Przykład programu
Rozwiązanie równania postaci:

$$a \cdot x^2 + b \cdot x + c = 0$$

```
program trojmian;
var a, b, c : real;
var delta , x1 , x2 : real;
begin
writeln('Podaj wspolczynniki a, b, c równania kwadratowego: ');
readln(a, b, c);       { wczytanie 3 współczynników }
if a = 0 then
   writeln('To nie jest równanie kwadratowe')
else
   begin
   delta := (b * b) - (4 * a * c);    {obliczenie delty }
   writeln ('Delta = ', delta);
   if delta < 0 then
      writeln('Brak rozwiazan')
   else if delta > 0 then
           begin     { dwa pierwiastki }
           x1 := (- b - sqrt(delta)) / (2 * a);
           x2 := (- b + sqrt(delta)) / (2 * a);
           writeln('X1 = ' , x1);
           writeln('X2 = ' , x2);
           end
        else begin    { jeden pierwiastek }
             x1 := (- b) / (2 * a);
             writeln('X1 = ' , x1);
             end;
   end;
end.
```

Podstawowe elementy występujące w zapisie programu w Pascalu:

Dopuszczane znaki w zapisach:

litery: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z _

cyfry: 1 2 3 4 5 6 7 8 9 0

znaki specjalne: odstęp + - * / = ^ < > ( )[ ] { } . , : ; ' # @

stosowane są także znaki sterujące (niewidoczne??) nowy wiersz, i in (o kodach ASCII od 0 do 31)

Przy pomocy wymienionych znaków i przy przestrzeganiu odpowiednich reguł tworzone są poszczególne elementy zapisu całego programu. Są to:

Słowa kluczowe
Nazwy
Stałe
Operatory

Przy tworzeniu tych jednostek zapisu stosowana jest zasada że małe i duże litery alfabetu są utożsamiane np. napisy:

begin DELTA readln

mogą być zapisane:

BEGIN delta ReAdLn

Zapisy ALA i ala oznaczają tę samą nazwę. Podobnie np. słowo kluczowe **for** może być zapisane: **FOR**.

Podstawowe definicje

Słowo kluczowe

Każdy z następujących napisów:

and, array, asm, begin, case, const, constructor, destructor, div, do, downto, else, end, file, for, function, goto, if, implemenation, in, inline, inherited, interface, label, mod, nil, not, object, of, or, packed, procedure, program, record, repeat, set, shl, shr, string, then, to, type, unit, until, uses, var, while, with, xor.

Słowa te służą do budowy zapisów języka.

Nazwa

Nazwą jest ciąg cyfr, liter i znaku _ zaczynający się od litery.

Nazwy służą do identyfikacji różnych obiektów definiowanych przez programistę wg potrzeb.

Stała: liczbowa, napisowa (w apostrofach) lub inna.

Operatory do zapisywania działań np. arytmetycznych

Pojęcia zmiennej, typu, deklaracji i instrukcji

W przedstawionym wyżej przykładzie zapisu algorytmu dla oznaczenia pewnych wielkości używamy zmiennych.
Pojecie zmiennej w programowaniu ma znaczenie podobne jak w matematyce i fizyce. Podobnie jak tam zmienne w Pascalu powinny mieć identyfikator (nazwę) i wartość.

Wszystkie zmienne występujące w zapisie programu musza być wstępnie zadeklarowane. Deklaracje zmiennych (i innych obiektów występujących w części wykonawczej programu) muszą występować w pierwszej (deklaracyjnej) części programu.

W jez. programowania z pojęciem zmiennej wiążą się:
- nazwa,
- typ,
- wartość - przed pierwszym nadaniem jest to wartość przypadkowa,

- miejsce przechowywania wartości – w pamięci operacyjnej komputera.

W deklaracji wymieniany jest identyfikator zmiennej oraz określany jest typ zmiennej. Typ jednoznacznie wyznacza zbiór wartości, które może przyjmować zmienna oraz zbiór operacji którym zmienna może podlegać.

