

Programovacie paradigmy

alias *Koštofka* programovacích štýlov a jazykov

Peter Borovanský, KAI, I-18, borovan(a)ii.fmph.uniba.sk

<http://dai.fmph.uniba.sk/courses/PARA/>

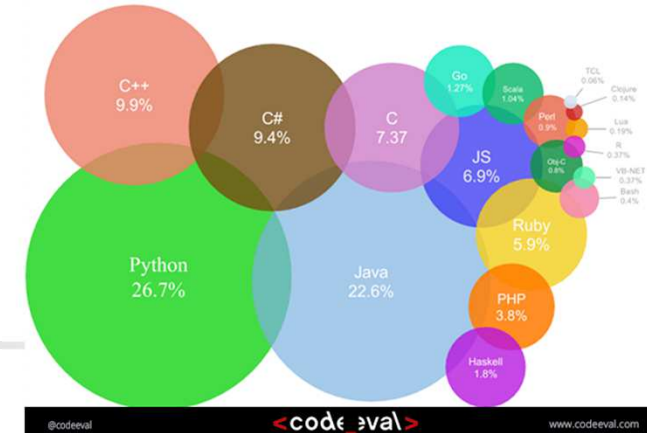
MS-Teams: prihlasovací kód je [rclngbt](#)

Pondelok, 14:50, I-9



Paradigmy 2022

(úvodné slovo)



Keď sa povie programovacie paradigmy, väčšina *klasikov* si spomenie na **funkcionálnu** a **logickú** paradigmu a **procedurálnu**, s **objektovou** odchýlkou.

- procedurálnu paradigmu poznáte z jazykov, ktoré nevznikli za vášho života, napr. v roku 1985 (C++) a 1995 (Java, JavaScript, Python). **Čo sa nič odvtedy neudialo?**
- v prvej časti kurzu sa pozrieme na konkurentnú paradigmu s jazykom Go, procedurálny programovací jazyk s podporou **konkurentného programovania**, čo je ťažká, ale dôležitá disciplína...
- v druhej časti kurzu sa pozrieme na funkcionálnu paradigmu, každý z dnešných moderných programovacích jazykov v nejakej miere podporuje **funkcionálne programovanie**
- v tretej časti kurzu sa pozrieme na logickú paradigmu, **logické programovanie** s ohraničeniami (constraint logic programming)

~~@Deprecated~~

Java 10 -release 3.2018

príklad, čo dobre poznáte



- Java je *zabetónovaná* od Java 1.0, 1995, ale snaží sa implementovať novinky nespútaných nových jazykov, ktoré nepotrebujú byť spätne kompatibilné.

@Deprecated

- v ktorom z dnešných moderných programovacích jazykov sa dnes ešte píše povinne bodkočiarka (za príkazom) ?
- je to len dôsledok toho, že pred releasom Java 1.0 sa zdalo mnoho iných vážnejších problémov, ktoré riešiť, že sa J.Gosling neobťažoval domyslieť gramatiku jazyka (resp. syntaktický analyzátor), aby bodkočiarky neboli treba.
- ... a odvtedy povinne píšme ;cr,lf podľa vzoru jazyka Algol (1960)/C (1985).
- ... a vôbec to nie je o bodkočiarke...

Cieľom je, aby ste pochopili, koľko rôznych 'bodkočiarok' v jazyku Java existuje najmä kvôli spätnej kompatibilite.

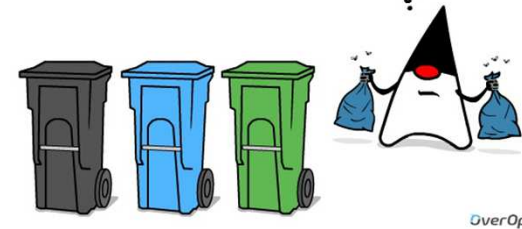


Java 17

long-term-support (LTS) release 14.9.2021

<http://www.java-countdown.xyz/>

- Je Java mŕtva ? Asi nie, na jar vyšla Java 16 nedávno 17. (rýchla odpoveď...)
- Ale neumiera ? (... to už je iná otázka ...)
- Pozreli ste si novinky, ktoré Java 12/13/14/15/16/17 prináša ?
- Viete napísať program v Java 11, ktorý nie je v jazyku Java 10, Java 9 ?
- V histórii Javy ako iného jazyka nastali výraznejšie dve revolúcie:
 - 2004 Java 5 prišla s generickými typmi (polymorfizmus),
 - 2014 Java 8 prišla so streamami a lambdami, resp. s funkciami – paradigm shift
- ale fakticky najväčší prínos Java 8 bol asi Stream API... (ste videli v [Prog4](#))





Paradigm shift - StreamAPI

Java old-school (before Java8)

```
static List<String> preber(List<String> lst,
                          String word) {
    List<String> result = new ArrayList<>();
    for (String elem : lst)
        if (elem.contains(word))
            result.add(elem.toUpperCase());
    return result;
}
```

Príklad imperatívneho kódu, kde popisujeme presne postup, ako sa zo vstupu dostaneme k výstupu

Java new-wave (after Java8)

```
static List<String> preberStreamAPI(List<String> lst,
                                     String word) {
    return lst
        .stream()
        .parallelstream()
        .filter(elem -> elem.contains(word))
        .map(String::toUpperCase)
        .collect(Collectors.toList());
}
```

Ďaleko menej sa staráme o to, ako sa operácia prevedie, skôr vyjadrujeme, čo chceme dostať. De-facto, nič nám nebráni operáciu vykonávať paralelne, aj keď nevieme o implementačných detailoch, čo to presne urobí, v akom poradí. Kto vie, ako sa implementuje parallelstream ?

Paradigm shift – Konkurentnost'

```
static int globalVar = 0;  
static Thread smallThread(String label, int delta) {  
    return
```

```
        new Thread(new Runnable() {                // OLD SCHOOL  
            @Override  
            public void run() {  
                System.out.println(label + (globalVar += delta));  
            }  
        });
```

```
        new Thread(                                // NEW STYLE  
            () -> System.out.println(label + (globalVar += delta))  
        );
```

```
    }  
    public static void main(String[] args) throws InterruptedException {  
        Thread t1 = smallThread("prvy", 10); t1.start();  
        Thread t2 = smallThread("druhy", 20); t2.start();  
        t1.join(); t2.join();  
        System.out.println(globalVar);  
    }
```

```
druhy20  
prvy30  
30
```

```
prvy10  
druhy30  
30
```

```
druhy30  
prvy10  
30
```

```
prvy30  
druhy20  
30
```

```
druhy20  
prvy10  
20
```



Konkurencia vs. Paralelizmus

Otázku, ktorú si môžete/máte položiť po Prog4:

*sú Thready a **synchronized**, ... jediné formy či modely konkurencie (v Jave) ?*

Btw, java.lang.Thread existuje od JDK-1

Aké iné modely konkurencie nájdete v JDK (úmyselne nechané do prémie) ?

