Dipartimento di
Informatica e Sistemistica
"Antonio Ruberti"

SAPIENZA
UNIVERSITÀ DI ROMA

History of Programming Languages

Giuseppe De Giacomo

Università degli Studi di Roma "La Sapienza"
http://www.dis.uniroma1.it/~*degiacomo/*

Ingegneria Informatica
Corso di Storia dell'Informatica
A.A.2006/07

Programming Languages Timeline

- Color programming language timeline http://www.oreilly.com/news/graphics/prog_lang_poster.pdf
- Original programming language timeline http://www.levenez.com/lang/history.html

The 50's

- Fortran
- Lisp
- Cobol

FORTRAN

John Backus
(cf. Backus-Naur Form – BNF)

- FORTRAN: “FORmula TRANslating system” (1954) by John Backus and his group @ IBM
- First high level language
- First (industrial) compiled language
- Excellent compiled code
- Focused on scientific calculations (originally it had only integers and floating points)
- Strongly influenced by hardware (IBM 704) - first computer with floating point math
- FORTRAN IV (1962, ANSI standard in 1966) the most used Programming languages for scientific computations till 1978! Then FORTRAN 77 (ANSI 1978)
- Still used for scientific calculations
- Current standard version FORTRAN 90 *(its includes object-oriented features!)*

IBM 704

FORTRAN

- Originally only integer and floating point (real)
- No explicit variable type declaration: variables starting by I,J,K,L,M,N are integers, others are floating points
- No stack, no heap ⇒ no recursion no dynamic memory allocation
- Mathematical if: `IF (arithmetic expression) N1, N2, N3`
- Definite iteration: `DO N1 var = first, last`
- Goto: lots of them!
- Arrays (max 3 dimensional), only complex type
- FORTRAN IV introduced explicit variable typing, strings, logical if, subroutines (procedures) with parameters
- FORTRAN 77 introduced, structured control flow primitives and arrays

FORTRAN IV example

```
// JOB
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
C-------------------------------------------------------
C COMPUTE THE CRITIAL VALUES FOR A QUADRAITIC EQN
C 0=A*X**2+B*X+C
C RETURNS DISCRIMINANT, ROOTS, VERTEX, FOCAL LENGTH, FOCAL POINT
C X1 AND X2 ARE THE ROOTS
C-------------------------------------------------------
      SUBROUTINE QUADR(A,B,C,DISCR,X1,X2,VX,VY,FL,FPY)
      REAL A,B,C,DISCR,X1,X2,VX,VY,FL,FPY

C DISCRIMINANT, VERTEX, FOCAL LENGTH, FOCAL POINT Y
      DISCR = B**2.0 - 4.0*A*C
      VX = -B / (2.0*A)
      VY = A*VX**2.0 + B*VX + C
      FL = 1.0 / (A * 4.0)
      FPY = VY + FL
      FL = ABS(FL)

C COMPUTE THE ROOTS BASED ON THE DISCRIMINANT
      IF(DISCR) 110,120,130

C -VE DISCRIMINANT, TWO COMPLEX ROOTS, REAL=X1, IMG=+/-X2
110   X1 = -B / (2.0*A)
      X2 = SQRT(-DISCR) / (2.0*A)
      RETURN

C ZERO DISCRIMINANT, ONE REAL ROOT
120   X1 = -B / (2.0*A)
      X2 = X1
      RETURN

C +VE DISCRIMINANT, TWO REAL ROOTS
130   X1 = (-B + SQRT(DISCR)) / (2.0*A)
      X2 = (-B - SQRT(DISCR)) / (2.0*A)
      RETURN
C
C NEXT STORE SUBROUTINE ON DISK USING DUP
      END
// DUP
*DELETE             QUADR
*STORE      WS  UA  QUADR
```

```
// JOB
// FOR
*LIST SOURCE PROGRAM
*IOCS(CARD,1132 PRINTER)
*ONE WORD INTEGERS
C-------------------------------------------------------
C PROCESS DATA CARDS WITH A,B,C
C UNTIL A=0
C-------------------------------------------------------

      DATA ICARD,IPRT /2,3/
      REAL A,B,C
      REAL DISCR,XR1,XR2,VX,VY,FL,FPY

      WRITE(IPRT,901)
901   FORMAT(' ------------------------------------------------')

C READ A B C, IF A=0 THEN EXIT
100   READ(ICARD,801)A,B,C
801   FORMAT(3F8.3)

C     EXIT WHEN A IS ZERO
      IF (A) 110,9000,110

C PRINT A B C
110   WRITE(IPRT,902)A,B,C
902   FORMAT(' QUADRATIC A=',F8.3,' B=',F8.3,' C=',F8.3)

C COMPUTE AND PRINT THE CRITICAL VALUES
      CALL QUADR(A,B,C,DISCR,XR1,XR2,VX,VY,FL,FPY)
      WRITE(IPRT,903) DISCR
903   FORMAT(' DISCRIMINANT=',F9.4)
      WRITE(IPRT,904) VX,VY
904   FORMAT(' VERTEX X=',F9.4,'  Y=',F9.4)
      WRITE(IPRT,905) FL
905   FORMAT(' FOCAL LENGTH=',F9.4)
      WRITE(IPRT,906) VX,FPY
906   FORMAT(' FOCAL POINT X=',F9.4,'  Y='F9.4)

      IF (DISCR) 120,130,140
```

FORTRAN IV example

```
C -VE DISCRIMINANT, TWO COMPLEX ROOTS
120   WRITE(IPRT,913) XR1, XR2
913   FORMAT(' COMPLEX ROOTS =(',F9.4,'  +/-',F9.4,'I)')
      GO TO 200

C ZERO DISCRIMINANT, ONE REAL ROOT
130   WRITE(IPRT,912) XR1
912   FORMAT(' ROOT  X =',F9.4)
      GO TO 200

C +VE DISCRIMINANT, TWO REAL ROOTS
140   WRITE(IPRT,911) XR1, XR2
911   FORMAT(' ROOTS X1=',F9.4,'   X2=',F9.4)
C --- GO TO 200

C END OF QUAD
200   WRITE(IPRT,901)
      GO TO 100

C END OF PROGRAM
C DATA FOLLOWS XEQ CARD
9000  CALL EXIT
      END
// XEQ
+001.000+000.000+000.000
+001.000+002.000+003.000
+002.000+002.000+000.000
+002.000+000.000-004.000
+000.500+000.000-004.000
+000.250+002.000-002.000
-004.000+000.000-004.000
+002.730-007.200-003.750
+000.000+000.000+000.000
```

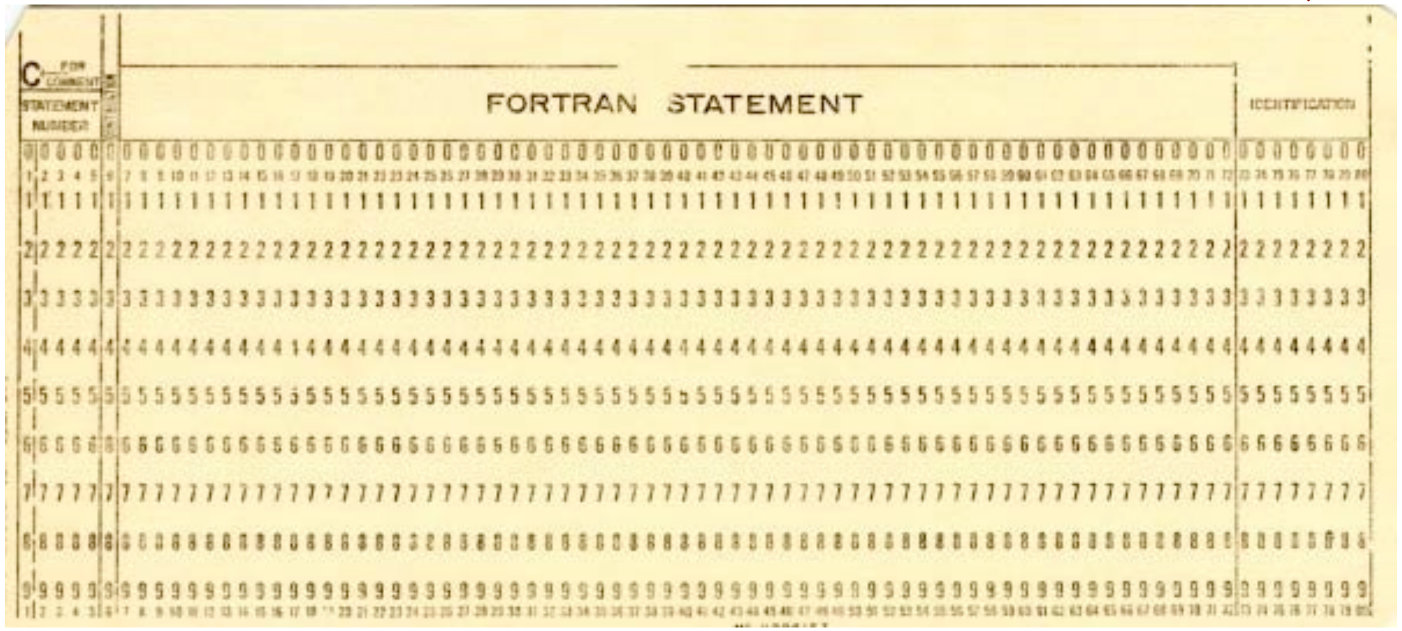
Fortran punch cards

Columns 1..5 for statement numbers

Column 6 for continuing from previous statement

Columns 7.. 72 for writing statements

Columns 73..80 were cards sequence number (in case they needed to be resorted)



Fortran ('till FORTRAN 77) used to have a rigid formatting of the text that was inherited from the IBM 704 punch cards used to insert programs

FORTAN references

- Backus et al: “The FORTRAN Automatic Coding System for the IBM 704 EDPM” (1956) - the very first manual
  http://www.fh-jena.de/~kleine/history/languages/FortranAutomaticCodingSystemForTheIBM704.pdf

- Backus’s “History of FORTRAN I, II, III”
  http://community.computerhistory.org/scc/projects/FORTRAN/paper/p165-backus.pdf

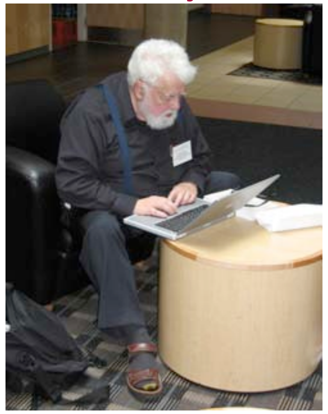
- “Real Programmers Don’t Use Pascal”
  http://www.pbm.com/~lindahl/real.programmers.html

LISP

John McCarthy
(cf. Artificial Intelligence, Logic)

- LISP: “LISt Processing language”
  (1958) by John McCarthy @ MIT
- LISP: “Lots of Irritating Superfluous Parentheses” ☺
- Based on Lambda Calculus
  (theory of mathematical functions - Church&Kleene in ’30s)
- Introduces functional programming
  (still the best known functional programming language)
- Main language used in AI
- By now surprisingly efficient
- Emacs and AutoCad are implemented in LISP
- Parentheses are not a scandal anymore cf. XML
- Current versions
  - Scheme
  - Common Lisp
  - CLOS (Common Lisp Object System, with OO features)

He’s still very active!



