History and Spirit of C

Olve Maudal, Cisco Systems

45 minute session at NDC TechTown, Kongsberg, 19. October 2017

The history of C in 90 seconds

http://thesourceny.com/welcome/wp-content/uploads/2014/02/stage-red-curtain.jpg

At Bell Labs.

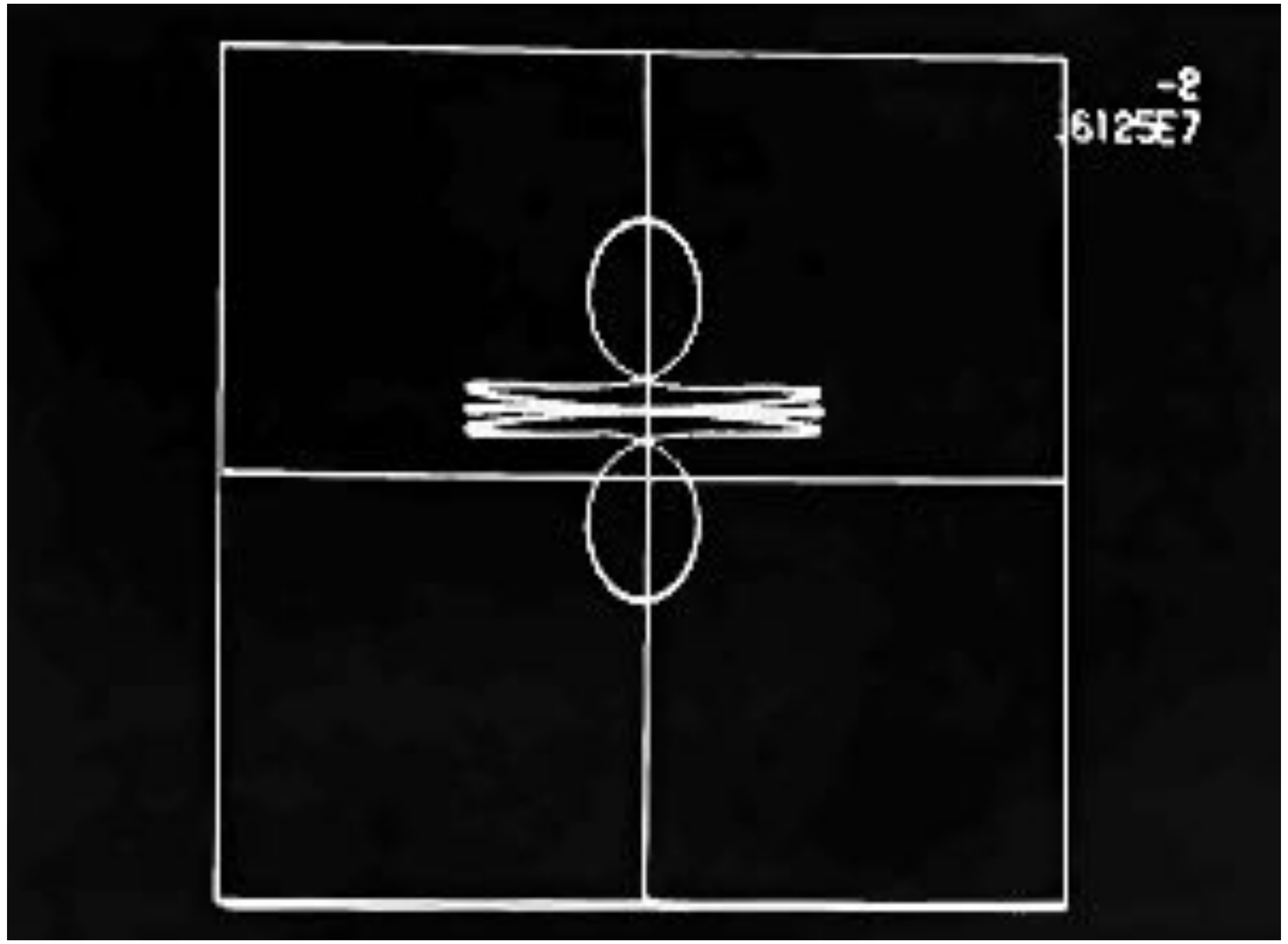
http://www3.nd.edu/~atrozzol/BellLabs1959.jpg

Back in 1969.

http://www.multicians.org/picnics.html

Ken Thompson wanted to play.

