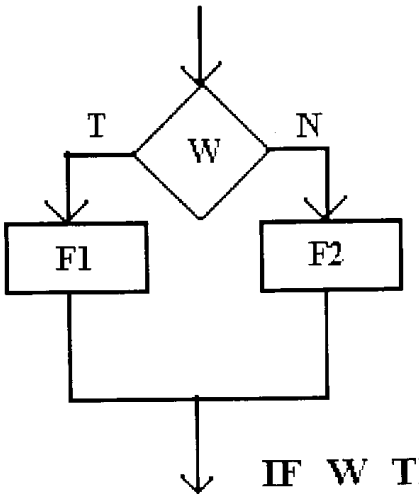


3. Węzły, które tworzą **warunek prosty**



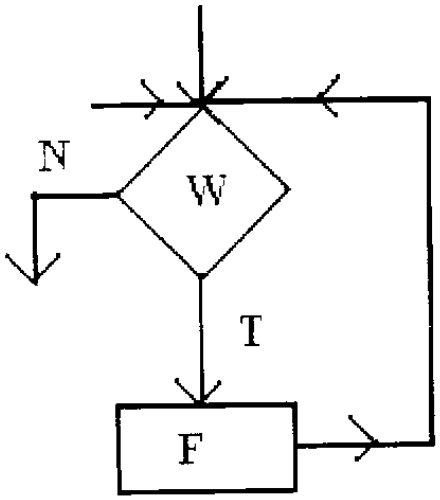
IF W THEN N

4. Węzły, które tworzą **warunek złożony**



IF W THEN F1 ELSE F2

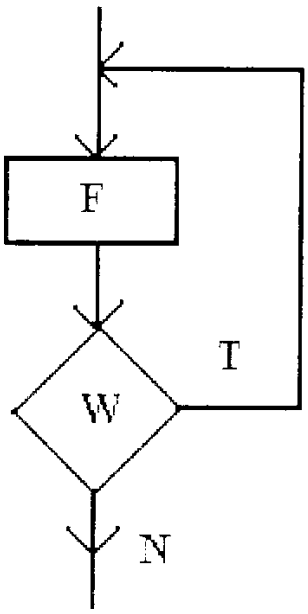
5. Węzły, które tworzą **cykl** – program działa w pętli warunkowej



