Wykład 1

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

Wstęp do informatyki

Liczba godzin:
Semestr 1, wyk. 28 godz., ćw. 28 godz.

Wykład:
Dr inż. Marek Gajęcki
poniedziałek 9:35

Ćwiczenia:
Dr inż. Marek Gajęcki
Dr inż. Tomasz Jurczyk
wtorek 14:40, 16:15
środa 14:40, 16:15
Dr hab. inż. Renata Słota
wtorek 16:15

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

Slajdy i inne materiały:
http://galaxy.agh.edu.pl/~mag/ad-wdi/

Wstęp do informatyki

Wyznaczanie oceny końcowej:

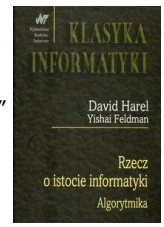
1. Aby uzyskać pozytywną ocenę końcową niezbędne jest uzyskanie pozytywnej oceny z ćwiczeń oraz egzaminu.
2. Obliczamy średnią arytmetyczną z ocen zaliczenia ćwiczeń i egzaminów uzyskanych we wszystkich terminach.
3. Wyznaczamy ocenę końcową na podstawie zależności:
   if sr>4.75 then ok:=5.0 else
 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

Literatura

- D. Harel „Rzecz o istocie informatyki - algorytmika”
- J.G. Brookshear „Informatyka w ogólnym zarysie”
- N. Wirth „Algorytmy + struktury danych = programy”
- J. Bentley „Perełki oprogramowania”
- J. Bentley „Więcej perełek oprogramowania”
- D.E. Knuth „Sztuka programowania”
- T.H. Cormen „Wprowadzenie do algorytmów”



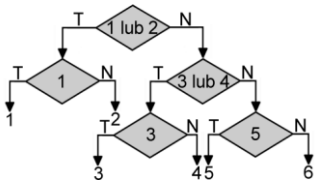
Informatyka (Computer Science)

Informatyka to nauka o przetwarzaniu informacji.

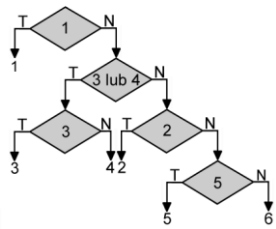
Aktualnie obejmuje wiele zagadnień, między innymi:
- algorytmika, struktury danych, języki programowania
- mikroprocesory, architektury komputerów
- bazy danych, systemy operacyjne, sieci komputerowe
- kompilatory, kryptografia, metody numeryczne
- inżynieria oprogramowania, projektowanie systemów
-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

Identyfikacja elementów zbioru

S1:



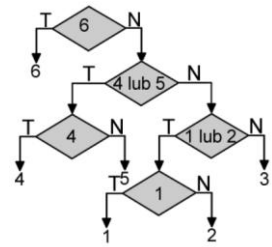
S2:



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

Identyfikacja elementów zbioru

S3:



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

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

Teoria informacji

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

Claude Shannon

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

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

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

Przykłady:

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

Dane dzisiaj

1024 Bajty = 1 Kilobajt
1024 Kilobajty = 1 Megabajt
1024 Megabajty = 1 Gigabajt
1024 Gigabajty = 1 Terabajt
1024 Terabajty = 1 Petabajt
1024 Petabajty = 1 Eksabajt
1024 Exabajty = 1 Zettabajt
1024 Zettabajty = 1 Jottabajt

Pomiędzy zaraniem cywilizacji a rokiem 2003 zostało stworzonych 5 Exabajtów informacji, dziś tworzymy tyle danych w 2 dni.

Prezes Google, Eric Schmidt, 2010 Konferencja Google Atmosphere

Cała wiedza świata

- W 2000 r. tylko 25 proc. wszystkich danych zebranych na świecie była w **formie cyfrowej** (reszta na papierze, taśmie filmowej, płytach winylowych, kasetach itp.).
- W 2013 r. w cyfrowej formie zapisane zostało 98 proc. wszystkich danych zgromadzonych na świecie – 1200 eksabajtów (EB).

Kompresja danych

Ciąg danych
AABACADABA

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

2) Kodowanie Huffmana

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

Po zakodowaniu

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

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

Podstawowe pojęcia

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

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

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

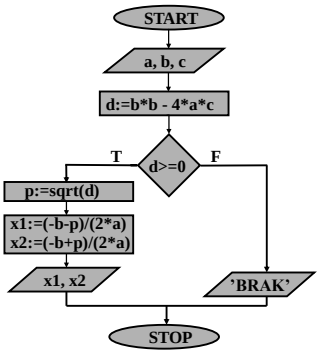
Przykład zapisu algorytmu

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

Postać werbalna algorytmu**:**

„Mając dane współczynniki a,b,c :
    oblicz d = b*b-4*a*c
Jeżeli d jest nieujemne :
    oblicz p = sqrt(d)
    oblicz x1 = (-b-p)/(2*a)
    oblicz x2 = (-b+p)/(2*a)
    wypisz wartości x1, x2
Jeżeli d jest ujemne :
    wypisz "BRAK PIERWIASTKÓW"

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)

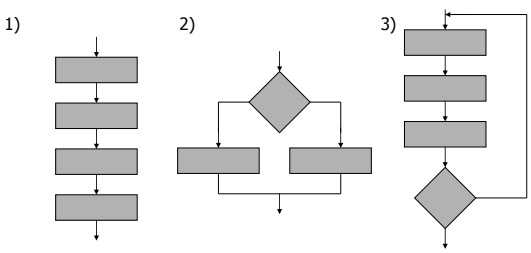
```
begin
  read(a,b);
  while a<>b do
    if a>b then
      a:=a-b
    else
      b:=b-a;
  writeln(a)
end.
```

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

Budowa algorytmów

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

1) 2) 3)



Początek nauki algorytmiki

- Programy z pętlami
- Zadania z tablicami jednowymiarowymi
- Zadania z tablicami wielowymiarowymi
- Zastosowania rekordów
- Procedury i funkcje
- Rekurencja
- Wskaźniki, alokacja pamięci
- Podstawowe algorytmy

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.
```

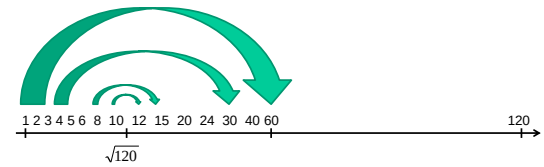
Rozkład na czynniki pierwsze

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.
```

Położenie podzielników liczby

Podzielniki liczby 120



Rozkład na czynniki pierwsze

Rozwiązanie dobre

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

Porównanie rozwiązań

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

Czy można jeszcze szybciej?

Pytania i zadania

- Ile informacji zawiera 10 znakowe słowo, którego każdy znak z jednakowym prawdopodobieństwem jest jedną z liter a, b, c ?
- Jak stworzyć kody Huffmana dla zbiorów 5,6,7 elementowych?
- Jak przyspieszyć działanie algorytmu Euklidesa?
- Ile czasu potrzebuje w najgorszym przypadku trzeci algorytm aby rozłożyć na czynniki 50 cyfrową liczbę?
- Jak przyspieszyć działania programu rozkładu na czynniki pierwsze?
- Proszę zaproponować algorytm rozkładający liczbę naturalną N na sumę składników, tak aby ich iloczyn był największy, np. 5 -> 2*3, 7 -> 2*2*3, 9 -> 3*3*3.

Wykład 2

- Aspekty języka programowania
- Instrukcje w języku proceduralnym
- Konstrukcje strukturalne

Przykładowy program

```
procedure main()
  p:=[]
  n:=1
  repeat
   (n+:=1)%!p=0 | put(p,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)
- Produkcje
- Metasymbole
  - **< >** - symbol pomocniczy
  - **::=** - symbol produkcji
  - **|** - symbol alternatywy
  - **[ ]** - wystąpienie 0 lub 1 raz (EBNF)
  - **{ }** - powtórzenie 0 lub więcej razy (EBNF)

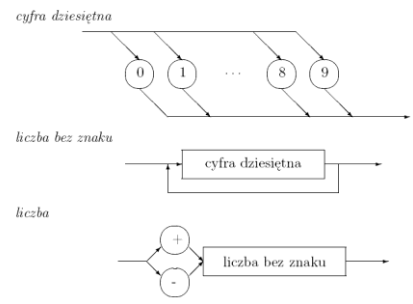
Przykład

<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



•6

Gramatyka języka

<pgm> ::= <pgmHeading> <pgmDeclarations> <codeBlock> "."
<pgmHeading> ::= program <pgmIdentifier> ";"
<pgmDeclarations> ::= { <pgmDeclaration> }
<pgmDeclaration> ::= <varDeclaration> | <typeDeclaration> | <procDeclaration>
....
<codeBlock> ::= begin { <statement> } end
<statement> ::= <assignStatement> | <ifStatement> | <whileStatement> | <procCall>
<assignStatement> ::= <variable> ":=" <expression> ";"
<ifStatement > ::= if <expression> then <codeBlock> [else <codeBlock>] ";"
<whileStatement> ::= while <expression> do <codeBlock> ";"
....
<identifier> ::= <letter> { <letter> | <digit> }
<longint> ::= <digit> { <digit> }
<relOperator> ::= "=" | "<>" | "<" | "<=" | ">" | ">="
<addOperator> ::= "+" | "-" | or
<multOperator> ::= "*" | div | and
<letter> ::= "A" | ... | "Z" | "a" | ... | "z"
<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Semantyka

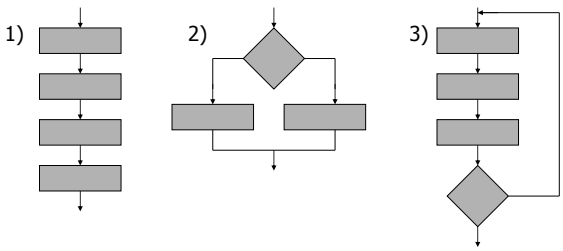
Opis każdej konstrukcji językowej poprawnej składniowo

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

•8

Struktury sterujące przebiegiem programu

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



Instrukcje proste i strukturalne

proste	**strukturalne**
• przypisania	• warunkowa
• wywołania funkcji	• wyboru
• operacja we/wy	• iteracyjne

10

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
- Nieobliczalność wyrażenia

11

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

12

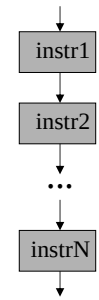
Operacja we/wy

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

Przykłady:
        cout  << x;
        cout  << x+y;
        cout  << ”hello”;
        cin >> x;

Problemy:
• Konwersja typów
```

