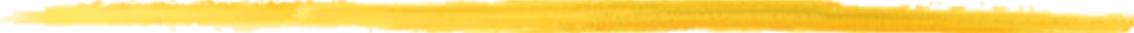


Wykład 1



- 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. Tomasz Jurczyk

poniedziałek 16:00 – 19:00

wtorek 14:30 – 17:30

Cel wykładu:

wprowadzenie podstawowych pojęć związanych z informatyką,
zapoznanie z programowaniem w języku proceduralnym.

Slajdy z wykładu:

System Moodle

Wstęp do informatyki



Wyznaczanie oceny końcowej:

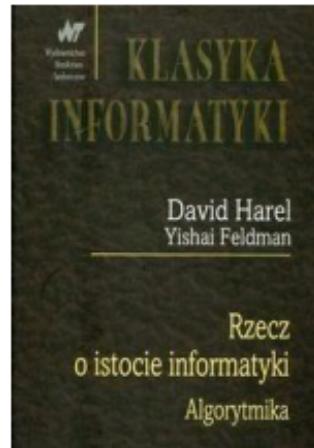
1. Aby uzyskać pozytywną ocenę końcową niezbędne jest uzyskanie pozytywnej oceny z ćwiczeń oraz egzaminu.
2. Obliczamy średnią arytmetyczną z ocen zaliczenia ćwiczeń i egzaminów uzyskanych we wszystkich terminach.
3. Wyznaczamy ocenę końcową na podstawie zależności:
if $sr > 4.75$ then $ok := 5.0$ else
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)



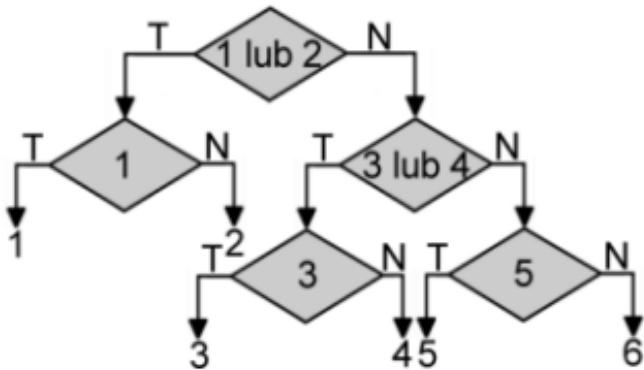
Informatyka to nauka o przetwarzaniu informacji.

Aktualnie obejmuje wiele zagadnień, między innymi:

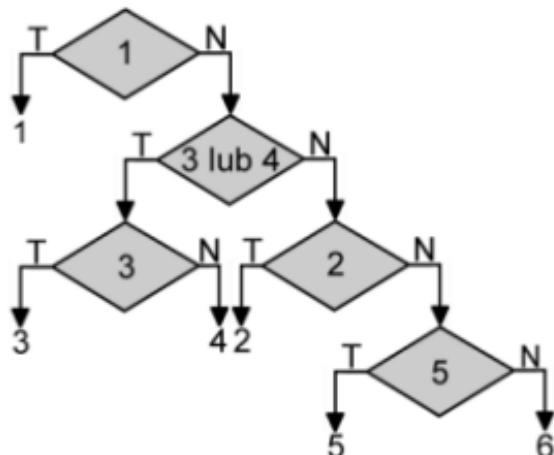
- algorytmika, struktury danych, języki programowania
- mikroprocesory, architektury komputerów
- bazy danych, systemy operacyjne, sieci komputerowe
- kompilatory, kryptografia, metody numeryczne
- inżynieria oprogramowania, projektowanie systemów
- grafika, sztuczna inteligencja, aplikacje internetowe
- złożoność obliczeniowa
- ...

Identyfikacja elementów zbioru

S1:



S2:

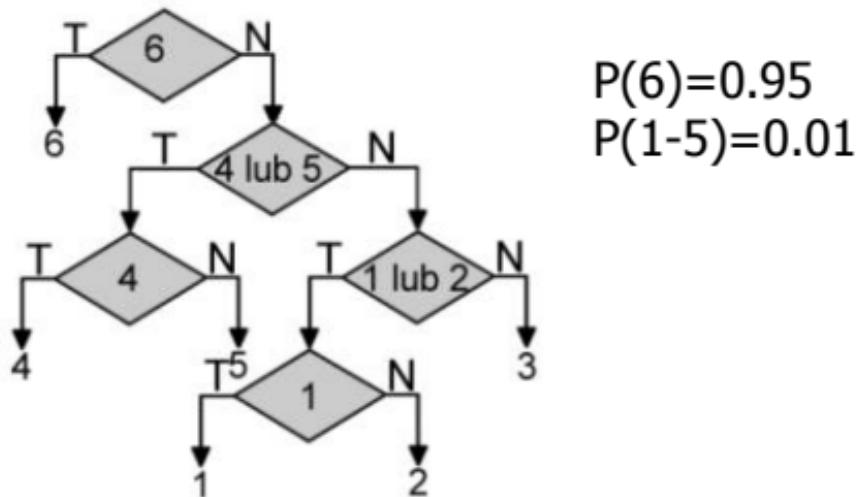


$$E(S1) = \frac{1}{6} \cdot 2 + \frac{1}{6} \cdot 2 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 3 = 2.66$$

$$E(S2) = \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 4 + \frac{1}{6} \cdot 4 = 3$$

Identyfikacja elementów zbioru

S3:



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

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

Teoria informacji

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

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



Claude Shannon

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

gdzie:

p_i ($p_i > 0$, $\sum p_i = 1$) 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 Huffmmana

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
2) 0 0 10 0 110 0 111 0 10 0

20 bitów
16 bitów

Ilość informacji

$$H(X) = 0.6 \cdot \log(0.6) + 0.2 \cdot \log(0.2) + 0.1 \cdot \log(0.1) + 0.1 \cdot \log(0.1) = 15.7$$

Podstawowe pojęcia

Zadanie algorytmiczne – polega na określeniu:

- wszystkich poprawnych danych wejściowych
- oczekiwanych wyników jako funkcji danych wejściowych

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

Algorytm może występować w postaci:

- werbalnej (*opis słowny*)
- symbolicznej (*schemat blokowy*)
- programu

Przykład zapisu algorytmu

Problem: równanie kwadratowe

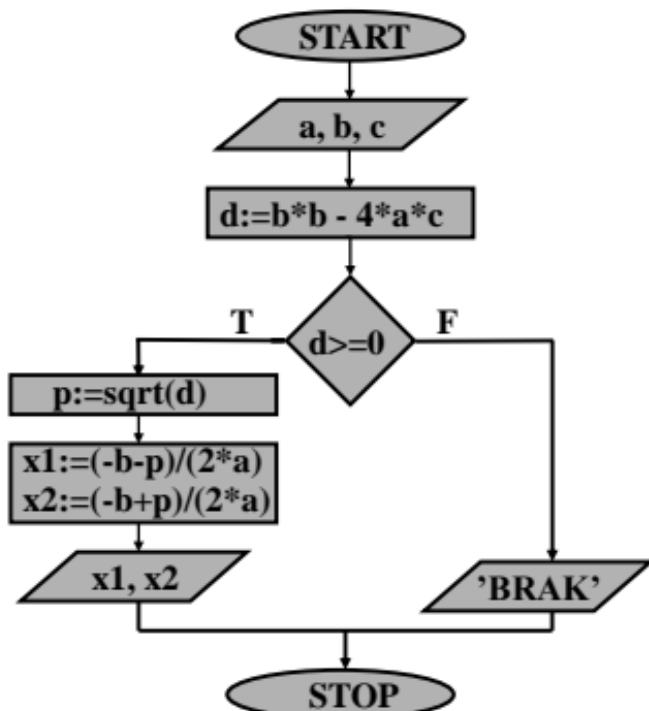
Dane: współczynniki **a, b, c**

Wyjście: pierwiastki **x1, x2** lub informacja o ich braku

Postać verbalna 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)

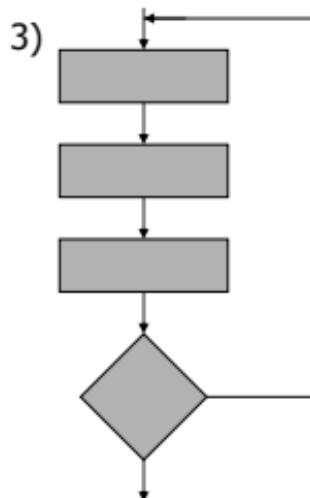
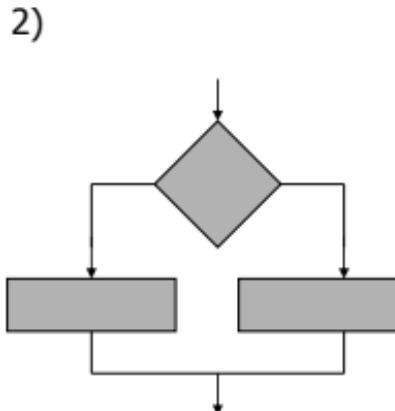
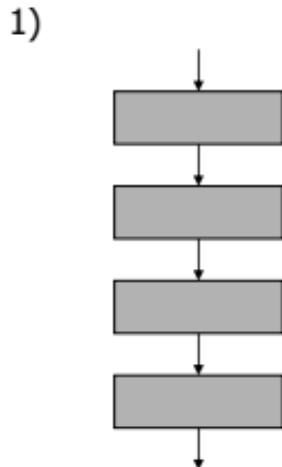


```
begin
    read(a,b);
    while a<>b do
        if a>b then
            a:=a-b
        else
            b:=b-a;
    writeln(a)
end.
```

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, alokacja pamięci

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 writeln(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?

Wykład 2



- Aspekty języka programowania
- Instrukcje w języku proceduralnym
- Konstrukcje 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 **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)
- Produkce
- Metasymbole

< >

- symbol pomocniczy

::=

- symbol produkcji

|

- symbol alternatywy

[]

- wystąpienie 0 lub 1 raz (EBNF)

{ }

- powtórzenie 0 lub więcej razy (EBNF)

Przykład

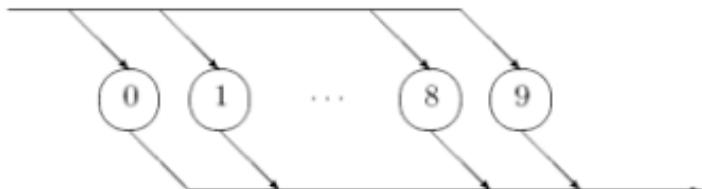
<cyfra dziesiętna> ::= 0|1|2|3|4|5|6|7|8|9

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

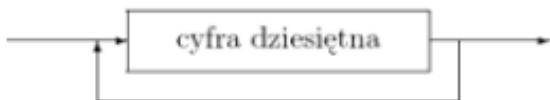
<liczba> ::= + <liczba bez znaku> | - <liczba bez znaku>

Diagramy składniowe

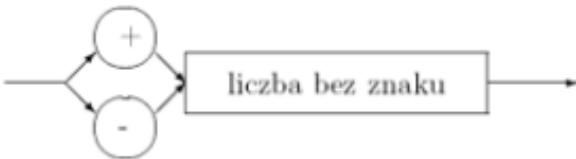
cyfra dziesiętna



liczba bez znaku



liczba



Semantyka

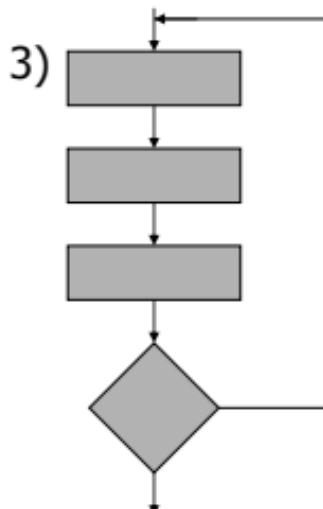
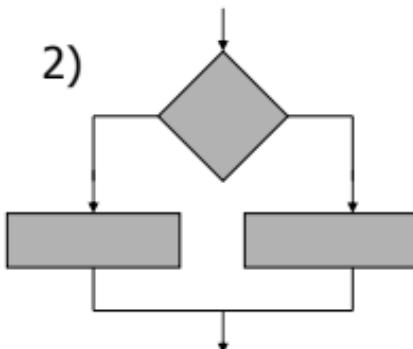
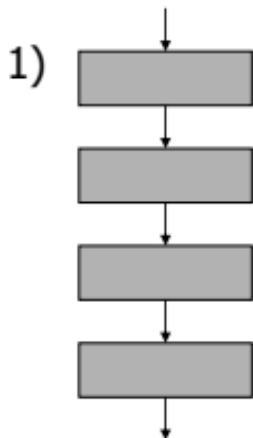


Opis każdej konstrukcji językowej poprawnej składniowo

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

Struktury sterujące przebiegiem programu

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



Instrukcje proste i strukturalne



proste

- przypisania
- wywołania funkcji
- operacja we/wy

strukturalne

- warunkowa
- wyboru
- iteracyjne

Instrukcja przypisania

`<zmienna> = <wyrażenie>;`

Przykłady:

`a = 0;`

`a = a+1;`

`y = sin(6*x);`

Problemy:

- Zgodność typów
- Konwersja typów (rzutowanie typów)
- Zakres liczb
- Nieokreśloność zmiennych

Instrukcja wywołania funkcji

```
nazwa();  
x=nazwa();  
nazwa(p1,p2,p3);
```

Przykłady:

```
y=sqrt(x);  
printf("%d",x);  
rozwiaz(a,b,c);
```

Problemy:

- Błędna liczba argumentów
- Niezgodność typu argumentów

Operacja we/wy



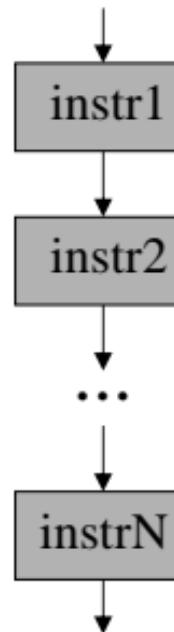
```
cout << wyr;  
cin >> zm;
```

Przykłady:

```
cout << x;  
cout << x+y;  
cout << "hello";  
cin >> x;
```

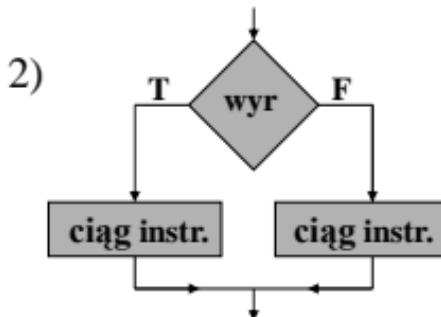
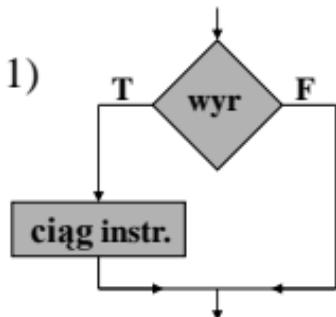
Ciąg instrukcji

*<instrukcja1>;
<instrukcja2>;
...
<instrukcjaN>;*



Instrukcja skoku warunkowego

- 1) **if** (<wyrażenie>) { <ciąg instrukcji> }
- 2) **if** (<wyrażenie>) { <ciąg instrukcji> }
else { <ciąg instrukcji> }

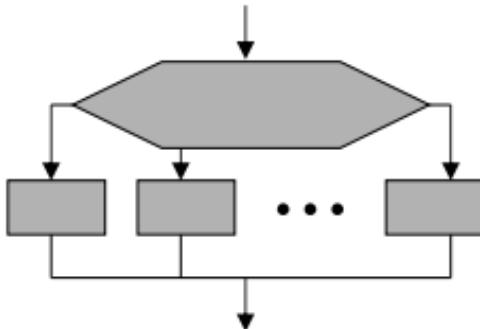


Instrukcja wyboru

```
switch (<wyrażenie>) {  
    case <etykieta1> : <instr1>; break;  
    case <etykieta2> : <instr2>; break;  
    ...  
    default <instr>;  
}
```

Przykład:

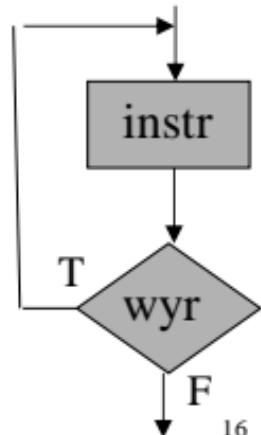
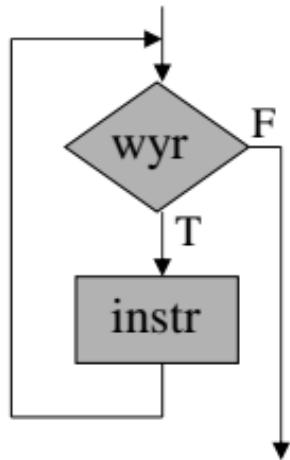
```
switch (dzien_tygodnia) {  
    case 0 : printf("niedziela"); break;  
    case 1 : printf("poniedziałek"); break;  
    ...  
    case 6 : printf("sobota"); break;  
    default : printf("zla wartosc");  
}
```



Instrukcje pętli

while (<wyrażenie>) { <ciąg instrukcji> }

do { <ciąg instrukcji> } **while** (<wyrażenie>);



Konstrukcje strukturalne

begin ... end

if ... then ...

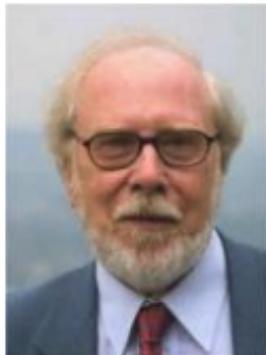
if ... then ... else ...

case ... of ...

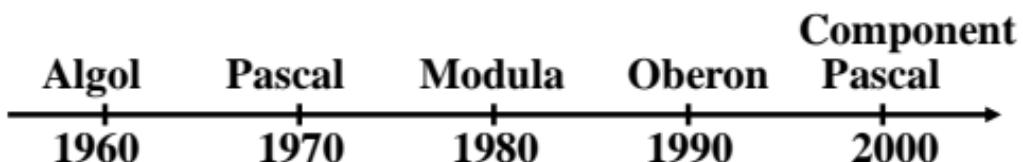
while ... do ...

repeat ... until ...

for ... to ... do ...

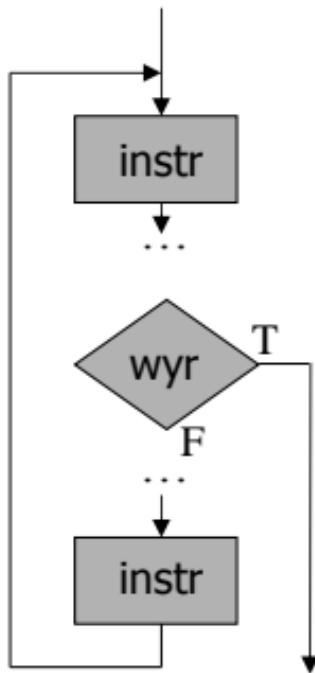


Niklaus Wirth



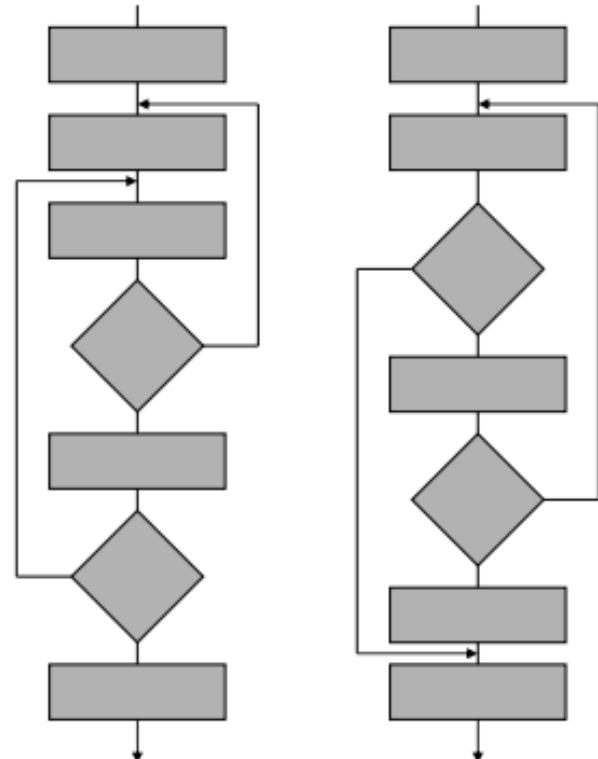
Instrukcja break

```
while (True) {  
    ...  
    if (<wyrażenie>) break;  
    ...  
}
```



Instrukcja skoku

```
// instrukcje  
etykieta:  
// instrukcje  
  
if (<wyrażenie>) {  
    goto etykieta;  
}  
  
goto etykieta;
```



Pytanie: jak zrealizować to bez użycia goto ?

Instrukcja pętli for

```
for (<inst1> ; <war> ; <inst2>) <inst3>;  
  
<inst1>;  
while (<war>) {  
    <inst3>;  
    <inst2>;  
}
```

Przykład:

```
s=0;  
for (i=0; i<10; i=i+1) s=s+i;
```

Pytania i zadania

- Zapisać w notacji EBNF składnie instrukcji warunkowej oraz pętli.
- Które z 6 instrukcji: *if*, *if else*, *case*, *while*, *do while*, *for* można usunąć z języka aby nadal można było w nim programować?
- Liczb pierwszych jest nieskończoność wiele. Tylko kilka z nich spełnia warunek:

$$\text{sum_p}(N)=N$$

gdzie:

$\text{sum_p}(N)$ to suma p -tych potęg cyfr p -cyfrowej liczby N

$$\text{np. } \text{sum_p}(2012)=16+0+1+16=33$$

Należy napisać program odnajdujący wszystkie takie liczby.

A co z naszym programem?

Wykład 3



- Typy danych
- Typy skalarne
- Operacje na typach skalarnych
- Typ tablicowy

Typy danych



- Skalarne
- Strukturalne
- Wskaźnikowe

Typy skalarne - uporządkowane i skończone zbiory wartości.

Przykłady:

logiczny - bool

znakowy - char

"całkowity" - int

"rzeczywisty" - float

Działania na typach skalarnych

- **typ logiczny** (*bool*)

True False

! || && == !=

- **typ całkowity** (*int, long int, long long int*)

12 -21

+ - * / % == != < <= > >=

- **typ rzeczywisty** (*float, double*)

12.3 2e-23

+ - * / == != < <= > >=

- **typ znakowy** (*char, unsigned char*)

'a' 'b' '\n'

== != < <= > >=

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 całkowite

FPC

Type	Range	Size in bytes
Byte	0 .. 255	1
Shortint	-128 .. 127	1
Smallint	-32768 .. 32767	2
Word	0 .. 65535	2
Integer	either smallint or longint	size 2 or 4
Cardinal	longword	4
Longint	-2147483648 .. 2147483647	4
Longword	0 .. 4294967295	4
Int64	-9223372036854775808 .. 9223372036854775807	8
QWord	0 .. 18446744073709551615	8

Liczby całkowite



C/C++

- short int, int, long int, long long int
- signed, unsigned

Standard C99:

- int ma minimum 16 bitów
- long int jest co najmniej takiego rozmiaru, co int
- long long int ma minimum 64 bity

W praktyce:

- short int ma 16 bitów
- int jest równy long int i ma 32 bity
- long long int ma 64 bity

Liczby zmiennopozycyjne

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, C/C++

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

Funkcje standardowe

- cos() sin() tan() acos() asin() atan()
- cosh() sinh() tanh() acosh() asinh() atanh()
- exp() log() log10() log2()
- pow() sqrt() cbrt()
- fabs() abs() ceil() floor()
- system() abort() exit()

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	Ҫ	144	É	160	á	176	ܶ	192	ܲ	208	ܴ	224	ܵ	240	=
129	Ӯ	145	ӕ	161	ି	177	ܰ	193	ܳ	209	ܵ	225	ܷ	241	ܻ
130	େ	146	Ӕ	162	୍ୟ	178	ܱ	194	ܴ	210	ܶ	226	ܸ	242	ܼ
131	ା	147	ୠ	163	ୁ	179	ܲ	195	ܵ	211	ܷ	227	ି	243	୤
132	ା	148	ୠ	164	ନ	180	ܲ	196	ܲ	212	ܷ	228	୪	244	ମ
133	ା	149	ୠ	165	ଝ	181	ܲ	197	ܵ	213	ܷ	229	୧	245	ଜ
134	ା	150	୦	166	୩	182	ܲ	198	ܵ	214	ܷ	230	୩	246	୷
135	୯	151	୹	167	୦	183	ܲ	199	ܵ	215	ܷ	231	୫	247	୧
136	୧	152	ୟ	168	୧	184	ܲ	200	ܷ	216	ܷ	232	୩	248	୦
137	୧	153	୕	169	୮	185	ܲ	201	ܷ	217	ܷ	233	ୣ	249	.
138	୧	154	୔	170	୭	186	ܲ	202	ܷ	218	ܷ	234	୧	250	.
139	୧	155	୦	171	୮	187	ܲ	203	ܵ	219	ܷ	235	୧	251	୪
140	୧	156	୬	172	୯	188	ܲ	204	ܵ	220	ܷ	236	୧	252	୮
141	୧	157	୩	173	୧	189	ܲ	205	=	221	ܷ	237	୪	253	୧
142	ୀ	158	୧	174	୮	190	ܲ	206	ܷ	222	ܷ	238	୧	254	୧
143	ୀ	159	୧	175	୯	191	ܲ	207	ܷ	223	ܷ	239	୮	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	estoniski, 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. alfabetu 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: 0xxxxxx

00000080 – 000007FF: 110xxxxx 10xxxxxx

00000800 – 0000FFFF: 1110xxxx 10xxxxxx 10xxxxxx

00010000 – 001FFFFFF: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

00200000 – 03FFFFFF: 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

04000000 – 7FFFFFFF: 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

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

Typ tablicowy

Deklarowanie zmiennych:

```
int t1[10];  
double t2[3][4];  
int t[ ] = { 0,3,3,6,1,4,6,2,5,0,3,5 };
```

Odwołanie do elementów:

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

Cechy:

- statyczny rozmiar
- elementy indeksowane (numerowane) od 0

Problemy:

- odwołanie poza obszar tablicy

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 $n < 200$.

Wykład 4



- Strukturalne typy danych
 - Tablice
 - Tablice znaków (napisy)
 - Struktury (rekordy)
 - Pliki

Typ tablicowy

Deklarowanie zmiennych:

```
int t1[10];  
double t2[3][4];  
int t[ ] = { 0,3,3,6,1,4,6,2,5,0,3,5 };
```

Odwołanie do elementów:

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

Cechy:

- statyczny rozmiar
- elementy indeksowane (numerowane) od 0

Problemy:

- odwołanie poza obszar tablicy

Tablice znaków (napisy)

Stałe napisowe:

```
write('to jest tekst');  
printf("to jest tekst");  
cout << "to jest tekst";
```

Pascal

Język C

Język C++

Deklarowanie:

```
char buf[11];  
char imie[] = "Jola";
```

Operacje we/wy:

```
cin >> buf;  
cout << buf;
```

Uwaga na spacje!