WHILE W DO F

WHILE warunek
    wyrażenia
END

6. Węzły, które tworzą **iteracje** – program działa w pętli.

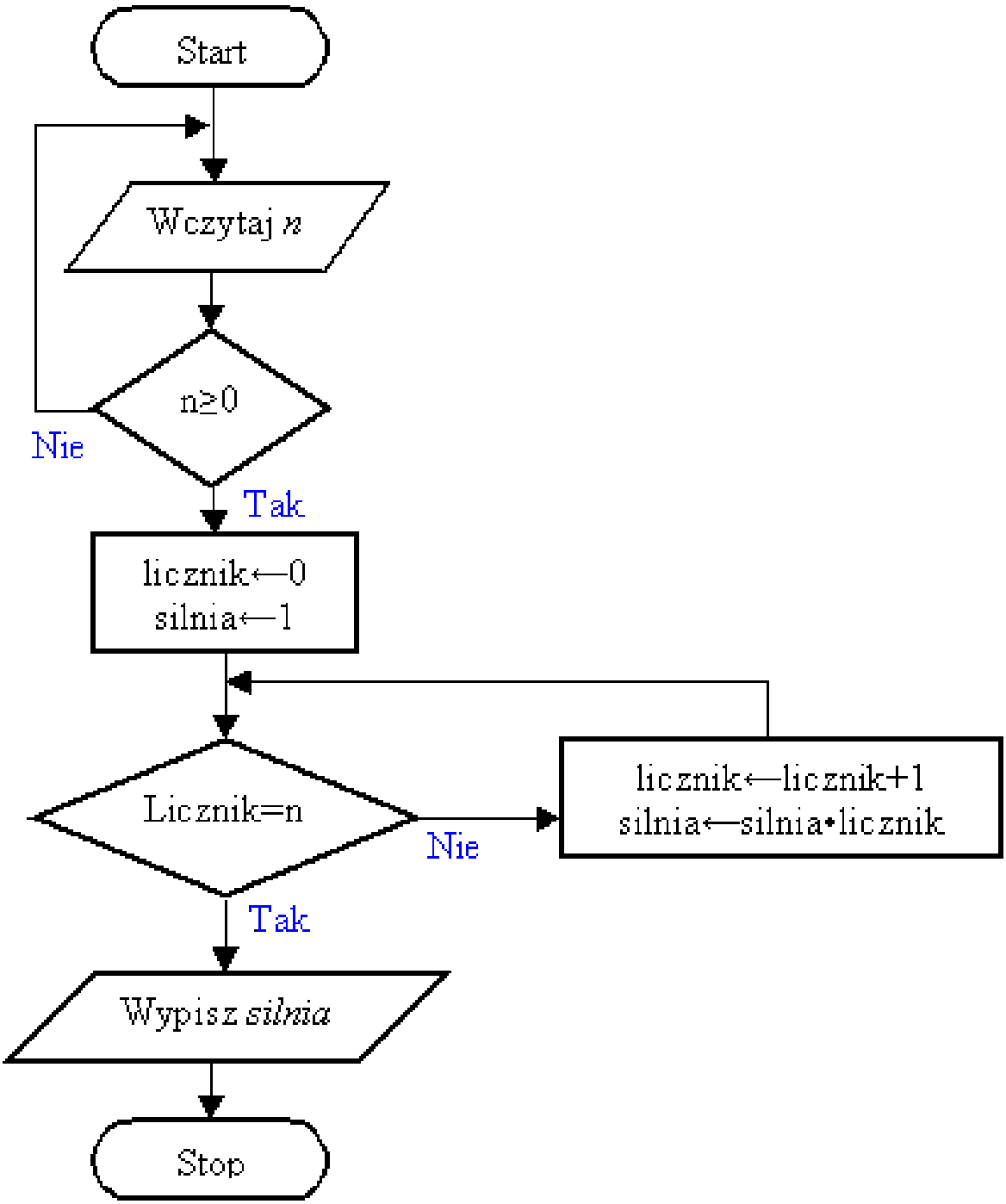


DO F UNTIL W

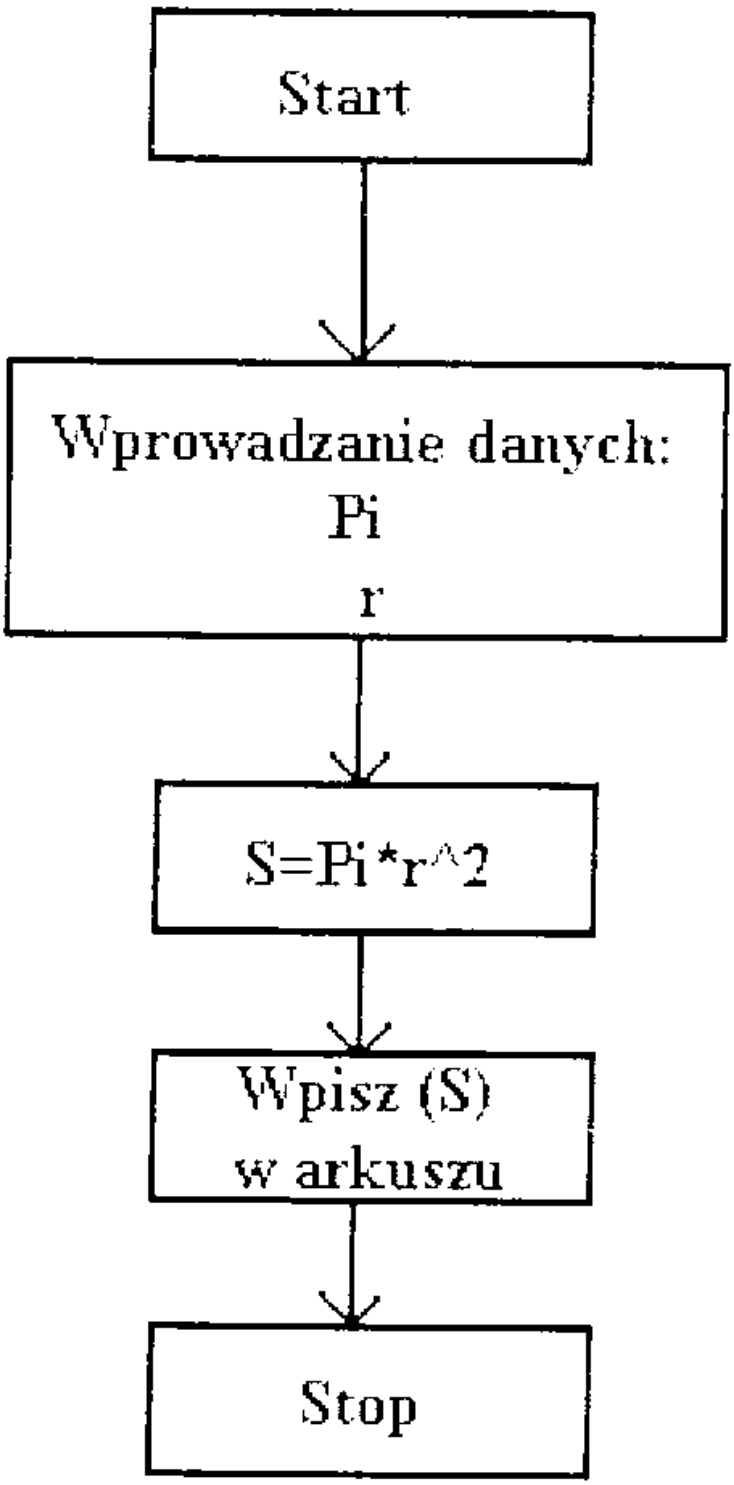
Przykładowy algorytm

Zdefiniujmy iteracyjną wersję silni. Dla przypomnienia: rekurencyjna definicja silni wygląda następująco

0!=1
n!=n* (n-1)!