- ?
- ??
- ???

Aké iné modely nájdete napr. v Kotle (pre tých čo bežia VMA) ?

- coroutines



Paradigm shift – Clojures/Lambdas

```
static void compare1(String[] arr) {  
    Arrays.sort(arr, new Comparator<String>() {  
        @Override  
        public int compare(String o1, String o2) {  
            return o1.toUpperCase().compareTo(o2.toUpperCase());  
        }  
    });  
    for(String elem : arr) System.out.println(elem);  
}
```

```
static void compare2(String[] arr) {  
    Stream.of(arr)  
        .sorted((o1, o2) -> o1.toUpperCase().compareTo(o2.toUpperCase()))  
        .sorted((o1, o2) -> o1.compareToIgnoreCase(o2))  
        .sorted(String::compareToIgnoreCase)  
        .forEach(System.out::println);  
}
```

```
String[] test = {"zelen", "GULA", "Srdce", "zALUD"}; compare1(test);  
System.out.println(List.of(test));
```

```
String[] test = {"zelen", "GULA", "Srdce", "zALUD"}; compare2(test);  
System.out.println(List.of(test));
```

[GULA, Srdce, zALUD, zelen]

[zelen, GULA, Srdce, zALUD]



Java 17 - features

- sealed interface/class (zapečatená trieda)
- switch pattern

sealed interface S **permits** A, B, C {}

final class A implements S {}

final class B implements S {}

record C(int i) implements S {} // Implicitly final

```
static int testSealedCoverage(S s) {
```

```
    return switch (s) {
```

```
        case A a -> 1;
```

```
        case B b -> 2;
```

```
        case C c -> 3;
```

```
    };
```

```
}
```

What's New in Java 19: The end of Kotlin (dec 2019)

<https://www.youtube.com/watch?v=te3OU9fxC8U>



KotlinConf'19

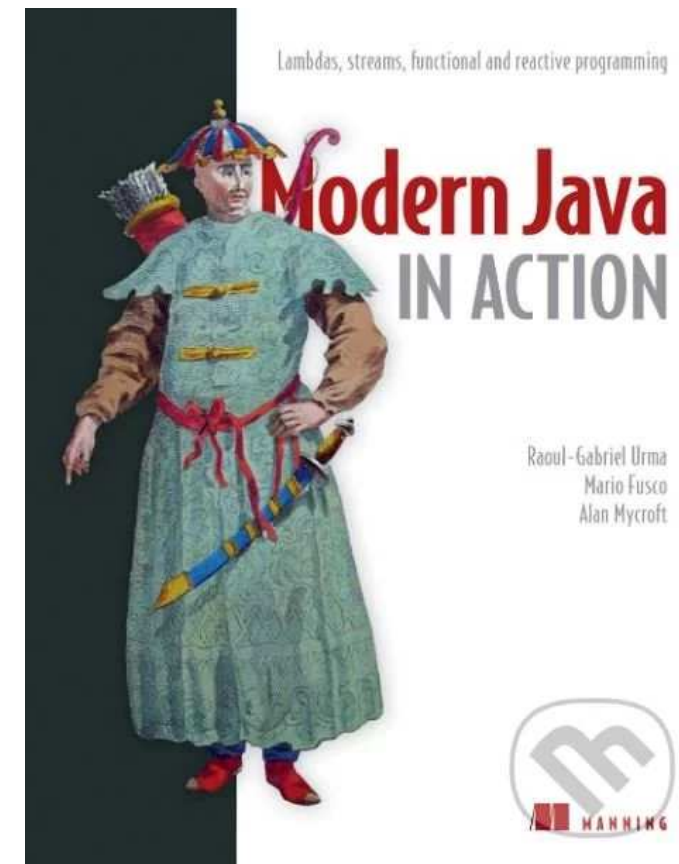
What's new in Java 19: The end of Kotlin?

Jake Wharton



Čo by som si mohol prečítať po Programovanie 4

- **Modern Java in Action : Lambdas, streams, functional and reactive programming**





Paradigmy 2022

(úvodné slovo)

V rámci kurzu sa stretnete s:

- konkuretným jazykom GO,
- funkcionálnym jazykom Haskell,
- logickým programovacím jazykom Prolog.

Prednáška dáva intro (ochutnávku) rôznych paradigiem, spôsobov rozmýšľania

- nadväzujúce magisterské predmety idú hlbšie do princípov tej-ktorej paradigmy
- očakáva sa schopnosť dohľadať si detaily potrebné k DÚ
- a očakáva to dnes každý z vašich potenciálnych zamestnávateľov...

Ak vám tento prístup nevyhovuje, lepšie si zvoliť iný PV predmet...

O čom to bude ?

(Paradigmy 2022)

Programming language features:

- static/dynamic typing
- type inference
- duck typing
- goroutines
- block chain
- coroutines
- tail recursion optimisation
- function as value
- lazy evaluation
- list comprehension
- pattern-matching
- backtracking and lot of recursion
- Horn clause, (SLD) resolution, choice point, unification
- constraint logic programming



Hlavné paradigmy

- objektovo-orientovaná paradigma
- procedurálna paradigma
- funkcionálna paradigma
- logická (relačná) paradigma
- konkurentná paradigma

Paradigmy nie sú disjunktné množiny
počas prednášky sa pozrieme na tri z nich v najčistejšej forme:

CP – Go,

FP – Haskell,

LP – Prolog.



Konkrétnejšie ?

