

## Wykład 1

- Zakres przedmiotu „Wstęp do informatyki”
- Literatura
- Informatyka
- Pojęcia podstawowe

## Zakres przedmiotu

- Pojęcie algorytmu, języka programowania, programu.
- Składnia i semantyka języka programowania, sposoby opisu składni języka. Notacja BNF (EBNF).
- Mechanizmy języka Pascal - konstrukcje strukturalne.
- Procedury i funkcje - przekazywanie parametrów, rekurencja, zagnieżdżanie procedur i funkcji.
- Typy danych w Pascalu.
- Liczby stałe i zmiennoprzecinkowe w maszynie - reprezentacja i działanie.
- Model Von Neumanna. Architektura prostego komputera.

1/2

## Zakres przedmiotu

- System operacyjny.
- Pojęcia kompilatora, translatora, assemblera, linkera. Kompilacja i wykonanie, a interpretacja programu.
- Elementy kompilatora, fazy kompilacji.
- Rodzaje języków programowania.
- Złożoność obliczeniowa czasowa i pamięciowa. Klasy złożoności obliczeniowej. Problem stopu.

2/2

## Literatura

- D. Harel „Rzecz o istocie informatyki - algorytmika”
- N. Wirth „Algorytmy + struktury danych = programy”
- N. Wirth „Wstęp do programowania systematycznego”
- J. Bentley „Perełki oprogramowania”
- A.V. Aho, J.E. Hopcroft, J.D. Ulman „Projektowanie i analiza algorytmów komputerowych”
- W. M. Turski „Metodologia programowania”
- D. Van Tessel „Praktyka programowania”
- E. W. Dijkstra „Umiejętności programowania”

1/2

## Literatura

- A. Marciniak „Turbo Pascal 7.0 z elementami programowania” tom 1
- A. Struzińska - Walczak, K. Walczak „Programowanie w języku Turbo Pascal 7.0”
- L. Buczkowski „Programowanie w Turbo Pascalu 7.0. Podręcznik dla początkujących”

2/2

## Informatyka (Computer Science)

**Informatyka** to nauka o pozyskiwaniu, przetwarzaniu, przechowywaniu, interpretowaniu, zabezpieczaniu informacji.

Podstawową rolę w przetwarzaniu informacji odgrywa człowiek. Urządzenie i oprogramowanie pełni rolę pomocniczą. Człowiek formułuje problem, zadanie, przepis na jego rozwiązanie.

Istotą informatyki jest **informacja oraz realizowane nad nią operacje.**

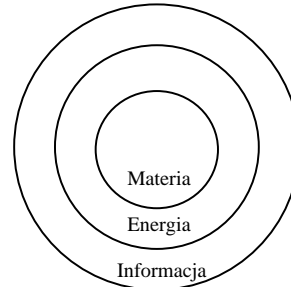
## Podstawowe pojęcia

**Informacja** [łac.] - każdy czynnik zmniejszający stopień niewiedzy o badanym zjawisku, umożliwiający człowiekowi, organizmowi żywemu lub urządzeniu automatycznemu polepszenie znajomości otoczenia i w sprawniejszy sposób przeprowadzenie celowego działania.

**Informacja to „różnica, która robi różnicę”**

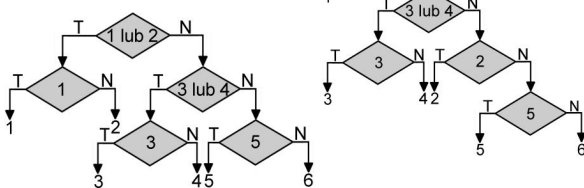
Gregory Bateson

## Sfery rzeczywistości



## Identyfikacja elementów zbioru

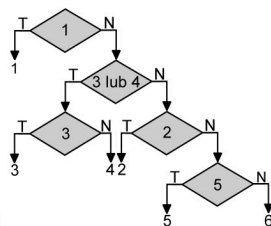
S1:



$$E(S1) = 1/6 \cdot 2 + 1/6 \cdot 2 + 1/6 \cdot 3 + 1/6 \cdot 3 + 1/6 \cdot 3 + 1/6 \cdot 3 = 8/3 = 2.(6)$$

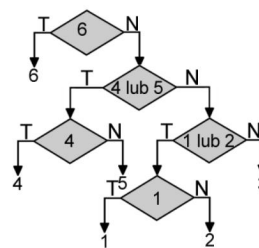
$$E(S2) = 1/6 \cdot 1 + 1/6 \cdot 3 + 1/6 \cdot 3 + 1/6 \cdot 3 + 1/6 \cdot 4 + 1/6 \cdot 4 = 3$$

S2:



## Identyfikacja elementów zbioru c.d.

S3:



$$P(6) = 0.95$$

$$P(1-5) = 0.01$$

$$E(S1) = 0.01 \cdot 2 + 0.01 \cdot 2 + 0.01 \cdot 3 + 0.01 \cdot 3 + 0.01 \cdot 3 + 0.95 \cdot 1 = 2.98$$

$$E(S3) = 0.01 \cdot 4 + 0.01 \cdot 4 + 0.01 \cdot 3 + 0.01 \cdot 3 + 0.01 \cdot 3 + 0.95 \cdot 1 = 1.12$$

## Teoria informacji

rok 1948 ↔ Claude Shannon

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

Ilością informacji zawartej w zbiorze  $X = \{x_1, x_2, \dots, x_N\}$ , gdzie prawdopodobieństwem elementu  $x_i$  jest liczba  $p_i$  ( $p_i > 0$ ,  $\sum p_i = 1$ ), nazywamy liczbę:

$$H(X) = - \sum p_i \cdot \log_2(p_i)$$

**Przykłady:**

{0,1}	1 bit	{0..F}	4 bity
{0..9}	3.32 bity	zbiór liter języka ang.	4.70 bity
		zbiór liter języka pol.	5.0 bitów

## Podstawowe pojęcia

**Zadanie algorytmiczne:**

- dopuszczalny zestaw danych wejściowych
- specyfikacja wyników jako funkcji danych wejściowych

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



**Komunikatywność**

## Podstawowe pojęcia

**Język programowania** - zbiór konwencji umożliwiających komunikatywność

**Program** - algorytm wyrażony w języku programowania

## Sposoby zapisu algorytmu

- werbalnej (*opis słowny*)
- symbolicznej (*schemat blokowy*)
- pseudokodu
- języka programowania

## Przykład zapisu algorytmu

**Problem:** równanie kwadratowe

**Dane:** współczynniki A, B, C

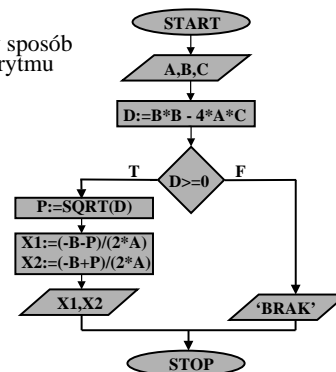
**Wyjście:** pierwiastki X1, X2 lub informacja o ich braku

**Postać werbalna** algorytmu:

„Mając dane współczynniki A,B,C oblicz...”

## Przykład zapisu algorytmu

Symboliczny sposób zapisu algorytmu



## Przykład zapisu algorytmu

Zapis algorytmu w języku programowania:

```
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);
  write(x1,x2);
end
else
  writeln('BRAK');
```

## Algorytm Euklidesa

Algorytm Euklidesa - algorytm obliczający największy wspólny dzielnik.

