

# Informatyka 1

## Wykład II

### *Algorytm, podstawowe notacje, typy danych i wyrażenia*

*Robert Muszyński  
ZPCiR IIAiR PWr*

**Zagadnienia:** pojęcie algorytmu, diagramy algorytmów, przejście od algorytmu do programu, zapis składni programu, typy danych, operatory, wyrażenia, zmienne, instrukcje, struktura programu, instrukcja złożona, sekwencja instrukcji

Copyright © 2001–2005 Robert Muszyński

---

Niniejszy dokument zawiera materiały do wykładu na temat podstaw programowania w językach wysokiego poziomu. Jest on udostępniony pod warunkiem wykorzystania wyłącznie do własnych, prywatnych potrzeb i może być kopiowany wyłącznie w całości, razem ze stroną tytułową.

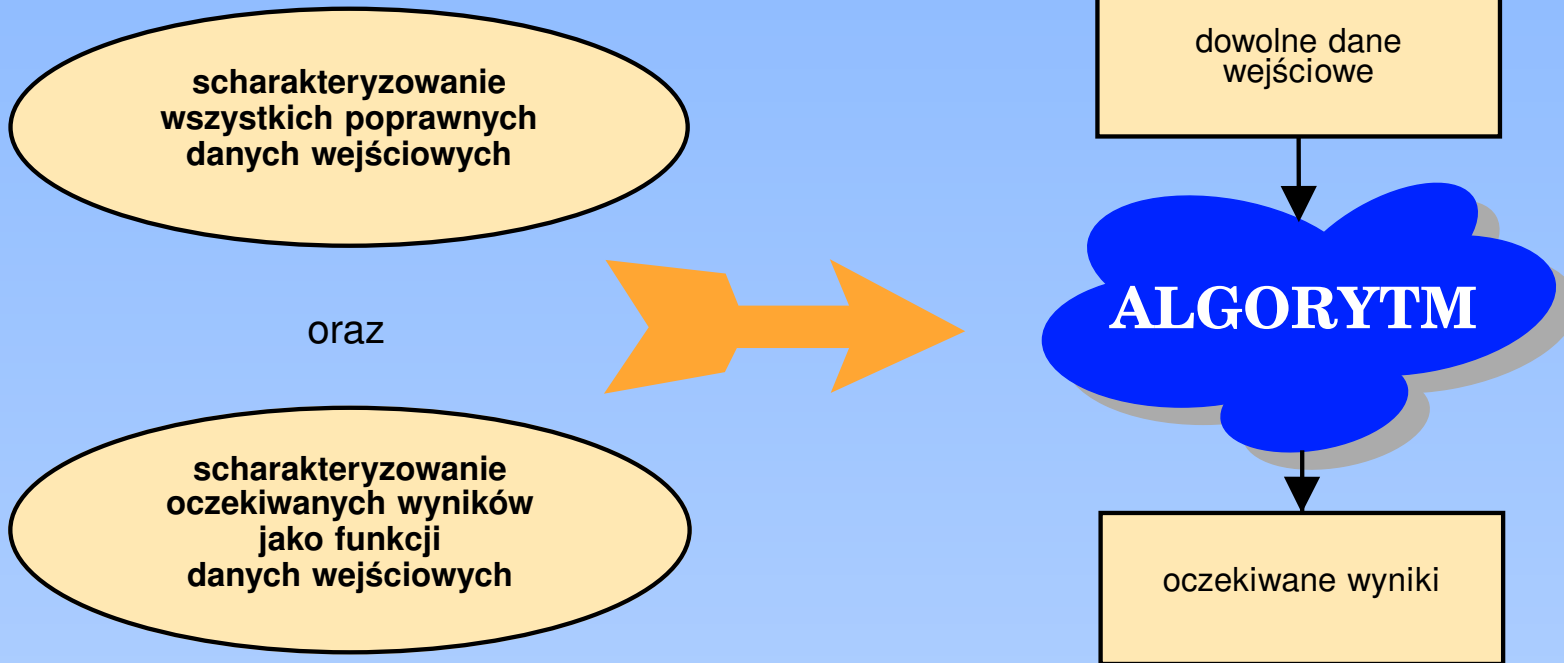
# Algorytm

**scharakteryzowanie  
wszystkich poprawnych  
danych wejściowych**

oraz

**scharakteryzowanie  
oczekiwanych wyników  
jako funkcji  
danych wejściowych**

# Algorytm



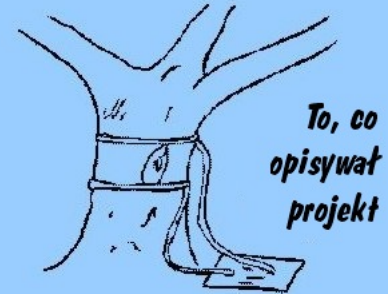
**Etapy budowy  
systemu  
informatycznego  
dla przedsiębiorstwa**



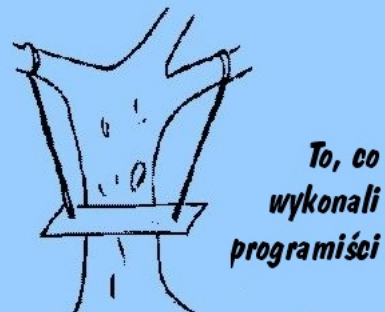
## **Etapy budowy systemu informatycznego dla przedsiębiorstwa**



## **Etapy budowy systemu informatycznego dla przedsiębiorstwa**



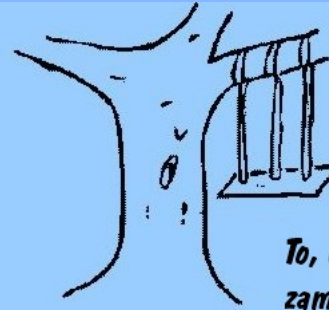
## **Etapy budowy systemu informatycznego dla przedsiębiorstwa**







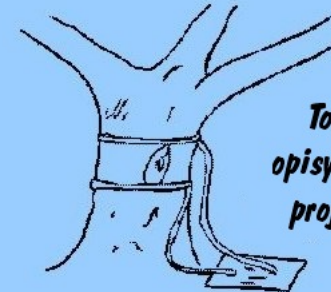
## Etapy budowy systemu informatycznego dla przedsiębiorstwa



*To, co klient zamówił*



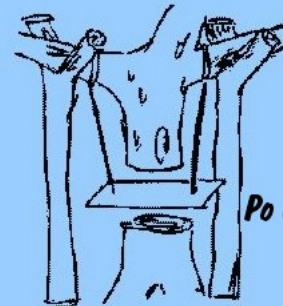
*To, co analityk zrozumiał*



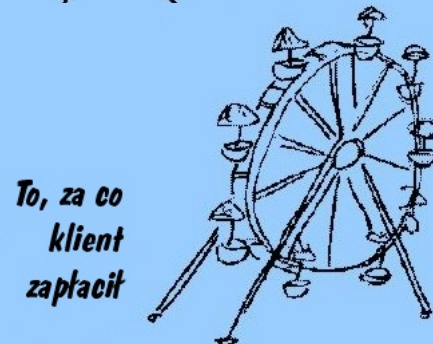
*To, co opisywał projekt*



*To, co wykonali programiści*

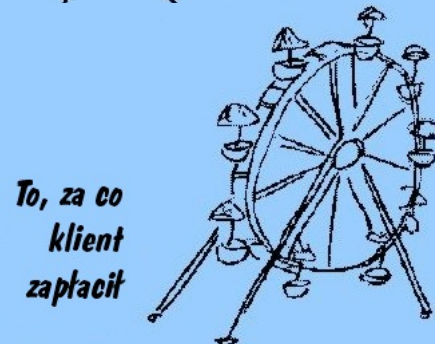
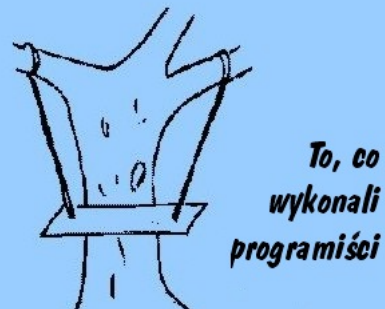


*Po uruchomieniu i wdrożeniu*



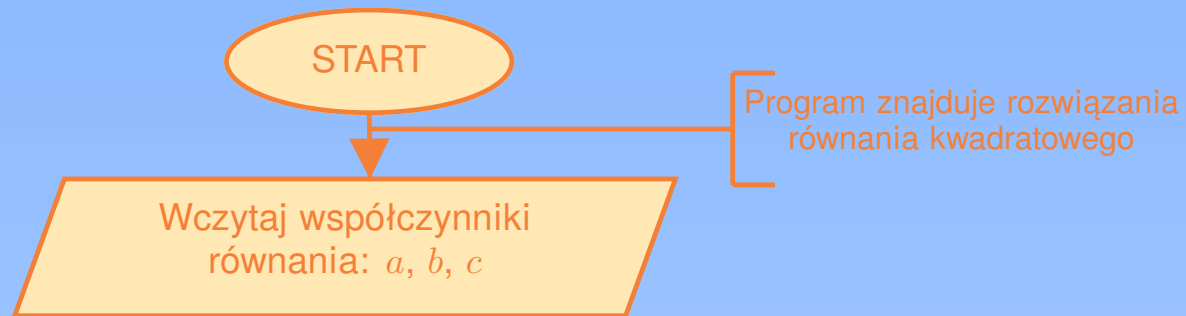
*To, za co klient zapłacił*

## Etapy budowy systemu informatycznego dla przedsiębiorstwa

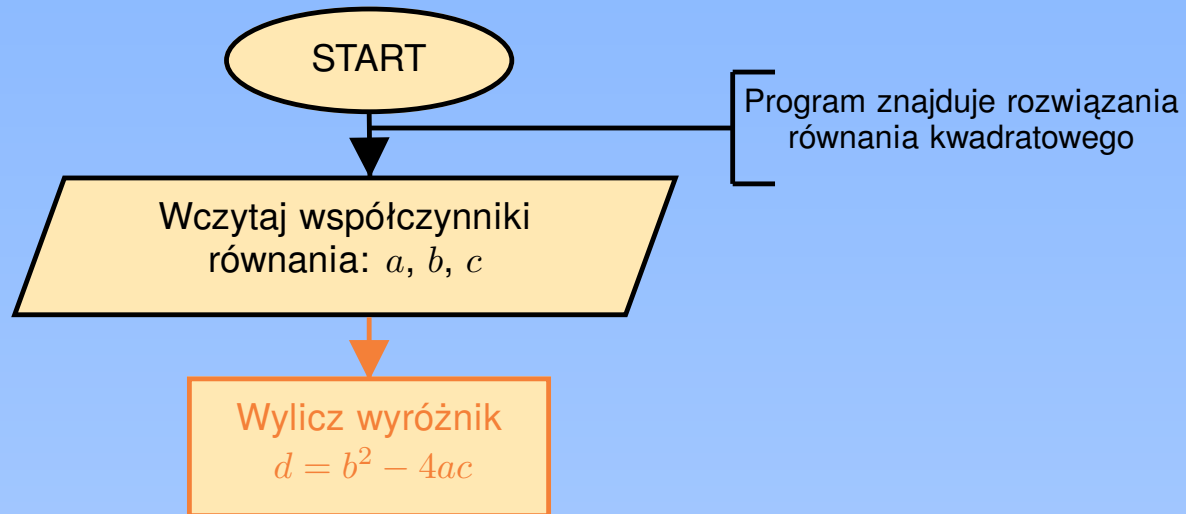


Autor nieznan

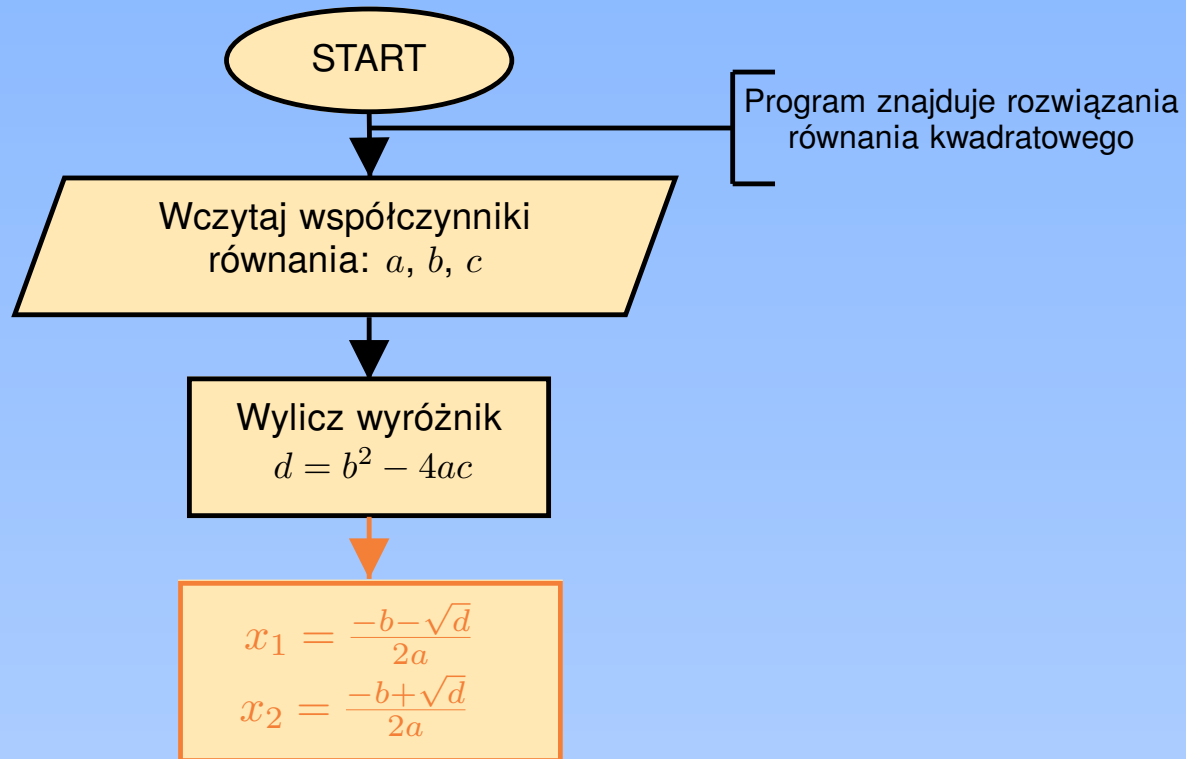
# Przykładowy algorytm



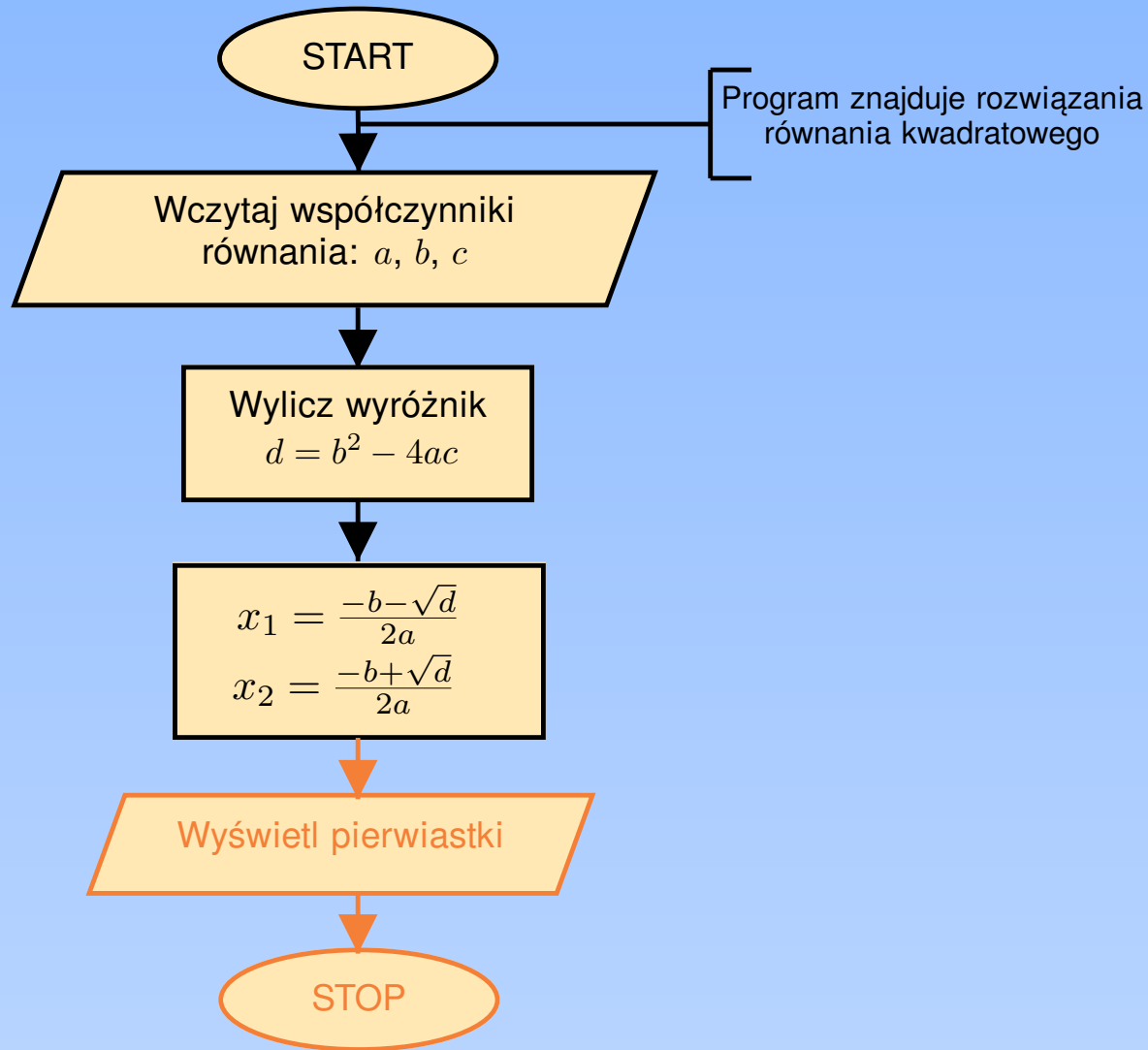
# Przykładowy algorytm



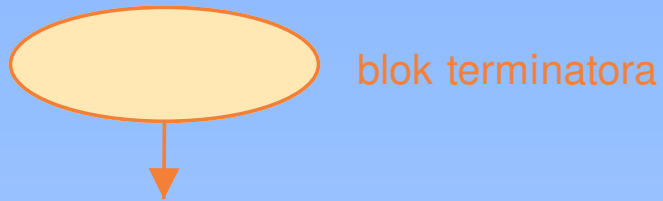
# Przykładowy algorytm



# Przykładowy algorytm

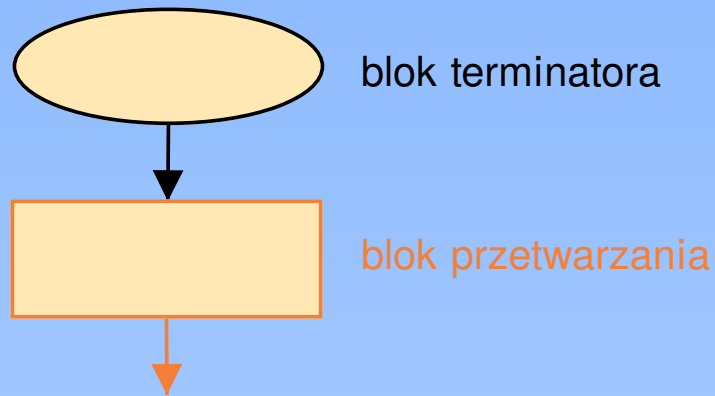


# Diagramy algorytmów



Elementy składowe schematów blokowych algorytmów.