13

Ciąg instrukcji

```
{
  <instrukcja1>;
  <instrukcja2>;
  ...
  <instrukcjaN>;
}
```

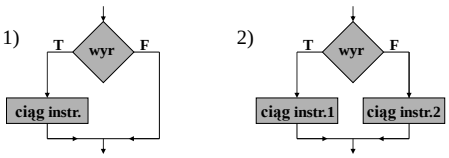


14

Instrukcja skoku warunkowego

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

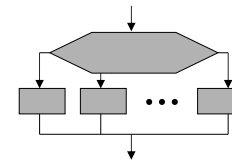
2) **if** (<*wyrażenie*>) **{** <*ciąg instrukcji 1*> **}**
else { <*ciąg instrukcji 2*> **}**



15

Instrukcja wyboru

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



Przykład:

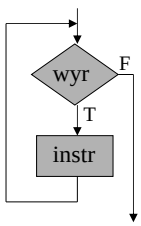
```
switch (dzien_tygodnia) {
  case 0 : printf(”niedziela”);  break;
  case 1 : printf(”poniedziałek”); break;
  ...
  case 6 : printf(”sobota”); break;
  default : printf(”zla wartosc”);
}
```

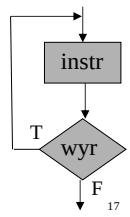
16

Instrukcje pętli

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

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



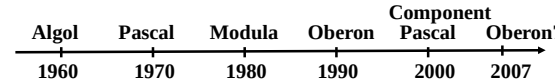


17

Konstrukcje strukturalne

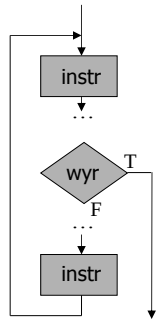
begin ... end
if ... then ...
if ... then ... else ...
case ... of ...
while ... do ...
repeat ... until ...
for ... to ... do ...

Niklaus Wirth



Instrukcja break

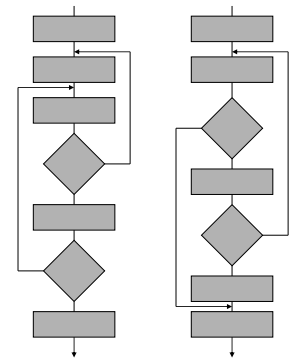
```
while (True) {
  ...
  if (<wyrażenie>) break;
  ...
}
```



19

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>) {
  <ciąg instrukcji>;
}
```

```
<inst1>;
while (<war>) {
 <ciąg instrukcji>;
  <inst2>;
}
```

Przykład:

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

21

Pytania i zadania

- Czym różni się notacja BNF od notacji EBNF?
- Zapisać w notacji EBNF składnię instrukcji warunkowej oraz pętli.
- Zapisać w notacji EBNF składnię wyrażenia arytmetycznego.
- Które z 6 instrukcji: *if , if else, switch, while, do while, for* można usunąć z języka aby nadal można było w nim programować?
- Liczb pierwszych jest nieskończenie wiele. Tylko 7 z nich spełnia warunek:

  sum_p(N)=N

  gdzie:

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

  np. sum_p(2016)=16+0+1+1296=1313

  sum_p(2017)=16+0+1+2401=2418

  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
  ! or and == !=
- **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

Typ	Zakres	Rozmiar
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 , long int — 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

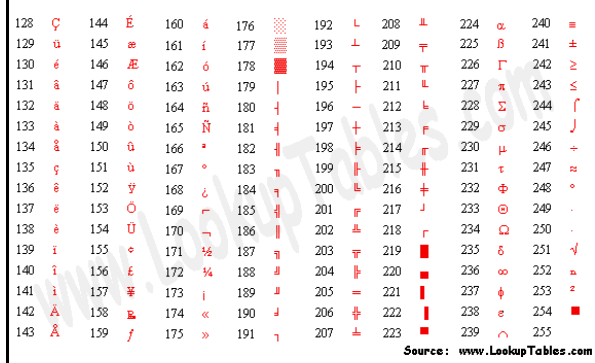
Funkcje standardowe

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

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



Standard ISO-8859

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

Unicode

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

Unicode - wspólny dla całego świata zestaw znaków.

Unicode

UTF-8

128 znaków (ASCII) kodowanych jest za pomocą 1 bajta.
1920 znaków (alfabety łaciński, grecki, armeński, hebrajski, arabski, koptyjski i cyrylica) kodowanych jest za pomocą 2 bajtów.
63488 znaków (m.in. alfabety chiński i japoński) kodowanych jest za pomocą 3 bajtów.
Pozostałe 2147418112 znaki (jeszcze nie przypisane) można zakodować za pomocą 4, 5 lub 6 bajtów.

UCS-2

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

UCS-4

Wszystkie znaki zapisywane są za pomocą 4 bajtów.

Unicode (UTF-8)

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

Kodowanie „polskich” znaków

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

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<>rewers(n) do
    n := n+rewers(n)        { dodawanie arytmetyczne }
end.
```

rewers(n) to liczba n zapisana od końca
Czy powyższy program zakończy się dla każdej liczby naturalnej?
Proszę sprawdzić to dla wszystkich liczb n<200.

17

Wykład 4

- Strukturalne typy danych
  - Tablice
  - Tablice znaków (napisy)
  - Klasa string
  - 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');	Pascal
printf("to jest tekst");	Język C
cout << "to jest tekst";	Język C++

Deklarowanie:

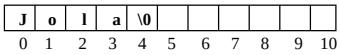
```
char buf[11];
char imie[] = "Jola";
```

Operacje we/wy:

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

Uwaga na spacje!

Reprezentacja:



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);
```

Klasa string

Plik nagłówkowy

```
#include <string>
```

Deklarowanie:

```
string buf;
string imie = "Jola";
```

Operacje we/wy:

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

Operacje:

```
buf = imie;
if (buf==imie) ...
buf = buf + "koniec"
buf[0] = '_';
```

Klasa string - metody

empty()	Zwraca wartość true jeżeli napis jest pusty.
size(), length()	Zwraca liczbę znaków w napisie.
at()	Zwraca znak o podanym położeniu, tak jak operator [], wyjątek w przypadku wyjścia poza zakres napisu.
clear()	Usuwa wszystkie znaki z napisu.
erase(...)	Usuwa wybrane znaki.
find(...)	Znajduje podciąg w ciągu.
swap(...)	Zamienia miejscami dwa napisy, a staje się b, a b staje się a.
substr(...)	Zwraca podciąg na podstawie indeksu początkowego i długości podciągu.
append(...)	Dodaje zadany napis na końcu istniejącego ciągu.
c_str()	Zwraca napis w stylu języka C (stały wskaźnik typu const char*).

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

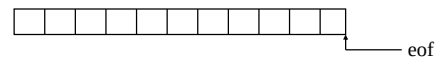
```
struct data
{
  int dzien;
  int miesiac;
  int rok;
};

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 - odczyt

```
int main()
{
    string linia;
    fstream plik;

    plik.open("plik.txt", ios::in);
    if(plik.good())
    {
        while(!plik.eof())
        {
            getline(plik, linia);
            cout << linia << endl;
        }
        plik.close();
    }
    return(0);
}
```

Pliki - zapis

```
int main()
{
    fstream plik;

    plik.open("plik.txt", ios::out | ios::trunc);
    if(plik.good())
    {
        plik << "tekst";
        plik.close();
    }

    return(0);
}
```

Pytania i zadania

- Co zwraca funkcja abs() dla najbardziej ujemnej liczby?
- Co powoduje x++ dla największej reprezentowanej w systemie liczby?
- Co zwraca operacja % gdy jeden/oba argumenty są ujemne?
- Jaka jest kolejność bajtów w liczbie typu int?
- Jak w języku C/C++ umieszczone są w pamięci tablice dwuwymiarowe?
- Jak mając dostępną funkcję trunc() zrealizować funkcję round()?
- Jak zamienić wartościami dwie zmienne typu int bez użycia zmiennej pomocniczej?
- Jak wyznaczyć wartość maksymalną 10 elementowej tablicy nie używając operatorów relacyjnych?
- Jak sprawdzić zakres liczb całkowitych nie mając dokumentacji?
- Jak wyznaczyć dokładność liczb zmiennopozycyjnych bez dokumentacji?
- Napisać w C/C++ najkrótszy program wypisujący sam siebie.