**Dane:** liczby naturalne a,b

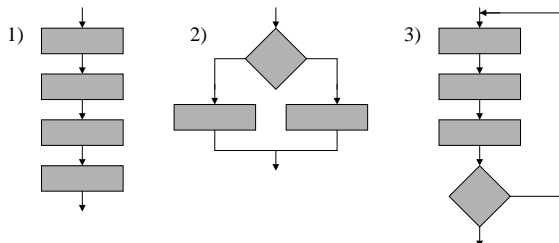
**Wynik:** NWD(a,b)

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

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

## Struktury sterujące przebiegiem programu

- 1) Bezpośrednie następstwo
- 2) Wybór warunkowy
- 3) Iteracja, iteracja warunkowa



## Wykład 2

- Składnia i semantyka
- Instrukcje w Pascalu
- Konstrukcje strukturalne

## Aspekty języka programowania

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

1/2

## Aspekty języka programowania

### Pytania o składnię:

- Jaki jest typ zmiennej  $X$  w programie  $P$
- Czy poprawną konstrukcją jest `if  $a > 0$  then else  $a := 0$  ?`

### Pytania o semantykę:

- Jaka jest wartość zmiennej  $X$  po wykonaniu procedury  $P$  ?
- Ile razy będzie wywołana procedura  $P$  ?

### Pytania o pragmatykę:

- Ile pamięci potrzeba do wykonania programu?
- Który z programów  $P$  i  $Q$  jest bardziej efektywny?

2/2

## Sposoby opisu składni języka programowania

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

## Notacja EBNF

### Elementy notacji EBNF:

- Symbole pomocnicze (nieterminalne)
- Symbole końcowe (terminalne)
- Produkcje
- Metasybole
  - $\langle \rangle$  - symbol pomocniczy
  - $::=$  - symbol produkcji
  - $|$  - symbol alternatywy
  - $\{ \}$  - powtórzenie 0 lub więcej razy

## Notacja EBNF

### Przykład (EBNF)

```
<cyfra> ::= 0|1|2|3|4|5|6|7|8|9  
<ciąg cyfr> ::= <cyfra> {<cyfra>}  
<liczba całkowita> ::= <ciąg cyfr> | + <ciąg cyfr> | - <ciąg cyfr>
```

### Przykład (BNF)

```
<cyfra> ::= 0|1|2|3|4|5|6|7|8|9  
<ciąg cyfr> ::= <cyfra> | <ciąg cyfr> <cyfra>  
<liczba całkowita> ::= <ciąg cyfr> | + <ciąg cyfr> | - <ciąg cyfr>
```

## Składnia Pascala

## Diagramy składni dla Pascala

## Instrukcje w Pascalu

### proste

- przypisania
- procedury
- skoku
- pusta

### strukturalne

- grupująca (złożona)
- warunkowa
- wyboru
- iteracyjna (3 rodzaje)
- wiążąca

## Instrukcja podstawienia

`<zmienna> ::= <wyrażenie>`

Przykład:

```
a := 0  
a := a+1  
y := sin(6*x)
```

`<zml> ::= <zml2> ::= <wyrażenie>`  
`<zml> ::= <zml2>`

## Instrukcja pusta

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

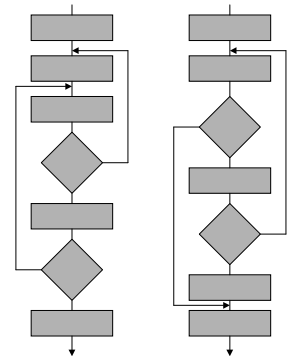
```
procedure ppp;  
begin  
  { instrukcja pusta }  
end;
```

## Instrukcja pusta

```
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.
```



## Konstrukcje strukturalne

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

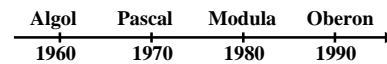
case <wyrażenie> of  
 <wartość1>: <instrukcja1>;  
 <wartość2>: <instrukcja2>;  
 ...  
 else <instrukcja>  
end

1/2

## Konstrukcje strukturalne

while <wyrażenie> do <instrukcja>

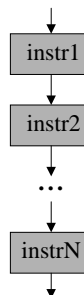
repeat  
 <instrukcja1>;  
 <instrukcja2>;  
 ...  
until <wyrażenie>



2/2

## Instrukcja grupująca

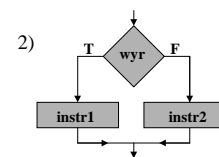
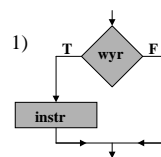
```
begin
  <instrukcja1>;
  <instrukcja2>;
  ...
  <instrukcjaN>
end
```



## Instrukcja skoku warunkowego

Na przykładzie Pascala

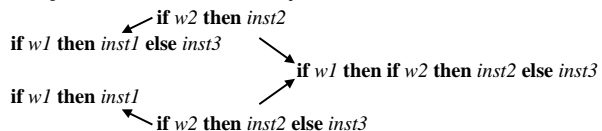
- 1) if <wyrażenie> then <instrukcja>
- 2) if <wyrażenie> then <instrukcja1> else <instrukcja2>



1/2

## Instrukcja skoku warunkowego

### Niejednoznaczność semantyczna (Pascal)



### Rozwiązanie problemu (Oberon)

```

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

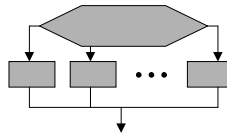
2/2

## Instrukcja wyboru

(Pascal)

```

case wyrażenie of
etykieta1 : instr1;
etykieta2 : instr2;
...
else instr
end
  
```



(Icon)

```

case wyrażenie of
wyrażenie1 : instr1
wyrażenie2 : instr2
...
default : instr
end
  
```

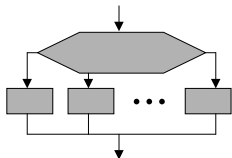
```

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

## 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;
  
```

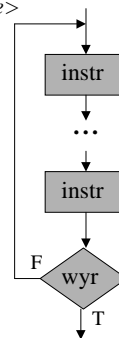
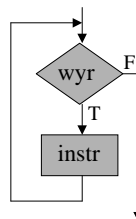
## Instrukcje pętli (Pascal)

```

repeat <ciąg_instrukcji> until <wyrażenie>
while <wyrażenie> do <instrukcja>
  
```

```

if <wyrażenie> then break
  
```



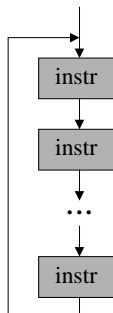
## Instrukcje pętli (Oberon)

```

while <wyrażenie> do <ciąg_instrukcji> end
repeat <ciąg_instrukcji> until <wyrażenie>
loop <ciąg_instrukcji> end
  
```