Typ = zbior wartosci

W drugiej części programu rozpoczynającej się słowem BEGIN umieszczane są instrukcje opisujące czynności wykonywane na zadeklarowanych wcześniej zmiennych. W czasie wykonywania programu instrukcje są realizowane sekwencyjnie z góry w dół.
Deklaracje nazywane są czasem instrukcjami biernymi ponieważ ich "wykonywanie" ogranicza się jedynie do zarezerwowania odpowiedniego miejsca w pamięci komputera oraz przyporządkowania nazwy zmiennej do odpowiedniego adresu pamięci. Wielkość pamięci przydzielanej dla każdej zmiennej zależy jedynie od typu zmiennej, a nie np. od jej wartości.

Struktura programu

Dwie podstawowe części:

- deklaracyjna
- wykonawcza.

Składowe całego programu:

Nagłówek programu
Część deklaracyjna a w niej opis:
- typów,
- stałych,
- zmiennych,
- etykiet,
- procedur i funkcji.

Część wykonawcza
- begin
- instrukcje
- end

Kropka

Inaczej to samo:

Program nazwa; { nagłówek można opuścić }

{ deklaracje w skład których mogą wchodzić sekcje deklaracji:
- stałych rozpoczynające się słowem: CONST
- zmiennych VAR
- typów TYPE
- procedury PROCEDURE
- funkcje FUNCTION }

begin

{ instrukcje }

end.

Deklaracje

Typy standardowe:

Całkowite: niewielki zakres, niewielka liczność zbioru wartości, dokładne odwzorowanie liczb całkowitych z określonego zakresu

Typ	zakres	sposób zapamiętania
shortint	-128..127	1 bajt
integer	**-32768 32767**	**2 bajty (z bitem znaku)**
longint	-2147483648..+2147483647	4 bajty
byte	0..255	1 bajt
word	0..65535	2 bajty

Przykłady stałych: -5, 1234

Na wartościach całkowitych mogą być wykonane następujące operacje:

+	dodawanie
-	odejmowanie
*****	mnożenie
div	dzielenie całkowite
mod	branie reszty z dzielenia całkowitego

Wartości całkowite można porównywać operatorami relacji:

< = > <= >= <>

Funkcje standardowe o wartościach całkowitych:

a) o argumencie całkowitym:

abs(i)	moduł | i |
sqr(i)	kwadrat i
succ(i)	nastepnik = i+1
pred(i)	poprzednik = i-1

b) o argumencie rzeczywistym:

round(r)	najbliższa liczba całkowita (zaokrąglenie) round(4,8) = 5 round(-4,8)=-5
trunc(r)	najbliższa całkowita w kierunku zera (obcięcie) trunc(4,8) = 4 trunc(-4,8)=-4

Typ rzeczywisty

Typ rzeczywisty ma przeliczalny zbiór wartości będący podzbiorem zbioru liczb rzeczywistych. Wartości te mogą być przedstawiane w notacji zmiennoprzecinkowej z ustalona liczba cyfr znaczących.
W Turbo Pascalu predefiniowane zostało pięć typów rzeczywistych, różniących się miedzy sobą zakresem wartości i liczbą cyfr znaczących (co wynika ze sposobu pamiętania liczb).

typ	zakres	liczba cyfr znaczacych	rozmiar w bajtach
real	**$2.9*10^{-39}..1.7*10^{38}$**	**11 - 12**	**6**
single	$2.9*10^{-39}..1.7*10^{38}$	7-8	4
double	$5.0*10^{-324}..1.7*10^{308}$	15-16	8
extended	$3.4*10^{-4932}..1.1*10^{4932}$	19-20	10
comp	$9.2*10^{18}..9.2*10^{18}$	19-20	8

Przykłady - zapisy stałych rzeczywistych:

1.2 -123.1234 -12.12E-2 1e5

Operacje na wartościach rzeczywistych:
+ dodawanie
- odejmowanie
* mnożenie
/ dzielenie