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 …	*dodatkowe funkcje*
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
- możliwość użycia rekurencji

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

Składnia definicji funkcji

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

Przykład:

```
double srednia( double x, double y)
{
  return (x+y)/2.0;
}
```

1/2

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()
{
  ...
}
```

2/2

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 (argumenty)

```
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

1/2

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
}
```

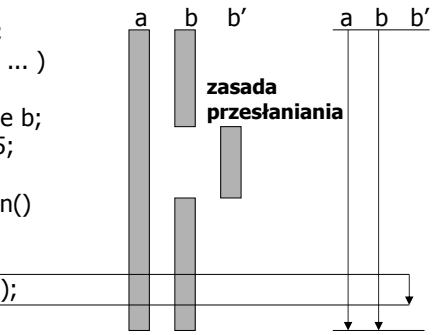
Przez referencję

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

void main()
{

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

2/2

Obszar określoności i czas trwania

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



Przykłady zastosowania

Funkcja obliczająca silnię *(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:

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

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

Przykłady zastosowania

Ciąg Fibonacciego

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

1	1	2	3	5	8	13	21				

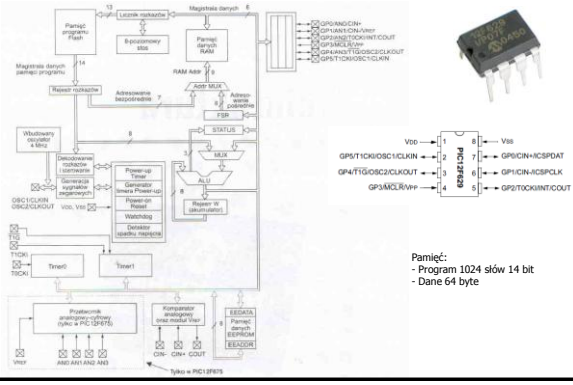
```
int  fib(int n)
{
  if (n<3)
    return 1;
  else
    return  fib(n-1)+fib(n-2);
}
```

Wizualizacja rekurencji

```
int fib(int n) {
  int result;

  cout << "start " << n << endl;

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

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

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

```
start 5
  start 4
    start 3
      start 2
      stop 2
      star t1
      stop 1
    stop 3
    start 2
    stop 2
  stop 4
  start 3
    start 2
    stop 2
    start 1
    stop 1
  stop 3
stop 5
```

Architektura PIC 12F629



Pamięć:
- Program 1024 słów 14 bit
- Dane 64 byte

Lista rozkazów

Mnemonik	Arg.	Opis w jęz. asq.	Dzia
NOP		No Operation	Cykl pusty
CLRW		Clear W	Zeruj W
CLRF	f	Clear F	Zeruj F
COMF	f, d	Complement F	Zanegu F wynik prześlij do d
MOVF	f, d	Move F to d	Prześlij F do d
MOVWF	f	Move F to W	Prześlij W do F
MOVLW	k	Move k to W	Prześlij k do W
ADDWF	f, d	Add W to F	Dodaj W do F wynik prześlij do d
ADDLW	k	Add k to W	Dodaj k do W
INCF	f, d	Increment F	Zwiększ F o 1, wynik prześlij do d
SUBWF	f, d	Subtract W from F	Odejmij W od F, wynik prześlij do d
SUBLW	k	Subtract k from W	Odejmij k od W
DECF	f, d	Decrement F	Zmniejsz F o 1, wynik prześlij do d
ANDWF	f, d	AND W and F	Iloczyn logiczny W i F, wynik prześlij do d
ANDLW	k	AND k and W	Iloczyn logiczny k i W
IORWF	f, d	OR W with F	Suma logiczna W i F, wynik prześlij do d
IORLW	k	OR k with W	Suma logiczna k i W
XORWF	f, d	XOR W with F	Suma modulo 2 W i F, wynik prześlij do d
XORLW	k	XOR k with W	Suma modulo 2 k i W
RLF	f, d	Rotate Left F	Przesuń F o 1 bit w lewo, wynik prześlij do d
RRF	f, d	Rotate Right F	Przesuń F o 1 bit w prawo, wynik prześlij do d
SWAPF	f, d	Swap nibbles in F	Zamień tetrady w F, wynik prześlij do d
BCF	f, b	Bit Clear F	Wyzeruj bit b w F
BSF	f, b	Bit Set F	Ustaw bit b w F
BTFSC	f, b	Bit Test F, Skip if Clear	Jeśli bit b w F = 0, to omiń następną instrukcję
BTFSS	f, b	Bit Test F, Skip if Set	Jeśli bit b w F = 1, to omiń następną instrukcję
INCFSZ	f, d	Inc. F Skip if Zero	Zwiększ F o 1, jeśli wynik = 0 omiń następną instr.
DECFSZ	f, d	Dec. F Skip if Zero	Zmniejsz F o 1, jeśli wynik = 0 omiń następną instr.
GOTO	s	Go To address s	Skok bezwarunkowy do etykiety s
CALL	s	Call subroutine	Wywołaj podprogram o etykiecie s
RETURN		Return from subroutine	Powrót z podprogramu
RETLW	k	Return with k in W	Powrót ze stałą k w W
RETFIE		Return From Interrupt	Powrót z przerwania
CLRWDT		Clear WDT	Zeruj licznik WDT
SLEEP		Go to standby mode	Przejdź w stan uśpienia

gdzie:
f – 7-bitowy adres rejestru (0...127)
F – zawartość rejestru o adresie f (0...255)
W – zawartość rejestru roboczego (0...255)
d – adres wyniku operacji (0,1):
gdy $d = 0$, wynik operacji jest przesyłany do W
gdy $d = 1$, wynik operacji jest przesyłany do F
b – numer bitu (0...7)
k – 8-bitowa stała liczbowa (0...255)
s – 11-bitowa etykieta (0...2047)

Przykładowy program

```
#include "12f629.h"

int nwd(int a, int b) {
  while (a!=b) {
    if (a>b) a=a-b;
    else b=b-a;
  }
  return a;
}


void main() {
  int x,y,z;

  x=24;
  y=30;
  z=nwd(x,y);
}
```

Kompilacja

```
Carry   EQU  0
Zero_   EQU  2
a       EQU  0x23
b       EQU  0x24
x       EQU  0x20
y       EQU  0x21
z       EQU  0x22

                        ;int nwd(int a, int b)
nwd
                        ;  while (a!=b) {
m001    MOVF  a,W
        XORWF b,W
        BTFSC 0x03,Zero_
        GOTO  m003
                        ;    if (a>b) a=a-b;
        MOVF  a,W
        SUBWF b,W
        MOVF  a,W
        XORWF b,W
        BTFSC 0x03,Carry
        XORLW .128
        ANDLW .128
        BTFSS 0x03,Zero_
        GOTO  m002
        MOVF  b,W
        SUBWF a,1
                        ;    else b=b-a;
        GOTO  m001
m002    MOVF  a,W
        SUBWF b,1
                        ;  }
        GOTO  m001
                        ;  return a;
m003    MOVF  a,W
        RETURN
                        ;}
```

```
                        ;void main()
main
                        ;  x=24;
        MOVLW .24
        MOVWF x
                        ;  y=30;
        MOVLW .30
        MOVWF y
                        ;  z=nwd(x,y);
        MOVF  x,W
        MOVWF a
        MOVF  y,W
        MOVWF b
        CALL  nwd
        MOVF  a,W
        MOVWF z
                        ;}
        END
```

```
:10000000162823082406031914282308240223089
:1000100024060318803A8039031D1128240BA302FE
:10002000012823084020128230808001830A00092
:10003000103A1002008A3002108A40001202308ED
```

Maksymy i rady programistyczne

- Programy mają być czytane przez ludzi.
- Czytelność jest zwykle ważniejsza niż sprawność.
- Najpierw projekt potem kodowanie.
- 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”

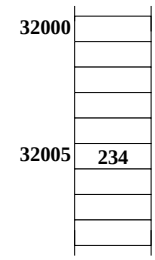
Wykład 7-8

- 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;
...
}
```



Instrukcja podstawienia

<zmienna> = <wyrażenie>

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

z ***op*** = wyr ⟺ z = z ***op*** wyr

Czas istnienia nazwy

Program w C/C++ —kompilacja→ Program wykonywalny

nazwa zmiennej → adres w pamięci

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

Język Icon

Funkcje transformujące

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

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

Język Turbo Pascal

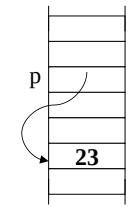
Zmienna wskaźnikowa

Zmienna wskaźnikowa - zmienna, która przechowuje adres innej zmiennej.

Zmienna wskazywana - zmienna, na którą wskazuje zmienna wskaźnikowa.

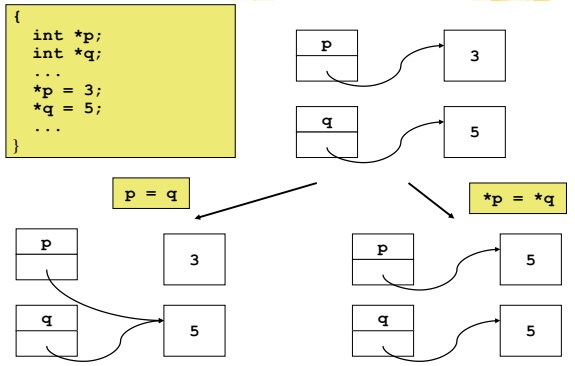
Deklaracja zmiennej wskaźnikowej:

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

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



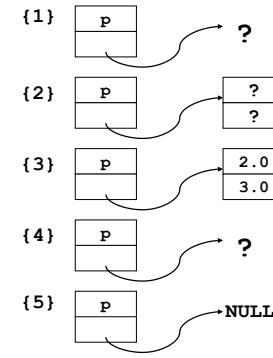
Zmienna wskaźnikowa i wskazywana

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



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}
```