Reprezentacja:

J	o	l	a	\0					
0	1	2	3	4	5	6	7	8	9

Operacje na łańcuchach

Znaki specjalne:

- '\n' nowy wiersz chr(10)
- '\r' nowy wiersz chr(13)
- '\t' tabulator poziomy
- '\v' tabulator pionowy
- '\\' \ (ang. backslash)
- '\'' apostrof
- '\"' cudzysłów

Operacje na łańcuchach

Funkcje:

```
size_t strlen(const char *s);  
char *strcpy(char *s1, const char *s2);  
char *strcat(char *s1, const char *s2);  
int strcmp(const char *s1, const char *s2);  
char *strstr(const char *s1, const char *s2);
```

Funkcje we/wy:

```
int printf(const char *format,...);  
int scanf(const char *format,...);  
char *gets(char *s);  
int puts(const char *s);  
int getch(void);
```

Struktury (Rekordy)

Definiowanie:

```
struct zespolona  
{  
    double re;  
    double im;  
};
```

Deklarowanie zmiennych:

```
zespolona z1,z2;  
zespolona t[100];
```

Nadawanie wartości:

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

Struktury - przykład

```
struct data
{
    int dzien;
    int miesiac;
    int rok;
};

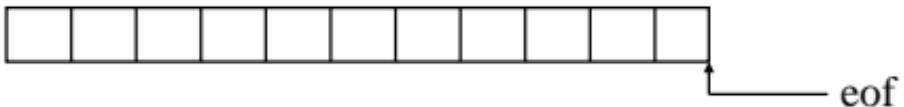
struct osoba
{
    char imie[20];
    char nazwisko[30];
    data urodziny;
    bool kobieta;
};
```

Pliki

Dostęp do pliku:

- sekwencyjny (taśmy)
- swobodny (dyski)

Struktura o elementach tego samego typu



Znak końca wiersza w pliku tekstowym:

- DOS CR LF
- UNIX LF
- MacOS CR

Pliki operacje

Język C++:

```
#include <fstream>

ofstream wyjscie("wyniki.txt");
wyjscie << x1 << " " << x2;
wyjscie.close
```

Język C

```
#include <stdio.h>

wyjscie=fopen("wyniki.txt","w");
fprintf(wyjscie,"%d %d",x1,x2);
fclose(wyjscie);
```

Pliki operacje



Obsługa błędów (C++):

```
#include <iostream>

int a,b;

ifstream wejscie("dane.txt");
if (!wejscie)
{
    cout << "nie mogę otworzyć pliku";
    return 1;
}
wejscie >> a >> b;
wejscie.close();

wyjscie << a << " " << b;
```


Wykład 5-6



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

Budowa programu

Program w języku imperatywnym składa się:

- opisu danych, struktur danych
- opisu czynności do wykonania (*instrukcji*)

Struktura programu

#include ...

import bibliotek

int a,b,c;

deklaracja zmiennych globalnych

int ala(...)
{ ... }

definicje funkcji

int main()
{ ... }

główna funkcja

Procedury i funkcje



Cel stosowania:

- dekompozycja problemu
- wielokrotne wykonanie
- poziomy abstrakcji
- oddzielna kompilacja

Projektowanie algorytmu:

- syntetyczne (*bottom-up*)
- analityczne (*top-down*)

Składnia definicji funkcji

```
typ_wyniku nazwa_funkcji ( parametry formalne)
{
    wewnętrze_funkcji;
}
```

Przykład:

```
double srednia( double x, double y)
{
    return (x+y)/2;
}
```

Zagłębianie procedur (Pascal)

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

Deklarowanie funkcji



```
void jeden(...)  
{  
    ...  
}
```

```
double dwa(...)  
{  
    ...  
}
```

```
int main()  
{  
    ...  
}
```

Przykładowa funkcja

Problem: obliczanie wartości $c=a^b$

```
void power(double a, int b, double &c)
{
    int i;
    i = b; c = 1.0;
    while (i>0)
    {
        c = c * a;
        i = i-1;
    }
}
```

Deklaracja funkcji:

- identyfikator funkcji
- nagłówek funkcji
- parametry formalne
- treść funkcji

Wywołanie funkcji:

- aktualne parametry

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

Przekazywanie parametrów

- przez wartość
- przez referencję

Użycie:

Przez wartość

dane do funkcji

Przez referencję

dane z funkcji

dane do i z funkcji

Przekazywanie parametrów

Przez wartość

```
void p1(int a)
{
    a = a+1;
    cout << a;
}

void main()
{
    p1(2);          // 3
    b = 5;
    p1(b);          // 6
    cout << b;      // 5
}
```

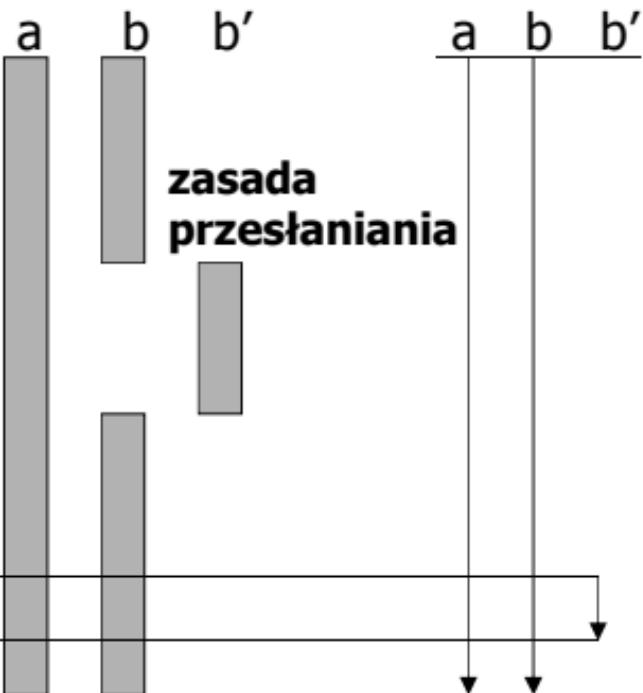
Przez referencję

```
void p1(int &a)
{
    a = a+1;
    cout << a;
}

void main()
{
    b = 5;
    p1(b);          // 6
    cout << b;      // 6
}
```

Obszar określoności i czas trwania

```
int a,b;  
void x( ... )  
{  
    double b;  
    b=1.5;  
}  
int main()  
{  
    b=3;  
    x( ... );  
}
```



Przykłady zastosowania

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

```
int silnia(int n)
{
    int i,s;      // zmienne lokalne
    s = 1;
    for (i=1; i<=n; i++) s = s*i;
    return s;
}
```

Przykłady zastosowania

Procedura (funkcja) rekurencyjna:

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

```
int silnia(int n)
{
    if (n<2)
        return 1;
    else
        return n*silnia(n-1);
}
```

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

```
int fib(int n)
{
    if (n<3)
        return 1;
    else
        return fib(n-1)+fib(n-2);
}
```

Problem rekurencji

```
int w;          // zmienna globalna

int fib(int n)
{
    cout << "start" << n << endl;

    if (n<3) return 1;
    else return fib(n-1)+fib(n-2);

    cout << "stop" << n << endl;
}

int main()
{
    w = fib(5);
    cout << w;
}
```

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

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.

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 7-8



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

Zmienna

Aspekty zmiennej:

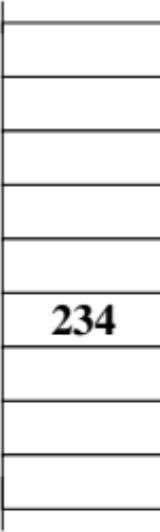
- nazwa
- adres (lokacja)
- wartość
- typ
- rozmiar

```
int x;  
  
{  
    ...  
    x=234;  
    ...  
}
```

32000

32005

234



Instrukcja podstawienia

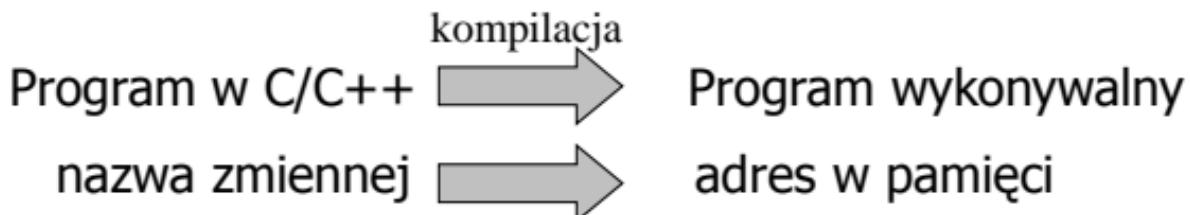
<zmienna> = <wyrażenie>

$z1 = z2 = z3 = \text{wyrażenie}$ (podstawienie wielokrotne)

$z1 := z2$ (podstawienie symetryczne)

$z \ op = \text{wyr} \quad \longleftrightarrow \quad z = z \ op \text{ wyr}$

Czas istnienia nazwy



```
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^\wedge	$*a$

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

Język Turbo Pascal

Zmienna wskaźnikowa

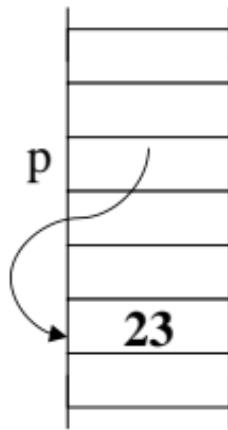
Zmienna wskaźnikowa - zmienna, która przechowuje adres innej zmiennej.

Zmienna wskazywana - zmienna, na którą wskazuje zmienna wskaźnikowa.



Deklaracja zmiennej wskaźnikowej:

```
int i=23;  
int *p;  
  
p = &i;  
*p = 29;
```

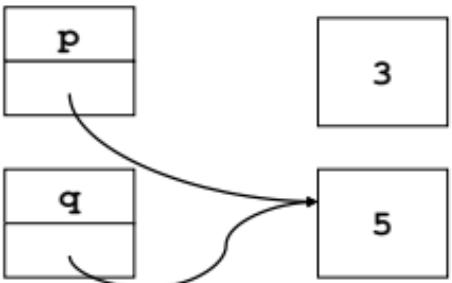


Zmienna wskaźnikowa i wskazywana

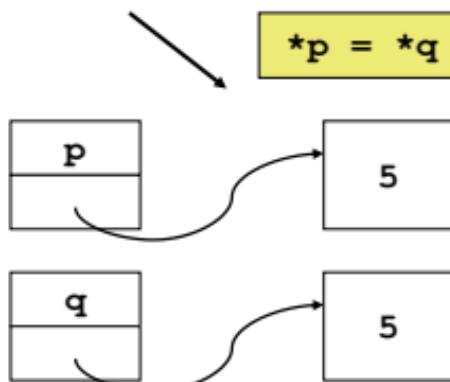
```
{  
    int *p;  
    int *q;  
    ...  
    *p = 3;  
    *q = 5;  
    ...  
}
```



$p = q$

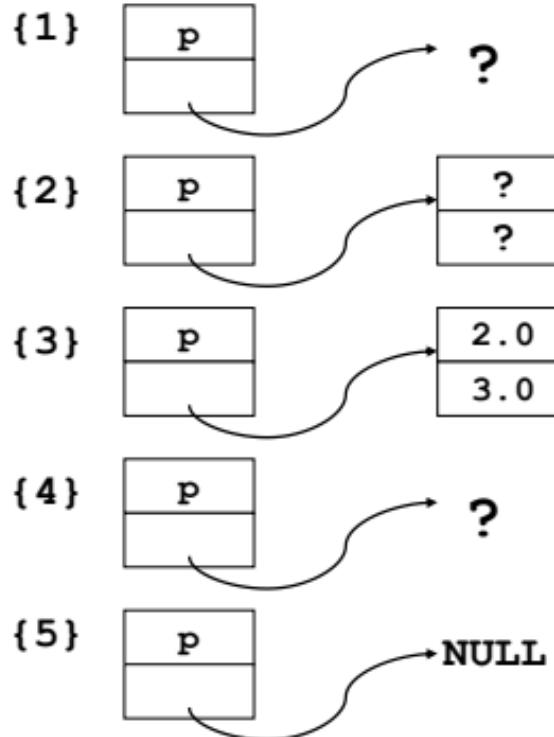


$*p = *q$



Alokacje i zwalnianie pamięci

```
struct punkt {  
    double x,y;  
};  
  
punkt *p;          {1}  
  
p = new punkt;    {2}  
...  
p->x = 2.0;       {3}  
p->y = 3.0;       {3}  
...  
delete p;          {4}  
...  
p = NULL;         {5}
```



Operacje na zmiennych wskaźnikowych

deklarowanie: typ *p;

alokacja zmiennej: p = new typ;

zwalnianie pamięci: delete p;

przypisanie: =

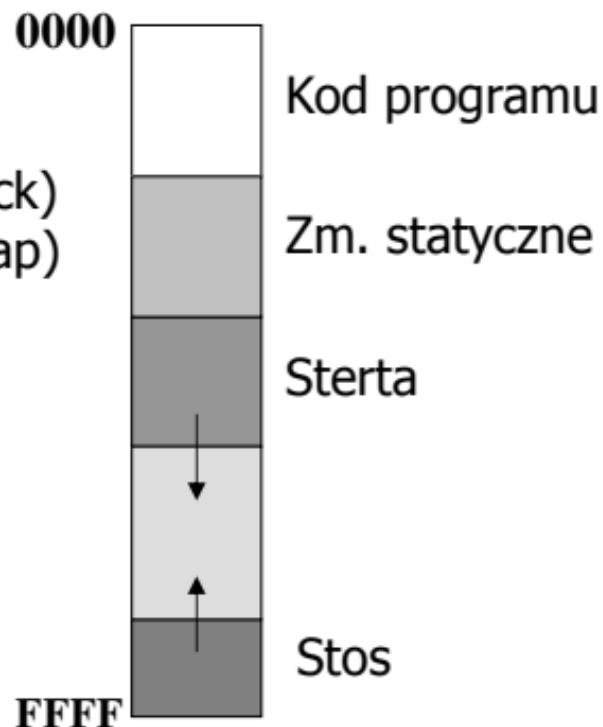
operacje logiczne: == !=

wartość „pusta”: NULL

wypisanie wartości: cout << p;

Przydział pamięci

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



Zastosowania typu wskaźnikowego

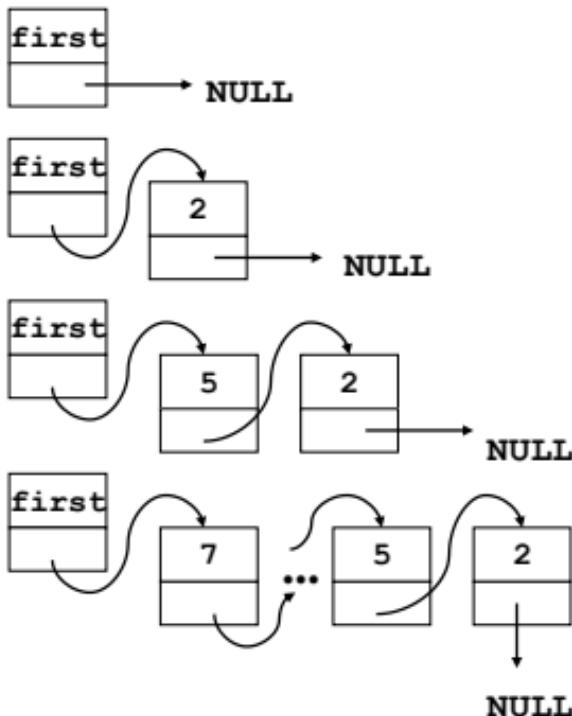


- Zmienne dużych rozmiarów
- Nieregularne struktury danych:
 - stos, kolejka, talia, lista
 - struktura drzewiasta
 - struktura grafowa

Tworzenie łańcucha odsyłaczowego (listy)

