Podstawy Programowania

Ogłoszenia

Ocena końcowa

- Kartkówki (9 p.)
- Projekt (21 p.)
- Kolokwium (30 p.)

- Aktywność

Ocena końcowa

- [0-29] – 2
- [30-39] – 3
- [40-44] – 3,5
- [45-49] – 4
- [50-54] – 4,5
- [55-60] - 5

Książki

- Turbo Pascal. Programowanie
  Autor: Tomasz M. Sadowski
- Praktyczny kurs Turbo Pascala.
  Autor: Tomasz M. Sadowski
- Turbo Pascal 7.0 z elementami programowania
  Autor: Andrzej Marciniak
- PASCAL - Programowanie w Pascalu
  Autor Marek Gierliński

Materiały

- http://turbopascal.helion.pl
- http://programowanie.tnb.pl/viewpage.php?page_id=1
- http://www.centrumpascal.republika.pl/
- http://pascal-programming.info/lesson1.php
- http://www.pascal.eu.org/

Środowisko programistyczne

- Lazarus
- http://www.lazarus.freepascal.org/index.php?page=downloads

- Windows / Linux

- Dream Spark

Język programowania

- Symbole
  - Zmienne, struktury, procedury, funkcje
- Słowa kluczowe
- Syntaktyka (składnia)
  - Reguły opisujące dozwolone konstrukcje
    - struktury sterujące,
    - struktury danych,
    - Zasady tworzenia poprawnych symboli,
    - interpunkcja
- Semantyka
  - znaczenie i zasady interpretacji poprawnych składniowo wyrażeń

- Język programowania wysokiego poziomu stworzony przez Niklausa Wirtha w 1971 r
- Nauka programowania strukturalnego.

program nazwa_programu; {opcjonalne}

{sekcja deklaracji}

begin

{punkt wejścia}

end.

{koniec programu }

Sekcja deklaracji

- deklaracja modułów;
  - USES
- deklaracja etykiet;
  - LABEL
- deklaracja stałych;
  - CONST
- deklaracja typów;
  - TYPE
- deklaracja zmiennych;
  - VAR
- deklaracja podprogramów;
  - PROCEDURE
  - FUNCTION

```
begin
   pierwsza instrukcja;
   druga instrukcja;
   ...
end.
```

```
program HelloWorld;
   begin
        writeln('Hello world');
   end.
```

Komentarze

- (* komentarz – stara składnia *)
- { komentarz }
- // jednoliniowy

- Niepoprawne: (* komentarz }

Słowa kluczowe

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

Identyfikator

- Ciąg znaków z następującego zbioru:
  - litery a-z, A-Z
    - duże i małe litery nie są rozróżniane w Pascalu, w innych językach czasami są
  - cyfry 0-9
  - znak podkreślenia _

Identyfikator

- Pierwszym znakiem identyfikatora nie może być cyfra
- Identyfikator nie może być identyczny z żadnym ze słów kluczowych języka Pascal (keyword)
- Identyfikator zdefiniowany przez użytkownika nie powinien być identyczny z nazwą istniejącego podprogramu z biblioteki (np. write)
- Specyfikacja języka Pascal nie nakłada żadnego ograniczenia na długość identyfikatora, jednak realizacje wykorzystują limitowane liczby znaków

Identyfikator

Zmienna **zmienna** **Id** **_id** **Id2** **Tablica**	**2zmienna** **3343** **&zmienna** **‘zmienna** **Zmienna’** **Begin** **Function**

Zmienna

- Co to jest zmienna
  - Wskazanie na pewien wydzielony obszar pamięci
    - Ustalony rozmiar
    - Posada identyfikator
    - Może przechowywać pewną wartość (zależną od typu)

Zmienna

- var nazwa_zmiennej: typ_zmiennej;

```
var l:integer;
var s, s1:string;
z:char;
```

```
program Nazwa_programu;
   var
        i:integer;
        s,s1:string;
        c:char;

   begin
        ...
   end.
```

- nazwa_zmiennej := nowa_wartość;

- nowa_wartość
  - stała

  i := 1;
  - wyrażenie

  i := i + 1;

```
Program przyklad_1;
var
    k:integer;
begin
      k:=5;
      k:=k+k;
      writeln (k);
End.
```

Jaka wartość ?