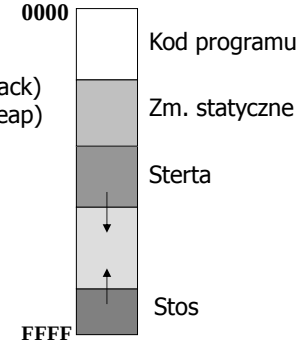


Operacje na zmiennych wskaźnikowych

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)



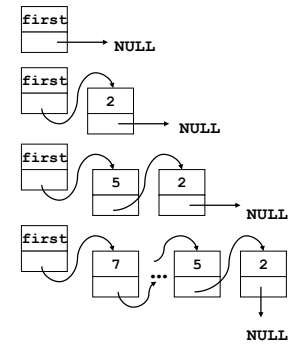
Zastosowania typu wskaźnikowego

- 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)

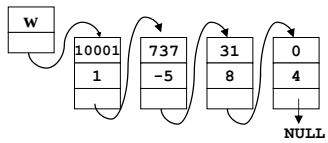
W(x)=x^4+5x^3-7x^2+x+3

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

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

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

```
struc 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

Wybrane algorytmy
- Wyszukiwanie połówkowe
- Sortowanie
  - Metody proste
  - Metody szybkie

Wyszukiwanie połówkowe

```
int t[MAX];

int find(int t[MAX], int sz) {
  int lewy = 0;
  int prawy = MAX-1;
  int sr;

  while (lewy<=prawy) {
    sr = (lewy+prawy)/2;
    if (t[sr]==sz)
      return sr;
    else if (t[sr]<sz)
      lewy = sr+1;
    else
      prawy = sr-1;
  }
  return -1;
}
```

Wyszukiwanie połówkowe

```
int t[MAX];

int find(int t[MAX], int sz) {
  int lewy = 0;
  int prawy = MAX-1;
  int sr;

  while (lewy<=prawy) {
    sr = (lewy+prawy)/2;
    if (t[sr]<sz)
      lewy = sr+1;
    else
      prawy = sr-1;
  }
  if (lewy<MAX and t[lewy]==sz) return lewy;
  return -1;
}
```

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 przestawianiu obiektów, aż do chwili osiągnięcia ich uporządkowania takiego, że dla danej funkcji porządkującej *f*:

  $f(a_1) <= f(a_2) <= ... <= f(a_n)$

  (*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 przestawiać 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:
  - metody proste $O(n^2)$,
  - metody szybkie O(n log n),
  - metody liniowe O(n).
- Według stabilności (rzadko istotna):
  - stabilne,
  - niestabilne.

7

Sortowania proste I

Przez proste wstawianie:

- dzielimy ciąg na wynikowy i źródłowy; w każdym kroku, począwszy od drugiego elementu, przenosimy element ciągu źródłowego do ciągu wynikowego, wstawiając go w odpowiednim miejscu,
- zachowanie naturalne, algorytm stabilny, działa w miejscu,
- próba poprawy przez wstawianie połówkowe: zmienia się tylko liczba porównań, a nie PRZESUNIĘĆ

8

Sortowania proste I cd.

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

void proste_wstawianie(int a[MAX]) {

  for (int i=2; i<MAX; i++){
    int x = a[i];
    a[0] = x;      // wartownik
    int j = i-1;
    while (x<a[j]) {
      a[j+1] = a[j];
      j = j-1;
    }
    a[j+1] = x;
  }
}
```

9

Sortowania proste I cd.

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

void proste_wstawianie2(int a[MAX]) {
  int lewy,prawy,sr;
  int x;

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

10

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 małych rozmiarów tablic.

11

Sortowania proste II cd.

```
int a[MAX];  // sortowane elementy 0..MAX-1

void proste_wybieranie(int a[MAX]) {

  for (int i=0; i<MAX-1; i++) {
    int nm = i;                  // wybieranie minimum
    for (int j=i+1; j<MAX; j++) {
      if (a[j]<a[nm]) nm = j;
    }
    int tmp=a[nm]; a[nm]=a[i];  a[i]=tmp;
  }
}
```

12

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 ⇒ nie stosuje się.

Sortowania proste III cd.

```
int a[MAX]; // sortowane elementy 0..MAX-1

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

  for (int i=1; i<MAX; i++)
    for (int j=MAX-1; j>=i; j--)
      if (a[j-1]>a[j]) {
        int tmp=a[j-1]; a[j-1]=a[j]; a[j]=tmp;
      }
}
```


Sortowania szybkie

Sortowanie grzebieniowe Combsort:

- pochodzi z roku 1991,
- oparta na metodzie bąbelkowej,
- 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:

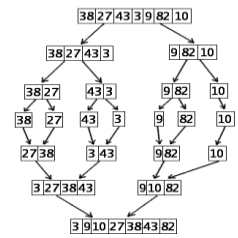
- podstawowy - za rozpiętość przyjmij długość tablicy, podziel rozpiętość przez 1.3, odrzuć część ułamkową, będzie to pierwsza rozpiętość badanych par obiektów, badaj wszystkie pary o tej rozpiętości, jeżeli naruszają monotoniczność, to przestaw; wykonuj w pętli (rozpiętość podziel znów przez 1.3); kontynuując 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..MAX-1

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

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


Sortowanie przez scalanie

1. Podziel ciąg danych na dwie równe części
2. Zastosuj sortowanie przez scalanie dla każdej z nich oddzielnie, chyba że pozostał już tylko jeden element;
3. Połącz posortowane podciągi w jeden.



Sortowanie przez scalanie

```
void merge (int t[], int start, int srodek, int koniec) {

  int *t_pom = new int[(koniec-start)];   // tablica pomocnicza
  int i = start, j = srodek+1, k = 0;

  while (i<=srodek and j<=koniec) {
    if (t[j]<t[i]) { t_pom[k] = t[j]; j++; }
    else { t_pom[k] = t[i]; i++; }
    k++;
  }

  while (i<=srodek) { t_pom[k] = t[i]; i++; k++; }
  while (j<=koniec) { t_pom[k] = t[j]; j++; k++; }

  for (i=0; i<=koniec-start; i++) t[start+i] = t_pom[i];
}
```


Sortowanie przez scalanie

```
void merge_sort (int t[], int start, int koniec){

  int srodek;

  if (start != koniec) {
    srodek = (start+koniec)/2;
    merge_sort (t, start, srodek);
    merge_sort (t, srodek+1, koniec);
    merge (t, start, srodek, koniec);
  }
}
```

Liczba operacji podczas sortowania tablicy N elementowej jest proporcjonalna do N*log(N)

Sortowanie porównanie czasów

Rozmiar tablicy	Bubble sort	Merge sort
10^3	4,9 ms	0.22 ms
10^4	446 ms	2.44 ms
10^5	44.6 s	29.38 ms
10^6	74 min	340.4 ms
10^7	123 h	4 s

Wykład 10

- Struktury liniowe o zmiennym podłożu
  - Stos, kolejka
  - Implementacje struktur
- Zastosowania stosu
  - Derekursywacja funkcji
  - Obliczanie wartości wyrażenia

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 (dostęp do elementów z obu stron),
  - 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)`.
- Uwaga:
  - Stos określany 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)`.
- Uwaga:
  - Kolejka 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

Stos implementacja tablicowa

```
struct stos {
  int t[MAX];
  int size;
};

void init(stos &st) { st.size=0; }
void push(stos &st, int el) { st.t[st.size++]=el; } // brak kontr
int pop(stos &st) { return st.t[--st.size]; }    // brak kontroli
bool empty(stos &st) { return (st.size==0); }


int main() {
  stos s;

  init(s);
  for (int i=0; i<10; i++) push(s,i*i);
  while (!empty(s)) cout<<pop(s)<<endl;
}
```

Stos implementacja tablicowa

```
struct stos {
  int t[MAX];
  int size;

  void init();
  void push(int el);
  int pop();
  bool empty();
};

void stos::init() { size=0; }
void stos::push(int el) { t[size++]=el; }   // bez kontroli
int stos::pop() { return t[--size]; }       // bez kontroli
bool stos::empty() { return (size==0); }

int main() {
  stos s;
  s.init();
  for (int i=0; i<10; i++) s.push(i*i);
  while (!s.empty()) cout<<s.pop()<<endl;
}
```

7

Stos implementacja tablicowa

```
struct stos {
  int t[MAX];
  int size;

  stos();
  void push(int el);
  int pop();
  bool empty();
};

stos::stos() { size=0; }
void stos::push(int el) { t[size++]=el; }   // bez kontroli
int stos::pop() { return t[--size]; }       // bez kontroli
bool stos::empty() { return (size==0); }

int main() {
  stos s; // konstruktor wywołany automatycznie
  s.init();
  for (int i=0; i<10; i++) s.push(i*i);
  while (!s.empty()) cout<<s.pop()<<endl;
}
```

8

Sortowania przez podział

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,

9

Sortowania przez podział

```
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);
}
```

10

Sortowania przez podział

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

  init(st);
  push(st,l); push(st,p);
  while (!empty(st)) {
    p=pop(st); l=pop(st);