UWAGA:
W Pascalu nie ma operatora potęgowania. Do obliczeń można wykorzystać zależność:

$$x^b = \exp(b * \ln(x))$$

Funkcje standardowe o wartościach rzeczywistych:

abs(x)	$\lvert x \rvert$
sqr(x)	x^2
sqrt(x)	$x^{0.5}$
ln(x)	ln x
exp(x)	e^x
sin(x)	sin x
cos(x)	cos x
arctan(x)	arc tg x
Frac(x)	część ułamkowa argumentu
Int(x)	część całkowita argumentu
pi	zwraca wartość pi=3.141592653589793238

Typ znakowy - char

Zbiorem wartości tego typu (mającego identyfikator CHAR) jest zbiór znaków, uporządkowany zgodnie z rozszerzonym zbiorem znaków ASCII.
Stałą znakowa jest każdy napis znakowy o długości 1.

Stała znakowa:

‘ znak ‘

Przykłady: ‘a’ ‘Z’ ‘*’

Funkcje znakowe:

chr(n) – znak o kodzie n.
ord(z) – wartość kodu znaku z

Typ logiczny – boolean

Przeznaczony dla wartości logicznych. Obejmuje zbiór 2 wartości. Używa się identyfikatorów stałych logicznych:

true prawda logiczna
false fałsz logiczny

Jedna z funkcji o wyniku logicznym to:

odd(i) - wynik **true** jeśli argument całkowity jest nieparzysty

Operatory logiczne:

not negacja
and iloczyn logiczny
or suma logiczna
xor nierównoważność (exclusive or)

Wynik operacji relacji (porównania) jest logiczny.

Typ łańcuchowy – string

Typy łańcuchowy (napisowy) wykorzystywany jest do reprezentowania napisów składających się z dowolnych znaków alfabetu (z zestawu 256 znaków ASCII).
Stałe tego typu są napisami w apostrofach.

Operator napisowy:

\+ - łączenie napisów (konkatenacja)

Istnieje bardzo dużo funkcji dotyczących napisów. Np.:

lenght(napis) - długość (liczba znaków) napisu

Z wykorzystaniem poznanych typów standardowych można przedstawić przykładową redakcję deklaracji zmiennych w postaci:

```
var  i, j :  integer;
     k : word;
     znak : char;
     x, y : real;
     l: boolean;
     napis: string;          { długość domyślna 255 znaków }
     krotki: string[10];     { ograniczenie długości łańcucha do 10 znaków }
```

Można odnosić się do poszczególnych znaków łańcucha. Np.:

krotki[2]	drugi znak
napis[i+1]	znak o numerze i+1

Przyjmuje się, że po zadeklarowaniu zmienne przyjmują wartości przypadkowe. Przez użyciem zmiennej w obliczeniach należy zmienną zainicjować instrukcją przypisania lub poprzez wczytanie wartości zmiennej.

Wyrażenia

Wyrażeniem nazywamy zapis obliczeń składający się ze stałych, zmiennych (będących argumentami) i łączących je operatorów.
Ponadto w skład wyrażeń może wchodzić obliczanie wartości jakiejś funkcji (wraz z wyznaczeniem wartości jej argumentów).
W wyrażeniach mogą występować elementy różnych typów, jednak poprawne wyrażenia to takie, w których każdy z operatorów działa na argumentach o właściwym typie (np. oba argumenty operatora **div** są całkowite).
Typem wyrażenia nazywamy typ wartości wynikowej wyrażenia.
Kolejność wykonywania działań wskazują nawiasy, a poza nimi priorytet operatorów - tj. ustalone reguły pierwszeństwa.
Priorytet operatora zależy od klasy, do której należy dany operator.

Główne poznane dotychczas operatory należą do dwóch klas:

- operatory multiplikatywne: *** div mod / and**
- operatory addytywne: **+ - or xor**

Operatory multiplikatywne maja pierwszeństwo przed addytywnymi.