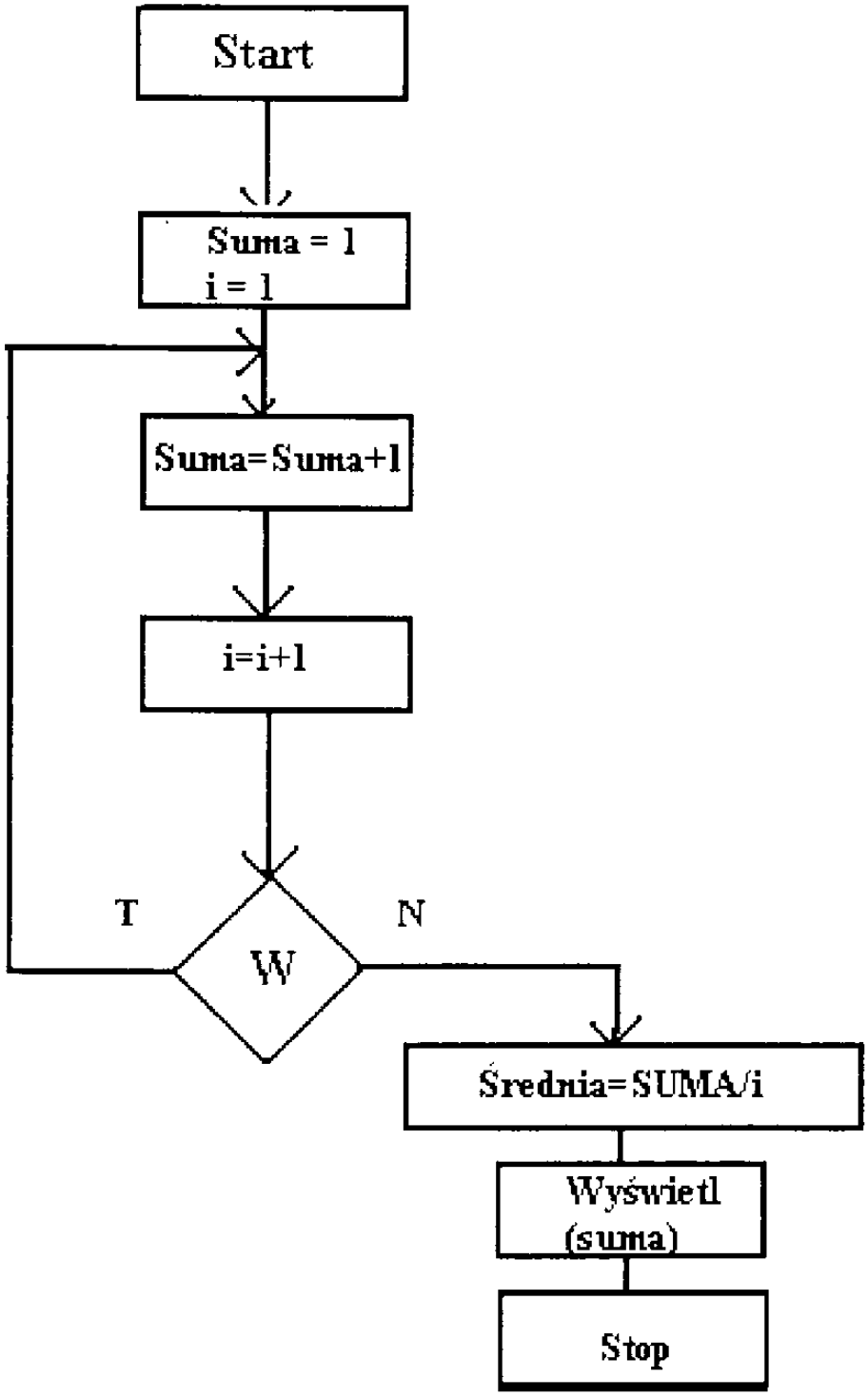


Algorytm 1: Oblicz pole koła o zadanym promieniu

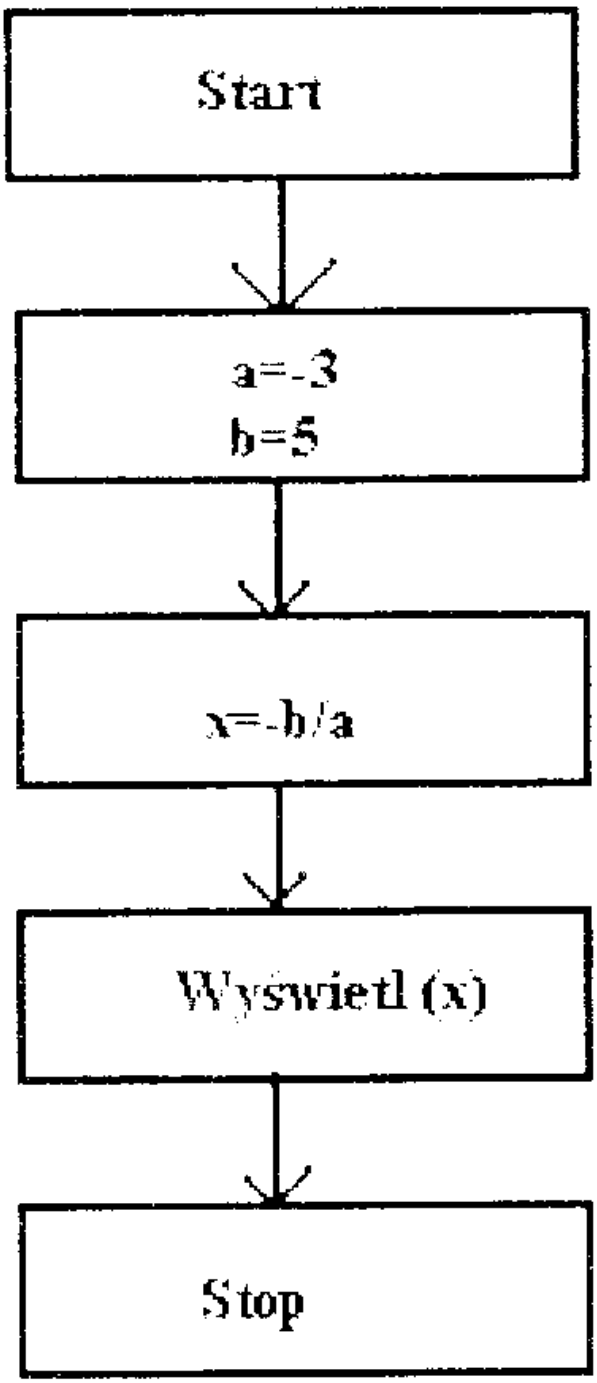


Algorytm 2: Oblicz wartość średnią z sumy pierwszych dziesięciu liczb

Przykład iteracji – wielokrotne wykonywanie określonych instrukcji

