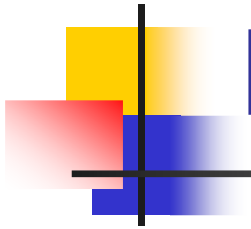


Procedurálne programovanie

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

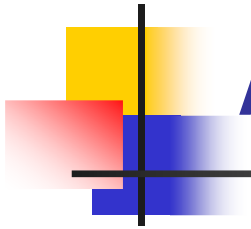
- prehľad od assemblerov k *Fortranom*,
- história procedurálnych jazykov (ilustrácia na Fortrane),
- alternatívny pohľad na goto ☺
- jazyky procedurálneho programovania (COBOL, Algol, Basic, Ada),

Cvičenie:

- rekurzia, rekurzia, rekurzia, …
  - rekurzia s obmedzeniami: mini-Python
- výpočty na „handicapovaných" modeloch
  - Minského stroj, BF, …

Na budúce:

- Python pre pascalistov a céčkarov

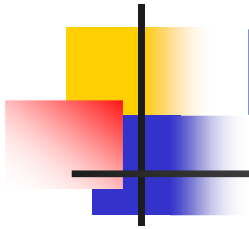


Assembler

- (strojovo) závislý na architektúre
- zaviedol pojem návestia (6-8 znakov)
- (prakticky) nemá štruktúrované príkazy
- má podprogramy ako prototyp procedúr

```
result dw      0                ; int result
       mov     cx, 5            ; cx = 5
       mov     ax, 1            ; ax = 1
       cmp     cx, 0            ; test cx == 0 ?
       je      print            ; if true, goto print
       mov     bx, cx           ; bx = cx
       mov     ax, 1            ; ax = 1
       mov     bx, 1            ; bx = 1
cyklus:                         ; navestie cyklu
       mul     bx               ; (dx,ax) = ax*bx
       cmp     dx, 0            ; test dx == 0 ?
       jne     overflow         ; if false, goto overflow
       inc     bx               ; bx++
       loop    cyklus           ; while(--cx>0)goto cyklus
       mov     result, ax       ; result = ax
print:                          ; podprogram pre tlac
       .....
```

•programu v assembleri nemožno rozumieť bez detailnej znalosti architektúry procesora
•na prvé priblíženie uveďme aspoň registre



Registre

- 32 bitové: EAX, EDX , ECX, EBX
- 16 bitové: AX, DX, CX, BX
- 8 bitové: AX=(AH, AL), DX=(DH, DL), CX=(CH, CL), BX=(BH, BL)

špecializované:

- ESP – (stack pointer) vrchol systémového zásobníka
- EBP - (base/frame pointer) argumenty posledne volanej procedúry
- EDI, ESI – indexovacie registre, napr. pre indexovanie v poli, reťazci, ...

segmentové:

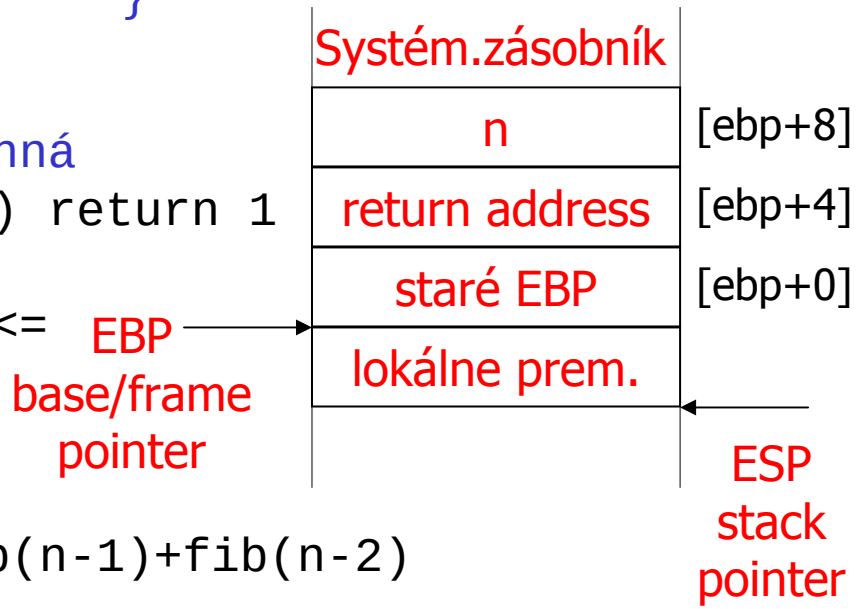
- CS - code segment,
- DS – data segment,
- SS - stack segment,

http://en.wikipedia.org/wiki/X86_assembly_language

Čo s procedúrami ?

```
int fib(int n) {
  if (n <=2) return 1;
  else return fib(n-1)+fib(n-2);
}
int main(int argc, char* argv[]) {
  fib(10);
}
```

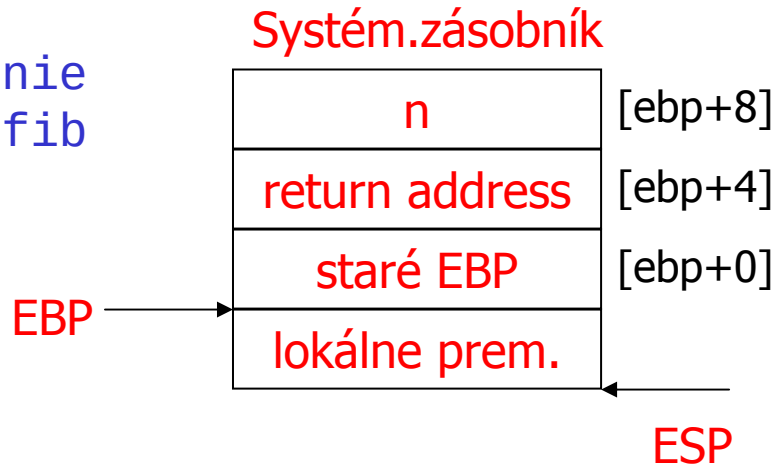
```
fib:    push ebp        ; vstup do
        mov  ebp, esp   ; procedúry
        push ebx        ; lok.premenná
                        ; if (n < 2) return 1
        cmp dword ptr [ebp+8],2
        jnle eee        ; jump not <=
        mov eax,1       ; return 1
        pop ebx         ; výstup z
        pop ebp         ; procedúry
eee:                            ; fib(n-1)+fib(n-2)
        mov edx,[ebp+8]         ; n
        dec edx                 ; n-1
        push edx                ; n-1 ako argument na stacku
        call fib                ; rek. Volanie fib(n-1)
        pop ecx                 ; vyber argument n-1 zo stacku
        mov ebx, eax            ; uloz do ebx výsledok fib(n-1)
```



Čo s procedúrami ?