LISP

- Recursion
- Introduces linked lists (a very powerful complex data structure): pointers and records
- Dynamic memory allocation
- Efficiency through shared memory without side effect
- Garbage collection (cf JAVA)
- tail recursion optimization
- Metaprogramming
- Mature versions surprisingly efficient
- Static vs dynamic scope dialects for a long time
- Two main variants (both with static scope)
  - Scheme (Steele&Sussman)
  - Common Lisp (ANSI 1994)

Guy L. Steele, Jr.
now working on Java

Gerald Jay Sussman
coauthor of the ISCP book

LISP example

```
;prefix notation
(+ 10 20)
;output 30

(* (+ 10 20) 3)
;output 30

;quoting
`(1 2 3)    abbrev (quote (1 2 3))
;output (1 2 3)

;list processing
(car `(1 2 3) )
; output 1

(cdr `(1 2 3))
; output (2 3)

(cons 1 `(2 3))
; output (1 2 3)
```

car & cdr were the original name of the IBM 704 assembler macros performing the operation

```
;function definition
(define factorial (n)
   (if (<= n 1)
      1
      (* n (factorial (- n 1)))
   )
)

(factorial 3)
; output 6


;metaprogramming
(cons `* (cdr `(+ 10 20)))
; output (* 10 20)

(eval
   (cons `* (cdr `(+ 10 20)))
)
; output 200
```

eval forces a further evaluation on the argument list

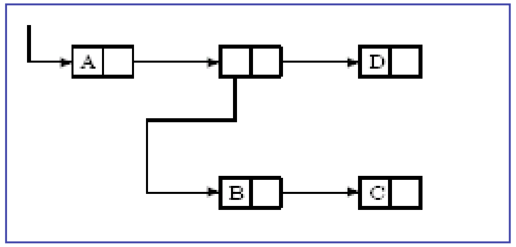
Linked Lists

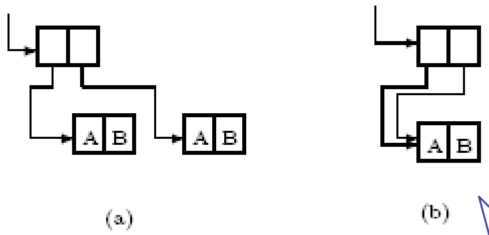
From:

Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

John McCarthy, Massachusetts Institute of Technology, Cambridge, Mass. *

April 1960





Linked list for
`(A (B C) D)`

Possible linked lists for
`((A B)(A B))`

Shared memory (no side effects)

LISP references

- John McCarthy: Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I. *Communications of the ACM* (1960)
  `http://www-formal.stanford.edu/jmc/recursive.pdf`

- John McCarthy's History of Lisp
  `http://www-formal.stanford.edu/jmc/history/lisp/lisp.html`

- Harold Abelson and Gerald Jay Sussman: "*Structure and Interpretation of Computer Programs*"
  `http://www-mitpress.mit.edu/sicp/`

- Guy L. Steele, Jr., Richard P. Gabriel's, The Evolution of Lisp
  `http://www.dreamsongs.com/NewFiles/HOPL2-Uncut.pdf`

COBOL

- COBOL: “Common Business Oriented Language” (1960)
- Idea: “Program in English”: program should be so easy to read that the management should be able to read them
- Disregarded by the scientific community
- The main language to write business oriented program ‘till the 80’s
- It introduced the notion of record

- Example: `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.
```

- Current version COBOL 2002 (with object oriented features)

The 60’s

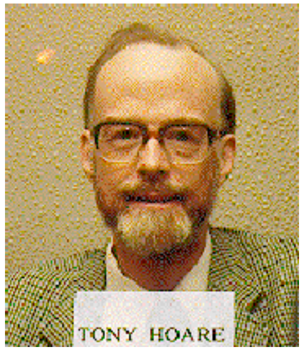
- Algol-60
- Simula-67

ALGOL-60

- ALGOL-60: “ALGOrithmic Language”
(1958-1968) by very many people IFIP(International Federation for Information Processing) , including John Backus, Peter Naur, Alan Perlis, Friedrich L. Bauer, John McCarthy, Niklaus Wirth, C. A. R. Hoare, Edsger W. Dijkstra
- Join effort by Academia and Industry
- Join effort by Europe and USA
- ALGOL-60 the most influential imperative language ever
- First language with syntax formally defined (BNF)
- First language with structured control structures
  - If then else
  - While (several forms)
  - But still goto
- First language with … (see next)
- Did not include I/O considered too hardware dependent
- ALGOL-60 revised several times in early 60’s, as understanding of programming languages improved
- ALGOL-68 a major revision
  - by 1968 concerns on data abstraction become prominent, and ALGOL-68 addressed them
  - Considered too Big and Complex by many of the people that worked on the original ALGOL-60 (*C. A. R. Hoare*’ Turing Lecture, cf. ADA later)

Edsger W. Dijkstra
(cf. shortest path, semaphore)

C. A. R. Hoare
(cf. axiomatic semantics, quicksort, CSP)

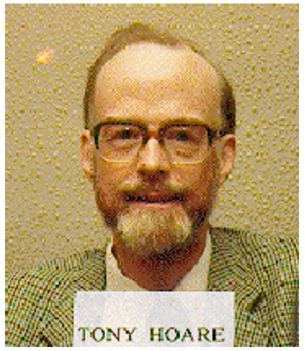


ALGOL-60

- First language with syntax formally defined (BNF)
  (after such a success with syntax, there was a great hope to being able to formally define semantics in an similarly easy and accessible way: this goal failed so far)
- First language with structured control structures
  - If then else
  - While (several forms)
  - But still goto
- First language with procedure activations based on the STACK (cf. recursion)
- First language with well defended parameters passing mechanisms
  - Call by value
  - Call by name (sort of call by reference)
  - Call by value result (later versions)
  - Call by reference (later versions)
- First language with explicit typing of variables
- First language with blocks (static scope)
- Data structure primitives: integers, reals, booleans, arrays of any dimension; (no records at first),
- Later version had also references and records (originally introduced in COBOL), and user defined types

Edsger W. Dijkstra
(cf. shortest path, semaphore)

C. A. R. Hoare
(cf. axiomatic semantics, quicksort, CSP)



BNF of ALGOL procedure declaration

From Algol orignal paper

```
<formal parameter> ::= <identifier>

<formal parameter list> ::= <formal parameter> |
        <formal parameter list> <parameter delimiter>
        <formal parameter>

<formal parameter part> ::= <empty> | ( <formal parameter list> )

<identifier list> ::= <identifier> |
        <identifier list> , <identifier>

<value part> ::= value <identifier list> ; |
        <empty>

<specifier> ::= string | <type> | array |
        <type> array | label | switch |
        procedure | <type> procedure

<specification part> ::= <empty> | <specifier> <identifier list> ; |
        <specification part> <specifier> <identifier list>

<procedure heading> ::= <procedure identifier> <formal parameter part> ;
        <value part> <specification part>

<procedure body> ::= <statement> | <code>

<procedure declaration> ::= procedure <procedure heading>
        <procedure body> | <type> procedure
        <procedure heading> <procedure body>
```

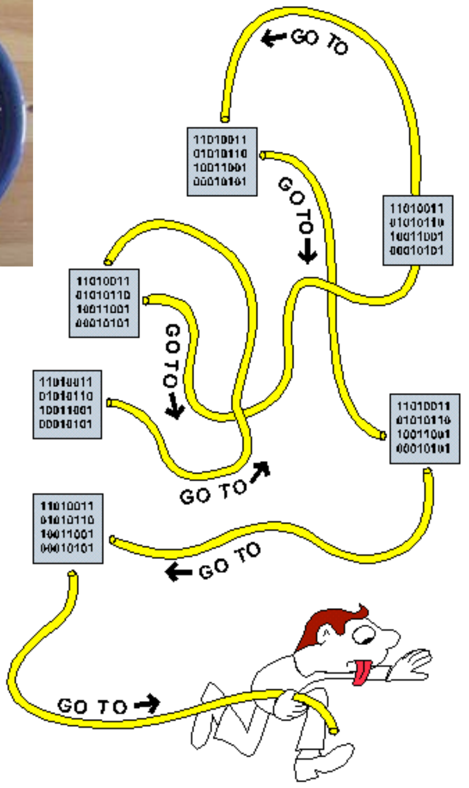
ALGOL example

```
procedure Spur (a) Order: (n); value n;
array a; integer n; real s;
begin integer k;
s:=0;
for k:=1 step 1 until n do
  s:=s+a[k,k]
end

procedure Transpose (a) Order: (n);
   value n;
array a; integer n;
begin real w; integer i, k;
for i := 1 step 1 until n do
    for k := 1+i step 1 until n do
        begin w:=a[i,k];
              a[i,k]:=a[k,i];
              a[k,i]:=w
        end
end Transpose

integer procedure Step (u); real u;
Step:=if 0 ≤ u ∧ u ≤ 1 then 1 else 0
```

From Algol-60 orignal paper

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

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;

array a; integer n, m, i, k; real y;
begin integer p, q;
  y := 0;
  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

procedure Innerproduct (a, b)
   Order: (k, p) Result: (y); value k;
integer k, p; real y, a, b;
s:=0;
for p:=1 step 1 until k do
  s:=s+a × b;
y:=s
end Innerproduct
```

ALGOL-60: structured programming



- Spagetti code
- **1966** Corrado Boehm & Giuseppe Jacopini ("La Sapienza"): prove that the only control structures needed are **if-then-else** and **while**
- **1968** Edsger Dijkstra: "Goto is harmful"
- **1974** Donald Knuth: shows good use of goto
- **1995** Java does not include goto!

```
10 dim i
20 i = 0
30 i = i + 1
40 if i <> 10 then goto 90
50 if i = 10 then goto 70
60 goto 30
70 print "Program Completed."
80 end
90 print i; " squared = "; i * i
100 goto 30
```

Spagetti code in Basic

Structured code in Basic

```
dim i
for i = 1 to 10
    print i; " squared = "; i * i
next
print "Program Completed."
```

ALGOL references

- Peter Naur (ed): Revised Report on the Algorithmic Language Algol-60 In Communications of the ACM, 6(1):1--17, 1963. **`http://www.masswerk.at/algol60/report.htm`**
- Edsger W. Dijkstra: Recursive programming. In Saul Rosen, editor, Programming Systems and Languages, chapter 3C, pages 221-227. McGraw-Hill, New York, 1960. Introduces the STACK.
- Niklaus Wirth and C. A. R. Hoare: A contribution to the development of ALGOL. Communications of the ACM, 9(6):413-432, June 1966.