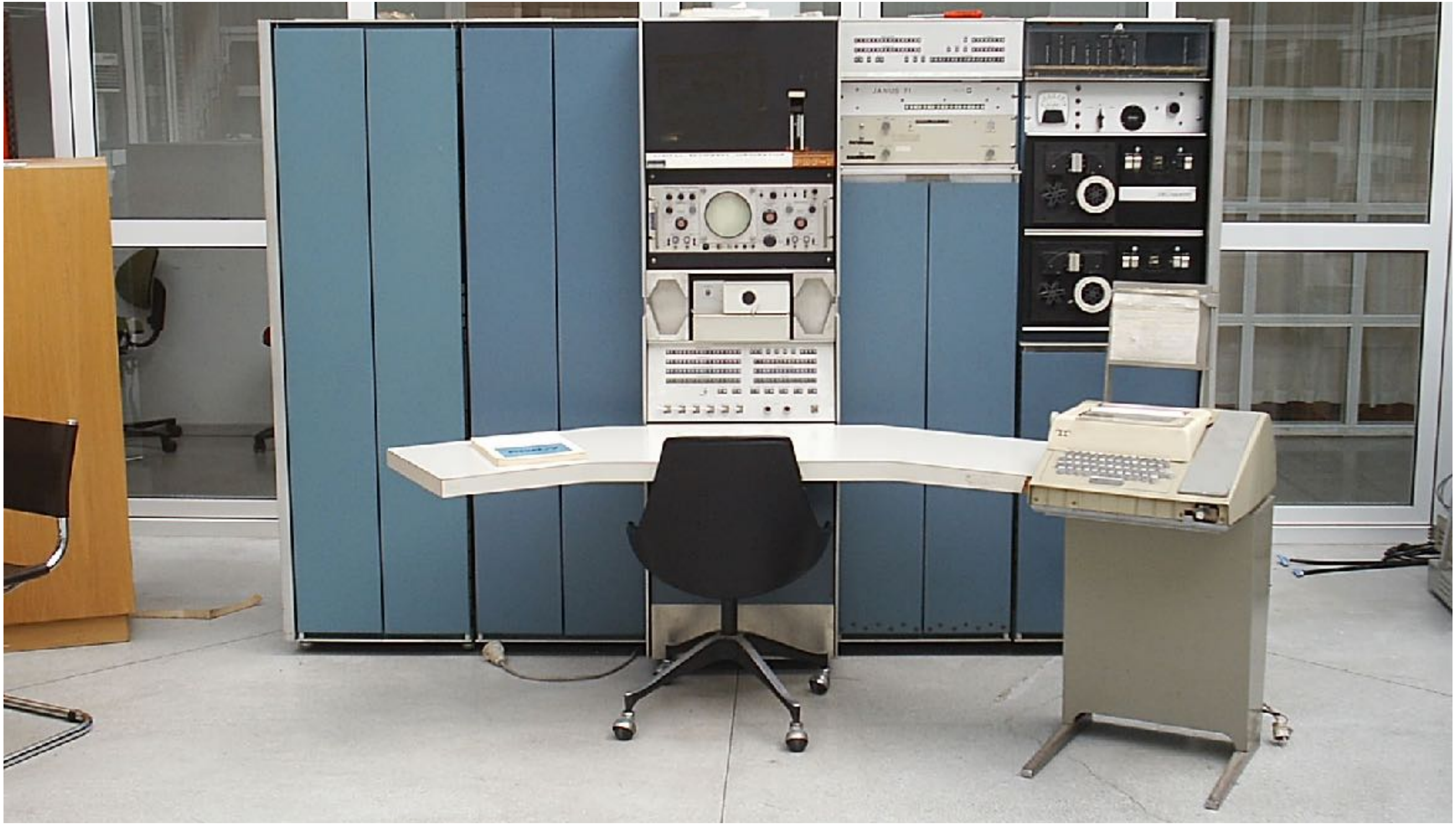
Ken



Space Travel

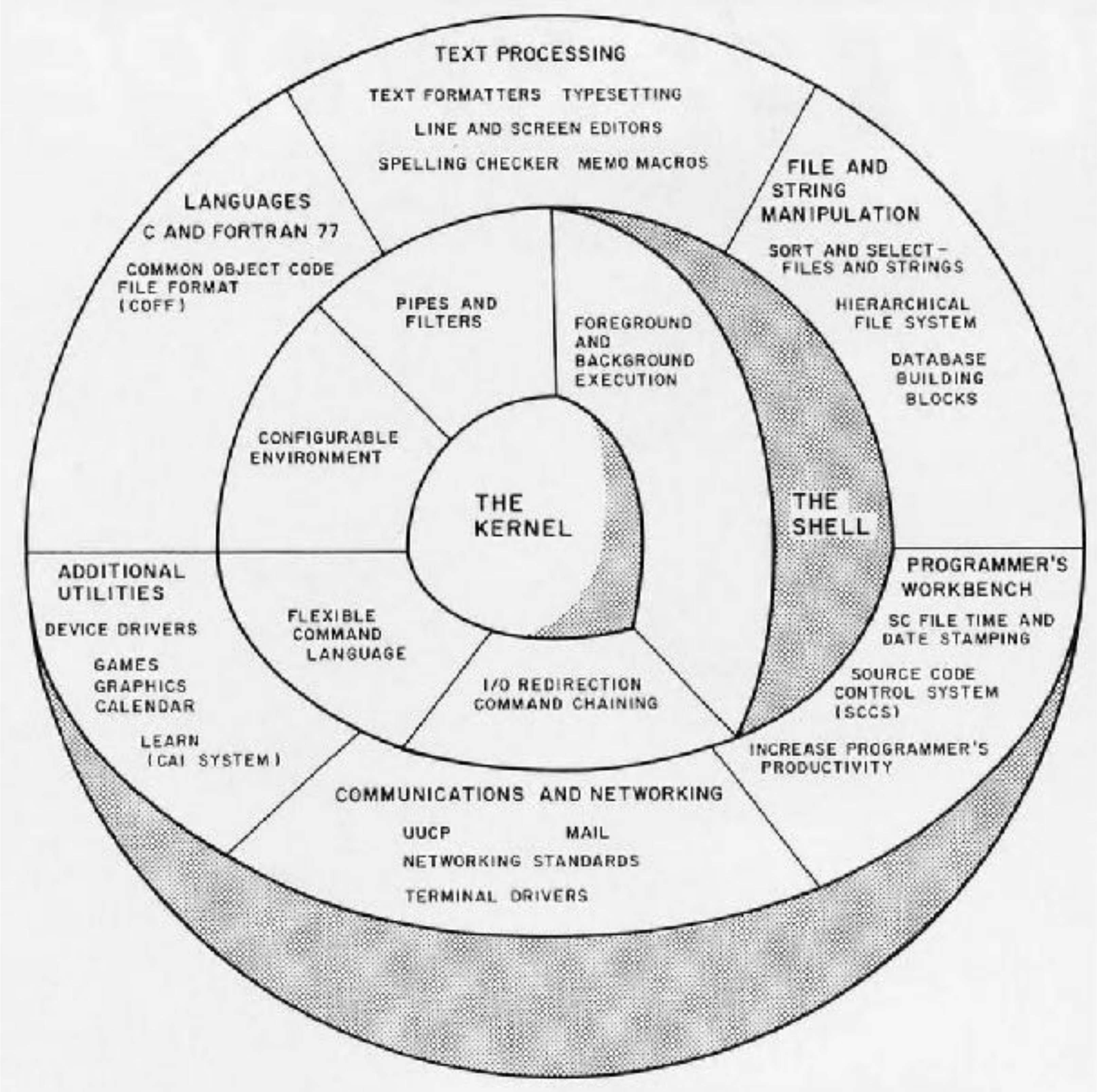
http://upload.wikimedia.org/wikipedia/commons/3/36/Ken_n_dennis.jpg

He found a little used PDP-7.



http://en.wikipedia.org/wiki/PDP-7#/media/File:Pdp7-oslo-2005.jpeg

Ended up writing a nearly complete operating system from scratch.


TEXT PROCESSING
TEXT FORMATTERS TYPESETTING
LINE AND SCREEN EDITORS
SPELLING CHECKER MEMO MACROS
LANGUAGES
C AND FORTRAN 77
COMMON OBJECT CODE FILE FORMAT (COFF)
FILE AND STRING MANIPULATION
SORT AND SELECT - FILES AND STRINGS
HIERARCHICAL FILE SYSTEM
DATABASE BUILDING BLOCKS
PIPES AND FILTERS
FOREGROUND AND BACKGROUND EXECUTION
CONFIGURABLE ENVIRONMENT
THE KERNEL
THE SHELL
ADDITIONAL UTILITIES
DEVICE DRIVERS
GAMES
GRAPHICS
CALENDAR
LEARN (CAI SYSTEM)
FLEXIBLE COMMAND LANGUAGE
I/O REDIRECTION COMMAND CHAINING
PROGRAMMER'S WORKBENCH
SC FILE TIME AND DATE STAMPING
SOURCE CODE CONTROL SYSTEM (SCCS)
INCREASE PROGRAMMER'S PRODUCTIVITY
COMMUNICATIONS AND NETWORKING
UUCP
MAIL
NETWORKING STANDARDS
TERMINAL DRIVERS

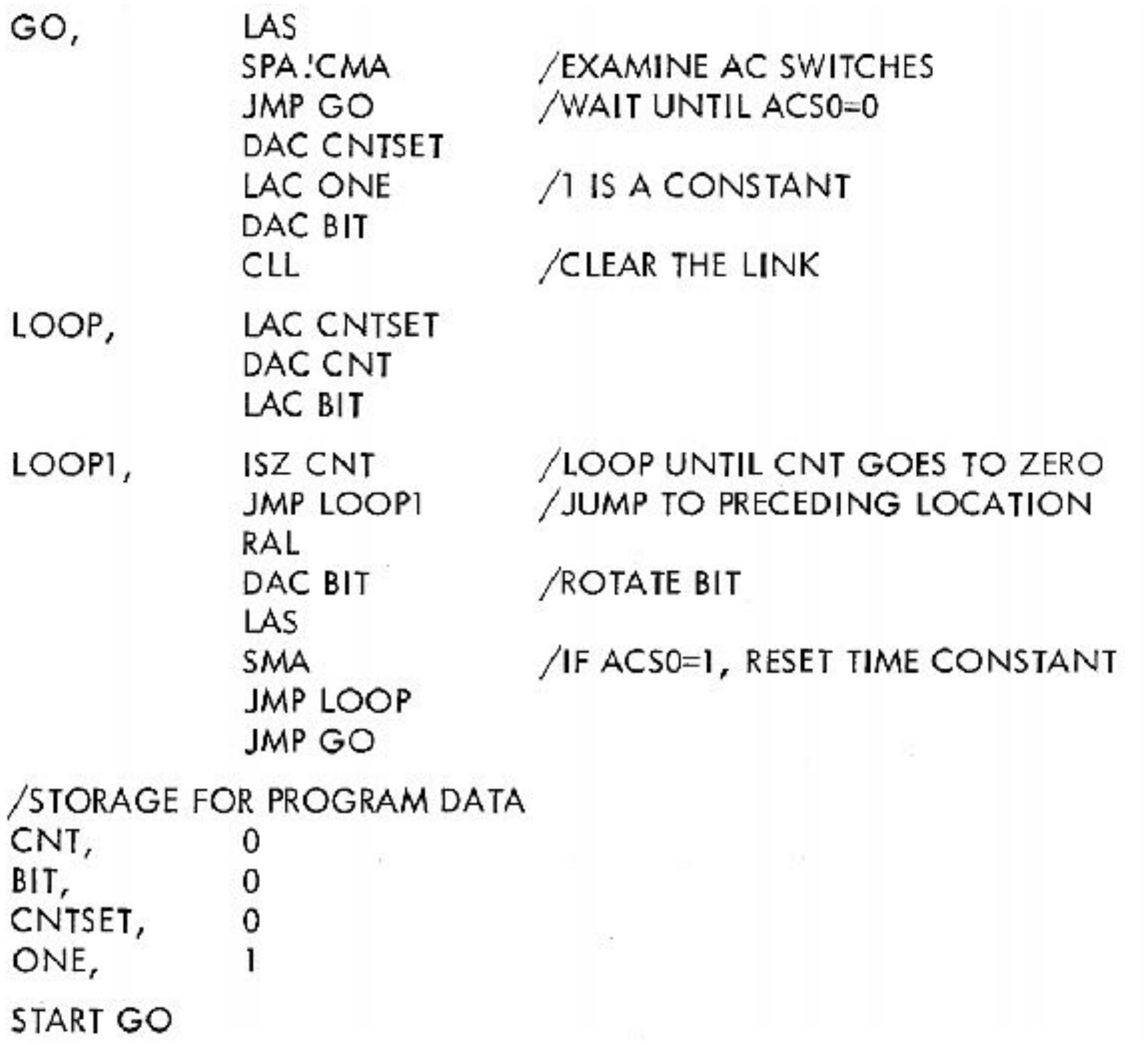
https://archive.org/stream/byte-magazine-1983-08/1983_08_BYTE_08-08_The_C_Language#page/n190/mode/1up

In about 4 weeks.

“Essentially one person for a month, it was just my self.”

(Ken Thompson, 1989 Interview)

In pure assembler of course.