Czy writeln wymagany ?

```
Program przyklad_1;
var
   k:integer;
begin
     k:=k+k;
     writeln (k);
End.
```

Jaka wartość ?

Czy writeln wymagany ?

Typy danych

- W matematyce zmienne klasyfikuje się zgodnie z ich pewnymi własnościami
- W programowaniu klasyfikacja zmiennych jest również bardzo ważna
- Typ danych charakteryzowany jest przez
  - Zbiór wartości W
  - Zbiór operacji O

  T = (W, O)

Ogólny podział typów

- Typy proste
  - typy porządkowe
    - wyliczeniowy
    - okrojony
    - całkowity
    - logiczny
    - znakowy
  - typy rzeczywiste
- Typy złożone (strukturalne)
  - tablice
  - rekordy
  - zbiory
  - plik
- Typ łańcuchowy
- Typ wskaźnikowy
- Typ obiektowy

Typy porządkowe

- Definiują uporządkowany zbiór wartości
- Każda wartość (oprócz pierwszej) ma unikalnego poprzednika
- Każda wartość (oprócz ostatniej) ma unikalnego następnika
- Każda wartość posiada swój numer porządkowy
- Relacja uporządkowania

Typy porządkowe

- Podzbiór zbioru liczb całkowitych
- Liczba porządkowa elementu typu całkowitego jest równa wartości tego elementu
- Dla kompilatora Delphi 32-bitowego

integer [-2147483648 .. 2147483647] (32-b ze znakiem)
cardinal [0 .. 294967295] (32-b bez znaku)
shortint [-128..127] (8-b ze znakiem)
smallint [-32768..32768] (16-b ze znakiem)
longint [-2147483648..2147483647] (32-b ze znakiem)
byte [0..255] (8-b bez znaku)
word [0..65535] (16-b bez znaku)
... ...

stałe MaxInt i MaxLongint

Typy porządkowe

- Typy porządkowe
  - Ord – nr porządkowy
  - Pred – poprzednik
  - Succ – następnik

- Dec
  Jeżeli x jest zmienną całkowitą to
  Dec(x) ⇔ x := x – 1
  Dec(x, N) ⇔ x := x –N
- Inc
- Analogicznie jak Dec
  Inc (I)⇔ I := Succ (I)

- Low – najmniejsza wartość
- High – największa wartość

- Odd – badanie parzystości (działa tylko dla typów całkowitych)

- Boolean(1) = true; integer(7)=7;

Zakres

```
program przekraczanie_zakresu;  {Przetestować w domu}
    var
        i: shortint;
        y: smallint;
        x: integer;
    begin
        i := High(shortint);
        i := i + 1;
        writeln(i);
        x := 40000;
        y := x;
        writeln(y);
        readln;
    end.
```

```
program zakres_przekroczony;
var
    x: longint;
    y: shortint;
begin
    { x := MaxLongint;      }
    y := MaxLongint; { kompilator: błąd }
    writeln(y)
end.
```

- Kompilator wykrywa tylko niektóre błędy

```
program zakres_przekroczony2;
var
   x: longint;
   y: shortint;
begin
   x := MaxLongint;
   y := x; { kompilator: poprawnie }
   writeln(y) {wypisze się: -1}
end.
```

```
program zakres_przekroczony3;
var
   x: longint;
begin
   x :=MaxLongInt +100; { kompilator: błąd }
   writeln(x)
end.
```

```
program zakres_przekroczony4;
var
   x: longint;
begin
   x:= MaxLongInt;
   x := x+100; { kompilator: poprawnie }
   writeln(x) {wypisze się: -2147483549}
end.
```

Zapisywanie liczb całkowitych

- Ciąg cyfr dziesiętnych, ewentualnie poprzedzony znakiem “-” (minus)
  - 123, -84, 0, 789, 00004
- ciąg cyfr szesnastkowych poprzedzony znakiem “$”
  - $10 (czyli dziesiętnie 16),
  - $ff, $FF (255),
  - $5 (5)

Typ wyliczeniowy

- Przy jego definicji podajemy wszystkie możliwe wartości
- Definicja:
- type nazwa_typu = opis_typu;
- type nazwa_typu =
- (pierwszy_identyfikator, drugi, ..., ostatni);

type

dayofweek = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);

Wartościami mogą być identyfikatory (zasady nazewnictwa zmiennych).

```
var
d: dayofweek;
…
d := Mon;
```

Na zmiennych tego typu nie można wykonywać operacji arytmetycznych, ale jedynie operacje typowe dla typów porządkowych (ord, pred i succ).