  Introdueces records and references and call by result and value result.
- C. A. Hoare: The emperor's old clothes, ACM Turing Award Lecture 1980 **`http://www.braithwaite-lee.com/opinions/p75-hoare.pdf`**
- C. Boehm and G. Jacopini: Flow diagrams, Turing machines, and languages with only two formation rules. Communications of the ACM, 9(5):366-371, 1966.
- Edsger W. Dijkstra: Go to statement considered harmful. 11(3):147-148, March 1968.
- Donald E. Knuth: Structured programming with 'go to' statments. Communications of the ACM, 6(4):261-301, 1974.
- E. W. Dijkstra: Notes on Structured Programming EWD249 (1972). **`http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF`**
- Alan J. Perlis: Epigrams on Programming SIGPLAN Notices Vol. 17, No. 9, September 1982 **`http://www-pu.informatik.unituebingen.de/users/klaeren/epigrams.html`**

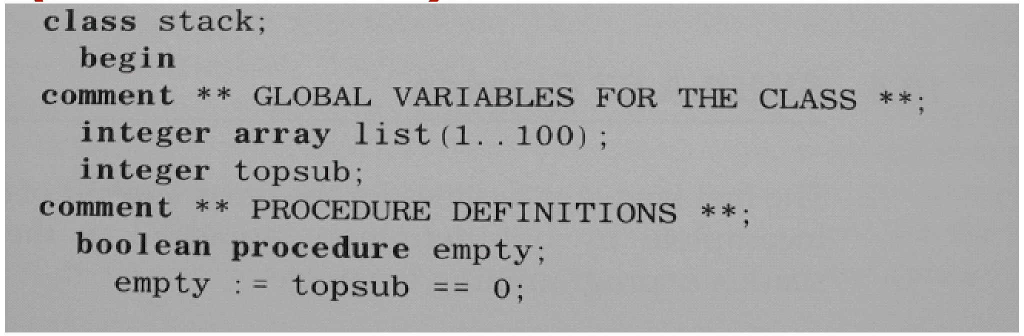
Simula-67

- Simula-67 (1967) Ole-Johan Dahl and Kristen Nygaard @ Norwegian Computing Centre in Oslo
- Originally realized for discrete-event simulation, is the first object oriented language
- Based on Algol
- Introduces for the first time:
  - Classes
  - Objects (instances of classes)
  - ISA
  - Virtual functions
  - But, no multiple inheritance
- At first was the inspiration for introducing data abstraction mechanisms in all subsequent languages, but not object orientation, whose value was understood much later.
- The idea of merging the concept of type with that of Class by Dahal, was disregarded
- Surprisingly similar to modern Object-Oriented languages!

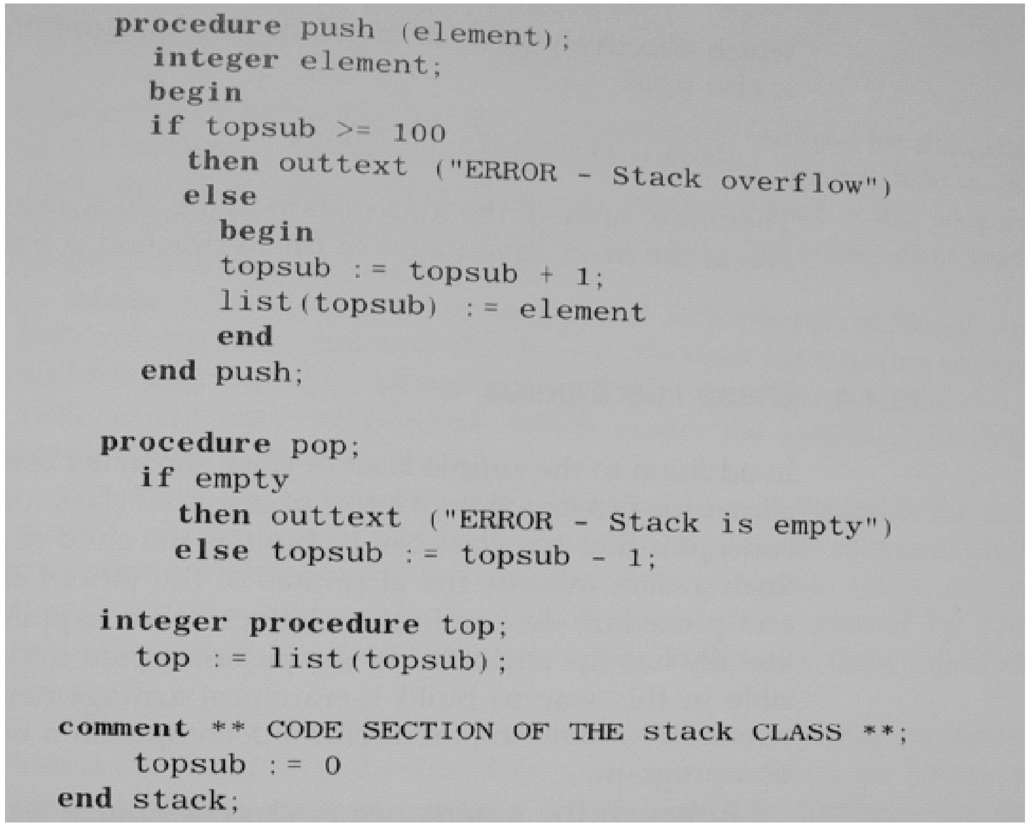
Ole-Johan Dahl (1978)

Kristen Nygaard (1978)

Simula-67 example (definition)

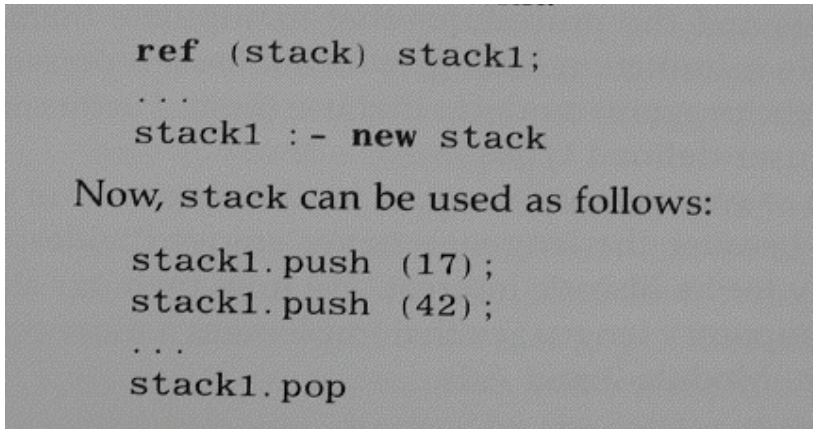


```
class stack;
  begin
comment ** GLOBAL VARIABLES FOR THE CLASS **;
  integer array list(1..100);
  integer topsub;
comment ** PROCEDURE DEFINITIONS **;
  boolean procedure empty;
    empty := topsub == 0;
```



```
    procedure push (element);
      integer element;
      begin
      if topsub >= 100
        then outtext ("ERROR - Stack overflow")
        else
          begin
          topsub := topsub + 1;
          list(topsub) := element
          end
     end push;

   procedure pop;
     if empty
       then outtext ("ERROR - Stack is empty")
       else topsub := topsub - 1;

   integer procedure top;
     top := list(topsub);

 comment ** CODE SECTION OF THE stack CLASS **;
     topsub := 0
 end stack;
```

Simula-67 example (use)



```
ref (stack) stack1;
...
stack1 :- new stack
```

Now, stack can be used as follows:

```
stack1.push (17);
stack1.push (42);
...
stack1.pop
```

Simula-67 references

- Ole-Johan Dahl and Kristen Nygaard: How Object-Oriented Programming Started **`http://heim.ifi.uio.no/~kristen/FORSKNINGSDOK_MAPPE/F_OO_start.html`**
- Dahl, Ole-Johan and Nygaard, Kristen: SIMULA--an ALGOL-Based Simulation Language, Comunications o the ACM, v.9, n.9 (September 1966), pp. 671-678.
- In memory of Ole-Johan Dahl and Kristen Nygaard **`http://www.jot.fm/issues/issue_2002_09/eulogy`**

The 70's

- Pascal
- Prolog
- Smalltalk
- C

Pascal

- Pascal (1970) Niklaus Wirth @ ETH Zurich (Swiss Federal Institute of Technology in Zurich)
- Pascal is a cleaned and simplified version of Algol developed for teaching
- Wirth: Algorithms + Data Structures = Programs
- Incredible success: 'till '80 was considered world wide THE language to lean programming on
- Originally, not suitable for applications because of low support for modularity and separate compilations
- Technically excellent, but the spec. contain an ambiguity: equivalence by structure vs equivalence by name is not settled (eq. by name considered a better solution)
- PASCAL (ISO 1983)
- UCSD Pascal (1978), Kenneth Bowles @ UCSD, introduced UNITS for modularity and separate compilation, it also introduced the virtual p-machine from a PhD Thesis of a Wirth's student: Urs Ammann (cf, JVM in Java)
- Parts of the original Apple Macintosh operating system were written in variant of UCSD Pascal
- Turbo Pascal (1983) Anders Hejlsberg @ Boreland: uses Units but without making Pascal a viable programming language for large applications – many PC applications were written in TP
- Delphi (1995) Anders Hejlsberg @ Boreland (object-oriented variant of Pascal, still used for big applications)
- Now Anders Hejlsberg is the chief architect of Microsoft C#!

Niklaus Wirth
(in the '60s)

Niklaus Wirth
(today)

Pascal example

```
program mine(output);
  var i : integer;

  procedure print(var j: integer);

    function next(k: integer): integer;
    begin
      next := k + 1
    end;

  begin
    writeln('The total is: ', j);
    j := next(j)
  end;

begin
  i := 1;
  while i <= 10 do print(i)
end.
```

Pascal references

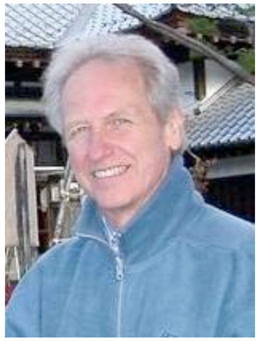
- Niklaus Wirth: The Programming Language Pascal. Acta Informatica, 1, (Jun 1971) 35-63
- Niklaus Wirth & Kathy Jensen: PASCAL - User Manual and Report. Springer-Verlag, 1974.
- Niklaus Wirth: Algorithms + Data Structures = Programs Prentice-Hall 1975
- Niklaus Wirth, M. Broy, ed, and E. Denert, ed: Pascal and its Successors in Software Pioneers: Contributions to Software Engineering. Springer-Verlag, 2002, **`http://www.swissdelphicenter.ch/en/niklauswirth.php`**
- Niklaus Wirth: Recollections about the Development of Pascal. ACM SIGPLAN Notices, Volume 28, No 3, March 1993.
- Brian W. Kernighan (cf. C): Why Pascal is Not My Favorite Programming Language (AT&T 1981) **`http://www.lysator.liu.se/c/bwk-on-pascal.html`**
- Anders Hejlsberg: Dr. Dobb's Excellence in Programming Award (2001) **`http://www.ddj.com/184404602`**