(toto je plán)

- História programovacích jazykov, 1x
- Go – procedurálna/konkurentná paradigma, cca 4x - október
 - konkurentné programovanie – go-routines
 - static typing
 - duck typing
- Haskell – funkcionálna paradigma, cca 4x - november
 - type inference
 - function as value
 - lazy evaluation
 - list comprehension
 - backtracking and recursion
 - pattern-matching
- Prolog – logická paradigma, cca 3x - december
 - Horn clause
 - resolution
 - choice point
 - unification



Jazyky rokov

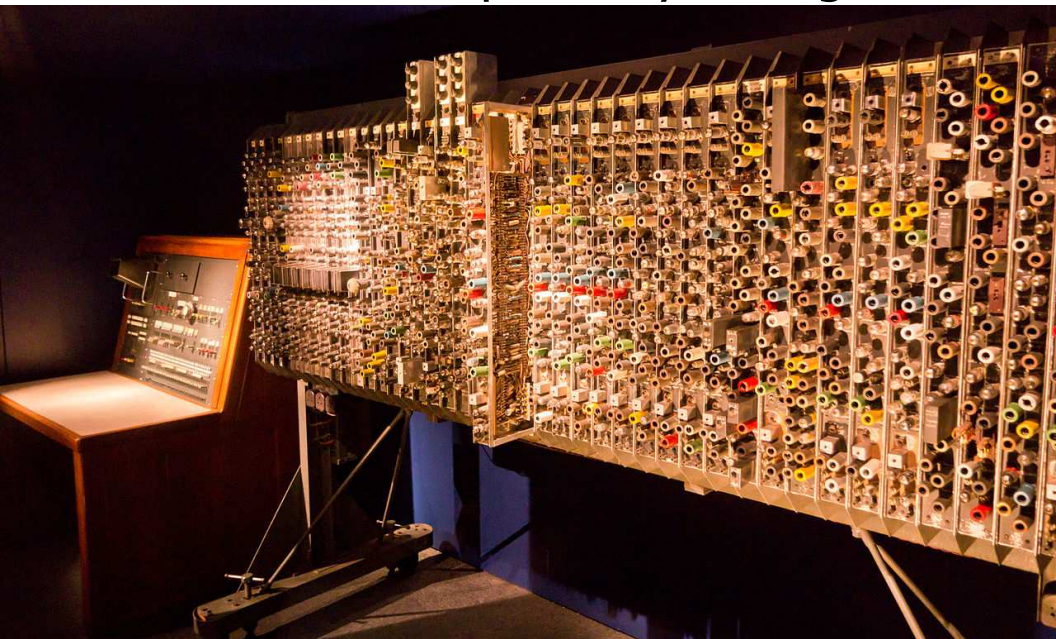
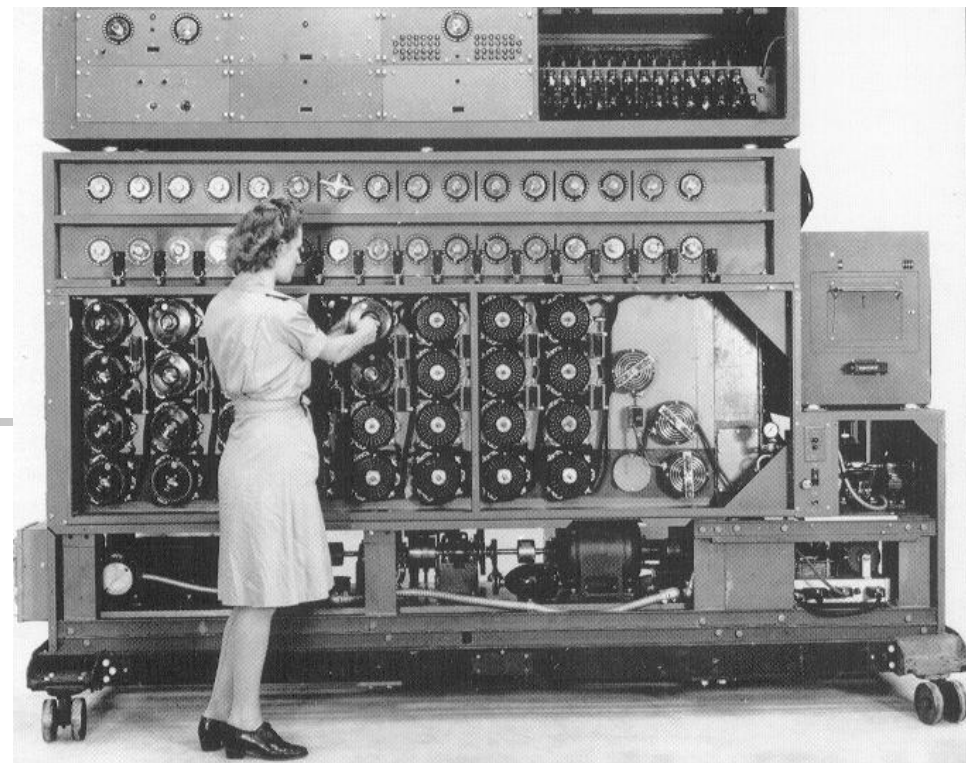
(1945-55-65-75-85-95-05-15)

- 1938 – binary code, The Bombe
- 1946 – Base 32, Ace

#computers = $O(1)$

#programmers = $O(1)$ - Alan Turing

We shall need a **great number** of **mathematicians of ability** because there will probably be a good deal of work of this kind to be done.



One of our difficulties will be the maintenance of an appropriate (i.e. discipline), so that we do not lose track of what we are doing

Jazyky rokov

(1945-55-65-75-85-95-05-15)

- **1953 Fortran, Lisp – Functional Programming**
- IBM

#computers = $O(10^2)$

#programmers = $O(10^3)$

- **Inžinieri,**
- **Matematici,**
- **Fyzici,**

...

ľudia, ktorí majú:

- svoju materskú profesiu,
- vybudované pracovné návyky,
- vedia už komunikovať



Jazyky rokov

(1945-55-**65**-75-85-95-05-15)