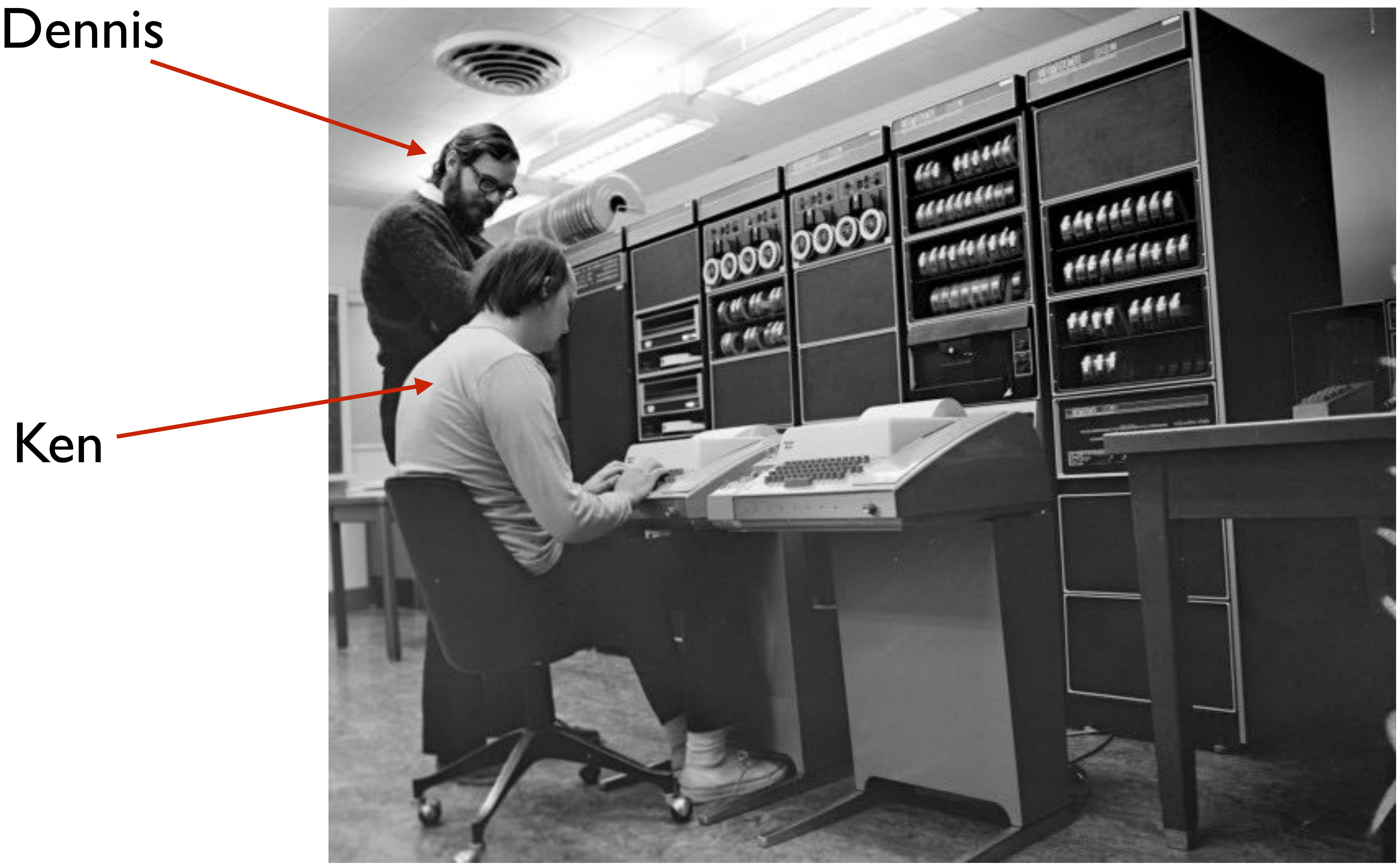
```
GO,         LAS
            SPA!CMA         /EXAMINE AC SWITCHES
            JMP GO          /WAIT UNTIL ACS0=0
            DAC CNTSET
            LAC ONE         /1 IS A CONSTANT
            DAC BIT
            CLL             /CLEAR THE LINK

LOOP,       LAC CNTSET
            DAC CNT
            LAC BIT

LOOP1,      ISZ CNT         /LOOP UNTIL CNT GOES TO ZERO
            JMP LOOP1       /JUMP TO PRECEDING LOCATION
            RAL
            DAC BIT         /ROTATE BIT
            LAS
            SMA             /IF ACS0=1, RESET TIME CONSTANT
            JMP LOOP
            JMP GO

/STORAGE FOR PROGRAM DATA
CNT,        0
BIT,        0
CNTSET,     0
ONE,        1

START GO
```

http://bitsavers.trailing-edge.com/pdf/dec/pdp7/PDP-7_AsmMan.pdf

Dennis Ritchie soon joined the effort.

http://upload.wikimedia.org/wikipedia/commons/3/36/Ken_n_dennis.jpg

While porting Unix to a PDP-11

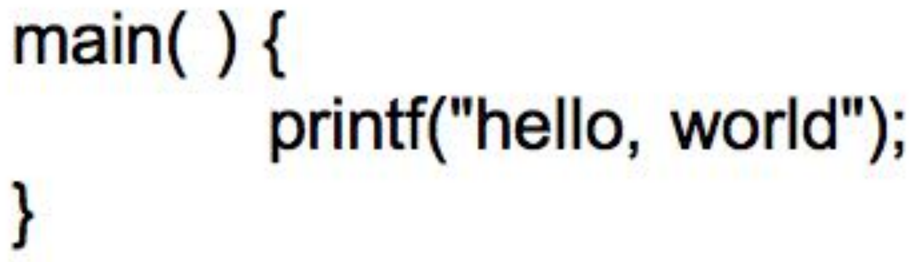


Dennis

Ken

http://cm.bell-labs.com/who/dmr/picture.html

they invented C,



```
main( ) {
        printf("hello, world");
}
```

http://cm.bell-labs.com/cm/cs/who/dmr/ctut.pdf

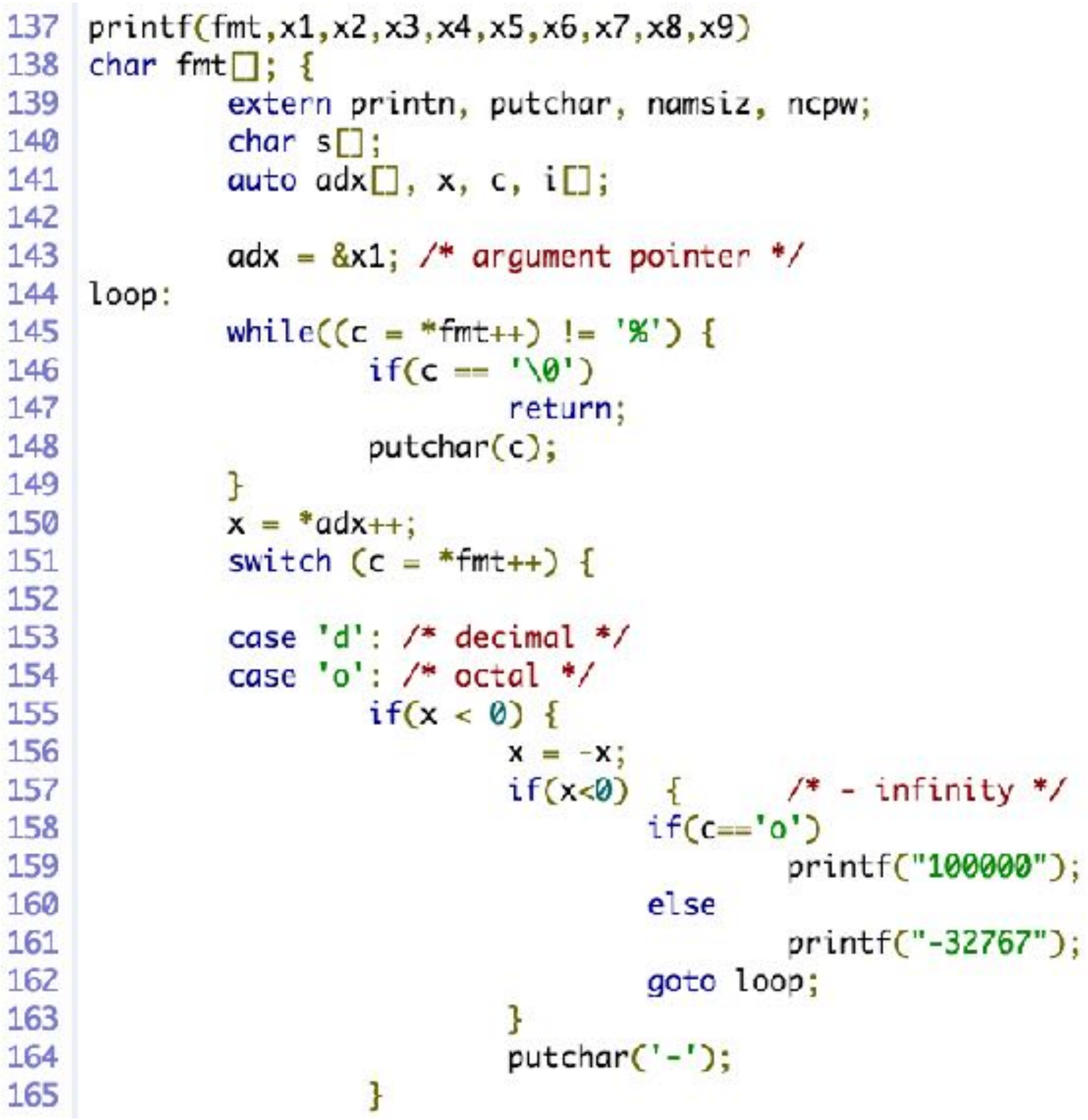
heavily inspired by Martin Richards' portable systems programming language BCPL.