- Porządek w typie jest zgodny z kolejnością wyliczenia w definicji
- Wartości typu wyliczeniowego są liczbami porządkowymi 0, 1, 2, .
- *enumerated type*
- *Konwersja*
  - *NazwaTypu(wartość)*

Typ wyliczeniowy

- Przykład:

```
program typ_ program;
    type
        typDniTyg = (pn, wt, sr, cz, pt, so, ni);
    var
        d: typDniTyg ;
        nr:integer;
    begin
        d := pn;
        d := succ(d); { wtorek};
        nr:= ord(pn);
        d:=typDniTyg(1);
    end .
```

Typ okrojony

- Jeśli zmienna przyjmuje wartości tylko z pewnego przedziału
- Definicja:
- type nazwa_typu = stała1 .. stała2;

```
type
    year = 1900..2000;
    letter = 'a'..'z';
typPoryRoku = (wiosna, lato, jesien, zima);
Nad0PR     = wiosna .. Jesien;
```

- Zmienne powyższych typów nie mogą otrzymywać wartości spoza wyznaczonego zakresu, co pozwala na lepszą kontrolę błędów wykonania programów.
- Podzbiór zakresu wartości innego typu porządkowego
- Typów okrojonych nie można tworzyć w oparciu o zakres liczb rzeczywistych, np. zakres 1.5..2.5 jest niepoprawny.
- Deklaracja tablic

```
var
    ilosc: array[year] of longint;
    licznik: array[letter] of integer;
```

- *subrange type*

Typ okrojony

type nazwa_typu = stała1 .. stała2;

stała1 < stała2 obie tego samego typu porządkowego

Typy anonimowe

- Zamiast

  type

  dtyg = (pn, wt, sr, cz, pt, so, ni);

  var

  d: dtyg;

Można nie definiować nazwy typu

var

d: (pn, wt, sr, cz, pt, so, ni); {OK.}

ee: (pn, wt, sr, cz, pt, so, ni); {błąd powtórna deklaracja}

Zgodność typów

- Dwa typy są zgodne jeżeli:
  - są to typy takie same
  - jeden z typów jest okrojonym typem drugiego albo obydwa są okrojonymi typami tego samego typu pierwotnego
  - obydwa są typami zbiorowymi o zgodnych typach podstawowych

Typ znakowy

- Typ znakowy
  - char
- W Delphi AnsiChar i WideChar
- char ⇔AnsiChar
- char [#0,..., #255]

Typ znakowy

- Zapisywanie znaków
  - pojedynczy znak ujęty w apostrofy
    - ’a’, ’0’, ’+’, ’#’, ’’’’
  - kod ASCII znaku, poprzedzony znakiem „#”
    - #0, #1, #10, #13, #27, #65 (to samo co ’A’), #255

Typ znakowy

- Podzbiory liter i cyfr
- Uporządkowane, spełniają zasady:
- X jest literą jeżeli:
  - (('A' <= X) and (X <= 'Z'))
  - or
  - (('a' <= X) and (X <= 'z'))
- X jest cyfrą jeżeli:
  - ('0'<= X) and (X <= '9’)

Typ znakowy

- Każdemu znakowi odpowiada liczba naturalna z zakresu 0−255 (kod ASCII).
- Ord określa kod znaku
- Chr na podstawie kodu określa znak

```
program kod_ASCII;
   var c: char;
begin
   read(c);
   writeln(ord(c))
end.
```

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Typ znakowy

- ord(c) - kod znaku
- chr(i) - znak na podstawie kodu

- Funkcja **ord** jest odwrotna do chr
  - ord((chr(i)) = i {i:integer; liczba}
  - chr(ord(c)) = c {c:char; znak}
  - chr(103) -> 'g'
  - ord('a') -> 97

Typ znakowy

- Szczególnie warte uwagi są funkcje:
- f(c) = ord(c) - ord('0')
  - pozycja cyfry c wśród cyfr ('0' <= c <= '9')
- g(i) = chr(i + ord('0'))

  i-ta cyfra (0 <= i <= 9)
- Przykłady:
  - f('8') = 8
  - g(4) = '4‘
  - f(g(i)) = i
  - g(f(c)) = c

```
program konwersja;
var
    x: integer;
    c: char;
begin
    writeln(‘podaj cyfre’);
    readln(x);
    c := chr(x+ord(‘0’));
    writeln(c);
end.
```