Operacje o tym samym priorytecie realizowane są w kolejności zapisu (od lewej do prawej).
Obowiązują także dwie zasady:

- wszystkie operatory musza być podane jawnie
- przy zapisie funkcji argumenty należy podawać w nawiasach.

UWAGI:

- najprostszym rodzajem wyrażenia jest pojedyncza stała lub zmienna
- wyrażenia służą nie tylko do zapisywania obliczeń arytmetycznych, ale także do obliczania wartości logicznych, przekształcania tekstu itd. - pojecie wyrażenia zostanie wiec znacznie rozszerzone.

Przykłady wyrażeń:

2*a+3*b a*sin(x)/(sqr(b)+sqr(c)) a/b*c

(a>0) and (b>0)

Kolejność priorytetów wszystkich operacji – od najwyższego:

nawiasy i obliczenie funkcji
not
*** / div mod and**
+ - or xor
= <> < <= > >=

Instrukcje proste

Instrukcja przypisania

<zmienna> **:=** <wyrażenie>

Instrukcja ta jest wykonywana w dwu krokach:
- wyznaczanie wartości wyrażenia
- przypisanie obliczonej wartości do zmiennej

Typy Zmiennej i Wyrażenia muszą być zgodne (w sensie przypisania). Dopuszczalne jest przypisanie wartości całkowitej pod zmienną rzeczywistą. Dokonywana jest konwersja.

Instrukcje czytania

read (<lista zmiennych>)

readln(<lista zmiennych>)

readln

nazw zmiennych których wartości ma wprowadzić (z klawiatury) użytkownik programu należy oddzielać przecinkami. Można wczytywać wartości typu całkowitego, rzeczywistego, znakowego i napisowego.

Przykład

read(a, b, c); readln(z, y, liczba);

Instrukcja **readln** nie zakończy się jeśli po podaniu wszystkich wartości nie naciśniemy Enter

Instrukcje drukowania (wyświetlania na ekranie)

write(lista wyrażeń)

writeln(lista wyrażeń)

writeln

W nawiasach należy podawać listę wyrażeń oddzielanych przecinkami.
Instrukcja wyznacza wartości wyrażeń i wyświetla te wartości na ekranie.
Instrukcja writeln po wyświetleniu wszystkich wartości wyprowadza nowy wiersz.

Przykłady

```
Var     nap : string;                { długość domyślna 255 znaków }
        lan : string[10];            { ograniczenie długości łańcucha do 10 znaków }
        znak: char;
        x, y: real;
        i: integer;

. . . . .

nap := ‘Ala ma kota’;                { Użycie stałej napisowej }
readln(lan);
znak := nap[1];
writeln(‘Litera A=’, znak, ‘ Litera m=’, nap[5],’ Napis = ‘, nap);
writeln(‘lan=’, lan);
i := 1;
i := i+1;
y := 2.5;
x := i+y;
writeln(‘ Wartosc x=’, x, ‘ Wartosc y=’, y:8:2, ‘ Wartosc i=’, i:8);
```

Przykład – obliczanie obwodu i pola koła

```
var r, obw, p: real;
begin
writeln('Podaj promien kola:');
readln(r);
obw := 2*pi*r;
p := pi*r*r;
writeln('Obwod kola=', obw);
writeln('Pole kola=', p);
readln;
end.
```

Oczywiście obliczenia można zorganizować inaczej:

```
obw := pi*r;
p := obw*r;
writeln('Obwod kola=', 2*obw:8:2);
writeln('Pole kola=', p:8:2);
```

Przykład – obliczanie obwodu i pola trójkąta na podstawie 3 boków

```
var a, b, c, p, pole, obw: real;
```

```
begin
writeln('Podaj 3 boki trojkata:');
readln(a, b, c);
obw := a+b+c;
p := obw/2;
pole := sqrt(p*(p-a)*(p-b)*(p-c));
writeln('Obwod trojkata=', obw:8:2);
writeln('Pole trojkata=', pole:8:2);
readln;
end.
```

Uruchomienie dla 1, 2, 3 powoduje błąd (dlaczego?).