    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) { push(st,l); push(st,j); }
    if (i < p) { push(st,i); push(st,p); }
  }
}
```

11

Postać wyrażenia

Elementy wyrażenia:
- Zmienne: x, y, z, ...
- Stałe: 12, -5, ...
- Operatory dwuargumentowe: ^ * / + - (priorytet i łączność)
- Minus unarny: -
- Nawiasy: ( )
- Funkcje: sin(x), cos(x), ...

Postać wyrostkowa, infiksowa:
(x+y) - (z*3)

Postać przedrostkowa, prefiksowa, notacja Łukasiewicza:
- (+(x,y), *(z,3))

Postać przyrostkowa, postfiksowa (Reverse Polish Notation, RPN):
x y + z 3 * -

Obliczanie wyrażeń

```
procedure onp(w) # zamienia wyrażenie infiksowe na postfiksowe
   while w ?:= 2(="(",tab(bal(')')),pos(-1))

   w ? every p:=bal('+-')
   if \p then return onp(w[1:p])||onp(w[p+1:0])||w[p]

   w ? every p:=bal('*/')
   if \p then return onp(w[1:p])||onp(w[p+1:0])||w[p]

   w ? p:=bal('^')
   if \p then return onp(w[1:p])||onp(w[p+1:0])||w[p]

   return(w)
end
```

Obliczanie wyrażeń

```
procedure value(w)
  st:=[]
  every el:=!w do
    if any('+-*/^',el) then push(st,a:=pop(st) & el(pop(st),a))
                       else push(st,variable(el))
  return pop(st)
end

global x,y,z

procedure main()
  x:=2
  y:=3
  z:=5
  while write(value(onp(read())))
end
```

Obliczanie wyrażeń

```
mag@wierzba:/home/glk/mag$ ./wyr
x+y
5
x*(y+z)
16
x^y^x
512
x^(y+z)/x
128
```

Zadania

1. Dana jest funkcja określona rekurencyjnie:
   f(0,b)=b+1
   f(a,0)=f(a-1,1) a>0
   f(a,b)=f(a-1,f(a,b-1)) a>0, b>0
   Proszę napisać funkcję w wersji rekurencyjnej oraz iteracyjnej.

2. Dana jest tablica int t[MAX] zawierająca nieuporządkowane liczby naturalne. Proszę zmodyfikować funkcję qs, tak aby szybko wyznaczyć sumę k najmniejszych wartości z tablicy. Przykładowe dane to MAX=1000000 i k=50.

3. Napisać w jezyku C++ program wyliczający wartości wyrażeń.

Wykład 11

- Reprezentacja liczb w komputerze
- Liczby stałopozycyjne
- Liczby zmiennopozycyjne
- Dokładność obliczeń

Pozycyjny system liczenia

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

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

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

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

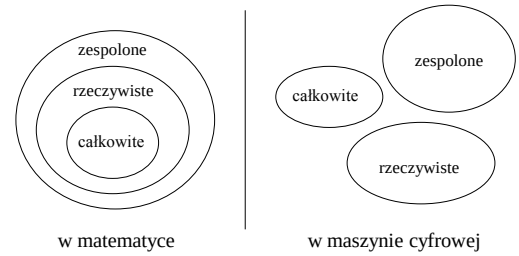
$0.11_{(2)} = 0*2^0 + 1*2^{-1} + 1*2^{-2} = 0.75$

Arytmetyka liczb w komputerze

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

Rodzaje liczb

- liczby całkowite
- liczby rzeczywiste
- liczby zespolone



w matematyce | w maszynie cyfrowej

Reprezentacja liczb stałopozycyjnych

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

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

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

Działania na liczbach kodu U2

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

6	0 0000110
+7	0 0000111
13	**0 0001101**

6	0 0000110
-7	1 1111001
-1	**1 1111111**

64	0 1000000
-64	1 1000000
0	**0 0000000**

-65	1 0111111
-65	1 0111111
-130	**0 1111110**

NADMIAR!

Liczby całkowite

Turbo Pascal

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

Java

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

Liczby całkowite

FPC

Typ	Zakres	Rozmiar
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 , long int — ma 32 bity
- long long int — ma 64 bity

Liczby zmiennopozycyjne

Cel: Oddzielenie zakresu od dokładności

Sposób zapisu:

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

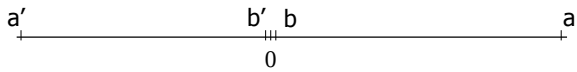
l - ***liczba***
m - ***mantysa***
c - ***cecha***

Liczby zmiennopozycyjne

Przykład 1. System dziesiętny

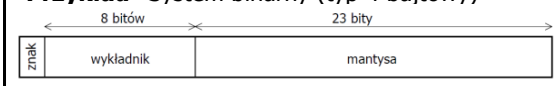
mantysa - 3 cyfry + znak (0.000 - 0.999)
cecha - 2 cyfry + znak (-99 - 99)

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



Liczby zmiennopozycyjne

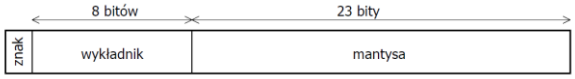
Przykład System binarny (typ 4 bajtowy)



- Znak mantysy: 0 lub 1
- Wykładnik w kodzie U2 [-128..127]
- Mantysa 0 lub [0.5..1)
- Liczba to mantysa * 2 cecha
- Liczba znormalizowana : najbardziej znacząca cyfra mantysy równa 1
- Zakres liczb: od $-(1-2^{-23})\cdot 2^{127}$ do $(1-2^{-23})\cdot 2^{127}$
- Minimalna dodatnia: $1/2\cdot 2^{-128}$

Liczby zmiennopozycyjne

Przykład: 4 bajtowe liczby typu float



Znak mantysy: 0 lub 1
Wykładnik (cecha) przesunięty o 127
Mantysa [1-2)
Liczba to mantysa * 2 cecha
Problem reprezentacji 0

Liczby zmiennopozycyjne

Przykład: 4 bajtowe liczby typu float

- liczby dodatnie: od 2^{-127} do $(2-2^{-23}) \cdot 2^{128}$
- liczby ujemne: od $-(2-2^{-23}) \cdot 2^{128}$ do -2^{-127}
- przepełnienie (nadmiar): Liczby$>(2-2^{-23}) \cdot 2^{128}$

Liczby zmiennopozycyjne

```
union fi {
  float a;
  unsigned int b;
};

int main() {
  fi x;
  int t[100];

  while (true) {
    cin >> x.a;