```

if <wyrażenie> then exit
  
```



## Wykład 3

- Typy danych w Pascalu
- Typy skalarne
- Typy strukturalne

## Typy danych w Pascalu

- Skalarne
- Strukturalne
- Wskaźnikowe

**Typy skalarne** - uporządkowane i skończone zbiory wartości.

Typ skalarny w Pascalu to **typ prosty** albo **typ real**

**Typy proste:**

1. Typy elementarne - integer, char, Boolean lub typ wyliczeniowy
2. Typy okrojone - ograniczenie zakresu typu elementarnego



## Skalarne typy danych

- **typy numeryczne**
  - całkowite (*integer, word, longint, short, byte*)
  - rzeczywiste (*real, float, double, extended*)
  - zespolone (*complex*)
- **typ znakowy** (*char*)
- **typ logiczny** (*boolean*)


## Działania na typach skalnychych

- **typ całkowity** (*integer, word, longint, short, byte*)  
12 -21  
+ - \* div mod = <> < <= > >=
- **typ rzeczywisty** (*real, float, double, extended*)  
12.3 2e-23  
+ - \* / = <> < <= > >=
- **typ znakowy** (*char*)  
'a' 'b' #123  
= <> < <= > >=
- **typ logiczny** (*boolean*)  
true false  
not or and = <>

## Funkcje standardowe

abs(x)		integer → integer
sqr(x)		real → real
sin(x)		real → real
cos(x)		
arctan(x)		
exp(x)		
ln(x)		
sqrt(x)		

## Funkcje standardowe

odd(i)	true dla nieparzystych
trunc(x)	trunc(2.7)=2 trunc(-2,7)=-2
round(x)	 trunc(x+0,5) dla x>=0 trunc(x-0,5) dla x<0
ord(z)	char → integer
chr(i)	integer → char
succ(i)	
pred(i)	

## Typy strukturalne w Pascalu

- tablice (*array*)
- łańcuchy znaków (*string*) - Turbo Pascal
- rekordy (*record*)
- pliki (*file*)
- zbiory (*set*)
- stos, kolejka, lista, talia
- drzewo, graf



## Typ tablicowy w Pascalu

### Deklarowanie:

```
type
  tab1=array[1..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;
```

### Odwołanie do elementów:

```
t1[a+3]:=t1[a+4];
```

### Cechy:

- statyczny rozmiar
- możliwość podstawiania zmiennych tablicowych

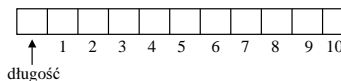
## Łańcuch znaków

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

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

### Przykład:

```
string[10];
```



## Operacje na łańcuchach

### 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 łańcuchach 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

### Deklaracje:

```
type data = record
  dzien : 1..31;
  miesiac : 1..12;
  rok : 0..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;
```

1/2

## Rekordy - przykład

### Przypisanie:

```
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);
```

2/2

## Pliki

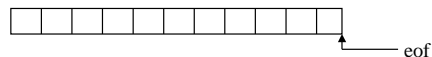
### Dostęp do pliku:

- sekwencyjny (taśmy)
- swobodny (dyski)

### Deklarowanie:

```
var file of T;  
    file of integer;  
    file of osoba;  
    file of char; (text)
```

### Struktura o elementach tego samego typu



## Operacje wykonywane na plikach

### Deklarowanie

```
var f : file of T;
```

### Procedury i funkcje:

- assign(f, nazwa); (open, fopen)
- reset(f);
- rewrite(f);
- read(f, zm);
- write(f, zm);
- close(f);
- eof(f);
- seek(f, pos); Turbo Pascal

## Pliki tekstowe

### Deklarowanie:

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

### Procedury i funkcje:

- readln()
- writeln()
- eoln()

### Znak końca linii:

- DOS CR LF
- UNIX LF

## Kopiowanie pliku tekstowego

```
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;
```

## Zbiory

### Deklarowanie:

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

```
set of char;  
set of integer;
```

### Operacje na zbiorach:

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

## 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');
```

## Sito Eratostenesa

Przykład wykorzystanie zbiorów:

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	
---	---	--	---	--	---	--	--	--	----	--	----	--

1/2

## 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=[ ];
end.
```

2/2

## Procedury i funkcje

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

## Struktura programu w Pascalu

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
end.	

## Procedury i funkcje

**Cel stosowania:**

- dekompozycja problemu
- wielokrotne wykonanie
- poziomy abstrakcji
- oddzielna kompilacja - moduły (Turbo Pascal)

**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.
```

## Procedury standardowe

```
new(p);
dispose(p);
halt;

???
```

## 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*)
- przez procedurę lub funkcję

**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

## Efekty uboczne

```
var
  b,c:integer;
function f(var a:integer):integer;
begin
  a := a*2;
  f := a+1;
end;

begin
  b := 3;
  c := f(b)+f(b);
  writeln(b,c);
  b := 3;
  c := 2*f(b);
  writeln(b,c);
end.
```

## Obszar określoności i czas trwania

```
var a,b: integer;
procedure x( ... );
var b:real;
begin
  b:=1.5;
end;
begin
  b:=3;
  x( ... );
end;
```

zasada przesłaniania

## 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

Procedura (funkcja) rekurencyjna:

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

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

## Przykłady zastosowania

Ciąg Fibonacciego

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

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;
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
stop1
stop3
stop5
```

## Oddzielna kompilacja-moduły(Turbo Pascal)

### 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.

1/2

## Maksymy i rady programistyczne

- 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”

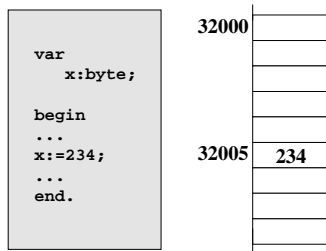
## Wykład 6

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

## Zmienna

### Aspekty zmiennej:

- nazwa
- adres (lokacja)
- wartość
- typ
- rozmiar



## Instrukcja podstawienia

zmienna := wyrażenie

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

z op := wyr  $\longleftrightarrow$  z := z op wyr

## Czas istnienia nazwy

Program w Pascalu  $\xrightarrow{\text{kompilacja}}$  Program wykonywalny  
nazwa zmiennej  $\xrightarrow{\quad}$  adres w pamięci

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

Język Icon

## Funkcje transformujące

	Pascal	Język C
zmienna $\rightarrow$ adres (dostarcza adres zmiennej)	<b>addr(x)</b> <b>@x</b>	<b>&amp;x</b>
adres $\rightarrow$ zmienna	<b>a^</b>	<b>*a</b>

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

## Zmienna wskaźnikowa

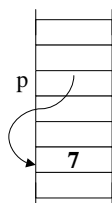
**Zmienna wskaźnikowa** - zmienna, która przechowuje adres innej zmiennej.

**Zmienna wskazywana** - zmienna, na którą wskazuje zmienna wskaźnikowa.



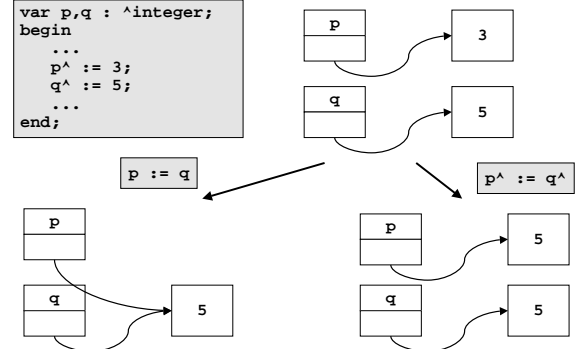
Deklaracja zmiennej wskaźnikowej:

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



## Zmienna wskaźnikowa i wskazywana

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



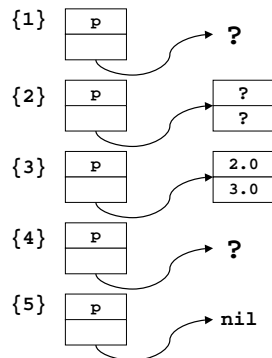
## Alokacje i zwalnianie pamięci

```

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: **:=**  
 operatory logiczne: **= <>**  
 wartość „pusta”: **nil**  
 dostęp: **a^.x[3]^7].b**