```
struct node {  
    int w;  
    node *next;  
};  
  
node *first;  
node *p;  
int s;  
  
first=NULL;  
for (int i=1; i<n; i++) {  
    cin >> s;  
    p = new node;  
    p->next=first;  
    p->w=s;  
    first=p;  
}
```

Etapy tworzenia listy:



Zastosowanie łańcucha (listy)

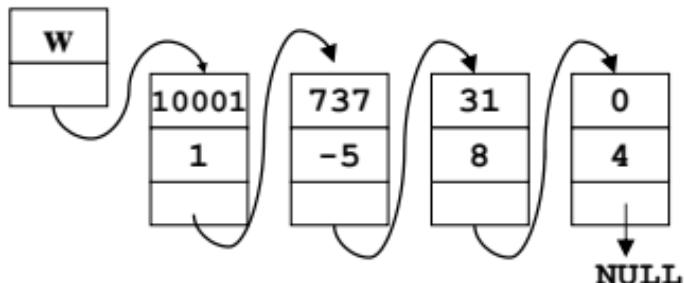
$$W(x) = x^4 + 5x^3 - 7x^2 + x + 3$$

```
double wiel[max];
```

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

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

```
struct node {  
    int wsp;  
    int wyk;  
    node *next;  
};
```



Wypisanie wartości wielomianu

Iteracyjnie

```
void wypisz1(node *p)
{
    while (p!=NULL)
    {
        if (p->wsp>0) cout << "+";
        cout << p->wsp << "x^" << p->wyk;
        p=p->next;
    }
}
```

Wypisanie wartości wielomianu

Rekurencyjnie

```
void wypisz2(node *p)
{
    if (p!=NULL)
    {
        if (p->wsp>0) cout << "+";
        cout << p->wsp << "x^" << p->wyk;
        wypisz2(p->next);
    }
}
```


Wykład 9-10



- 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

```
int t[MAX];  
  
lewy = 0;  
prawy = MAX-1;  
found = false;  
while (lewy<=prawy && !found) {  
    sr = (lewy+prawy)/2;  
    if (t[sr]==sz)  
        found = true;  
    else if (t[sr]<sz)  
        lewy = sr+1;  
    else  
        prawy = sr-1;  
}  
if (found) cout<<sr; else cout<<"not found";
```

Wyszukiwanie połówkowe



```
int t[MAX];  
  
lewy = 0;  
prawy = MAX-1;  
while (lewy<=prawy) {  
    sr=(lewy+prawy)/2;  
    if (t[sr]<sz)  
        lewy = sr+1;  
    else  
        prawy = sr-1;  
}  
if (t[lewy]==sz) cout<<lewy; else cout<<"not found";
```

Sortowanie

- **Sortowaniem** nazywamy *proces ustawiania zbioru obiektów w określonym porządku.*
- Szerokie zastosowanie algorytmów sortowania:
 - możliwość pokazania wielości algorytmów rozwiązujących ten sam problem,
 - konieczność dokonywania analizy algorytmów,
 - pokazują, że warto szukać nowych algorytmów,
 - zależność wyboru algorytmów od struktury przetwarzania danych (metody sortowania wewnętrzne i zewnętrzne).

Sortowanie cd.

- Sortowanie polega na przedstawianiu obiektów, aż do chwili osiągnięcia ich uporządkowania takiego, że dla danej funkcji porządkującej f :

$$f(a_1) \leq f(a_2) \leq \dots \leq f(a_n)$$

(wartość tej funkcji nazywa się kluczem)

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

Klasyfikacja metod

- Według rodzaju struktury:
 - sortowanie tablicy,
 - sortowanie listy, łańcucha,
 - sortowanie pliku.
- Według miejsca sortowania:
 - wewnętrzne,
 - zewnętrzne.
- Według podziału algorytmu na etapy:
 - bezpośrednie (jeden etap - obiekty ulegają przestawieniom),
 - pośrednie - dwa etapy:
 - etap logiczny - informacja jak przedstawiać obiekty, można wykonać kilka etapów logicznych,
 - etap fizyczny - nie zawsze konieczny.

Klasyfikacja metod cd.

- Według wykorzystania pamięci:
 - metody intensywne (in situ),
 - metody ekstensywne (dodatkowa pamięć - metody szybsze).
- Według efektywności (podział umowny):
 - metody proste $O(n^2)$,
 - metody szybkie $O(n \log n)$,
 - metody liniowe $O(n)$.
- Według stabilności (rzadko istotna):
 - stabilne,
 - niestabilne.

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:
 - liczba porównań nie zależy od ustawienia początkowego elem.
 - zmienia się tylko liczba porównań, a nie PRZESUNIĘĆ

Sortowania proste I cd.

```
int a[MAX]; // sortowane elementy 1..n

void proste_wstawianie(int a[], int n) {
    int i,j;
    int x;

    for (i=2; i<=n; i++) {
        x = a[i]; a[0] = x; {metoda wartownika}
        j = i-1;
        while (x < a[j]) {
            a[j+1] = a[j];
            j = j-1;
        }
        a[j+1] = x;
    }
}
```

Sortowania proste I cd.

```
int t[MAX]; // sortowane elementy 1..n

void proste_wstawianie2(int a[], int n) {
    int i,j,lewy,prawy,m;
    int x;

    for (i=2; i<=n; i++) {
        x=a[i];
        lewy=1; prawy=i-1;           {wstawianie połówkowe}
        while (lewy <= prawy) {
            m = (lewy+prawy)/2;
            if (x<a[m]) prawy=m-1;
            else lewy=m+1;
        }
        for (j=i-1; j>=lewy; j--) a[j+1]=a[j];
        a[lewy]=x;
    }
}
```

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, zalecane dla $n \leq 50$.

Sortowania proste II cd.

```
int a[MAX]; // sortowane elementy 0..n

void proste_wybieranie(int a[], int n) {
    int i,j,k;

    for (i=0; i<n; i++) {
        k=i; {wybieranie minimum}
        for (j=i+1; j<n; j++)
            if (a[j]<a[i]) k=j;
        swap(a[k],a[i]);
    }
}
```

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:
 - 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 cd.

```
int a[MAX]; // sortowane elementy 0..n

void proste_babelkowe(int a[], int n) {
    int i,j;
    int x;

    for (i=1; i<=n; i++)
        for (j=n; j>=i; j--)
            if (a[j-1]>a[j])
                swap(a[j-1],a[j]);
}
```

Sortowania szybkie



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), ale algorytm prosty (bez rekurencji).

Sortowania szybkie cd.



Warianty sortowania Comsort:

- 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.
- Combsort 11 - rozpiętość 9 i 10 zastępujemy 11.

Sortowania szybkie cd.

```
int a[MAX]; // sortowane elementy 0..n

void Combsort(int a[], int n) {
    int top,gap,i,j;
    int x;
    bool swapped=true;

    gap=n;
    while (gap>1 || swapped) {
        gap=max(int(gap/1.3),1);
        top=n-gap;
        swapped=false;
        for (i=0; i<=top; i++) {
            j=i+gap;
            if (a[i]>a[j]) {
                swap(a[i],a[j]);
                swapped=true;
            }
        }
    }
}
```

Sortowania szybkie II



Sortowanie przez podział – 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,

Sortowania szybkie II cd.

```
int a[MAX];

void qs(int l, int p) {
    int i,j,x,tmp;

    i=l; j=p;
    x=a[(l+p)/2];
    while (i<=j) {
        while (a[i] < x) do i++;
        while (a[j] > x) do j--;
        if (i<=j) {
            tmp=a[j]; a[j]=a[i]; a[i]=tmp;
            i++;
            j--;
        }
    }