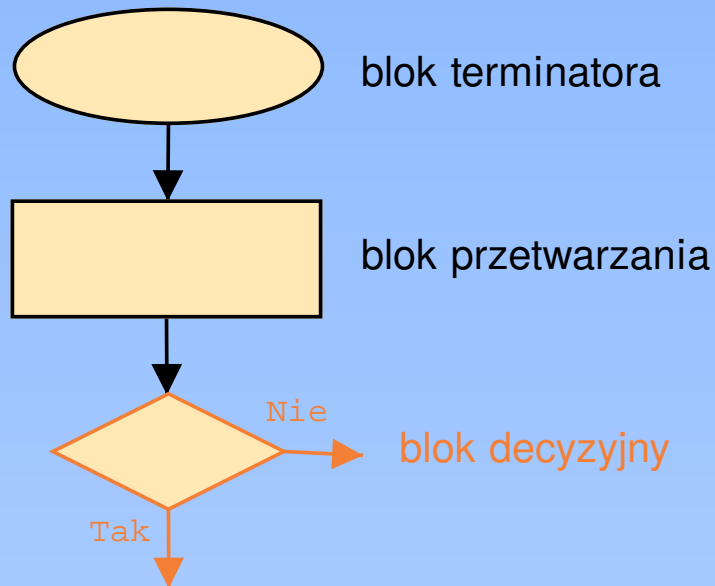
# Diagramy algorytmów



Elementy składowe schematów blokowych algorytmów.

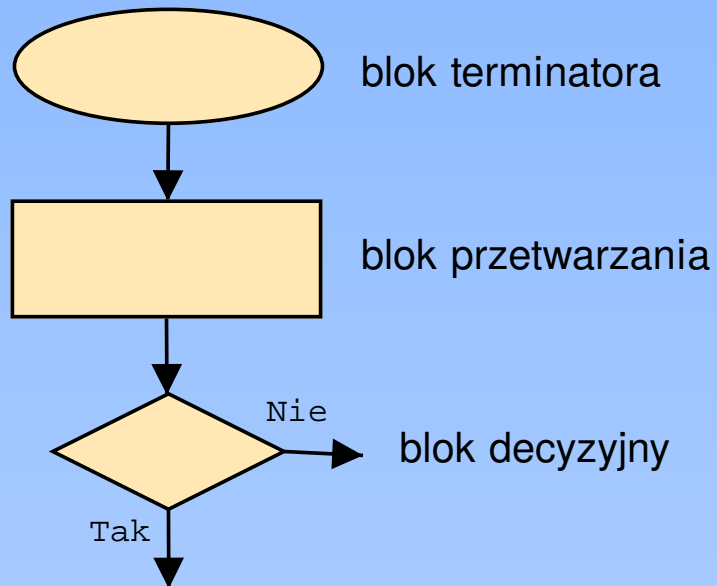


# Diagramy algorytmów



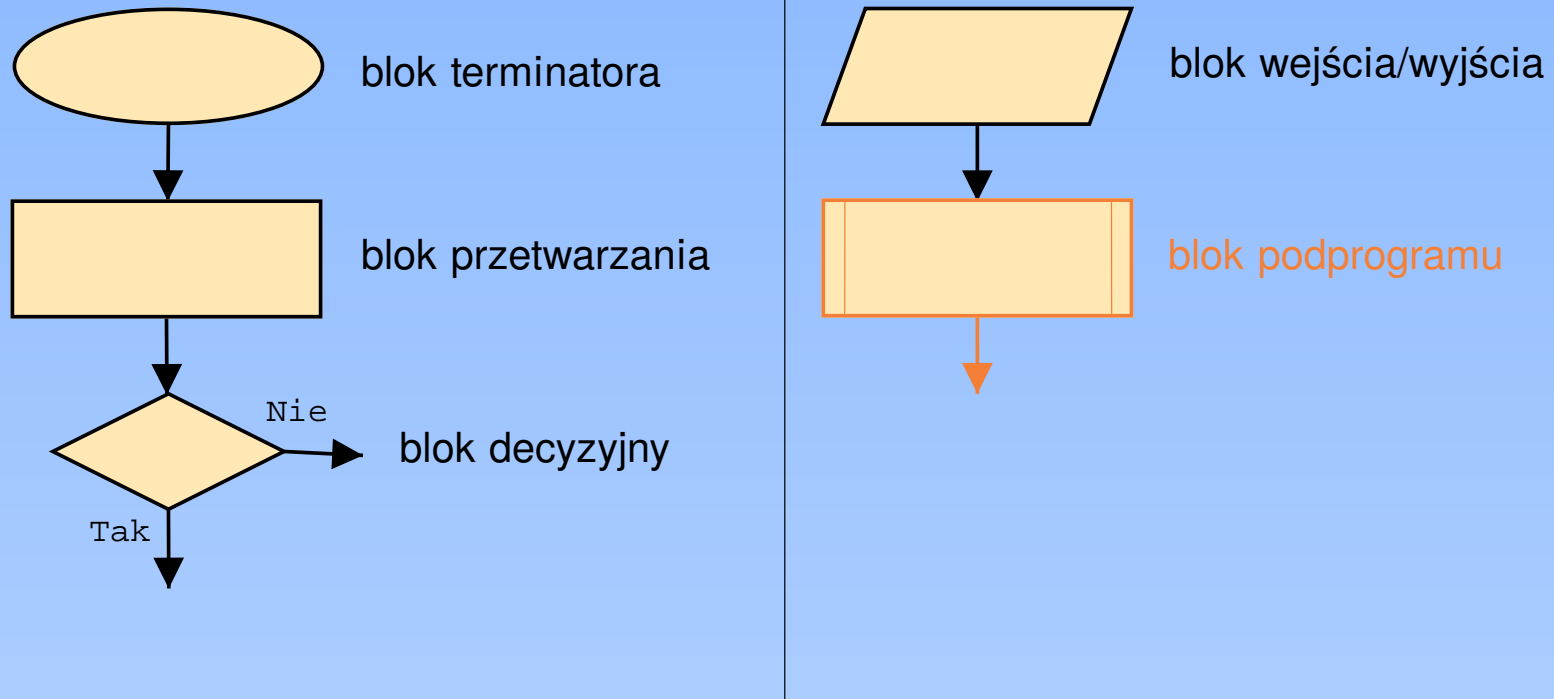
Elementy składowe schematów blokowych algorytmów.

# Diagramy algorytmów



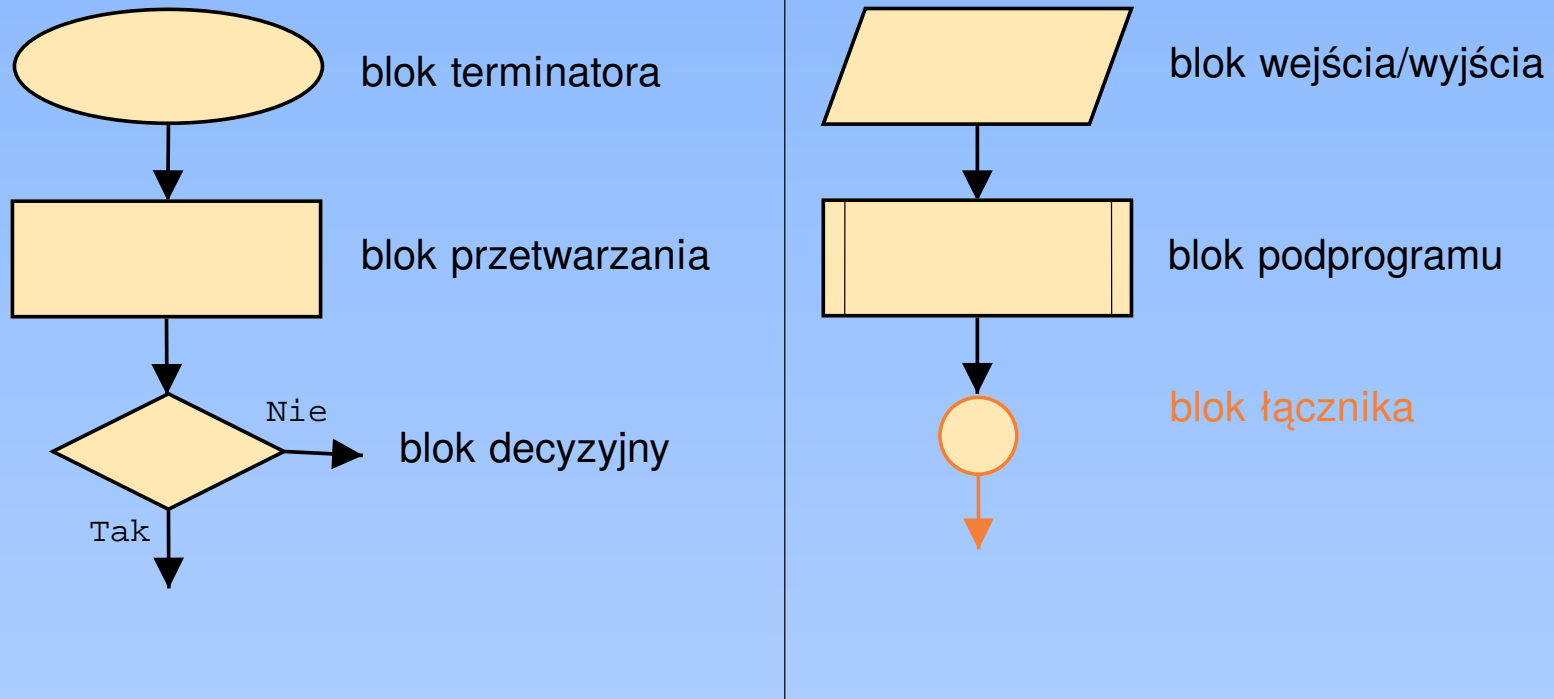
Elementy składowe schematów blokowych algorytmów.

# Diagramy algorytmów



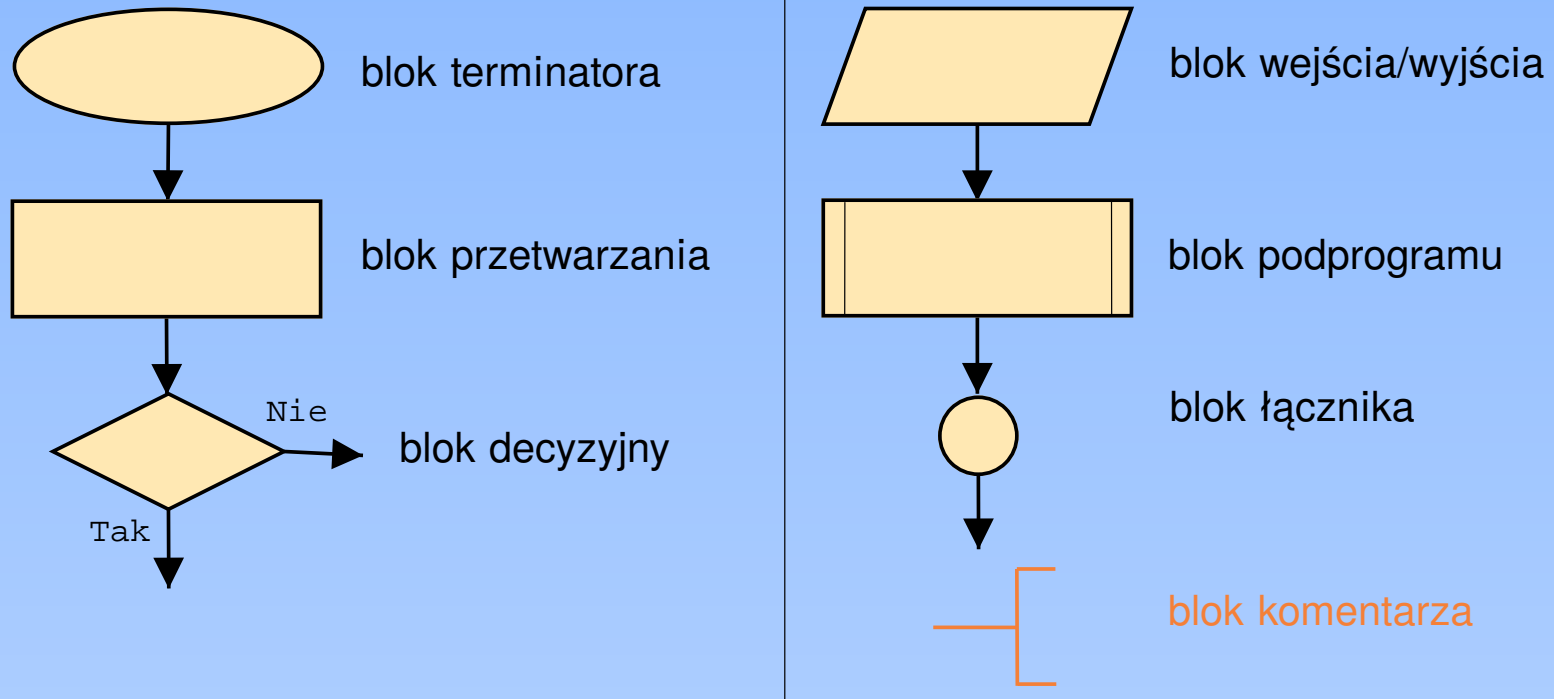
Elementy składowe schematów blokowych algorytmów.

# Diagramy algorytmów



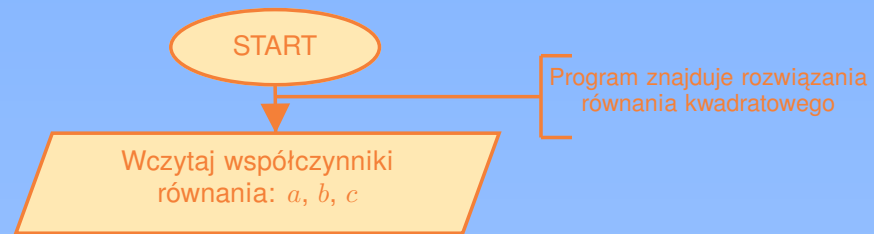
Elementy składowe schematów blokowych algorytmów.

# Diagramy algorytmów

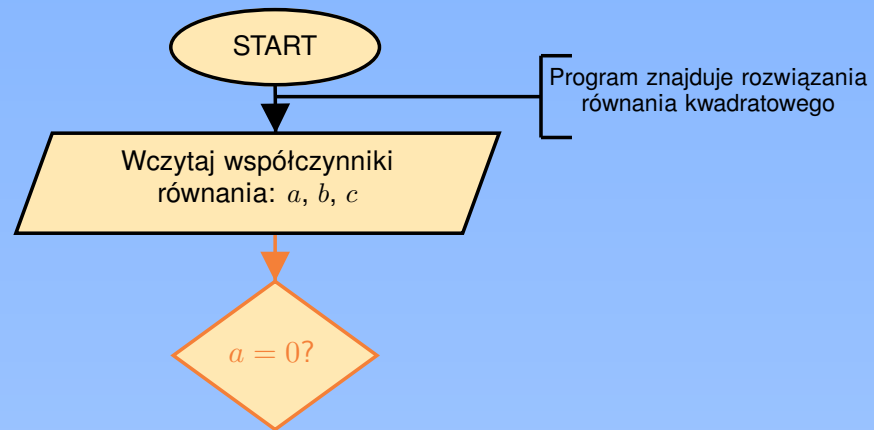


Elementy składowe schematów blokowych algorytmów.

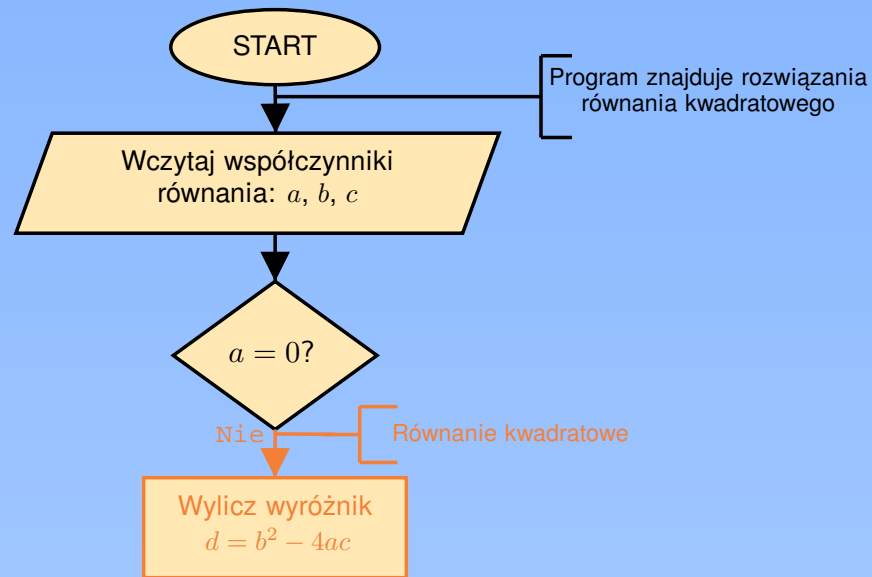
# Przykładowy algorytm — cd.