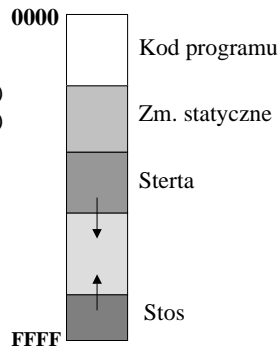
Turbo Pascal

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
- Nieregularne struktury danych:
  - lista, stos, talia
  - drzewo
  - graf

## Tworzenie łańcucha odsyłaczowego (listy)

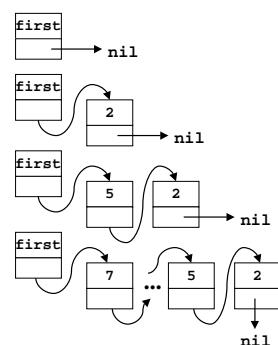
```

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 nieregularnych struktur danych

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

```

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

```

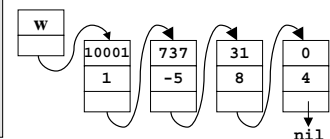
	0	1	2	3	4	5
	3	1	-7	5	1	0

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

```

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;
```

## Wstawianie elementu do listy

```
procedure wstaw(var first:pnode; m:pnode; wart:integer);
var r : pnode;
begin
  new(r); r^.w:=wart;
  if m=first then
  begin
    r^.next:=first; first:=r;
  end
  else
  begin
    p:=first;
    while first^.next<>m do
      first:=first^.next;
    r^.next:=first^.next;
    first^.next:=r;
    first:=p;
  end;
end;
```

## Usuwanie elementu z listy

```
procedure usun(var first : pnode; m : pnode);
var r : pnode;
begin
  if m=first then
  begin
    first:=first^.next;
    dispose(m);
  end
  else
  begin
    r:=first;
    while r^.next<>m do
      r:=r^.next;
    r^.next:=m^.next;
    dispose(m);
  end;
end;
```

## Pozycyjny system liczenia

$$1999 = 1 \cdot 10^3 + 9 \cdot 10^2 + 9 \cdot 10^1 + 9 \cdot 10^0$$

$$1010_{(2)} = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$$

$$127_{(8)} = 1 \cdot 8^2 + 2 \cdot 8^1 + 7 \cdot 8^0$$

$$1.7 = 1 \cdot 10^0 + 7 \cdot 10^{-1}$$

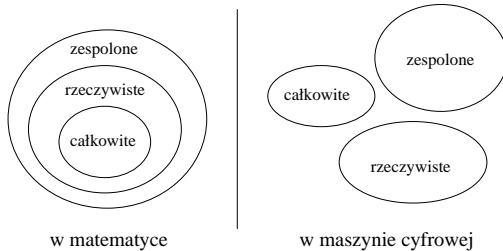
$$0.11_{(2)} = 0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} = 0.75$$

## Arytmetyka liczb w maszynie

- 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
$$a+(b+c) \neq (a+b)+c$$
- nie działa prawo rozdzielności mnożenia względem dodawania
$$a \cdot (b+c) \neq a \cdot b + a \cdot c$$

## Rodzaje liczb

- liczby całkowite
- liczby rzeczywiste
- liczby zespolone



## Reprezentacja liczb stałoprzecinkowych

- znak modułu
- 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 111010

## Działania na liczbach kodu 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 <sup>63</sup> ..2 <sup>63</sup> -1	8

## Liczby zmiennoprzecinkowe

**Cel:** Oddzielenie zakresu od dokładności

**Sposób zapisu:**

- w matematyce

$$l = m \cdot 10^c$$

- w maszynie cyfrowej

$$l = m \cdot 2^c$$

l - *liczba*

m - *mantysa*

c - *cecha*

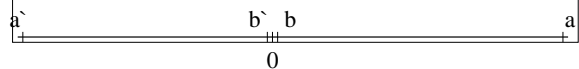
## Liczby zmiennoprzecinkowe

**Przykład 1.** System dziesiętny

mantysa - 3 cyfry + znak (0.001 - 0.999)

cecha - 2 cyfry + znak (-99 - 99)

dodatnia wartość maksymalna (a):  $0.999 \cdot 10^{99}$   
 ujemna wartość minimalna (a`):  $-0.999 \cdot 10^{99}$   
 dodatnia wartość minimalna (b):  $0.001 \cdot 10^{-99}$   
 ujemna wartość maksymalna (b`):  $-0.001 \cdot 10^{-99}$

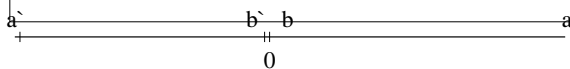


## Liczby zmiennoprzecinkowe

### Przykład 1. 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<sup>-</sup>):  $1\ 111...1 * 2^{01111111}$   
dodatnia wartość minimalna (b):  $0\ 000...1 * 2^{10000000}$   
ujemna wartość maksymalna (b<sup>-</sup>):  $1\ 000...1 * 2^{10000000}$



3/3

## 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
comp	-9.2E18 .. 9.2E18	19-20	8

### 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 stałoprzecinkowe dwójkowe:

- 16-bitowy (SHORT INTEGER)
- 32-bitowy (INTEGER)
- 64-bitowy (EXTENDED INTEGER)

### Format stałoprzecinkowy dziesiętny BCD:

- 80-bitowy (liczba 18 cyfrowa ze znakiem)

1/2

## Standard ANSI IEEE 754

### Formaty zmiennoprzecinkowe:

- zwykły pojedynczej precyzji (SINGLE)  
 $m=23+1, c=8$
- rozszerzony o pojedynczej precyzji (SINGLE, EXTENDED)  
 $m \geq 32, c \geq 11$
- zwykły o podwójnej precyzji (DOUBLE)  
 $m=52+1, c=11$
- rozszerzony o podwójnej precyzji (DOUBLE, EXTENDED)  
 $m \geq 64, c \geq 15$

2/2

## Dokładność obliczeń zmiennoprzecinkowych

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	0.0301711320123249	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.	0.03970000000000	
2.	0.15407173000000	
3.	0.545072626044421	
4.	1.288978001188800	
5.	0.171519142109176	
...		
50.	1.314491071714711	
51.	0.074303954005774	
52.	0.280652583280420	
53.	0.886312715615762	
54.	1.188600172876488	
55.	0.516089578619899	
56.	1.265312954999400	
57.	<b>0.258201197729656</b>	
...		
92.	1.137929025629373	
93.	0.667068700408049	
94.	1.333332848439945	
95.	0.000001939572845	
96.	0.000007758280094	
97.	0.000031032939806	
98.	0.000124128870095	
99.	0.000496469256453	
100.	0.001985137580646	

iter	p+r*p*(p-1)	(1+r)*p-r*p*p
1.	0.03970000000000	0.03970000000000
2.	0.15407173000000	0.15407173000000
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.	<b>0.258201197729656</b>	<b>0.258291969485672</b>
...		
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 real	$w = \frac{(\sqrt{1+u} - \sqrt{1-u})}{u}$	$w = \frac{2}{(\sqrt{1+u} + \sqrt{1-u})}$
0.10000000	1.00125550119628	1.00125550119628
0.01000000	1.00001250054629	1.00001250054629
0.00100000	1.00000012500095	1.00000012500095
0.00010000	1.00000000124965	1.00000000124965
0.00001000	1.00000000001273	1.00000000001273
0.00000100	1.00000000000000	1.00000000000000
0.00000010	1.00000000000000	1.00000000000000
0.00000001	0.99999999999909	1.00000000000000
0.00000000	0.99999999999609	1.00000000000000

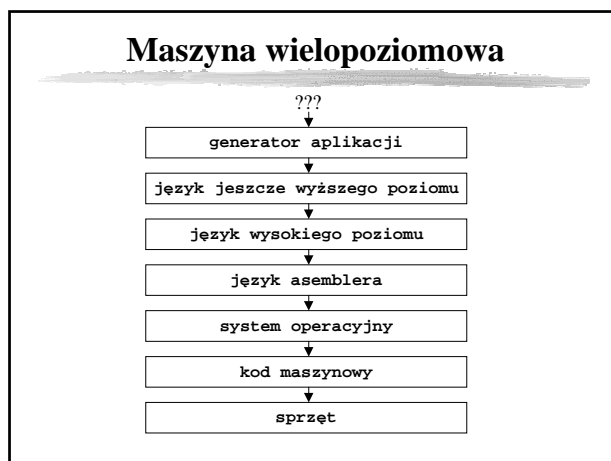
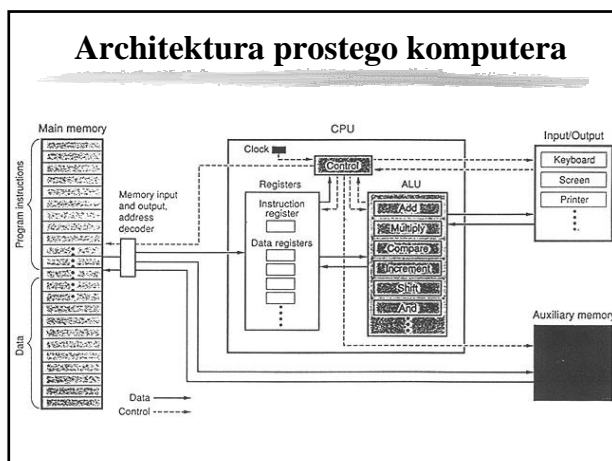
  

typ double	$w = \frac{(\sqrt{1+u} - \sqrt{1-u})}{u}$	$w = \frac{2}{(\sqrt{1+u} + \sqrt{1-u})}$
0.10000000	1.00125550119637	1.00125550119638
0.01000000	1.00001250054690	1.00001250054691
0.00100000	1.00000012500005	1.00000012500005
0.00010000	1.00000000125000	1.00000000125000
0.00001000	1.00000000001250	1.00000000001250
0.00000100	1.00000000000018	1.00000000000013
0.00000010	1.00000000000000	1.00000000000000
0.00000001	0.99999999999864	1.00000000000000
0.00000000	0.999999999996103	1.00000000000000

typ extended	$w = \frac{(\sqrt{1+u} - \sqrt{1-u})}{u}$	$w = \frac{2}{(\sqrt{1+u} + \sqrt{1-u})}$
0.10000000	1.00125550119638	1.00125550119638
0.01000000	1.00001250054691	1.00001250054691
0.00100000	1.00000012500005	1.00000012500005
0.00010000	1.00000000125000	1.00000000125000
0.00001000	1.00000000001250	1.00000000001250
0.00000100	1.00000000000011	1.00000000000013
0.00000010	1.00000000000000	1.00000000000000
0.00000001	0.99999999999864	1.00000000000000
0.00000000	0.999999999996103	1.00000000000000

wartości dokładne	$w = \frac{(\sqrt{1+u} - \sqrt{1-u})}{u}$	$w = \frac{2}{(\sqrt{1+u} + \sqrt{1-u})}$
0.10000000		
0.01000000		
0.00100000		
0.00010000		
0.00001000		
0.00000100		
0.00000010		
0.00000001		
0.00000000		



## Poziom języka programowania

Język  
Pascal