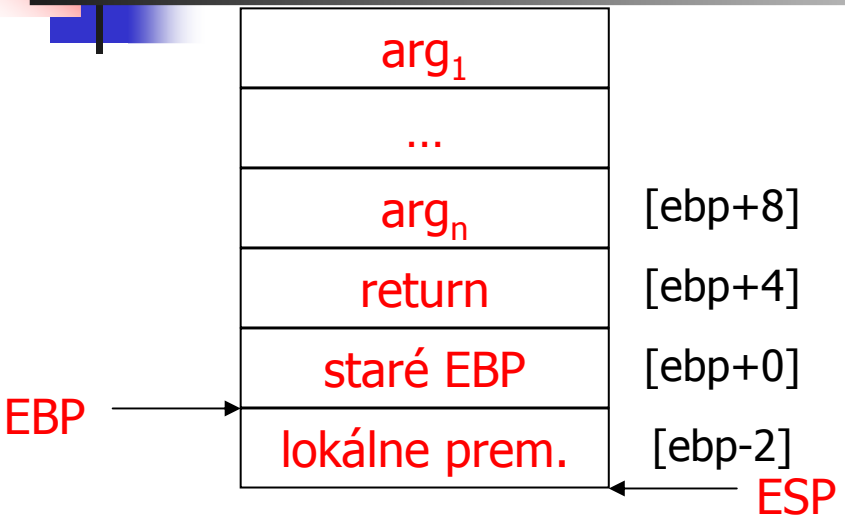
```
int fib(int n) {
  if (n <=2) return 1;
  else return fib(n-1)+fib(n-2);
}
int main(int argc, char* argv[]) {
  fib(10);
}
```

```
mov ebx, eax          ; uloz medzivýsledok fib(n-1)
mov eax, [ebp+8]      ; n
add eax,-2            ; n-2
push eax              ; argument n-2 na stack
call fib              ; fib(n-2)
pop ecx               ; 'vystackuj' n-2
add ebx,eax           ; fib(n-1)+fib(b-2)
mov eax, ebx          ; vysledok do eax

pop ebx               ; ukoncenie
pop ebp               ; proc. fib
ret
```

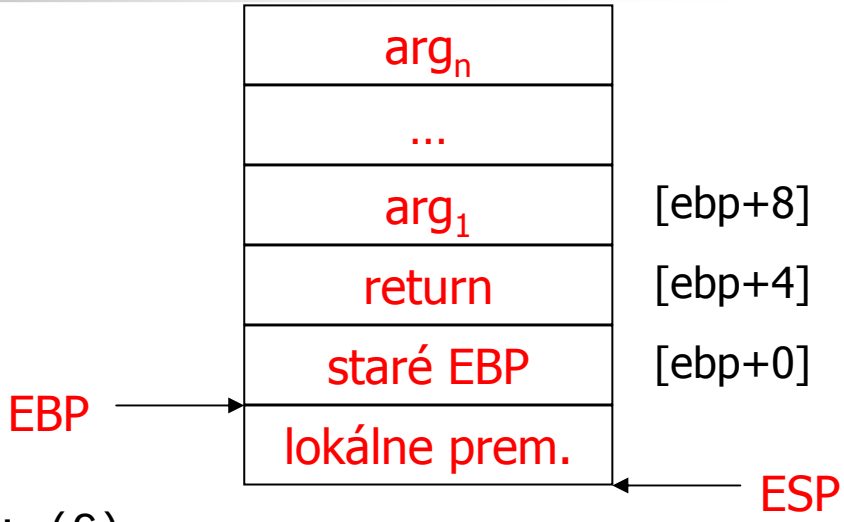


Pascal vs. C-calling sequence

Existujú dve základné sekvencie ukladania argumentov pri volaní proc. Prečo, odkiaľ pochádza dôvod ?




```
fib: (Pascal)
        push    ebp     ; vstup
        mov     ebp, esp
        . . . .
        pop     ebp     ; výstup
        ret     4*n     ; veľkosť arg.
        ========================
        push    n       ; volanie fib
        call    fib
```

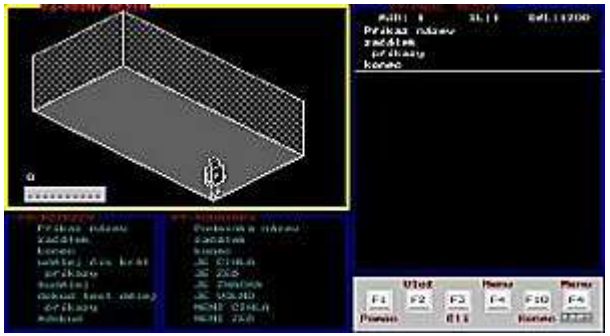



```
fib: (C)
        push    ebp     ; vstup
        mov     ebp, esp
        . . . .
        pop     ebp     ; výstup
        ret
    ============================
        push    n       ; volanie
        call    fib
        sub     esp,4*n;veľkosť arg.
```

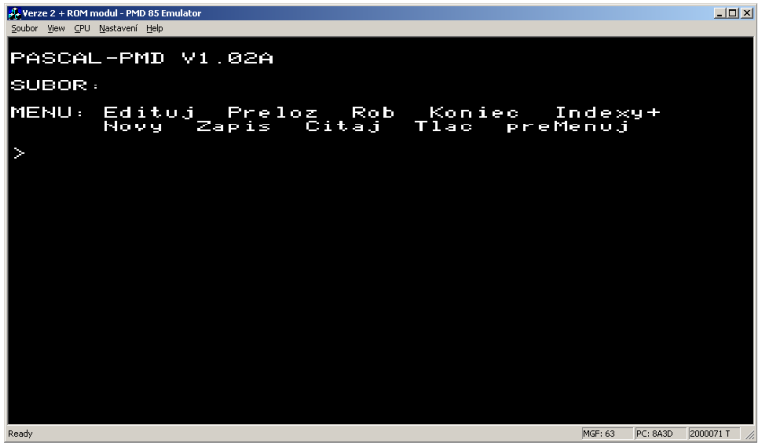
Štruktúrovaný assembler

```
        cx = 5;
        ax = 1;
        bx = 1;
        do {
          (dx,ax) = ax*bx;
          if (dx != 0) goto overflow;
          bx++;
        } while(--cx>0);
        result = ax;
print:                                  ; podprogram pre tlac
        .....
```