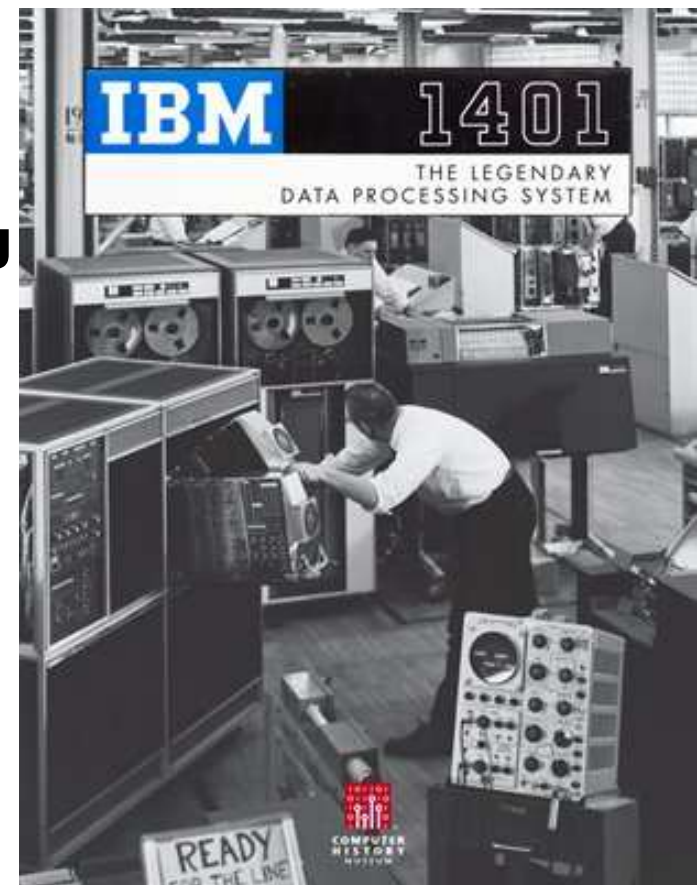
- **1960 Algol**
- **1960 COBOL, PL/1**
- **1962 APL**
- **1967 Simula - OOP**
- **1968 Pascal**
- **1968 No-Goto – Structured Programming**
- **1968 C**

- IBM 1401, IBM 360

#computers = $O(10^4)$

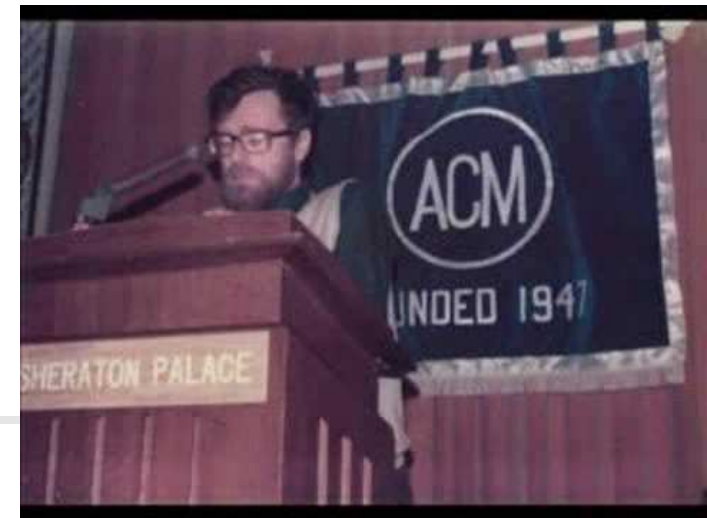
#programmers = $O(10^5)$

Ukazuje sa prvý vážny nedostatok ľudí schopných ovládať počítače, tak vznikajú prvé univerzitné programy Computer Science...



GOTO story

(štruktúrované programovanie)



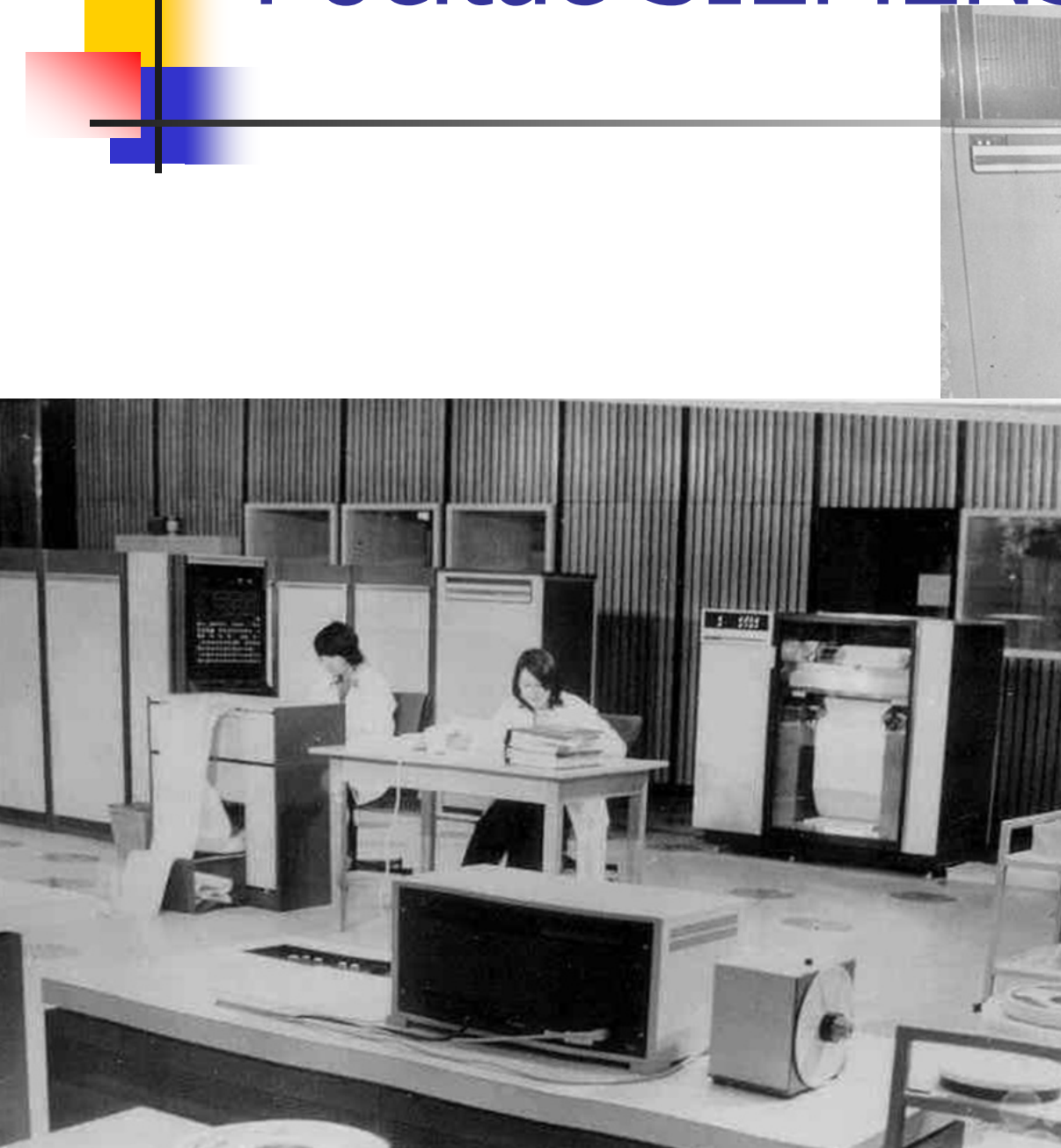
A Case against the GO TO Statement.

by Edsger W. Dijkstra
Technological University
Eindhoven, The Netherlands

Since a number of years I am familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce. Later I discovered why the use of the go to statement has such disastrous effects and did I become convinced that the go to statement should be abolished from all "higher level" programming languages (i.e. everything except -perhaps- plain machine code).