    if (l < j) qs(l, j);
    if (i < p) qs(i, p);
}
```


Wykład 12



- Reprezentacja liczb w maszynie cyfrowej
- Liczby stałoprzecinkowe
- Liczby zmiennoprzecinkowe
- Dokładność obliczeń

Pozycyjny system liczenia

$$1999 = 1*10^3 + 9*10^2 + 9*10^1 + 9*10^0$$

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

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

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

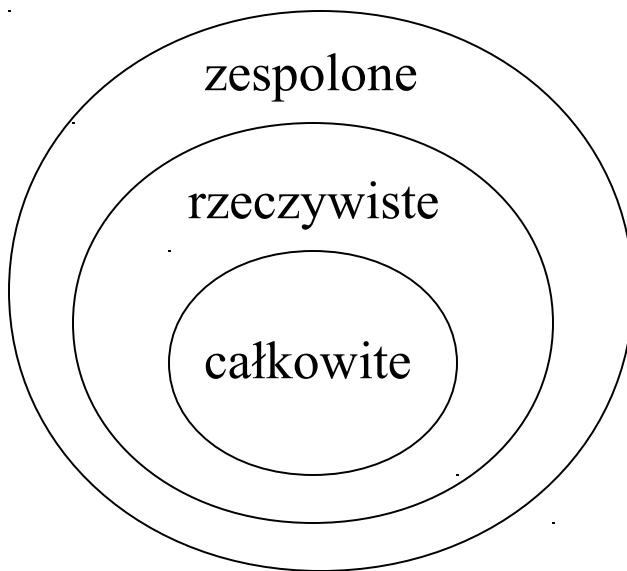
$$0.11_{(2)} = 0*2^0 + 1*2^{-1} + 1*2^{-2} = 0.75$$

Arytmetyka liczb w maszynie

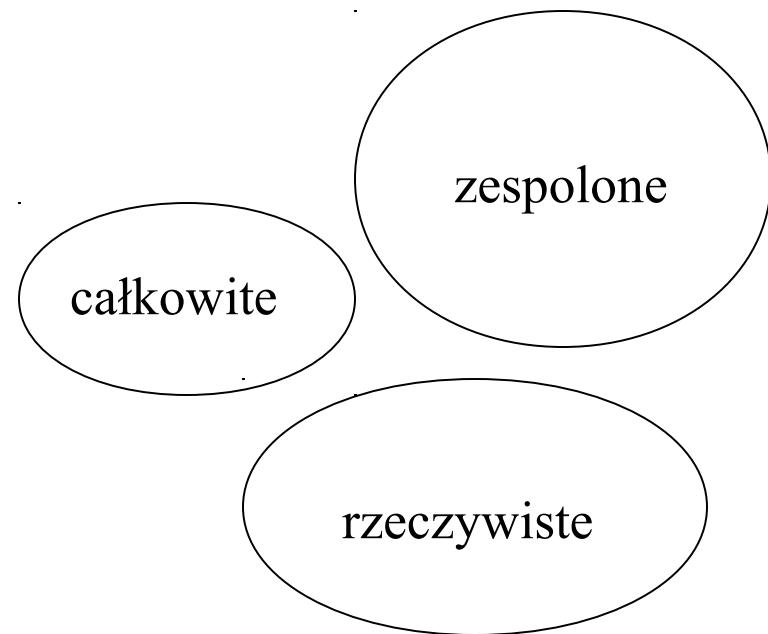
- różna od arytmetyki używanej przez ludzi
 - system dwójkowy
 - skończona i ustalona precyzja
- własności liczb o skończonej precyzyji (zakresie) są inne
- zbiór liczb o skończonej precyzyji (zakresie) nie jest zamknięty na żadne działanie
- nie działa prawo łączności
$$a + (b + c) \Leftrightarrow (a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

Rodzaje liczb

- liczby całkowite
- liczby rzeczywiste
- liczby zespolone



w matematyce



w maszynie cyfrowej

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	$\begin{array}{r} 1\ 1111001 \\ +1 \\ \hline 1\ 1111010 \end{array}$

Działania na liczbach kodu U2

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

$$\begin{array}{r} 6 \mid 0\ 0000110 \\ +7 \mid 0\ 0000111 \\ \hline 13 \mid \mathbf{0\ 0001101} \end{array}$$

$$\begin{array}{r} 6 \mid 0\ 0000110 \\ -7 \mid 1\ 1111001 \\ \hline -1 \mid \mathbf{1\ 1111111} \end{array}$$

$$\begin{array}{r} 64 \mid 0\ 1000000 \\ -64 \mid 1\ 1000000 \\ \hline 0 \mid \mathbf{0\ 0000000} \end{array}$$

$$\begin{array}{r} -65 \mid 1\ 0111111 \\ -65 \mid 1\ 0111111 \\ \hline -130 \mid \mathbf{0\ 1111110} \end{array}$$

↑ 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 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)

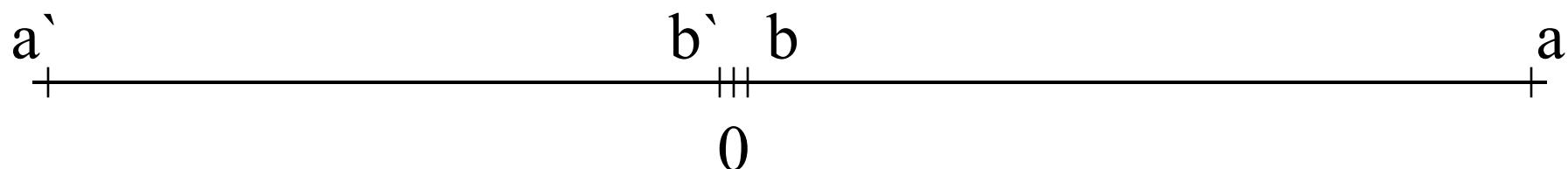
cecha - 2 cyfry + znak (-99 - 99)

dodatnia wartość maksymalna (a): $0.999 * 10^{99}$

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 zmiennoprzecinkowe

Przykład 1. System binarny (typ 4 bajtowy Single)

mantysa - 23 bity + 1bit na znak

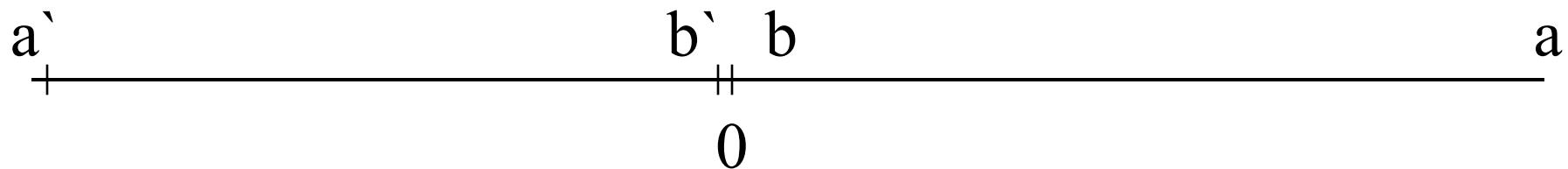
cecha - 8 bitów

dodatnia wartość maksymalna (a): $0\ 111..1 * 2^{0111111}$

ujemna wartość minimalna (a'): $1\ 111...1 * 2^{0111111}$

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
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*)

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)
 $m \geq 64, c \geq 15$

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	0.0397000000000000	0.0397000000000000
2.	0.154071718454361	0.1540717300000000	0.1540717300000000
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	$p+r^*p^*(p-1)$	$(1+r)^*p-r^*p^*p$
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

$$w = \frac{(\sqrt{1+u} - \sqrt{1-u})}{u}$$

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

typ real

0.100000000	1.00125550119628
0.010000000	1.00001250054629
0.001000000	1.00000012500095
0.000100000	1.00000000124965
0.000010000	1.00000000001273
0.000001000	1.00000000000000
0.000000100	1.00000000000000
0.000000010	0.99999999999909
0.000000001	0.99999999996089

1.00125550119628
1.00001250054629
1.00000012500095
1.0000000124965
1.00000000001273
1.00000000000000
1.00000000000000
1.00000000000000
1.00000000000000

typ double

0.100000000	1.00125550119637
0.010000000	1.00001250054690
0.001000000	1.00000012500005
0.000100000	1.00000000125000
0.000010000	1.00000000001250
0.000001000	1.00000000000018
0.000000100	1.00000000000000
0.000000010	0.999999999999864
0.000000001	0.999999999996103

1.00125550119638
1.00001250054691
1.00000012500005
1.00000001250000
1.00000000001250
1.00000000000013
1.00000000000000
1.00000000000000
1.00000000000000

$$w = \frac{(\sqrt{1+u} - \sqrt{1-u})}{u}$$

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

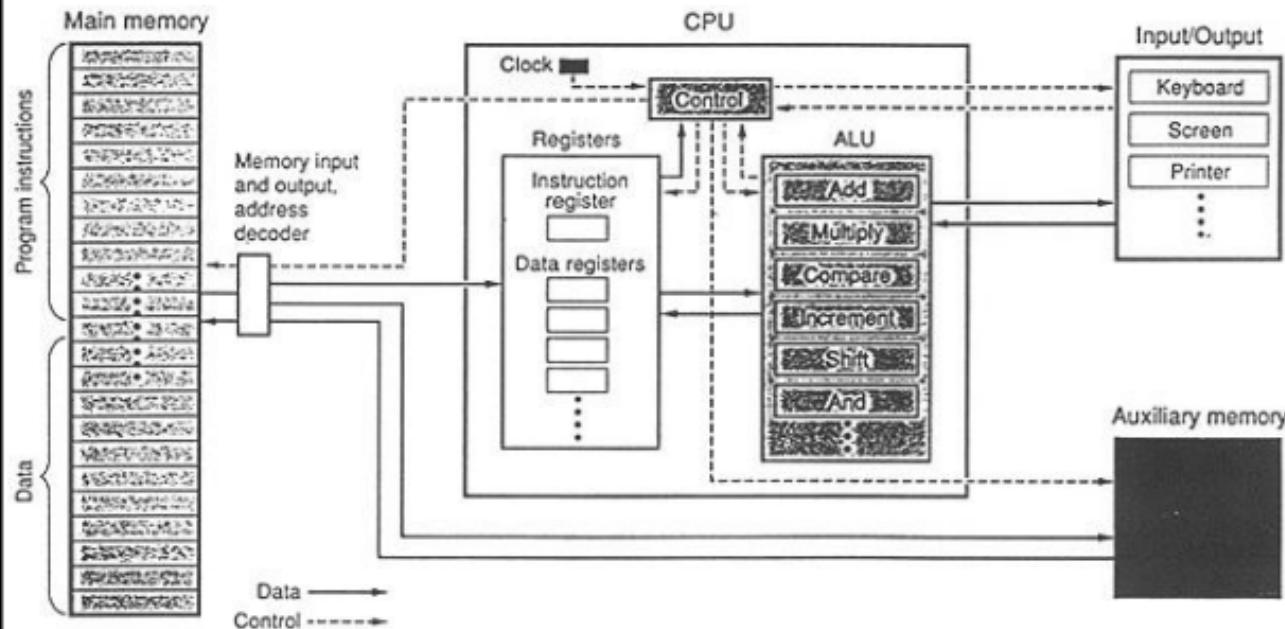
typ extended

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

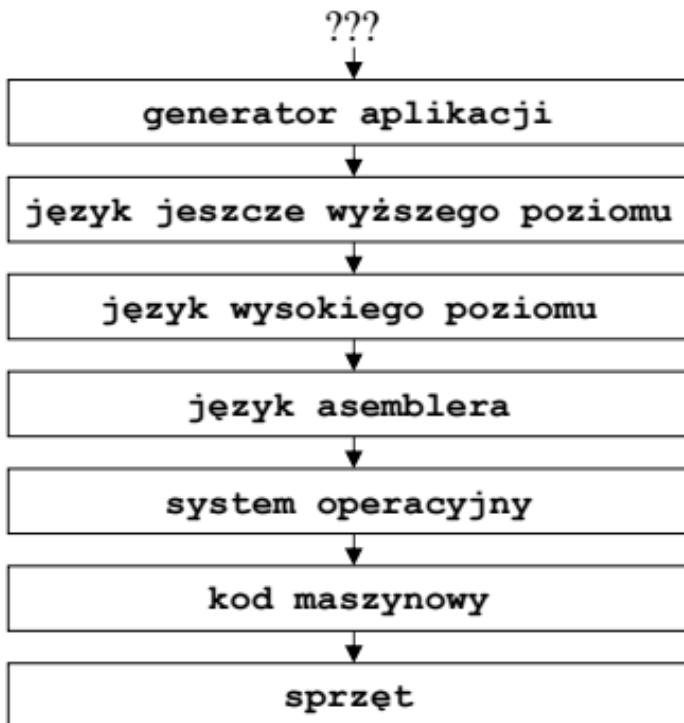
wartości dokładne

0.100000000	1.00125550119637747391785450350122038359405315883
0.010000000	1.00001250054690722874466702747516135163757833258
0.001000000	1.00000012500005468753222658432008437730121685070
0.000100000	1.00000000125000000546875003222656271820068519554
0.000010000	1.00000000001250000000054687500003222656250218200
0.000001000	1.000000000000125000000000546875000003222656250
0.000000100	1.000000000000001250000000005468750000003222
0.000000010	1.00000000000000012500000000005468750000000000
0.000000001	1.0000000000000000125000000000005468750000000000

Architektura prostego komputera



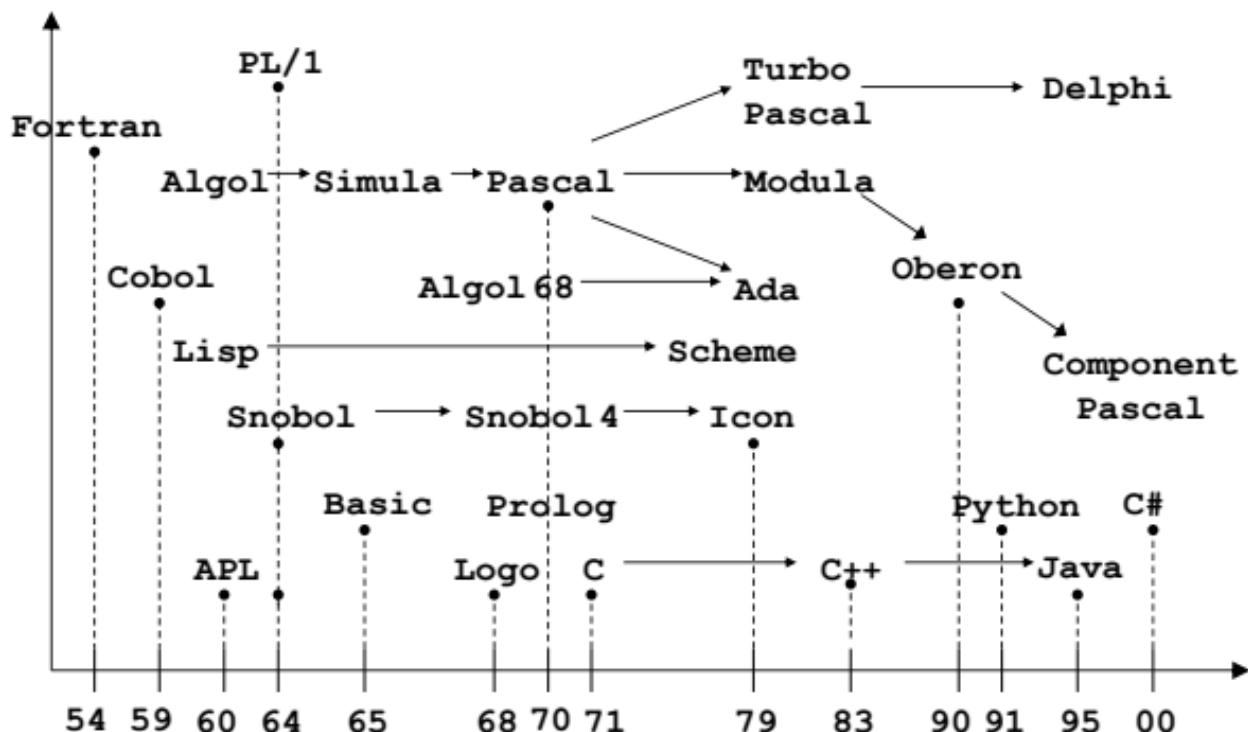
Maszyna wielopoziomowa



Poziom języka programowania

Język Pascal	Język assemblera	Kod maszynowy
while i<>j do	LOOP	A000 10 B0 00
if i>j then i := i - j	SUB J	A003 12 B0 02
else j := j - i;	JZERO EXIT	A006 24 10 1E
	JNEG SUBI	A009 25 A0 12
	STORE I	A00C 11 B0 00
	JUMP LOOP	A00F 23 A0 00
	SUBI LOAD J	A012 10 B0 02
	SUB I	A015 12 B0 00
	STORE J	1018 11 B0 02
	JUMP LOOP	101B 23 A0 00
	EXIT	101E
		:
	LOAD 10	:
	STORE 11	B000 }
	SUB 12	B003 zmienne
	:	B006
	JUMP 23	:
	JZERO 24	:
	JNEG 25	:

Historia języków programowania



Popularność języków programowania

Position Nov 2013	Position Nov 2012	Delta in Position	Programming Language	Ratings Nov 2013	Delta Nov 2012	Status
1	1	=	C	18.155%	-1.07%	A
2	2	=	Java	16.521%	-0.83%	A
3	3	=	Objective-C	9.406%	-0.90%	A
4	4	=	C++	8.369%	-1.33%	A
5	6	↑	C#	6.024%	+0.43%	A
6	5	↓	PHP	5.379%	-0.35%	A
7	7	=	(Visual) Basic	4.396%	-0.64%	A
8	8	=	Python	3.110%	-0.95%	A
9	23	↑↑↑↑↑↑↑↑↑↑	Transact-SQL	2.521%	+2.05%	A
10	11	↑	JavaScript	2.050%	+0.77%	A
11	15	↑↑↑	Visual Basic .NET	1.969%	+1.20%	A
12	9	↓↓↓	Perl	1.521%	-0.66%	A
13	10	↓↓↓	Ruby	1.303%	-0.44%	A
14	14	=	Pascal	0.715%	-0.17%	A
15	13	↓↓	Lisp	0.708%	-0.25%	A
16	19	↑↑↑	MATLAB	0.656%	+0.04%	B
17	12	↓↓↓↓	Delphi/Object Pascal	0.649%	-0.35%	A-
18	17	↓	PL/SQL	0.605%	-0.03%	A-
19	24	↑↑↑↑	COBOL	0.585%	+0.11%	B
20	20	=	Assembly	0.532%	-0.05%	B

www.tiobe.com/tpci.htm

Evolution of imperative programming

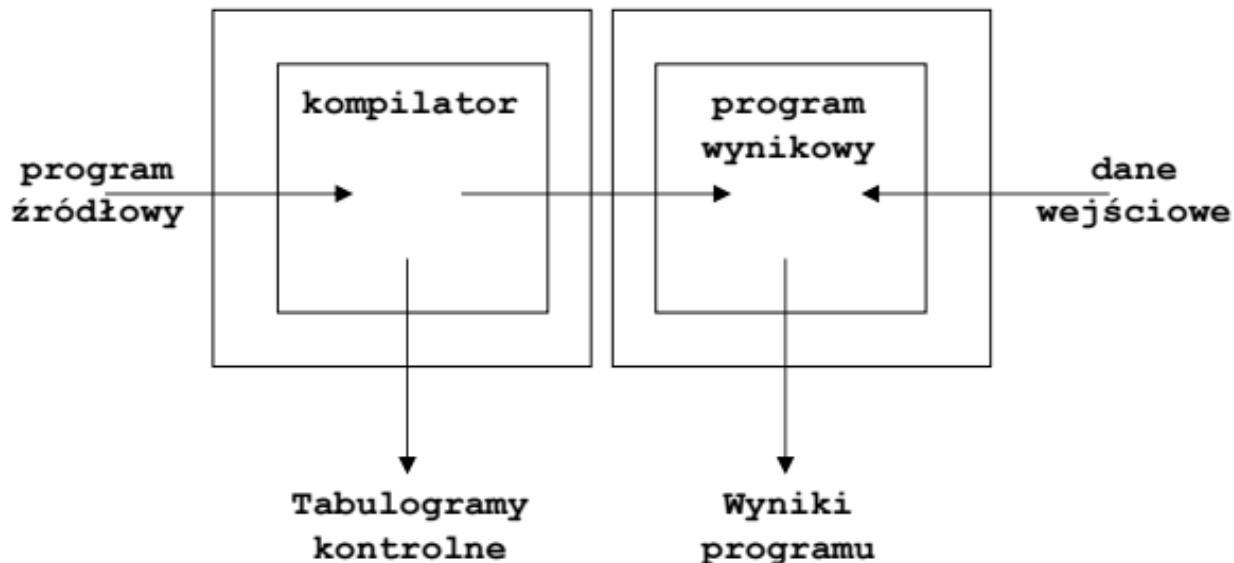
Decade	Programming technology	Key advance
1940s	machine codes	programmable machines
1950s	assembly languages	symbols
1960s	high-level languages	expressions and machine-independence
1970s	structured programming	structured types and control structures
1980s	modular programming	separation of interface from implementation
1990s	object-oriented programming	polymorphism
2000s	component-oriented Programming	dynamic and safe composition
2010s	Service Oriented Architecture	loosely coupled units

Programy wspomagające programowanie

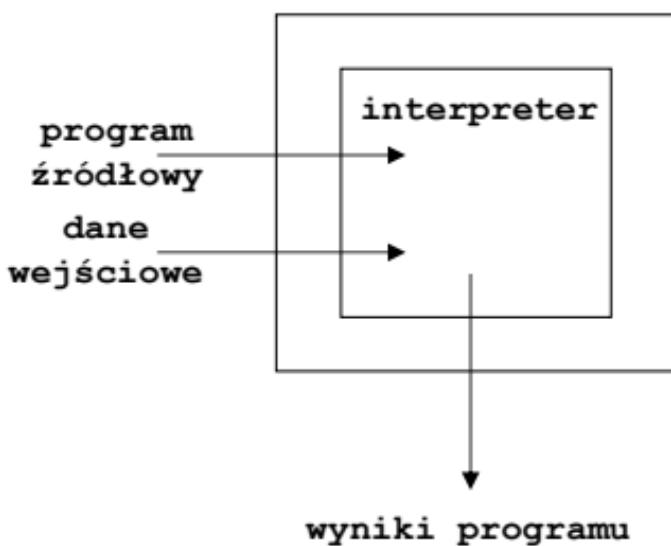


- asembler
- kompilator
- translator
- kompilator przechodni (cross compiler)
- linker
- debugger
- profiler
- visual interface builder

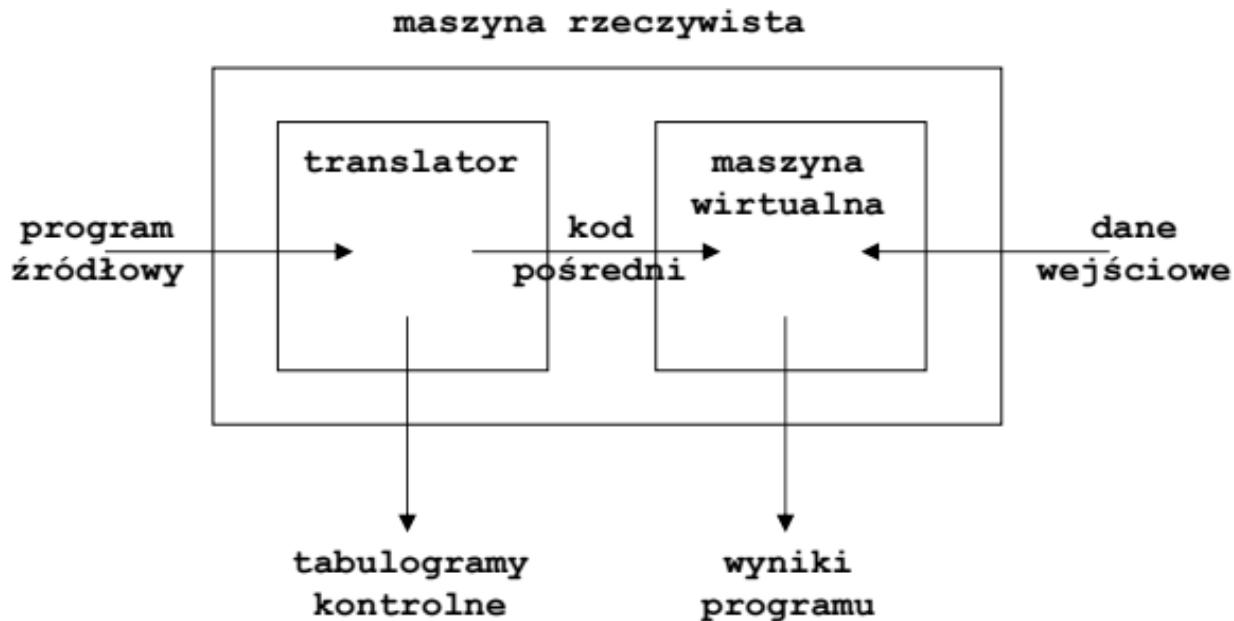
Kompilacja i wykonanie programu



Interpretacja programu



Translacja do kodu pośredniego



Efektywność języków programowania

Na podstawie : <http://shootout.alioth.debian.org/>

Programy testowe (benchmark)

binary-trees	nsieve
chameneos	nsieve-bits
cheap-concurrency	partial-sums
fannkuch	pidigits
fasta	recursive
k-nucleotide	regex-dna
mandelbrot	reverse-complement
meteor-contest	spectral-norm
n-body	startup
	sum-file

Efektywność języków programowania

1.0 C gcc	1.22	9.4 Smalltalk VisualWorks	11.41
1.0 C++ g++	1.24	10 Erlang HiPE	12.19
1.2 D Digital Mars	1.41	11 Lua	13.99
1.2 Pascal Free Pascal	1.42	14 Scheme MzScheme	16.71
1.2 Clean	1.47	15 Pike	18.53
1.4 Oberon-2 OO2C	1.74	17 Python	21.29
1.5 Java 6 -server	1.87	18 Mozart/Oz	22.33
1.6 OCaml	1.92	21 Perl	25.03
1.6 Ada 95 GNAT	1.97	23 PHP	27.65
1.6 Eiffel SmartEiffel	1.98	25 Icon	31.00
1.7 SML MLton	2.05	40 Tcl	48.29
1.7 Lisp SBCL	2.07	44 JavaScript SpiderMonkey	53.93
1.8 BASIC FreeBASIC	2.22	52 Ruby	63.41
1.9 CAL	2.32	64 Prolog SWI	77.64
1.9 Scala	2.36		
1.9 Haskell GHC	2.38		
2.2 Nice	2.66		
2.4 C# Mono	2.94		
3.1 Forth bigForth	3.74		
3.1 Fortran G95	3.81		

Języki programowania ogólnego zastosowania

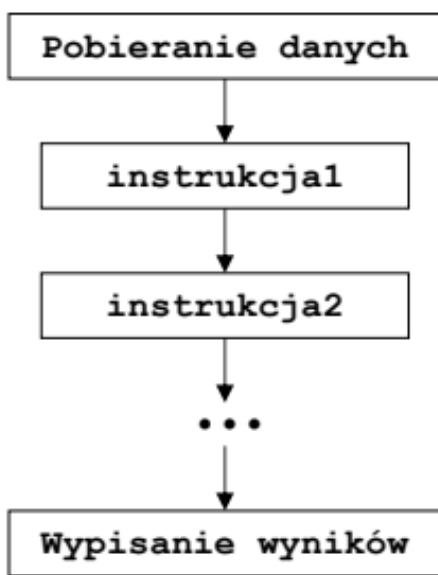
- języki imperatywne (instrukcyjne)
- języki aplikatywne (funkcyjne)
- języki deklaratywne (logiczne)

Problem:

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

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

Język imperatywny (instrukcyjny)

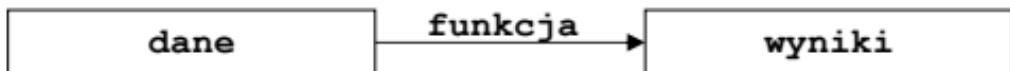


