Wykład 1

- Informacje o przedmiocie
- Zakres przedmiotu
- Literatura
- Pojęcia podstawowe

Wstęp do informatyki

Liczba godzin:
Semestr 1, wyk. 30 godz., ćw. 30 godz.
Wykład:
Dr inż. Marek Gajęcki
poniedziałek 9.30 – 11.00
Ćwiczenia:
Dr inż. Marek Gajęcki
Dr inż. Renata Słota
Dr inż. Paweł Pietras
Dr inż. Tomasz Jurczyk
wtorek 15.00 – 18.00
Cel wykładu:
wprowadzenie podstawowych pojęć związanych z informatyką, zapoznanie z programowaniem w języku proceduralnym.
Slajdy z wykładu:
System Moodle - https://upel.agh.edu.pl/weaiie/login/index.php

Wstęp do informatyki

Wyznaczanie oceny końcowej:

1. Aby uzyskać pozytywną ocenę końcową niezbędne jest uzyskanie pozytywnej oceny z ćwiczeń oraz egzaminu.
2. Obliczamy średnią arytmetyczną z ocen zaliczenia ćwiczeń i egzaminów uzyskanych we wszystkich terminach.
3. Wyznaczamy ocenę końcową na podstawie zależności:
   if sr>4.75 then ok:=5.0 else
   if sr>4.25 then ok:=4.5 else
   if sr>3.75 then ok:=4.0 else
   if sr>3.25 then ok:=3.5 else ok:=3.0
4. Jeżeli ocenę z ćwiczeń i ocenę z egzaminu uzyskano w pierwszym terminie oraz ocena końcowa jest mniejsza niż 5.0 to ocena końcowa jest podnoszona o 0.5

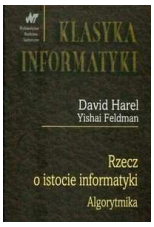
Zakres przedmiotu

- Pojęcia podstawowe
- Mechanizmy języka strukturalnego
- Skalarne typy danych w językach programowania
- Strukturalne typy danych w językach programowania
- Procedury i funkcje - przekazywanie parametrów
- Rekurencja
- Typ wskaźnikowy
- Zastosowanie wskaźników
- Przykłady struktur danych
- Przykłady algorytmów
- Architektura komputera
- Rodzaje języków programowania
- Złożoność obliczeniowa, klasy złożoności

Literatura

- D. Harel „Rzecz o istocie informatyki - algorytmika”
- J.G. Brookshear „Informatyka w ogólnym zarysie”
- N. Wirth „Algorytmy + struktury danych = programy”

- J. Bentley „Perełki oprogramowania”
- J. Bentley „Więcej perełek oprogramowania”
- D.E. Knuth „Sztuka programowania”
- T.H. Cormen „Wprowadzenie do algorytmów”



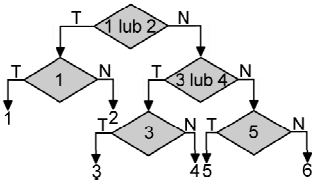
Informatyka (Computer Science)

Informatyka to nauka o przetwarzaniu informacji.

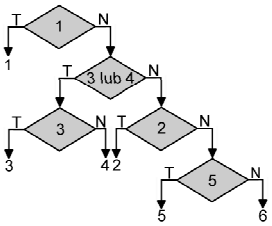
Aktualnie obejmuje wiele zagadnień, między innymi:
- algorytmika, struktury danych, języki programowania
- mikroprocesory, architektury komputerów
- bazy danych, systemy operacyjne, sieci komputerowe
- kompilatory, kryptografia, metody numeryczne
- inżynieria oprogramowania, projektowanie systemów
- grafika, sztuczna inteligencja, aplikacje internetowe
- złożoność obliczeniowa
- …

Identyfikacja elementów zbioru

S1:



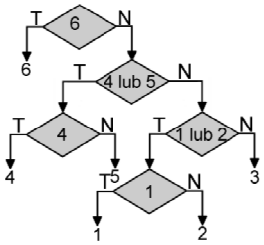
S2:



E(S1)=1/6•2+1/6•2+1/6•3+1/6•3+1/6•3+1/6•3=2.66
E(S2)=1/6•1+1/6•3+1/6•3+1/6•3+1/6•4+1/6•4=3

Identyfikacja elementów zbioru

S3:

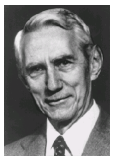


P(6)=0.95
P(1-5)=0.01

E(S1)=0.01•2+0.01•2+0.01•3+0.01•3+0.01•3+0.95•3=2.98
E(S3)=0.01•4+0.01•4+0.01•3+0.01•3+0.01•3+0.95•1=1.12

Teoria informacji

Ilość informacji zawarta w danym zbiorze jest miarą stopnia trudności rozpoznania elementów tego zbioru.



Claude Shannon

Ilością informacji zawartej w zbiorze $X=\{x_1,x_2,\ldots,x_N\}$, nazywamy liczbę:

$$H(X) = -\sum_{i=1}^{N} p_i \bullet \log_2(p_i)$$

gdzie:
p_i ($p_i>0$, $\Sigma p_i=1$) jest prawdopodobieństwem wystąpienia elementu x_i

Przykłady:

{0,1}	1 bit
{0..9}	3.32 bita
zbiór liter języka ang.	4.7 bita

Kompresja danych

Ciąg danych
AABACADABA

1) Kodowanie proste
A - 00
B - 01
C - 10
D - 11

2) Kodowanie Huffmana

A - 0	0.6
B - 10	0.2
C - 110	0.1
D - 111	0.1

Po zakodowaniu

1)	00 00 01 00 10 00 11 00 01 00	20 bitów
2)	0 0 10 0 110 0 111 0 10 0	16 bitów

Ilość informacji
H(X) = 0.6*log(0.6)+0.2*log(0.2)+0.1*log(0.1)+0.1*log(0.1)=15.7

Podstawowe pojęcia

Zadanie algorytmiczne – polega na określeniu:
- wszystkich poprawnych danych wejściowych
- oczekiwanych wyników jako funkcji danych wejściowych

Algorytm - specyfikacja ciągu elementarnych operacji, które przekształcają dane wejściowe na wyniki.

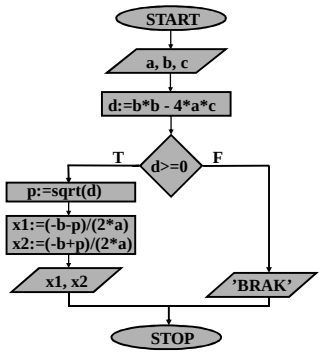
Algorytm może występować w postaci:
- werbalnej *(opis słowny)*
- symbolicznej *(schemat blokowy)*
- programu

Przykład zapisu algorytmu

Problem: równanie kwadratowe
Dane: współczynniki **a**, **b**, **c**
Wyjście: pierwiastki **x1**, **x2** lub informacja o ich braku

Postać werbalna algorytmu**:**
„Mając dane współczynniki a, b, c oblicz ...”

Zapis symboliczny a program




```
var
  a,b,c,d,p,x1,x2:real;

begin
  read(a,b,c);
  d:=b*b-4*a*c;
  if d>=0 then
  begin
    p:=sqrt(d);
    x1:=(-b-p)/(2*a);
    x2:=(-b+p)/(2*a);
    writeln(x1,x2)
  end
  else
    writeln('BRAK')
end.
```

Algorytm Euklidesa

Algorytm Euklidesa (około 300 r. p.n.e.) obliczający największy wspólny dzielnik.

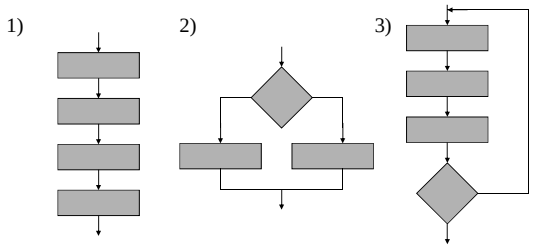
Dane: liczby naturalne a,b
Wynik: NWD(a,b)

```
begin
  read(a,b);
  while a<>b do
    if a>b then
      a:=a-b
    else
      b:=b-a;
  writeln(a)
end.
```

a	b
24	30
24	6
18	6
12	6
6	6

Budowa algorytmów

1) Bezpośrednie następstwo
2) Wybór warunkowy
3) Iteracja warunkowa



Początek nauki algorytmiki

- Programy z pętlami
- Zadania z tablicami jednowymiarowymi
- Zadania z tablicami wielowymiarowymi
- Zastosowania rekordów
- Procedury i funkcje
- Rekurencja
- Wskaźniki

Dlaczego Pascal?