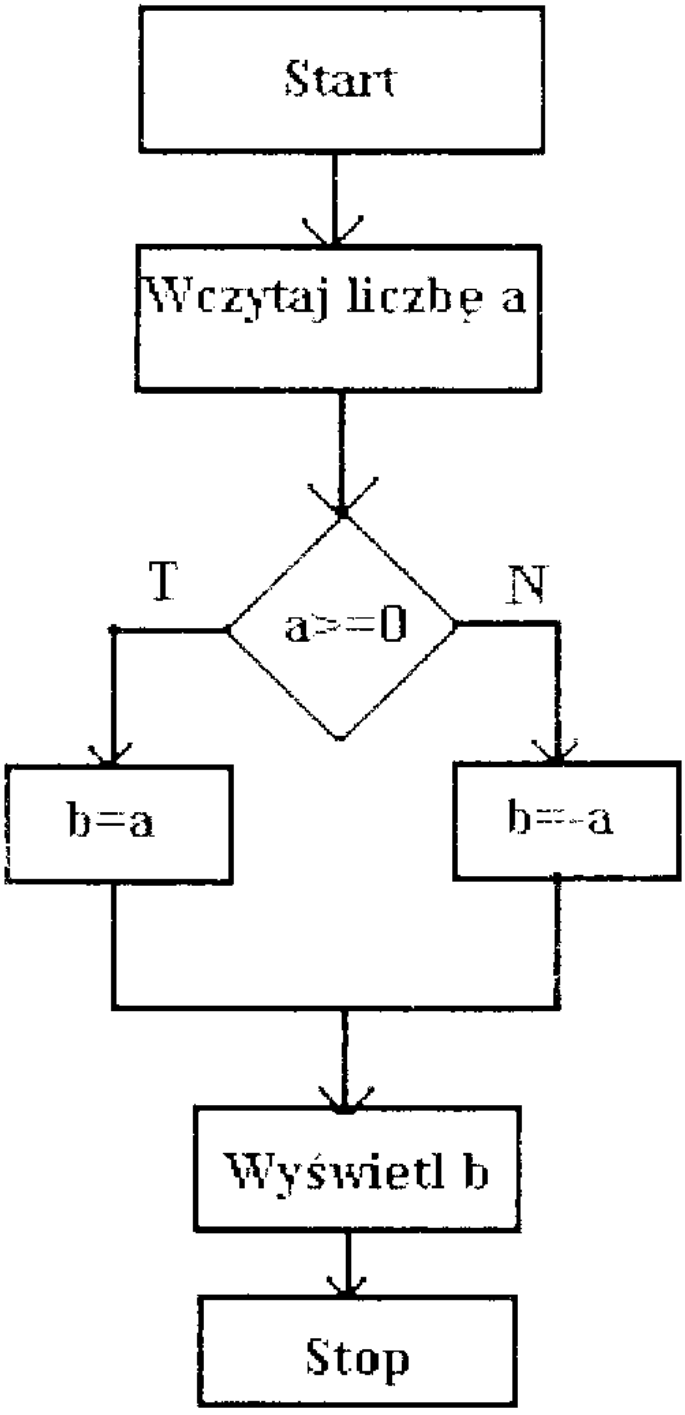


Algorytm 3: Rozwiąż równanie liniowe ax+b=0



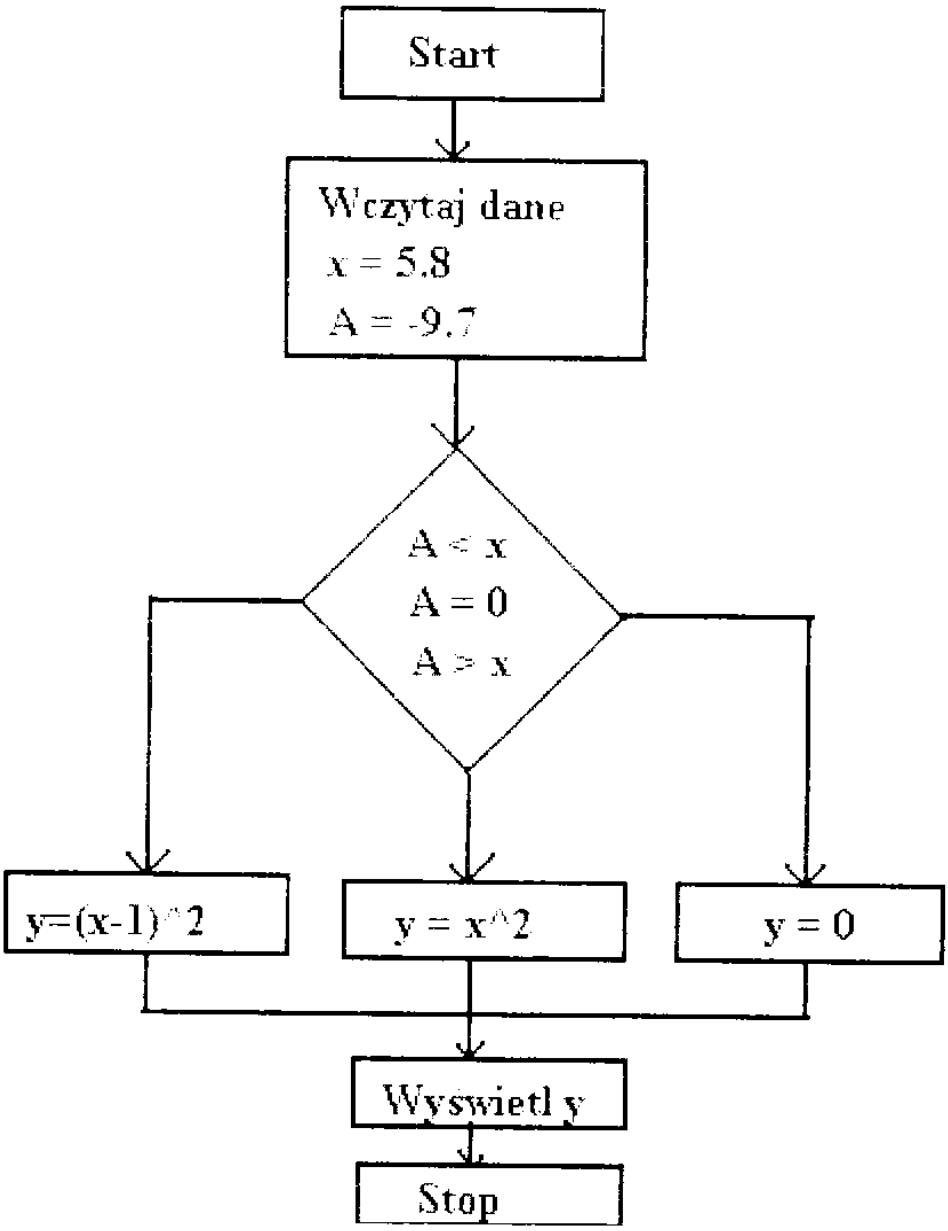
Algorytm 4: Oblicz wartość bezwzględną liczby – Przykład warunku złożonego

```
IF warunek
    wyrażenie
  ELSEIF warunek
    wyrażenie
  ELSE
    wyrażenie
  END
```