```
while i<>j do
  if i>j then i:=i-j
  else j:=j-i;
```

Język  
asemblera

```
LOOP LOAD I
SUB J
JZERO EXIT
JNEG SUBI
STORE I
JUMP LOOP
SUBI LOAD J
SUB I
STORE J
JUMP LOOP
EXIT
```

Kod  
maszynowy

```
A000 10 B0 00
A003 12 B0 02
A006 24 10 1E
A009 25 A0 12
A00C 11 B0 00
A00F 23 A0 00
A012 10 B0 02
A015 12 B0 00
1018 11 B0 02
101B 23 A0 00
101E
:
:
:
B000
B003
B006 } zmienne
:
:
:
JUMP 23
JZERO 24
JNEG 25
```

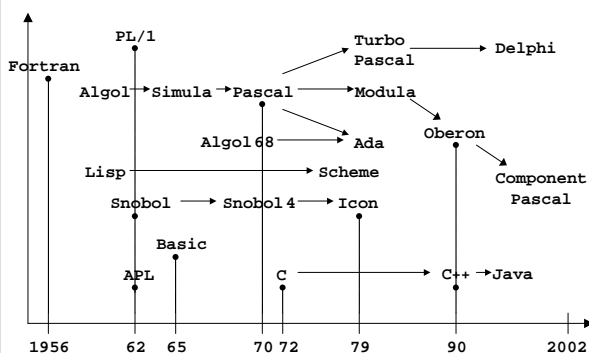
```
LOAD 10
STORE 11
SUB 12
:
JUMP 23
JZERO 24
JNEG 25
```

## Poziom języka programowania

Table 1. Ratios of logical source-code statements to function points for selected programming languages.

Language	Nominal level	Source statements per function point		
		Low	Mean	High
First generation	1.00	220	320	500
Basic assembly	1.00	200	320	450
Macro assembly	1.50	130	213	300
C	2.50	60	128	170
Basic (interpreted)	2.50	70	128	165
Second generation	3.00	55	107	165
Fortran	3.00	75	107	160
Algol	3.00	68	107	165
Cobol	3.00	65	107	170
CM52	3.00	70	107	135
Jovial	3.00	70	107	165
Pascal	3.50	50	91	125
Third generation	4.00	45	80	125
PL/I	4.00	65	80	95
Modula 2	4.00	70	80	90
Ada 83	4.50	60	71	80
Lisp	5.00	25	64	80
Forth	5.00	27	64	85
Quick Basic	5.50	38	58	90
C++	6.00	30	53	125
Ada 9X	6.50	28	49	110
Database	8.00	25	40	75
Visual Basic (Windows)	10.00	20	32	37
APL (default value)	10.00	10	32	45
Smalltalk	15.00	15	21	40
Generators	20.00	10	16	20
Screen painters	20.00	8	16	30
SQL	27.00	7	12	15
Spreadsheets	50.00	3	6	9

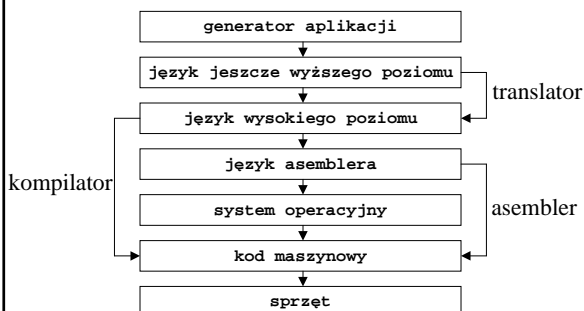
## Historia języków programowania



## Evolution of imperative programming

Decade	Programming technology	Key advance
1940s	machine codes	programmable machines
1950s	assembly languages	symbols
1960s	high-level languages	expressions and machine-independence
1970s	structured programming	structured types and control structures
1980s	modular programming	separation of interface from implementation
1990s	object-oriented programming	polymorphism
2000s	component-oriented programming	dynamic and safe composition

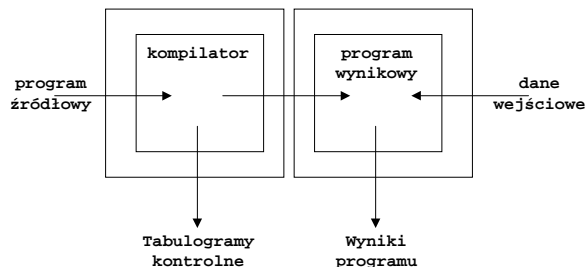
## Translator, kompilator, asembler



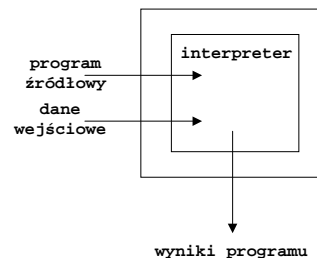
## Programy wspomagające programowanie

- asembler
- kompilator
- translator
- kompilator przechodni (cross compiler)
- linker
- debugger
- profiler
- visual interface builder

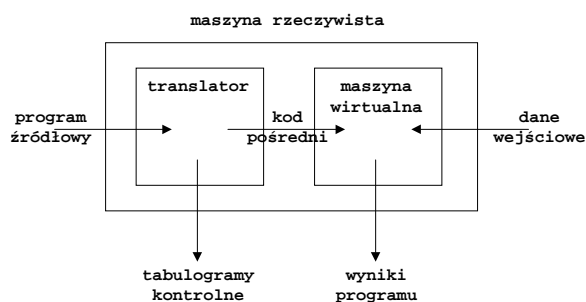
## Kompilacja i wykonanie programu



## Interpretacja programu



## Translacja do kodu pośredniego



## Kompilator - fazy kompilacji

- preprocessing (preprocesor)
- analiza leksykalna (skaner)
- analiza składniowa (parser)
- optymalizacja niezależna od architektury
- optymalizacja zależna od architektury
- dołączanie bibliotek (linker)
- generacja kodu

## Języki programowania ogólnego zastosowania

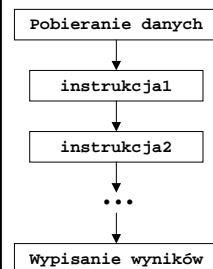
- języki imperatywne (instrukcyjne)
- języki aplikatywne (funkcyjne)
- języki deklaratywne (logiczne)

Problem:

Trójka Pitagorejska to 3 liczby naturalne spełniające równanie  $a^2 + b^2 = c^2$ .

Znaleźć wszystkie trójki, w których c nie przekracza ustalonego N.

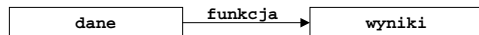
## Język imperatywny (instrukcyjny)