Instrukcja grupująca (złożona)

Składnia:

begin <*zestaw instrukcji*> **end**

Przy pomocy nawiasów **begin** oraz **end** możemy połączyć kilka instrukcji i uczynić z nich jedną instrukcję złożoną.
Część wykonawcza programu jest instrukcją grupującą.
Wewnątrz instrukcji grupującej można umieszczać inne instrukcje grupujące.

Instrukcja warunkowa

Składnia:

if <*wyrażenie*> **then** <*instrukcja1*>;

lub

if <*wyrażenie*> **then** <*instrukcja1*>
else <*instrukcja2*>

W obu przypadkach wykonanie instrukcji jest uzależnione od spełnienia warunku określonego *wyrażeniem*. W wariancie pierwszym *instrukcja1* jest wykonywana gdy warunek jest spełniony tj. *wyrażenie* ma wynikową wartość logiczną **true**. Jeżeli wartość ta jest **false** to nie wykonuje się żadnych działań.
Drugi wariant instrukcji **if** powoduje wykonanie *instrukcja1* jeśli *wyrażenie* ma wartość **true** (tj. gdy warunek zapisany w wyrażeniu jest spełniony) lub *instrukcja2* w przeciwnym przepadku.
Instrukcje **if** można wykorzystywać w sposób zagnieżdżony – jedną z instrukcji wykonywanych warunkowo może być inna instrukcja **if**. Patrz przykład poniżej.

Przykłady

```
if a = 0 then
   writeln('To nie jest równanie kwadratowe')

if a<0 then a := -a;
```

```
if   a>b  then wiekse := a
          else wiekse := b;


if delta < 0 then writeln('Brak rozwiazan')
else if delta > 0 then
        begin
        x1 := (- b - sqrt(delta)) / (2 * a);
        x2 := (- b + sqrt(delta)) / (2 * a);
        writeln('X1 = ' , x1);
        writeln('X2 = ' , x2);
        end
     else begin
          x1 := (- b) / (2 * a);
          writeln('X1 = ' , x1);
          end;
```

Przykład programu – ile dodatnich (z dwu)

```
var a, b: real;
begin
writeln('Podaj dwie liczby: ');
readln(a, b);
if (a > 0) and (b > 0) then
   writeln('Sa 2 liczby dodatnie')
else if (a <= 0) and (b <= 0) then
                      writeln('Nie ma liczb dodatnich')
     else writeln('Jest jedna liczba dodatnia');
readln;
end.
```

Inny zapis:

```
var a, b: real;
    ile: integer;
begin
writeln('Podaj dwie liczby: ');
readln(a, b);
ile := 0;
if (a > 0) then ile := ile+1;
if (b > 0) then ile := ile+1;
writeln('Ilosc liczb dodatnich: ', ile);
readln;
end.
```

Zapis drugi jest lepszy ponieważ łatwo go rozbudować do przypadku trzech lub więcej liczb.

Przypadek N liczb wymaga pętli

Trzy rodzaje pętli:

Pętla repeat

Składnia:

repeat <*instrukcje*> **until** <*warunek_końca*>;

Instrukcje są powtarzane, aż do chwili gdy warunek będzie spełniony.

Przykład

```
suma := 0;
writeln(‘ Podawaj liczby dodatnie. Zakończ liczbą zero );
repeat
    readln(a);
    suma := suma + a;
until a <> 0;
writeln(‘Suma=’, suma:8:2);
```

Przykład programu

Rozwiązywanie równania COS(X) – X = 0 metodą iteracji prostej

```
var
    x, x0: REAL;
begin
read(x);
repeat
  x0 := x;
  x:= COS( x );
until ABS( x - x0 ) < 0.01;
writeln ( 'Pierwiastek rownania = ', x:8:5 );
end.
```

Przybliżenie początkowe przyjąć 0.