- prosta składnia
- czytelne wyrażenia
- przejrzyste typy danych
- czytelne komunikaty o błędach kompilacji
- kontrola odwołania poza tablicę
- pozwala wyjaśnić mechanizmy: przekazywania parametrów, wskaźników, alokacji pamięci, itp.

Kompilatory języka Pascal

- Turbo Pascal
  - Zalety: mały, prosty, darmowy, zintegrowane środowisko, wbudowany debuger
  - Wady: system DOS, małe tablice, brak 64 bitowych zmiennych typu integer
- FPC
  - Zalety: darmowy, brak ograniczeń na tablice, integer 64 bit, systemy windows i linux
  - Wady:

Przykład zadania

Problem: Rozkład liczby na czynniki pierwsze

Proszę napisać program, który dla wczytanej liczby naturalnej wypisuje jej rozkład na czynniki pierwsze.

Przykład:

120 : 2, 2, 2, 3, 5

Rozkład na czynniki pierwsze

Rozwiązanie proste

```
var
  n,b : int64;

begin
  read(n);
  b:=2;
  while (n>1) do begin
    if n mod b = 0 then begin
      writeln(b);
      n := n div b;
    end else begin
      b := b+1;
    end
  end
end.
```

Rozkład na czynniki pierwsze

Rozwiązanie lepsze

```
begin
  read(n);
  while n mod 2 = 0 do begin
    writeln(2);
    n := n div 2;
  end
  b:=3;
  while (n>1) do begin
    if n mod b = 0 then begin
      writeln(b);
      n := n div b;
    end else begin
      b := b+2;
    end
  end
end.
```

Rozkład na czynniki pierwsze

Rozwiązanie dobre

```
begin
  read(n);
  while n mod 2 = 0 do begin
    writeln(2);
    n := n div 2;
  end
  b:=3;
  while (n>1) and (b<=sqrt(n)) do begin
    if n mod b = 0 then begin
      writeln(b);
      n := n div b;
    end else begin
      b := b+2;
    end
  end
  if n>1 then writen(n)
end.
```

Porównanie rozwiązań

Liczba cyfr	Algorytm prosty	Algorytm lepszy	Algorytm dobry
6	0.07s	0.04s	0.0s
7	0.58s	0.28s	0.0s
8	7.75s	3.64s	0.0s
9	1m7.2s	35s	0.01s
10	9m43s	4m52s	0.01s
11	1h23m	41m31s	0.04s
12	12h27m	6h13m	0.13s
13	4d9h	2d4h30m	0.42s
14	37d12h	18d18h	1.23s

Czy można jeszcze szybciej?

Pytania i zadania

- Ile informacji zawiera 10 znakowe słowo, którego każdy znak z jednakowym prawdopodobieństwem jest jedną z liter a, b, c ?
- Jak stworzyć kody Huffmana dla zbiorów 5,6,7 elementowych?
- Jak przyspieszyć działanie algorytmu Euklidesa?
- Ile czasu potrzebuje w najgorszym przypadku trzeci algorytm aby rozłożyć na czynniki 30 cyfrową liczbę?
- Jak przyspieszyć działania programu rozkładu na czynniki pierwsze?

- Hasłem do Moodla jest jeden z czynników pierwszych liczby N spełniającej równanie:

  sum_13(N)=N

gdzie:

sum_13(N) to suma 13 potęg cyfr liczby N
np. sum_13(2011)=8192+0+1+1=8194

Wykład 2

- Aspekty języka programowania
- Instrukcje w języku Pascal
- Programowanie strukturalne

Przykładowy program

```
procedure main()
  l:=[]
  n:=1
  repeat
    if not((n+:=1)%!l=0) then put(l,write(n))
end
```

- Czy jest to poprawny program ?
- Co robi ten program ?
- Czy można go przyspieszyć ?

Podstawowe pojęcia

Aspekty języka programowania:

- **Syntaktyka** (składnia) - zbiór reguł określający formalnie poprawne konstrukcje językowe
- **Semantyka** - opisuje znaczenie konstrukcji językowych, które są poprawne składniowo
- **Pragmatyka** - opisuje wszystkie inne aspekty języka

Sposoby opisu składni języka

- Notacja **BNF** *(Backus-Naur Form)*
- Notacja **EBNF** *(Extended Backus-Naur Form)*
- Diagramy syntaktyczne *(Syntax Diagram)*

John Warner Backus

Peter Naur

Notacja EBNF

Elementy notacji EBNF:

- Symbole pomocnicze (nieterminalne)
- Symbole końcowe (terminalne)
- Produkcje
- Metasymbole

< >	- symbol pomocniczy
::=	- symbol produkcji
|	- symbol alternatywy
[ ]	- wystąpienie 0 lub 1 raz (EBNF)
{ }	- powtórzenie 0 lub więcej razy (EBNF)

Notacja EBNF

Przykład *(BNF)*

<cyfra> ::= 0|1|2|3|4|5|6|7|8|9

<liczba bez znaku> ::= <cyfra> | <liczba bez znaku> <cyfra>

<liczba> ::= + < liczba bez znaku > | - < liczba bez znaku >

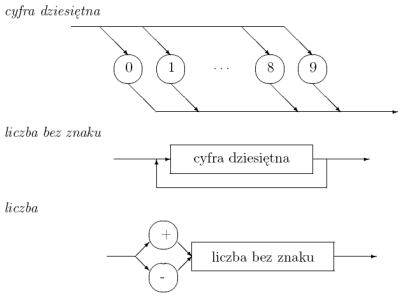
Przykład *(EBNF)*

<cyfra> ::= 0|1|2|3|4|5|6|7|8|9

< liczba bez znaku > ::= <cyfra> {<cyfra>}

<liczba> ::= + < liczba bez znaku > | - < liczba bez znaku >

Diagramy składniowe



•7

Semantyka

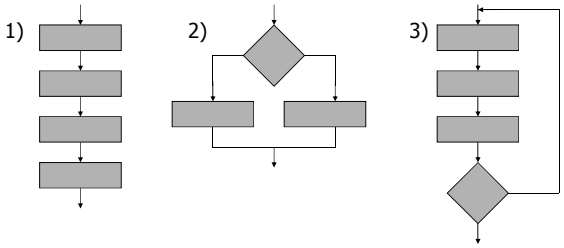
Opis każdej konstrukcji językowej poprawnej składniowo

- **Semantyka denotacyjna** – opis w postaci funkcji przekształcającej dane wejściowe w dane wyjściowe
- **Semantyka operacyjna** – opis stanu komputera przed i po wykonaniu instrukcji

•8

Struktury sterujące przebiegiem programu

1) Bezpośrednie następstwo
2) Wybór warunkowy
3) Iteracja warunkowa



Instrukcje w Pascalu

proste
- przypisania
- procedury
- skoku
- pusta

strukturalne
- grupująca (złożona)
- warunkowa (2 rodzaje)
- wyboru
- iteracyjna (3 rodzaje)

Instrukcja przypisania

Składnia
<zmienna> **:=** <wyrażenie>

Przykłady
a := a+1
y := sin(6*x)

Warianty w innych językach
<zm1> **:=** <zm2> **:=** <wyrażenie>
<zm1> **:=:** <zm2>
<zm1>,<zm2> **:=** <wyr1>,<wyr2>

Problemy:
zgodność typów, konwersja typów
zakres liczb, nieokreśloność zmiennych

Instrukcja wywołania procedury

Składnia
nazwa
nazwa(p1,p2,p3,...)

Przykłady
clrscr
rozwiaz(a,b,c,x1,x2)

Realizacja
niezbędne użycie stosu albo „inline”
przekazywanie parametrów
zwracanie wartości – funkcje
rekurencja

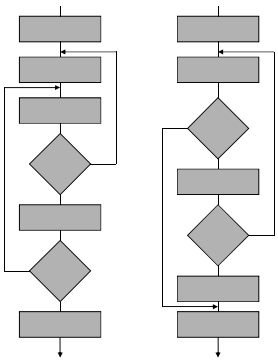
Instrukcja pusta

```
begin
  a:=6;
  { instrukcja pusta }
end;
```

```
begin
    read(a,b);
    while a<>b do ;
       if a>b then a:=a-b else b:=b-a;
    write(a);
end;
```

Instrukcja skoku

```
label
    lab1, lab2;
begin
   ...
   if wyr then goto lab1;
   ...
   lab1 : inst;
   ...
   goto lab1;
end.
```

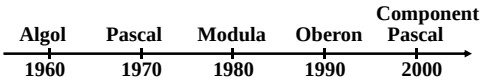


Pytanie: jak zrealizować to bez użycia goto ?