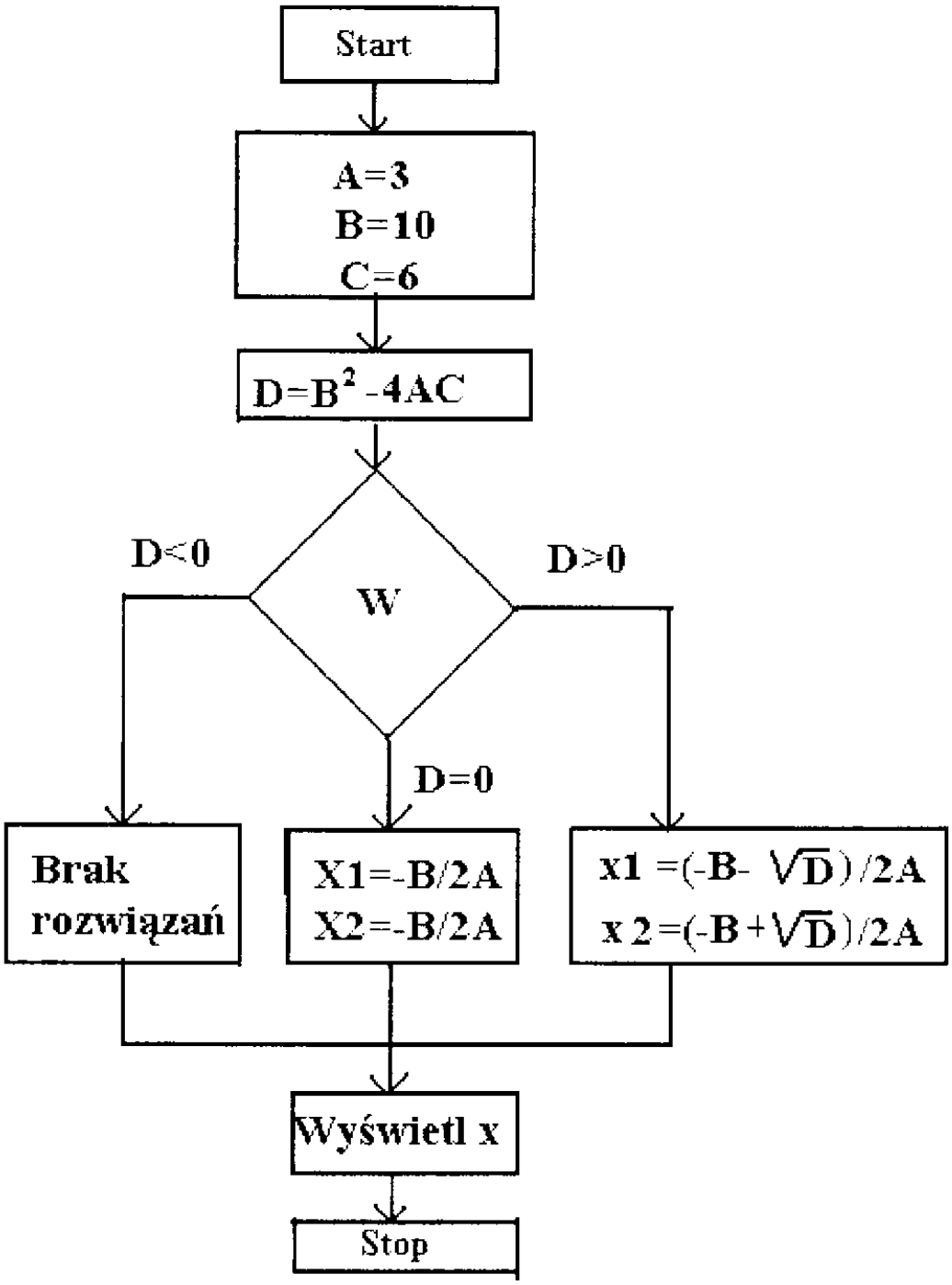


Algorytm 5: Obliczanie wartości y

$$y = \begin{cases} (x-1)^2 & dla \quad A < x \\ x^2 & dla \quad A = 0 \\ 0 & dla \quad A > x \end{cases}$$



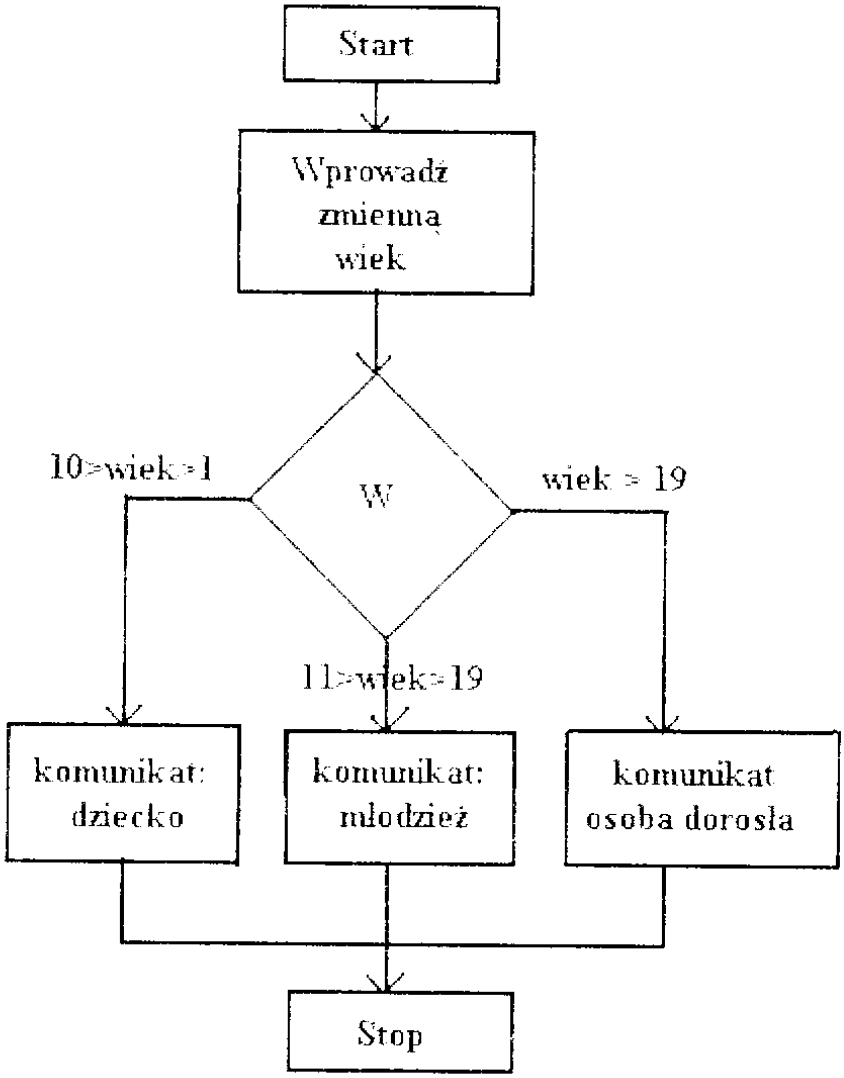
Algorytm 6: Rozwiąż równanie drugiego stopnia $Ax^2+Bx+C=0$



Algorytm 7: Algorytm wykorzystujący instrukcję wyboru

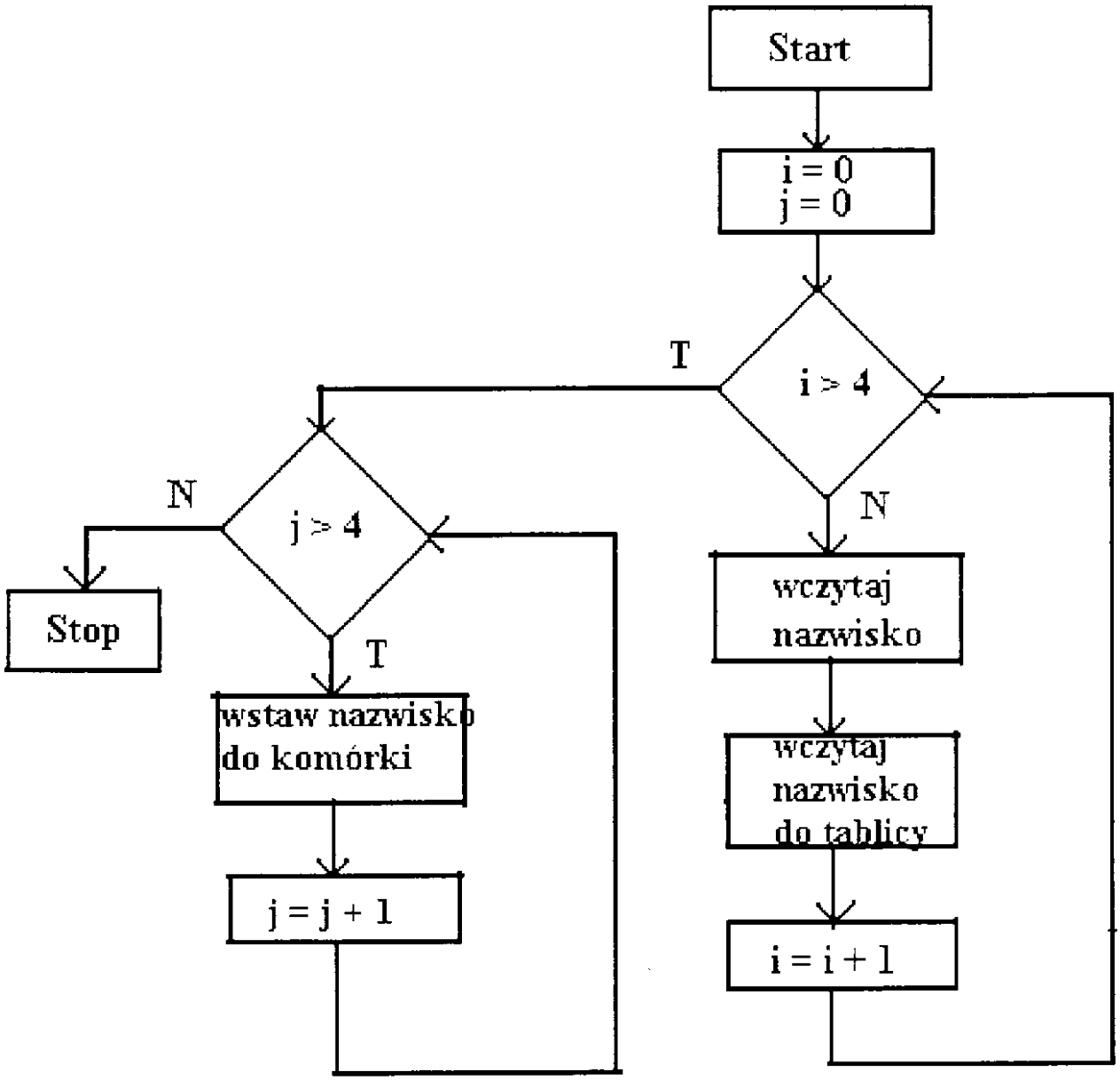
```
SWITCH wyrażenie wyboru
      CASE przypadek,
        wyrażenie, ..., wyrażenie
      CASE {przypadek_1, przypadek_2, przypadek_3,...}
        wyrażenie, ..., wyrażenie
      OTHERWISE,
        wyrażenie, ..., wyrażenie
    END
```