```

Program TrojkiPitagorejskie;
const
  max=10;
var
  a,b,c:integer;
begin
  for c:=1 to max 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.

```

Język aplikatywny (funkcyjny)



```
TrojkiPitagorejskie(M) = [] , M=1
```

```
TrojkiPitagorejskie(M) = T(M,M-1) ++ TrojkiPitagorejskie(M-1)
```

```
T(c,b) = [] , b=1
```

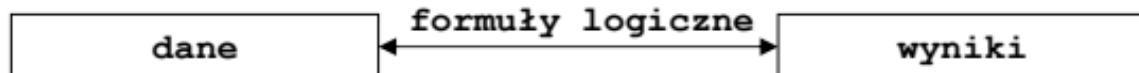
```
T(c,b) = [(a,b,c)] ++ T(c,b-1) , a=trunc(a) where a=sqrt(c*c-b*b)
```

```
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)



TrojkiPitagoreskie (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=3, B=4, C=5

A=4, B=3, C=5

A=6, B=8, C=10

A=8, B=6, C=10

Kategorie języków - popularność

Category	Ratings Dec 2012	Delta Dec 2011
Object-Oriented Languages	58.5%	+2.1%
Procedural Languages	36.9%	-0.2%
Functional Languages	3.2%	-1.3%
Logical Languages	1.4%	-0.6%

Category	Ratings Dec 2012	Delta Dec 2011
Statically Typed Languages	71.4%	+0.4%
Dynamically Typed Languages	28.6%	-0.4%

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, dostarczając użytkownikom zestawu środków ułatwiających projektowanie, kodowanie, uruchamianie i eksploatację programów oraz w tym samym czasie sterując 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:

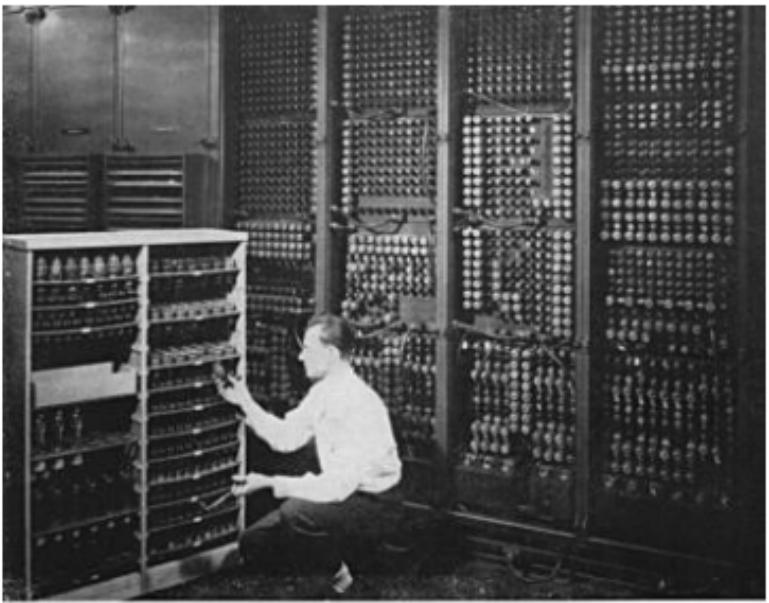
- funkcje systemowe dostarczane programowi użytkownika
- określone obszary pamięci - bufory
- pamięć zewnętrzna
- katalogi i pliki
- translatory, kompilatory

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)

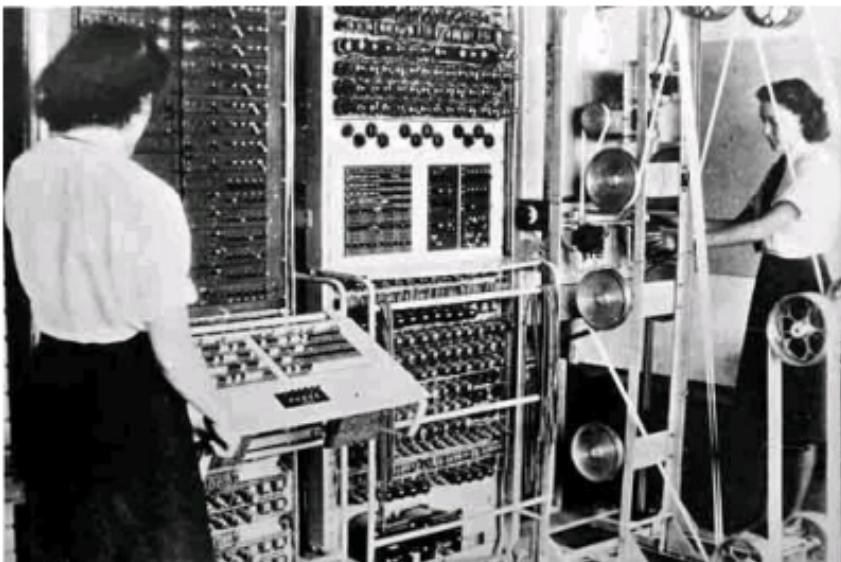
Pierwsze komputery



Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

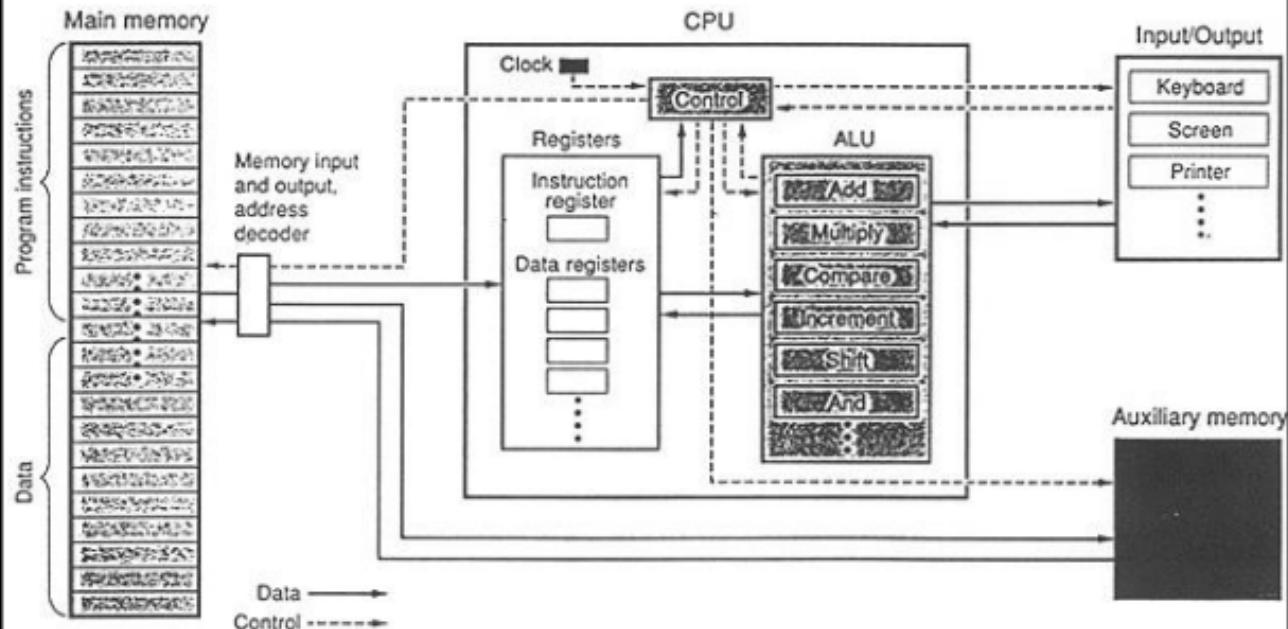
ENIAC komputer skonstruowany w latach 1943-45