    for (int i=31; i>=0; i--) { t[i]=x.b%2; x.b=x.b/2; }
    cout << t[0] << " ";
    for (int i=1; i<=8; i++) cout << t[i];
    cout << " ";
    for (int i=9; i<=31; i++) cout << t[i];
    cout << endl;
  }
}
```

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

Standard ANSI IEEE 754

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

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

1/2

Dokładność obliczeń

Obliczenia iteracyjne

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

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

iter	typ **extended**	wartość dokładna
1.	0.039700000000000	0.039700000000000000000000000000000000000000
2.	0.154071730000000	0.154071729999999999999999999999999999999999
3.	0.545072626044421	0.545072626044421299999999999999999999999999
4.	1.288978001188800	1.288978001188800564146077444318929999999999
5.	0.171519142109176	0.171519142109176561091327376119366950153184 8
...		
50.	1.314491071714711	1.314489760648214812383780183964860775887564
51.	0.074303954005774	0.074309050045856049321771955401870933940599
52.	0.280652583280420	0.280670695427271580509655379893949005186949
53.	0.886312715615762	0.886354663894201577758577930930197101949547
54.	1.188600172876488	1.188544884955797172985886514749765353448385
55.	0.516089578619899	0.516262709159421280638670823473078496363871
56.	1.265312954999400	1.265469282031809506189821162818058704556637
57.	**0.258201197729656**	0.257639616828927735003403861501470932678094
...		
92.	1.137929025629373	0.271467233262087610549464473055410611454272
93.	0.667068700408049	0.864785556843432384355119764618126915551368
94.	1.333332848439945	1.215580049398713266568656814984364161700898
95.	0.000001939572845	0.429415628106318510122842974986664350734709
96.	0.000007758280094	1.164469167439441909702911183666688004618360
97.	0.000031032939806	0.589911344006446644012487937540734261222550
98.	0.000124128870095	1.315659194663309877447649200395210188456166
99.	0.000496469256453	0.069759429146912168060579275901754241874755
100.	0.001985137580646	0.264438582722939495435219262057398282867365

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

typ **real**

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

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

typ **double**

0.100000000	1.00125550119637	1.00125550119638
0.010000000	1.00001250054690	1.00001250054691
0.001000000	1.00000012500005	1.00000012500005
0.000100000	1.00000000125000	1.00000000125000
0.000010000	1.00000000001250	1.00000000001250
0.000001000	1.00000000000018	1.00000000000013
0.000000100	1.00000000000000	1.00000000000000
0.000000010	0.99999999999864	1.00000000000000
0.000000001	0.99999999996103	1.00000000000000

typ **extended**

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

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

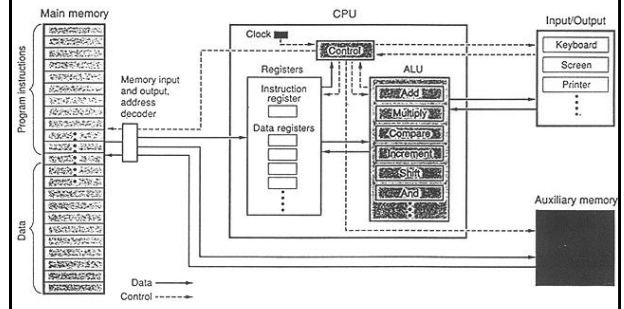
wartości dokładne

0.100000000	1.00125550119637747391785450350122038359405315883
0.010000000	1.00001250054690722874466702747516135163757833258
0.001000000	1.00000012500005468753222658432008437730121685070
0.000100000	1.00000000125000005468750032226562718200685195 54
0.000010000	1.00000000001250000000054687500003222656250218200
0.000001000	1.00000000000012500000000005468750000032226562 50
0.000000100	1.00000000000000125000000000005468750000000 3222
0.000000010	1.00000000000000001250000000000000054687500000000
0.000000001	1.00000000000000000012500000000000000005468750000

Wykład 12-13

- Architektura komputera
- Języki programowania
- System operacyjny
- Superkomputery

Architektura prostego komputera

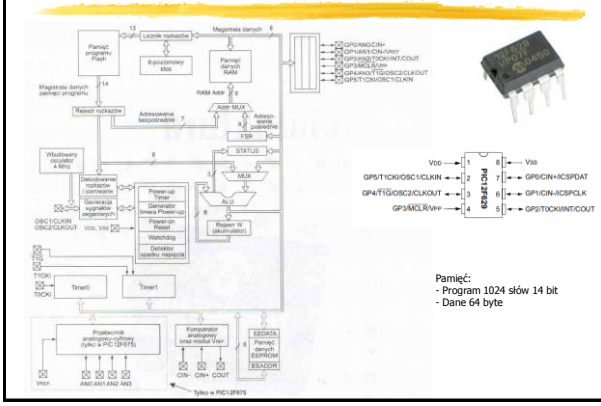


Poziom języka programowania

Język Pascal	Język asemblera	Kod maszynowy
`while i<>j do`	`LOOP LOAD I`	`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`
		`:`
		`:`
		`:`
		`B000` } zmienne
		`B003`
		`B006`
		`:`
		`:`

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

Architektura PIC 12F629



Pamięć:
- Program 1024 słów 14 bit
- Dane 64 byte

Lista rozkazów

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb	LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS							
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff		C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff		Z	1,2
CLRF	f	Clear f	1	00 0001 1fff ffff		Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx		Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff		Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff		Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff			1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff		Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff			1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff		Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff		Z	1,2
MOVWF	f	Move W to f	1	00 0000 1fff ffff			
NOP	-	No Operation	1	00 0000 0xx0 0000			
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff		C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff		C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff		C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff			1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff		Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS							
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff			1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff			1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff			3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff			3
LITERAL AND CONTROL OPERATIONS							
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk		C,DC,Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk		Z	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk			
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100		TO,PD	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk			
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk		Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk			
RETFIE	-	Return from interrupt	2	00 0000 0000 1001			
RETLW	k	Return with literal in W	2	11 01kx kkkk kkkk			
RETURN	-	Return from Subroutine	2	00 0000 0000 1000			
SLEEP	-	Go into Standby mode	1	00 0000 0110 0011		TO,PD	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk		C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk		Z	

Lista rozkazów

Mnemonik	Arg.	Opis w jęz. asg.	Opis
NOP		No Operation	Cykl pusty
CLRW		Clear W	Zeruj W
CLRF	f	Clear F	Zeruj F
COMF	f, d	Complement F	Zanegуj F, wynik prześlij do d
MOVF	f, d	Move F to d	Prześlij F do d
MOVWF	f	Move F to W	Prześlij W do F
MOVLW	k	Move k to W	Prześlij k do W
ADDWF	f, d	Add W to F	Dodaj W do F, wynik prześlij do d
ADDLW	k	Add k to W	Dodaj k do W
INCF	f, d	Increment F	Zwiększ F o 1, wynik prześlij do d
SUBWF	f, d	Subtract W from F	Odejmij W od F, wynik prześlij do d
SUBLW	k	Subtract k from W	Odejmij k od W
DECF	f, d	Decrement F	Zmniejsz F o 1, wynik prześlij do d
ANDWF	f, d	AND W and F	Iloczyn logiczny W i F, wynik prześlij do d
ANDLW	k	AND k and W	Iloczyn logiczny k i W
IORWF	f, d	OR W with F	Suma logiczna W i F, wynik prześlij do d
IORLW	k	OR k with W	Suma logiczna k i W
XORWF	f, d	XOR W with F	Suma modulo 2 W i F, wynik prześlij do d
XORLW	k	XOR k with W	Suma modulo 2 k i W
RLF	f, d	Rotate Left F	Przesuń F o 1 bit w lewo, wynik prześlij do d
RRF	f, d	Rotate Right F	Przesuń F o 1 bit w prawo, wynik prześlij do d
SWAPF	f, d	Swap nibbles in F	Zamień tetrady w F, wynik prześlij do d
BCF	f, b	Bit Clear F	Wyzeruj bit b w F
BSF	f, b	Bit Set F	Ustaw bit b w F
BTFSC	f, b	Bit Test F, Skip if Clear	Jeśli bit b w F = 0, to omiń następną instrukcję
BTFSS	f, b	Bit Test F, Skip if Set	Jeśli bit b w F = 1, to omiń następną instrukcję
INCFSZ	f, d	Inc. F Skip if Zero	Zwiększ F o 1, jeśli wynik = 0 omiń następną instr.
DECFSZ	f, d	Dec. F Skip if Zero	Zmniejsz F o 1, jeśli wynik = 0 omiń następną instr.
GOTO	s	Go To address s	Skok bezwarunkowy do etykiety s
CALL	s	Call subroutine	Wywołaj podprogram o etykiecie s
RETURN		Return from subroutine	Powrót z podprogramu
RETLW	k	Return with k in W	Powrót ze stałą k w W
RETFIE		Return From Interrupt	Powrót z przerwania
CLRWDT		Clear WDT	Zeruj licznik WDT
SLEEP		Go to standby mode	Przejdź w stan uśpienia

gdzie:
f – 7-bitowy adres rejestru (0...127)
F – zawartość rejestru o adresie f (0...255)
W – zawartość rejestru roboczego (0...255)
d – adres wyniku operacji (0,1):
gdy $d = 0$, wynik operacji jest przesyłany do W
gdy $d = 1$, wynik operacji jest przesyłany do F
b – numer bitu (0...7)
k – 8-bitowa stała liczbowa (0...255)
s – 11-bitowa etykieta (0...2047)

Przykładowy program

```
#include "12f629.h"

int nwd(int a, int b) {
  while (a!=b) {
    if (a>b) a=a-b;
    else b=b-a;
  }
  return a;
}


void main() {
  int x,y,z;

  x=24;
  y=30;
  z=nwd(x,y);
}
```

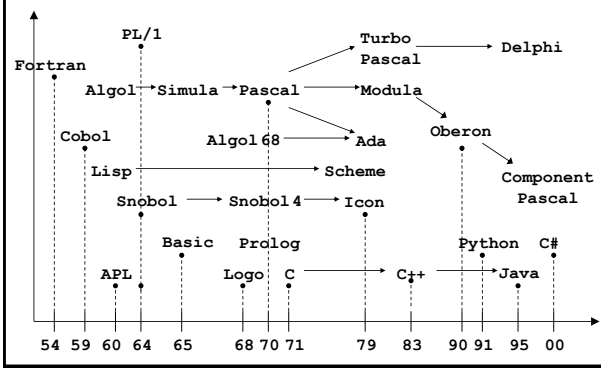
Kompilacja

```
Carry   EQU  0
Zero_   EQU  2
a       EQU  0x23
b       EQU  0x24
x       EQU  0x20
y       EQU  0x21
z       EQU  0x22

                        ;int nwd(int a, int b)
nwd
                        ;  while (a!=b) {
m001    MOVF  a,W
        XORWF b,W
        BTFSC 0x03,Zero_
        GOTO  m003
                        ;    if (a>b) a=a-b;
        MOVF  a,W
        SUBWF b,W
        MOVF  a,W
        XORWF b,W
        BTFSC 0x03,Carry
        XORLW .128
        ANDLW .128
        BTFSS 0x03,Zero_
        GOTO  m002
        MOVF  b,W
        SUBWF a,1
                        ;    else b=b-a;
        GOTO  m001
m002    MOVF  a,W
        SUBWF b,1
                        ;  }
        GOTO  m001
                        ;  return a;
m003    MOVF  a,W
        RETURN
                        ;}
```

```
                        ;void main()
main
                        ;  x=24;
        MOVLW .24
        MOVWF x
                        ;  y=30;
        MOVLW .30
        MOVWF y
                        ;  z=nwd(x,y);
        MOVF  x,W
        MOVWF a
        MOVF  y,W
        MOVWF b
        CALL  nwd
        MOVF  a,W
        MOVWF z
                        ;}