Konstrukcje strukturalne

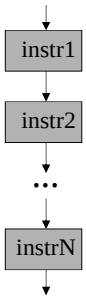
begin ... end
if ... then ...
if ... then ... else ...
case ... of ...
while ... do ...
repeat ... until ...
for ... to ... do ...

Niklaus Wirth



Instrukcja grupująca

begin
<instrukcja1>;
<instrukcja2>;
...
<instrukcjaN>
end

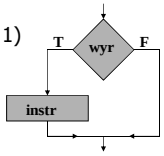


Instrukcja skoku warunkowego

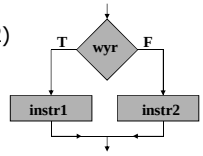
Na przykładzie języka **Pascal**

1) **if** <*wyrażenie*> **then** <*instrukcja*>
2) **if** <*wyrażenie*> **then** <*instrukcja1*> **else** <*instrukcja2*>

1)



2)



1/2

Instrukcja skoku warunkowego

Niejednoznaczność semantyczna (Pascal)

if *w1* **then** *inst1* **else** *inst3*

- **if** *w2* **then** *inst2*
- **if** *w1* **then if** *w2* **then** *inst2* **else** *inst3*
- **if** *w2* **then** *inst2* **else** *inst3*

if *w1* **then** *inst1*

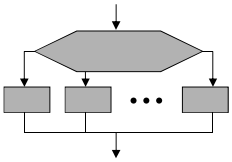
Rozwiązanie problemu (Oberon)

if *wyrażenie* **then** *ciąg_instrukcji*
{ **elsif** *wyrażenie* **then** *ciąg_instrukcji* }
[else *ciąg_instrukcji* **]**
end

2/2

Instrukcja wyboru

case <*wyrażenie*> **of**
<*etykieta1*> : <*instr1*>;
<*etykieta2*> : <*instr2*>;
...
else <*instr*>
end

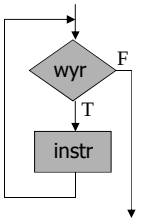


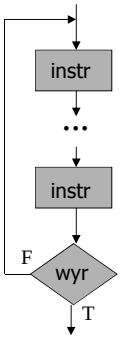
```
case dzien_tygodnia of
   0 : write('niedziela');
   1 : write('poniedzialek');
   ...
   6: write('sobota');
else write('zla wartosc');
end;
```

Instrukcje pętli (Pascal)

repeat <*ciąg_instrukcji*> **until** <*wyrażenie*>

while <*wyrażenie*> **do** <*instrukcja*>





Instrukcje pętli (Oberon)

Realizacja w Oberonie

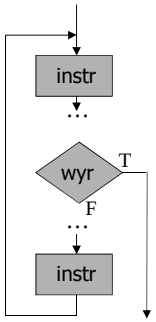
loop <*ciąg_instrukcji*> **end**

if <*wyrażenie*> **then** exit

Realizacja w Pascalu

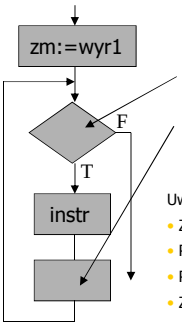
while true **do begin**
<*ciąg_instrukcji*>
end

if <*wyrażenie*> **then** break



Instrukcje pętli (Pascal)

for <zm> **:=** <*wyr1*> **to** <*wyr2*> **do** <*instr*>
downto



zm<=wyr2 albo zm>=wyr2

zm:=succ(zm) albo zm:=pred(zm)

Uwagi:
- Zmienna sterująca typu prostego (integer,char,boolean)
- Problem określoności zmiennej sterującej po zakończeniu pętli
- Problem przerywania pętli
- Zakaz modyfikowania zmiennej sterującej oraz wyr1 i wyr2

Instrukcje pętli (Pascal)

Pętla for przykłady:

```
for i:=1 to 10 do writeln(i)
for i:=10 downto 1 do writeln(i)

for i:=1 to max do tab[i]:=0

for i:=1 to 10 do begin
  for j:=1 to 10 do write(i*j : 6);
  writeln;
end

for zn:='a' to 'z' do write(zn)
```

Pytania i zadania

- Zapisać w notacji EBNF składnie instrukcji warunkowej oraz pętli.
- Które z 6 instrukcji: *if then, if then else, case, while, repeat, for* można usunąć z języka aby nadal można było w nim programować?

- Liczb pierwszych jest nieskończenie wiele. Tylko kilka z nich spełnia warunek:

  sum_p(N)=N

gdzie:

  sum_p(N) to suma p-tych potęg cyfr p-cyfrowej liczby N

  np. sum_p(2012)=16+0+1+16=33

Należy napisać program odnajdujący wszystkie takie liczby.

•24

Wykład 3

- Skalarne typy danych w języku Pascal
- Reprezentacja liczb w maszynie cyfrowej
- Dokładność obliczeń

Typy danych w języku Pascal

- Skalarne
- Strukturalne
- Wskaźnikowe

Typy skalarne - uporządkowane i skończone zbiory wartości.

Typy skalarne w Pascalu:
- **typ logiczny** (*boolean*)
- **typ znakowy** (*char*)
- **typy numeryczne**
  - stałopozycyjne (*integer*, *word*, *longint*, *short*, *byte*)
  - zmiennopozycyjne (*real*, *single*, *double*, *extended*)

Typy proste:
1. Typy elementarne - stałopozycyjny, char, Boolean, wyliczeniowy
2. Typy okrojone - ograniczenie zakresu typu elementarnego

Działania na typach skalarnych

- **typ logiczny** (*boolean*)
  true false
  not or and = <>
- **typ znakowy** (*char*)
  'a' 'b' #32
  = <> < <= > >=
- **typ stałopozycyjny** (*integer*, *word*, *longint*, *short*, *byte*)
  12 -21
  + – * div mod = <> < <= > >=
- **typ zmiennopozycyjny** (*real*, *single*, *double*, *extended*)
  12.3 2e-23
  + – * / = <> < <= > >=

Kod ASCII

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

Rozszerzenie kodu ASCII

128	Ç	144	É	160	á	176	▒	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

Standard ISO-8859

Strona kodowa	Języki
iso-8859-1	afrykanerski, albański, angielski, baskijski, duński, fareski, fiński, francuski, galicyjski, hiszpański, irlandzki, islandzki, kataloński, niderlandzki, niemiecki, norweski, portugalski, szkocki, szwedzki, włoski
iso-8859-2	chorwacki, czeski, polski, rumuński, serbski, słowacki, słoweński, węgierski
iso-8859-3	esperanto, maltański
iso-8859-4	estoński, grenlandzki, lapoński, litewski, łotewski
iso-8859-5	białoruski, bułgarski, macedoński, rosyjski, serbski, ukraiński
iso-8859-6	arabski
iso-8859-7	grecki
iso-8859-8	hebrajski
iso-8859-9	turecki
iso-8859-10	eskimoski, lapoński
iso-8859-11	tajski
iso-8859-13	litewski, łotewski
iso-8859-14	bretoński, gaelicki, szkocki, walijski

Unicode

Jak wyświetlić tekst wielojęzyczny?
Jak wyświetlić różne alfabety?
(cyrylica, alfabety: hebrajski, chiński, japoński, koreański czy tajlandzki)

Unicode - wspólny dla całego świata zestaw znaków.

Unicode

UTF-8

128 znaków (ASCII) kodowanych jest za pomocą 1 bajta.

1920 znaków (alfabety łaciński, grecki, armeński, hebrajski, arabski, koptyjski i cyrylica) kodowanych jest za pomocą 2 bajtów.

63488 znaków (m.in. alfabety chiński i japoński) kodowanych jest za pomocą 3 bajtów.

Pozostałe 2147418112 znaki (jeszcze nie przypisane) można zakodować za pomocą 4, 5 lub 6 bajtów.

UCS-2

Wszystkie znaki zapisywane są za pomocą 2 bajtów. Kodowanie to pozwala na zapisanie tylko 65536 początkowych znaków Unikodu.