# Przykładowy algorytm — cd.

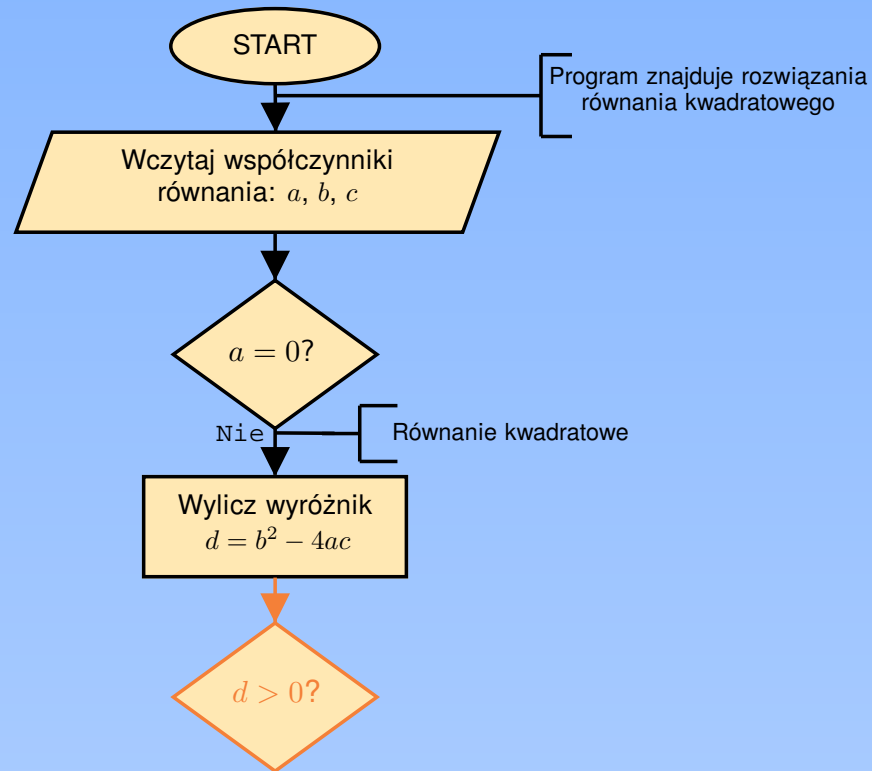


# Przykładowy algorytm — cd.

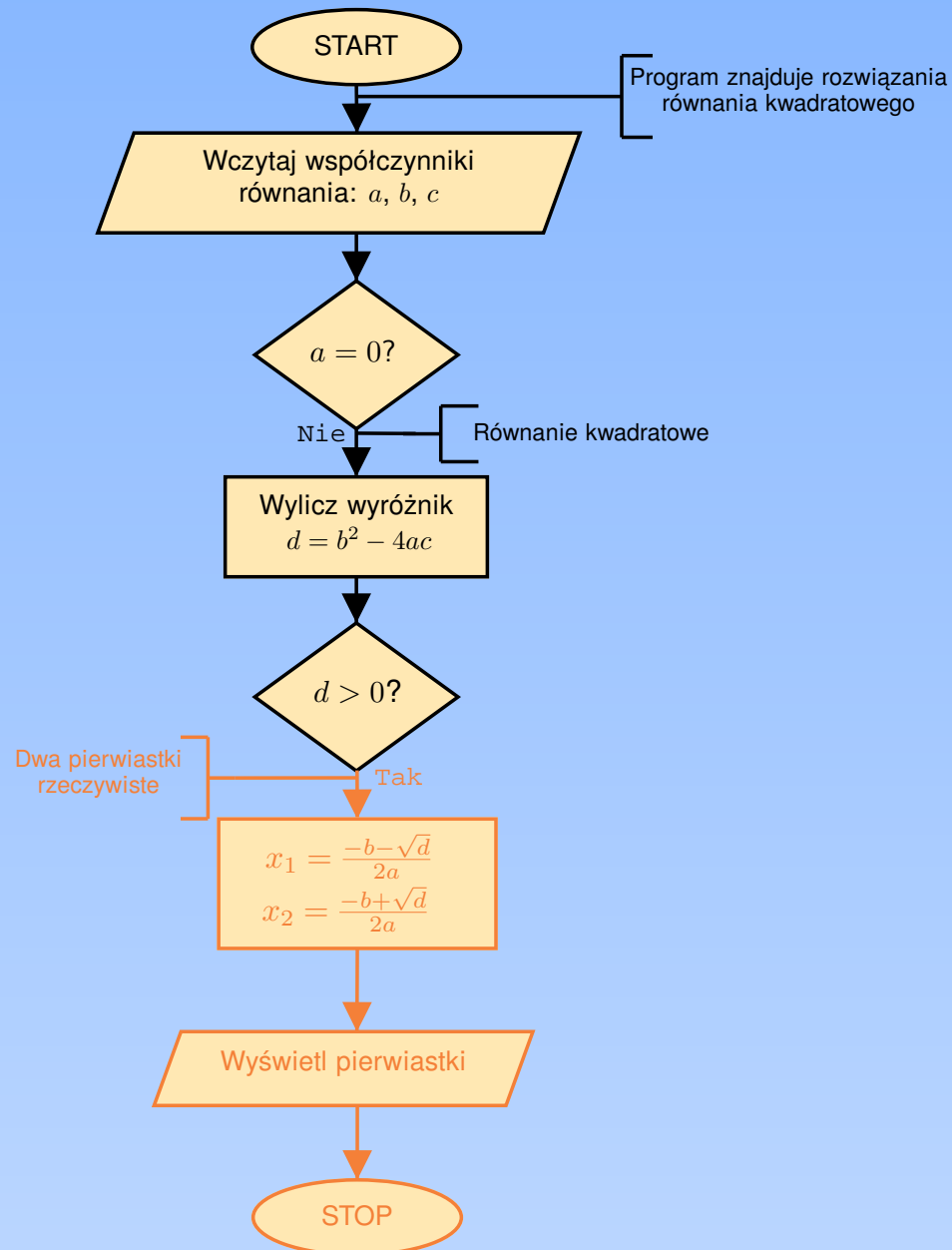




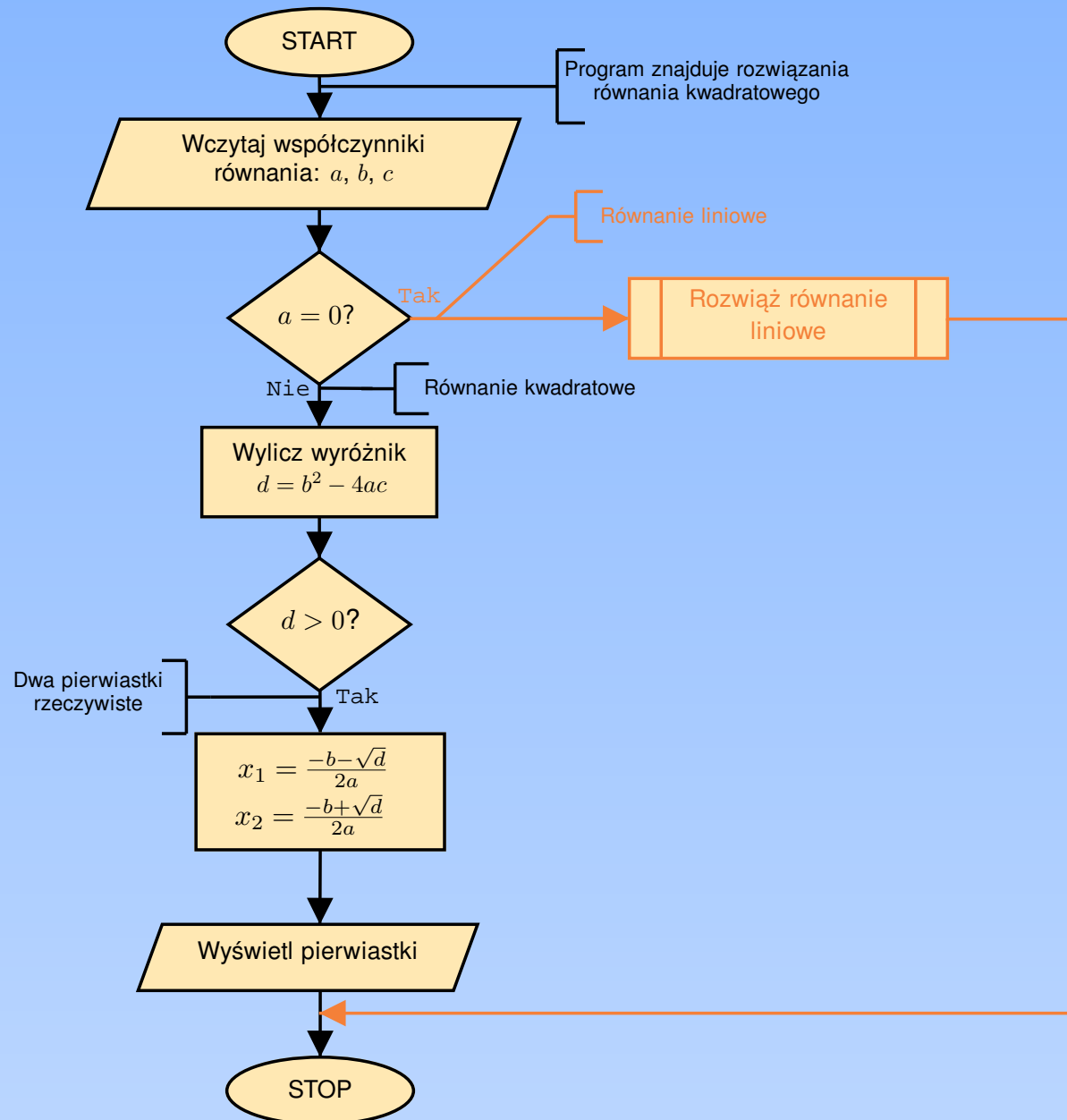
# Przykładowy algorytm — cd.



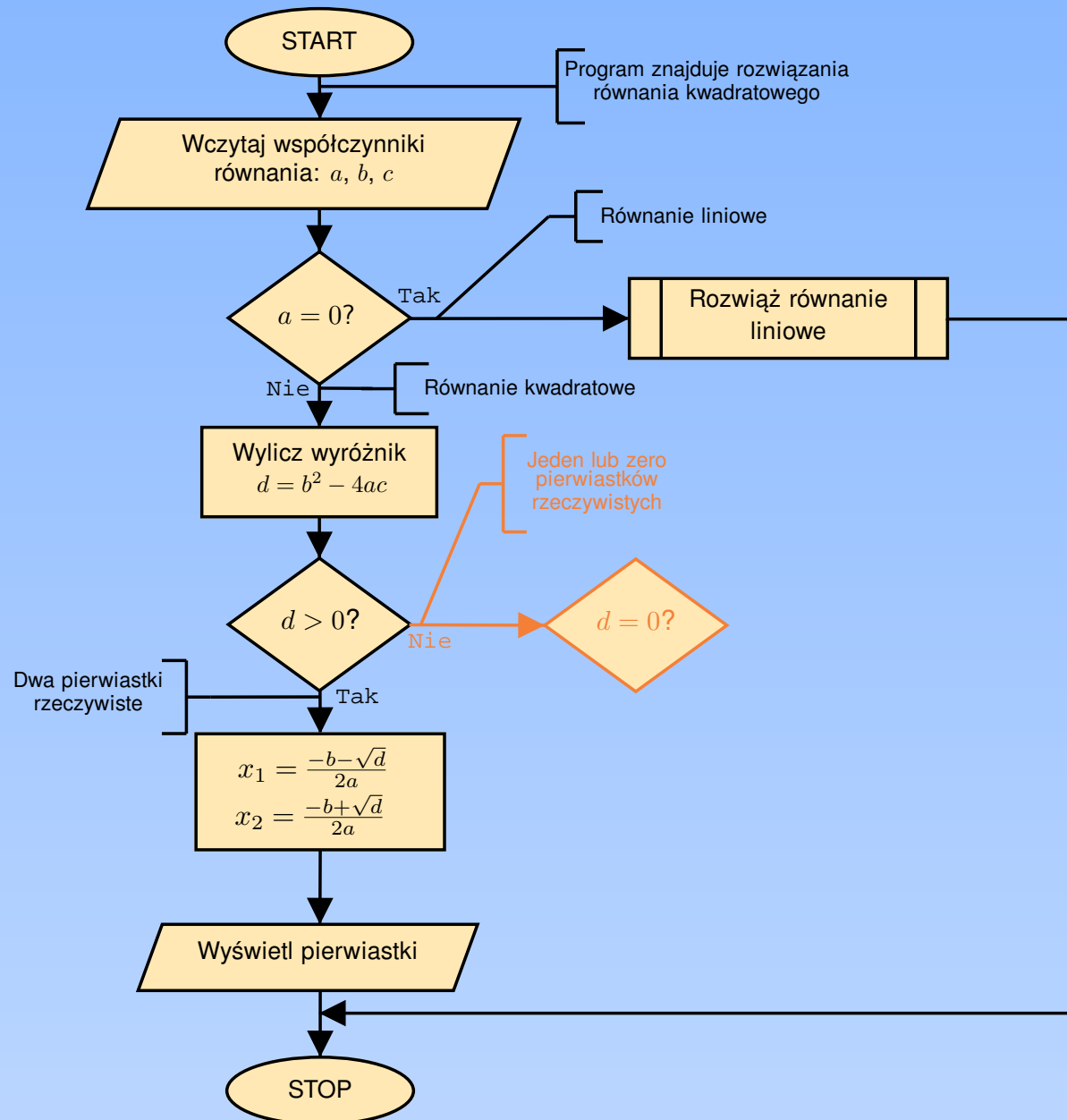
# Przykładowy algorytm — cd.



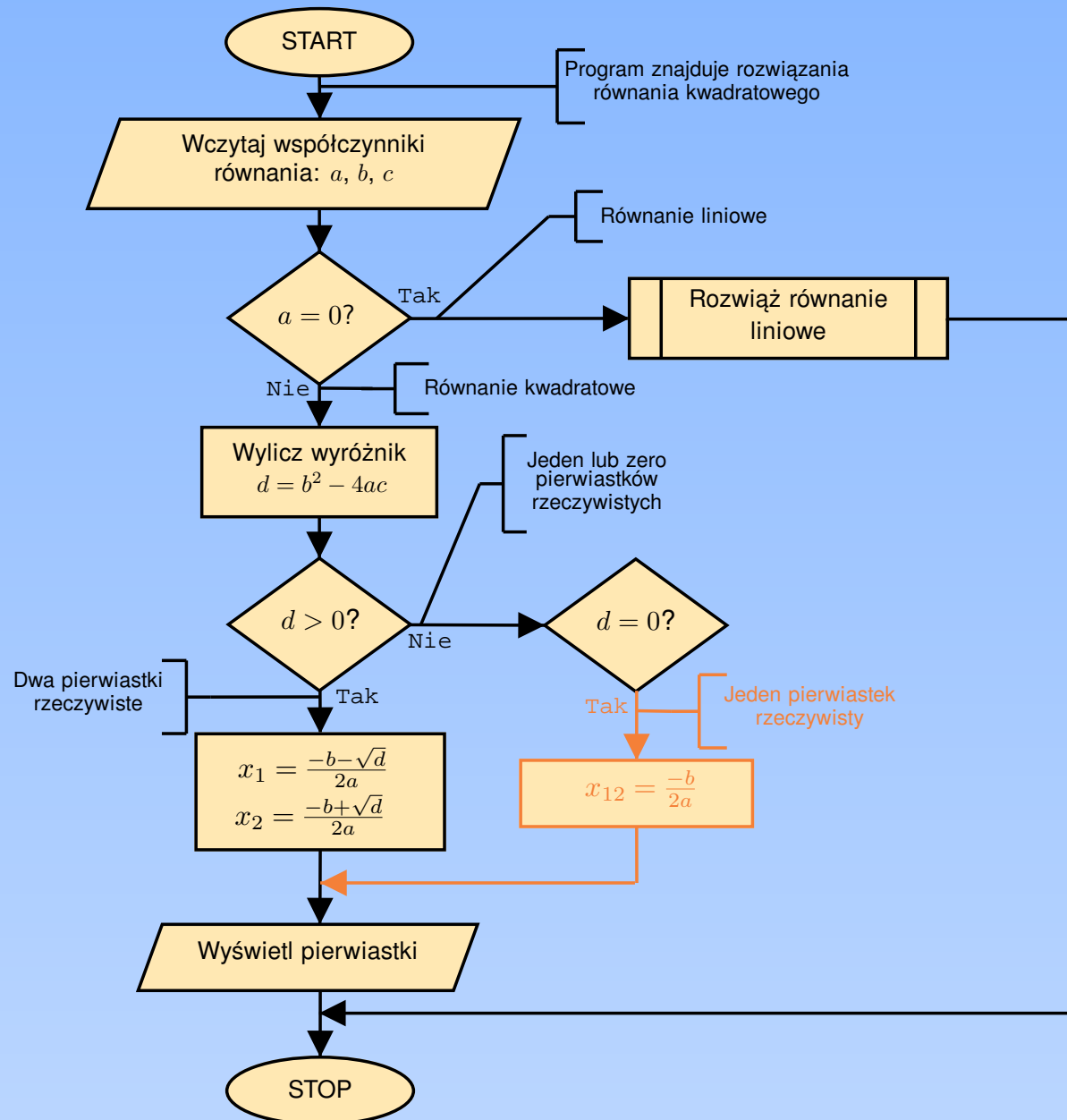
# Przykładowy algorytm — cd.