Typ logiczny

- boolean [FALSE, TRUE]
  - (W Delphi również ByteBool, WordBool, LongBool)
- Dla wartości typu boolean:
  - False < True
  - Ord(False) = 0
  - Ord(True) = 1
  - Succ(False) = True
  - Pred(True) = False

Typy rzeczywiste

- Definiują pewien podzbiór liczb rzeczywistych
- Dla kompilatora Delphi 32-bitowego

real [5.0e-324..1.7e308] (15.. 16 cyfr dziesiętnych, 8B)

real48 [2.9e-39..1.7e38] (11 .. 12 cyfr dziesiętnych, 6B)

single [1.5e-45..3.4e38] (7 .. 8 cyfr 4B)

double [5.0e-324..1.7e308] (15..16 cyfr , 8B)

extended [3.6e-4951..1.1e4932] (19..20 cyfr, 10B)

comp [-2e63 + 1..2e63 + 1] (19..20 cyfr, tylko całkowite , 8B)

Typy rzeczywiste

- Zapisywanie liczb rzeczywistych
- Zapis dziesiętny
  - .14, -2.71828, 2.71828, 4.0, 4, -9999.999
- Zapis wykładniczy
- <mantysa>E<cecha>

  =mantysa*10 (cecha)

  0.345E-5 (=0,00000345 =0,345*10-5))
  345E6
  3.45E+8

Konwersja typów

- Konwersje typów liczbowych
  - automatyczna konwersja
    - całkowita --> rzeczywistej
  - brak konwersji automatycznej
    - rzeczywista --> całkowita
      - trunc (l_rzecz) obcina miejsca po przecinku
        - » trunc(37.85)=37
      - round(l_rzecz) zaokrągla liczbę
        - » round(37.85)=38

Konwersja całkowita -> rzeczywista

```
program konwersja1;
var
   x: integer;
   y: real;
begin
   x:=345;
   y := x;   { kompilator: poprawnie }
end.
```

Konwersja rzeczywista -> całkowita

```
program konwersja2;
var
   x: integer;
   y: real;
begin
   y:=345;
   x := y;   { kompilator: błąd }
end.
```

Zbiory

- Zbiór oznacza się wyliczając jego elementy i zamykając ich listę w [ ]
- []– zbiór pusty
- [2, 8]– zbiór zawierający liczby 2 i 8
- [2 .. 8]– zbiór zawierający liczby od 2 do 8
- [’a’.. ’z’]– zbiór małych liter alfabetu
- [’0’.. ’9’]– zbiór cyfr

Typ zbiorowy

- type T = set of T0;
- Typem podstawowym T0 może być może być tylko typ porządkowy
- W TP i Delphi tylko typ okrojony o granicach należących do przedziału 0..255 albo typ
- wyliczeniowy o liczbie elementów nie większej niż 256
- Wartości typu T --> podzbiory wartości typu T0
- T jest zbiorem wszystkich podzbiorów typu T0

Typ zbiorowy

- Przykład:
  - type T = set of 2..4;
- Zbiór wartości typu T:
- [] [2] [3] [4] [2,3] [2,4] [3,4] [2,3,4]
- Zawiera dwa charakterystyczne zbiory:
  - [] - zbiór pusty zbiór pusty
  - [2,3,4] - zbiór podstawowy
- Zbiór traktowany jest jako jednostka niepodzielna.

  Nie można odwoływać się do poszczególnych elementów typu zbiorowego

Typ zbiorowy

- Mając deklaracje:
  - type zbiór_liczb = set of 0..30;
  - type zbiór_znaków = set of char;
  - type dni_tygodnia = (PO,WT,SR,CZ,PT,SO,NI);
  - type zbiór_dni = set of dni_tygodnia;
- ... można zdefiniować zmienne:
  - zc: zbiór_liczb;
  - zz: zbiór_znaków;
  - zd: zbiór_dni;
- ... i dokonać przypisań:
  - zc := [2, 4, 6, 8, 10];
  - zz := [‘a’, ‘b’, ‘c’];
  - zd := [PO, SR, PT];