UCS-4

Wszystkie znaki zapisywane są za pomocą 4 bajtów.

Unicode (UTF-8)

00000000 – 0000007F: 0*xxxxxxx*
00000080 – 000007FF: 110*xxxxx* 10*xxxxxx*
00000800 – 0000FFFF: 1110*xxxx* 10*xxxxxx* 10*xxxxxx*
00010000 – 001FFFFF: 11110*xxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*
00200000 – 03FFFFFF: 111110*xx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*
04000000 – 7FFFFFFF: 1111110*x* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*

Kodowanie „polskich" znaków

Znak	ISO 8859-2	CP-1250	Unicode	UTF-8
ą	161	165	261	196 133
ć	198	198	263	196 135
ę	202	202	281	196 153
ł	163	163	322	197 130
ń	209	209	324	197 132
ó	211	211	211	195 179
ś	166	140	347	197 155
ź	172	143	378	197 186
ż	175	175	380	197 188
Ą	177	185	260	196 132
Ć	230	230	262	196 134
Ę	234	234	280	196 152
Ł	179	179	321	197 129
Ń	241	241	323	197 131
Ó	243	243	243	195 147
Ś	182	156	346	197 154
Ź	188	159	377	197 185
Ż	191	191	379	197 187

Funkcje standardowe

```
abs(x)          integer → integer
sqr(x)            real  → real

sin(x), cos(x)
arctan(x)         real  → real
exp(x), ln(x)
sqrt(x)

odd(i)          true dla nieparzystych
trunc(x)        trunc(2.7)=2   trunc(-2.7)=-2
round(x)

ord(z)          char    → integer
chr(i)          integer → char

succ(i)
pred(i)
```

Pozycyjny system liczenia

$1999 = 1*10^3 + 9*10^2 + 9*10^1 + 9*10^0$
$1010_{(2)} = 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0$
$127_{(8)} = 1*8^2 + 2*8^1 + 7*8^0$

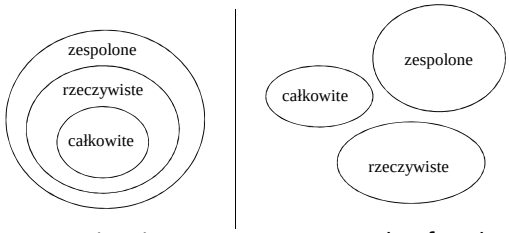
$1.7 = 1*10^0 + 7*10^{-1}$
$0.11_{(2)} = 0*2^0 + 1*2^{-1} + 1*2^{-2} = 0.75$

Arytmetyka liczb w komputerze

- różna od arytmetyki używanej przez ludzi
  - system dwójkowy
  - skończona i ustalona precyzja
- własności liczb o skończonej precyzji (zakresie) są inne
- zbiór liczb o skończonej precyzji (zakresie) nie jest zamknięty na żadne działanie
- nie działa prawo łączności
  a+(b+c) <> (a+b)+c
- nie działa prawo rozdzielności mnożenia względem dodawania
  a*(b+c) <> a*b + a*c

Rodzaje liczb

- liczby całkowite
- liczby rzeczywiste
- liczby zespolone



w matematyce | w maszynie cyfrowej

Reprezentacja liczb stałopozycyjnych

- znak moduł
- kod uzupełnień do jedności **U1**
- kod uzupełnień do dwóch **U2**

przykład: typ 8-bitowy, 1 bit na znak, zakres liczb: -128..127

liczba	znak-moduł	U1	U2
6	0 0000110	0 0000110	0 0000110
-6	1 0000110	1 1111001	1 1111001 +1 = 1 1111010

Działania na liczbach w kodzie U2

Przykład: typ 8-bitowy, 1 bit na znak, zakres liczb: -128..127

6	0 0000110	6	0 0000110	
+7	0 0000111	-7	1 1111001	
13	0 0001101	-1	1 1111111	
64	0 1000000	-65	1 0111111	
-64	1 1000000	-65	1 0111111	
0	0 0000000	-130	0 1111110	NADMIAR!

Liczby całkowite

Turbo Pascal

typ	zakres	rozmiar
shortint	-128..127	1
integer	-32768..32767	2
longint	-2147483648..214748647	4
byte	0..255	1
word	0..65535	2

Java

typ	zakres	rozmiar
byte	-128..127	1
short	-32768..32767	2
int	-2147483648..214748647	4
long	-2^63..2^63-1	8

Liczby zmiennopozycyjne

Cel: Oddzielenie zakresu od dokładności

Sposób zapisu:

- w matematyce
  $l = m * 10^c$
- w komputerze
  $l = m * 2^c$

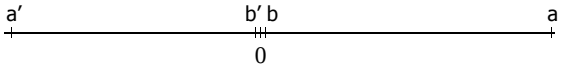
l - ***liczba***
m - ***mantysa***
c - ***cecha***

Liczby zmiennopozycyjne

Przykład 1. System dziesiętny

mantysa - 3 cyfry + znak	(0.001 - 0.999)
cecha - 2 cyfry + znak	(-99 - 99)

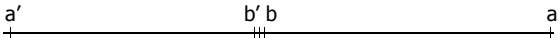
dodatnia wartość maksymalna (a):	$0.999*10^{99}$
ujemna wartość minimalna (a'):	$-0.999*10^{99}$
dodatnia wartość minimalna (b):	$0.001*10^{-99}$
ujemna wartość maksymalna (b'):	$-0.001*10^{-99}$



Liczby zmiennopozycyjne

Przykład 2. System binarny (typ 4 bajtowy Single)
mantysa - 23 bity + 1bit na znak
cecha - 8 bitów

dodatnia wartość maksymalna (a):	$0\ 111..1*2^{01111111}$
ujemna wartość minimalna (a'):	$1\ 111...1*2^{01111111}$
dodatnia wartość minimalna (b):	$0\ 000...1*2^{10000000}$
ujemna wartość maksymalna (b'):	$1\ 000...1*2^{10000000}$



Liczby rzeczywiste

Turbo Pascal

typ	zakres	dokładność	rozmiar
real	2.9E-39 .. 1.7E38	11-12	6
single	1.5E-45 .. 3.4E38	7-8	4
double	5.0E-324 .. 1.7E308	15-16	8
extended	3.4E-4932 .. 1.1E4932	19-20	10

Java

typ	zakres	dokładność	rozmiar
float	1.5E-45 .. 3.4E38	7-8	4
double	5.0E-324 .. 1.7E308	15-16	8

Standard ANSI IEEE 754

Formaty stałopozycyjne:
- 16-bitowy (SHORT INTEGER)
- 32-bitowy (INTEGER)
- 64-bitowy (EXTENDED INTEGER)

Formaty zmiennopozycyjne:
- pojedynczej precyzji (SINGLE)
  m=23+1, c=8
- podwójnej precyzji (DOUBLE)
  m=52+1, c=11

1/2

Dokładność obliczeń

Obliczenia iteracyjne

```
var
   p,q,r : T;
   i:integer;
begin
   r := 3.0;
   p := 0.01;
   for  i:=1 to 50 do
   begin
     q := p+r*p*(1-p);
     writeln(q);
     p := q;
   end;
end.
```

iter	typ **single**	typ **double**	typ **extended**
1.	0.039699997752904	0.039700000000000	0.039700000000000
2.	0.154071718454361	0.154071730000000	0.154071730000000
3.	0.545072615146637	0.545072626044421	0.545072626044421
4.	1.288977980613708	1.288978001188800	1.288978001188800
5.	0.171519219875336	0.171519142109176	0.171519142109176
6.	0.597820341587067	0.597820120107100	0.597820120107100
7.	1.319113850593567	1.319113792413797	1.319113792413797
8.	0.056271348148584	0.056271577646256	0.056271577646256
9.	0.215585991740227	0.215586839232630	0.215586839232630
10.	0.722912013530731	0.722914301179572	0.722914301179571
11.	1.323842763900757	1.323841944168441	1.323841944168441
12.	0.037692066282033	0.037695297254730	0.037695297254729
13.	0.146506190299988	0.146518382713553	0.146518382713550
14.	0.521632552146912	0.521670621435226	0.521670621435216
15.	1.270228624343872	1.270261773935059	1.270261773935051
...		...	...
43.	1.234706044197082	0.616380848687958	0.616385837799877
44.	0.365327119827271	1.325747342863969	1.325748848078739
45.	1.060916781425476	0.030171320123249	0.030165367768645
46.	0.867033898830414	0.117954354819061	0.117931622836727
47.	1.212892293930054	0.430077729813901	0.430002888352197
48.	0.438246011734009	1.165410358209967	1.165304101435091
49.	1.176805377006531	0.587097523770616	0.587415459276028
50.	0.552608847618103	1.314339587829697	1.314491071714711
		^	^