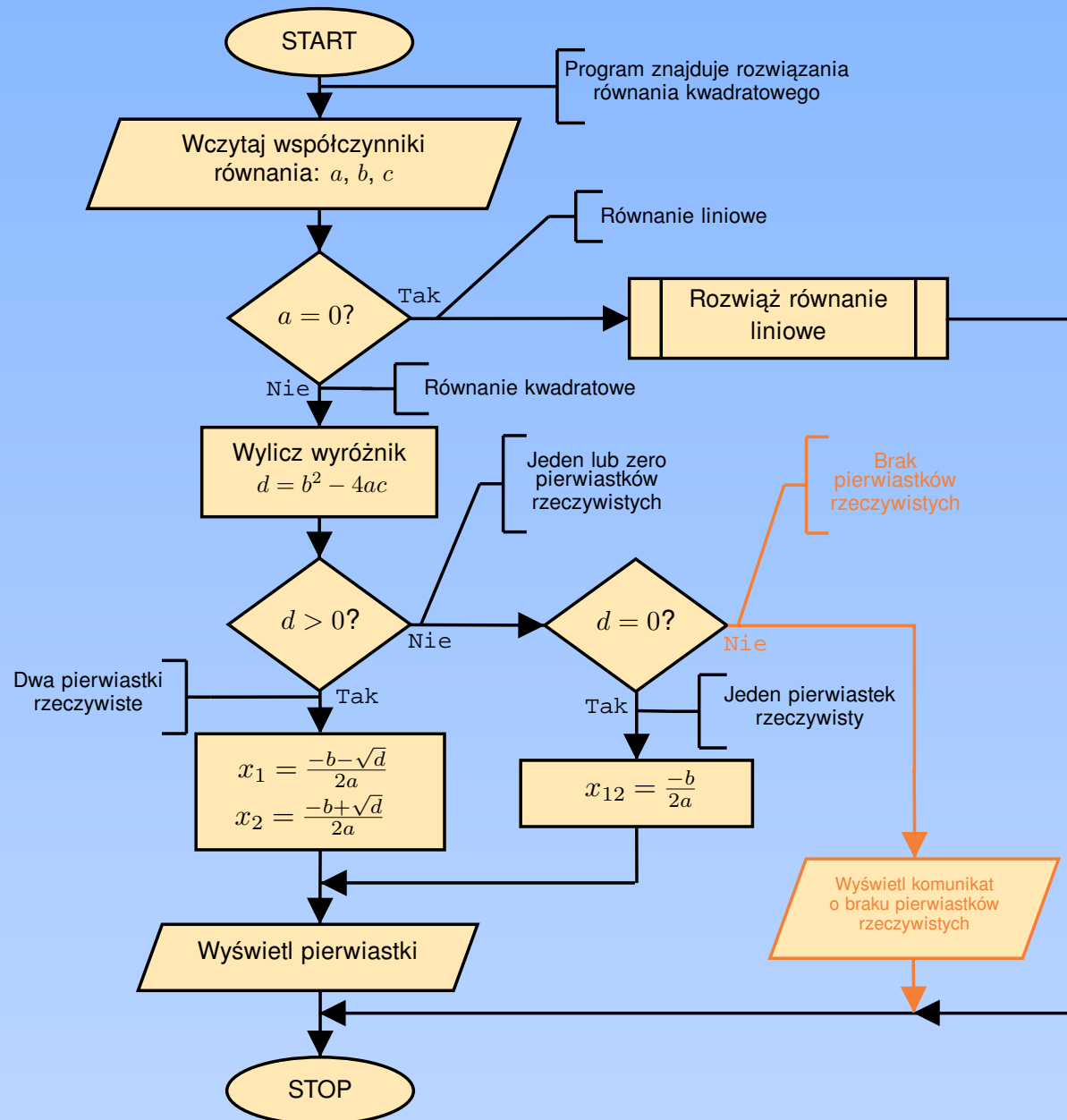
# Przykładowy algorytm — cd.



# Przykładowy algorytm — cd.



# Przykładowy algorytm — cd.



# Składnia programu

- **Notacja MBNF**

LHS = RHS

# Składnia programu

- **Notacja MBNF**

LHS = RHS

- ★ **Symbole nieterminalne:** zdania, grupy podmiotu;



# Składnia programu

- **Notacja MBNF**

$$\text{LHS} = \text{RHS}$$

- ★ Symbole nieterminalne: zdania, grupy podmiotu;
- ★ **symbole terminalne**: „bezbarwne”, „zielone”, „pomysły”, „śpią”, „wściekle”;

# Składnia programu

- **Notacja MBNF**

LHS = RHS

- ★ Symbole nieterminalne: zdania, grupy podmiotu;
- ★ symbole terminalne: „bezbarwne”, „zielone”, „pomysły”, „śpią”, „wściekle”;
- ★ **operatorzy**: konkatencja, alternatywa — |, opcja — [], powtórzenie — {}, grupowanie — ().

**Zdanie języka**

||

**Symbol początkowy**

+

**Reguły zastępowania**

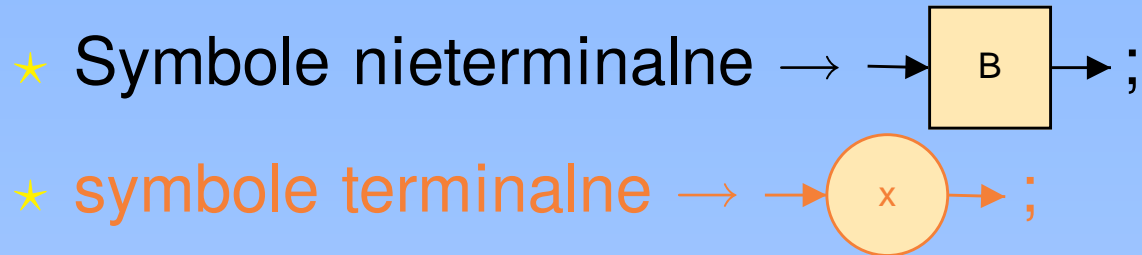
# Składnia programu

- **Diagramy składni**



# Składnia programu

- **Diagramy składni**



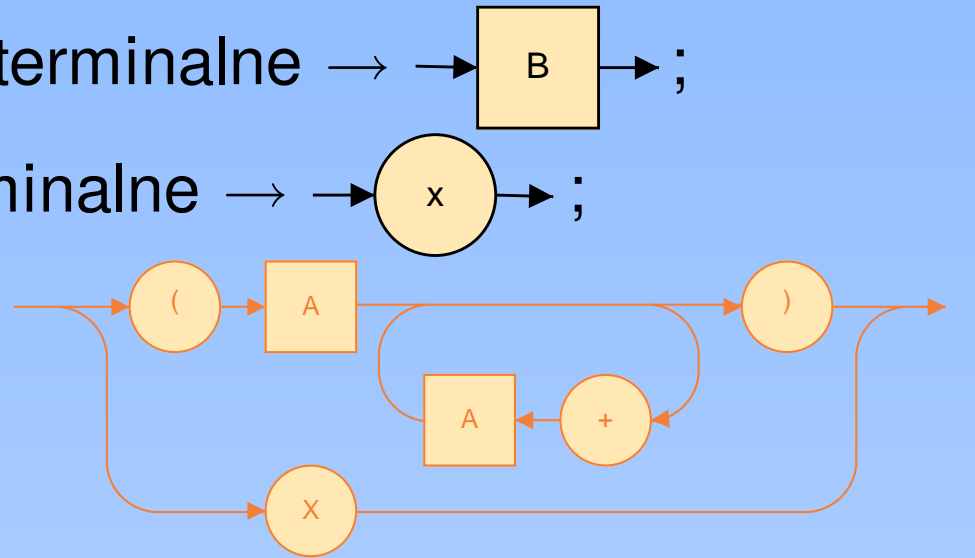
# Składnia programu

- **Diagramy składni**

- ★ Symbole nieterminalne → → **B** → ;

- ★ symbole terminalne → → **x** → ;

- ★ **operatory** →



# Przykładowe konstrukcje

- **Notacja MBNF**

```
liczba-calkowita = [ znak-liczby ]  
                  liczba-calkowita-bez-znaku.
```

# Przykładowe konstrukcje

- **Notacja MBNF**

```
liczba-calkowita = [ znak-liczby ]  
                  liczba-calkowita-bez-znaku.
```

```
znak-liczby = "+" | "-".
```



# Przykładowe konstrukcje

- **Notacja MBNF**

```
liczba-calkowita = [ znak-liczby ]  
                  liczba-calkowita-bez-znaku.  
znak-liczby = "+" | "-".  
liczba-calkowita-bez-znaku = ciag-cyfr.
```

# Przykładowe konstrukcje

- **Notacja MBNF**

```
liczba-calkowita = [ znak-liczby ]  
                  liczba-calkowita-bez-znaku.
```

```
znak-liczby = "+" | "-".
```

```
liczba-calkowita-bez-znaku = ciag-cyfr.
```

```
ciag-cyfr = cyfra { cyfra } .
```

# Przykładowe konstrukcje

- **Notacja MBNF**

```
liczba-calkowita = [ znak-liczby ]  
                  liczba-calkowita-bez-znaku.
```

```
znak-liczby = "+" | "-".
```

```
liczba-calkowita-bez-znaku = ciag-cyfr.
```

```
ciag-cyfr = cyfra { cyfra } .
```

```
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6"  
       | "7" | "8" | "9".
```

# Przykładowe konstrukcje

- **Notacja MBNF**

```
liczba-calkowita = [ znak-liczby ]  
                  liczba-calkowita-bez-znaku.
```

```
znak-liczby = "+" | "-".
```

```
liczba-calkowita-bez-znaku = ciag-cyfr.
```

```
ciag-cyfr = cyfra { cyfra } .
```

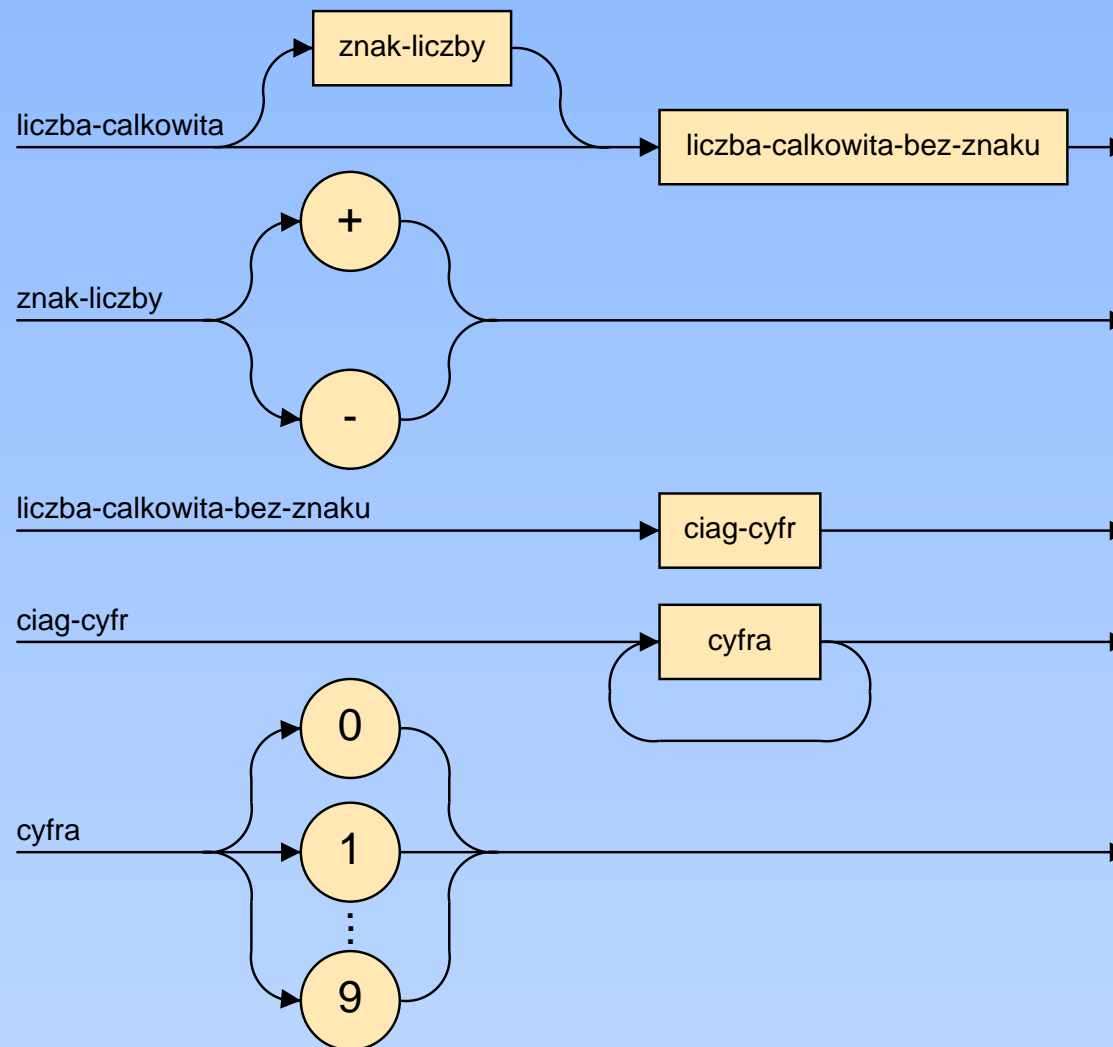
```
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6"  
       | "7" | "8" | "9".
```

Porównaj

```
ciag-cyfr = { cyfra }.
```

# Przykładowe konstrukcje

- Diagram składni



## Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |  
      dzien "-" miesiac "-" rok .
```

## Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |  
      dzien "-" miesiac "-" rok .  
dzien = cyfra | cyfra cyfra .
```

## Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |  
      dzien "-" miesiac "-" rok .  
dzien = cyfra | cyfra cyfra .  
miesiac = [ "1" ] cyfra .
```



## Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |  
      dzien "-" miesiac "-" rok .  
dzien = cyfra | cyfra cyfra .  
miesiac = [ "1" ] cyfra .  
rok = cyfra cyfra | cyfra cyfra cyfra cyfra .
```

## Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
      dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
miesiac = [ "1" ] cyfra .
rok = cyfra cyfra | cyfra cyfra cyfra cyfra .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" |
        "7" | "8" | "9" .
```

## Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
      dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
miesiac = [ "1" ] cyfra .
rok = cyfra cyfra | cyfra cyfra cyfra cyfra .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" |
       "7" | "8" | "9" .
```

Które z poniższych ciągów znaków spełniają definicję data? Jeśli nie spełniają to dlaczego?

12-12-12

## Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
      dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
miesiac = [ "1" ] cyfra .
rok = cyfra cyfra | cyfra cyfra cyfra cyfra .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" |
       "7" | "8" | "9" .
```

Które z poniższych ciągów znaków spełniają definicję data? Jeśli nie spełniają to dlaczego?

12-12-12

3.10.67

## Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
      dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
miesiac = [ "1" ] cyfra .
rok = cyfra cyfra | cyfra cyfra cyfra cyfra .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" |
       "7" | "8" | "9" .
```

Które z poniższych ciągów znaków spełniają definicję data? Jeśli nie spełniają to dlaczego?

12-12-12

3.10.67

0-0-00

## Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
      dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
miesiac = [ "1" ] cyfra .
rok = cyfra cyfra | cyfra cyfra cyfra cyfra .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" |
       "7" | "8" | "9" .
```

Które z poniższych ciągów znaków spełniają definicję data? Jeśli nie spełniają to dlaczego?

12-12-12

00-00-00

3.10.67

0-0-00

## Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
      dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
miesiac = [ "1" ] cyfra .
rok = cyfra cyfra | cyfra cyfra cyfra cyfra .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" |
       "7" | "8" | "9" .
```

Które z poniższych ciągów znaków spełniają definicję data? Jeśli nie spełniają to dlaczego?