Martin Richards, Dec 2014

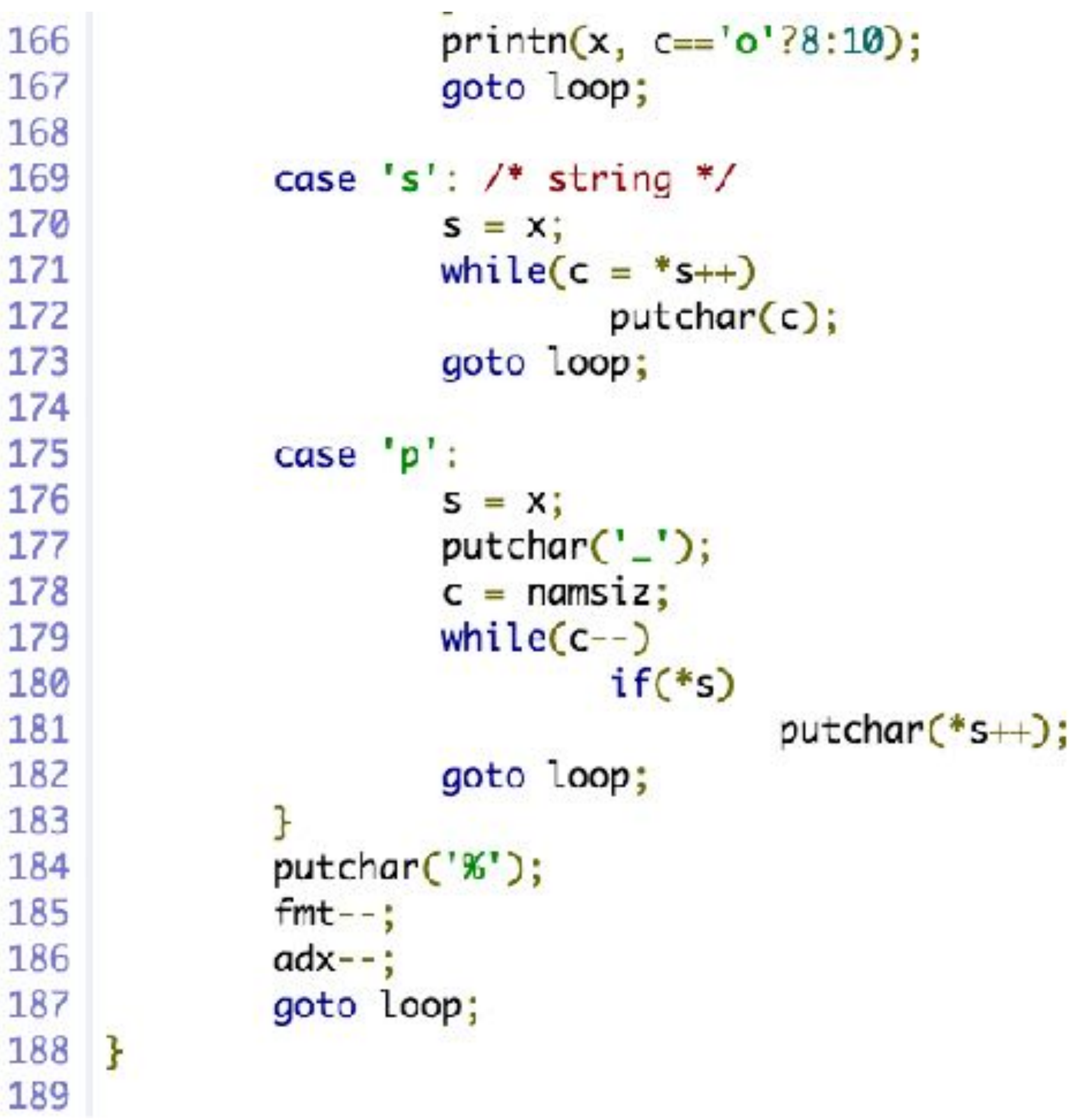
```
GET “LIBHDR”
LET START() BE WRITES(“Hello, World”)
```

http://cm.bell-labs.com/cm/cs/who/dmr/ctut.pdf

In 1972 Unix was rewritten in C.



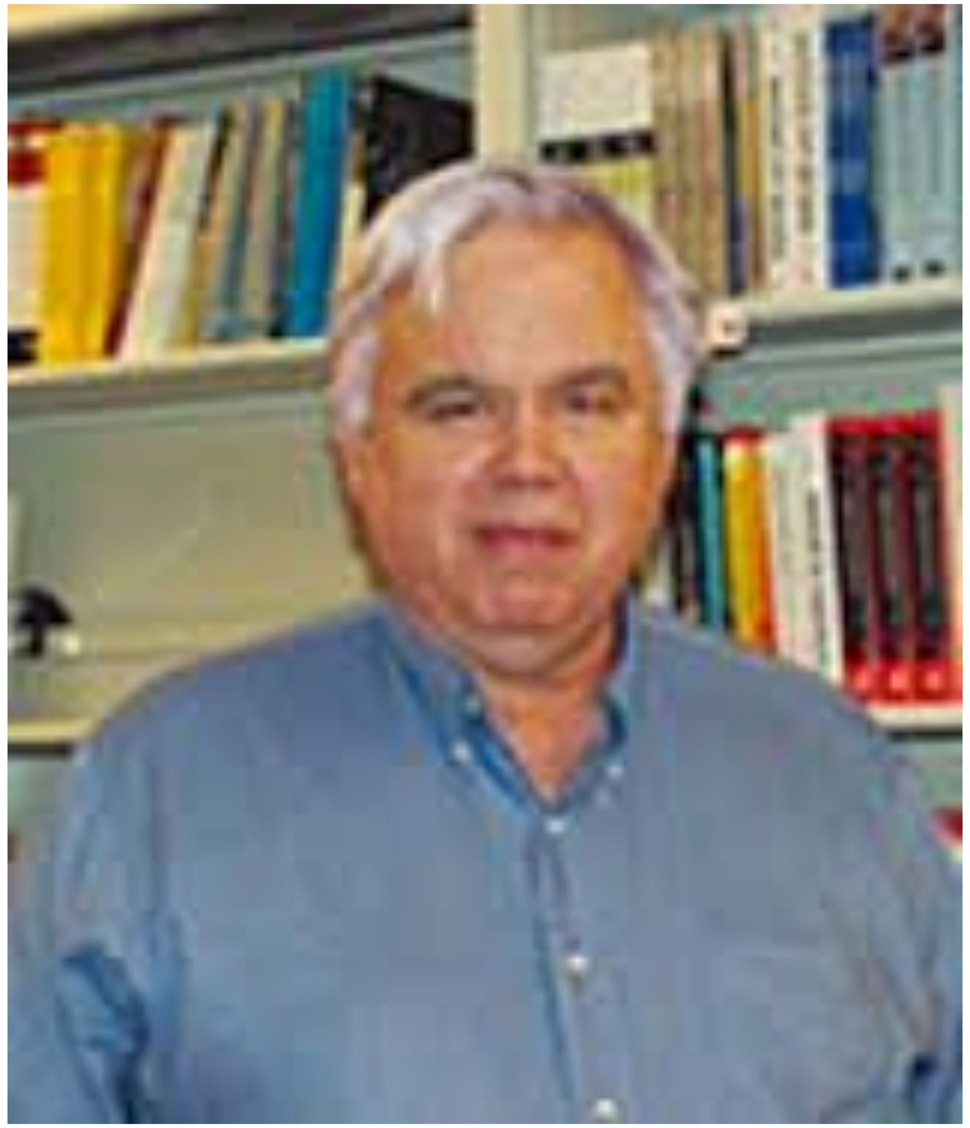
```
printf(fmt,x1,x2,x3,x4,x5,x6,x7,x8,x9)
char fmt[]; {
        extern printn, putchar, namsiz, ncpw;
        char s[];
        auto adx[], x, c, i[];

        adx = &x1; /* argument pointer */
loop:
        while((c = *fmt++) != '%') {
                if(c == '\0')
                        return;
                putchar(c);
        }
        x = *adx++;
        switch (c = *fmt++) {

        case 'd': /* decimal */
        case 'o': /* octal */
                if(x < 0) {
                        x = -x;
                        if(x<0)  {      /* - infinity */
                                if(c=='o')
                                        printf("100000");
                                else
                                        printf("-32767");
                                goto loop;
                        }
                        putchar('-');
                }
```



```
                printn(x, c=='o'?8:10);
                goto loop;

        case 's': /* string */
                s = x;
                while(c = *s++)
                        putchar(c);
                goto loop;

        case 'p':
                s = x;
                putchar('_');
                c = namsiz;
                while(c--)
                        if(*s)
                                putchar(*s++);
                goto loop;
        }
        putchar('%');
        fmt--;
        adx--;
        goto loop;
}
```

https://code.google.com/p/unix-jun72/source/browse/trunk/src/c/c03.c

Due to Steve Johnsons Portable C Compiler,



Fact: from “The Development of the C Language” by Dennis Ritchie

Unix and C could be ported to all kinds of computer architectures.