Typ zbiorowy - operacje

- type T = set of T0;
- var A, B, C: T;

- C := A + B;
- C := A * B;
- C := A;
- C := A – B;
- A = B
- A <> B
- A <= B
- A >= B
- el in B

Typ zbiorowy - przykład

```
type
    t_okrojony= 0..20;
var
    a : set of t_okrojony;   {21potencjalnych elementów}
    i : t_okrojony;
begin
    a := [4,9]; { 4 9 }
    a := a + [1,4,5];  { suma  suma zbiorów     zbiorów } { 1 4 5 9 }
    a := a * [1,2,9];  {przecięcie } { 1 9 }
    a := a - [3,9]; { różnica     różnica } { 1 }
end.
```

Typ zbiorowy - przykład

Znaleźć najmniejszy element niepustego zbioru A

```
program zbiory1;
const N = 100;
var
        x: 1..N;
        A: set of 1..N;
begin
  A := [10, 16, 12, 3, 8];
  x := 1;
while not (x in A) do
        x := succ(x);
write(x);
end.
```

Wyrażenia

- Wyrażenie → formuła lub reguła obliczania, wyznaczająca pewną wartość
- Wyrażenie → operandy, operatory
- Operatory
  - Jednoargumentowe (unarne)
  - Dwuargumentowe (binarne)
- Wyrażenia
  - Arytmetyczne
  - Logiczne
  - Wskaźnikowe

Priorytet operatorów

Operatory	Priorytet	Kategoria
@, not, +, -	(najwyższy)	unarne
*, /, div, mod, and, shl, shr		multiplikatywne
+, -, or, xor		addytywne
=, <>, <, <=, >, >=, in	(najniższy)	relacyjne

Zasady wyznaczania wartości wyrażeń

1. Wartości podwyrażeń ujętych w nawiasy okrągłe, poczynając od najbardziej zagnieżdżonych
2. Jeżeli wyrażenie nie zawiera nawiasów, to jego wartość wyznacza się zgodnie z priorytetem operatorów

$X - Y * Z \Leftrightarrow X - (Y * Z)$

$(X - Y) * Z$

$X = Y \text{ or } X = Z \Leftrightarrow (X = (Y \text{ or } X)) = Z$

$(X = Y) \text{ or } (X = Z)$

Typy całkowite - operatory

- Arytmetyczne

  + (dodawania)

  - (odejmowania)

  * (mnożenia)

  / (dzielenia) ➔ wynik typu REAL;

  Div (dzielenie całkowite) ➔ 7 div 3 = 2

  Mod (modulo) ➔ 7 mod 3 = 1

Typy całkowite – operatory cd.

- Relacji:

  <, <=, =, >, >=, <> ➔ wartość wyrażenia jest typu logicznego

- Logiczne:

  `not, and, or, xor, shl, shr`

  `not` ➔ zanegowanie wszystkich bitów argumentu, łącznie z bitem znakowym

```
                i ➔ 01101100
j := not i;
                j ➔ 10010011
```

- = równy 3 = 3.14 (false)
- <> różny od... 3 <> 3.14 (true)
- < mniejszy od... 3 < 3.14 (true)
- <= mniejszy lub równy... 3 <= 3.14 (true)
- > większy od... 3 > 3.14 (false)
- >= większy lub równy 3 >= 3 (true)

Typy całkowite – operatory cd.

- Logiczne

`and` ➔ koniunkcja. Iloczyn logiczny odpowiadających sobie par bitów argumentów.

```
i ➔ 00110011
j ➔ 10101010
k:= i and j;
k ➔ 00100010
```

Typy całkowite – operatory cd.

- Logiczne
- **`or`** ➔ alternatywa odpowiadających sobie par bitów argumentów

i ➔ 00110011

j ➔ 10101010

k:= i or j;

k ➔ 10111011

Typy całkowite – operatory cd.

- Logiczne
  - **`xor`** → różnica symetryczna odpowiadających sobie par bitów argumentów

```
          i → 00110011
          j → 10101010
k:= i and j;
          k → 10011001
```

Typy całkowite – operatory cd.

- Logiczne
  - **`shl`** ➔ przesunięcie w lewo argumentu pierwszego o liczbę miejsc określoną przez argument drugi.

```
i ➔ 0000 1101 (13₁₀)
j ➔ 2
j:= i shl j;
j ➔ 0011 0100  (52₁₀)
```

  - Jeżeli pierwszy argument jest liczbą bez znaku ⇔ pomnożenie go przez **`2^drugi_argument`**

Typy całkowite – operatory cd.