PROLOG

Alain Colmerauer

- Prolog: “PROgrammation en LOGique” (1972) by Alain Colmerauer (with Philippe Roussel) @ U. Marsille, & Robert Kowaski @ Imperial College, Lon
- Based on Logic - Predicate Calculus (Horn clauses in particular)
- Kowalski: “Algorithms = Logic + Control”
- Introduced Logic Programming (still the best known logic programming language)
- 2° most used language in AI (first is still LISP)
- Used for developing industrial strength expert systems
- Best known version
  - Edinburgh Prolog (1982)
  - ISO Prolog (ISO 1995)
- Parallel variant at the base of Fifth Generation Computer S Project (Japan 1982-1993)
- Currently many attempts to merge with constrain programming

Bob Kowalski



PROLOG

- Similar to functional programming, but based on relation instead of functions
  - input and output interchangeable
  - single input many output
- A single data structure: the term (as in Logic)
- Built in backtracking
- Very easy to write program that perfotm complex computations
- Procedural and declarative semantics

Study of prolog negation (aka negation as failure) were one of the motivation for studying nonmonotonic reasoning in AI

PROLOG example

```
eval(v(X)) :- true(v(X)).

eval(and(A,B)) :-
  eval(A), eval(B).

eval(or(A,B) :- eval(A).
eval(or(A,B) :- eval(B).

eval(not(A) :- not eval(A).

true(v(p1)).
true(v(p3)).

//eval: (p1 and p2) or p3
?- eval(or(and(v(p1),v(p2)),v(p3)).
true.
?- eval(v(p3)).
false.
```

```
sat(A) :- eval(A).

true(v(p1)).           //true.
true(v(p1)) :- fail.  //false

true(v(p2)).           //true.
true(v(p2)) :- fail.  //false

true(v(p3)).           //true.
true(v(p3)) :- fail.  //false

//sat: ((p1 and p2) and not
   p2)
?- sat(and(and(v(p1),v(p2)),
        not(v(p1))).
false.
```

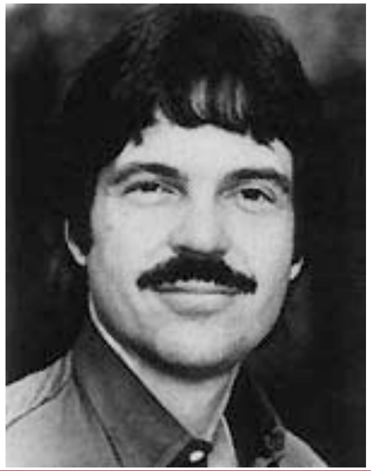
PROLOG references

- Alain Colmerauer and Philippe Roussel: The birth of Prolog, in The second ACM SIGPLAN conference on History of programming languages, p. 37-52, 1992. **`http://www.lim.univ-mrs.fr/%7Ecolmer/ArchivesPublications/HistoireProlog/19november92.pdf`**
- Robert Kowalski: The Early Years of Logic Programming, CACM January 1988.
- David H. D. Warren: Logic Programming and Compiler Writing. Softw., Pract. Exper. 10(2): 97-125 (1980)
- John W. Lloyd: Foundations of Logic Programming. Springer, Berlin, 1988.
- Keith L. Clark: Negation as Failure. Logic and Data Bases 1977: 293-322

Smalltalk

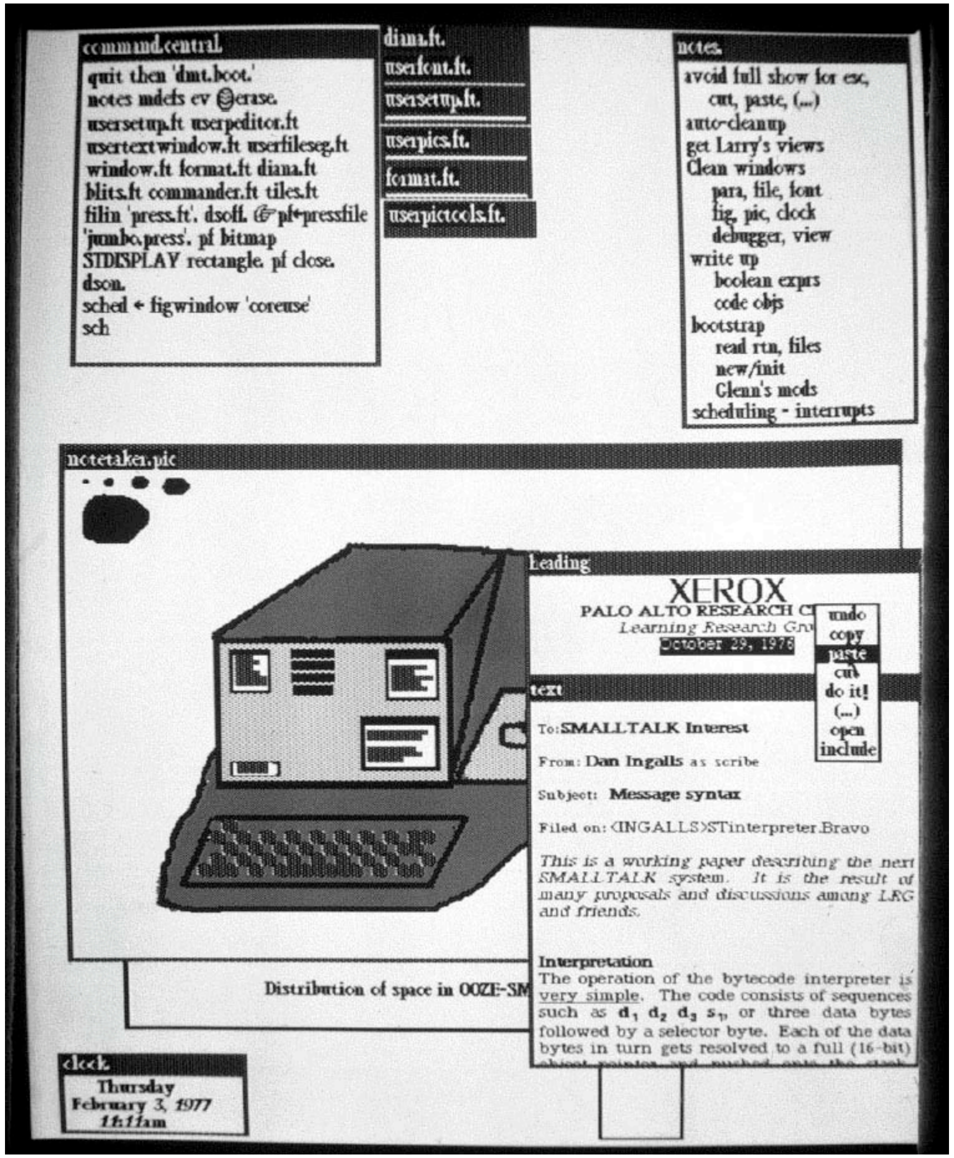
- Smalltalk (1972) by Alan Kay & Dan Ingalls @ Xerox PARC (Palo Alto Center of Research)
- Inspired by Simula67, but takes the radical view that everything is an object, thus forcing thinking in object oriented terms.
- Messages and communicating objects metaphor
- Introduces reflection (ie, runtime representation of objects and classes)
- Dynabook
- First graphical interface ever
  - Windows
  - Mouse
- Best known version Smalltalk-80
- No multiple inheritance

Alan Kay



Dan Ingalls

Smalltalk user interface example



```
command.central
quit then 'dmt.boot.'
notes mdefs ev @erase.
usersetup.ft userpeditor.ft
usertextwindow.ft userfileseg.ft
window.ft format.ft diana.ft
blits.ft commander.ft tiles.ft
filin 'press.ft'. dsoff. & pf←pressfile
'jumbo.press'. pf bitmap
STDISPLAY rectangle. pf close.
dson.
sched ← figwindow 'coreuse'
sch
```

```
diana.ft.
userfont.ft.
usersetup.ft.
userpics.ft.
format.ft.
userpictools.ft.
```

```
notes.
avoid full show for esc,
    cut, paste, (...)
auto-cleanup
get Larry's views
Clean windows
    para, file, font
    fig, pic, clock
    debugger, view
write up
    boolean exprs
    code objs
bootstrap
    read rtn, files
    new/init
    Glenn's mods
scheduling - interrupts
```

notetaker.pic

heading

XEROX
PALO ALTO RESEARCH C
Learning Research Gr
October 29, 1976

text

To: **SMALLTALK Interest**

From: **Dan Ingalls** as scribe

Subject: **Message syntax**

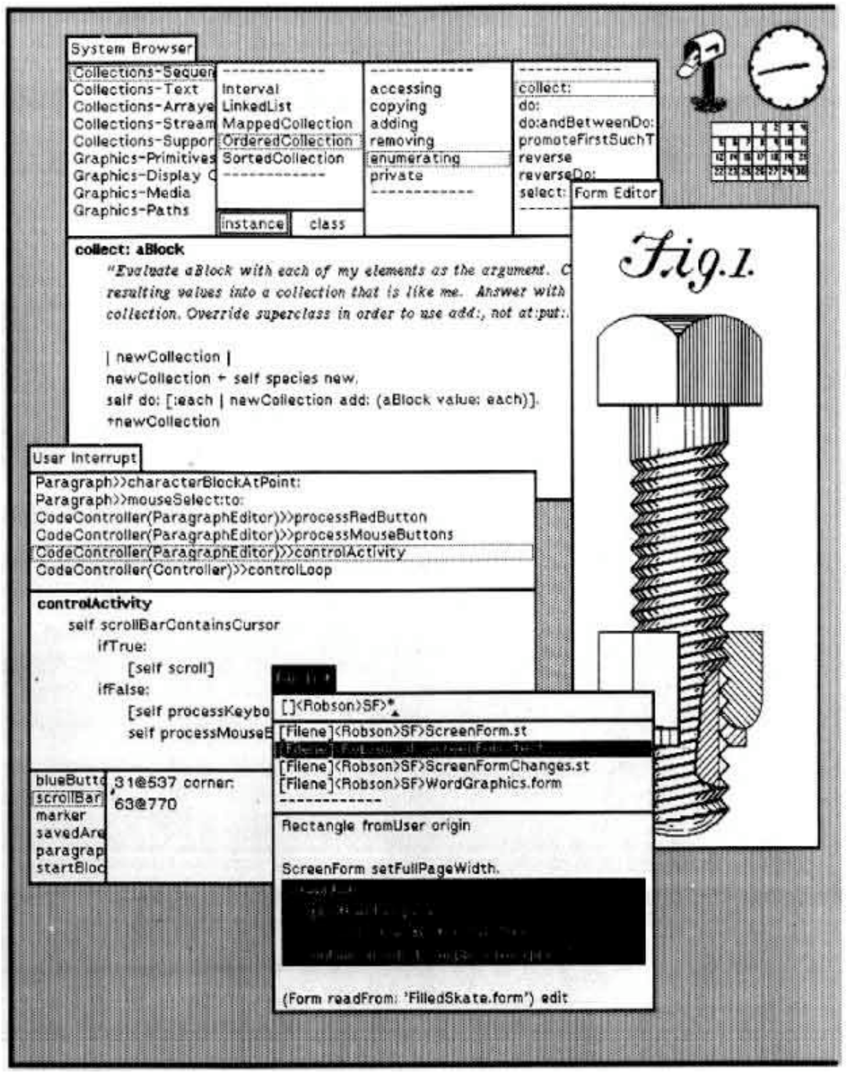
Filed on: <INGALLS>STinterpreter.Bravo

This is a working paper describing the next SMALLTALK system. It is the result of many proposals and discussions among LRG and friends.