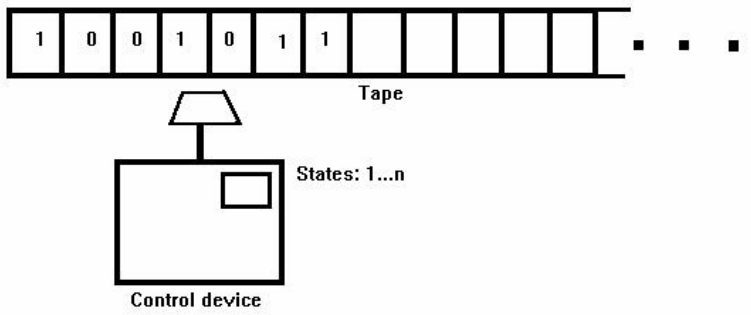
- priamočiary preklad do stroj.kódu,
- bežné štruktúrované príkazy, if, while, ...



PMD-2 (intel 8080):
- Žofka (Logo),
- Karel-3D (M.Vittek),
- Pascal (A.Blaho).

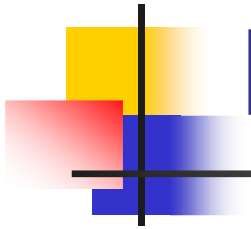


"Teoretický" assembler



existujú rôzne abstraktné modely výpočtov – slúžia na štúdium vypočítateľnosti v okrajových podmienkach, napr.:

- Turingov stroj
  (pracovná páska – neobmedzene veľa konečno-stavových bezmenných registrov, relatívne adresovanie na páske)
- Minského stroj
  (neobmedzene veľa neobmedzených pomenovaných registrov)
- jazyk P" (populárnejšia analógia BrainF*ck)
  (pracovná páska – neobmedzene veľa neobmedzených bezmenných registrov, relatívne adresovanie na páske)
- a množstvo ďalších (všetky sú silou ekvivalentné),



Minského (zápalkový) stroj

- neobmedzený počet registrov (premenných = zápalkových krabičiek)
- základné operácie: +reg, -reg
- test a cyklus: ak reg != 0 tak ..., pokial reg != 0 …
- žiadne procedúry ani rekurzia

```
# Urob kopiuN (kopiaN := n; pomocna := n)
  pokial hrka n
      -n
      +kopiaN
      +pomocna
  koniec pokial
  pokial hrka pomocna (n := pomocna)
      +n
      -pomocna
  koniec pokial
```

Brainf*ck

>	++ptr;
<	--ptr;
+	++*ptr;
-	--*ptr;
.	putchar(*ptr);
,	*ptr=getchar();
[	while (*ptr) {
]	}

10*(7,10,3,1)

```
++++++++++
[>+++++++>++++++++++>+++>+<<<<-]
>++.                            Print 'H'   72
>+.                             Print 'e'    101
+++++++.                        Print 'l'        101+7=108
.                               Print 'l'
+++.                            Print 'o'
>++.                            Print ' '
<<++++++++++++++++.             Print 'W'
>.                              Print 'o'
+++.                            Print 'r'
------.                         Print 'l'
--------.                       Print 'd'
>+.                             Print '!'
>.                              Print newline
```

Urban Müller, 1993
Böhm in 1964 jazyk P”

http://en.wikipedia.org/wiki/Brainfuck

http://www.kmonos.net/alang/etc/brainfuck.php

Brainf*ck príklady

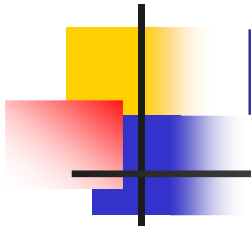
- sčítanie

```
++++++>++++++<      ; 6, 5, 0, 0, 0, …
[->>+<<]            ; 0, 5, 6, 0, 0, …
>                   ; 0, 5, 6, 0, 0, …
[->+<]              ; 0, 0, 11, 0, 0, …
>                   ; 0, 0, 11, 0, 0, …
.
```

- násobenie

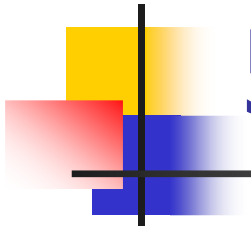
```
++++++>++++<        ; 6, 4, 0, 0, 0, …
[->                 ; 5, 4, 0, 0, 0, …
  [ - > + > + << ]  ; 5, 0, 4, 4, 0, …
  >                 ; 5, 0, 4, 4, 0, …
  [ - < + >] <<     ; 5, 4, 0, 4, 0, …
]
>>>.                ; 0, 0, 0, 24, 0, …
```

- interpreter jazyka BrainF*ck a naprogramujte v ňom faktoriál



Procedurálne programovanie

- volanie procedúry/funkcie/podprogramu
- spôsob predávania argumentov procedúre
  - volanie hodnotou
  - volanie referenciou
- rekurzia (nebola vždy samozrejmosťou)
- existujú len data a procedúry – objekty 1.rádu
- štruktúrované programovanie (gotoless)
- programovanie vo veľkom
- modularita, scoping
- globálne premenné, side-efect
- sémantika programov (najslabšia podmienka)

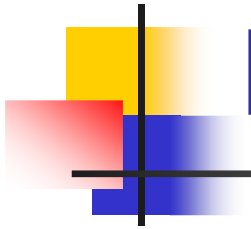


50-60 -te roky

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

- 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

Algol vs. Fortran z pohľadu kontinentov
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5396281&isnumber=5255174&tag=1



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

```
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
```

Cyklus, ktorý sčíta nepárne čísla

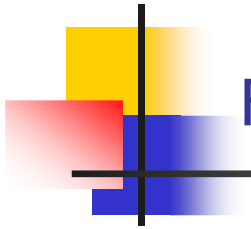
```
integer i, n, sum
sum = 0

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