iter	typ **extended**	wartość dokładna
1.	0.039700000000000	0.039700000000000000000000000000000000000000
2.	0.154071730000000	0.154071729999999999999999999999999999999999
3.	0.545072626044421	0.545072626044421299999999999999999999999999
4.	1.288978001188800	1.288978001188800564146077444318929999999999
5.	0.171519142109176	0.171519142109175610913273761193669501531848
...		
50.	1.314491071714711	1.314489760648214812383780183964860775887564
51.	0.074303954005774	0.074309050045856049321771955401870933940599
52.	0.280652583280420	0.280670695427271580509655379893949005186949
53.	0.886312715615762	0.886354663894201577758577930930197101949547
54.	1.188600172876488	1.188544884955797172985886514749765353448385
55.	0.516089578619899	0.516262709159421280638670823473078496363871
56.	1.265312954999400	1.265469282031809506189821162818058704556637
57.	**0.258201197729656**	0.257639616828927735003403861501470932678094
...		
92.	1.137929025629373	0.271467233262087610549464473055410611454272
93.	0.667068700408049	0.864785556843432384355119764618126915551368
94.	1.333332848439945	1.215580049398713266568656814984364161700898
95.	0.000001939572845	0.429415628106318510122842974986646350734709
96.	0.000007758280094	1.164469167439441909702911183666688004618360
97.	0.000031032939806	0.589911344006446644012487937540734261222550
98.	0.000124128870095	1.315659194663309877447649200395210188456166
99.	0.000496469256453	0.069759429146912168060579275901754241874755
100.	0.001985137580646	0.264438582722939495435219262057398282867365

iter	p+r*p*(p-1)	(1+r)*p-r*p*p
1.	0.039700000000000	0.039700000000000
2.	0.154071730000000	0.154071730000000
3.	0.545072626044421	0.545072626044421
4.	1.288978001188800	1.288978001188800
5.	0.171519142109176	0.171519142109176
...		...
50.	1.314491071714711	1.314491283524415
51.	0.074303954005774	0.074303130712665
52.	0.280652583280420	0.280649657149552
53.	0.886312715615762	0.886305938423724
54.	1.188600172876488	1.188609104239421
55.	0.516089578619899	0.516061608915165
56.	1.265312954999400	1.265287683072333
57.	**0.258201197729656**	**0.258291969485672**
...		...
92.	1.137929025629373	1.109698139224346
93.	0.667068700408049	0.744502676303456
94.	1.333332848439945	1.315158000144798
95.	0.000001939572845	0.071710304544597
96.	0.000007758280094	0.271414114844753
97.	0.000031032939806	0.864659594168129
98.	0.000124128870095	1.215729735311535
99.	0.000496469256453	0.428922573284173
100.	0.001985137580646	1.163766571518542

typ **extended**

$$w = \frac{\left(\sqrt{1+u} - \sqrt{1-u}\right)}{u} \qquad w = \frac{2}{\left(\sqrt{1+u} + \sqrt{1-u}\right)}$$

0.100000000	1.00125550119638	1.00125550119638
0.010000000	1.00001250054691	1.00001250054691
0.001000000	1.00000012500005	1.00000012500005
0.000100000	1.00000000125000	1.00000000125000
0.000010000	1.00000000001250	1.00000000001250
0.000001000	1.00000000000011	1.00000000000013
0.000000100	1.00000000000000	1.00000000000000
0.000000010	0.99999999999864	1.00000000000000
0.000000001	0.99999999996103	1.00000000000000

wartości dokładne

0.100000000	1.00125550119637747391785450350122038359405315883
0.010000000	1.00001250054690722874466702747516135163757833258
0.001000000	1.00000012500005468753222658432008437730121685070
0.000100000	1.00000000125000005468750032226562718200685195554
0.000010000	1.00000000001250000000054687500032226562502182000
0.000001000	1.00000000000012500000000005468750000032226562502
0.000000100	1.00000000000000125000000000005468750000000032226
0.000000010	1.00000000000000001250000000000000054687500000000
0.000000001	1.00000000000000000125000000000000000054687500000

Pytania i zadania

- Z którego typu numerycznego, stało- czy zmiennopozycyjnego, można zrezygnować w języku programowania? Zobacz język Lua.
- Dany jest program:

```
begin
  read(n);
  while n<>palindrom(n) do
    n := n+palindrom(n)
end.
```

Czy powyższy program zakończy się dla każdej liczby naturalnej?
Proszę sprawdzić to dla wszystkich liczb n<100.

28

Wykład 4

- Strukturalne typy danych
  - tablice *(array)*
  - rekordy *(record)*
  - napisy *(string)*
  - pliki *(file)*
  - zbiory *(set)*

Typ tablicowy w Pascalu

Definiowanie typów:

```
type
  tab1 = array[min..max] of T;
  tab2 = array[1..max1] of array[1..max2] of T;
  tab3 = array[1..max1] of tab1;
  tab4 = array[1..max1,1..max2] of T;
```

Deklarowanie zmiennych:

```
var
 t1, t2 : tab1;
```

Cechy:

- statyczny rozmiar
- możliwość podstawiania zmiennych tablicowych
- indeksowanie typem prostym (np. char)
- tablice wielowymiarowe
- problem odwołania poza tablicę

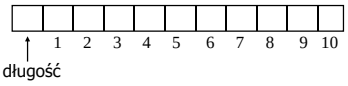
Napisy

```
write('to jest tekst');        Pascal
printf("to jest tekst");       Język C

string;
string[80];
string[255];
```

Przykład:

```
string[10];
```



Operacje na napisach

Deklarowanie:

```
var
  a,b : string[10];
```

Składanie:

```
a:='jeden';
b:='dwa';
a:=b+b;
```

Wczytywanie i wypisywanie:

```
write(a);            dwadwa
write(b[2]);         w
read(a);
```

Porównywanie:

```
if a<>b then write('różne');
```

Operacje na napisach c.d.

Przykładowe funkcje:

```
length(s:string):integer
pos(sub:string; s:string):byte
concat(s1, [s2..sn]:string):string
copy(s:string; ind:integer; count:integer):string
```

Przykładowe procedury:

```
insert(s1:string; var s:string; ind:integer)
delete(var s:string; ind:integer; count:integer)
```

Rekordy

Deklarowanie:

```
type zespolona=record
                 re:real;
                 im:real;
               end;
var z1,z2:zespolona;
```

Nadawanie wartości:

```
z1.re:=5;
z1.im:=6;
```

Podstawianie rekordów:

```
z2:=z1;
```

Rekordy - przykład

Definicje i deklaracje:

```
type data = record
              dzien : 1..31;
              miesiac : 1..12;
              rok : 1..3000;
            end;

type osoba = record
               imie : string[20];
               nazwisko : string[20];
               urodziny : data;
               kobieta : boolean;
             end;

var rob : osoba;
    tab : array[1..100] of osoba;
```

Rekordy - przykład

Przypisania:

```
rob.imie := 'Jola';
rob.urodziny.dzien := 7;
rob.urodziny.miesiac := 6;
rob.kobieta := true;
```

Podstawienie rekordów:

```
tab[1] := rob;
```

Dostęp:

```
if tab[1].kobieta then write(tab[1].imie);
```

Pliki

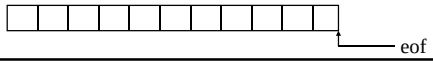
Dostęp do pliku:

- sekwencyjny (taśmy)
- swobodny (dyski)

Deklarowanie:

```
var
  f1 : file of integer;
  f2 : file of osoba;
  f3 : file of char;   (text)
```

Struktura o elementach tego samego typu



Operacje wykonywane na plikach

Procedury i funkcje:

```
assign(f, nazwa)        (open, fopen)
reset(f)
rewrite(f)
append(f)
read(f, zm)
write(f, zm)
eof(f)
seek(f, pos)
close(f)
```

Pliki tekstowe

Deklarownie:

```
var  f : text;     {file of char}
```

Dodatkowe procedury i funkcje:

```
readln( )
writeln( )
eoln( )
```

Znak końca wiersza:

- DOS, Windows — CR LF
- UNIX — LF
- MacOS — CR

Kopiowanie pliku tekstowego

```
assign(f1,'plik1.txt');
assign(f2,'plik2.txt');
reset(f1);
rewrite(f2);
while not eof(f1) do
begin
    while not eoln(f1) do
    begin
        read(f1,ch);
        write(f2,ch)
    end;
    readln(f1);
    writeln(f2)
end;
close(f1);
close(f2);
```

Zbiory mnogościowe

Deklarowanie:
var f : set of T; T - typ prosty

set of char;
set of integer;

Operacje na zbiorach:

+ - *	suma, różnica, iloczyn mnogościowy
in	przynależność elementu do zbioru
= <>	równość, różność
<=	zawieranie się zbiorów
:=	podstawianie

Główna wada:
Brak możliwości „iterowania” po elementach zbioru

Zastosowanie zbiorów

```
if (a<=3)and(a>=0)or(a>=10)and(a<=12) then ...

if a in [0..3, 10..12] then ...
```

```
var
     sam : set of char;

sam := ['a', 'e', 'i', 'o', 'u', 'y'];
if not (zn in sam) then write('spółgłoska');
```

Przykłady operacji na zbiorach

Eratostenes
(ur. 276 p.n.e. w Cyrenie, zm. 194 p.n.e.)

Sito Eratostenesa

2	3	4	5	6	7	8	9	10	11	12	13	14
2	3		5		7		9		11		13	
2	3		5		7				11		13	

Sito Eratostenesa

```
const
  n = 1000;
var
  sito,pierwsze : set of 2..n;
  next,j : integer;
begin
  sito:=[2..n];  pierwsze:=[ ];  next:=2;
  repeat
    while not (next in sito) do next:=succ(next);
    pierwsze:=pierwsze+[next];
    j:=next;
    while j<=n do begin
        sito:=sito-[j];
        j:=j+next;
    end;
  until sito=[ ];
  for j:=2 to n do
    if j in pierwsze then writeln(j)
end.
```

Pytania i zadania

- Co zwraca funkcja abs() dla najbardziej ujemnej liczby?
- Co zwraca funkcja succ() dla największej reprezentowanej w systemie liczby?
- Co zwraca operacja mod gdy jeden/oba argumenty są ujemne?
- Jaka jest kolejność bajtów w liczbie typu integer?
- Jak w języku Pascal umieszczone są w pamięci tablice dwuwymiarowe?
- Ile bajtów zajmuje zmienna typu set of 1..1000?
- Jak mając dostępną funkcję trunc() zrealizować funkcję round()?
- Jak zamienić wartościami dwie zmienne typu integer bez użycia zmiennej pomocniczej?
- Jak wyznaczyć wartość maksymalną 10 elementowej tablicy nie używając operatorów relacyjnych?
- Jak sprawdzić zakres liczb całkowitych nie mając dokumentacji?
- Jak wyznaczyć dokładność liczb zmiennopozycyjnych bez dokumentacji?
- Napisać w Pascalu najkrótszy program wypisujący samego siebie.

17

Wykład 5

- Procedury i funkcje
- Przekazywanie parametrów
- Obszar określoności i czas trwania zmiennej
- Procedury i funkcje rekurencyjne
- Maksymy programistyczne

Struktura programu

Program w języku imperatywnym składa się:
- opisu danych, struktur danych
- opisu czynności do wykonania *(instrukcji)*

Struktura programu w języku Pascal

```
uses                    { import bibliotek }
program                 { nagłówek programu }
label                   { definicje etykiet }
const                   { definicje stałych }
type                    { definicja typów }
var                     { deklaracja zmiennych }
procedure, function     { deklaracje procedur i funkcji }
begin
...                     { część operacyjna programu }
end.
```

Procedury i funkcje

Cel stosowania:
- dekompozycja problemu
- wielokrotne wykonanie
- poziomy abstrakcji
- oddzielna kompilacja - moduły

Projektowanie algorytmu:
- syntetyczne *(bottom-up)*
- analityczne *(top-down)*

Deklarowanie procedur i funkcji

```
procedure dwa(...);

{ definicje stałych }
{ definicje typów }
{ deklaracje zmiennych }
{ deklaracje procedur i funkcji }

begin
  ...
end;
```

1/2

Deklarowanie procedur i funkcji

```
program alfa;
{ definicje stałych i typów, deklaracje zmiennych }

procedure jeden(...);
begin
  ...
end;

procedure dwa(...);
begin
  ...
end;

begin
  ...
end.
```

2/2

Zagłębianie procedur i funkcji

```
procedure procA(...);
begin
  ...
end; { procA }

procedure procB(...);
  procedure procC(...);
  begin
  ...
  end; { procC }
begin
  ...
  procC(...);
end; { procB }
...

...
begin
  procA(...);
  procB(...);
end.
```

Przykładowa procedura

Problem: obliczanie wartości $c=a^b$

```
procedure power(a:real; b:integer; var c:real);
var i : integer;
begin
   i := b; c := 1.0;
   while i>0 do
   begin
      c := c * a;
      i := i-1;
   end;
end;
```

Deklaracja procedury:
- identyfikator procedury
- nagłówek procedury
- parametry formalne
- treść procedury

Wywołanie procedury:
- aktualne parametry

```
power(x,3,z);
```

Przekazywanie parametrów

- przez wartość *(by value)*
- przez zmienną *(by variable)*

Użycie:
by value
dane do procedury
by variable
dane z procedury
dane do i z procedury
duże dane do procedury

1/2

Przekazywanie parametrów

by value

```
procedure p1(a:integer);
begin
   a := a+1;
   writeln(a);
end;

begin
   p1(2);          {3}
   b := 5;
   p1(b);          {6}
   writeln(b);     {5}
end;
```

by variable

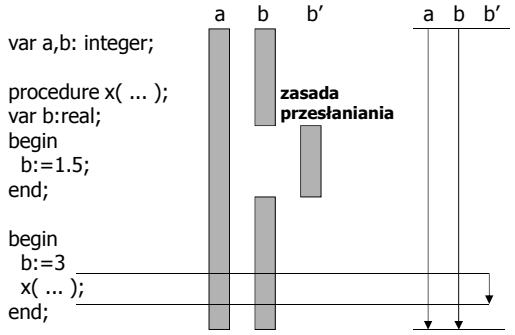
```
procedure p1(var a:integer);
begin
   a := a+1;
   writeln(a);
end;

begin

   b := 5;
   p1(b);          {6}
   writeln(b);     {6}
end;
```

2/2

Obszar określoności i czas trwania



Zapowiedź deklaracji

```
Procedure P( ... ):forward;
Procedure Q( ... );
begin
  ...
  P( ... );
  ...
end;
Procedure P( ... );
begin
  ...
  Q( ... );
  ...
end;
```

Przykłady zastosowania

Funkcja obliczająca silnię *(iloczyn 1*2*3*...*n)*

```
function silnia(n:integer):integer;
var
   i,s : integer; {zmienne lokalne}

begin
   s := 1;
   for i := 1 to n do s := s*i;
   silnia := s;
end;
```

Przykłady zastosowania

Funkcja rekurencyjna:

$$n! = \begin{cases} 1 & \text{dla } n<2 \\ n*(n-1)! & \text{dla } n>=2 \end{cases}$$

```
function silnia(n:integer):integer;
begin
   if n<2 then
      silnia := 1
   else
      silnia := n*silnia(n-1)
end;
```

Przykłady zastosowania

Ciąg Fibonacciego

$$F(n) = \begin{cases} 1 & \text{dla } n < 3 \\ F(n-1)+F(n-2) & \text{dla } n >= 3 \end{cases}$$

1	1	2	3	5	8	13	21				

```
function Fib(n:integer):integer;
begin
   if n<3 then
      Fib := 1
   else
      Fib := Fib(n-1)+Fib(n-2);
end;
```

Problem rekurencji

```
var
   w, poz : integer;  {zmienne globalne}

function fib(n:integer):integer;
begin
   poz := poz+1;
   writeln(' ':poz,'start',n);
   if n<3 then fib := 1
   else fib := fib(n-1)+fib(n-2);
   writeln(' ':poz,'stop',n);
   poz := poz-1;
end;

begin
   poz := 0;
   w := fib(5);
   writeln(w);
end.
```

```
start5
  start4
    start3
      start2
      stop2
      start1
      stop1
    stop3
    start2
    stop2
  stop4
  start3
    start2
    stop2
    start1
    stop1
  stop3
stop5
```