Interpretation
The operation of the bytecode interpreter is very simple. The code consists of sequences such as $\mathbf{d_1}$ $\mathbf{d_2}$ $\mathbf{d_3}$ $\mathbf{s_p}$, or three data bytes followed by a selector byte. Each of the data bytes in turn gets resolved to a full (16-bit)

Distribution of space in OOZE-SM

clock
Thursday
February 3, 1977
11:11am



System Browser

```
Collections-Sequen    ------------          ------------     ------------
Collections-Text      Interval              accessing        collect:
Collections-Arraye    LinkedList            copying          do:
Collections-Stream    MappedCollection      adding           do:andBetweenDo:
Collections-Suppor    OrderedCollection     removing         promoteFirstSuchT
Graphics-Primitives   SortedCollection      enumerating      reverse
Graphics-Display C    ------------          private          reverseDo:
Graphics-Media                              ------------     select:
Graphics-Paths        instance    class
```

```
collect: aBlock
    "Evaluate aBlock with each of my elements as the argument. C
    resulting values into a collection that is like me. Answer with
    collection. Override superclass in order to use add:, not at:put:.

    | newCollection |
    newCollection ← self species new.
    self do: [:each | newCollection add: (aBlock value: each)].
    ↑newCollection
```

Form Editor

User Interrupt

```
Paragraph>>characterBlockAtPoint:
Paragraph>>mouseSelect:to:
CodeController(ParagraphEditor)>>processRedButton
CodeController(ParagraphEditor)>>processMouseButtons
CodeController(ParagraphEditor)>>controlActivity
CodeController(Controller)>>controlLoop
```

```
controlActivity
    self scrollBarContainsCursor
        ifTrue:
            [self scroll]
        ifFalse:
            [self processKeybo
            self processMouseB
```

```
blueButt    31@537 corner:
scrollBar   63@770
marker
savedAre
paragrap
startBloc
```

```
[]<Robson>SF>*
[Filene]<Robson>SF>ScreenForm.st
[illegible]
[Filene]<Robson>SF>ScreenFormChanges.st
[Filene]<Robson>SF>WordGraphics.form
------------
Rectangle fromUser origin

ScreenForm setFullPageWidth.
[illegible]

(Form readFrom: 'FilledSkate.form') edit
```

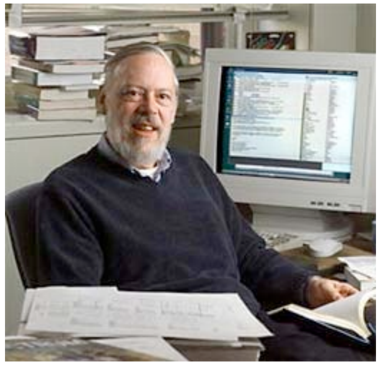
Smalltalk references

- Smalltalk site: **`http://www.smalltalk.org`**
- Alan C. Kay: The Early History of Smalltalk, Apple Computer 1993 **`http://gagne.homedns.org/%7etgagne/contrib/EarlyHistoryST.html`**
- Daniel H. H. Ingalls: Design Principles Behind Smalltalk, BYTE Magazine, August 1981. **`http://users.ipa.net/~dwighth/smalltalk/byte_aug81/design_principles_behind_smalltalk.html`**
- On the Dynabook: **`http://www.artmuseum.net/w2vr/archives/Kay/01_Dynabook.html#top`**
- Dynabook revised: an interview with Alan Kay **`http://www.squeakland.org/school/HTML/essays/dynabook_revisited.htm`**
- Alan C. Kay: First Courses in Computing Should be Child's Play, ACM Turing Award lecture 2004 **`http://www.cs.uni.edu/~wallingf/miscellaneous/alan-kay/turing-transcript.html`**

C

- C (1978) Dennis Ritchie & Brian Kernighan @ AT&T
- For programming UNIX operative system
- Similar to Pascal but simplified in many ways a language (eg, no nested definitions
- Modularization and separate compilation supported via separate files
- IO on library
- Dynamic memory management on library
- Extensive library collections
- Very efficient compiled code better that Fortran’s similar to Assembler
- No concurrency in the language but access to operative systems mechanisms
- Data structure integers, reals, arrays, records, references
- Explicit manipulation of references
- Call by value only (but of references as well)
- Java syntax for basic instruction is taken from C
- ANSI C (1988)

Dannis Ritchie



Brian Kernighan

C example

```
#include <stdio.h>

int main()
{
  int nums[5] = {20, 31, 17, 47, 14};
  int nums_length = 5;

  printf("Unsorted Array:\n");
  display_array(nums, nums_length);

  bubble_sort(nums, nums_length);

  printf("Sorted Array:\n");
  display_array(nums, nums_length);
  return 0;
}

/* Function that simply displays each element of
input_array. */
void display_array(int input_array[], int
input_size)
{
  int i;

  for (i = 0; i < input_size; i++)
  {
    printf("%d ", input_array[i]);
  }
  printf("\n\n");
}
```

```
/* Bubble sort function. Sorts an array of ints
in descending order. */
void bubble_sort(int array[], int arrayLength)
{
  int i, j, flag = 1;
  int temp;

  for(i = 1; (i <= arrayLength) && flag; i++)
  {
     flag = 0;
     for(j = 0; j < (arrayLength - i); j++)
     {
       if (array[j + 1] > array[j])
       {
          temp = array[j + 1];
          array[j + 1] = array[j];
          array[j] = temp;
          flag = 1;
       }
     }
  }
}
```

C references

- Brian Kernighan, Dennis Ritchie: The C Programming Language. Also known as K&R — The original book on C.
  - 1st, Prentice Hall 1978; ISBN 0-131-10163-3. Pre-ANSI C.
  - 2nd, Prentice Hall 1988; ISBN 0-131-10362-8. ANSI C.

- Samuel P. Harbison, Guy L. Steele: C: A Reference Manual.
  - 4th, Prentice Hall 1994; ISBN 0-133-26224-3.
  - 5th, Prentice Hall 2002; ISBN 0-130-89592-X.

- The Development of the C Language by Dennis M. Ritchie
  `http://cm.bell-labs.com/cm/cs/who/dmr/chist.html`

- Dennis M. Ritchie's homepage
  `http://cm.bell-labs.com/cm/cs/who/dmr/`

The 80’s

- Modula-2
- Ada

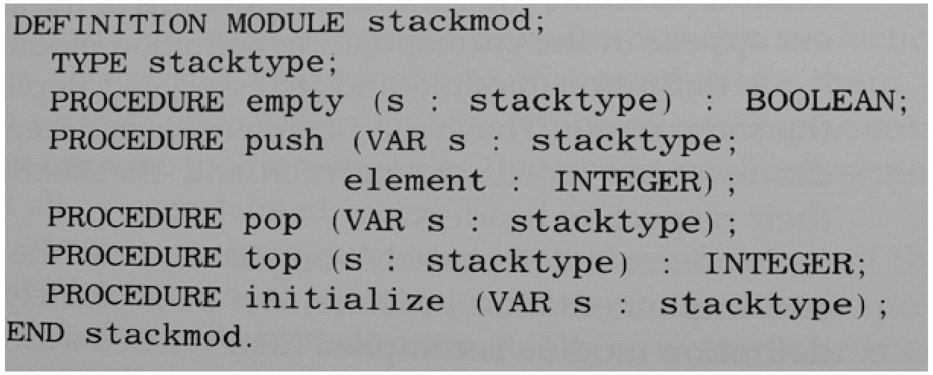
Dipartimento di Informatica e Sistemistica "Antonio Ruberti"

SAPIENZA UNIVERSITÀ DI ROMA

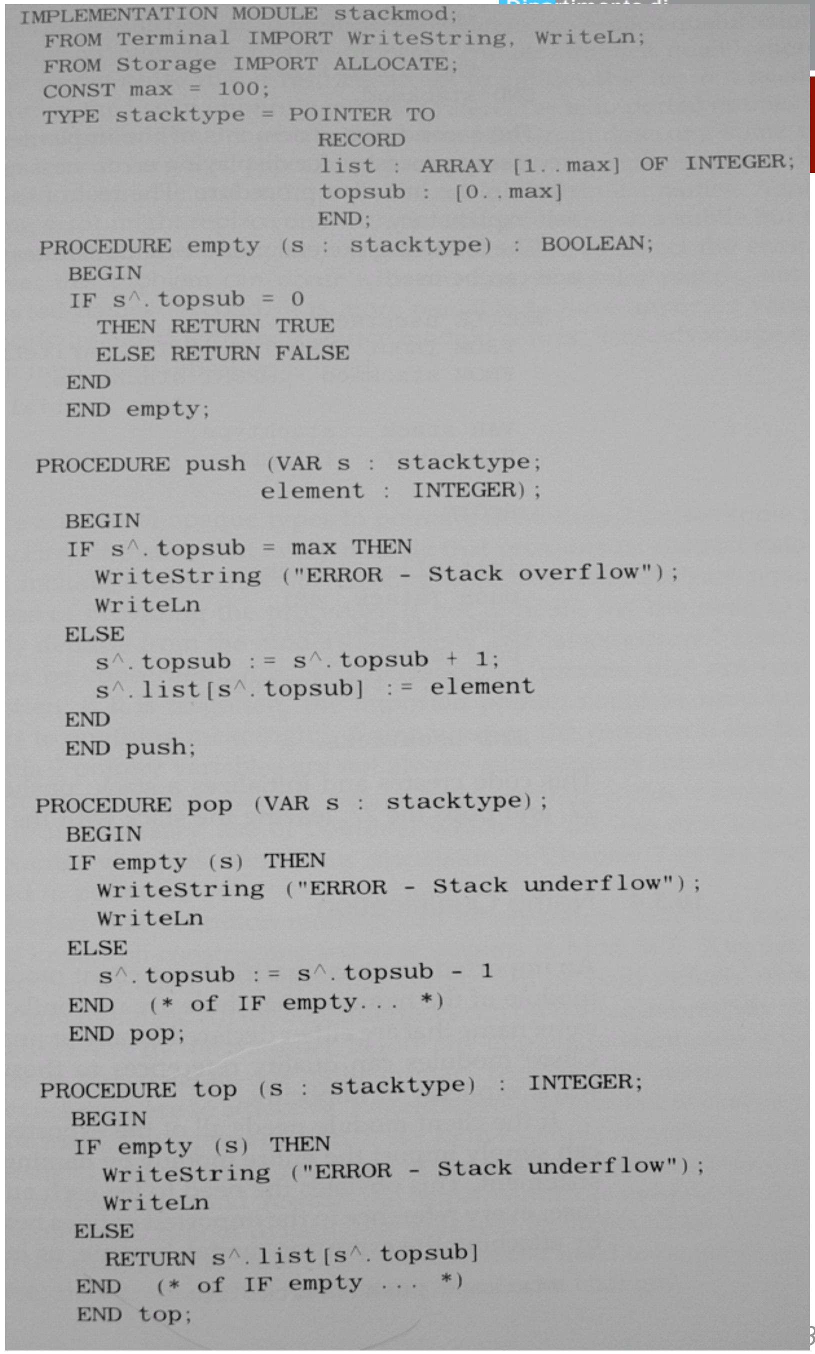
Modula-2

- Modula-2 (1980) Niklaus Wirth @ ETH Zurich
- Extends Pascal so as to deal with very large programs
- Introduces a very well defined and refined notion of module while keeping the language simple
- Introduces high level forms of concurrency: monitors and efficient low level access – was used as the only programming language for Lilith workstations @ ETH Zurich
- Main concerns:
  - Structured programming (as Pascal & Algol)
  - Data abstraction
  - Modularity
- Was not object oriented
- Current version (ISO 1999) extended with object oriented constructs and generics