- Logiczne
  - **`shr`** ➔ przesunięcie w prawo argumentu pierwszego o liczbę miejsc określoną przez argument drugi.

```
i ➔ 0000 1100 (12_10)
j ➔ 2
j:= i shr j;
j ➔ 0000 0011  (3_10)
```

  - Jeżeli pierwszy argument jest liczbą bez znaku ⇔ podzielenie go przez `2^drugi_argument`

Operator	Znaczenie	Logiczny	Bitowy
Not	Negacja	Not true = false	Not 5 = 250
And	Iloczyn logiczny	True and false = false	7 and 15 = 7
Or	Suma logiczna	True or false = true	7 or 128 = 135
Shl	Przesunięcie bitowe w lewo		7 shl 2 = 28
Shr	Przesunięcie bitowy w prawo		128 shr 4 = 8
Xor	Suma modulo 2	True xor true = false	7 xor 15 = 8

Funkcje standardowe dla typów całkowitych

- Succ(x) - następny argument(jeśli istnieje)
- Pred(x) - poprzedni argument(jeśli istnieje)

- Abs(x) - wartość bezwzględna x
- Sqr(x) - kwadrat x
- Sqrt(x) – pierwiastek kwadratowy dla x>0 (zwraca real)
- Odd(x) – wynik typu boolean, True dla x nieparzystych
- Exp(x) e^x (zwraca real)
- Ln(x) logarytm naturalny dla x >0 (zwraca real)

Wywołania funkcji mogą być elementami wyrażeń

Operatory arytmetyczne

Operator	Opis	Przykład
*	Mnożenie	2*2 = 4
/	Dzielenie	2/2 = 1
div	Dzielenie całkowite	2 div 3 = 0
Mod	Reszta z dzielenia	2 div 3 = 1
+	Dodawanie	2+2 = 4
-	Odejmowanie	2-4 = -2
- (jednoargumentowy)	Wartość ujemna	

Typ logiczny - operatory

- Not
- And
- Or
- Xor

1. Wyrażenia arytmetyczne powiązane operatorami relacji stanowią wyrażenia logiczne.
2. Zmiennej logicznej można nadać wartość wyrażenia

```
var
   b, c, d: boolean;
   i, j, k, l: integer;
b := i <= j;
c := k <= l;
d := b and c;
d := b or c;
```

Typ rzeczywisty - operatory

- Arytmetyczne:
  - +, -, *, / - dają wynik real;
- Relacji:
  - <, <=, =, >, >=, <>

Typ rzeczywisty – funkcje standardowe

- Abs(x)
- Sqr(x)
- Sqrt(x)

- sin(x), cos(x), arctan(x)
- Ln(x)
- Exp(x)

- Round(x) – zaokrąglenie do najbliższej całkowitej
  K := round(5.6) ➔ k = 6
  K := round(-1.5) ➔ k = -2
- Trunc(x) – konwersja liczby rzeczywistej do całkowitej przez obcięcie części ułamkowej
  K := trunc(5.6) ➔ k = 5
- Frac(x) – częśc ułamkowa
- Int(x) - część całkowita

Przykłady wyrażeń

- 2+2,
- 2*(3+4),
- (1+2)/(3+4),
- (a+4)/17.45, (wynik rzeczywisty)
- ln(x), sqr(x), sqrt(x),
- sin(x)/cos(x) (nie ma tan(x)!),
- sqr(sin(x))+sqr(cos(x))

Tablice jednowymiarowe

- Z wykorzystaniem definicji typu

```
type
    typ_tab_nazwa =
    array [typ_porządkowy] of typ_elementów;

type
    tdni= (pn,wt,sr,cz,pt,so,ni);
    typ_tab = array [tdni] of integer;
```

- anonimowy całkowity typ okrojony:

```
    tabl = array [1..10] of real;
```

- Definicja

  var

  nazwa: typ_tab;
- Przykład

  var

  tab1: typ_tab ;

```
tab1[pn]:=22;
tab1[wt]:=tab1[pn]+8;
writeln(tab1[wt]);
```

- Definicje zakresów
  var
  A: array [’A’..’Z’] of integer;
  B: array [-15..80] of integer;
  C: array [boolean] of integer;