Oddzielna kompilacja-moduły

Deklaracja modułu

```
unit Mat;
interface
function log10(x:real):real;

implementation
function log10(x:real):real;
begin
   log10:=ln(x)/ln(10);
end;

begin
end.
```

Użycie modułu

```
uses
   Mat;

var x,y:real;

begin
   ...
   y:=log10(x);
   ...
end.
```

Maksymy i rady programistyczne

- Programy mają być czytane przez ludzi.
- Dawaj więcej komentarzy niż będzie ci, jak sądzisz potrzeba.
- Stosuj komentarze wstępne.
- Stosuj przewodniki w długich programach.
- Komentarz ma dawać coś więcej, niż tylko parafrazę tekstu programu.
- Błędne komentarze są gorsze niż zupełny brak komentarzy.
- Stosuj odstępy dla poprawienia czytelności.
- Używaj dobrych nazw mnemonicznych.
- Wystarczy jedna instrukcja w wierszu.
- Porządkuj listy według alfabetu.
- Nawiasy kosztują mniej niż błędy.
- Stosuj wcięcia dla uwidocznienia struktury programu i danych.

D. Van Tassel „Praktyka programowania"

1/2

Wykład 7

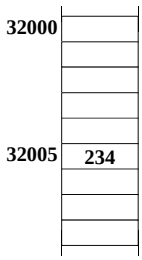
- Zmienna i jej aspekty
- Zmienna wskaźnikowa
- Przydział pamięci dla zmiennych
- Działania na zmiennych wskaźnikowych
- Zastosowanie typu wskaźnikowego

Zmienna

Aspekty zmiennej:
- nazwa
- adres (lokacja)
- wartość
- typ
- rozmiar

```
var
  x:byte;

begin
...
  x:=234;
...
end.
```



Instrukcja podstawienia

zmienna := wyrażenie

z1 := z2 := z3 := wyrażenie (podstawienie wielokrotne)
z1:=: z2 (podstawienie symetryczne)

z ***op*** := wyr ⟺ z := z ***op*** wyr

Czas istnienia nazwy

kompilacja

Program w Pascalu ⇒ Program wykonywalny

nazwa zmiennej ⇒ adres w pamięci

```
procedure main()
  i:=5;  j:=7;  k:=11
  nazwa:=read()          #j
  m:=variable(nazwa)
  write(m)               #7
  variable(nazwa):=3
  write(j)               #3
end
```

Język Icon

Funkcje transformujące

	Pascal	Język C
zmienna ⇒ adres (dostarcza adres zmiennej)	addr(x) @x	&x
adres ⇒ zmienna	a^	*a

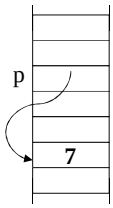
```
var
   x:real; absolute adres;
```

Zmienna wskaźnikowa

Zmienna wskaźnikowa - zmienna, która przechowuje adres innej zmiennej.
Zmienna wskazywana - zmienna, na którą wskazuje zmienna wskaźnikowa.

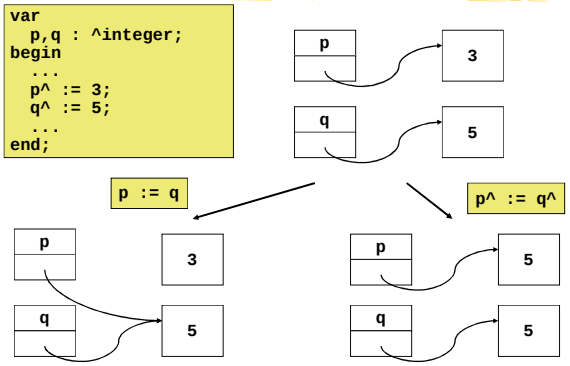
Deklaracja zmiennej wskaźnikowej:

```
var
  p : ^integer;
begin
  p^ := 7;
end;
```



Zmienna wskaźnikowa i wskazywana

```
var
  p,q : ^integer;
begin
  ...
  p^ := 3;
  q^ := 5;
  ...
end;
```

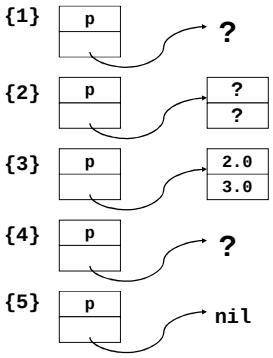


Alokacje i zwalnianie pamięci

```
type
  punkt=record
          x,y:real;
        end;

var
  p : ^punkt;   {1}

begin
  new(p);       {2}
   ...
  P^.x := 2.0;  {3}
  p^.y := 3.0;  {3}
   ...
  dispose(p);   {4}
   ...
  p := nil;     {5}
end;
```



Operacje na zmiennych wskaźnikowych

- *alokacja zmiennej:* **new(p);**
- *zwalnianie pamięci:* **dispose(p);**
- *przypisanie:* **:=**
- *operacje logiczne:* **= <>**
- *wartość „pusta”:* **nil**
- *dostęp:* **a^.x[3]^[7].b**

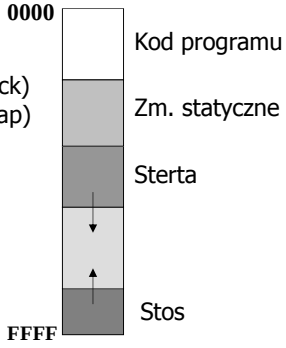
Turbo Pascal

typ pointer

GetMem(var p:pointer; size:word)
freeMem(var p:pointer; size:word)

Przydział pamięci

- statyczny
- dynamiczny ze stosu (stack)
- dynamiczny ze sterty (heap)



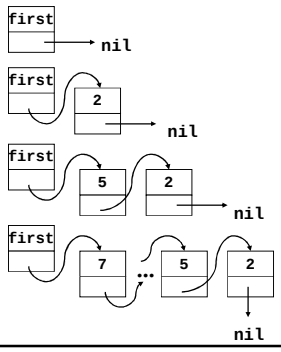
Zastosowania typu wskaźnikowego

- Zmienne dużych rozmiarów
- Dynamiczne struktury danych:
  - stos, kolejka, lista
  - drzewo
  - graf

Tworzenie łańcucha odsyłaczowego (listy)

```
type pnode = ^node;
     node = record;
            w : integer;
            next : pnode;
            end;
var first,p :pnode;
    i,s : integer;
begin
   first:=nil;
   for i:=1 to n do
   begin
      read(s); new(p);
      p^.next:=first;
      p^.w:=s;
      first:=p;
   end;
end.
```

Etapy tworzenia listy:



Zastosowanie łańcucha odsyłaczowego

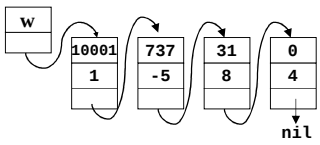
W(x)=x^4+5x^3-7x^2+x+3

```
type
   wielom=array[0..max]of real
```

0	1	2	3	4	5
3	1	-7	5	1	0

W(x)=x^10001-5x^737+8x^31+4

```
type
   link=^node;
   node=record
      wsp:integer;
      wyk:inetger;
      next:link;
end;
```



Wypisanie wartości wielomianu

Iteracyjnie

```
Procedure wypisz1(p:link);
begin
   while p<>nil do
   begin
      if p^.wsp>0 then write('+');
      write(p^.wsp,'x^',p^.wyk);
      p:=p^.next;
   end;
end;
```

Wypisanie wartości wielomianu

Rekurencyjnie

```
Procedure wypisz2(p:link);
begin
   if p<>nil do
   begin
      if p^.wsp>0 then write('+');
      write(p^.wsp,'x^',p^.wyk);
      wypisz2(p^.next);
   end;
end;
```

Wykład 9

- Struktury liniowe o zmiennym podłożu
  - Stos, kolejka
  - Implementacje struktur
- Wybrane algorytmy
  - Wyszukiwanie połówkowe
  - Sortowanie
  - Metody proste
  - Metody szybkie

Struktury liniowe o zmiennym podłożu

- Są to struktury **nie posiadające adresacji** (!).
- Dostęp do poszczególnych elementów struktury jest organizowany poprzez **wyróżnienia**.
- Do tych struktur należą:
  - stos,
  - kolejka,
  - talia niesymetryczna i symetryczna,
  - lista jednokierunkowa i dwukierunkowa.