Edsger Dijkstra (1968). "Go To Statement Considered Harmful", *Communications of the ACM*
Frank Rubin (1987). "GOTO Considered Harmful" Considered Harmful, *Communications of the ACM*
Donald Moore; ... (1987). "'GOTO Considered Harmful' Considered Harmful" Considered Harmful?,
Dijkstra, Edsger On a Somewhat Disappointing Correspondence (EWD-1009)

Počítač SIEMENS 4004 ÚVTVŠ



v Mlynskej doline
H3/H6

APL language

<https://youtu.be/DTpQ4Kk2wA?t=612>



APL

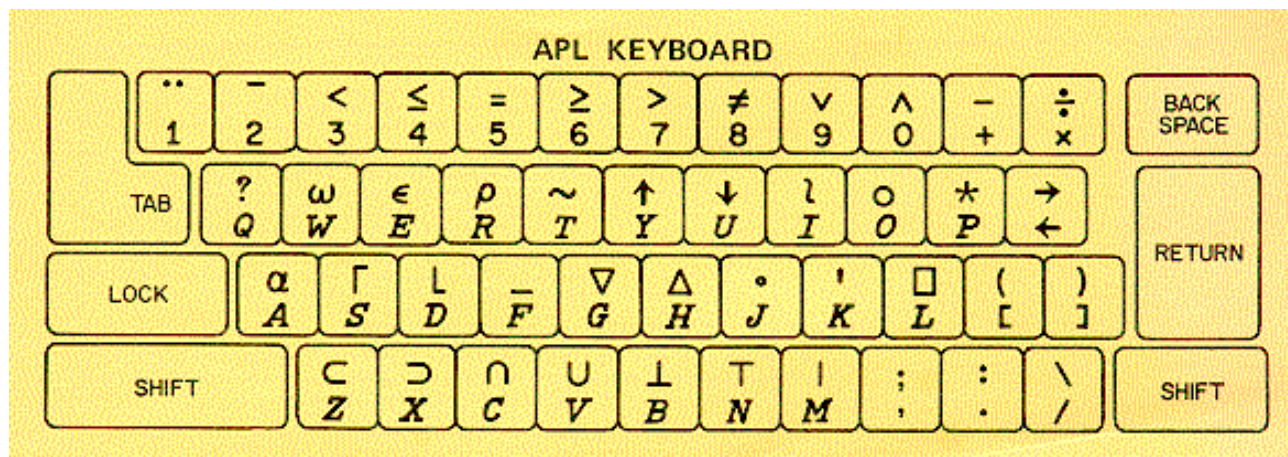
APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums - Edsger W.Dijkstra

- 1957 Kenneth E. Iverson, Turingova cena
- interaktívny interpreter (v čase sálových počítačov)
- priradenie vektora hodnôt 4 5 6 7 to N.
- pričítanie 4 k vektoru (dostaneme 8 9 10 11)
- tlač súčtu N, t.j. 22.

$N \leftarrow 4\ 5\ 6\ 7$

$N + 4$

$+ / N$





APL

"APL, in which you can write a program to simulate shuffling a deck of cards and then dealing them out to several players in four characters, none of which appear on a standard keyboard." David Given

52?52

$$(\sim R \in R \circ . \times R) / R \leftarrow 1 \downarrow iR$$

Here's how to read it, from right to left:

1. i creates a vector containing integers from 1 to R (if $R = 6$ at the beginning of the program, iR is 1 2 3 4 5 6)
2. Drop first element of this vector (\downarrow function). So, $1 \downarrow iR$ is 2 3 4 5 6
3. Set R to the vector (\leftarrow , assignment primitive), i.e. $R = 2\ 3\ 4\ 5\ 6$
4. Generate outer product of R multiplied by R , i.e. a matrix which is the *multiplication table* of R by R ($\circ.\times$ function)
5. Build a vector the same length as R with 1 in each place where the corresponding number in R is in the outer product matrix (\in , set inclusion function), i.e. 0 0 1 0 1
6. Logically negate the values in the vector (change zeros to ones and ones to zeros) (\sim , negation function), i.e. 1 1 0 1 0
7. Select the items in R for which the corresponding element is 1 ($/$), i.e. 2 3 5

Apollo 11

- júl 1969, 2kB+32kB ROM, 1MHz
- 1980, prvé IBM XT 16kB, 4MHz
- OS: multitasking pre 8 taskov
- I/O: signály, senzory, snímače vo frekvencii Hz (raz z sekundu ;-)
- Interface: skôr ako blikajúci switch
- Errors: 1201, 1202 - príliš veľa dát – radšej pristal N.Armstrong bez počítača
- Updates: treba prekódovať EPROM-ku

Dnes:

- používame QuadCore (2.23 GHz, 32..256GB+3GB RAM) na telefonovanie, ...
- často nevieme naprogramovať komunikáciu s odozvou, napr. cez Bluetooth,
- každé ráno nájdeme v mobile niekoľko updatov našich apps
- málo kto (už dnes) vie napísať aplikáciu do 16kB

$$2^{(49/2)} = \sim \text{milión} = 16\text{M}$$
$$2\text{kB} * 16\text{M} = 32\text{GB}$$

$$2^{(39/2)} = 1\text{M}$$
$$16\text{kB} * 1\text{M} = 16\text{GB}$$



Jazyky rokov

(1945-55-65-75-85-95-05-15)

- 1972 Prolog – Logické programovanie
- 1972 SQL – Relačné programovanie
- 1973 C