Pętla while

Składnia:

while <*warunek kontynuacji*> **do** <*instrukcja*>

Instrukcja jest wykonywana kolejny raz jeśli warunek jest spełniony

Przykład

```
suma := 0;
writeln(‘ Podawaj liczby dodatnie. Zakończ liczbą zero );
readln(a)
while a <> 0 do
    begin
```

```
    suma := suma + a;
    readln(a);
    end;
writeln('Suma=', suma:8:2);
```

Pętle **while** i **repeat** stosowane są, gdy liczba przebiegów pętli jest nieznana, a można określić warunek jej zakończenia.
Każdy algorytm zapisany pętlą **while** można łatwo przekształcić do zapisu przy pomocy pętli **repeat** i odwrotnie.

Przykład programu

Obliczanie ln(2) wg wzoru:

$$1 - 1/2 + 1/3 - 1/4 + ... = \ln(2)$$

```
var i: integer;
    i, suma, znak: real;
begin
znak := -1;
suma := 0;
i := 1;
while i < 100 do
   begin
   znak := - znak;
   suma := suma + 1 / i * znak;
   i := i + 1
   end;
writeln ( 'LN(2)=',suma ,' Dokladnie=' ,ln(2));
end.
```

Pętla for

Składnia:

```
for <zmienna> := <wyr_pocz> to <wyr_kon> do <instrukcja>;

for <zmienna> := <wyr_pocz> downto <wyr_kon> do <instrukcja>;
```

Instrukcja jest wykonywana po raz pierwszy dla wartości *zmiennej* równej *wyr_pocz*, a następnie powtarzana wielokrotnie dla kolejnych wartości zmiennej. Po raz ostatni *instrukcja* jest wykonywana przy wartości *zmiennej* równej *wyr_kon*.
Instrukcja w wariancie drugim realizowana jest przy malejących wartościach *zmiennej*. W tym przypadku powinno zachodzić *wyr_pocz* > *wyr_kon*.

Przykłady

```
for i := 1 to n do
   writeln( 'i=', i, ' i*i=', i*i);

readln(a, b);
```

```
krok:=(b-a)/10;
writeln('Tablicowanie funkcji sin(x) w przedziale [' ,a:5:2,
                          ', ',b:5:2, '] z krokiem ', krok:5:2);
for i := 0 to 10 do
   begin
   x:=a+i*krok;
   writeln( 'sin(', x:5:2, ')=', sin(x):5:2);
   end;
```

Przykłady programów

Dane jest N liczb. Ile z nich jest dodatnich.

```
var n, i, ile: integer;
    a: real;
begin
write('Podaj ilosc liczb. N=');
readln(n);
ile := 0;
for i:=1 to n do
   begin
   write('Podaj kolejna liczbe. A=');
   readln(a);
   if a>0 then ile := ile + 1;
   end;
writeln ( 'Wsrod ',n ,' liczb jest ', ile, ' dodatnich');
readln;
end.
```

Obliczenie średniej

```
var n, i: integer;
    a, suma: real;
begin
write('Podaj ilosc liczb. N=');
readln(n);
suma := 0;
for i:=1 to n do
   begin
   write('Podaj kolejna liczbe. A=');
   readln(a);
   suma := suma +a;
   end;
writeln ( 'Suma=', suma:8:2);
writeln ( 'Srednia=', suma/n:8:2);
readln;
end.
```

Wyznaczenie wartości maksymalnej ciągu.

```
var n, i: integer;
```

```
    a, max: real;
begin
write('Podaj ilosc liczb. N=');
readln(n);
write('Podaj pierwsza liczbe. A=');
readln(max);
for i:=2 to n do
   begin
   write('Podaj kolejna liczbe. A=');
   readln(a);
   if max<a then max:=a;
   end;
writeln ( 'Liczba najwieksza = ', max:8:2);
readln;
end.
```