Pierwsze komputery

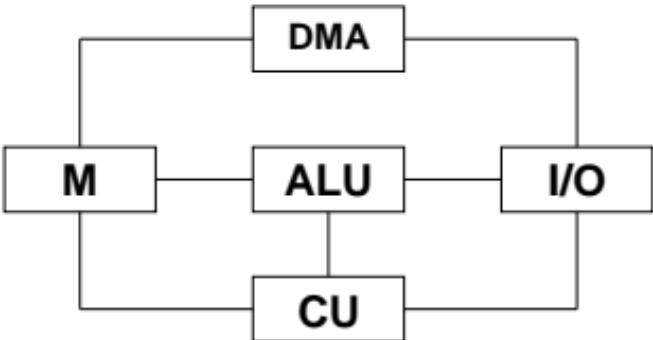
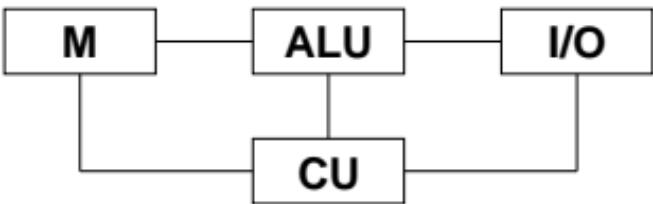


COLOSSUS - został zbudowany w 1941 r. w brytyjskim ośrodku kryptograficznym Bletchley Park.

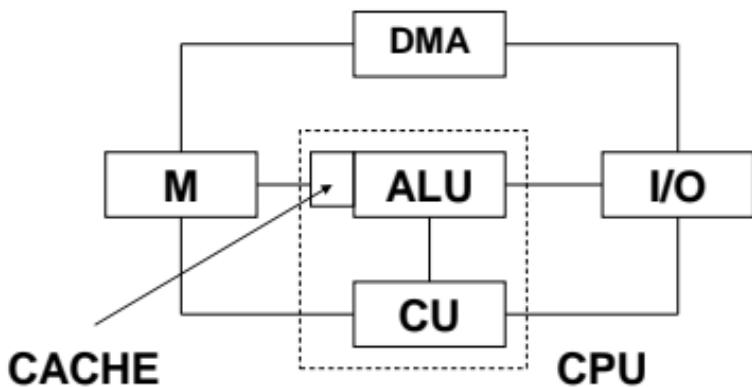
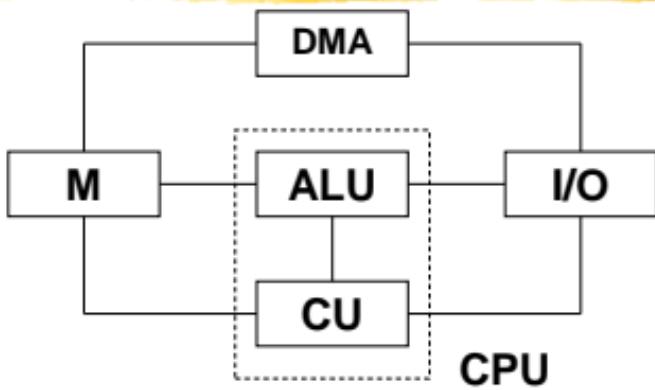
Architecture of a Serial Computer



Architektura von Neumann



Architektura von Neumann



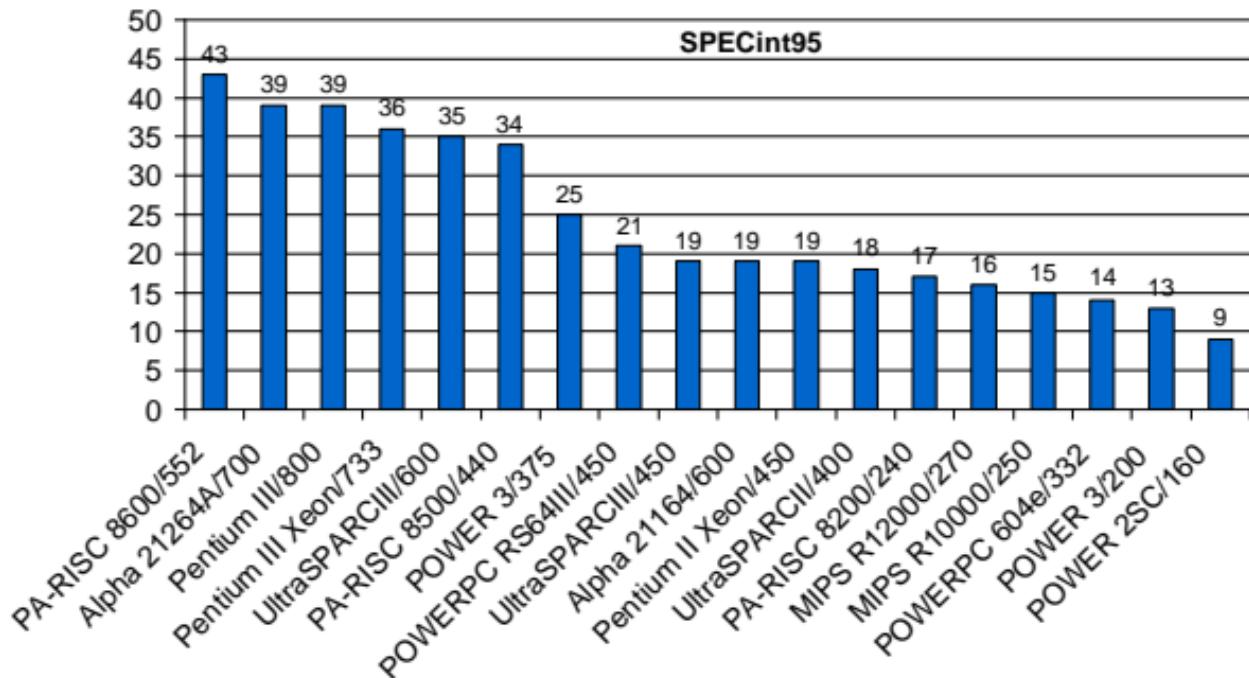
Miara wydajności - MIPS

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

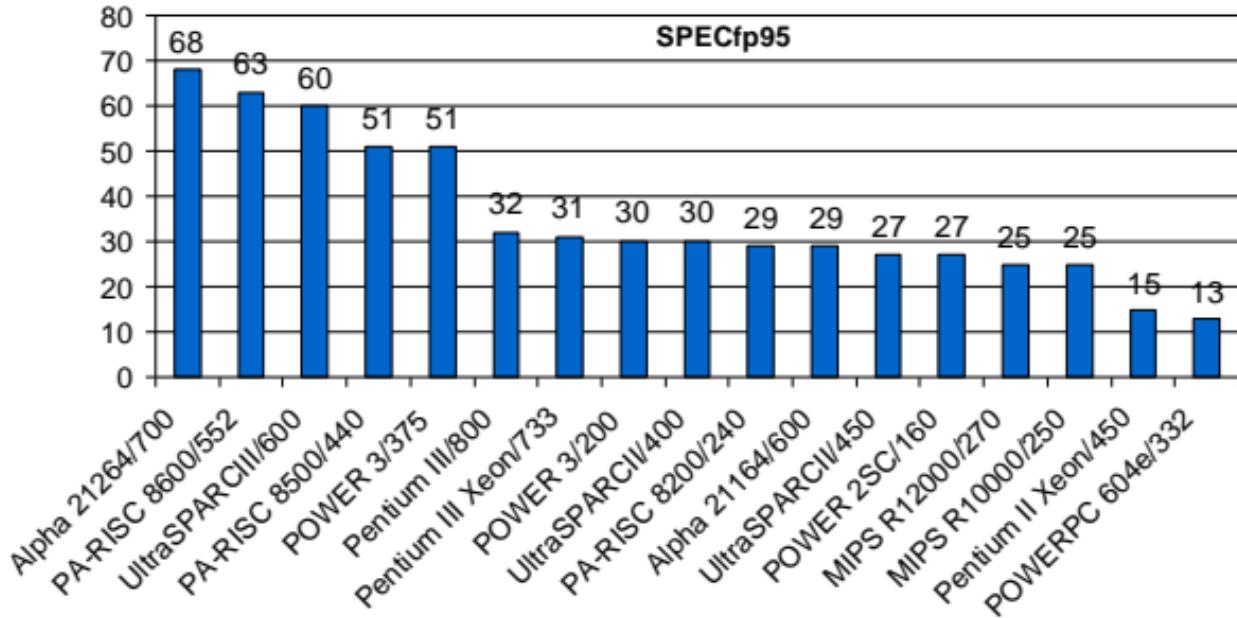
Miara wydajności - MFlops

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

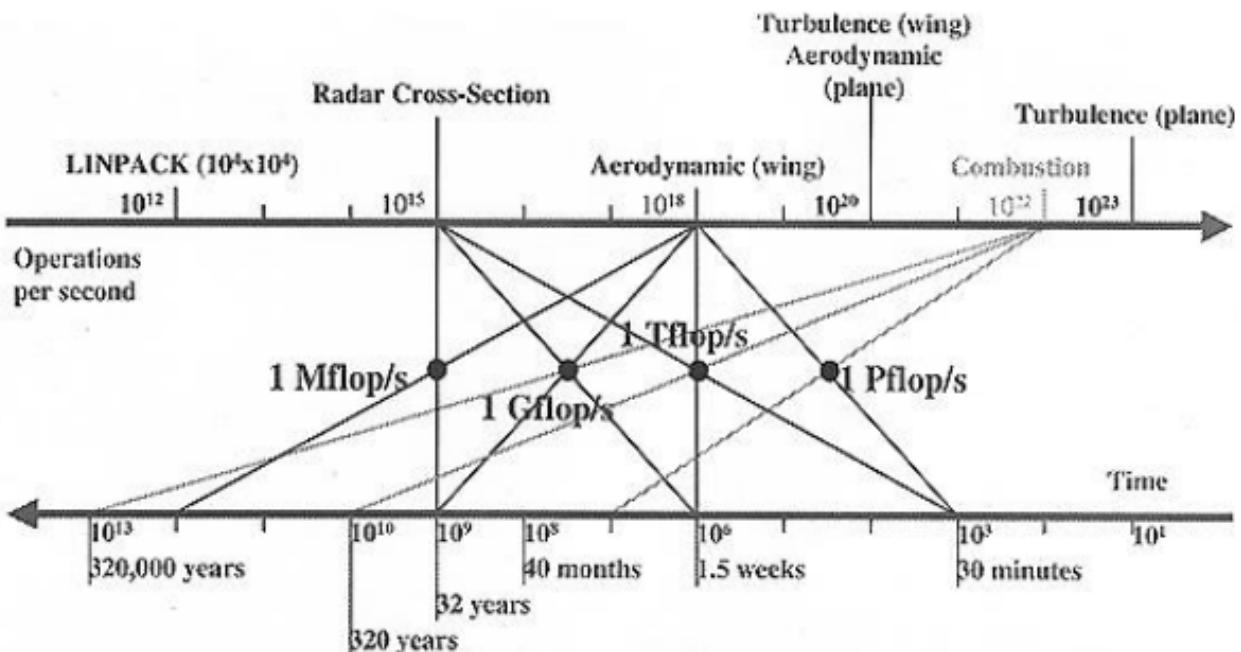
Staloprzecinkowa wydajność procesora



Zmiennoprzecinkowa wydajność procesora



The need for computer power



TOP 10 Sites for June 2013

For more information about the sites and systems in the list, click on the links or view the [complete list](#).