12-12-12

00-00-00

3.10.67

47-19-99

0-0-00

## Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
      dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
miesiac = [ "1" ] cyfra .
rok = cyfra cyfra | cyfra cyfra cyfra cyfra .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" |
       "7" | "8" | "9" .
```

Które z poniższych ciągów znaków spełniają definicję data? Jeśli nie spełniają to dlaczego?

12-12-12

00-00-00

3.10.67

47-19-99

0-0-00

12-2-997



## Notacja MBNF — dalsze przykłady

`ciag = "A" [ ciag ] "A" | "B" [ ciag ] "B" | "A" | "B" .`

## Notacja MBNF — dalsze przykłady

ciag = "A" [ ciag ] "A" | "B" [ ciag ] "B" | "A" | "B" .

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

ABBA

## Notacja MBNF — dalsze przykłady

ciag = "A" [ ciag ] "A" | "B" [ ciag ] "B" | "A" | "B" .

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

ABBA

BABA

## Notacja MBNF — dalsze przykłady

ciag = "A" [ ciag ] "A" | "B" [ ciag ] "B" | "A" | "B" .

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

ABBA

BABA

ABABA

## Notacja MBNF — dalsze przykłady

ciag = "A" [ ciag ] "A" | "B" [ ciag ] "B" | "A" | "B" .

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

ABBA

BABA

ABABA

AAABAAA

## Notacja MBNF — dalsze przykłady

ciąg = "A" [ ciąg ] "A" | "B" [ ciąg ] "B" | "A" | "B" .

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

ABBA

BABA

ABABA

AAABAAA

ABABA

## Notacja MBNF — dalsze przykłady

ciag = "A" [ ciag ] "A" | "B" [ ciag ] "B" | "A" | "B" .

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

ABBA

BABA

ABABA

AAABAAA

ABABA

AAABBAABBABBABAABBAAA

## Notacja MBNF — dalsze przykłady

jedzonko = ( ( bc ) | rmw | bcrmw )  
( p | h [ k ] ) [ d ] .



## Notacja MBNF — dalsze przykłady

jedzonko = ( ( bc ) | rmw | bcrmw )  
( p | h [ k ] ) [ d ] .

b = "bigos". c = "chleb". r = "rogalik". m = "masło".  
w = "wedlina". p = "piwo". h = "herbata".  
k = "cukier". d = "deser".

## Notacja MBNF — dalsze przykłady

jedzonko = ( ( bc ) | rmw | bcrmw )  
( p | h [ k ] ) [ d ] .

b = "bigos". c = "chleb". r = "rogalik". m = "masło".

w = "wedlina". p = "piwo". h = "herbata".

k = "cukier". d = "deser".

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

p

## Notacja MBNF — dalsze przykłady

jedzonko = ( ( bc ) | rmw | bcrmw )  
( p | h [ k ] ) [ d ] .

b = "bigos". c = "chleb". r = "rogalik". m = "masło".

w = "wedlina". p = "piwo". h = "herbata".

k = "cukier". d = "deser".

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

p

pd





## Notacja MBNF — dalsze przykłady

jedzonko = ( ( bc ) | rmw | bcrmw )  
( p | h [ k ] ) [ d ] .

b = "bigos". c = "chleb". r = "rogalik". m = "masło".

w = "wedlina". p = "piwo". h = "herbata".

k = "cukier". d = "deser".

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

p

pd

brmpd

rmwvk

bcrmwk

## Notacja MBNF — dalsze przykłady

```
jedzonko = ( ( bc ) | rmw | bcrmw )
           ( p | h [ k ] ) [ d ] .
```

```
b = "bigos". c = "chleb". r = "rogalik". m = "maslo".
```

```
w = "wedlina". p = "piwo". h = "herbata".
```

```
k = "cukier". d = "deser".
```

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

p

pd

brmpd

rmwpk

bcrmw h

bcp

## Notacja MBNF — dalsze przykłady

`liczba = [ znak-liczby ] liczba-bez-znaku.`



## Notacja MBNF — dalsze przykłady

`liczba = [ znak-liczby ] liczba-bez-znaku.`

`znak-liczby = "+" | "-".`

## Notacja MBNF — dalsze przykłady

`liczba = [ znak-liczby ] liczba-bez-znaku.`

`znak-liczby = "+" | "-".`

`liczba-bez-znaku = cyfra { cyfra }.`

## Notacja MBNF — dalsze przykłady

`liczba = [ znak-liczby ] liczba-bez-znaku.`

`znak-liczby = "+" | "-".`

`liczba-bez-znaku = cyfra { cyfra }.`

`cyfra = "0" | "1" | "2" | "3" | "4" | "5".`

## Notacja MBNF — dalsze przykłady

`liczba = [ znak-liczby ] liczba-bez-znaku.`

`znak-liczby = "+" | "-".`

`liczba-bez-znaku = cyfra { cyfra }.`

`cyfra = "0" | "1" | "2" | "3" | "4" | "5".`

Czy w sensie powyższych reguł poprawna jest liczba?

13

## Notacja MBNF — dalsze przykłady

liczba = [ znak-liczby ] liczba-bez-znaku.

znak-liczby = "+" | "-".

liczba-bez-znaku = cyfra { cyfra }.

cyfra = "0" | "1" | "2" | "3" | "4" | "5".

Czy w sensie powyższych reguł poprawna jest liczba?

13      +13

## Notacja MBNF — dalsze przykłady

liczba = [ znak-liczby ] liczba-bez-znaku.

znak-liczby = "+" | "-".

liczba-bez-znaku = cyfra { cyfra }.

cyfra = "0" | "1" | "2" | "3" | "4" | "5".

Czy w sensie powyższych reguł poprawna jest liczba?

13      +13      +7

## Notacja MBNF — dalsze przykłady

liczba = [ znak-liczby ] liczba-bez-znaku.

znak-liczby = "+" | "-".

liczba-bez-znaku = cyfra { cyfra }.

cyfra = "0" | "1" | "2" | "3" | "4" | "5".

Czy w sensie powyższych reguł poprawna jest liczba?

13      +13      +7      -666

## Notacja MBNF — dalsze przykłady

```
piwo = "Piast" | "EB" | "Lech" | "Zywiec" | "Okocim" .  
menu = piwo { piwo } .
```



## Notacja MBNF — dalsze przykłady

```
piwo = "Piast" | "EB" | "Lech" | "Zywiec" | "Okocim" .  
menu = piwo { piwo } .
```

Zamówienie: Dwa piwa prosze

## Notacja MBNF — dalsze przykłady

```
piwo = "Piast" | "EB" | "Lech" | "Zywiec" | "Okocim" .  
menu = piwo { piwo } .
```

Zamówienie: Dwa piwa prosze

```
zamowienie = piwo piwo .
```

## Notacja MBNF — dalsze przykłady

```
piwo = "Piast" | "EB" | "Lech" | "Zywiec" | "Okocim" .  
menu = piwo { piwo } .
```

Zamówienie: Dwa piwa prosze  
zamowienie = piwo piwo .

Zamówienie: Obojętne co i czy w ogóle, ale jeśli cokolwiek to jedno

## Notacja MBNF — dalsze przykłady

```
piwo = "Piast" | "EB" | "Lech" | "Zywiec" | "Okocim" .  
menu = piwo { piwo } .
```

Zamówienie: Dwa piwa prosze  
zamowienie = piwo piwo .

Zamówienie: Obojętne co i czy w ogóle, ale jeśli cokolwiek to jedno  
zamowienie = [ piwo ] .

## Notacja MBNF — dalsze przykłady

```
piwo = "Piast" | "EB" | "Lech" | "Zywiec" | "Okocim" .  
menu = piwo { piwo } .
```

Zamówienie: Dwa piwa prosze  
zamowienie = piwo piwo .

Zamówienie: Obojętne co i czy w ogóle, ale jeśli cokolwiek to jedno  
zamowienie = [ piwo ] .

Zamówienie: Dwa Żywce lub EB i Lecha prosze

## Notacja MBNF — dalsze przykłady

```
piwo = "Piast" | "EB" | "Lech" | "Zywiec" | "Okocim" .  
menu = piwo { piwo } .
```

Zamówienie: Dwa piwa prosze  
zamowienie = piwo piwo .

Zamówienie: Obojętne co i czy w ogóle, ale jeśli cokolwiek to jedno  
zamowienie = [ piwo ] .

Zamówienie: Dwa Żywce lub EB i Lecha prosze  
zamowienie = [ "Zywiec" "Zywiec" ] | ( "EB" "Lech" ) .

## Notacja MBNF — dalsze przykłady

Czy poniższe konstrukcje definiują liczby całkowite?

`liczba-cal-kowita = [ znak-liczby ]`

`liczba-cal-kowita-bez-znaku.`

`znak-liczby = "+" | "-".`

`liczba-cal-kowita-bez-znaku = ciag-cyfr.`

`ciag-cyfr = cyfra { cyfra } .`

`cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6"  
| "7" | "8" | "9".`

## Notacja MBNF — dalsze przykłady

Czy poniższe konstrukcje definiują liczby całkowite?

`liczba-cal-kowita = [ znak-liczby ]`

`liczba-cal-kowita-bez-znaku.`

`znak-liczby = "+" | "-".`

`liczba-cal-kowita-bez-znaku = ciag-cyfr.`

`ciag-cyfr = cyfra { cyfra } .`

`cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6"  
| "7" | "8" | "9".`

`konstrukcja = cyfra .`



## Notacja MBNF — dalsze przykłady

Czy poniższe konstrukcje definiują liczby całkowite?

`liczba-cal-kowita = [ znak-liczby ]`

`liczba-cal-kowita-bez-znaku.`

`znak-liczby = "+" | "-".`

`liczba-cal-kowita-bez-znaku = ciag-cyfr.`

`ciag-cyfr = cyfra { cyfra } .`

`cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6"  
| "7" | "8" | "9".`

`konstrukcja = cyfra .`

`konstrukcja = "+" cyfra cyfra cyfra .`

## Notacja MBNF — dalsze przykłady

Czy poniższe konstrukcje definiują liczby całkowite?

`liczba-cal-kowita = [ znak-liczby ]`

`liczba-cal-kowita-bez-znaku.`

`znak-liczby = "+" | "-".`

`liczba-cal-kowita-bez-znaku = ciag-cyfr.`

`ciag-cyfr = cyfra { cyfra } .`

`cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6"  
| "7" | "8" | "9".`

`konstrukcja = cyfra .`

`konstrukcja = "+" cyfra cyfra cyfra .`

`konstrukcja = [ "-" | "+" ] cyfra { cyfra } .`

## Notacja MBNF — dalsze przykłady

Czy poniższe konstrukcje definiują liczby całkowite?

`liczba-cal-kowita = [ znak-liczby ]`

`liczba-cal-kowita-bez-znaku.`

`znak-liczby = "+" | "-".`

`liczba-cal-kowita-bez-znaku = ciag-cyfr.`

`ciag-cyfr = cyfra { cyfra } .`

`cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6"  
| "7" | "8" | "9".`

`konstrukcja = cyfra .`

`konstrukcja = "+" cyfra cyfra cyfra .`

`konstrukcja = [ "-" | "+" ] cyfra { cyfra } .`

`konstrukcja = [ "-" | "+" ] { cyfra } .`

## Notacja MBNF — dalsze przykłady

Czy poniższe konstrukcje definiują liczby całkowite?

`liczba-cal-kowita = [ znak-liczby ]`

`liczba-cal-kowita-bez-znaku.`

`znak-liczby = "+" | "-".`

`liczba-cal-kowita-bez-znaku = ciag-cyfr.`

`ciag-cyfr = cyfra { cyfra } .`

`cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6"  
| "7" | "8" | "9".`

`konstrukcja = cyfra .`

`konstrukcja = "+" cyfra cyfra cyfra .`

`konstrukcja = [ "-" | "+" ] cyfra { cyfra } .`

`konstrukcja = [ "-" | "+" ] { cyfra } .`

`konstrukcja = [ ( "+" | "-" ) [ cyfra [ cyfra ] ] ] .`

## Notacja MBNF — równoważność reguł

Czy poniższe pary reguł są sobie równoważne?

(1)  $\text{cyfra} = "0" \mid "1" \mid "2" \mid "3" \mid "4" \mid "5" \mid "6" .$

(2)  $\text{cyfra} = \text{mala-cyfra} \mid \text{duza-cyfra} .$

$\text{mala-cyfra} = "0" \mid "1" \mid "2" \mid "3" \mid "4" .$

$\text{duza-cyfra} = "3" \mid "4" \mid "5" \mid "6" .$

## Notacja MBNF — równoważność reguł

Czy poniższe pary reguł są sobie równoważne?

(1) `cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" .`

(2) `cyfra = mala-cyfra | duza-cyfra .`

`mala-cyfra = "0" | "1" | "2" | "3" | "4" .`

`duza-cyfra = "3" | "4" | "5" | "6" .`

---

(1) `czas = godzina [ ":" minuta ] [ ":" sekunda ] .`

(2) `czas = godzina [ ":" minuta [ ":" sekunda ] ] .`

## Notacja MBNF — równoważność reguł

Czy poniższe pary reguł są sobie równoważne?

(1) `cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" .`

(2) `cyfra = mala-cyfra | duza-cyfra .`

`mala-cyfra = "0" | "1" | "2" | "3" | "4" .`

`duza-cyfra = "3" | "4" | "5" | "6" .`

---

(1) `czas = godzina [ ":" minuta ] [ ":" sekunda ] .`

(2) `czas = godzina [ ":" minuta [ ":" sekunda ] ] .`

---

(1) `ciag-cyfr = { cyfra } .`

(2) `ciag-cyfr = "" | cyfra ciag-cyfr .`

## Notacja MBNF — równoważność reguł

Czy poniższe pary reguł są sobie równoważne?

(1) liczba = cyfra cyfra { cyfra cyfra } .

(2) liczba = cyfra cyfra .  
liczba = liczba liczba .



## Notacja MBNF — równoważność reguł

Czy poniższe pary reguł są sobie równoważne?

(1)  $\text{liczba} = \text{cyfra cyfra} \{ \text{cyfra cyfra} \} .$

(2)  $\text{liczba} = \text{cyfra cyfra} .$   
 $\text{liczba} = \text{liczba liczba} .$

---

(1)  $\text{nr-telefonu} = \text{cyf cyf "-" cyf cyf "-" cyf cyf} .$

(2)  $\text{nr-telefonu} = \text{cyf cyf cyf "-" cyf cyf cyf} .$

## Notacja MBNF — równoważność reguł

Czy poniższe pary reguł są sobie równoważne?

(1) liczba = cyfra cyfra { cyfra cyfra } .

(2) liczba = cyfra cyfra .  
liczba = liczba liczba .

---

(1) nr-telefonu = cyf cyf "-" cyf cyf "-" cyf cyf .

(2) nr-telefonu = cyf cyf cyf "-" cyf cyf cyf .

---

(1) w1 = cyfra { cyfra } .

(2) w2 = { cyfra } cyfra .

# Kategorie składniowe Pascala

symbol-pascalowy = identyfikator | dyrektywa | liczba  
| etykieta | napis | symbol-specjalny.

# Kategorie składniowe Pascala

symbol-pascalowy = identyfikator | dyrektywa | liczba  
| etykieta | napis | symbol-specjalny.

symbol-specjalny = "+" | "-" | "\*" | "/" | "=" | "<"  
| ">" | "[" | "]" | "." | "," | ":"  
| ";" | "^" | "(" | ")" | "\*\*" | "<>"  
| "<=" | ">=" | ":=" | ".." | "><"  
| "=>" | ".." | slowo-kluczowe.

# Kategorie składniowe Pascala

symbol-pascalowy = identyfikator | dyrektywa | liczba  
| etykieta | napis | symbol-specjalny.

symbol-specjalny = "+" | "-" | "\*" | "/" | "=" | "<"  
| ">" | "[" | "]" | "." | "," | ":"  
| ";" | "^" | "(" | ")" | "\*\*" | "<>"  
| "<=" | ">=" | ":=" | ".." | "><"  
| "=>" | ".." | slowo-kluczowe.

slowo-kluczowe = "AND" | "AND\_THEN" | "ARRAY" | "BEGIN"  
| "BINDABLE" | "CASE" | "CONST" | "DIV"  
| "DO" | "DOWNTO" | "ELSE" | "END"  
| "EXPORT" | "FILE" | "FOR" | "FUNCTION"  
| "GOTO" | "IF" | "IMPORT" | "IN"  
| "LABEL" | "MOD" | "MODULE" | "NIL"  
| "NOT" | "OF" | "ONLY" | "OR"  
| "OR\_ELSE" | "OTHERWISE" | "PACKED"  
| "POW" | "PROCEDURE" | "PROGRAM"  
| "PROTECTED" | "QUALIFIED" | "RECORD"  
| "REPEAT" | "RESTRICTED" | "SET"  
| "THEN" | "TO" | "TYPE" | "UNTIL"  
| "VALUE" | "VAR" | "WHILE" | "WITH".

## Kategorie składniowe Pascala — cd.

```
liczba-rzeczywista =  
    [ znak-liczby ] liczba-rzeczywista-bez-znaku.
```

## Kategorie składniowe Pascala — cd.

```
liczba-rzeczywista =  
    [ znak-liczby ] liczba-rzeczywista-bez-znaku.
```

```
liczba-rzeczywista-bez-znaku =  
    liczba-calkowita-bez-znaku "." ciag-cyfr  
    [ "e" mnoznik-skalujacy ]  
| liczba-calkowita-bez-znaku "e" mnoznik-skalujacy.
```

## Kategorie składniowe Pascala — cd.

```
liczba-rzeczywista =  
    [ znak-liczby ] liczba-rzeczywista-bez-znaku.  
  
liczba-rzeczywista-bez-znaku =  
    liczba-calkowita-bez-znaku "." ciag-cyfr  
        [ "e" mnoznik-skalujacy ]  
    | liczba-calkowita-bez-znaku "e" mnoznik-skalujacy.  
  
mnoznik-skalujacy = liczba-calkowita.
```



# Typy danych

- Podstawowe typy proste
  - ★ liczby całkowite — INTEGER

# Typy danych

- Podstawowe typy proste
  - ★ liczby całkowite — INTEGER
  - ★ liczby rzeczywiste — REAL

# Typy danych

- Podstawowe typy proste
  - ★ liczby całkowite — INTEGER
  - ★ liczby rzeczywiste — REAL
  - ★ **zmienne logiczne** — BOOLEAN

# Typy danych

- Podstawowe typy proste
  - ★ liczby całkowite — INTEGER
  - ★ liczby rzeczywiste — REAL
  - ★ zmienne logiczne — BOOLEAN
  - ★ zbiór znaków ASCII — CHAR

# Typy danych

- Podstawowe typy proste
  - ★ liczby całkowite — INTEGER
  - ★ liczby rzeczywiste — REAL
  - ★ zmienne logiczne — BOOLEAN
  - ★ zbiór znaków ASCII — CHAR
- Podstawowe typy złożone
  - ★ typ okrojony — `Miesiace = 1..12;`

# Typy danych

- Podstawowe typy proste
  - ★ liczby całkowite — INTEGER
  - ★ liczby rzeczywiste — REAL
  - ★ zmienne logiczne — BOOLEAN
  - ★ zbiór znaków ASCII — CHAR
- Podstawowe typy złożone
  - ★ typ okrojony — `Miesiace = 1..12;`
  - ★ typ wyliczeniowy — `DniTygodnia = (Pon,Wto,Sro,Czw,Pia,Sob,Nie);`

# Typy danych

- Podstawowe typy proste

- ★ liczby całkowite — INTEGER
- ★ liczby rzeczywiste — REAL
- ★ zmienne logiczne — BOOLEAN
- ★ zbiór znaków ASCII — CHAR

- Podstawowe typy złożone

- ★ typ okrojony — `Miesiace = 1..12;`
- ★ typ wyliczeniowy — `DniTygodnia = (Pon,Wto,Sro,Czw,Pia,Sob,Nie);`
- ★ **tablice** — `Zal = ARRAY [1..200] OF 2..5;`

# Typy danych

- Podstawowe typy proste

- ★ liczby całkowite — INTEGER
- ★ liczby rzeczywiste — REAL
- ★ zmienne logiczne — BOOLEAN
- ★ zbiór znaków ASCII — CHAR

- Podstawowe typy złożone

- ★ typ okrojony — `Miesiace = 1..12;`
- ★ typ wyliczeniowy — `DniTygodnia = (Pon,Wto,Sro,Czw,Pia,Sob,Nie);`
- ★ tablice — `Zal = ARRAY [1..200] OF 2..5;`
- ★ rekordy — `Zespolona = RECORD`
  - `Re, Im: REAL;`
  - `END;`



# Typy danych

- Podstawowe typy proste

- ★ liczby całkowite — INTEGER
- ★ liczby rzeczywiste — REAL
- ★ zmienne logiczne — BOOLEAN
- ★ zbiór znaków ASCII — CHAR

- Podstawowe typy złożone

- ★ typ okrojony — `Miesiace = 1..12;`
- ★ typ wyliczeniowy — `DniTygodnia = (Pon,Wto,Sro,Czw,Pia,Sob,Nie);`
- ★ tablice — `Zal = ARRAY [1..200] OF 2..5;`
- ★ rekordy — `Zespolona = RECORD  
                    Re, Im: REAL;  
                    END;`
- ★ zbiory — `Litery = SET OF CHAR;`

## Definicja typów danych