W jakim przedziale wiekowym mieści się osoba o zadanej liczbie lat?



Algorytm 7: Wpisywanie danych do tablicy jednowymiarowej

Wczytanie danych do tablicy i następnie wstawienie danych z tablicy do arkusza

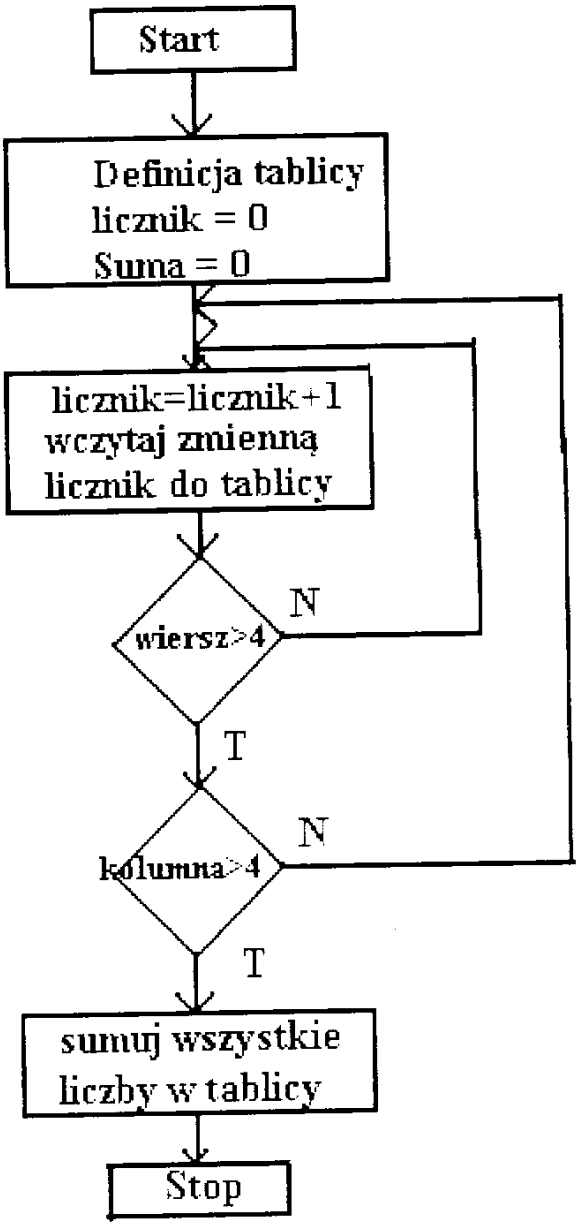


Algorytm 7: Wpisywanie danych do tablicy dwuwymiarowej

Przykład pętli bezwarunkowej

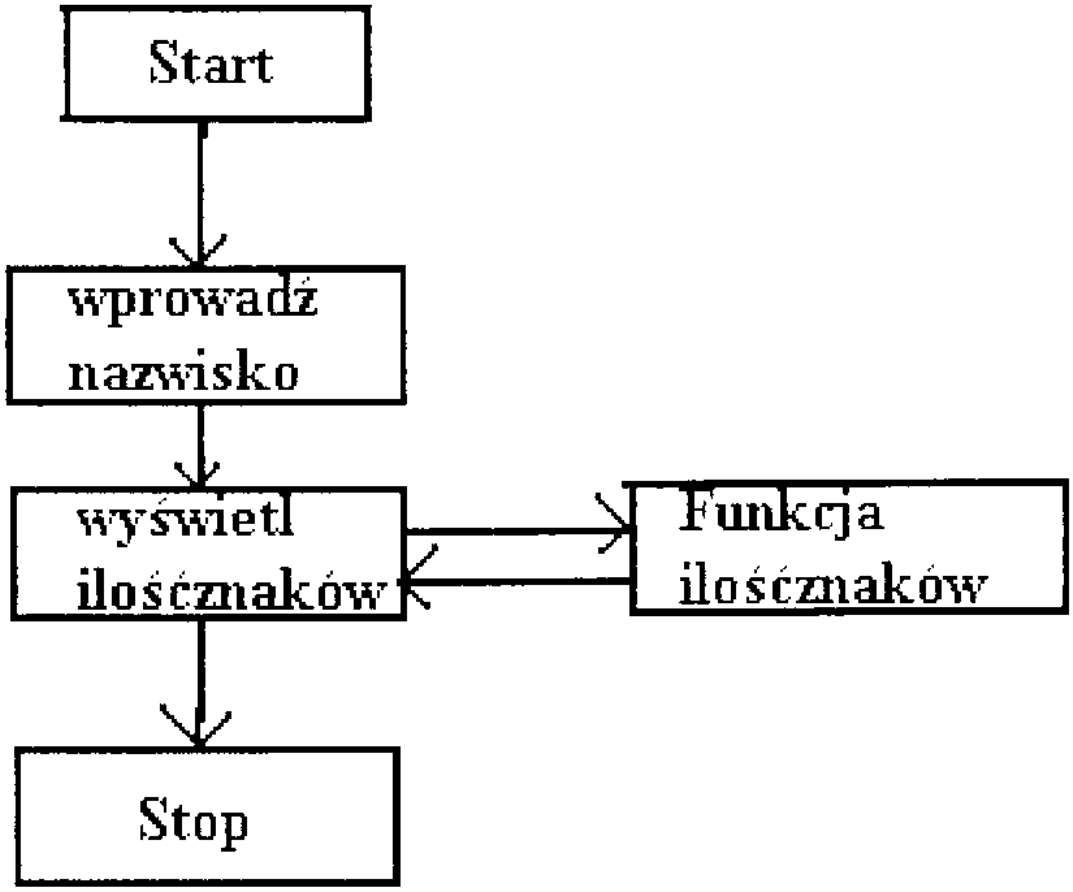
FOR zmienna = wartość, wyrażenie, ..., wyrażenie END