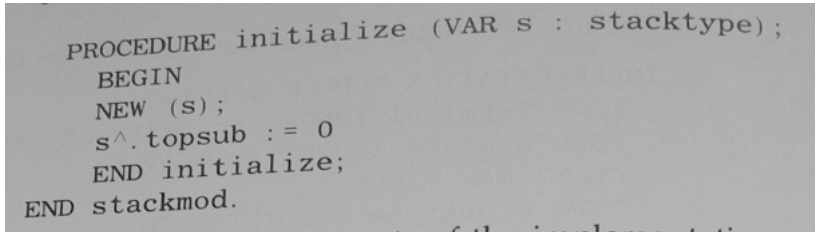
Modula-2 example (definition)



```
DEFINITION MODULE stackmod;
  TYPE stacktype;
  PROCEDURE empty (s : stacktype) : BOOLEAN;
  PROCEDURE push (VAR s : stacktype;
                  element : INTEGER);
  PROCEDURE pop (VAR s : stacktype);
  PROCEDURE top (s : stacktype) : INTEGER;
  PROCEDURE initialize (VAR s : stacktype);
END stackmod.
```

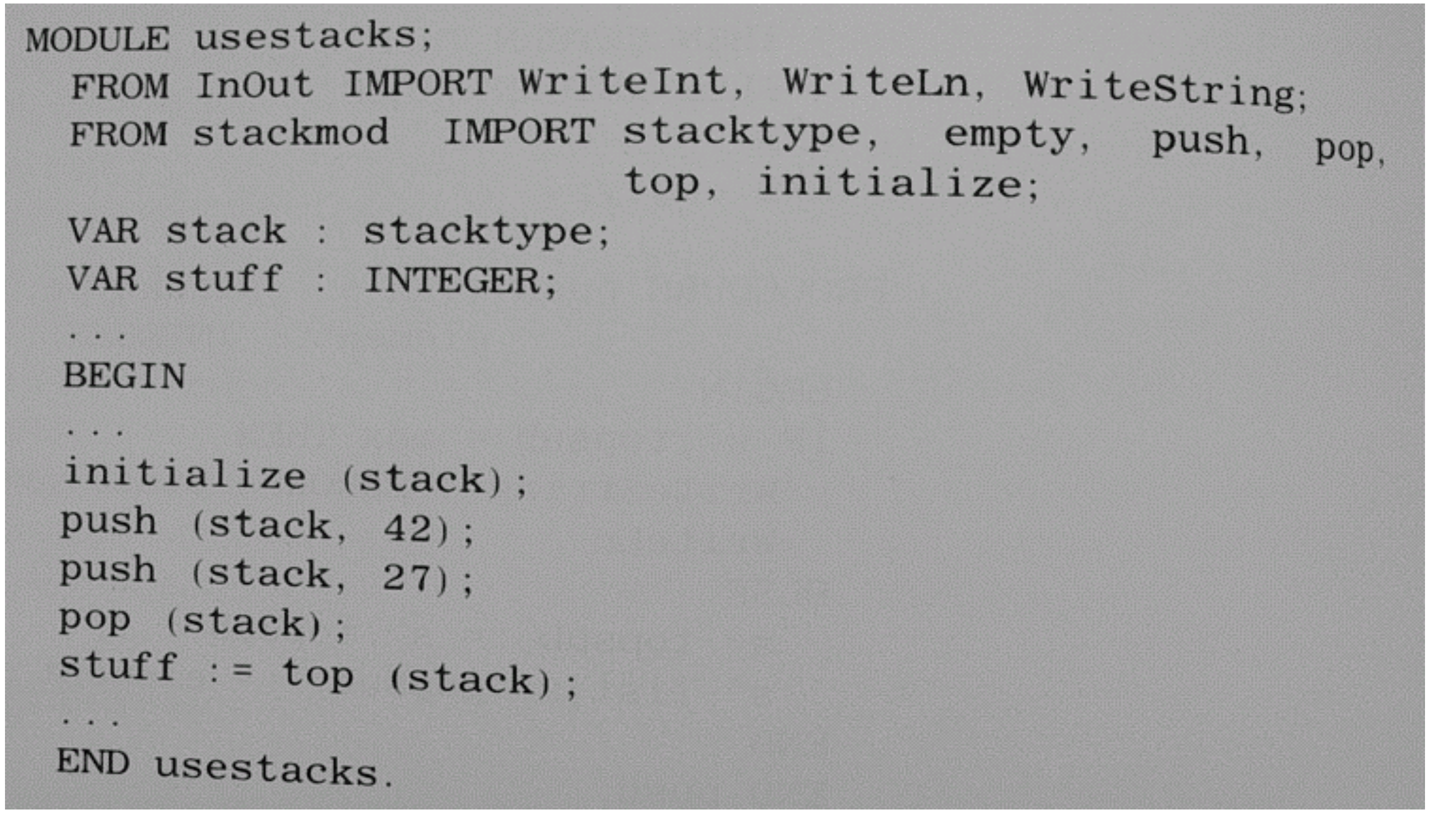


```
IMPLEMENTATION MODULE stackmod;
  FROM Terminal IMPORT WriteString, WriteLn;
  FROM Storage IMPORT ALLOCATE;
  CONST max = 100;
  TYPE stacktype = POINTER TO
                      RECORD
                      list : ARRAY [1..max] OF INTEGER;
                      topsub : [0..max]
                      END;
  PROCEDURE empty (s : stacktype) : BOOLEAN;
    BEGIN
    IF s^.topsub = 0
      THEN RETURN TRUE
      ELSE RETURN FALSE
    END
    END empty;

  PROCEDURE push (VAR s : stacktype;
                  element : INTEGER);
    BEGIN
    IF s^.topsub = max THEN
      WriteString ("ERROR - Stack overflow");
      WriteLn
    ELSE
      s^.topsub := s^.topsub + 1;
      s^.list[s^.topsub] := element
    END
    END push;

  PROCEDURE pop (VAR s : stacktype);
    BEGIN
    IF empty (s) THEN
      WriteString ("ERROR - Stack underflow");
      WriteLn
    ELSE
      s^.topsub := s^.topsub - 1
    END  (* of IF empty... *)
    END pop;

  PROCEDURE top (s : stacktype) : INTEGER;
    BEGIN
    IF empty (s) THEN
      WriteString ("ERROR - Stack underflow");
      WriteLn
    ELSE
      RETURN s^.list[s^.topsub]
    END  (* of IF empty ... *)
    END top;
```



```
  PROCEDURE initialize (VAR s : stacktype);
    BEGIN
    NEW (s);
    s^.topsub := 0
    END initialize;
END stackmod.
```

Modula-2 example (use)



```
MODULE usestacks;
  FROM InOut IMPORT WriteInt, WriteLn, WriteString;
  FROM stackmod  IMPORT stacktype,  empty,  push,  pop,
                        top,  initialize;
  VAR stack  :  stacktype;
  VAR stuff  :  INTEGER;
  ...
  BEGIN
  ...
  initialize (stack);
  push (stack, 42);
  push (stack, 27);
  pop (stack);
  stuff := top (stack);
  ...
  END usestacks.
```

Modula-2 references

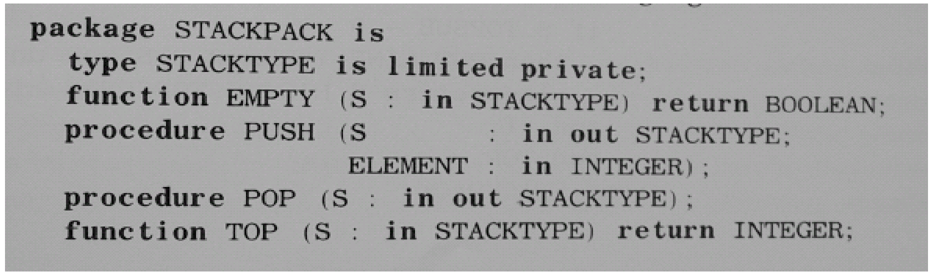
- Niklaus Wirth: Design and Implementation of Modula. Software - Practice and Experience, 7, 3-84 (1977).
- Niklaus Wirth: A Brief History of Modula and Lilith **`http://www.modulaware.com/mdlt52.htm`**
- Niklaus Wirth: ETH Report 36. The original report on Modula-2 by Niklaus Wirth, March 1980 **`http://users.ugent.be/~fschoonj/modula2/wirth-modula2/Wirth_Modula2.pdf`**
- ISO Modula-2 Language Reference **`http://www.excelsior-usa.com/doc/isom2.html`**

Ada

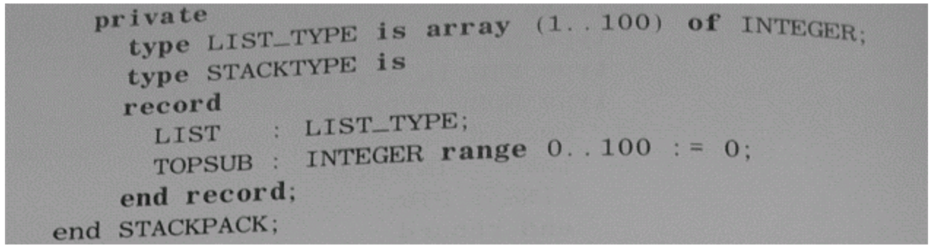
Jean David Ichbiah

- Ada (ANSI 1983) by Jean David Ichbiah @ Honeywell Bull, France and his team (and may other in fact)
- under contract by U.S. Department of Defense (DoD), and also European Economic Community through a series of competitions
- Ada stands for Ada Lovelance daughter of Lord Byron, though by Augustus De Morgan, fascinated by Charles Babbage's mechanical calculating machine, considered the first programmer.
- Main idea: being the ultimate programming language
- Based on Pascal and Algol but also Simula
- Very comprehensive
- big, for many, such as C.A.R Hoare, too big, as Algol-68
- Modularity ("packages")
- Data abstraction
- Parallel processing
- Generics
- Not object oriented
- Current version ADA95 extended with object oriented constructs