```
sekcja-definicji-typow = [ "TYPE" definicja-typu ";"  
                          { definicja-typu ";" } ] .
```

## Definicja typów danych

```
sekcja-definicji-typow = [ "TYPE" definicja-typu ";"  
                          { definicja-typu ";" } ] .
```

```
definicja-typu = identyfikator "=" typ .
```

## Definicja typów danych

```
sekcja-definicji-typow = [ "TYPE" definicja-typu ";"  
                          { definicja-typu ";" } ] .
```

```
definicja-typu = identyfikator "=" typ .
```

```
typ = identyfikator-typu | nowy-typ .
```

## Definicja typów danych

```
sekcja-definicji-typow = [ "TYPE" definicja-typu ";"  
                          { definicja-typu ";" } ] .
```

```
definicja-typu = identyfikator "=" typ .
```

```
typ = identyfikator-typu | nowy-typ . . . .
```

## Definicja typów danych

```
sekcja-definicji-typow = [ "TYPE" definicja-typu ";"  
                          { definicja-typu ";" } ] .
```

```
definicja-typu = identyfikator "=" typ .
```

```
typ = identyfikator-typu | nowy-typ . ...
```

---

```
TYPE MojTyp = REAL;
```

## Definicja typów danych

```
sekcja-definicji-typow = [ "TYPE" definicja-typu ";"  
                          { definicja-typu ";" } ] .
```

```
definicja-typu = identyfikator "=" typ .
```

```
typ = identyfikator-typu | nowy-typ . ...
```

---

```
TYPE MojTyp = REAL;  
    Miesiace = 1..12;
```

## Definicja typów danych

```
sekcja-definicji-typow = [ "TYPE" definicja-typu ";"  
                          { definicja-typu ";" } ] .
```

```
definicja-typu = identyfikator "=" typ .
```

```
typ = identyfikator-typu | nowy-typ . ...
```

---

```
TYPE MojTyp = REAL;  
    Miesiace = 1..12;  
    DniTygodnia = (Po,Wt,Sr,Cz,Pi,So,Ni);
```



## Definicja typów danych

```
sekcja-definicji-typow = [ "TYPE" definicja-typu ";"  
                          { definicja-typu ";" } ] .
```

```
definicja-typu = identyfikator "=" typ .
```

```
typ = identyfikator-typu | nowy-typ . ...
```

---

```
TYPE MojTyp = REAL;  
    Miesiace = 1..12;  
    DniTygodnia = (Po,Wt,Sr,Cz,Pi,So,Ni);  
    Zal = ARRAY [1..200] OF INTEGER;
```

## Definicja typów danych

```
sekcja-definicji-typow = [ "TYPE" definicja-typu ";"  
                          { definicja-typu ";" } ] .
```

```
definicja-typu = identyfikator "=" typ .
```

```
typ = identyfikator-typu | nowy-typ . ...
```

---

```
TYPE MojTyp = REAL;  
  Miesiace = 1..12;  
  DniTygodnia = (Po,Wt,Sr,Cz,Pi,So,Ni);  
  Zal = ARRAY [1..200] OF INTEGER;  
  Zespolona = RECORD  
    Re, Im: MojTyp;  
  END;
```

## Definicja typów danych

```
sekcja-definicji-typow = [ "TYPE" definicja-typu ";"  
                          { definicja-typu ";" } ] .
```

```
definicja-typu = identyfikator "=" typ .
```

```
typ = identyfikator-typu | nowy-typ . ...
```

---

```
TYPE MojTyp = REAL;  
  Miesiace = 1..12;  
  DniTygodnia = (Po,Wt,Sr,Cz,Pi,So,Ni);  
  Zal = ARRAY [1..200] OF INTEGER;  
  Zespolona = RECORD  
    Re, Im: MojTyp;  
  END;  
Liczba = Zespolona;
```

## Definicja typów danych

```
sekcja-definicji-typow = [ "TYPE" definicja-typu ";"  
                          { definicja-typu ";" } ] .
```

```
definicja-typu = identyfikator "=" typ .
```

```
typ = identyfikator-typu | nowy-typ . ...
```

---

```
TYPE MojTyp = REAL;  
  Miesiace = 1..12;  
  DniTygodnia = (Po,Wt,Sr,Cz,Pi,So,Ni);  
  Zal = ARRAY [1..200] OF INTEGER;  
  Zespolona = RECORD  
    Re, Im: MojTyp;  
  END;  
  Liczba = Zespolona;  
  Tablica = ARRAY [1..10] OF Zespolona;
```

# Identyfikatory

identyfikator = litera { [ podkreślenie ]  
( litera | cyfra ) } .

# Identyfikatory

identyfikator = litera { [ podkreslenie ]  
( litera | cyfra ) } .

podkreslenie = "\_"











# Identyfikatory

identyfikator = litera { [ podkreślenie ]  
( litera | cyfra ) } .

podkreślenie = "\_"

litera = małe i duże litery alfabetu łacińskiego

cyfra = cyfry dziesiętne

**Identyfikator nie może pokrywać się ze słowami kluczowymi.**

---

Które z napisów są poprawnymi identyfikatorami?

moja\_dana      **Moja\_Dana**

# Identyfikatory

identyfikator = litera { [ podkreslenie ]  
( litera | cyfra ) } .

podkreslenie = "\_"

litera = małe i duże litery alfabetu łacińskiego

cyfra = cyfry dziesiętne

**Identyfikator nie może pokrywać się ze słowami kluczowymi.**

---

Które z napisów są poprawnymi identyfikatorami?

moja\_dana      Moja\_Dana      **MOJA\_DANA**

# Identyfikatory

identyfikator = litera { [ podkreślenie ]  
( litera | cyfra ) } .

podkreślenie = "\_"

litera = małe i duże litery alfabetu łacińskiego

cyfra = cyfry dziesiętne

**Identyfikator nie może pokrywać się ze słowami kluczowymi.**

---

Które z napisów są poprawnymi identyfikatorami?

moja\_dana      Moja\_Dana      MOJA\_DANA

moja\_dana2



# Identyfikatory

identyfikator = litera { [ podkreślenie ]  
( litera | cyfra ) } .

podkreślenie = "\_"

litera = małe i duże litery alfabetu łacińskiego

cyfra = cyfry dziesiętne

**Identyfikator nie może pokrywać się ze słowami kluczowymi.**

---

Które z napisów są poprawnymi identyfikatorami?

moja_dana	Moja_Dana	MOJA_DANA
moja_dana2	tmp21d3233	12tmp

# Identyfikatory

identyfikator = litera { [ podkreślenie ]  
( litera | cyfra ) } .

podkreślenie = "\_"

litera = małe i duże litery alfabetu łacińskiego

cyfra = cyfry dziesiętne

**Identyfikator nie może pokrywać się ze słowami kluczowymi.**

---

Które z napisów są poprawnymi identyfikatorami?

moja\_dana      Moja\_Dana      MOJA\_DANA

moja\_dana2      tmp21d3233      12tmp

moja



# Identyfikatory

identyfikator = litera { [ podkreślenie ]  
( litera | cyfra ) } .

podkreślenie = "\_"

litera = małe i duże litery alfabetu łacińskiego

cyfra = cyfry dziesiętne

**Identyfikator nie może pokrywać się ze słowami kluczowymi.**

---

Które z napisów są poprawnymi identyfikatorami?

moja_dana	Moja_Dana	MOJA_DANA
moja_dana2	tmp21d3233	12tmp
moja	moja_	

# Identyfikatory

identyfikator = litera { [ podkreślenie ]  
( litera | cyfra ) } .

podkreślenie = "\_"

litera = małe i duże litery alfabetu łacińskiego

cyfra = cyfry dziesiętne

**Identyfikator nie może pokrywać się ze słowami kluczowymi.**

---

Które z napisów są poprawnymi identyfikatorami?

moja_dana	Moja_Dana	MOJA_DANA
moja_dana2	tmp21d3233	12tmp
moja	moja_	moja__dana

# Identyfikatory

identyfikator = litera { [ podkreślenie ]  
( litera | cyfra ) } .

podkreślenie = "\_"

litera = małe i duże litery alfabetu łacińskiego

cyfra = cyfry dziesiętne

**Identyfikator nie może pokrywać się ze słowami kluczowymi.**

---

Które z napisów są poprawnymi identyfikatorami?

moja_dana	Moja_Dana	MOJA_DANA
moja_dana2	tmp21d3233	12tmp
moja	moja_	moja__dana
moja-dana		

# Identyfikatory

identyfikator = litera { [ podkreślenie ]  
( litera | cyfra ) } .

podkreślenie = "\_"

litera = małe i duże litery alfabetu łacińskiego

cyfra = cyfry dziesiętne

**Identyfikator nie może pokrywać się ze słowami kluczowymi.**

---

Które z napisów są poprawnymi identyfikatorami?

moja_dana	Moja_Dana	MOJA_DANA
moja_dana2	tmp21d3233	12tmp
moja	moja_	moja__dana
moja-dana	moja dana	

# Identyfikatory

identyfikator = litera { [ podkreślenie ]  
( litera | cyfra ) } .

podkreślenie = "\_"

litera = małe i duże litery alfabetu łacińskiego

cyfra = cyfry dziesiętne

Identyfikator nie może pokrywać się ze słowami kluczowymi.

---

Które z napisów są poprawnymi identyfikatorami?

moja_dana	Moja_Dana	MOJA_DANA
moja_dana2	tmp21d3233	12tmp
moja	moja_	moja__dana
moja-dana	moja dana	moja^dana

# Typy danych — zakres wartości

- liczby całkowite

reprezentacja	min	max
2 bajtowa	-32,768	32,767
4 bajtowa	-2,147,483,648	2,147,483,647

## Typy danych — zakres wartości

- liczby całkowite

reprezentacja	min	max
2 bajtowa	-32,768	32,767
4 bajtowa	-2,147,483,648	2,147,483,647

- liczby rzeczywiste

reprezentacja	min	max
4 bajtowa	1.401298e-45	3.402823e+38
8 bajtowa	4.94065645841246544e-324	1.79769313486231470e+308

# Typy danych — zakres wartości

- liczby całkowite

reprezentacja	min	max
2 bajtowa	-32,768	32,767
4 bajtowa	-2,147,483,648	2,147,483,647

- liczby rzeczywiste

reprezentacja	min	max
4 bajtowa	1.401298e-45	3.402823e+38
8 bajtowa	4.94065645841246544e-324	1.79769313486231470e+308

- zmienne logiczne

FALSE	TRUE
-------	------



## Typy danych — zakres wartości

- liczby całkowite

reprezentacja	min	max
2 bajtowa	-32,768	32,767
4 bajtowa	-2,147,483,648	2,147,483,647

- liczby rzeczywiste

reprezentacja	min	max
4 bajtowa	1.401298e-45	3.402823e+38
8 bajtowa	4.94065645841246544e-324	1.79769313486231470e+308

- zmienne logiczne

FALSE	TRUE
-------	------

- znaki ASCII

CHR(0)	CHR(255)
--------	----------

# Ograniczanie zakresu zmiennych

```
TYPE unsigned_int = 0..65536;
```

## Ograniczanie zakresu zmiennych

```
TYPE unsigned_int = 0..65536;  
    capital = 'A'..'Z';
```

# Ograniczanie zakresu zmiennych

```
TYPE unsigned_int = 0..65536;  
    capital = 'A'..'Z';  
    digits1 = '0'..'9';
```

## Ograniczanie zakresu zmiennych

```
TYPE unsigned_int = 0..65536;  
    capital = 'A'..'Z';  
    digits1 = '0'..'9';  
    digits2 = 0..9;
```

# Przykłady typów wyliczeniowych i okrojonych

```
TYPE kolory = (czerwony, niebieski, zielony, Azja);
```

# Przykłady typów wyliczeniowych i okrojonych

```
TYPE kolory = (czerwony, niebieski, zielony, Azja);
```

```
kontynenty = (Afryka, Ameryka_Pd, Ameryka_Pn,  
             Antarktyda, Australia, Azja, Europa);
```

## Przykłady typów wyliczeniowych i okrojonych

```
TYPE kolory = (czerwony, niebieski, zielony, Azja);  
kontynenty = (Afryka, Ameryka_Pd, Ameryka_Pn,  
              Antarktyda, Australia, Azja, Europa);  
tydzien = (pon,wto,sro,czw,pia,sob,nie);  
weekend=sob..nie;  
robocze=pon..pia;
```



## Przykłady typów wyliczeniowych i okrojonych

```
TYPE kolory = (czerwony, niebieski, zielony, Azja);  
kontynenty = (Afryka, Ameryka_Pd, Ameryka_Pn,  
              Antarktyda, Australia, Azja, Europa);  
tydzien = (pon,wto,sro,czw,pia,sob,nie);  
weekend=sob..nie;  
robocze=pon..pia;
```

---

```
tydzien=(pon,wto,sro,czw,pia,sob,nie);  
weekend=(sob,nie);  
robocze=(pon,wto,sro,czw,pia);
```

## Przykłady typów wyliczeniowych i okrojonych

```
TYPE kolory = (czerwony, niebieski, zielony, Azja);  
kontynenty = (Afryka, Ameryka_Pd, Ameryka_Pn,  
              Antarktyda, Australia, Azja, Europa);  
tydzien = (pon,wto,sro,czw,pia,sob,nie);  
weekend=sob..nie;  
robocze=pon..pia;
```

---

```
tydzien=(pon,wto,sro,czw,pia,sob,nie);  
weekend=(sob,nie);  
robocze=(pon,wto,sro,czw,pia);
```

---

```
robocze=(pon,wto,sro,czw,pia);  
tydzien=(robocze,sob,nie);
```

## Przykłady typów wyliczeniowych i okrojonych

```
X = 1..100;
```

```
Y = 1..1;
```

```
Z = true..true;
```

```
V = INTEGER;
```

```
Przyjaciele = Ameryka_Pd..Ameryka_Pn
```

# Zmienne

Zmienne posiadają:

- a) nazwę, która musi być identyfikatorem pascalowym i różnić się od słów kluczowych Pascala,

# Zmienne

Zmienne posiadają:

- a) nazwę, która musi być identyfikatorem pascalowym i różnić się od słów kluczowych Pascala,
- b) typ, który określa, jakie informacje będą przechowywane w zmiennej; nazwa i typ zmiennej są wymienione w jej deklaracji,

# Zmienne

Zmienne posiadają:

- a) nazwę, która musi być identyfikatorem pascalowym i różnić się od słów kluczowych Pascala,
- b) typ, który określa, jakie informacje będą przechowywane w zmiennej; nazwa i typ zmiennej są wymienione w jej deklaracji,
- c) aktualną wartość,

# Zmienne

Zmienne posiadają:

- a) nazwę, która musi być identyfikatorem pascalowym i różnić się od słów kluczowych Pascala,
- b) typ, który określa, jakie informacje będą przechowywane w zmiennej; nazwa i typ zmiennej są wymienione w jej deklaracji,
- c) aktualną wartość,
- d) alokację, która jest miejscem w pamięci, gdzie ma być przechowywana wartość zmiennej,

# Zmienne

Zmienne posiadają:

- a) nazwę, która musi być identyfikatorem pascalowym i różnić się od słów kluczowych Pascala,
- b) typ, który określa, jakie informacje będą przechowywane w zmiennej; nazwa i typ zmiennej są wymienione w jej deklaracji,
- c) aktualną wartość,
- d) alokację, która jest miejscem w pamięci, gdzie ma być przechowywana wartość zmiennej,
- e) zakres, który jest miejscem w programie, gdzie można odwoływać się do zmiennej,



# Zmienne

Zmienne posiadają:

- a) nazwę, która musi być identyfikatorem pascalowym i różnić się od słów kluczowych Pascala,
- b) typ, który określa, jakie informacje będą przechowywane w zmiennej; nazwa i typ zmiennej są wymienione w jej deklaracji,
- c) aktualną wartość,
- d) alokację, która jest miejscem w pamięci, gdzie ma być przechowywana wartość zmiennej,
- e) zakres, który jest miejscem w programie, gdzie można odwoływać się do zmiennej,
- f) czas trwania, to jest czas, w jakim mogą wystąpić odwołania do zmiennej.

# Zmienne

Zmienne posiadają:

- a) nazwę, która musi być identyfikatorem pascalowym i różnić się od słów kluczowych Pascala,
- b) typ, który określa, jakie informacje będą przechowywane w zmiennej; nazwa i typ zmiennej są wymienione w jej deklaracji,
- c) aktualną wartość,
- d) alokację, która jest miejscem w pamięci, gdzie ma być przechowywana wartość zmiennej,
- e) zakres, który jest miejscem w programie, gdzie można odwoływać się do zmiennej,
- f) czas trwania, to jest czas, w jakim mogą wystąpić odwołania do zmiennej.

- 
- a), b) — określone treścią programu,
  - c), d) — ustalone chwilowo w trakcie wykonywania programu,
  - e), f) — określone w Pascalu na stałe.

## Deklaracja zmiennych

```
sekcja-deklaracji-zmiennych = [ "VAR" deklaracja-zmiennych ";"  
                               { deklaracja-zmiennych ";" } ] .
```

## Deklaracja zmiennych

```
sekcja-deklaracji-zmiennych = [ "VAR" deklaracja-zmiennych ";"  
                               { deklaracja-zmiennych ";" } ] .
```