```
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

A good FORTRAN programmer can write FORTRAN code in any language

- 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

"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
```

http://en.wikibooks.org/wiki/Fortran/Fortran_examples

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)
```

http://www.fortranlib.com/gotoless.htm

Teoréma štruktúrovaného programovania

na všetko stačí:

- príkaz priradenia,
- sekvencia príkazov,
- if-then, if-then-else
- while

(sú)Boj „skutočných programátorov“ a „pojedačov koláčov“

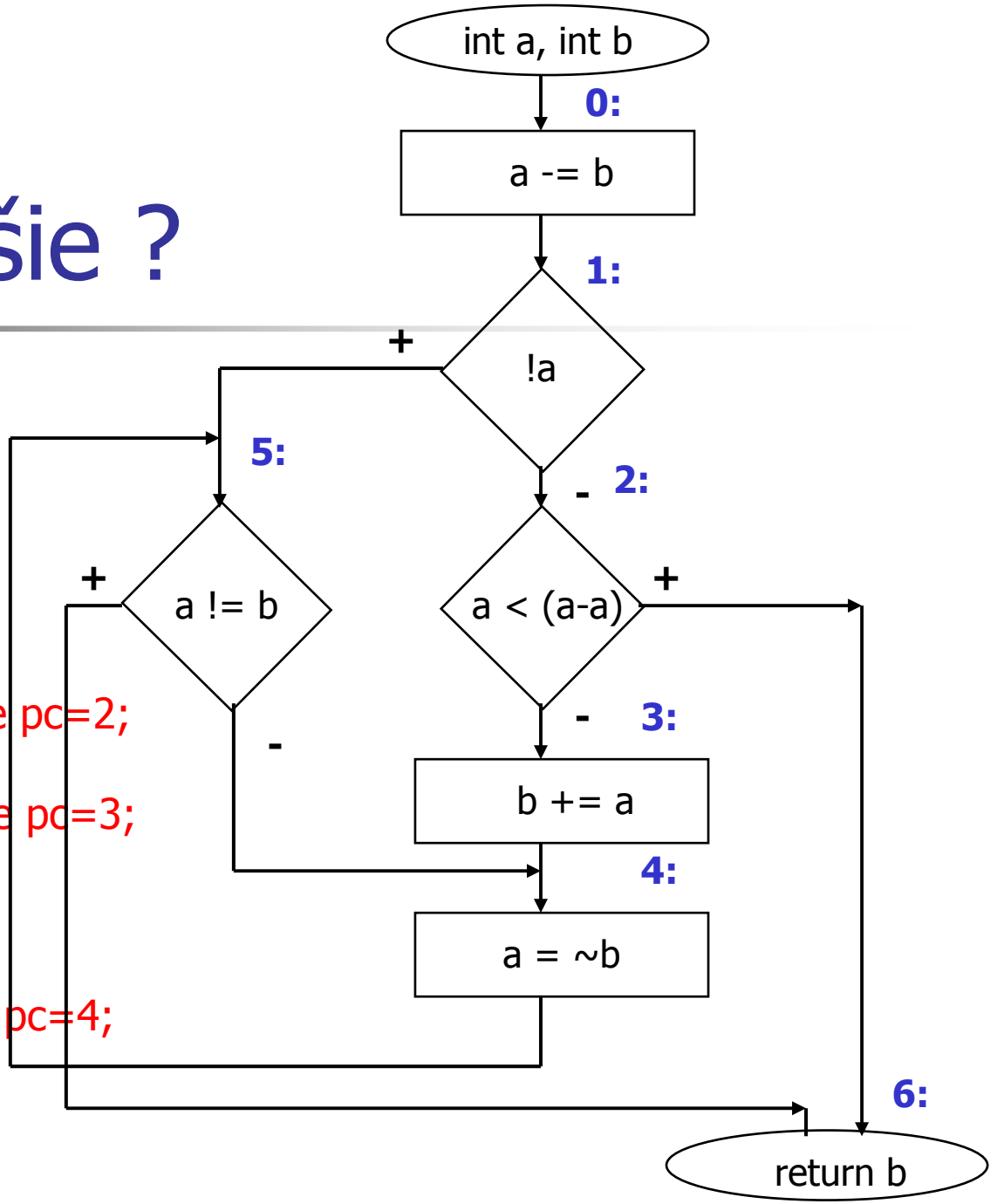
- E.Dijkstra: *Go To Statement Considered Harmful*, CACM, 1968
- F.Rubin: "'GOTO Considered Harmful' Considered Harmful„ CACM , 1987
- D.Moore: """GOTO Considered Harmful" Considered Harmful' Considered Harmful?„ CACM, 1987

viac na http://david.tribble.com/text/goto.html

P” prvý "GOTO-less" štruktúrovaný programovací jazyk ekvivalentný Turingovmu stroju, 1964

Je to lepšie ?

```
int max(int a,int b) {
int pc = 0;
  while (true)
    switch (pc) {
      case 0: a -= b; pc=1;
              break;
      case 1: if (a==0) pc=5; else pc=2;
              break;
      case 2: if (a<a-a) pc=6; else pc=3;
              break;
      case 3: b+=a; pc=4; break;
      case 4: a=~b; pc=5; break;
      case 5: if (a!=b) pc=6; else pc=4;
              break;
      case 6: return b;
    }
}
```



COBOL - *COmmon Business Oriented Language*

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence, E.Dijkstra

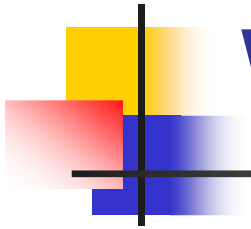
```
COMPUTE X = (-B + (B ** 2 - (4 * A * C)) **.5) / (2 * A)
```

```
MULTIPLY B BY B GIVING B-SQUARED.
MULTIPLY 4 BY A GIVING FOUR-A.
MULTIPLY FOUR-A BY C GIVING FOUR-A-C.
SUBTRACT FOUR-A-C FROM B-SQUARED GIVING RESULT-1.
COMPUTE RESULT-2 = RESULT-1 ** .5.
SUBTRACT B FROM RESULT-2 GIVING NUMERATOR.
MULTIPLY 2 BY A GIVING DENOMINATOR.
DIVIDE NUMERATOR BY DENOMINATOR GIVING X.
```

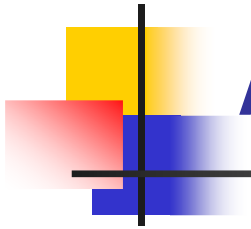
OO ext. of COBOL should be called
ADD 1 TO COBOL GIVING COBOL

SQL ???