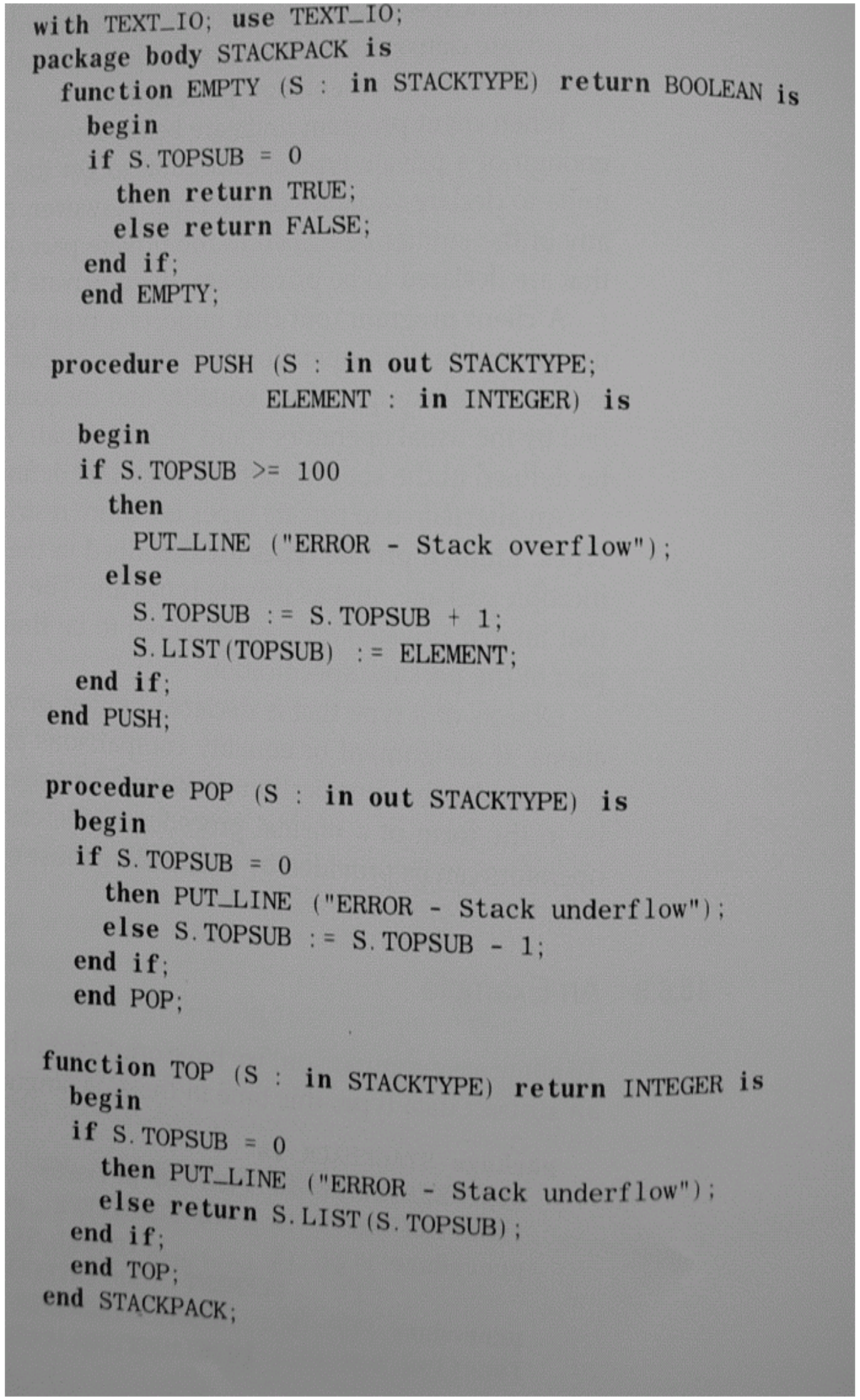
Ada example (definition)



```
package STACKPACK is
   type STACKTYPE is limited private;
   function EMPTY (S : in STACKTYPE) return BOOLEAN;
   procedure PUSH (S        : in out STACKTYPE;
                   ELEMENT : in INTEGER);
   procedure POP (S : in out STACKTYPE);
   function TOP (S : in STACKTYPE) return INTEGER;
```

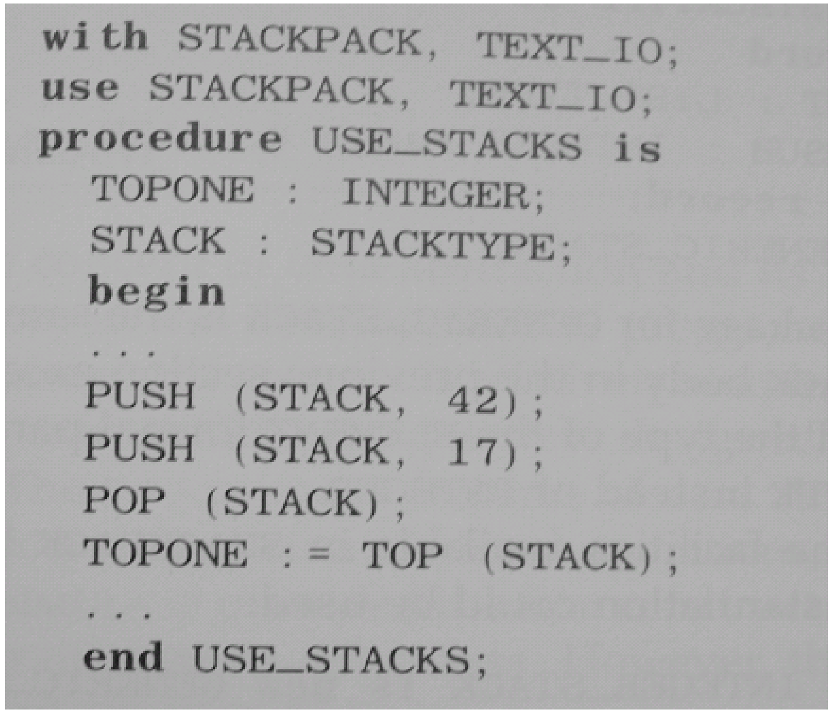


```
     private
       type LIST_TYPE is array (1..100) of INTEGER;
       type STACKTYPE is
       record
         LIST    : LIST_TYPE;
         TOPSUB  : INTEGER range 0..100 := 0;
       end record;
   end STACKPACK;
```



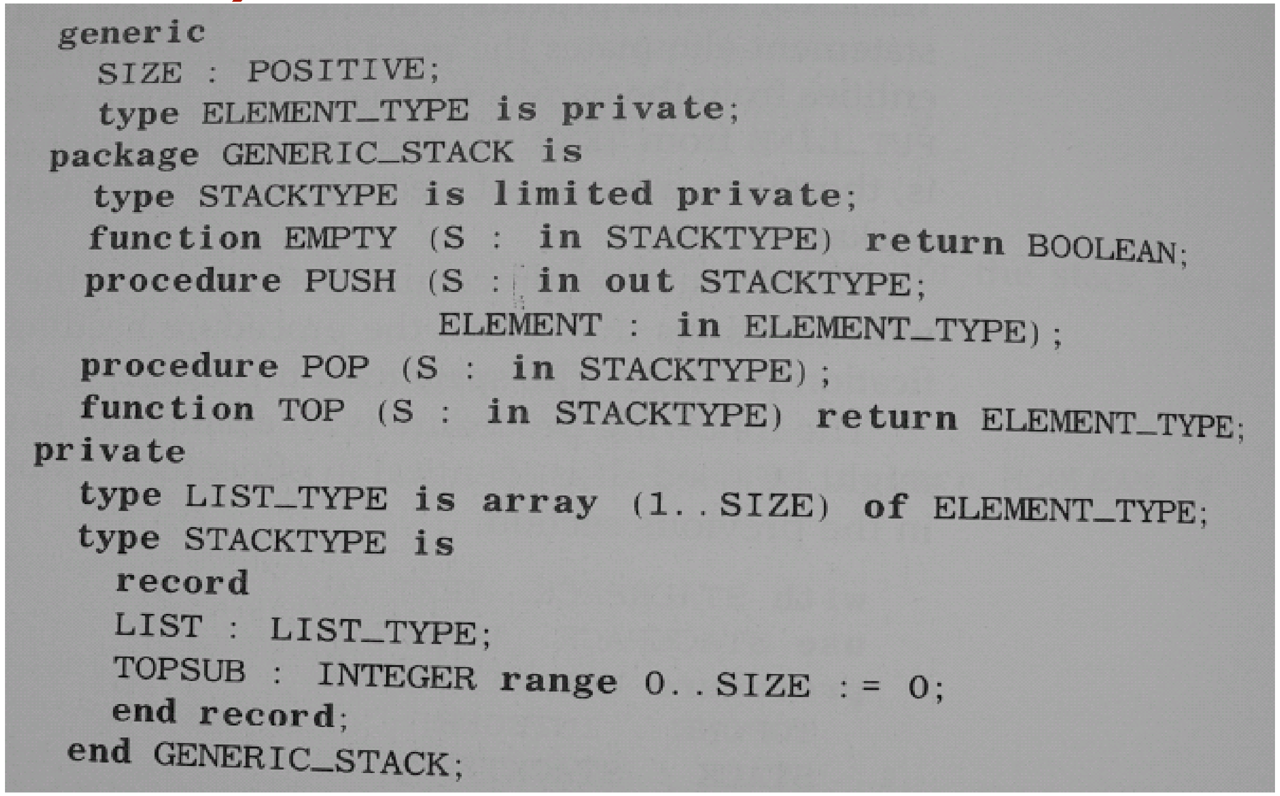
```
with TEXT_IO; use TEXT_IO;
package body STACKPACK is
  function EMPTY (S : in STACKTYPE) return BOOLEAN is
    begin
    if S.TOPSUB = 0
      then return TRUE;
      else return FALSE;
    end if;
    end EMPTY;

  procedure PUSH (S : in out STACKTYPE;
                  ELEMENT : in INTEGER) is
    begin
    if S.TOPSUB >= 100
      then
        PUT_LINE ("ERROR - Stack overflow");
      else
        S.TOPSUB := S.TOPSUB + 1;
        S.LIST(TOPSUB) := ELEMENT;
    end if;
  end PUSH;

  procedure POP (S : in out STACKTYPE) is
    begin
    if S.TOPSUB = 0
      then PUT_LINE ("ERROR - Stack underflow");
      else S.TOPSUB := S.TOPSUB - 1;
    end if;
    end POP;

  function TOP (S : in STACKTYPE) return INTEGER is
    begin
    if S.TOPSUB = 0
      then PUT_LINE ("ERROR - Stack underflow");
      else return S.LIST(S.TOPSUB);
    end if;
    end TOP;
  end STACKPACK;
```