```
deklaracja-zmiennych = lista-identyfikatorow ":" typ .
```

## Deklaracja zmiennych

sekcja-deklaracji-zmiennych = [ "VAR" deklaracja-zmiennych ";"  
{ deklaracja-zmiennych ";" } ] .

deklaracja-zmiennych = lista-identyfikatorow ":" typ .

lista-identyfikatorow = identyfikator { "," identyfikator } .





## Deklaracja zmiennych

```
sekcja-deklaracji-zmiennych = [ "VAR" deklaracja-zmiennych ";"  
                               { deklaracja-zmiennych ";" } ] .
```

```
deklaracja-zmiennych = lista-identyfikatorow ":" typ .
```

```
lista-identyfikatorow = identyfikator { "," identyfikator } .
```

---

```
VAR krok      : INTEGER;  
    delta     : REAL;  
    delta1    : MojTyp;
```



## Deklaracja zmiennych

```
sekcja-deklaracji-zmiennych = [ "VAR" deklaracja-zmiennych ";"  
                                { deklaracja-zmiennych ";" } ] .
```

```
deklaracja-zmiennych = lista-identyfikatorow ":" typ .
```

```
lista-identyfikatorow = identyfikator { "," identyfikator } .
```

---

```
VAR krok      : INTEGER;  
    delta     : REAL;  
    delta1    : MojTyp;  
    delta2    : Zespolona;
```



## Deklaracja zmiennych

```
sekcja-deklaracji-zmiennych = [ "VAR" deklaracja-zmiennych ";"  
                                { deklaracja-zmiennych ";" } ] .
```

```
deklaracja-zmiennych = lista-identyfikatorow ":" typ .
```

```
lista-identyfikatorow = identyfikator { "," identyfikator } .
```

---

```
VAR krok      : INTEGER;  
    delta     : REAL;  
    delta1    : MojTyp;  
    delta2    : Zespolona;  
    i, j, k   : INTEGER;  
    z1        : Zal;
```

## Deklaracja zmiennych

```
sekcja-deklaracji-zmiennych = [ "VAR" deklaracja-zmiennych ";"  
                                { deklaracja-zmiennych ";" } ] .
```

```
deklaracja-zmiennych = lista-identyfikatorow ":" typ .
```

```
lista-identyfikatorow = identyfikator { "," identyfikator } .
```

---

```
VAR krok      : INTEGER;  
    delta     : REAL;  
    delta1    : MojTyp;  
    delta2    : Zespolona;  
    i, j, k   : INTEGER;  
    z1        : Zal;  
    z2, z3    : Zal;
```

## Deklaracja zmiennych

```
sekcja-deklaracji-zmiennych = [ "VAR" deklaracja-zmiennych ";"  
                                { deklaracja-zmiennych ";" } ] .
```

```
deklaracja-zmiennych = lista-identyfikatorow ":" typ .
```

```
lista-identyfikatorow = identyfikator { "," identyfikator } .
```

---

```
VAR krok      : INTEGER;  
    delta     : REAL;  
    delta1    : MojTyp;  
    delta2    : Zespolona;  
    i, j, k   : INTEGER;  
    z1        : Zal;  
    z2, z3    : Zal;  
    z4, z5    : ARRAY [1..200] OF INTEGER;
```

# Wyrażenia

- **zmienne** — `VAR Promien, Kat, Luk: REAL;`

# Wyrażenia

- zmienne — `VAR Promien, Kat, Luk: REAL;`
- stałe
  - ★ jawne — `3, 3.14, TRUE, 'a', 'ala ma kota'`

# Wyrażenia

- **zmienne** — `VAR Promien, Kat, Luk: REAL;`
- **stałe**
  - ★ **jawne** — `3, 3.14, TRUE, 'a', 'ala ma kota'`
  - ★ **symboliczne** — `CONST pi = 3.14159, im = 'ala';`



# Wyrażenia

- zmienne — `VAR Promien, Kat, Luk: REAL;`
- stałe
  - ★ jawne — `3, 3.14, TRUE, 'a', 'ala ma kota'`
  - ★ symboliczne — `CONST pi = 3.14159, im = 'ala';`
- wywołania funkcji — `SIN(Kat/180*pi)`

# Wyrażenia

- zmienne — `VAR Promien, Kat, Luk: REAL;`
- stałe
  - ★ jawne — `3, 3.14, TRUE, 'a', 'ala ma kota'`
  - ★ symboliczne — `CONST pi = 3.14159, im = 'ala';`
- wywołania funkcji — `SIN(Kat/180*pi)`
- wyrażenia operatorowe — `(a * b) + (c * d)`  
`(Kat >= 0.0) AND (Kat <= 180.0)`

# Reguły wyliczania wyrażeń

a) **stałe jawne**: ich wartość jest im równa,

## Reguły wyliczania wyrażeń

- a) **stałe jawne:** ich wartość jest im równa,
- b) **stałe symboliczne i zmienne:** ich wartość jest im przypisana, przy czym dla zmiennych może ulegać zmianie w trakcie pracy programu,

## Reguły wyliczania wyrażeń

- a) **stałe jawne:** ich wartość jest im równa,
- b) **stałe symboliczne i zmienne:** ich wartość jest im przypisana, przy czym dla zmiennych może ulegać zmianie w trakcie pracy programu,
- c) **wywołania funkcji i wyrażenia operatorowe:** wpierw wyliczane są wartości argumentów (które same są wyrażeniami), a następnie operator lub funkcja wylicza swoją wartość.

# Wyrażenia — przykłady

## Stałe jawne

0    -0    (-0)    (-121)    (-00121)    -(121)

## Wyrażenia — przykłady

### Stałe jawne

0	-0	(-0)	(-121)	(-00121)	-(121)
+7,345	16.94	-21e12	-21.13e12.13	21.1e3e5	

## Wyrażenia — przykłady

### Stałe jawne

0	-0	(-0)	(-121)	(-00121)	-(121)
+7,345	16.94	-21e12	-21.13e12.13	21.1e3e5	
-21e+21	-2.1e-21	.21	2.1	21.	21



# Wyrażenia — przykłady

## Stałe jawne

0	-0	(-0)	(-121)	(-00121)	-(121)
+7,345	16.94	-21e12	-21.13e12.13	21.1e3e5	
-21e+21	-2.1e-21	.21	2.1	21.	21
Jasiu	'Jasiu'	"Jasiu"	Prawda	True	"True"

## Wyrażenia — przykłady

### Stałe jawne

0      -0      (-0)      (-121)      (-00121)      -(121)  
 +7,345      16.94      -21e12      -21.13e12.13      21.1e3e5  
 -21e+21      -2.1e-21      .21      2.1      21.      21  
 Jasiu      'Jasiu'      "Jasiu"      Prawda      True      "True"

---

### Stałe symboliczne

Dwa = 2      Cztery = 2\*Dwa      Cztery = 2\*2      Cztery = 4

## Wyrażenia — przykłady

### Stałe jawne

```
0      -0      (-0)      (-121)      (-00121)      -(121)
+7,345  16.94  -21e12  -21.13e12.13  21.1e3e5
-21e+21  -2.1e-21  .21    2.1    21.    21
Jasiu    'Jasiu'  "Jasiu"  Prawda   True    "True"
```

---

### Stałe symboliczne

```
Dwa = 2      Cztery = 2*Dwa  Cztery = 2*2  Cztery = 4
Dwa = 4      Dwa = '4'    Dwa = 2.1    Dwa = 'Ala'
```

## Wyrażenia — przykłady

### Stałe jawne

```
0      -0      (-0)      (-121)      (-00121)      -(121)
+7,345  16.94  -21e12  -21.13e12.13  21.1e3e5
-21e+21  -2.1e-21  .21    2.1    21.    21
Jasiu    'Jasiu'  "Jasiu"  Prawda   True    "True"
```

---

### Stałe symboliczne

```
Dwa = 2      Cztery = 2*Dwa  Cztery = 2*2  Cztery = 4
Dwa = 4      Dwa = '4'    Dwa = 2.1    Dwa = 'Ala'
```

---

### Wywołania funkcji

```
TRUNC(odsetki)      TRUNC(0.8*odsetki)      ROUND(0.8)
```

## Wyrażenia — przykłady

### Stałe jawne

```
0      -0      (-0)      (-121)      (-00121)      -(121)
+7,345  16.94  -21e12  -21.13e12.13  21.1e3e5
-21e+21  -2.1e-21  .21      2.1      21.      21
Jasiu    'Jasiu'  "Jasiu"  Prawda    True     "True"
```

---

### Stałe symboliczne

```
Dwa = 2      Cztery = 2*Dwa      Cztery = 2*2      Cztery = 4
Dwa = 4      Dwa = '4'        Dwa = 2.1        Dwa = 'Ala'
```

---

### Wywołania funkcji

```
TRUNC(odsetki)      TRUNC(0.8*odsetki)      ROUND(0.8)
SQRT(SIN(alfa))    TRUNC(SIN(alfa)+COS(alfa))
```

# Wyrażenia — przykłady

## Wyrażenia operatorowe

`12 * dwapi + 3.5 * (calka + calka2)`

# Wyrażenia — przykłady

## Wyrażenia operatorowe

```
12 * dwapi + 3.5 * (calka + calka2)
```

```
12 dwapi + 3.5 * (2pi + 3pi)
```

# Wyrażenia — przykłady

## Wyrażenia operatorowe

`12 * dwapi + 3.5 * (calka + calka2)`

`12 dwapi + 3.5 * (2pi + 3pi)`

`kat > 0 AND kat < 2`



# Wyrażenia — przykłady

## Wyrażenia operatorowe

`12 * dwapi + 3.5 * (calka + calka2)`

`12 dwapi + 3.5 * (2pi + 3pi)`

`kat > 0 AND kat < 2`

`(kat > 0 + 3) + 2`

# Wyrażenia — przykłady

## Wyrażenia operatorowe

`12 * dwapi + 3.5 * (calka + calka2)`

`12 dwapi + 3.5 * (2pi + 3pi)`

`kat > 0 AND kat < 2`

`(kat > 0 + 3) + 2`

`podstawa DIV 0 + 2`

## Wyrażenia — przykłady

### Wyrażenia operatorowe

`12 * dwapi + 3.5 * (calka + calka2)`

`12 dwapi + 3.5 * (2pi + 3pi)`

`kat > 0 AND kat < 2`

`(kat > 0 + 3) + 2`

`podstawa DIV 0 + 2`

- 
- Skąd wiemy jak je konstruować?

## Wyrażenia — przykłady

### Wyrażenia operatorowe

`12 * dwapi + 3.5 * (calka + calka2)`

`12 dwapi + 3.5 * (2pi + 3pi)`

`kat > 0 AND kat < 2`

`(kat > 0 + 3) + 2`

`podstawa DIV 0 + 2`

- 
- Skąd wiemy jak je konstruować?
  - Jak rozstrzygać które z nich są poprawne?

## Wyrażenia — przykłady

### Wyrażenia operatorowe

`12 * dwapi + 3.5 * (calka + calka2)`

`12 dwapi + 3.5 * (2pi + 3pi)`

`kat > 0 AND kat < 2`

`(kat > 0 + 3) + 2`

`podstawa DIV 0 + 2`

- 
- Skąd wiemy jak je konstruować?
  - Jak rozstrzygać które z nich są poprawne?
  - A które mają sens?

# Operatory

- Podstawowe operatory proste

\* porównanie — =

# Operatory

- Podstawowe operatory proste
  - \* porównanie — =
  - \* przypisanie — :=

# Operatory

- Podstawowe operatory proste
  - \* porównanie — =
  - \* przypisanie — :=
- ★ operatory arytmetyczne
  - \* **dodawanie** — +



# Operatory

- Podstawowe operatory proste
  - \* porównanie — =
  - \* przypisanie — :=
- ★ operatory arytmetyczne
  - \* dodawanie — +
  - \* odejmowanie — -

# Operatory

- Podstawowe operatory proste
  - \* porównanie — =
  - \* przypisanie — :=
- ★ operatory arytmetyczne
  - \* dodawanie — +
  - \* odejmowanie — -
  - \* mnożenie — \*

# Operatory

- Podstawowe operatory proste

- \* porównanie — =

- \* przypisanie — :=

- ★ operatory arytmetyczne

- \* dodawanie — +

- \* odejmowanie — -

- \* mnożenie — \*

- \* dzielenie — /, DIV, MOD

# Operatory

- Podstawowe operatory proste
  - \* porównanie — =
  - \* przypisanie — :=
- ★ operatory arytmetyczne
  - \* dodawanie — +
  - \* odejmowanie — -
  - \* mnożenie — \*
  - \* dzielenie — /, DIV, MOD
- ★ operatory logiczne
  - \* alternatywa — OR

# Operatory

- Podstawowe operatory proste
  - \* porównanie — =
  - \* przypisanie — :=
- ★ operatory arytmetyczne
  - \* dodawanie — +
  - \* odejmowanie — -
  - \* mnożenie — \*
  - \* dzielenie — /, DIV, MOD
- ★ operatory logiczne
  - \* alternatywa — OR
  - \* koniunkcja — AND

# Operatory

- Podstawowe operatory proste
  - \* porównanie — =
  - \* przypisanie — :=
- ★ operatory arytmetyczne
  - \* dodawanie — +
  - \* odejmowanie — -
  - \* mnożenie — \*
  - \* dzielenie — /, DIV, MOD
- ★ operatory logiczne
  - \* alternatywa — OR
  - \* koniunkcja — AND
  - \* negacja — NOT

# Operatory

- Podstawowe operatory proste
  - \* porównanie — =
  - \* przypisanie — :=
- ★ operatory arytmetyczne
  - \* dodawanie — +
  - \* odejmowanie — -
  - \* mnożenie — \*
  - \* dzielenie — /, DIV, MOD
- ★ operatory logiczne
  - \* alternatywa — OR
  - \* koniunkcja — AND
  - \* negacja — NOT
- ★ pozostałe operatory relacyjne — <, <=, <>, >=, >

# Operatory

- Podstawowe operatory proste
  - \* porównanie — =
  - \* przypisanie — :=
- ★ operatory arytmetyczne
  - \* dodawanie — +
  - \* odejmowanie — -
  - \* mnożenie — \*
  - \* dzielenie — /, DIV, MOD
- ★ operatory logiczne
  - \* alternatywa — OR
  - \* koniunkcja — AND
  - \* negacja — NOT
- ★ pozostałe operatory relacyjne — <, <=, <>, >=, >
- Podstawowe operatory złożone
  - ★ konstruktory
  - ★ selektory



# Operatory — priorytety

priorytet	operatory				
	grupu- jące	arytmetyczne	logiczne	mногоś- ciowe	relacyjne
najwyższy	( )	–	NOT	*	
		* / DIV MOD	AND		
		+ –	OR	+ –	
najniższy				IN	< <= = <> >= >

# Instrukcje

*Instrukcja* — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego.

# Instrukcje

*Instrukcja* — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego.

## Przykłady instrukcji

```
WRITELN('Oto jest instrukcja');
```

# Instrukcje

*Instrukcja* — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego.

## Przykłady instrukcji

```
WRITELN('Oto jest instrukcja');  
READ(liczba1);
```

# Instrukcje

*Instrukcja* — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego.

## Przykłady instrukcji

```
WRITELN('Oto jest instrukcja');  
READ(liczba1);  
wynik := 3;
```

# Instrukcje

*Instrukcja* — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego.

## Przykłady instrukcji

```
WRITELN('Oto jest instrukcja');  
READ(liczba1);  
wynik := 3;  
IF wynik = 3 THEN  
    WRITELN('Wynik poprawny')  
ELSE  
BEGIN  
    wynik := 7;  
    WRITELN('Wynik niepoprawny')  
END;
```

# Instrukcje

*Instrukcja* — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego.

## Przykłady instrukcji

```
WRITELN('Oto jest instrukcja');  
READ(liczba1);  
wynik := 3;  
IF wynik = 3 THEN  
    WRITELN('Wynik poprawny')  
ELSE  
BEGIN  
    wynik := 7;  
    WRITELN('Wynik niepoprawny')  
END;
```

# Instrukcje

*Instrukcja* — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego.

## Przykłady instrukcji

```
WRITELN('Oto jest instrukcja');  
READ(liczba1);  
wynik := 3;  
IF wynik = 3 THEN  
    WRITELN('Wynik poprawny')  
ELSE  
    BEGIN  
        wynik := 7;  
        WRITELN('Wynik niepoprawny')  
    END;  
END;
```



# Instrukcje

*Instrukcja* — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego.

## Przykłady instrukcji

```
WRITELN('Oto jest instrukcja');  
READ(liczba1);  
wynik := 3;  
IF wynik = 3 THEN  
    WRITELN('Wynik poprawny')  
ELSE  
BEGIN  
    wynik := 7;  
    WRITELN('Wynik niepoprawny')  
END;
```

# Instrukcje

*Instrukcja* — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego.

## Przykłady instrukcji

```
WRITELN('Oto jest instrukcja');  
READ(liczba1);  
wynik := 3;  
IF wynik = 3 THEN  
    WRITELN('Wynik poprawny')  
ELSE  
BEGIN  
    wynik := 7;  
    WRITELN('Wynik niepoprawny')  
END;
```

# Instrukcja przypisania

`instrukcja-przypisania = zmienna " := " wyrażenie.`

# Instrukcja przypisania

instrukcja-przypisania = zmienna " := " wyrażenie.

---

x := 1

# Instrukcja przypisania

instrukcja-przypisania = zmienna " := " wyrażenie.

---

`x := 1`

`x := 1 + 2`

# Instrukcja przypisania

instrukcja-przypisania = zmienna " := " wyrażenie.

---

x := 1

x := 1 + 2

x := 'ala ma kota'

# Instrukcja przypisania

instrukcja-przypisania = zmienna " := " wyrażenie.

---

`x := 1`

`x := 1 + 2`

`x := 'ala ma kota'`

`x := y + 3`

# Instrukcja przypisania

instrukcja-przypisania = zmienna " := " wyrażenie.

---

`x := 1`

`x := 1 + 2`

`x := 'ala ma kota'`

`x := y + 3`

`x := x + 3`



# Instrukcja przypisania

instrukcja-przypisania = zmienna " := " wyrażenie.

---

`x := 1`

`x := 'ala ma kota'`

`x := x + 3`

`x := 1 + 2`

`x := y + 3`

`x := SIN(alfa*180/3.14)`

# Zgodność typów w instrukcji przypisania

$x := 1$

# Zgodność typów w instrukcji przypisania

$x := 1$

$x := 1.0$

# Zgodność typów w instrukcji przypisania

`x := 1`            `x := 1.0`  
`x := 'a'`

# Zgodność typów w instrukcji przypisania

`x := 1`

`x := 1.0`

`x := 'a'`

`x := TRUE`

# Zgodność typów w instrukcji przypisania

`x := 1`

`x := 1.0`

`x := 'a'`

`x := TRUE`

`x := y`

# Zgodność typów w instrukcji przypisania

`x := 1`

`x := 1.0`

`x := 'a'`

`x := TRUE`

`x := y`

`x := x + y`

# Zgodność typów w instrukcji przypisania