http://www.computerhistory.org/collections/catalog/102691249

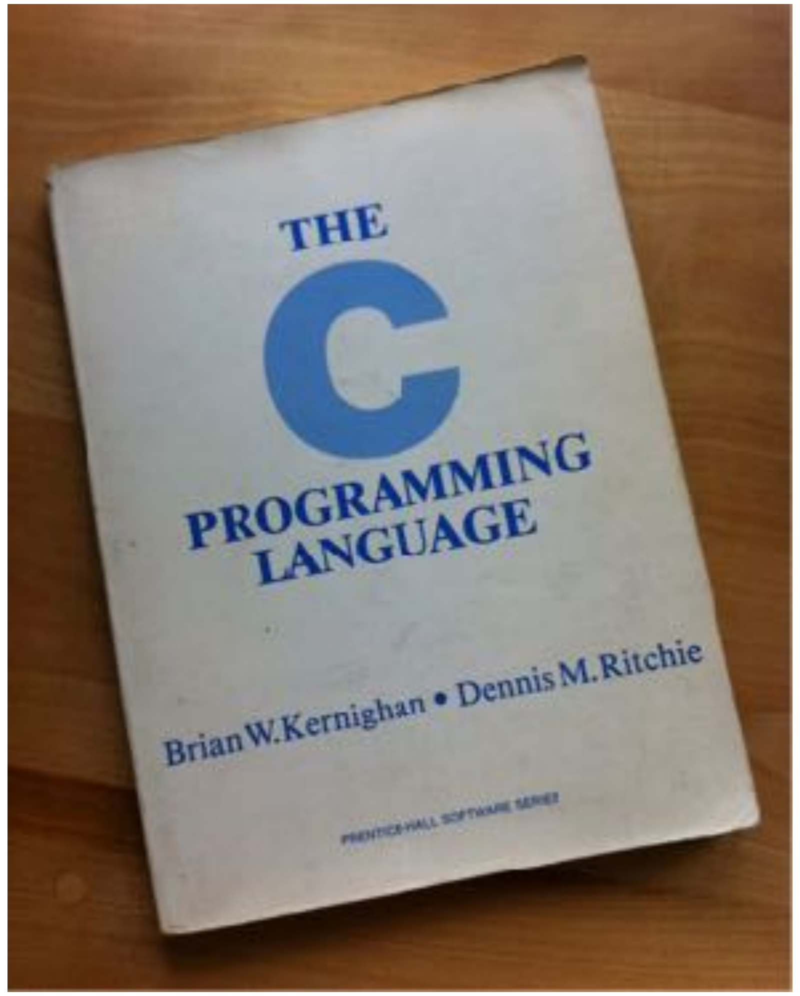
http://www.technikum29.de/en/computer/early-computers

http://en.wikipedia.org/wiki/IBM_System/370

http://alegion63.tripod.com/bob/id6.html

C became the most successful programming language ever.

Initially K&R and PCC was the only reference for C.

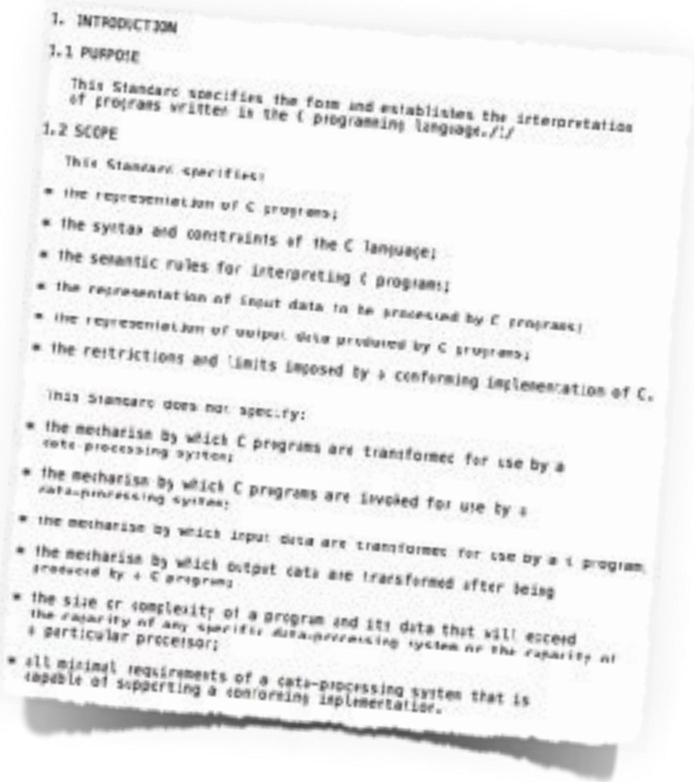

THE
C
PROGRAMMING
LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
PRENTICE-HALL SOFTWARE SERIES

K&R
(1978)

http://blogs.ipmsonline.com/wp-content/uploads/2014/11/

With significant contributions from C++ (Bjarne Stroustrup), the C language got standardized

in 1989/1990, and thereafter updated in 1999 and 2011.



1. INTRODUCTION

1.1 PURPOSE

This Standard specifies the form and establishes the interpretation of programs written in the C programming language./1/

1.2 SCOPE

This Standard specifies:

- the representation of C programs;
- the syntax and constraints of the C language;
- the semantic rules for interpreting C programs;
- the representation of input data to be processed by C programs;
- the representation of output data produced by C programs;
- the restrictions and limits imposed by a conforming implementation of C.

This Standard does not specify:

- the mechanism by which C programs are transformed for use by a data-processing system;
- the mechanism by which C programs are invoked for use by a data-processing system;
- the mechanism by which input data are transformed for use by a C program;
- the mechanism by which output data are transformed after being produced by a C program;
- the size or complexity of a program and its data that will exceed the capacity of any specific data-processing system or the capacity of a particular processor;
- all minimal requirements of a data-processing system that is capable of supporting a conforming implementation.

ANSI/ISO C
(C89/C90)