```

Program TrojkiPitagorejskie;
const
  max=10;
var
  a,b,c:integer;
begin
  for c:=1 to m do
    for b:=1 to c-1 do
      begin
        a:=trunc(sqrt(c*c-b*b));
        if a*a+b*b=c*c then
          writeln(a,b,c);
        end;
      end;
    end;
  end.
  
```

## Język aplikacyjny (funkcyjny)



```

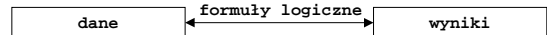
TrojkiPitagorejskie(M)=[], M=1
TrojkiPitagorejskie(M)=T(M,M-1)++TrojkiPitagorejskie(M-1)

T(c,b)=[], b=1
T(c,b)=[(a,b,c)]++T(c,b-1), a=trunc(a) where a=sqrt(c*c-b*b)
T(b,c)=T(c,b-1)

Trojki Pitagorejskie(10)

[(6,8,10),(8,6,10),(3,4,5),(4,3,5)]
  
```

## Język deklaratywny (logiczny)



```

TrojkiPitagorejskie (A,B,C,M)
if Nat(1,C,M) and Nat(1,B,M) and Nat(1,A,M) and A*A+B*B=C*C

Nat (K,X,L)
if K<=L and X=K or K<=L and K1=K+1 and Nat(K1,X,L).

TrójkiPitagorejskie (A,B,C,10)