- A[’A’]:=22;
- B[A[’A’]-22]:=A[’A’]+8;
- C[false]:=53*A[’A’];
- writeln(C[false]);

- Uwaga na ograniczenia pamięci
- type
  - tint = array [integer] of char;

- Turbo Pascal i DOS – max: 65520 bajty

Sprawdzanie poprawności zakresów

- {$R+}
- {$R-}

- poprawność sprawdzana w trakcie
  - działania programu,
  - kompilacji
- indeksy mogą być wyrażeniami
- Zwalnia wykonywanie programu

```
program tabZakr;
   var
        zm:array [1..4] of integer;
        i:integer;
   begin
        i:=10;
        zm[i+8]:=4;
   end.
```

```
program tabZakr;
    var
        zm:array [1..4] of integer;
        i:integer;
    begin
        i:=10;
{$R+}
        zm[i+8]:=4;
{$R-}
    end.
```

Tablice wielowymiarowe

- Tablica, której elementami są tablice:

- tabWiel: array[tporz1] of array [tporz2] of typ_elementow;

- array [tporz1,tporz2,tporz3] of typ_elementów;

type

TtabWiel= array [0..6, ’a’..’z’, boolean] of char;

- var

  Tab:TtabWiel;

- ...
- zm[3, ’b’, true]:=‘z’;

- Tab:Array [1..4,’a’..’b’] of integer
  - ‘a’ ‘b’
- 1
- 2
- 3
- 4

Tab[2,’a’]:=4;

```
program test;
var
    a: array [0..3, 0..3] of integer;
    i,j : integer;
begin
    for i:=0 to 3 do
        for j:=0 to 3 do
            a[i,j]:= i * j;
    for i:=0 to 3 do
        begin
            for j:=0 to 3 do
                write(a[i,j]:2,' ');
            writeln;
        end;
end.
```

Łańcuchy znakowe

- przechowują ciąg znaków
- Udostępnione są dodatkowe operacje

```
Var
    s1: string;    { max 255 znaków }
    s2: string[20]; { max 20 znaków }
```

Operacje na łańcuchach

- Przypisanie ciągu znaków

  s1 := ’Hello world’;
- Dla ciągu znaków o długości > 255, znaki powyżej 255 pomijane
- Łączenie ciągów znakowych
  - s1 := s1+’!’;
- Wypisanie tekstu
- Write(s1);

Tablice znakowe indeksowane są od 1

S1[1] = ‘H’

Pod komórką zerową znajduje się znak w kodzie ASCII, którego pozycja w tablicy odpowiada długości łańcucha wypełnionego tekstem.

Łatwiej

Length(s1); -> długość łańcucha

Write / Writeln

- Formatowanie
- x:=456;
- writeln(’>’, x,’<’); >456<

- writeln(’>’, x:6,’<’); > 456<

- writeln(’>’, x:2,’<’); >456<

- y:=true;
- writeln(‘>’, y:8, ‘<’); > TRUE<

- writeln(’>’,Pi,’<’); > 3.1415926536E+00<
- writeln(’>’,Pi:0,’<’); > 3.1E+00<
- writeln(’>’,Pi:0:3,’<’); >3.142<
- writeln(’>’,Pi:8:3,’<’); > 3.142<
- writeln(’>’,-Pi,’<’); >-3.1415926536E+00<
- writeln(’>’,-Pi:0:3,’<’); >-3.142<

Read / Readln

- read(nazwa_zmiennej);
- readln(d,z,w);

- Zatrzymuje wykonanie programu
- Oczekuje na wprowadzenie wartości
- Przerywa wykonywanie aplikacji, gdy wprowadzona wartość nieodpowiednia dla oczekiwanego typu
- Buforowane

- read – odczytuje wartości z bufora, pozostawiając resztę do kolejnych odczytów.
- readln - odczytuje wartości z bufora, czyści pozostałe dane w buforze.

- readln(Integer, Integer, Integer);
  - kolejne liczby
  - oddzielone spacją
- readln(string,integer);
  - tekst oddzielony od liczby przez ENTER
- readln(integer,string);
  - spacja(dołączana do tekstu)
- readln(char,integer);
  - bez spacji
- readln(integer,char);
  - spacja=znak