```
x := 1           x := 1.0
x := 'a'        x := TRUE
x := y          x := x + y
x := y + z
```



# Zgodność typów w instrukcji przypisania

<code>x := 1</code>	<code>x := 1.0</code>
<code>x := 'a'</code>	<code>x := TRUE</code>
<code>x := y</code>	<code>x := x + y</code>
<code>x := y + z</code>	<code>x := y &lt; z</code>

## Zgodność typów w instrukcji przypisania

```
x := 1           x := 1.0
x := 'a'        x := TRUE
x := y          x := x + y
x := y + z      x := y < z
```

---

```
TYPE Miesiace = 1..12;
     DniTygodnia = (Pon, Wto, Sro, Czw, Pia, Sob, Nie);
```

## Zgodność typów w instrukcji przypisania

```
x := 1           x := 1.0
x := 'a'         x := TRUE
x := y           x := x + y
x := y + z       x := y < z
```

---

```
TYPE Miesiace = 1..12;
      DniTygodnia = (Pon, Wto, Sro, Czw, Pia, Sob, Nie);
VAR  dzien      : DniTygodnia;
      miesiac    : Miesiace;
      liczba     : INTEGER;
```

## Zgodność typów w instrukcji przypisania

```
x := 1           x := 1.0
x := 'a'        x := TRUE
x := y          x := x + y
x := y + z      x := y < z
```

---

```
TYPE Miesiace = 1..12;
      DniTygodnia = (Pon, Wto, Sro, Czw, Pia, Sob, Nie);
VAR  dzien      : DniTygodnia;
      miesiac    : Miesiace;
      liczba     : INTEGER;
```

```
liczba := dzien;
```

## Zgodność typów w instrukcji przypisania

```
x := 1           x := 1.0
x := 'a'        x := TRUE
x := y          x := x + y
x := y + z      x := y < z
```

---

```
TYPE Miesiace = 1..12;
      DniTygodnia = (Pon, Wto, Sro, Czw, Pia, Sob, Nie);
VAR   dzien      : DniTygodnia;
      miesiac    : Miesiace;
      liczba     : INTEGER;

liczba := dzien;
liczba := miesiac;
```

## Zgodność typów w instrukcji przypisania

```
x := 1           x := 1.0
x := 'a'        x := TRUE
x := y          x := x + y
x := y + z      x := y < z
```

---

```
TYPE Miesiace = 1..12;
      DniTygodnia = (Pon, Wto, Sro, Czw, Pia, Sob, Nie);
VAR   dzien      : DniTygodnia;
      miesiac    : Miesiace;
      liczba     : INTEGER;

liczba := dzien;
liczba := miesiac;
miesiac := liczba;
```

# Struktura programu w Pascalu

```
PROGRAM policz(INPUT,OUTPUT);

CONST MaxInt = 20;
TYPE  Zakres = 0..MaxInt;
VAR   zmienna : Zakres;
      pomoc   : INTEGER;

BEGIN
  WRITELN('Podaj liczbe w zakresie 0..',MaxInt:1);
  READLN(zmienna);
  pomoc := 0;
  WHILE zmienna < MaxInt DO
  BEGIN
    pomoc := pomoc + 1;
    zmienna := zmienna + 1;
  END;
  WRITELN('Zmienna osiagnela wartosc ',MaxInt:1,
          ' w ',pomoc:1,' krokach.');
```

END.

# Struktura programu w Pascalu

```
PROGRAM policz(INPUT,OUTPUT);

CONST MaxInt = 20;
TYPE  Zakres = 0..MaxInt;
VAR   zmienna : Zakres;
      pomoc   : INTEGER;

BEGIN
  WRITELN('Podaj liczbe w zakresie 0..',MaxInt:1);
  READLN(zmienna);
  pomoc := 0;
  WHILE zmienna < MaxInt DO
  BEGIN
    pomoc := pomoc + 1;
    zmienna := zmienna + 1;
  END;
  WRITELN('Zmienna osiagnela wartosc ',MaxInt:1,
          ' w ',pomoc:1,' krokach.');
```

END.

- Nagłówek programu



# Struktura programu w Pascalu

```
PROGRAM policz(INPUT,OUTPUT);

CONST MaxInt = 20;
TYPE  Zakres = 0..MaxInt;
VAR   zmienna : Zakres;
      pomoc   : INTEGER;

BEGIN
  WRITELN('Podaj liczbe w zakresie 0..',MaxInt:1);
  READLN(zmienna);
  pomoc := 0;
  WHILE zmienna < MaxInt DO
  BEGIN
    pomoc := pomoc + 1;
    zmienna := zmienna + 1;
  END;
  WRITELN('Zmienna osiagnela wartosc ',MaxInt:1,
          ' w ',pomoc:1,' krokach.');
```

END.

- Nagłówek programu
- Blok programu

# Struktura programu w Pascalu

```
PROGRAM policz(INPUT,OUTPUT);

CONST MaxInt = 20;
TYPE  Zakres = 0..MaxInt;
VAR   zmienna : Zakres;
      pomoc   : INTEGER;

BEGIN
  WRITELN('Podaj liczbe w zakresie 0..',MaxInt:1);
  READLN(zmienna);
  pomoc := 0;
  WHILE zmienna < MaxInt DO
  BEGIN
    pomoc := pomoc + 1;
    zmienna := zmienna + 1;
  END;
  WRITELN('Zmienna osiagnela wartosc ',MaxInt:1,
          ' w ',pomoc:1,' krokach.');
```

END.

- Nagłówek programu
- Blok programu
  - ★ Sekcja deklaracji stałych

# Struktura programu w Pascalu

```
PROGRAM policz(INPUT,OUTPUT);

CONST MaxInt = 20;
TYPE  Zakres = 0..MaxInt;
VAR   zmienna : Zakres;
      pomoc   : INTEGER;

BEGIN
  WRITELN('Podaj liczbe w zakresie 0..',MaxInt:1);
  READLN(zmienna);
  pomoc := 0;
  WHILE zmienna < MaxInt DO
  BEGIN
    pomoc := pomoc + 1;
    zmienna := zmienna + 1;
  END;
  WRITELN('Zmienna osiagnela wartosc ',MaxInt:1,
          ' w ',pomoc:1,' krokach.');
```

END.

- Nagłówek programu
- Blok programu
  - ★ Sekcja deklaracji stałych
  - ★ Sekcja definicji typów

# Struktura programu w Pascalu

```
PROGRAM policz(INPUT,OUTPUT);

CONST MaxInt = 20;
TYPE  Zakres = 0..MaxInt;
VAR   zmienna : Zakres;
      pomoc   : INTEGER;

BEGIN
  WRITELN('Podaj liczbe w zakresie 0..',MaxInt:1);
  READLN(zmienna);
  pomoc := 0;
  WHILE zmienna < MaxInt DO
  BEGIN
    pomoc := pomoc + 1;
    zmienna := zmienna + 1;
  END;
  WRITELN('Zmienna osiagnela wartosc ',MaxInt:1,
          ' w ',pomoc:1,' krokach.');
```

END.

- Nagłówek programu
- Blok programu
  - ★ Sekcja deklaracji stałych
  - ★ Sekcja definicji typów
  - ★ Sekcja deklaracji zmiennych

# Struktura programu w Pascalu

```
PROGRAM policz(INPUT,OUTPUT);

CONST MaxInt = 20;
TYPE  Zakres = 0..MaxInt;
VAR   zmienna : Zakres;
      pomoc   : INTEGER;

BEGIN
  WRITELN('Podaj liczbe w zakresie 0..',MaxInt:1);
  READLN(zmienna);
  pomoc := 0;
  WHILE zmienna < MaxInt DO
  BEGIN
    pomoc := pomoc + 1;
    zmienna := zmienna + 1;
  END;
  WRITELN('Zmienna osiagnela wartosc ',MaxInt:1,
          ' w ',pomoc:1,' krokach.');
```

END.

- Nagłówek programu
- Blok programu
  - ★ Sekcja deklaracji stałych
  - ★ Sekcja definicji typów
  - ★ Sekcja deklaracji zmiennych
  - ★ Sekcja operacyjna

# Struktura programu w Pascalu

```
PROGRAM policz(INPUT,OUTPUT);

CONST MaxInt = 20;
TYPE  Zakres = 0..MaxInt;
VAR   zmienna : Zakres;
      pomoc   : INTEGER;

BEGIN
  WRITELN('Podaj liczbe w zakresie 0..',MaxInt:1);
  READLN(zmienna);
  pomoc := 0;
  WHILE zmienna < MaxInt DO
  BEGIN
    pomoc := pomoc + 1;
    zmienna := zmienna + 1;
  END;
  WRITELN('Zmienna osiagnela wartosc ',MaxInt:1,
          ' w ',pomoc:1,' krokach.');
```

END.

- Nagłówek programu
  - Blok programu
    - ★ Sekcja deklaracji stałych
    - ★ Sekcja definicji typów
    - ★ Sekcja deklaracji zmiennych
    - ★ Sekcja operacyjna
- ||  
Instrukcja złożona

# Struktura programu w Pascalu

```
PROGRAM policz(INPUT,OUTPUT);

CONST MaxInt = 20;
TYPE  Zakres = 0..MaxInt;
VAR   zmienna : Zakres;
      pomoc   : INTEGER;

BEGIN
  WRITELN('Podaj liczbe w zakresie 0..',MaxInt:1);
  READLN(zmienna);
  pomoc := 0;
  WHILE zmienna < MaxInt DO
  BEGIN
    pomoc := pomoc + 1;
    zmienna := zmienna + 1;
  END;
  WRITELN('Zmienna osiagnela wartosc ',MaxInt:1,
          ' w ',pomoc:1,' krokach.');
```

END.

- Nagłówek programu
  - Blok programu
    - ★ Sekcja deklaracji stałych
    - ★ Sekcja definicji typów
    - ★ Sekcja deklaracji zmiennych
    - ★ Sekcja operacyjna
- ||  
Instrukcja złożona
- \* BEGIN

# Struktura programu w Pascalu

```
PROGRAM policz(INPUT,OUTPUT);

CONST MaxInt = 20;
TYPE  Zakres = 0..MaxInt;
VAR   zmienna : Zakres;
      pomoc   : INTEGER;

BEGIN
  WRITELN('Podaj liczbe w zakresie 0..',MaxInt:1);
  READLN(zmienna);
  pomoc := 0;
  WHILE zmienna < MaxInt DO
  BEGIN
    pomoc := pomoc + 1;
    zmienna := zmienna + 1;
  END;
  WRITELN('Zmienna osiagnela wartosc ',MaxInt:1,
          ' w ',pomoc:1,' krokach.');
```

END.

- Nagłówek programu
- Blok programu
  - ★ Sekcja deklaracji stałych
  - ★ Sekcja definicji typów
  - ★ Sekcja deklaracji zmiennych
  - ★ Sekcja operacyjna
    - ||  
Instrukcja złożona
    - \* BEGIN
    - \* Ciąg instrukcji



# Struktura programu w Pascalu

```
PROGRAM policz(INPUT,OUTPUT);

CONST MaxInt = 20;
TYPE  Zakres = 0..MaxInt;
VAR   zmienna : Zakres;
      pomoc   : INTEGER;

BEGIN
  WRITELN('Podaj liczbe w zakresie 0..',MaxInt:1);
  READLN(zmienna);
  pomoc := 0;
  WHILE zmienna < MaxInt DO
  BEGIN
    pomoc := pomoc + 1;
    zmienna := zmienna + 1;
  END;
  WRITELN('Zmienna osiagnela wartosc ',MaxInt:1,
          ' w ',pomoc:1,' krokach.');
```

**END.**

- Nagłówek programu
- Blok programu
  - ★ Sekcja deklaracji stałych
  - ★ Sekcja definicji typów
  - ★ Sekcja deklaracji zmiennych
  - ★ Sekcja operacyjna
    - ||
    - Instrukcja złożona
    - \* BEGIN
    - \* Ciąg instrukcji
    - \* **END**

# Struktura programu w Pascalu

```
program = naglowek-programu ";" blok-programu ".".
```

# Struktura programu w Pascalu

```
program = naglowek-programu ";" blok-programu ".".  
naglowek-programu = "PROGRAM" identyfikator  
                  [ "(" parametry-programu ")" ].
```

## Struktura programu w Pascalu

```
program = naglowek-programu ";" blok-programu ".".  
naglowek-programu = "PROGRAM" identyfikator  
                [ "(" parametry-programu ")" ].  
parametry-programu = lista-identyfikatorow.
```

# Struktura programu w Pascalu

```
program = naglowek-programu ";" blok-programu ".".  
naglowek-programu = "PROGRAM" identyfikator  
                [ "(" parametry-programu ")" ].  
parametry-programu = lista-identyfikatorow.  
lista-identyfikatorow = identyfikator {"," identyfikator}.
```

# Struktura programu w Pascalu

```
program = naglowek-programu ";" blok-programu ".".  
naglowek-programu = "PROGRAM" identyfikator  
                [ "(" parametry-programu ")" ].  
parametry-programu = lista-identyfikatorow.  
lista-identyfikatorow = identyfikator {"," identyfikator}.  
blok-programu = blok
```

# Struktura programu w Pascalu

```
program = naglowek-programu ";" blok-programu ".".  
naglowek-programu = "PROGRAM" identyfikator  
                [ "(" parametry-programu ")" ].  
parametry-programu = lista-identyfikatorow.  
lista-identyfikatorow = identyfikator {"," identyfikator}.  
blok-programu = blok  
blok = sekcja-deklaracji-stalych  
      sekcja-definicji-typow  
      sekcja-deklaracji-zmiennych  
      sekcja-operacyjna.
```

# Struktura programu w Pascalu

```
program = naglowek-programu ";" blok-programu ".".  
naglowek-programu = "PROGRAM" identyfikator  
                [ "(" parametry-programu ")" ].  
parametry-programu = lista-identyfikatorow.  
lista-identyfikatorow = identyfikator {"," identyfikator}.  
blok-programu = blok  
blok = sekcja-deklaracji-stalych  
      sekcja-definicji-typow  
      sekcja-deklaracji-zmiennych  
      sekcja-operacyjna.  
sekcja-operacyjna = instrukcja-zlozona.
```



# Struktura programu w Pascalu

```
program = naglowek-programu ";" blok-programu ".".  
naglowek-programu = "PROGRAM" identyfikator  
                [ "(" parametry-programu ")" ].  
parametry-programu = lista-identyfikatorow.  
lista-identyfikatorow = identyfikator {"," identyfikator}.  
blok-programu = blok  
blok = sekcja-deklaracji-stalych  
      sekcja-definicji-typow  
      sekcja-deklaracji-zmiennych  
      sekcja-operacyjna.  
sekcja-operacyjna = instrukcja-zlozona.  
instrukcja-zlozona = "BEGIN" ciag-instrukcji "END".
```

# Struktura programu w Pascalu

```
program = naglowek-programu ";" blok-programu ".".  
naglowek-programu = "PROGRAM" identyfikator  
                [ "(" parametry-programu ")" ].  
parametry-programu = lista-identyfikatorow.  
lista-identyfikatorow = identyfikator {"," identyfikator}.  
blok-programu = blok  
blok = sekcja-deklaracji-stalych  
      sekcja-definicji-typow  
      sekcja-deklaracji-zmiennych  
      sekcja-operacyjna.  
sekcja-operacyjna = instrukcja-zlozona.  
instrukcja-zlozona = "BEGIN" ciag-instrukcji "END".  
ciag-instrukcji = instrukcja { ";" instrukcja }.
```

# Struktura programu w Pascalu

```
program = naglowek-programu ";" blok-programu ".".  
naglowek-programu = "PROGRAM" identyfikator  
                [ "(" parametry-programu ")" ].  
parametry-programu = lista-identyfikatorow.  
lista-identyfikatorow = identyfikator {"," identyfikator}.  
blok-programu = blok  
blok = sekcja-deklaracji-stalych  
      sekcja-definicji-typow  
      sekcja-deklaracji-zmiennych  
      sekcja-operacyjna.  
sekcja-operacyjna = instrukcja-zlozona.  
instrukcja-zlozona = "BEGIN" ciag-instrukcji "END".  
ciag-instrukcji = instrukcja { ";" instrukcja }.
```

```
PROGRAM krotki;  
BEGIN  
END.
```

```
PROGRAM podstawienie;  
VAR  wynik : BOOLEAN;  
     x,y   : REAL;  
     i     : INTEGER;  
BEGIN  
  wynik := (x = y) = (i < x);  
END.
```

```
PROGRAM podstawienie;  
VAR  wynik : BOOLEAN;  
     x,y   : REAL;  
     i     : INTEGER;  
BEGIN  
  wynik := (x = y) = (i < x);  
END.
```

**Struktury danych + Algorytm = Program**

# Indeks

- Algorytm
  - Przykładowy algorytm
  - Przykładowy algorytm
- Budowa systemu informatycznego
- Diagramy algorytmów
- Składnia programu
  - Notacja MBNF
    - Przykłady
    - Przykady cd.
  - Diagramy składni
    - Przykłady
- Kategorie składniowe Pascala
- Typy danych
  - Definicja typów danych
  - Zakres wartości
  - Ograniczanie zakresu
  - Przykłady typów wyliczeniowych i okrojonych
- Identyfikatory

- Zmienne
  - Deklaracja zmiennych
- Wyrażenia
  - Reguły wyliczania
  - Przykłady
- Operatory
  - Priorytety
- Instrukcje
  - Instrukcja przypisania
  - Zgodność typów
- Struktura programu w Pascalu
- Motto