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National University of Defense Technology China	Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,964	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 Villaf 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
5	DOE/SC/Argonne National Laboratory United States	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945
6	Texas Advanced Computing Center/Univ. of Texas United States	Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462,462	5,168.1	8,520.1	4,510
7	Forschungszentrum Juelich (FZJ) Germany	JUQUEEN - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	458,752	5,008.9	5,872.0	2,301
8	DOE/NNSA/LLNL United States	Vulcan - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	393,216	4,293.3	5,033.2	1,972

TITAN computer



U.S. Department of
ENERGY
Office of
Science

OAK
RIDGE
National Laboratory

OLCF
Oak Ridge Leadership Computing Facility

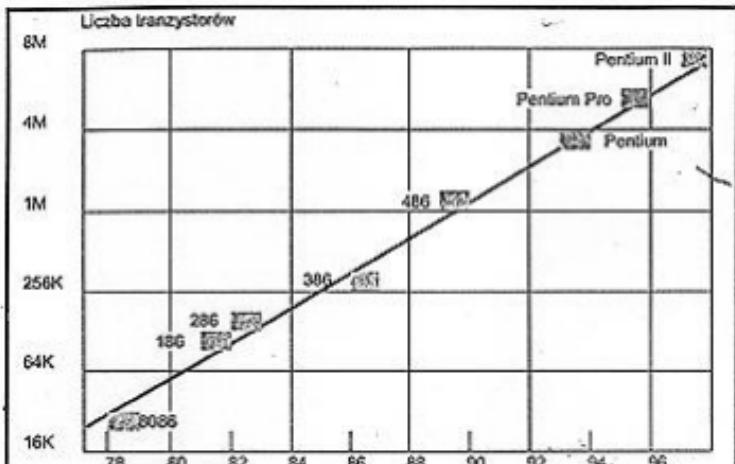
K computer



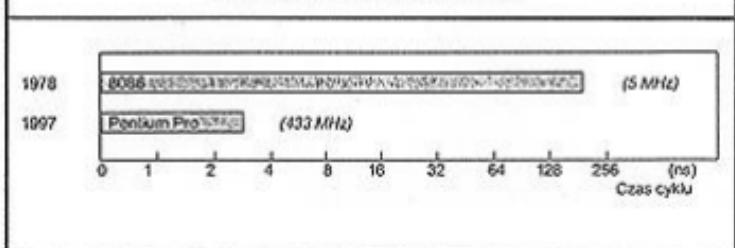
Ważniejsze wydarzenia z historii komputerów

1945	• John von Neumann: "The First Draft of a Report on the EDVAC"	
1946	ENIAC	Pierwsza elektroniczna maszyna cyfrowa (18 tysięcy lamp, 1500 przekaźników)
1951	UNIVAC I	Pierwszy komputer komercyjny (48 sprzedanych egzemplarzy)
1952	1AS	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 tyś. Egzemplarzy)
1971	Intel 4004	Pierwszy mikroprocesor (4-bitowy)
1974	Intel 8080	Pierwszy "CPU on a chip", protoplasta "8-bitowców": Z80, 8085
1974	CRAY-1	"Superkomputer" (pierwszy produkt Cray Research, maksymalna szybkość 150 M flops)
1977	Intel 8048	Pierwszy "computer on a chip", nowsza wersja (stosowana do dziś): 8051
1978	VAX	Pierwszy "superminikomputer" (DEC, 32-bitowy)
1981	IBM PC	Początek ery "pcetów" (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



Rys. 1. Złożoność procesorów Intel x86

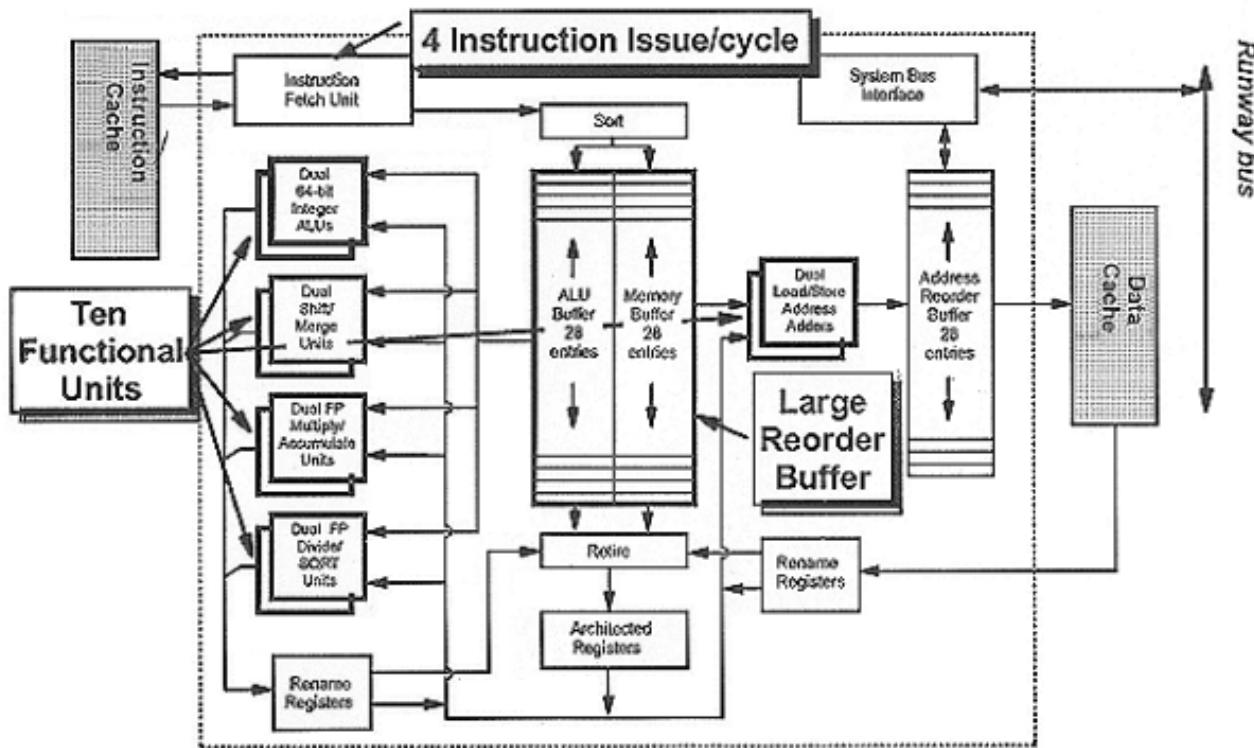


Porównanie mikroprocesorów

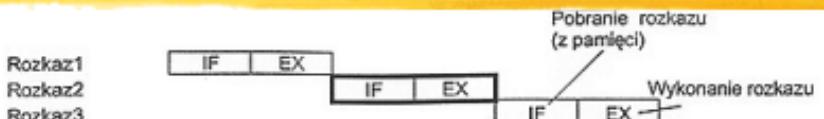
Procesor (producent)	Rozka- zów/cykl	Cache I, D (KB)	Liczba wypro- wadzeń	Zegar (MHz)	SPECint95	SPECfp95	Liczba tranzystorów (mln)
21164 Alpha (DEC)	4	16+96	499	500	15,4	21,1	9,3
PowerPC 604e (IBM/Motorola)	4	32	255	225	9,0	7,5	5,1
Pentium Pro (Intel)	3	16 + (256)	387	200	8,2	6,7	5,5
Pentium II (Intel)	?	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.

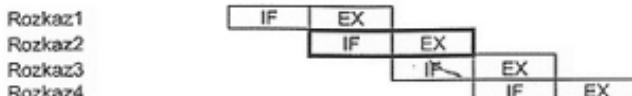
PA-8000 - Block Diagram



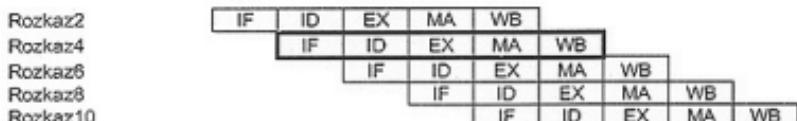
Sposoby przetwarzania strumienia rozkazów



PRZETWARZANIE SZEREGOWE



PRZETWARZANIE POTOKOWE (pipeline)



PRZETWARZANIE WIELOPOTOKOWE (superscalar)

Wykład 15



- Złożoność czasowa
- Notacja $O()$
- Przykłady problemów
- Problemy nierozwiązywalne

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 $f(k) \leq c g(k)$ dla prawie wszystkich wartości k .

Przykłady:

$O(\log N)$

logarytmiczna

$O(N)$

liniowa

$O(N * \log N)$

$O(N^2)$

kwadratowa (wielomianowa)

$O(2^N)$

wykładnicza

Problem stałej:

$$f_1(N) = 10 * N$$

$$f_2(N) = 1000 * \log N$$

Złożoność czasowa a czas wykonania

Rozmiar problemu n \ 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 dnie	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 przyśpieszania 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

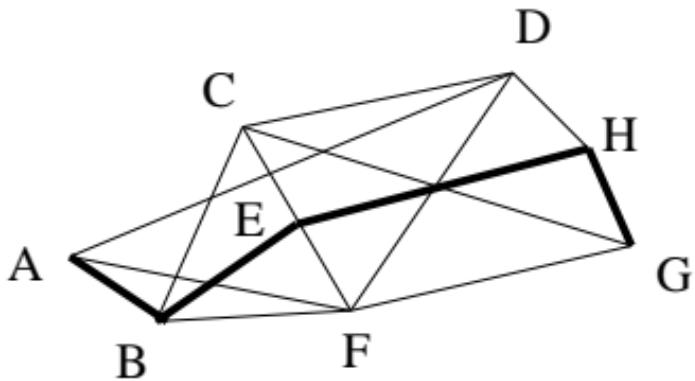
Złożoność algorytmów

Problem sortowania tablicy

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

N	proste	szylkie
10	100	33
10^6	10^{12}	$2 * 10^7$

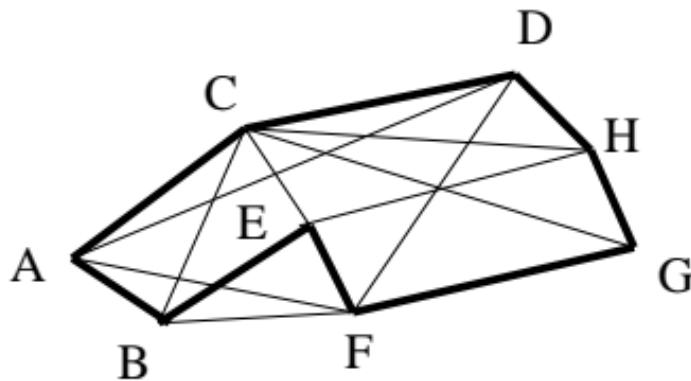
Problem najkrótszej drogi



Rozwiązanie: droga ABEHG

Znane algorytmy $O(N^2)$

Problem komiwojażera



Rozwiązanie: droga ABCDHGFEB

Prosty algorytm $O(N!)$

Znany algorytm $O(2^N)$

Problem tautologii

Czy jest tautologią wyrażenie $\sim(a \& b) \rightarrow (a \mid b)$

a	b	$\sim(a \& b) \rightarrow (a \mid b)$
0	0	0
0	1	1
1	0	1
1	1	1

Dana jest funkcja N zmiennych logicznych:

$$f(A_1, A_2, \dots, A_n) \rightarrow \{\text{true}, \text{false}\}$$

Czy zawsze przyjmuje ona wartość true?

W ogólnym przypadku należy sprawdzić wszystkie możliwości czyli złożoność problemu wynosi $O(2^N)$

Funkcja Ackermana-Hermesa

$$f(0, b) = b + 1$$

$$f(a, 0) = f(a-1, 1) \quad a > 0$$

$$f(a, b) = f(a-1, f(a, b-1)) \quad a > 0, b > 0$$

Ile wynosi $f(5, 5)$?

Funkcja Ackermana-Hermesa

$$f(a,b) = b+1 \quad \text{dla } a=0$$

$$f(a,b) = f(a-1,1) \quad \text{dla } b=0$$

$$f(a,b) = f(a-1, f(a,b-1)) \quad \text{dla } a>0 \text{ i } b>0$$

Wartości funkcji:

$$f(0,b) = b+1$$

$$f(1,b) = b+2$$

$$f(2,b) = 2 \cdot b + 3$$

$$f(3,b) = 2^{b+3} - 3$$

$$f(4,b) = 2^{2^{2^{\dots^2}} - 3}$$

$$2^{16}$$

$$2^{2^{2^{\dots^2}}}$$

$$2^{2^{2^{\dots^2}}}$$

$$2^{2^{2^{\dots^2}}}$$

$$2^{2^{2^{\dots^2}}}$$

$$2^{2^{2^{\dots^2}}}$$

$$2^{2^{2^{\dots^2}}}$$

$$2^{2^{2^{\dots^2}}}$$

$$F(5,5) = 2^{2^{2^{\dots^2}} - 3}$$

Problem stopu

```
read(x)
while x>0 do
    x:=x-1
```

```
read(x)
while x<>0 do
begin
    ...
    if x=0 then x:=1
end.
```

```
read(x)
while x<>1 do
begin
    if odd(x) then x:=3*x+1
                else x:=x div 2
end
```

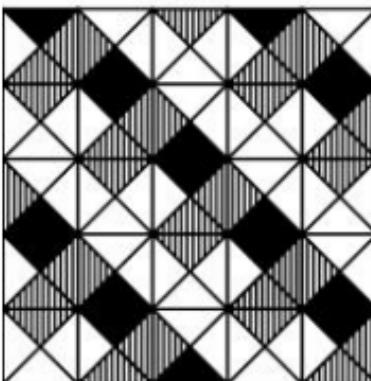
Problem stopu

```
Function Q(R:procedure; X:data) :boolean;  
  
Procedure S(W:procedure);  
begin  
    b:=Q(W,W);  
    if b then repeat until false  
        else halt;  
end;
```

Pytanie:

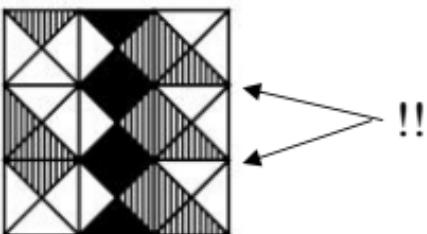
Czy wywołanie procedury S(S) się zakończy?

Problem domina



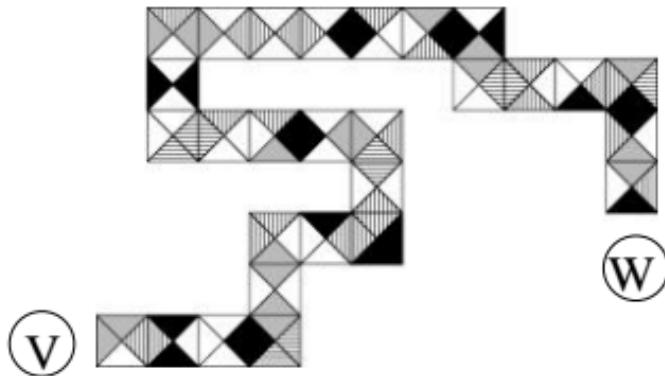
Rodzaje kafelków, którymi można pokryć każdy obszar

Problem domina



Rodzaje kafelków, którymi nie można pokryć nawet małych obszarów

Problem domino - wąż



Waż domino łączący punkty V i W