**Počet programátorov
sa zdvojnásobí každých 5 rokov**

**Takže každý druhý má < 5
rokov programátorskej praxe !!!**

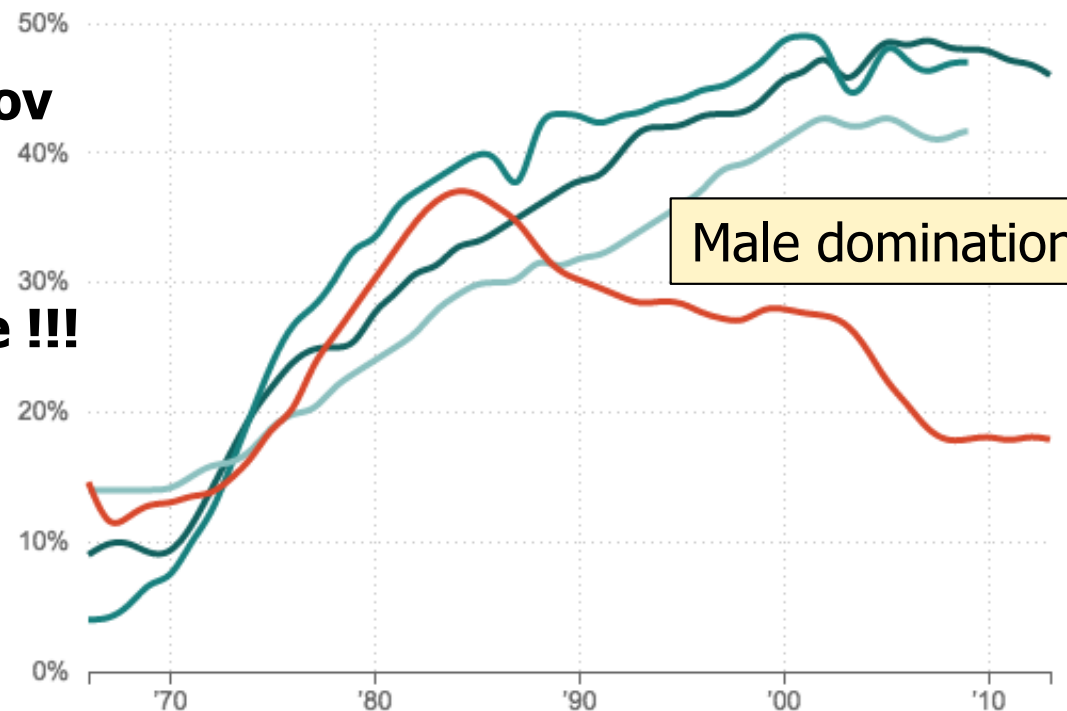
#computers = $O(10^6)$

#programmers = $O(10^7)$

What Happened To Women In Computer Science?

% Of Women Majors, By Field

Medical School Law School Physical Sciences Computer science





Jazyky rokov

(1945-55-65-75-**85**-95-05-15)

- **1980 C++**
- **1983 Ada**
- **1985 Eiffel**
- **1986 Objective-C**
- **1987 Perl**

Araine 5

1995



```
L_M_BV_32 := TBD.T_ENTIER_32S ((1.0/C_M_LSB_BV) * G_M_INFO_DERIVE(T_ALG.E_BV));
if L_M_BV_32 > 32767 then
    P_M_DERIVE(T_ALG.E_BV) := 16#7FFF#;
elsif L_M_BV_32 < -32768 then
    P_M_DERIVE(T_ALG.E_BV) := 16#8000#;
else
    P_M_DERIVE(T_ALG.E_BV) := UC_16S_EN_16NS(TDB.T_ENTIER_16S(L_M_BV_32));
end if;
P_M_DERIVE(T_ALG.E_BH) :=
    UC_16S_EN_16NS (TDB.T_ENTIER_16S ((1.0/C_M_LSB_BH) *
    G_M_INFO_DERIVE(T_ALG.E_BH)));
```



Jazyky rokov

(1945-55-65-75-85-**95**-05-15)

Internet !

- **1990 Haskell**
- **1991 Python**
- **1991 VB**
- **1995 Java**
- **1995 JavaScript**

Agile Manifesto

(2001)



Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.



Clean Code Blog

- <https://blog.cleancoder.com/>

- [Why Clojure?](#)

08-22-2019

- [Why won't it...](#)

07-22-2019

- [Classes vs. Data Structures](#)

06-16-2019

- [Types and Tests](#)

06-08-2019

- [737 Max 8](#)

05-18-2019

- [FP vs. OO List Processing](#)

12-17-2018





Clojure (ClojureScript)



- v IntelliJ nájdete Cursive plugin
- ```
(defn square [x] (* x x))

(println
 (take 25
 (map square (range))))
```
- Reinkarnácia Lispu z roku 1960
- generuje kód do JVM alebo do .JS

Bob Martin: Why Clojure

<http://blog.cleancoder.com/uncle-bob/2019/08/22/WhyClojure.html>



# Čo prinieslo Funkcionálne prog.

(len dve myšlienky)

- Garbage collection – mal už prvý LISP 1960
- Tail Recursion Optimisation (TRO)
- chvostová rekurzia
- *malý Gauss riešenie  $[n*(n+1)]/2$*



```
// rekurzia
fun recursiveSum(n: Long) : Long {
 return if (n <= 1) {
 n
 } else {
 n + recursiveSum(n - 1)
 }
}
fun main() {
 println(recursiveSum(100_000L))
}
Stack overflow...
```

Kód je v jazyku Kotlin, viete to čítať ?

```
// chvostová rekurzia/iterácia/cyklus
tailrec fun Sum(n: Long,
 accum: Long = 0
): Long {
 return if (n <= 0) {
 accum
 } else {
 Sum(n - 1, n + accum)
 }
}
fun main() {
 println(Sum(1_000_000_000L))
} // výsledok: 500000000500000000
```