        END
```

```
:1000000016282308240603191428230824022308 89
:1001000024060318803A8039031D11282408A302FE
:1002000012823084402012823080800183040002
:100030001E30A1002008A3021084000120230 8ED
```

Historia języków programowania



Popularność języków programowania

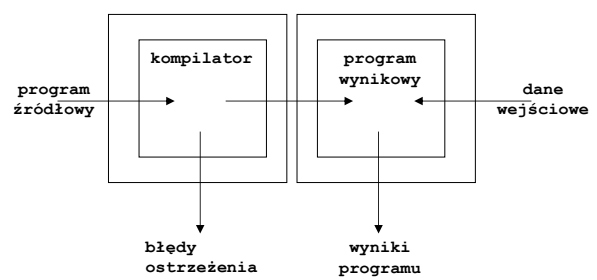
Sep 2016	Sep 2015	Change	Programming Language	Ratings	Change
1	1		Java	18.236%	-1.33%
2	2		C	10.955%	-4.67%
3	3		C++	6.657%	-0.13%
4	4		C#	5.493%	+0.58%
5	5		Python	4.302%	+0.64%
6	7	^	JavaScript	2.929%	+0.59%
7	6	v	PHP	2.847%	+0.32%
8	11	^	Assembly language	2.417%	+0.61%
9	8	v	Visual Basic .NET	2.343%	+0.28%
10	9	v	Perl	2.333%	+0.43%
11	13	^	Delphi/Object Pascal	2.169%	+0.42%
12	12		Ruby	1.965%	+0.18%
13	16	^	Swift	1.930%	+0.74%
14	10	vv	Objective-C	1.849%	+0.03%
15	17	^	MATLAB	1.626%	+0.65%
16	34	^^	Groovy	1.818%	+1.31%
17	14	v	Visual Basic	1.761%	+0.23%
18	19	^	R	1.684%	+0.64%
19	44	^^	Go	1.625%	+1.37%
20	18	v	PL/SQL	1.443%	+0.36%

www.tiobe.com/tpci.htm

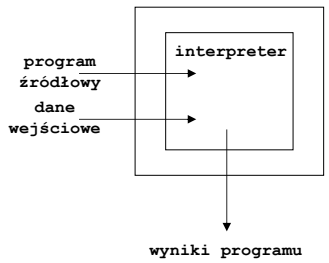
Rozwój technologii programowania

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

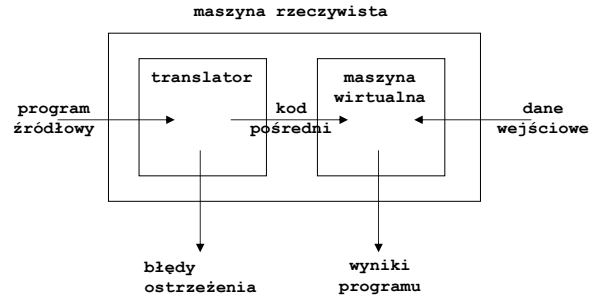
Kompilacja i wykonanie programu



Interpretacja programu



Translacja do kodu pośredniego

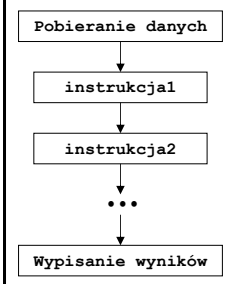


Języki programowania ogólnego zastosowania

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

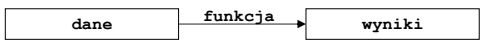
Problem:
Trójka Pitagorejska to 3 liczby naturalne spełniające równanie $a^2+b^2=c^2$.
Znaleźć wszystkie trójki, w których c nie przekracza ustalonego N.

Język imperatywny (instrukcyjny)



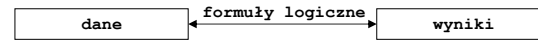

```
Program TrojkiPitagorejskie;
const
   N=10;
var
   a,b,c:integer;
begin
   for c:=1 to N do
      for b:=1 to c-1 do
      begin
         a:=trunc(sqrt(c*c-b*b));
         if a*a+b*b=c*c then
         writeln(a,b,c);
      end;
end.
```

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 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.

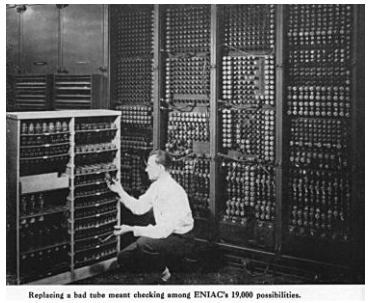
Zasoby sprzętowe:
- czas procesora (-ów)
- pamięć operacyjna
- urządzenia we/wy
- inne komputery

Zasoby programowe:
- funkcje systemowe dostarczane programowi użytkownika
- określone obszary pamięci - 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)

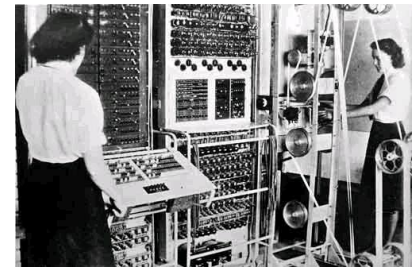
Pierwsze komputery



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

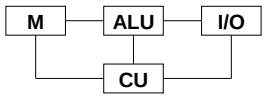
ENIAC komputer skonstruowany w latach 1943-45

Pierwsze komputery

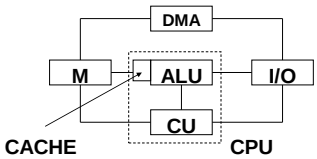


COLOSSUS - został zbudowany w 1941 r. w brytyjskim ośrodku kryptograficznym Bletchley Park.

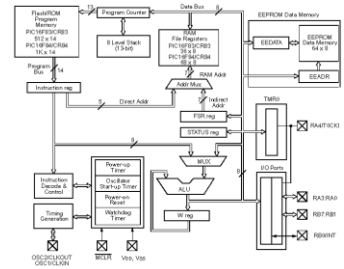
Architektura von Neumanna



Cechy:
- Rozkazy i dane przechowywane w tej samej pamięci
- Nie da się rozróżnić danych i rozkazów



Architektura Harwardzka



Cechy:
- Rozkazy i dane w oddzielnych pamięciach
- Rozkaz i dane mogą być różnej długości

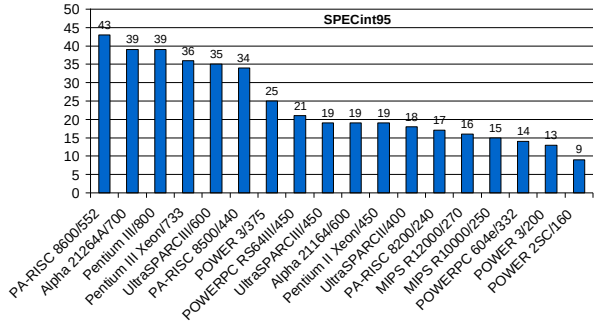
Miara wydajności - MIPS

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

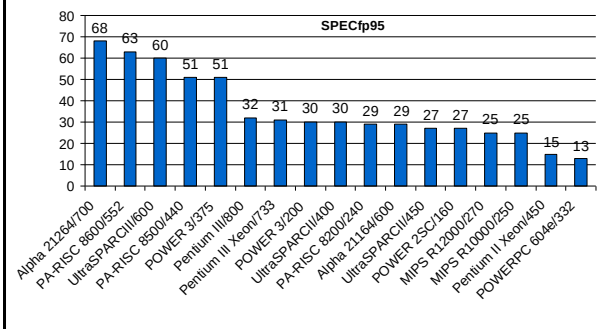
Miara wydajności - MFlops

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

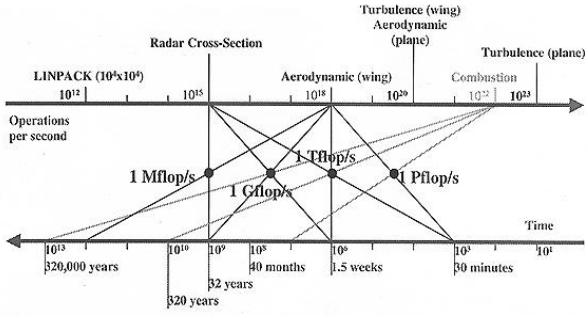
Stałoprzecinkowa wydajność procesora



Zmiennoprzecinkowa wydajność procesora



Potrzeba mocy obliczeniowej



TOP 10 Sites for June 2016

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 Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
2	National Super Computer Center in Guangzhou 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
3	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
4	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
5	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
6	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

Sunway TaihuLight

Koszty budowy Sunway TaihuLight wyniosły 273 milionów dolarów. Komputer składa się z 40960 64-bitowych procesorów RISC SW26010, każdy zawierający 260 rdzeni po 1,45 GHz. Daje to w sumie ponad 10 milionów rdzeni. Jego teoretyczna moc obliczeniowa wynosi 125 PFLOPS, a zmierzona testem LINPACK 93 PFLOPS. Sunway TaihuLight posiada 1,31 PB pamięci operacyjnej i wymaga do zasilania 15,4 MW mocy

TITAN computer

K computer

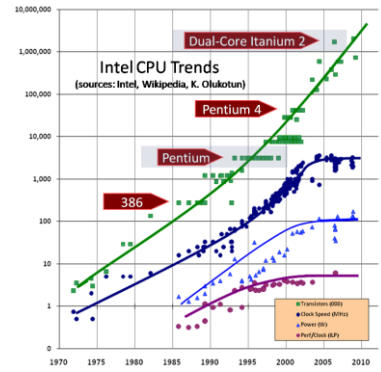
Prometheus

system operacyjny: Linux CentOS 7
konfiguracja: HP Apollo 8000
procesory: Intel Xeon (Haswell)
liczba dostępnych rdzeni : 53568
pamięć operacyjna: 279 TB
pamięć dyskowa: 10 PB
moc obliczeniowa: 2399 TFlops

Ważne 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	UN1VAC 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; pojecie "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 "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

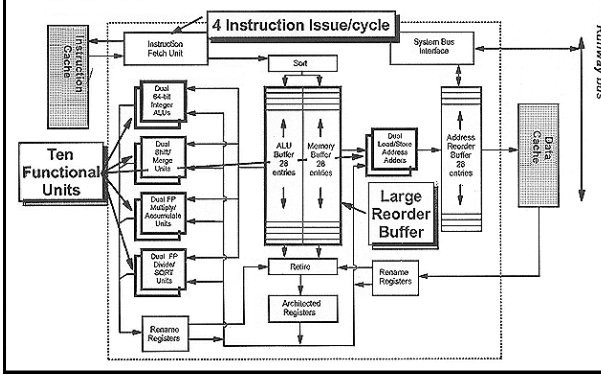


Porównanie mikroprocesorów

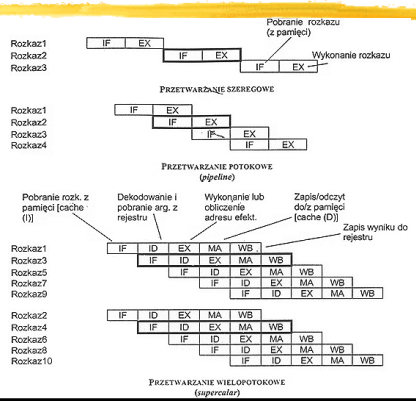
Procesor (producent)	Rozkazów/cykl	Cache I, D (KB)	Liczba wyprowadzeń	Zegar (MHz)	SPECint95	SPECfp95	Liczba tranzystorów (mln)
21164 Alpha (DEC)	4	16+96	499	500	15,4	21,1	9,3
PowerPC 604e (IBM/Motorola)	4	32	255	225	9,0	7.5	5.1
Pentium Pro (Intel)	3	16 + (256)	387	200	8,2	6,7	5,5
Pentium II (Intel)	?	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.

Architektura procesora PA-8000



Sposoby przetwarzania strumienia rozkazów



Wykład 14

- Złożoność obliczeniowa
- Notacja O( )
- Przykłady problemów
- Problemy nierozwiązywalne

Złożoność obliczeniowa algorytmu

Definiuje się ją jako:
- ilość zasobów komputerowych potrzebnych do jego wykonania (czas procesora, wielkość pamięci).

Wyróżnia się:
- złożoność pesymistyczną (ilość zasobów potrzebnych przy „najgorszych" danych wejściowych o rozmiarze *n*),
- złożoność oczekiwaną (ilość zasobów potrzebnych przy „typowych" danych wejściowych o rozmiarze *n*),

Funkcja złożoności obliczeniowej

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

Rząd złożoności obliczeniowej

Funkcja f(k) jest rzędu g(k), co zapisujemy O(g(k)), jeżeli istnieje taka stała c, że f(k)<=c g(k) dla prawie wszystkich wartości k.

Przykłady:

O(logN)	logarytmiczna
O(N)	liniowa
O(N*logN)	liniowo-logarytmiczna
$O(N^2)$	kwadratowa (wielomianowa)
$O(2^N)$	wykładnicza
O(N!)	silnia

Problem stałej:
f1(N)=10*N
f2(N)=1000*logN

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.

Złożoność czasowa a czas wykonania

Funkcja złożoności \ Rozmiar problemu n	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

Złożoność algorytmów

Problem wyszukiwania elementu w tablicy
- Wyszukiwanie liniowe O(N)
- Wyszukiwanie binarne O(logN)

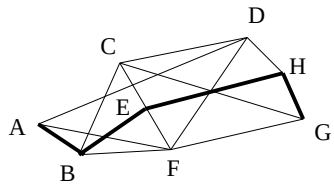
N	liniowy	binarny
10	10	4
10^3	10^3	10
10^6	10^6	20

Złożoność algorytmów

Problem sortowania tablicy
- Metody proste O(N^2)
- Metody szybkie O(N*logN)

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

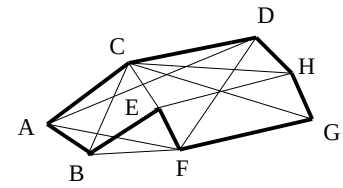
Problem najkrótszej drogi



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 ~(a&b) → (a|b)

a	b	~(a&b) → (a|b)
0	0	0
0	1	1
1	0	1
1	1	1

Dana jest funkcja N zmiennych logicznych:
$f(A_1,A_2,...,A_n) \rightarrow$ {true,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) a>0
f(a,b)=f(a-1,f(a,b-1)) a>0, b>0

Ile wynosi f(5,5) ?

Funkcja Ackermana-Hermesa

f(a,b)=b+1 dla a=0
f(a,b)=f(a-1,1) dla b=0
f(a,b)=f(a-1,f(a,b-1)) dla a>0 i b>0

Wartości funkcji:
f(0,b)=b+1
f(1,b)=b+2
f(2,b)=2*b+3
f(3,b)=2^{b+3}-3

f(4,b)= $2^{2^{\cdot^{\cdot^{2}}}}$ (b+3 dwójek) -3

F(5,5)= $2^{2^{\cdot^{\cdot^{2}}}}$ (2^{16} dwójek) -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.
```