Wczytanie danych do tablicy dwuwymiarowej



Algorytm 7: Przykład rekurencji – obliczanie ilości znaków w nazwisku

Rekurencja zachodzi wtedy, gdy podprogram wywołuje sam siebie za pośrednictwem innego podprogramu. Oznacza to, że w tekście pierwszego podprogramu znajduje się wywołanie drugiego, w tekście drugiego wywołanie pierwszego.



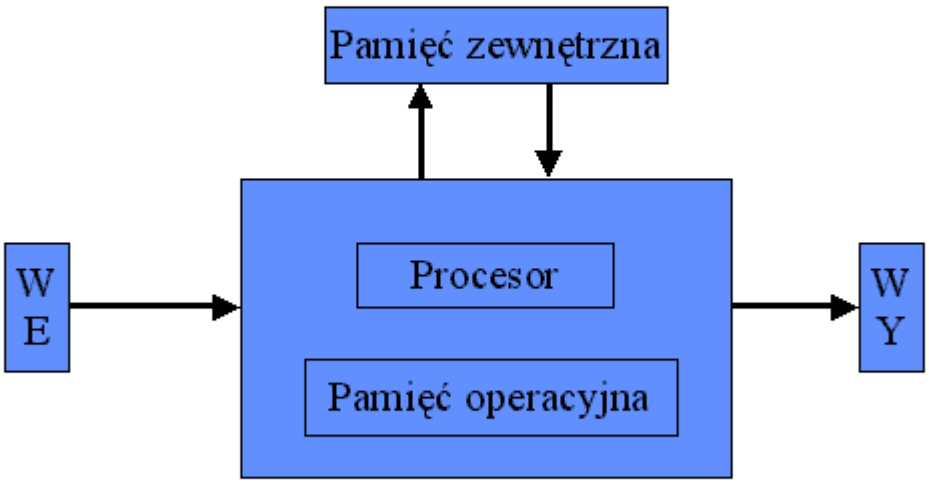
Podstawy programowania komputerów

(wg Zdzisław Szyjewski, Instytut Informatyki w Zarządzaniu, Uniwersytet Szczeciński)
http://iiwz.univ.szczecin.pl/zszyjewski/progr1

Podstawowe pojęcia:

- zasady działania komputera
- definicja pojęć program i programowanie
- algorytmizacja problemu
- środowisko pracy programisty
- języki programowania

Zasady budowy komputera

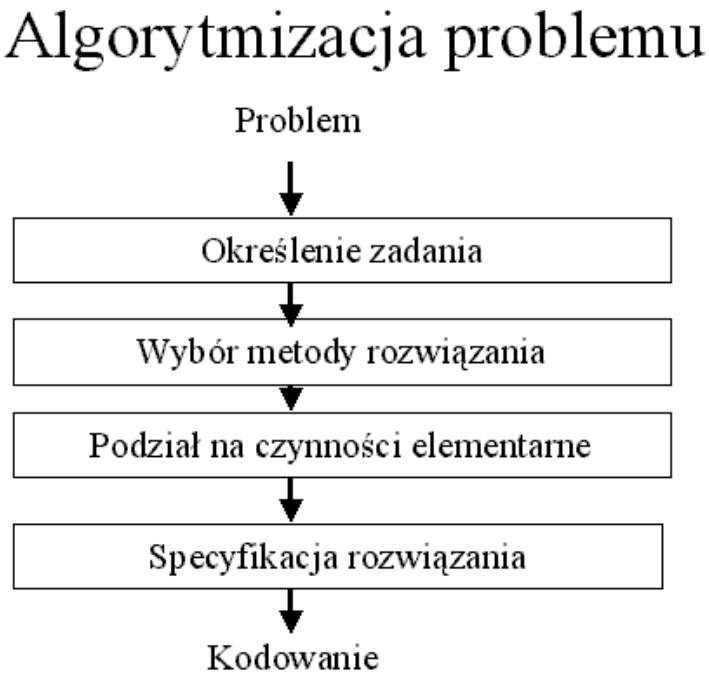


Program – jest to przekład problemu użytkownika na język maszyny (komputera). Programy stanowią skonkretyzowane sformułowanie abstrakcyjnych algorytmów na podstawie określonej reprezentacji i struktury danych (*Niklaus Wirth*).

Algorytmy + Struktury danych = Programy

Umiejętność programowania to specyficzny dla komputerów algorytmiczny sposób postawienia zadania, rozwiązanie go przy pomocy standardowych mechanizmów dostępnych w różnych językach programowania.

Algorytm jest zbiorem reguł pozwalającym mechanicznie wykonywać wszystkie czynności odpowiadające pewnemu typowi pracy. W informatyce jest to rozkład zadania na operacje elementarne akceptowalne przez komputer.



Podział czasu programowania

- Projektowanie programu – 1/3
- Kodowanie programu – 1/6
- Testowanie programu – 1/2

Metody specyfikacji algorytmu

- opis słowny
- zapis matematyczny
- pseudokodowanie
- metody graficzne
- metody tablicowe

Definicja języka

- alfabet – litery, cyfry, znaki specjalne
- składnia – zbiór reguł definiujących sposób konstruowania łańcucha symboli
- semantyka – określenie, jak te łańcuchy symboli należy rozumieć (wykonywać)

Środowisko programowania

- pamięć stała procesora
- system operacyjny
- oprogramowania podstawowe
- języki programowania
  - wewnętrzne
  - symboliczne
  - wyższego rzędu

Podstawowe pojęcia i zasady

- bajt
- słowo maszynowe
- rejestr
- systemy liczenia i prezentacji
  - system ósemkowy
  - system heksadecymalny

Wady i zalety języka wewnętrznego

- trudny w użyciu
- łatwo o błędy
- powolny proces programowania
- trudna modyfikacja
-

- pełne wykorzystanie pamięci stałej
- optymalizacja obliczeń

Język symboliczny – język symboliczny, zwany assemblerem, wprowadza ułatwienia dla programisty. Są to:

- mnemoniczne nazwy operacji
- operowanie nazwami a nie adresami
- możliwość stosowania makrooperacji
- możliwość swobodnego modyfikowania programu