A=2, B=4, C=5
A=4, B=3, C=5
A=6, B=8, C=10
A=8, B=6, C=10
  
```

## System operacyjny

System operacyjny jest to zbiór procedur (programów) przekształcający maszynę rzeczywistą w wirtualną.

System operacyjny jest to zorganizowany zespół programów, które pełnią rolę pośredniczącą między sprzętem, a użytkownikami, dostarczają użytkownikom zestawu środków ułatwiających projektowanie, kodowanie, uruchamianie i eksploatację programów oraz w tym samym czasie sterują przydziałem zasobów dla zapewnienia efektywnego działania.

## System operacyjny

Podstawowa funkcja systemu operacyjnego to zarządzanie zasobami.

Zasób systemu:

sprzęt lub program, który może być przydzielany systemowi operacyjnemu lub programowi użytkowemu.

Zasoby sprzętowe:

- czas procesora (-ów)
- pamięć operacyjna
- urządzenia we/wy
- inne komputery

Zasoby programowe:

- funkcje systemowe dostarczane programowi użytkownika
- określone obszary pamięci - bufora
- pamięć zewnętrzna
- katalogi i pliki
- translatory, kompilatory

## Kategorie systemów operacyjnych

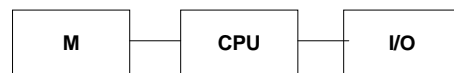
Tryby pracy:

- systemy do przetwarzania wsadowego (off-line, batch)
- systemy z podziałem czasu (on-line)
- system dla działania w czasie rzeczywistym (real-time)

Języki opisu systemów operacyjnych:

- Concurrent Pascal
- Modula-2
- Ada

## Bloki funkcjonalne komputera



DMA

### PAMIĘĆ

Zawsze wolniejsza od procesora.

Przyspieszacze:

- interleaving
- cache

### PROCESSOR

Najszybsza część komputera.

Równoległość:

- pipeline
- superscalar

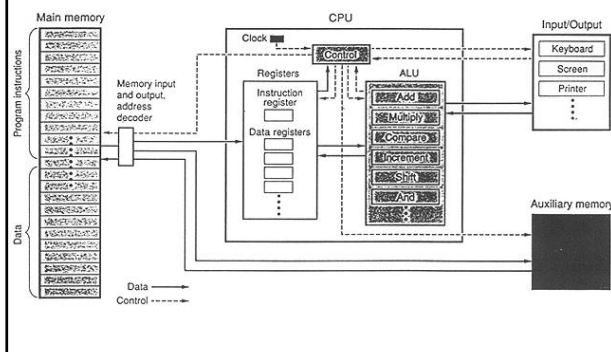
### WEJŚCIE-WYJŚCIE

Wąskie gardło systemu.

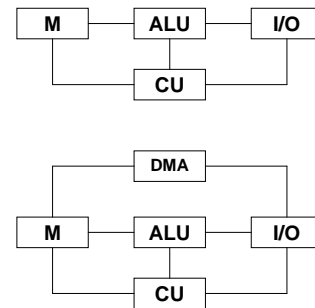
Wczesne wynalazki:

- kanały we-wy (DMA)
- przerywania

## Architecture of a Serial Computer

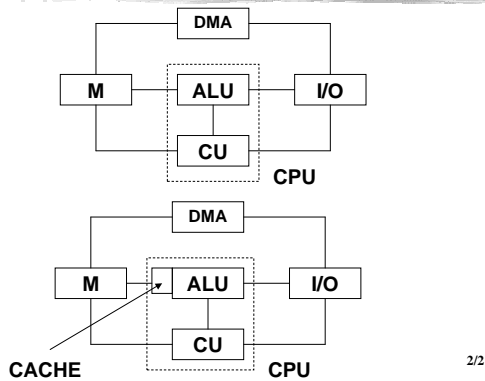


## Architektura von Neumanna



1/2

## Architektura von Neumanna



2/2

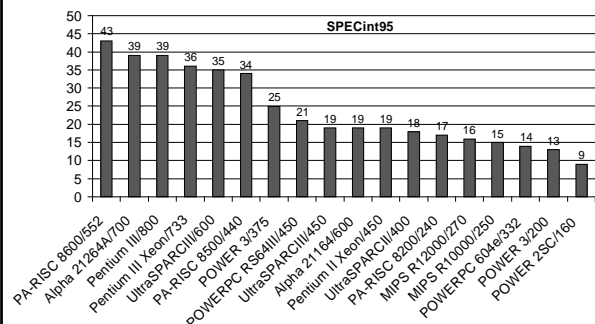
## Miara wydajności - MIPS

Komputer	Liczba procesorów	MIPS
DEC VAX 11/780 (rok 1978)	1	1
IBM S/390 G5/RX6	10	901
HP V2500 (440MHz)	24	1550
SUN E10000 (400MHz)	64	3000

## Miara wydajności - MFlops

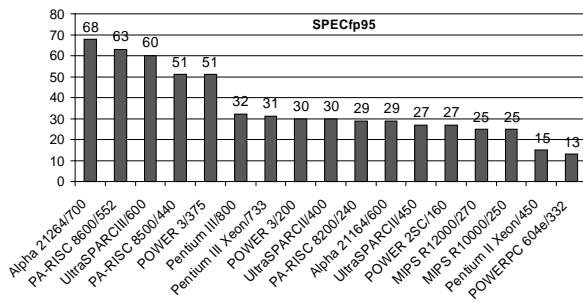
Komputer	DP Mflop/s	TPP Mflop/s	Rpeak Mflop/s
Cray T9xx (32 proc 2.2 ns)	-	29360	57600
Cray T9xx (8 proc 2.2 ns)	-	10880	14400
Cray T9xx (1 proc 2.2 ns)	705	1603	1800
HP N4000 (8 proc. 440 MHz)	-	6410	14080
HP N4000 (1 proc. 440 MHz)	375	1290	1760
SUN HPC 450 (4 proc. 400 MHz)	-	1841	3200
SUN H PC 450 (1 proc. 400 MHz)	183	552	800
PC PII Xeon 450 MHz	98	295	450

## Stałoprzecinkowa wydajność procesora

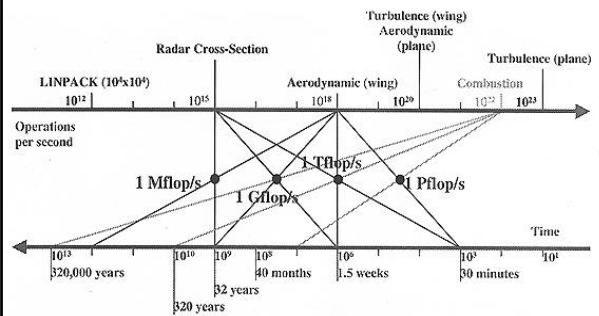




## Zmienneprzecinkowa wydajność procesora



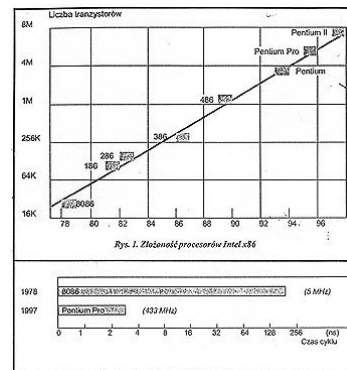
## The need for computer power



## Ważniejsze wydarzenia z historii komputerów

1945	• John von Neumann: "The First Draft of a Report on the EDVAC"
1946	ENIAC Pierwsza elektroniczna maszyna cyfrowa (18 tysięcy lamp, 1500 przełączników)
1951	UNIVAC I Pierwszy komputer komercyjny (48 sprzedanych egzemplarzy)
1952	IAS Realizacja EDVAC'a (J. von Neumann, Princeton)
1960	PDP-1 Pierwszy minikomputer (DEC, 50 egzemplarzy)
1964	IBM S/360 Pierwsza rodzina komputerów; pojęcie "architektury"
1965	PDP-8 Początek ery minikomputerów (DEC, 50 tys. Egzemplarzy)
1971	Intel 4004 Pierwszy mikroprocesor (4-bitowy)
1974	Intel 8080 Pierwszy "CPU on a chip"; protoplasta "8-bitowców": Z80, 8085
1974	CRAY-1 "Supercomputer" (pierwszy produkt Cray Research, maksymalna szybkość 150 M flops)
1977	Intel 8088 Pierwszy "computer on a chip", nowsza wersja (stosowana do dziś): 8051
1978	VAX Pierwszy "superminikomputer" (DEC, 32-bitowy)
1981	IBM PC Początek ery "pecetów" (procesor Intel 8088, system operacyjny MS DOS)
1981	Dataflow Machine Pierwszy działający komputer "dataflow" (Univ. of Manchester)
1982	RISC I Nowa koncepcja architektury (Berkeley); również procesor MIPS (Stanford); IBM ujawnia swoje wcześniejsze prace nad modelem IBM 801
1996	* 71 mln PC sprzedanych na świecie * 16 mln komputerów w Internecie

## Szybkość procesorów Intel x86

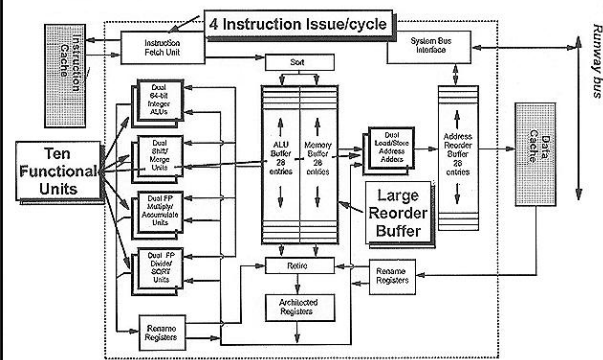


## Porównanie mikroprocesorów

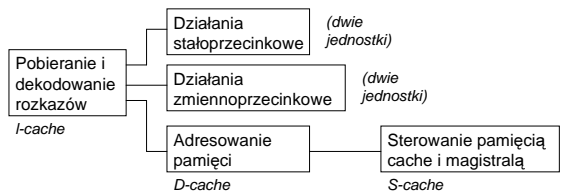
Procesor (producent)	Rozkazy/cykli	Cache I, D (KB)	Liczba wyprowadzeń	Zegar (MHz)	SPECint95	SPECfp95	Liczba tranzystorów (mln)
21164 Alpha (DEC)	4	16+96	499	500	15,4	21,1	9,3
PowerPC 604e (IBM/Motorola)	4	32	255	225	9,0	7,5	5,1