# Trail Recursion Optimisation

(Show in ByteCode)

```
public final static recursiveSum(J)J
```

```
L0 LINENUMBER 3 L0
```

```
LLOAD 0
```

```
LCONST_1
```

```
LCMP
```

```
IFGT L1
```

```
L2 LINENUMBER 4 L2
```

```
LLOAD 0
```

```
GOTO L3
```

```
L1 LINENUMBER 6 L1
```

```
FRAME SAME
```

```
LLOAD 0
```

```
LLOAD 0
```

```
LCONST_1
```

```
LSUB
```

```
INVOKESTATIC recursiveSum (J)J
```

```
LADD
```

```
L3 LINENUMBER 3 L3
```

```
FRAME SAME1 J
```

```
LRETURN L4
```

```
LOCALVARIABLE n J L0 L4 0
```

```
MAXSTACK = 6
```

```
MAXLOCALS = 2
```

```
public final static Sum(JJ)J
```

```
L0 LINENUMBER 11 L0
```

```
FRAME SAME
```

```
LLOAD 0
```

```
LCONST_0
```

```
LCMP
```

```
IFGT L1
```

```
L2 LINENUMBER 12 L2
```

```
LLOAD 2
```

```
GOTO L3
```

```
L1 LINENUMBER 14 L1
```

```
FRAME SAME
```

```
LLOAD 0
```

```
LCONST_1
```

```
LSUB
```

```
LLOAD 0
```

```
LLOAD 2
```

```
LADD
```

```
LSTORE 2
```

```
LSTORE 0
```

```
GOTO L0
```

```
L3 LINENUMBER 11 L3
```

```
FRAME SAME1 J
```



# Jazyky rokov

(1945-55-65-75-85-95-**05**-15)

---

- **2001 C#**
- **2003 Scala**
- **2007 Clojure**
- **2009 Go**




# Jazyky rokov

(1945-55-65-75-85-95-05-**15**)

---

- **2011 Dart**
- **2011 Kotlin**
- **2014 Swift**

# Historia programovacích jazykov

- 
- 1943 - ENIAC coding system
  - 1951 - Assembly Language
  - **1954 - FORTRAN** (J.Backus,IBM)
  - **1958 - LISP** (J.McCarthy)
  - 1958 - ALGOL (Backus-Naur)
  - 1959 - COBOL
  - 1962 - APL
  - 1962 - Simula (J.Dahl)
  - 1964 - BASIC
  - 1964 - PL/I
  - 1970 - Pascal (N.Wirth)
  - 1972 - C (D.Ritchie)
  - **1972 - Smalltalk** (A.Kay,Xerox)
  - **1972 - Prolog** (A.Colmenauer)
  - 1973 - ML
  - 1978 - SQL (T.Codd)
  - 1983 - Ada
  - 1983 - C++ (B.Stroustrup)
  - 1985 - Eiffel (B.Mayer)
  - 1987 - Perl
  - **1990 - Haskell**
  - **1990 - Python**
  - 1991 - Java (Sun)
  - 2000 - C#
  - 2006 - **Scala** (Martin Odersky)
  - 2007 - **Go**
  - 2014 - **Swift**
  - 2016 - **Kotlin**

*Computer Science without  
FORTRAN and COBOL is  
like birthday cake without  
ketchup and mustard.*



## 50-60 -te roky

---

- 1954 - **FORTRAN** (J.Backus,IBM)
  - vedecko-technické výpočty, numerické výpočty
- *1958 - LISP (J.McCarthy)*
- 1958 - **ALGOL** (Backus-Naur)
  - algoritmický jazyk, štruktúrované programy, riadiace štrukt.
- 1959 - **COBOL** (Pentagon)
  - biznis, financie
- *1962 - APL (Kenneth E. Iverson, Harvard)*
  - vektorovo orientovaný jazyk

# Fortran

Technické možnosti hardware ovplyvňujú  
programovacie jazyky a tým aj paradigmu

C It was the first programming language

C with the comments support!

```
WRITE (6,7)
```

```
7 FORMAT(15H Hello, world!)
```

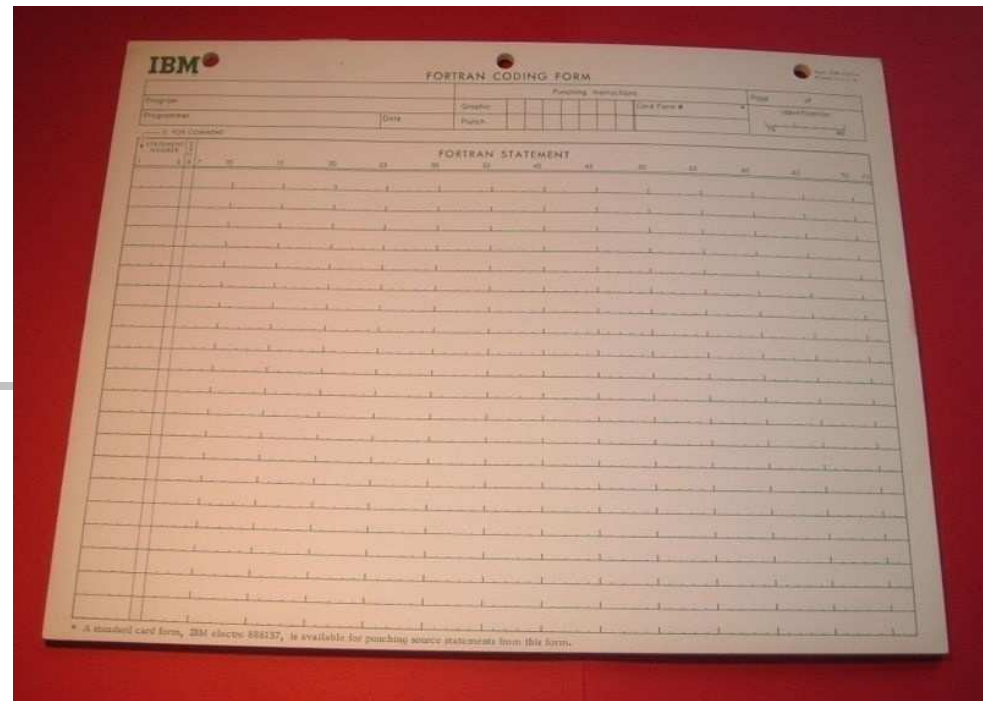
```
STOP
```

```
END
```

```
WRITE(*,17) A, B, C
```

```
17 FORMAT(F6.3, F6.3, F10.5)
```

```
printf("%f6.3%f6.3%f10.5", A, B, C);
```





# Fortran

*"Consistently separating words by spaces became a general custom about the tenth century A.D., and lasted until about 1957, when FORTRAN abandoned the practice." —Sun FORTRAN Reference Manual*