Hello World

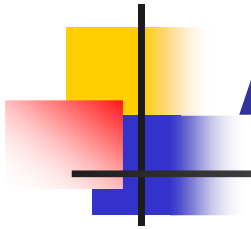


```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.    HELLOWORLD.
000300
000400*
000500 ENVIRONMENT DIVISION.
000600 CONFIGURATION SECTION.
000700 SOURCE-COMPUTER. RM-COBOL.
000800 OBJECT-COMPUTER. RM-COBOL.
000900
001000 DATA DIVISION.
001100 FILE SECTION.
001200
100000 PROCEDURE DIVISION.
100100
100200 MAIN-LOGIC SECTION.
100300 BEGIN.
100400     DISPLAY " " LINE 1 POSITION 1 ERASE EOS.
100500     DISPLAY "Hello world!" LINE 15 POSITION 10.
100600     STOP RUN.
100700 MAIN-LOGIC-EXIT.
100800     EXIT.
```



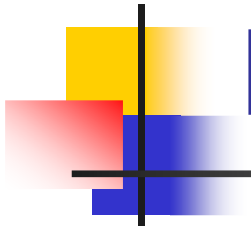
Algol

- imperatívny procedurálny jazyk, 1960
- Algol 58 (IAL), Algol 60, Algol 68, …
- Backus (Backus Naur form - BNF)
- blok programu: begin-end
- program *layout* je dôležitý – pretty printing
- prvý krok ku štruktúrovanému programovaniu
- čo platfotma (HW), to iný Algol
- žiadne I/O
- v knihách stále existuje Pidgin Algol



Algol 60 príklad

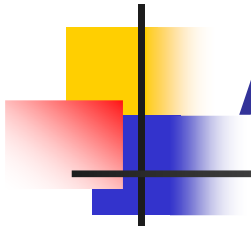
```
procedure Absmax(a) Size:(n, m) Result:(y) Subscripts:(i, k);
    value n, m; array a; integer n, m, i, k; real y;

comment The absolute greatest element of the matrix a, of size n by m
is transferred to y, and the subscripts of this element to i and k;

begin integer p, q;
    y := 0; i := k := 1;
    for p:=1 step 1 until n do
      for q:=1 step 1 until m do
        if abs(a[p, q]) > y then
            begin
              y := abs(a[p, q]);
               i := p; k := q
            end
end Absmax
```



BNF – aritmetický výraz

Formalizmus často používaný na zápis syntaxe programovacieho jazyka:

- <expression> ::= <arithmetic expression> | <Boolean expression>
- <adding operator> ::= **+** | **-**
- <multiplying operator> ::= ***** | **/** | **//**
- <primary> ::= <unsigned number> | <variable> | (<arithmetic expression>)
- <factor> ::= <primary> | <factor> ^ <primary>
- <term> ::= <factor> | <term> <multiplying operator> <factor>
- <arithmetic expression> ::= <term> |
  < arithmetic expression > <adding operator> <term>



Algol 68 vs. C++

- jazyk, čo predbehol dobu (a preto neprežil...)
- **van Wijngaarden W-grammar** – bezkontextová gramatika generuje nekonečnú množinu jazykov, t.j. môžete si definovať syntax jazyka, v ktorom budete programovať...
- potlačili jednoduchosť Algol 60,
- Programming Algol 68 Made Easy

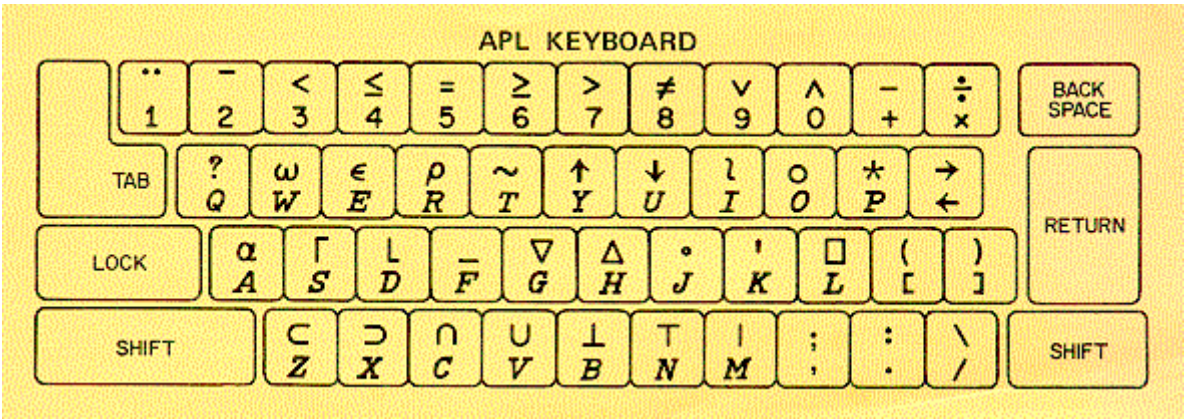
C++ nemá:
•vnorené funkcie,
•definovateľné operátory s prioritami,
•garbage collection,
•*use before define*,
•operátor priradenia := (= / ==),
•nonlocal GOTO,

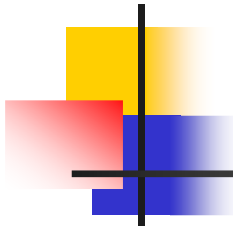
ALGOL 68 nemá:
•public/private access protection,
•preťaženie procedúr (len operátorov),
•explicitnú alokáciu pamäte
•dopredné deklarácie,
•pred-processing,
•zmätok s parametrami typu &,

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. $N \leftarrow 4\ 5\ 6\ 7$
- pričítanie 4 k vektoru (dostaneme 8 9 10 11) $N + 4$
- tlač súčtu N, t.j. 22. $+/N$





APL

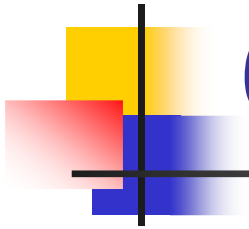
"APL, in which you can write a program to simulate shuffling a deck of cards and then dealing them out to several players in four characters, none of which appear on a standard keyboard." David Given 52?52