Pentium Pro (Intel)	3	16 + (256)	387	200	8,2	6,7	5,5
Pentium II (Intel)	?	32 + (512)	242	300	11,7	8,15	7,5
x704 (Exponential Technology)	3	4+32	?	533	15,0	?	?
PA-8000 (Hewlett-Packard)	4	(1 do 4K)	1085	180	11,8	20,2	3,9
R10000 (MIPS)	4	64	527	275	12,0	24,0	5,9
UltraSPARC II (Sun)	4	32	521	250	8,5	15	3,8
4004 (Intel) - 1971	-	-	16	0,75			0,0023

Wybrane mikroprocesory dostępne w 1997 r.

## PA-8000 - Block Diagram



## Struktura Alpha 21164

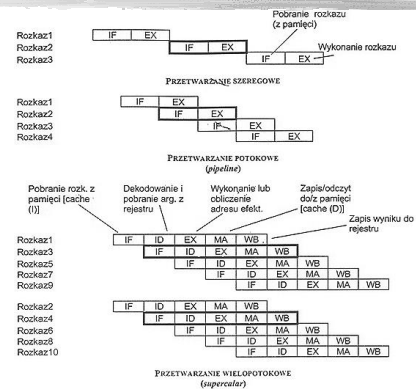


S0 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12

Stopnie przetwarzania potokowego →

Struktura Alpha 21164 (pięć jednostek funkcjonalnych działających współbieżnie)

## Sposoby przetwarzania strumienia rozkazów



## Złożoność czasowa algorytmu

Funkcja złożoności obliczeniowej algorytmu A rozwiązującego dany problem to funkcja przyporządkowująca każdej wartości rozmiaru konkretnego problemu maksymalną liczbę kroków elementarnych (lub jednostek czasu) maszyny cyfrowej potrzebnych do rozwiązania za pomocą algorytmu A konkretnego problemu o tym rozmiarze.

## Złożoność czasowa algorytmu

**Algorytm wielomianowy** (o złożoności czasowej wielomianowej) to taki, którego złożoność jest  $O(p(n))$ , gdzie  $p(n)$  jest wielomianem, a  $n$  jest rozmiarem problemu.

**Algorytm wykładniczy** (o złożoności czasowej wykładniczej) to taki, który nie jest wielomianowy.

## Notacja $O(.)$

Funkcja  $f(k)$  jest rzędu  $g(k)$ , co zapisujemy  $O(g(k))$ , jeżeli istnieje taka stała  $c$ , że  $f(k) \leq c \cdot g(k)$  dla prawie wszystkich wartości  $k$ .

Przykłady:

$O(\log N)$  logarytmiczna

$O(N)$  liniowa

$O(N \cdot \log N)$

$O(N^2)$  kwadratowa (wielomianowa)

$O(2^N)$  wykładnicza

Problem stałej:

$f1(N) = 10 \cdot N$

$f2(N) = 1000 \cdot \log N$

## Złożoność czasowa a czas wykonania

Rozmiar problemu Funkcja złożoności	10	20	30	40	50
$n$	$10 \cdot 10^{-6}$ sekundy	$20 \cdot 10^{-6}$ sekundy	$30 \cdot 10^{-6}$ sekundy	$40 \cdot 10^{-6}$ sekundy	$50 \cdot 10^{-6}$ sekundy
$n \log_2 n$	$33,2 \cdot 10^{-6}$ sekundy	$86,4 \cdot 10^{-6}$ sekundy	$147,2 \cdot 10^{-6}$ sekundy	$212,9 \cdot 10^{-6}$ sekundy	$282,5 \cdot 10^{-6}$ sekundy
$n^2$	$0,1 \cdot 10^{-3}$ sekundy	$0,4 \cdot 10^{-3}$ sekundy	$0,9 \cdot 10^{-3}$ sekundy	$1,6 \cdot 10^{-3}$ sekundy	$2,5 \cdot 10^{-3}$ sekundy
$n^3$	$1 \cdot 10^{-3}$ sekundy	$8 \cdot 10^{-3}$ sekundy	$27 \cdot 10^{-3}$ sekundy	$64 \cdot 10^{-3}$ sekundy	$125 \cdot 10^{-3}$ sekundy
$2^n$	0,001 sekundy	1 sekunda	17,9 minuty	12,7 dnia	35,7 lat
$3^n$	0,059 sekundy	58,1 minuty	6,53 roku	3 855 wieków	$2,3 \cdot 10^8$ wieków
$10^n$	2,8 godziny	31710 wieków	$3,17 \cdot 10^{14}$ wieków	$3,17 \cdot 10^{24}$ wieków	$3,17 \cdot 10^{34}$ wieków

## Maksymalne rozmiary problemu rozwiązywalnego w danym czasie

Złożoność czasowa	Rozmiar maksymalnego problemu rozwiązywalnego w ciągu		
	1 sekundy	1 minuty	1 godziny
$n$	1 000 000	60 000 000	3 600 000 000
$n \log_2 n$	62 746	2 801 417	133 378 058
$n^2$	1 000	7 745	60 000
$n^3$	100	391	1 532
$2^n$	19	25	30
$3^n$	12	16	20
$10^n$	6	7	9

## Efekt przyspieszania obliczeń

Złożoność czasowa	Rozmiar maksymalnego problemu rozwiązywalnego w ciągu 1 godziny na komputerze			
	aktualnym	10-krotnie szybszym	100-krotnie szybszym	1000-krotnie szybszym
$n$	$n_1$	$10n_1$	$100n_1$	$1000n_1$
$n^2$	$n_2$	$3,16 n_2$	$10n_2$	$31,62 n_2$
$n^3$	$n_3$	$2,15 n_3$	$4,63 n_3$	$10n_3$
$2^n$	$n_4$	$n_4+3,32$	$n_4+6,64$	$n_4+9,97$
$3^n$	$n_5$	$n_5+2,1$	$n_5+4,19$	$n_5+6,29$
$10^n$	$n_6$	$n_6+1$	$n_6+2$	$n_6+3$

## Złożoność algorytmów

Problem wyszukiwania elementu w tablicy

- Wyszukiwanie liniowe  $O(N)$
- Wyszukiwanie binarne  $O(\log N)$

N	liniowy	binarny
10	10	4
$10^3$	$10^3$	10
$10^6$	$10^6$	20

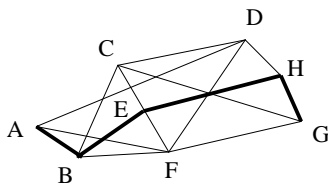
## Złożoność algorytmów

Problem sortowania tablicy

- Metody proste  $O(N^2)$
- Metody szybkie  $O(N \cdot \log N)$

N	proste	szybkie
10	100	33
$10^6$	$10^{12}$	$2 \cdot 10^7$

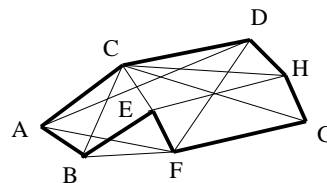
## Problem najkrótszej drogi



Rozwiązanie: droga ABEHG

Znane algorytmy  $O(N^2)$

## Problem komiwojażera



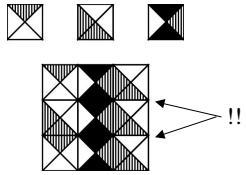
Rozwiązanie: droga ABCDHGFEB

Prosty algorytm  $O(N!)$

Znany algorytm  $O(2^N)$

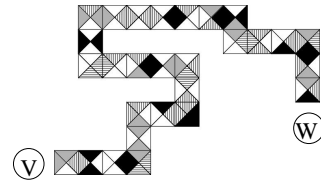


### Problem domina



Rodzaje kafelków, którymi nie można  
pokryć nawet małych obszarów

### Problem domina - wąż



Wąż domino łączący punkty V i W