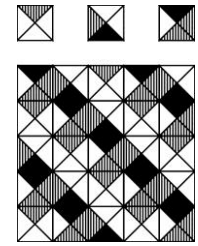
```
read(x)
while x<>1 do
begin
   if odd(x) then x:=3*x+1
             else x:=x div 2
end
```

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;
```

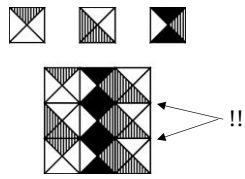
Pytanie:
Czy wywołanie procedury S(S) się zakończy?

Problem domina



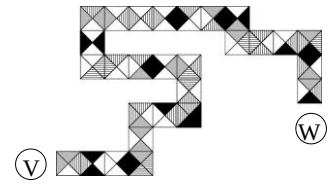
Rodzaje kafelków, którymi można pokryć każdy obszar

Problem domina



Rodzaje kafelków, którymi nie można pokryć nawet małych obszarów

Problem domina - wąż



Wąż domino łączący punkty V i W

Zadanie

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

sum_p(N)=N

gdzie:

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

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

Należy napisać program odnajdujący wszystkie takie liczby.

Dwa problemy:

- Liczby Armstronga
- Liczby pierwsze

Liczby Armstronga

Liczba Armstronga:

n-cyfrowa liczba naturalna która jest sumą swoich cyfr podniesionych do potęgi n.

Fakty:

Ilość liczb Armstronga jest skończona.

Dla liczby n-cyfrowej:

$max\_suma = n*9^n$ $min\_liczba = 10^{n-1}$

dla n=3 $n*9^n > 10^{n-1}$

dla n=100 $n*9^n < 10^{n-1}$

Rozwiązanie równiania $n*9^n = 10^{n-1}$ daje n=60.8478

Wniosek:

Nie istnieją liczby Armstronga większe niż 60 cyfrowe

Liczby pierwsze

Fakty:

Liczb pierwszych jest nieskończenie wiele. (Euklides)

Liczb pierwszych mniejszych od N jest około N/ln(N).

Wśród liczb 100 cyfrowych jedna na 300 jest pierwsza.

Sprawdzić czy duża liczba jest pierwsza jest łatwo!

Rozłożyć dużą liczbę na czynniki jest bardzo trudno!

Najszybszy znany algorytm wymaga czasu:

$T = \exp(\sqrt{\ln(n)*(\ln(\ln(n)))})$

Dla $n \cong 10^{200}$ czas wynosi $T \cong 10^{23}$ kroków

Jeżeli 1 krok = 1 mikrosekunda to czas wynosi 3 mld lat.

21

Potęga liczby

```
# oblicza a^z mod n

procedure fastexp(a,z,n)
  x:=1
  while z~=0 do {
    while z%2=0 do {
      z:=z/2
      a:=(a*a)%n
    }
    z:=z-1
    x:=(x*a)%n
  }
  return x
end
```

22

Test pierwszości liczby

```
# test Millera-Rabina, prawdopodobieństwo błędu (1/2)^n

procedure Miller(p,n)
   if p<2 then return 0
   if p~=2 & p%2=0 then return 0

   s:=p-1;
   while s%2=0 do s/:=2

   every i:=1 to n do {
      a:=random(p-1)+1
      temp:=s
      mod:=fastexp(a,temp,p)
      while temp~=p-1 & mod~=1 & mod~=p-1 do {
         mod:=(mod*mod)%p
         temp*:=2;
      }
      if mod~=p-1 & temp%2=0 then return 0
   }
   return 1
end
```

23

Liczby pierwsze Mersenne'a (2014 r.)

1. 2^2−1
2. 2^3−1
3. 2^5−1
4. 2^7−1
5. 2^13−1
6. 2^17−1
7. 2^19−1
8. 2^31−1
9. 2^61−1
10. 2^89−1
11. 2^107−1
12. 2^127−1
13. 2^521−1
14. 2^607−1
15. 2^1279−1
16. 2^2203−1
17. 2^2281−1

...

...

32. 2^756839−1
33. 2^859433−1
34. 2^1257787−1
35. 2^1398269−1
36. 2^2976221−1
37. 2^3021377−1
38. 2^6972593−1
39. 2^13466917−1
40. 2^20996011−1
41. 2^24036583−1
42. 2^25964951−1
43. 2^30402457−1
44. 2^32582657−1
45. 2^37156667−1
46. 2^42643801−1
47. 2^43112609−1
48. 2^57885161−1

24