Ada example (use)

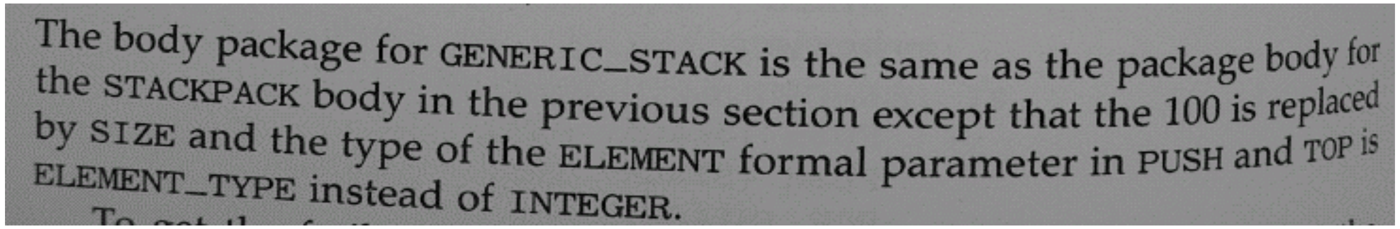


```
with STACKPACK, TEXT_IO;
use STACKPACK, TEXT_IO;
procedure USE_STACKS is
   TOPONE : INTEGER;
   STACK : STACKTYPE;
   begin
   . . .
   PUSH (STACK, 42);
   PUSH (STACK, 17);
   POP (STACK);
   TOPONE := TOP (STACK);
   . . .
   end USE_STACKS;
```

Ada generics example (definition)

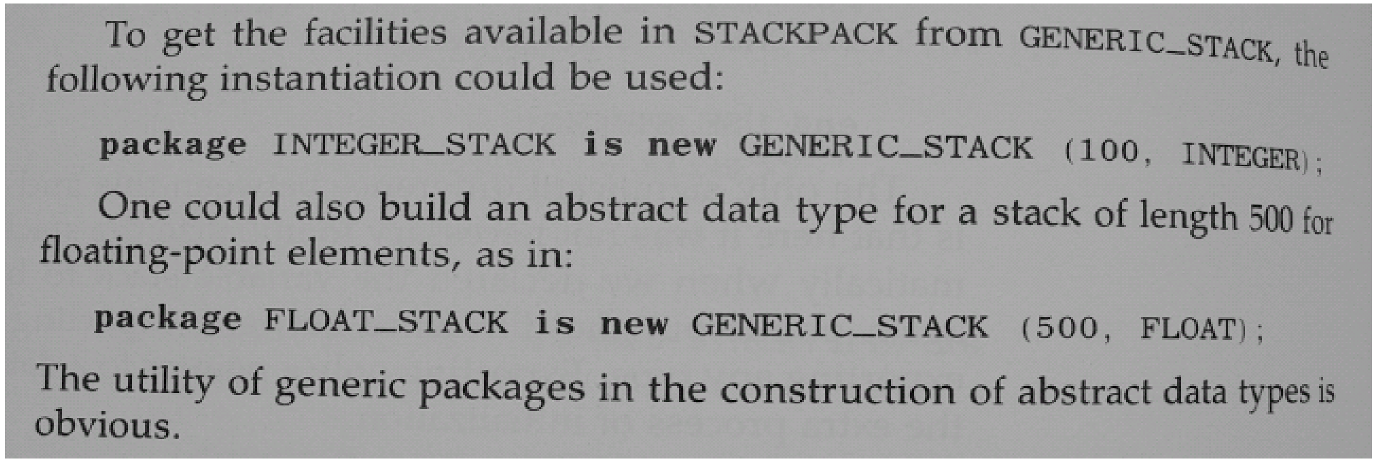


```
generic
  SIZE : POSITIVE;
  type ELEMENT_TYPE is private;
package GENERIC_STACK is
  type STACKTYPE is limited private;
  function EMPTY (S : in STACKTYPE) return BOOLEAN;
  procedure PUSH (S : in out STACKTYPE;
                  ELEMENT : in ELEMENT_TYPE);
  procedure POP (S : in STACKTYPE);
  function TOP (S : in STACKTYPE) return ELEMENT_TYPE;
private
  type LIST_TYPE is array (1..SIZE) of ELEMENT_TYPE;
  type STACKTYPE is
    record
    LIST : LIST_TYPE;
    TOPSUB : INTEGER range 0..SIZE := 0;
    end record;
end GENERIC_STACK;
```



The body package for GENERIC_STACK is the same as the package body for the STACKPACK body in the previous section except that the 100 is replaced by SIZE and the type of the ELEMENT formal parameter in PUSH and TOP is ELEMENT_TYPE instead of INTEGER.

Ada generics example (use)



To get the facilities available in STACKPACK from GENERIC_STACK, the following instantiation could be used:

```
package INTEGER_STACK is new GENERIC_STACK (100, INTEGER);
```

One could also build an abstract data type for a stack of length 500 for floating-point elements, as in:

```
package FLOAT_STACK is new GENERIC_STACK (500, FLOAT);
```

The utility of generic packages in the construction of abstract data types is obvious.

Ada references

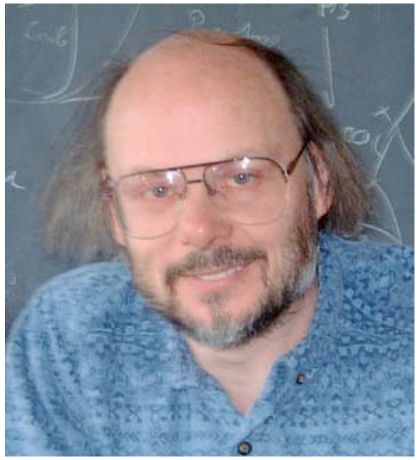
- "Ada: Past, Present, Future - An Interview with Jean Ichbiah, the Principal Designer of Ada", Communications of the ACM, Volume 27, Number 10, p. 990–997, October 1984.
- John Barnes: Programming in Ada plus Language Reference Manual, Addison-Wesley, ISBN 0-201-56539-0
- Grady Booch, Doug Bryan: Software Engineering with Ada, Addison-Wesley, 1986
- ACM SIGAda -- the Special Interest Group on Ada **`http://www.sigada.org/`**

The 90’s

- C++
- Java

C++

Bjarne Stroustrup



- C++ (1985) Bjarne Stroustrup @ AT&T
- Based on C and Simula-67
- But not SmallTalk (no reflection)
- Standard ISO 1998
- One of the best modern object oriented programming language available
- Objects can be passed as values or as references
- Multiple inheritance
- Templates (Generics)
- Most used language in applications: all Adobe products, all Microsoft products, most Apple products, etc. –see: **`http://www.research.att.com/~bs/applications.html`**

C++ example

```
// File Insieme-SS.h
#ifndef INSIEME_SS_H
#define INSIEME_SS_H

struct nodo {
  int info;
  nodo* next;
};

class Insieme
{public:
  Insieme();
  Insieme(const Insieme&);
  ~Insieme();
  Insieme& operator=(const Insieme&);
  bool EstVuoto();
  bool Membro(int);
  void Inserisci(int);
  void Elimina(int);
  int Scegli();
 private:
  nodo* inizio;
  static bool Appartiene(int,nodo*);
  static void Cancella(int,nodo*&);
  static nodo* Copia(nodo*);
  static void Rilascia(nodo*);
};
#endif
```

```
// File Insieme-SS.cpp
#include "Insieme-SS.h"

Insieme::Insieme()
{ inizio = NULL; }

Insieme::Insieme(const Insieme& t)
{ inizio = Copia(t.inizio); }

Insieme::~Insieme()
{ Rilascia(inizio); }

Insieme& Insieme::operator=(const Insieme& t)
{ if (this != &t)
     { Rilascia(inizio);
       inizio = Copia(t.inizio);
     }
  return *this;
}
…
```

C++ example

```
// File Insieme-SS.cpp
…
bool Insieme::EstVuoto()
{ return inizio == NULL; }

bool Insieme::Membro(int e)
{ return Appartiene(e,inizio); }

void Insieme::Inserisci(int e)
{ if (! Appartiene(e,inizio)) // per evitare
duplicazioni
     { nodo* temp = new nodo;
       temp->info = e;
       temp->next = inizio;
       inizio = temp;
     }
}

void Insieme::Elimina(int e)
{ Cancella(e,inizio); }

int Insieme::Scegli()
{ return inizio->info; }
…
```

```
bool Insieme::Appartiene(int e, nodo* p)
{ return (p != NULL) && ((p->info == e) ||
         (Appartiene(e,p->next)));
}

void Insieme::Cancella(int e, nodo* p) //cancella elem dalla lista
{ if (p != NULL)
     if (p->info == e)
        { nodo* q = p;
          p = p->next;
          delete q;
        }
     else Cancella(e,p->next);
}

nodo* Insieme::Copia(nodo* p)
{ if (p != NULL)
    { nodo* q = new nodo;
      q->info = p->info;
      q->next = Copia(p->next);
      return q;
    }
  else return NULL;
}

void Insieme::Rilascia(nodo* p)
{ if (p != NULL)
     { Rilascia(p->next);
       delete p;
     }
}
```

C++ references

- M. A. Ellis and B. Stroustrup: The Annotated C++ Reference Manual. Addison Wesley, ISBN 0-201-51459-1. May 1990
- B. Stroustrup: The C++ Programming Language (3rd Edition). Addison-Wesley Longman. Reading Mass. USA. 1997
- B. Stroustrup: A Perspective on ISO C++. The C++ Report. Vol 7/No 8, pp 22-28. October 1995
- The Development of the C Language by Dennis M. Ritchie
  `http://cm.bell-labs.com/cm/cs/who/dmr/chist.html`
- Bjarne Stroustrup's homepage
  `http://www.research.att.com/~bs/homepage.html`

Java

James Gosling

- Java (1995) James Gosling and others @ Sun
- Based on C, C++, Simula-67, Smalltalk
- Internet
- Java Virtual Machine
- Reflection (class can be checked at runtime)
- Objects can denoted only by reference variables
- Garbage collection
- Single inheritance
- Extensive Libraries, eg. Collection Framework, JDBC, etc.
- Generics from Java 1.5 (2004)
- Currently the most used language in enterprise applications

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

Java references

- Java home site:
  `http://java.sun.com/`
- The Java Language: an Overview (1995).
  `http://java.sun.com/docs/overviews/java/java-overview-1.html`
- Java Technology: The Early Years (2005)
  `http://java.sun.com/features/1998/05/birthday.html`
- Using and Programming Generics in J2SE 5.0 (2004)
  `http://java.sun.com/developer/technicalArticles/J2SE/generics/ind`
  `ex.html`

An additional references

- Robert W. Sebesta. Concepts of Programming Languages (7th Edition), Addison Wesley, 2005.
- Turing Award Lectures (many people mentioned in these slides were awared)
  `http://awards.acm.org/turing/`