Stos

- Wyróżnienia: wierzchołek stosu.
- Operacje proste (4 operacje):
  - inicjalizacja stosu – `init(s)`,
  - testowanie czy pusty – `empty(s)`,
  - dołączanie elementu na wierzchołek – `push(s,e)`,
  - pobieranie elementu z wierzchołka – `pop(s)`.
- Uwagi:
  - struktura pracuje jednym końcem,
  - określana jest jako struktura LIFO (Last In First Out).
- Zastosowanie:
  - przeglądanie grafu,
  - obliczania wartości wyrażenia,
  - usuwanie rekurencji z programu.

Kolejka

- Wyróżnienia: początek kolejki, koniec kolejki.
- Operacje proste (4 operacje):
  - inicjalizacja kolejki – `init(k)`,
  - testowanie czy pusty – `empty(k)`,
  - dołączanie elementu na koniec – `put(k,e)`,
  - pobieranie elementu z początku – `get(k)`.
- Uwagi:
  - struktura pracuje oboma końcami,
  - określana jest jako struktura FIFO (First In First Out).
- Zastosowanie:
  - przeglądanie grafu,
  - kolejka zadań o jednakowym priorytecie.

Implementacja struktur

- Tablicowa
  - Zalety: szybkość, prosta implementacja
  - Wady: ograniczenia pamięciowe
- Wskaźnikowa
- Mieszana

Wyszukiwanie połówkowe

```
type tablica = array [1..MAX] of integer;

function szukaj(var t:tablica; sz:integer):integer;
var lewy, prawy, sr : integer;
    found : Boolean;
begin
  lewy:=1;
  prawy:=MAX;
  found:=false;
  repeat
    sr:=(lewy+prawy) div 2;
    if t[sr]=sz then
      found:=true
    else if t[sr]<sz then
      lewy:=sr+1
    else
      prawy:=sr-1
  until found or (lewy>prawy);
  if found then szukaj:=sr else szukaj:=0
end;
```

Wyszukiwanie połówkowe

```
type tablica = array [1..MAX] of integer;

function szukaj(var t:tablica; sz:integer):integer;
var
  lewy, prawy, sr : integer;
begin
  lewy:=1;
  prawy:=MAX;
  repeat
    sr:=(lewy+prawy) div 2;
    if t[sr]<sz then
      lewy:=sr+1
    else
      prawy:=sr-1
  until lewy>prawy;
  if t[lewy]=sz then szukaj:=lewy else szukaj:=0
end;
```


Sortowanie

Sortowaniem nazywamy proces ustawiania zbioru obiektów w określonym porządku.

Szerokie zastosowanie algorytmów sortowania:
- wielość algorytmów rozwiązujących ten sam problem,
- konieczność dokonywania analizy algorytmów,
- pokazują, że warto szukać nowych algorytmów,
- zależność wyboru algorytmów od sortowanej struktury

Metodę sortowania nazywamy **stabilną**, jeżeli podczas sortowania pozostaje **nie zmieniony** względny porządek obiektów o identycznych kluczach.

Metody **in situ** i **ekstensywne** - wymagania dotyczące pamięci.

Sortowania proste I

Przez proste wstawianie:
- dzielimy ciąg na wynikowy i źródłowy; w każdym kroku, począwszy od i=2, przenosimy i-ty element ciągu źródłowego do ciągu wynikowego, wstawiając go w odpowiednim miejscu,
- zachowanie naturalne, algorytm stabilny, działa w miejscu,
- próba poprawy przez wstawianie połówkowe - nie warto!

Sortowania proste I cd.

```
type tablica = array [0..MAX] of integer;

procedure proste_wstawianie(var a:tablica; n:integer);
var i, j : integer;
    x : integer;
begin
  for i:=2 to n do
  begin
    x := a[i];   a[0] := x; {metoda wartownika}
    j := i-1;
    while x < a[j] do
    begin
      a[j+1] := a[j];
      j := j-1;
    end;
    a[j+1] := x;
  end;
end;
```


Sortowania proste II

Przez proste wybieranie:
- podział też na dwa ciągi; wybieramy element najmniejszy z ciągu źródłowego i wymieniamy go z pierwszym elementem tegoż ciągu źródłowego, aż pozostanie w nim jeden, największy element,
- lepszy od prostego wstawiania (mniejsza liczba przestawień), gorzej dla elementów posortowanych, niestabilna, działa w miejscu.

Sortowania proste II

```
type tablica = array [1..MAX] of integer;

procedure proste_wybieranie(var a:tablica; n:integer);
var
  i,j,k : integer;

begin
  for i:=1 to n-1 do
  begin
    k := i; {wybieranie minimum}
    for j:=i+1 to n do
      if a[j]<a[i] then k := j;
    a[k] :=: a[i];
  end;
end;
```

Sortowania proste III

Przez prostą zamianę (bąbelkowe):

- algorytm polega na porównywaniu i ewentualnej zamianie par sąsiadujących ze sobą elementów dopóty, dopóki wszystkie elementy zostaną posortowane.

Ulepszenia tej metody (też nie warto):

- zapamiętanie, czy dokonano zmianę,
- zapamiętanie pozycji ostatniej,
- zamiana kierunku przejść (sortowanie mieszane) - asymetria ciężkiego i lekkiego końca: korzyści tylko w przypadku prawie posortowanych ciągów elementów, czyli rzadko $\Rightarrow$ nie stosuje się.

Sortowania proste III

```
type tablica = array [1..MAX] of integer;

procedure proste_babelkowe(var a:tablica; n:integer);
var
  i,j : integer;
    x : integer;
begin
  for i:=2 to n do
    for j:=n downto i do
      if a[j-1] > a[j] then
        a[j-1] :=: a[j];
end;
```


Sortowania szybkie I

Sortowanie grzebieniowe Combsort:

- pochodzi z roku 1991, oparta na metodzie bąbelkowej,
- włączono tutaj empirię (współczynnik 1.3 wyznaczono doświadczalnie),
- złożoność O(n log n), statystyka gorsza niż Quicksort (1.5 - 2 razy).

Wariant podstawowy:

- za rozpiętość przyjmij długość tablicy, podziel rozpiętość przez 1.3, odrzuć część ułamkową, będzie to pierwsza rozpiętość badanych par obiektów, badaj wszystkie pary o tej rozpiętości, jeżeli naruszają monotoniczność, to przestaw; wykonuj w pętli (rozpiętość podziel znów przez 1.3); kontynuując do rozpiętości 1 przejdziesz na metodę bąbelkową; kontynuuj do uzyskania monotoniczności.

Wariant - Combsort 11

- rozpiętość 9 i 10 zastępujemy 11.

Sortowania szybkie I

```
procedure Combsort(var a:tablica; n:integer);
var top, gap, i, j : integer;
    x : integer;
    swapped : boolean;
begin
  gap := n;
  repeat
    gap := max(trunc(gap/1.3),1);
    top := n - gap;
    swapped := false;
    for i := 1 to top do begin
      j := i + gap;
      if a[i] > a[j] then begin
        a[i] :=: a[j];
        swapped := true;
      end;
    end;
  until (gap = 1) and not swapped;
end;
```


Sortowania szybkie II

Sortowanie przez podział (szybkie) – Quicksort

- wybieramy element dzielący, względem którego dzielimy tablicę na elementy mniejsze i większe, wymieniając elementy położone daleko od siebie, operację powtarzamy dla obu części tablicy, aż do podziału na części o długości 1.
- wersja rekurencyjna i nierekurencyjna,
- warianty: Horowitza, losowy, z medianą z 3,
- kombinacja z metodą prostą,
- wady: może się ukwadratowić - $O(n^2)$ (zły wybór ograniczenia), niestabilna

Sortowania szybkie I cd.

```
procedure sortowanieszybkie(var a:tablica; n:integer);
  procedure sortuj(l, p : integer);
  var i,j : integer;  x : integer;
  begin
    i := l; j := p;
    x := a[(l+p) div 2];
    repeat
      while a[i] < x do i := i+1;
      while a[j] > x do j := j-1;
      if i <= j then begin
        a[j] :=: a[i];
        i := i+1;
        j := j-1;
      end;
    until i > j;
    if l < j then sortuj(l, j);
    if i < p then sortuj(i, p);
  end;
begin
  sortuj(1, n);
end;
```