INTERNATIONAL STANDARD

ISO/IEC 9899

Programming languages — C

ISO IEC

C99

American National Standard

Information technology — Programming languages — C

Developed by

incits

Where IT all begins

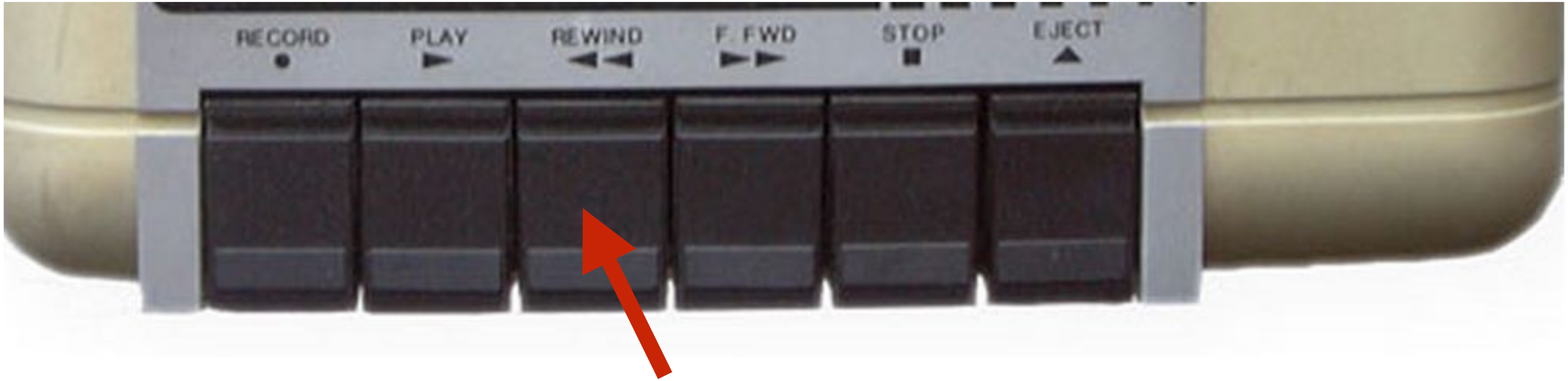
ANSI

C11

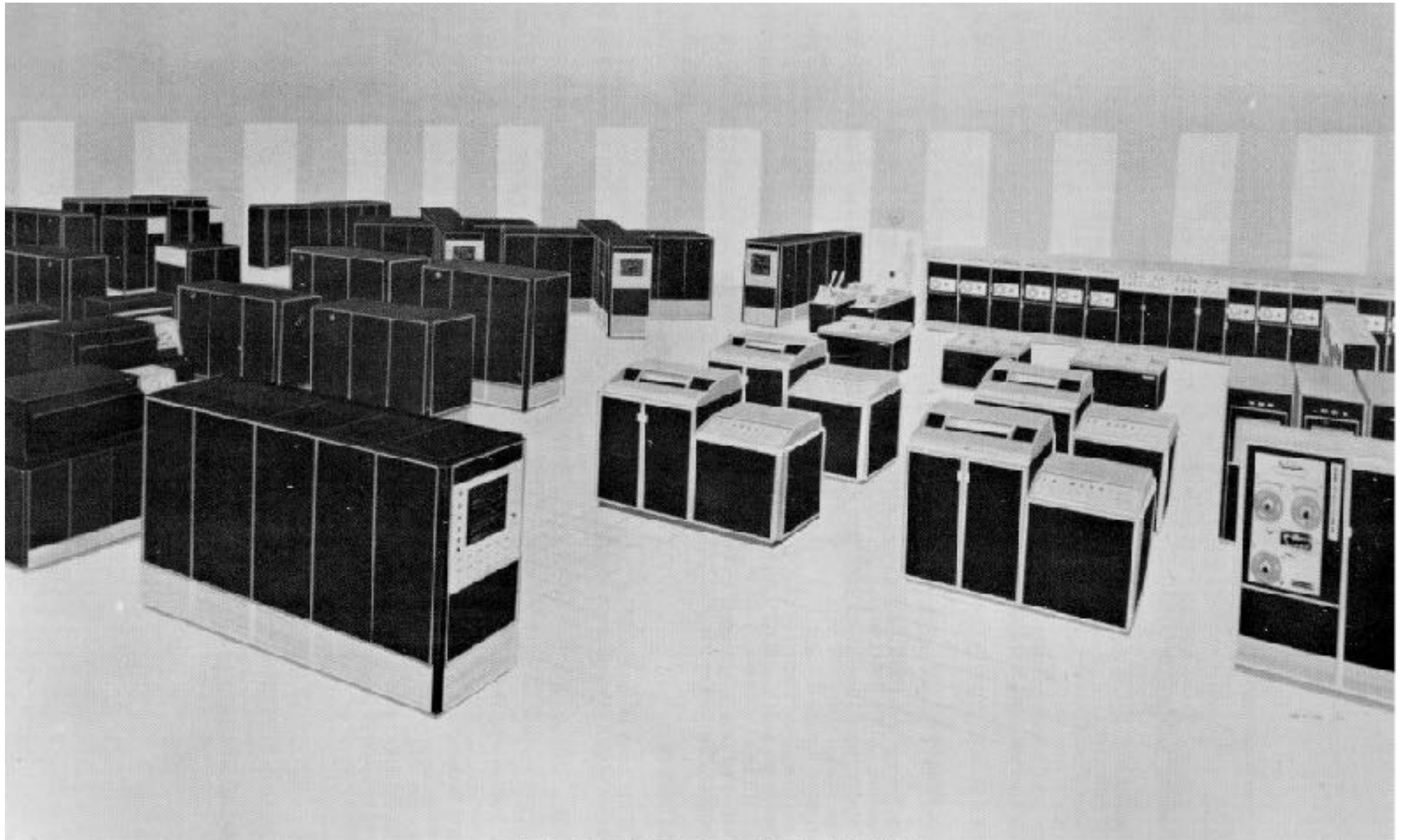
http://blogs.ipmsonline.com/wp-content/uploads/2014/11/

http://powerlisting.wikia.com/wiki/File:The-end.jpg

At Bell Labs. Back In 1969. Ken Thompson wanted to play. He found a little used PDP-7. Ended up writing a nearly complete operating system from scratch. In about 4 weeks. In pure assembler of course. Dennis Ritchie soon joined the effort. While porting Unix to a PDP-11 they invented C, heavily inspired by Martin Richards' portable systems programming language BCPL. In 1972 Unix was rewritten in C. Due to Steve Johnsons Portable C Compiler (PCC), Unix and C could be ported to all kinds of computer architectures. C became the most successful programming language ever. Initially the K&R and PCC was only reference for C. With significant contributions from C++ (Bjarne Stroustrup), the C language got standardized in 1989/1990, and thereafter updated in 1999 and 2011.


RECORD
PLAY
REWIND
F. FWD
STOP
EJECT

Ken Thompson, Dennis Ritchie and 20+ more technical staff from Bell Labs had been working on the very innovative Multics project for several years.



GE-645 SYSTEM

http://web.mit.edu/saltzer/www/multics.html

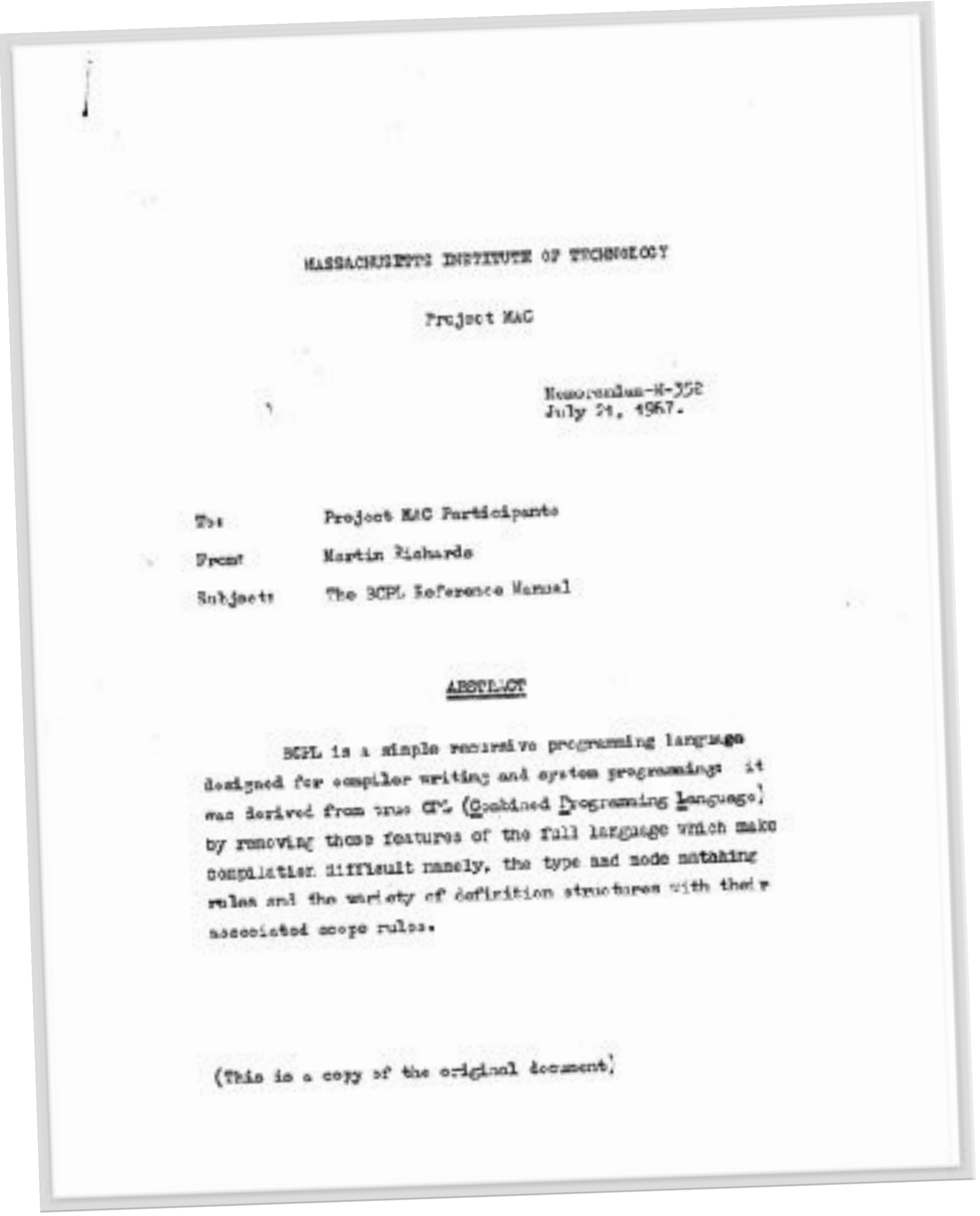
While working on the Multics projects, Dennis and Ken had also been exposed to the very portable and efficient systems programming language BCPL.

```
GET “LIBHDR”
LET START() BE WRITES(“Hello, World”)
```

"*Both of us were really taken by the language and did a lot of work with it.*" (Ken Thompson, 1989 interview)

http://www.princeton.edu/~hos/mike/transcripts/thompson.htm

BCPL (1967) was the brainchild of Martin Richards from the University of Cambridge



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Memorandum-M-352
July 21, 1967.