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

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

1. ι creates a vector containing integers from 1 to R (if R = 6 at the beginning of the program, ιR is 1 2 3 4 5 6)
2. Drop first element of this vector ($\downarrow$ function). So, 1$\downarrow\iota$R 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 (o.x 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 (ε , 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) (~ , 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

http://en.wikipedia.org/wiki/APL_(programming_language)



60-te roky

- 1964 - BASIC
- 1964 - PL/I
- 1970 - Pascal (N.Wirth)
- 1972 - C (D.Ritchie)

Basic

- 1964 John Kemeny,Thomas Kurtz

1. Be easy for beginners to use.
2. Be a general-purpose programming language.
3. Allow advanced features to be added for experts (while keeping the language simple for beginners).
4. Be interactive.
5. Provide clear and friendly error messages.
6. Respond fast for small programs.
7. Not require an understanding of computer hardware.
8. Shield the user from the operating system

Tiny Basic (BNF)

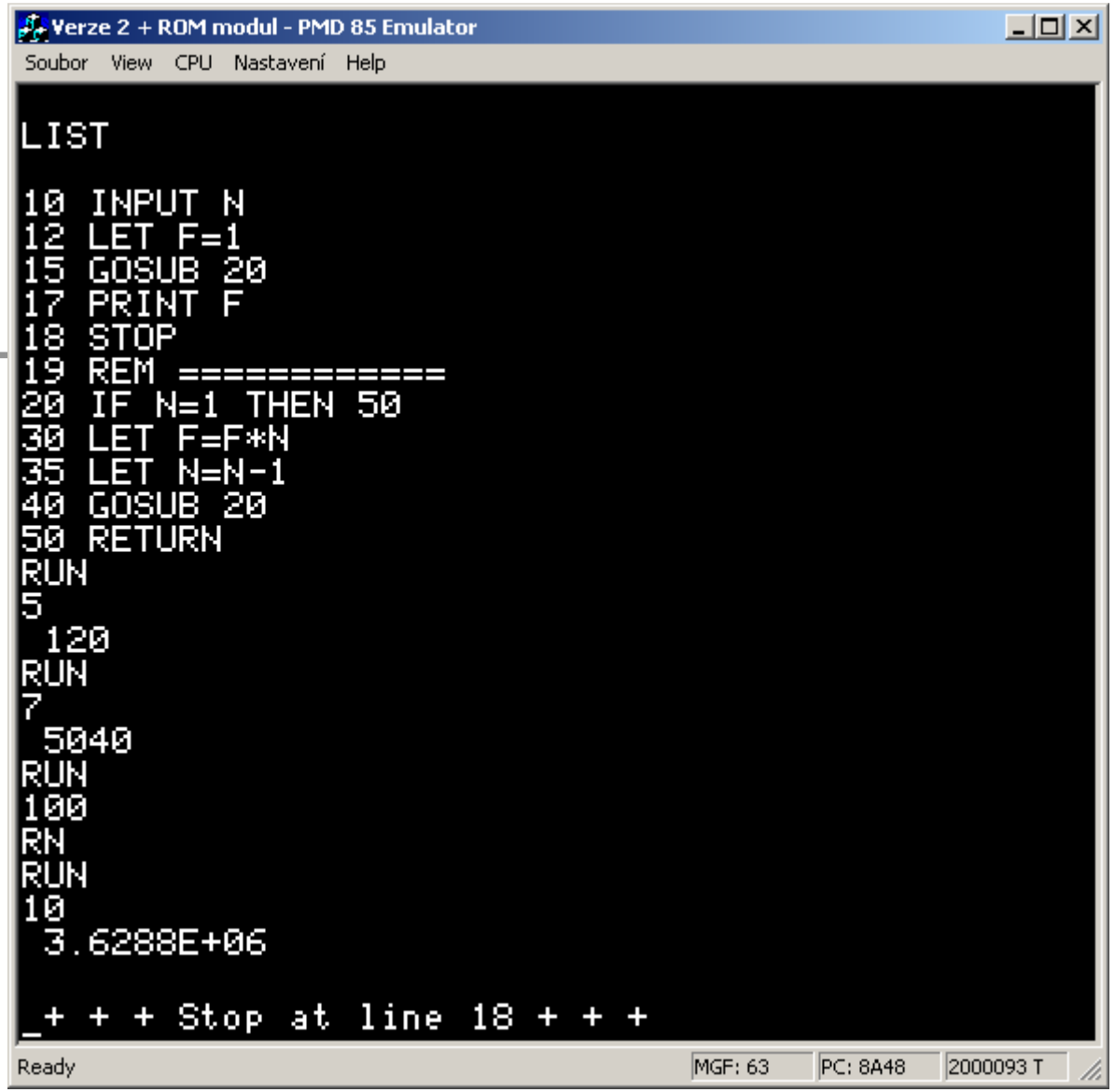
line ::= number statement CR | statement CR
statement ::= PRINT expr-list
IF expression relop expression THEN statement
GOTO expression
INPUT var-list
LET var = expression
GOSUB expression
RETURN

CLEAR
LIST
RUN
END

expr-list ::= (string|expression) (, (string|expression)*)
var-list ::= var (, var)*
expression ::= (+|-|ε) term ((+|-) term)*
term ::= factor ((*|/) factor)*
factor ::= var | number | (expression)
var ::= A | B | C .... | Y | Z
number ::= digit digit*
digit ::= 0 | 1 | 2 | 3 | ... | 8 | 9
relop ::= < (>|=|ε) | > (<|=|ε) | =

Emulátor

- nemá lokálne premenné
- podprogramy bez argumentov
- rekurzia nie je prirodzená...