## Cyklus, ktorý sčíta nepárne čísla

IF (X - Y) 100, 200, 300

if (x - y) < 0 then goto 100  
if (x - y) = 0 then goto 200  
if (x - y) > 0 then goto 300

if (x .lt. y) then goto 100  
if (x .eq. y) then goto 200  
if (x .gt. y) then goto 300

integer i, n, sum  
sum = 0

do 10 i = 1, n, 2  
sum = sum + i  
continue

10

do10i=1,n,2  
sum=sum+i  
10 continue

do10i=1100  
do10i=1,100

Fortran IV, Fortran 77, Fortran 95, Fortran 2003, ...  
What will the language of the year 2000 look like? ...  
Nobody knows but it will be called FORTRAN ☺



## Fortran

- FORTRAN (IBM)
  - polia, priradenie, 3-IF, GOTO, DO
- FORTRAN II
  - SUBROUTINE, CALL
- FORTRAN III
  - inline assembler, strojovo závislý na IBM
- FORTRAN IV
  - strojovo nezávislý, boolean, logický IF
- FORTRAN 66
  - INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL
  - external, 6-písmenové identifikátory
- FORTRAN 77
  - CHARACTER typ, DO WHILE-END DO, bitové operácie
- Fortran 90
  - case-sensitive, moduly, rekurzívne procedúry, overloading
  - pointre, allocate a deallocate, dynamické dát.štruktúry
  - exit, inline comments
- Fortran 2003
  - objektovo-orientovaný, dedenie, polymorfizmus,
  - procedúry ako pointre
- Fortran 2008

*A good  
FORTRAN  
programmer  
can write  
FORTRAN code  
in any language*



"GOD is REAL (unless declared INTEGER)."



# Fortran

---

```
FUNCTION NGCD(NA, NB)
 IA = NA
 IB = NB
1 IF (IB.NE.0) THEN
 ITEMP = IA
 IA = IB
 IB = MOD(ITEMP, IB)
 GOTO 1
 END IF
 NGCD = IA
 RETURN
END
```

```
program GCD
 integer m, n, r
10 print *, 'Please give values for m and n'
 read *, m, n
20 if (n .eq. 0) go to 30
 r = mod(m,n)
 m = n
 n = r
 go to 20
30 print *, 'gcd = ', m
 go to 10
end
```

```
! Hello World in Fortran 90 and 95
PROGRAM HelloWorld
 WRITE(*,*) "Hello World!"
END PROGRAM
```



# COME FROM

The author feels that the COME FROM will prove an invaluable contribution to the field of computer science. It is confidently predicted that this solution will be implemented in all future programming languages, and will be retrofitted into existing languages.

```
10 J=1
11 COME FROM 20
12 WRITE (6,40) J STOP
13 COME FROM 10
20 J=J+2
40 FORMAT (14)
```

```
I = 1
IF (I .LT. 10) COME FROM 50
I = I+1
50 WRITE (6,60) I
STOP
60 FORMAT (14)
```

```
DO 200 INDEX=1,10
10 X=1.
20 X=X*2.
30 X=X*3.
40 X=X*4.
50 X=X*5.
60 X=X*6.
70 X=X*7.
80 X=X*8.
90 X=X*9.
100 X=X*10.
COME FROM
(10,20,30,40,50,60,70,80,90,100),INDEX
WRITE (6,500) INDEX,X
200 CONTINUE
STOP
500 FORMAT (14,2X,F12.0)
```



# Security update for iOS/OSX

2014

Originálny Apple kód na verifikovanie certifikátu (2014, update iOS 7.0.6):  
Vidíte chybu v tomto kóde ? ... nevadí, ani Apple ju nenašiel...

```
if ((err = SSLFreeBuffer(hashCtx)) != 0)
 fail();
if ((err = ReadyHash(SSLHashSHA1, hashCtx)) != 0)
 fail();
if ((err = SSLHashSHA1.update(hashCtx, clientRandom)) != 0)
 fail();
if ((err = SSLHashSHA1.update(hashCtx, serverRandom)) != 0)
 fail();
if ((err = SSLHashSHA1.update(hashCtx, signedParams)) != 0)
 fail();
 fail();
if ((err = SSLHashSHA1.final(hashCtx, hashOut)) != 0)
 fail();
```

# Boing 737Max

2019

Boeing  
NYSE: BA

+ Sledovať

**375,63** USD **-7,31 (1,91 %) ↓**

Zavreté: 13. 9., 7:46 GMT-4 · Vylúčenie zodpovednosti  
Pred otvorením burzy 378,15 **+2,52 (0,67 %)**

1 deň

5 dni

1 mesiac

6 mesiacov

**DDD**

1 rok

5 rokov

Max.



# Boing 737Max – celá pravda

2020

Boeing  
NYSE: BA

+ Sledovať

**161,14** USD **-6,39 (3,81 %) ↓**

Zavreté: 18. 9., 19:59 GMT-4 · Vylúčenie zodpovednosti

Po zavretí 161,82 **+0,68 (0,42 %)**

1 deň

5 dni

1 mesiac

6 mesiacov

DDD

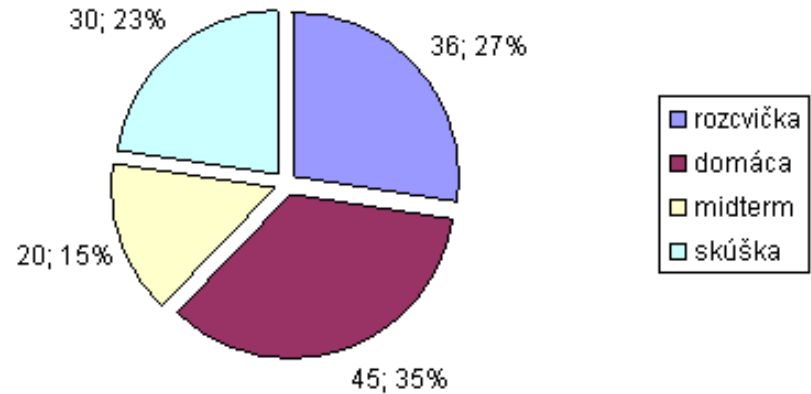
1 rok

**5 rokov**

Max.



# Pre koho nie/je prednáška



nie je:

- neznáša, resp. nevidí dôvod naučiť sa, **rekurziu**
- učí sa len to, čo môže **zajtra použiť** v robote
- verí, že s javou, c++, php **prežije do dôchodku...**
- nerád **pracuje cez semester**

je:

- uznáva tzv. **programovanie so zamyslením**, ako analógiu čítania s porozumením
- rád hľadá a rieši malé „triky“ programíky a rébusy