To: Project MAC Participants

From: Martin Richards

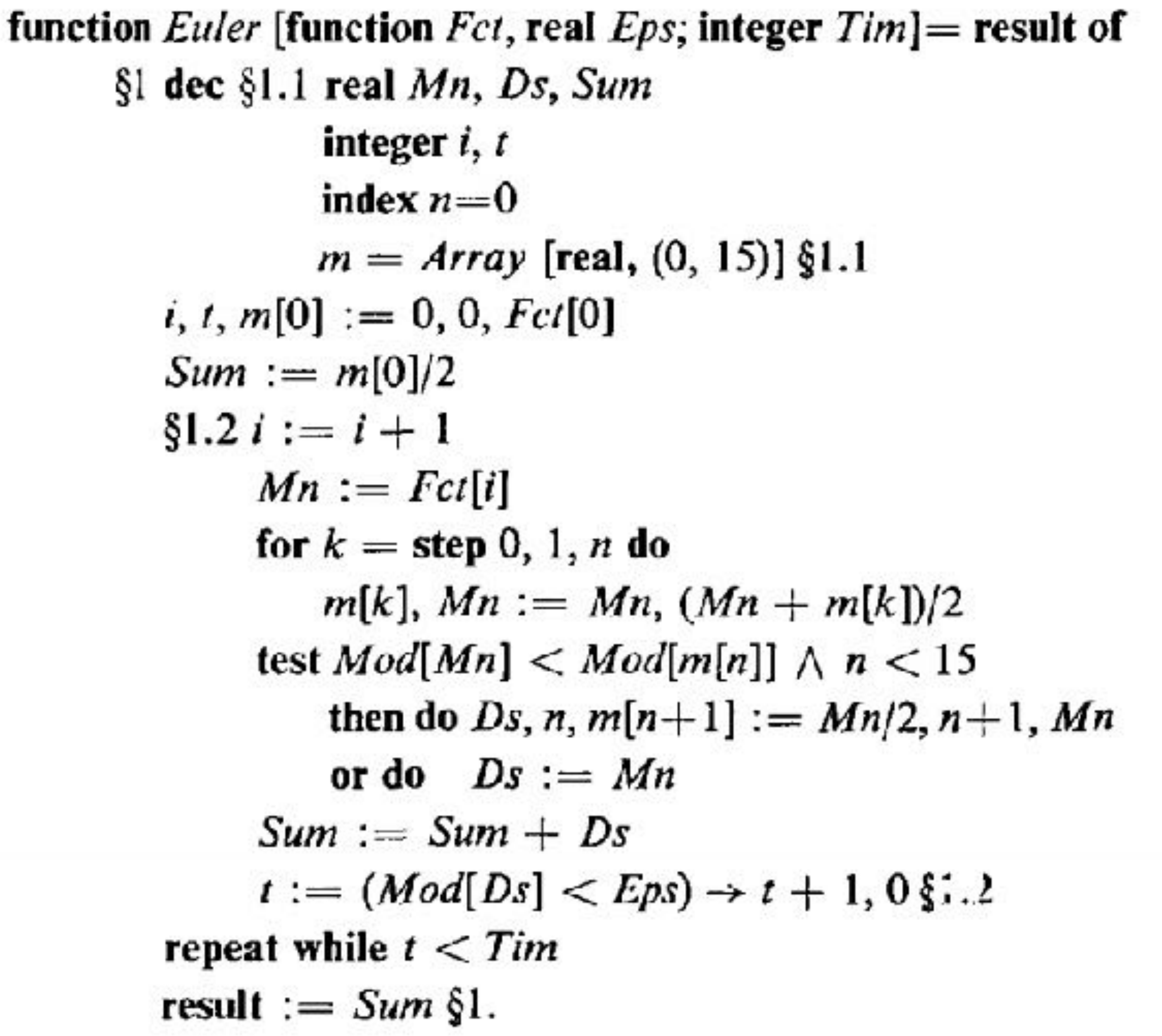
Subject: The BCPL Reference Manual

ABSTRACT

BCPL is a simple recursive programming language designed for compiler writing and system programming: it was derived from true CPL (Combined Programming Language) by removing those features of the full language which make compilation difficult namely, the type and mode matching rules and the variety of definition structures with their associated scope rules.

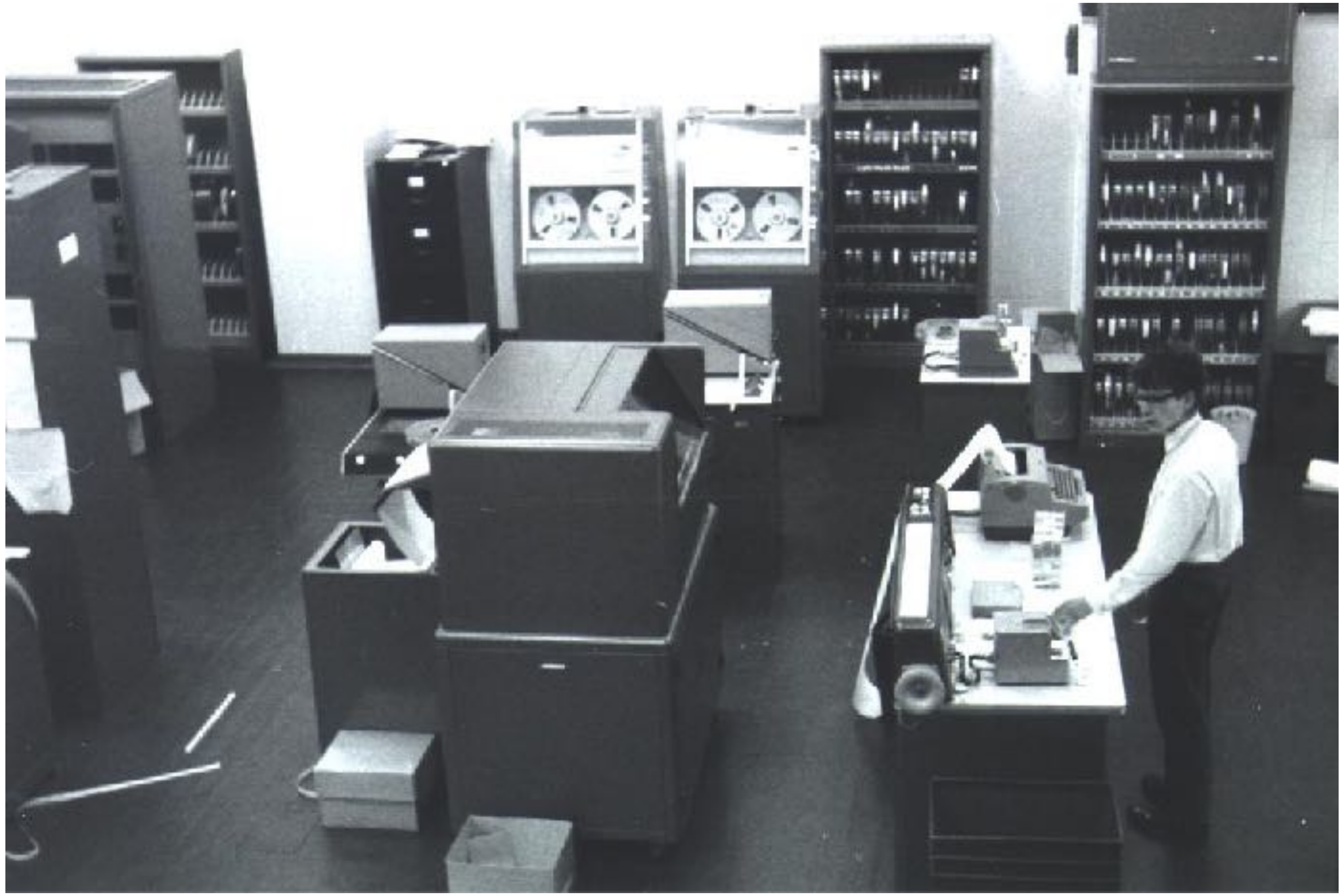
(This is a copy of the original document)

BCPL was a very much simplified version of CPL (1963).



```
function Euler [function Fct, real Eps; integer Tim]= result of
      §1 dec §1.1 real Mn, Ds, Sum
                  integer i, t
                  index n=0
                  m = Array [real, (0, 15)] §1.1
         i, t, m[0] := 0, 0, Fct[0]
         Sum := m[0]/2
         §1.2 i := i + 1
               Mn := Fct[i]
               for k = step 0, 1, n do
                   m[k], Mn := Mn, (Mn + m[k])/2
               test Mod[Mn] < Mod[m[n]] ∧ n < 15
                   then do Ds, n, m[n+1] := Mn/2, n+1, Mn
                   or do   Ds := Mn
               Sum := Sum + Ds
               t := (Mod[Ds] < Eps) → t + 1, 0 §1.2
         repeat while t < Tim
         result := Sum §1.
```

CPL was the language initially designed for the Atlas computer to be installed in Cambridge (ordered in 1961, operational in 1964).