Język symboliczny - przykład

```
START  X'1000'
BALR   12,0
USING  *,12
SR     1,1
DC     X'0011'
ST     1,RESULT1
LA     2,2
LA     3,4
DC     X'0023'
ST     2,RESULT2
L      3,=F'-2'
```

Języki proceduralne:

- ukierunkowane na problem, a nie na procesor
- postać programu czytelna dla człowieka
- całkowite operowanie na nazwach
- uogólnione nazwy operacji

Języki proceduralne – ukierunkowane na obliczenia (np. ALGOL, FORTRAN**):**

- bogaty aparat obliczeniowy
- słabe wspomaganie definiowania i operowania na danych

Języki proceduralne – ukierunkowane na przetwarzanie danych (np. COBOL, PL/1**):**

- cztery podstawowe działania
- bogaty aparat definiowania i operowania na danych

Inne języki proceduralne – uniwersalne

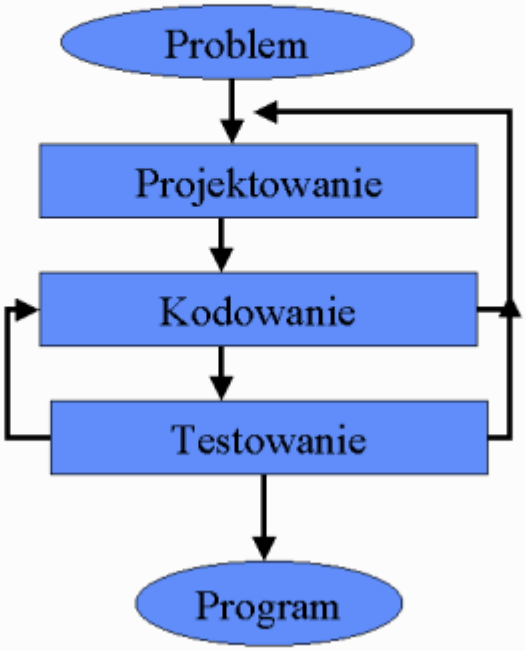
- BASIC
- ADA
- PASCAL
- C

Inne języki proceduralne – specjalizowane (np. SIMULA – symulacje komputerowe, PROLOG – systemy ekspertowe, LISP – operowanie na strukturach listowych, RPG – generator programów, MATLAB – obliczenia numeryczne i wizualizacja wyników)

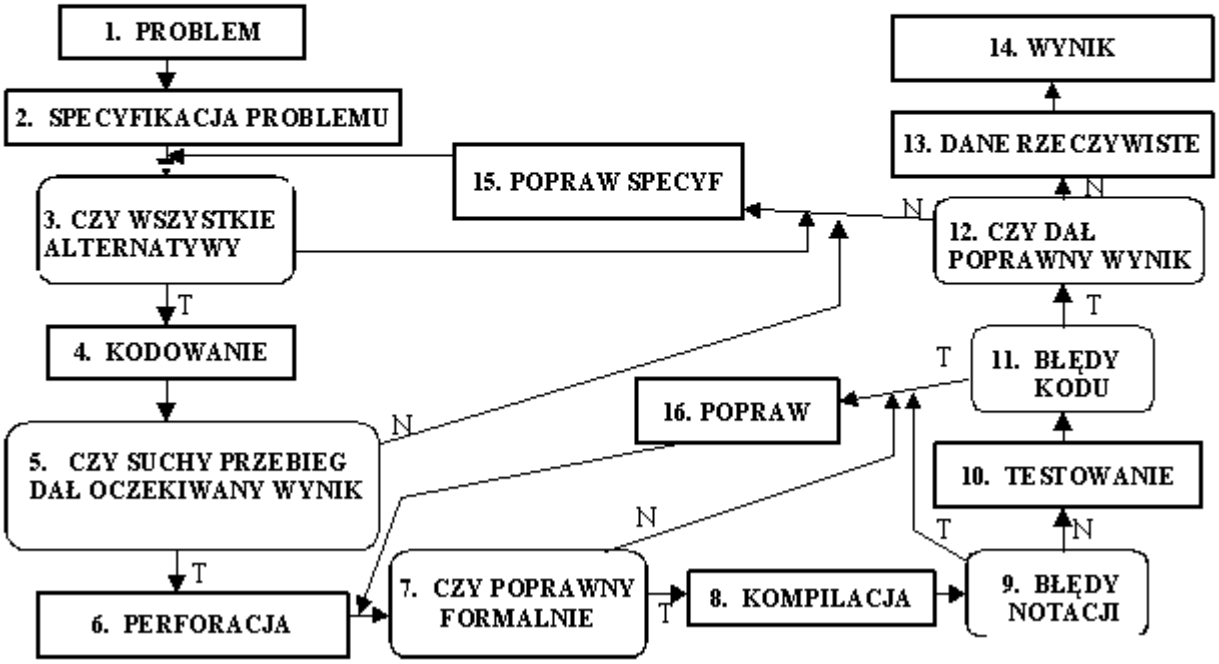
Tok procesu programowania

- podstawowe fazy programowania – projektowanie, kodowanie, testowanie
- rekurencyjny charakter toku prac
- wykorzystanie sprzętu komputerowego
- faza testowania

Podstawowe fazy programowania



Schemat blokowy procesu programowania



Kodowanie programu

- wybór języka
- organizacja pisania programu
  – praca indywidualna
  – praca zespołowa
- szkielet programu
- plan testów
- stosowane narzędzia

„Suchy przebieg" programu

- współpraca w zespole
- eliminacja prostych błędów
  – formalnych
  – logicznych
- pierwsza faza testowania
- inne spojrzenie na program, korzystanie z cudzych doświadczeń

Testowanie programu

Testowanie programu ma dwa główne cele:
– wykrycie i usunięcie błędów
– ocenę niezawodności

Jak wygląda testowanie programu?

- plan testów
- poprawność formalna
- poprawność logiczna
- zgodność ze specyfikacją
- testowanie niezawodności

Rekurencyjny charakter prac programistycznych

- eliminowanie błędów
- wcześniej wykryty błąd mniej kosztuje
- konieczność powrotu do wcześniejszych faz i ponowna sekwencyjna realizacja

Fazy testowania

- testowanie sytuacji normalnych
- testowanie sytuacji krańcowych
- testowanie wyjątków
- testowanie współdziałania programów – test systemu

Ocena niezawodności programu

- dane testowe i rzeczywiste
- testy obciążenia
- testy odporności
- testowanie bezpieczeństwa

Proces kompilacji programu

- postacie programu – źródłowa, wynikowa
- proces kompilacji – kompilatory
- translacja
- konsolidacja
- ładowanie
- interpretacja programu – interpretery

Interpretacja programu

Interpreter tłumaczy oraz od razu wykonuje każdą pojedynczą instrukcję