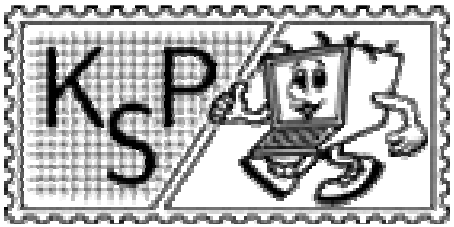
http://dai.fmph.uniba.sk/courses/PARA/soft/PMD85v13p.exe

http://www.youtube.com/watch?v=38559AI8RBU

Basic - example

```
10 INPUT "What is your name: "; U$
20 PRINT "Hello "; U$
30 REM
40 INPUT "How many stars do you want: "; N
50 S$ = ""
60 FOR I = 1 TO N
70 S$ = S$ + "*"
80 NEXT I
90 PRINT S$
100 REM
110 INPUT "Do you want more stars? "; A$
120 IF LEN(A$) = 0 THEN GOTO 110
130 A$ = LEFT$(A$, 1)
140 IF (A$ = "Y") OR (A$ = "y") THEN GOTO 40
150 PRINT "Goodbye ";
160 FOR I = 1 TO 200
170 PRINT U$; " ";
180 NEXT I
190 PRINT
```

```
10 print “Give values for m and n"
20 input m, n
30 if n = 0 then 80
40 let r = m mod n
50 let m = n
60 let n = r
70 goto 30
80 print "gcd = ", m
90 goto 10
100 end
```



www.ksp.sk

http://ipsc.ksp.sk/

Pascal

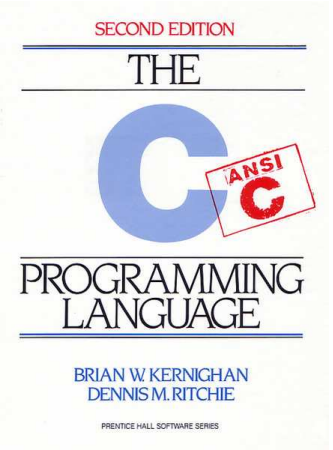
- N.Wirth, 1970, Zurich
- jednoduchosť

Pascal implementation, 1973

- Blaho, Vavro, Pascal EC1021, PMD-85, 1980,
- Paul Voda ~1976,
- Donald Knuth, Tex,
- ISO-Pascal – "jazyk len na výuku"
- B.Kernighan, 1981: Why Pascal Is Not My Favorite Programming Language
- Borland: Turbo Pascal (Hejlsberg, 50$), Delphi

```
function GCD(a,b : integer):integer;
begin
  if (b mod a) = 0 then Result := a
  else Result := GCD(b, a mod b);
end;
```

C



- jednoduché jadro jazyka doplnené rozsiahlymi knižnicami, matematické funkcie, práca so súbormi, dát.štruktúrami
- procedurálny aspekt s možnosťou štruktúrovaného štýlu programovania
- striktný typový systém, ktorý sa dá v prípade potreby obísť
- textový pred-procesor, makrá s parametrami
- prístup na úroveň HW, pointre a ich aritmetika
- minimalistická množina kľúčových slov
- pointre na funkcie a statické premenné (prvotná forma closure, run-time polymorfizums)
- statický scope premennej

Je dobré niekoho niečo nútiť ?

```
int mymax (int(*a)(int(*)(int(*)()),int(*)(int(*)(int**))), int(*b)(int(*)
    (int(*)()),int*,int(*)(int(*)()))){return
    (int)((((int(*)(int(*)(int(*)()),int( *)(int(*)())))a)>
    ((int(*)(int(*)(int(*)()),int(*)(int(*)()))b))?((int(*)(
    int(*)(int(*)()),int(*)(int(*)()))a):((int(*)(int(*)(int(*)()),int(*)(int
    (*)( ))))b));??>
```

```
printf("%d\n",mymax((int(*)(int(*)(int(*)()
),int(*)(int(*)(int**))))3,(int(*)(int(*)(int(*
)()),int*,int(*)(int(*)()))52));
```

```
int max(int a,int b) {
    a-=b;
    if(!a) goto d;
    if(a<(a-a)) goto e;
    b+=a; goto f;
d: if(a!=b) goto e;
f:  a=~b;
    goto d;
e: return b;
}
```

Obfucated code

International Obfuscated C Code Contest (http://www.ioccc.org/)

```
#define _ -F<00||--F-OO--;
int F=00,OO=00;main(){F_OO();printf("%1.3f\n",4.*-F/OO/OO);}F_OO()
{
          _-_-_-_
      _-_-_-_-_-_-_-_-_
    _-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_
    _-_-_-_-_-_-_-_-_-_-_
      _-_-_-_-_-_-_-_-_
          _-_-_-_
}
```

Veľa inšpriácie do života:
http://en.wikipedia.org/wiki/Obfuscated_code

Obfucation vs. Code Morphing

```
#include <stdio.h> main(t,_,a)char *a;{return!0<t?t<3?main(-79,-13,a+main(-87,1-_, main(-
    86,0,a+1)+a)):1,t<_?main(t+1,_,a):3,main(-94,-27+t,a)&&t==2?_<13? main(2,_+1,"%s %d
    %d\n"):9:16:t<0?t<-72?main(_,t,
    "@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+,/+#n+,/#\
    ;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \
    q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# \ ){nl]!/n{n#'; r{#w'r
    nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \ iwk{KK{nl]!/w{%'l##w#' i;
    :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \ ;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ')
    }+}{rl#'{n' )# \ }'+}##(!!/") :t<-50?_==*a?putchar(31[a]):main(-65,_,a+1):main((*a=='/')+t,_,a+1)
    :0<t?main(2,2,"%s"):*a=='/'||main(0,main(-61,*a, "!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m
    .vpbks,fxntdCeghiry"),a+1);}
```

"Využitie":

- asp, js, vbasic
- reverse engeneering

Transformácie:

- zmena tvaru programu(layout)
- syntaktická zmena
- sémantická

```
int i = 1;
while (i < 1000) {
  ... A[i] ...;
  i ++;
}
```

```
int i=11;
while (i < 8003) {
  ... A[(i-3)/8] ...;
  i += 8;
}
```

```
int i = 1;
while ((i < 1000) || (i % 1000 == 0)) {
  ...
  i ++;
}
```

http://www.cs.arizona.edu/~collberg/Research/Students/DouglasLow/obfuscation.html

Ada

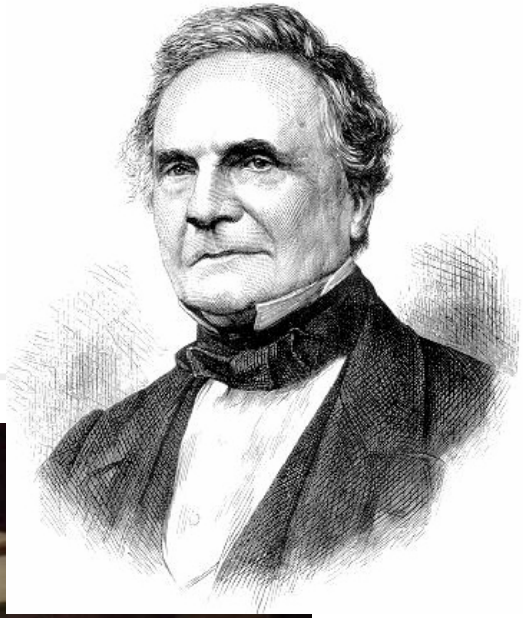
- 1970, US Department of Defense
- real-time systems,
- Military Standard reference manual, Dec 1980, ☺
- striktne typovaný, modularita, run-time checking, parallel processing, exception handling, generics, java-like memory management – GC
- Ada, ISO standard 1987
- Ada 95, object oriented

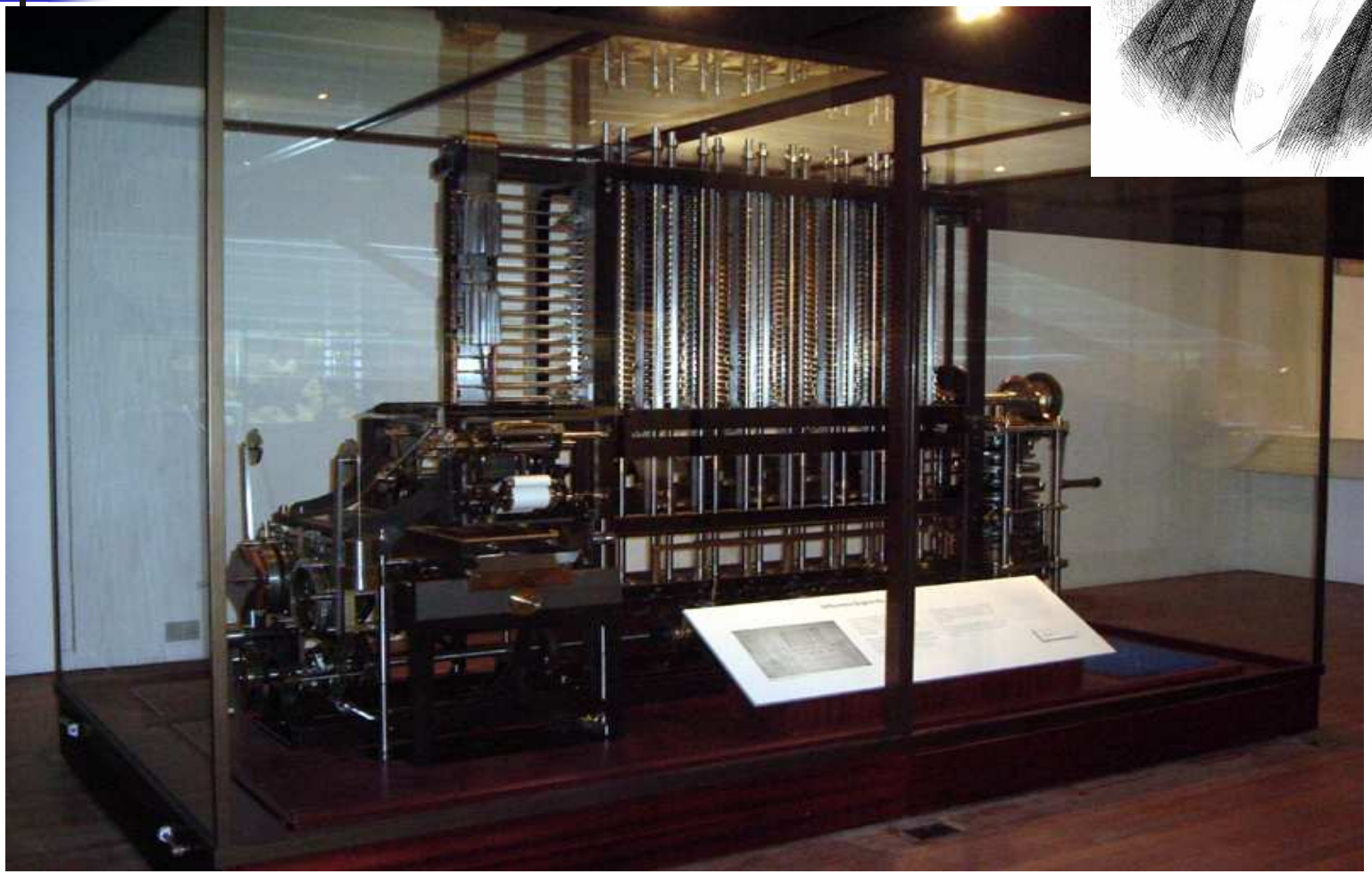
```
function GCD (Jin, Kin : Int) return Int is
    J, K, Tmp : Int;
  begin
    pragma Assert (Jin >= Kin);
    pragma Assert (Kin >= Int_0);
    J := Jin;
    K := Kin;
    while K /= Uint_0 loop
      Tmp := J mod K;
      J := K;
      K := Tmp;
    end loop;
    return J;
  end GCD;
```

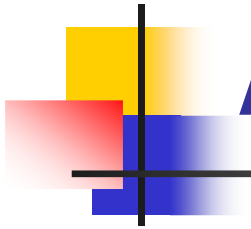
Ada Lovelance

- Countess of Lovelace (1815 – 1852)
- jediná legitímna dcéra poeta Lorda Byrona
- venovala sa vede a matematike (“the queen of parallelograms”)
- výpočet Bernoulliho čísiel na Analytical Engine, 1846
- prvá programátorka
- **Ada Byron's notes on the analytical engine – G**
- ***The Analytical Engine has no pretensions whatever to originate anything. It can do whatever we know how to order it to perform. It can follow analysis; but it has no power of anticipating any analytical relations or truths.***
  Garbage In, Garbage Out (**GIGO**)

Babbage engine







Ariane 5

- Boeing, European Space Agency Ariane, 1996
- software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. This resulted in an Operand Error. The data conversion instructions (in Ada code) were not protected from causing an Operand Error, although other conversions of comparable variables in the same place in the code were protected.
- As soon as the launcher lifts off, this function serves no purpose.
- This time sequence is based on a requirement of Ariane 4 and is not required for Ariane 5.

http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html