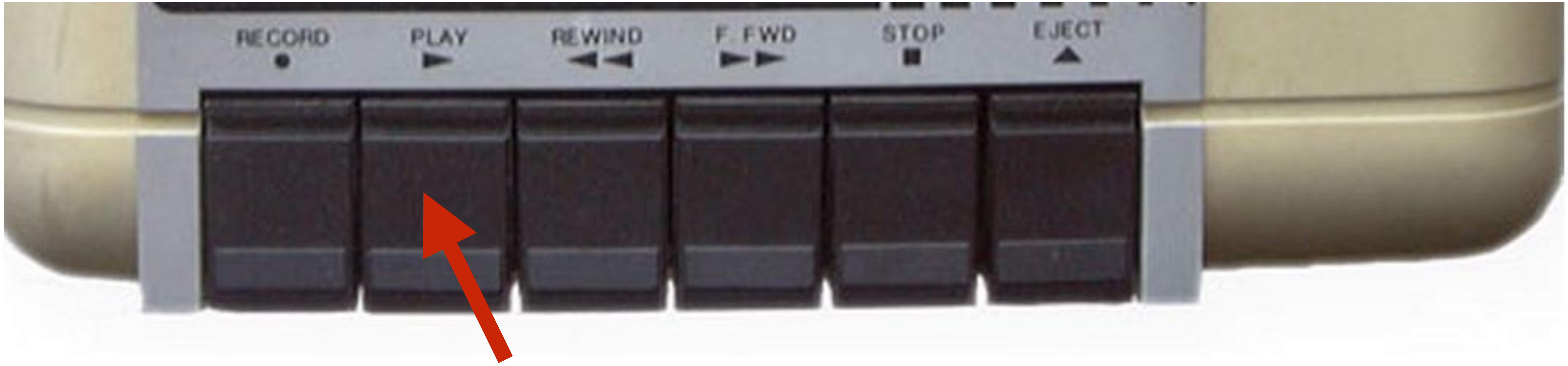
A replacement for EDSAC 2,

EDSAC 2 users in 1960

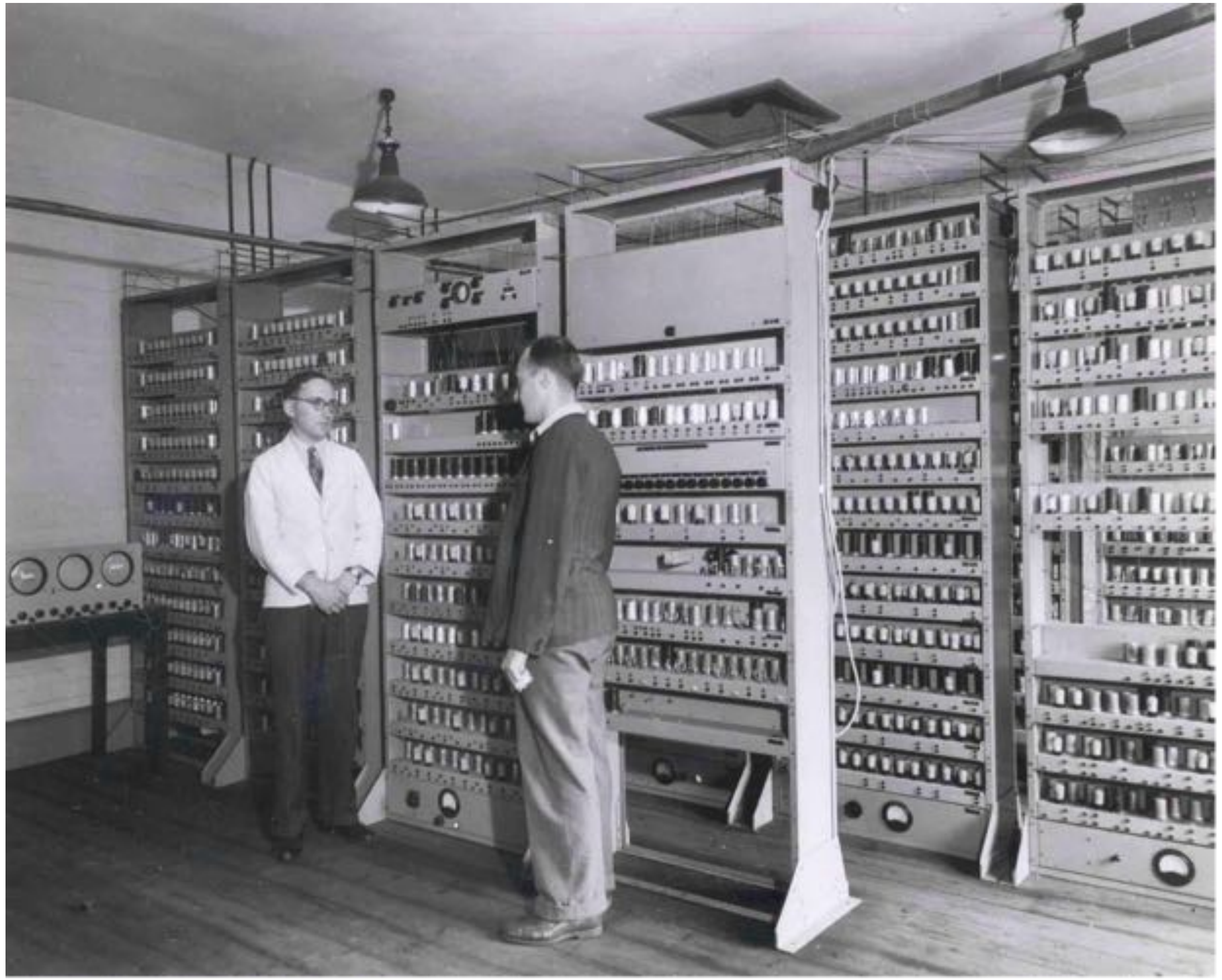
http://en.wikipedia.org/wiki/EDSAC_2

which was an upgrade of the original EDSAC computer (1949)

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


RECORD
●
PLAY
►
REWIND
◄◄
F. FWD
►►
STOP
■
EJECT
▲

EDSAC was arguably, the first electronic digital stored-program computer. It ran its first program May 6, 1949



Maurice Wilkes and Bill Renwick in front of the complete EDSAC

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

Maurice Wilkes' himself commenting on the 1951 film about how EDSAC was used in practice:

https://youtu.be/x-vS0WcJyNM

The EDSAC 1951 film
abridged version

Commentary by
M. V. Wilkes

EDSAC 2 users in 1960

http://en.wikipedia.org/wiki/EDSAC_2

A scaled down version of Atlas (called Titan / Atlas2) was ordered in 1961, delivered to Cambridge in 1963, but not usable until early 1964

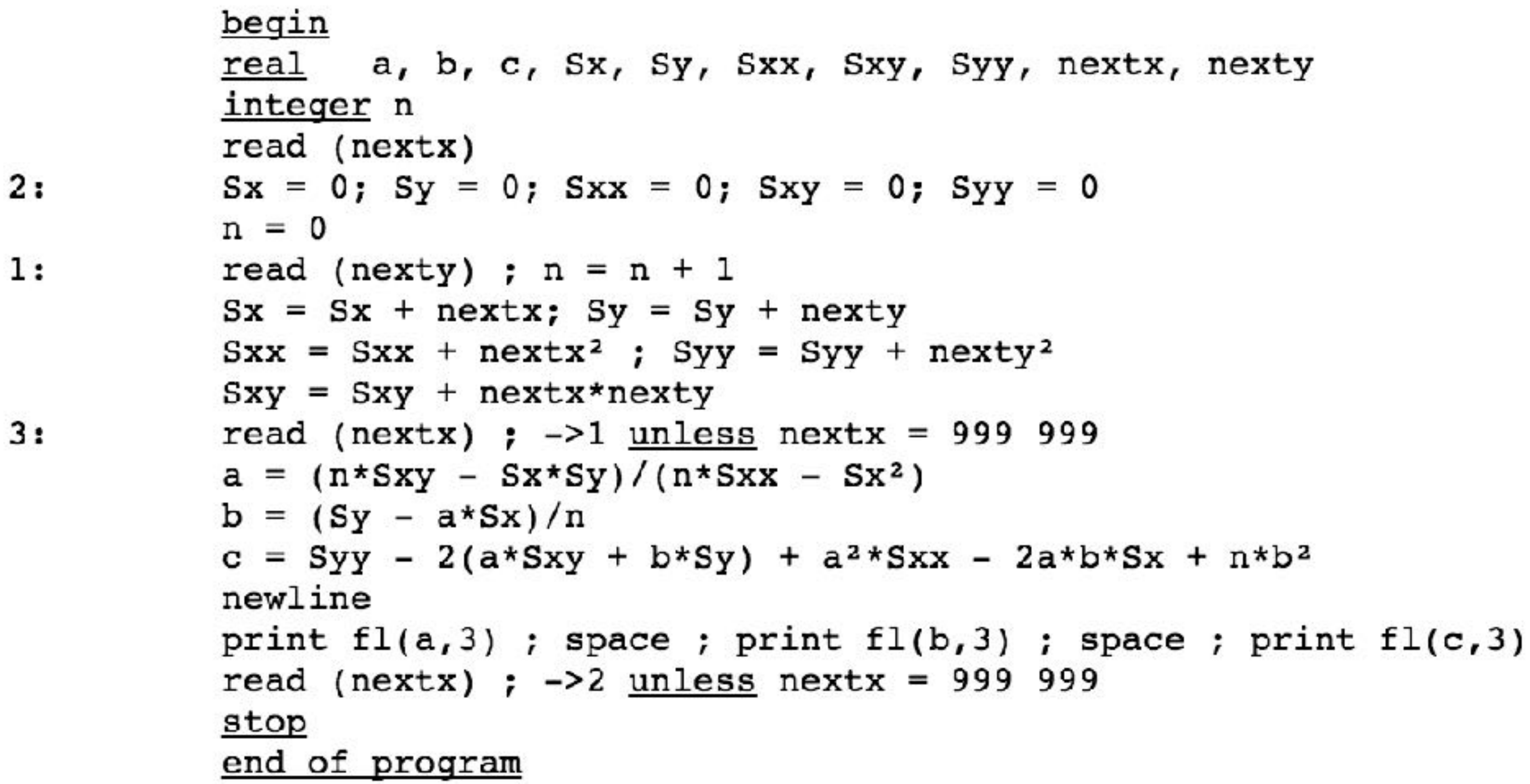
"How BCPL evolved from CPL", Martin Richards

http://en.wikipedia.org/wiki/Titan_(computer) a programming language was needed!

Many existing programming languages was concidered….

Atlas Autocode

(designed by Tony Brooker and Derrick Morris)

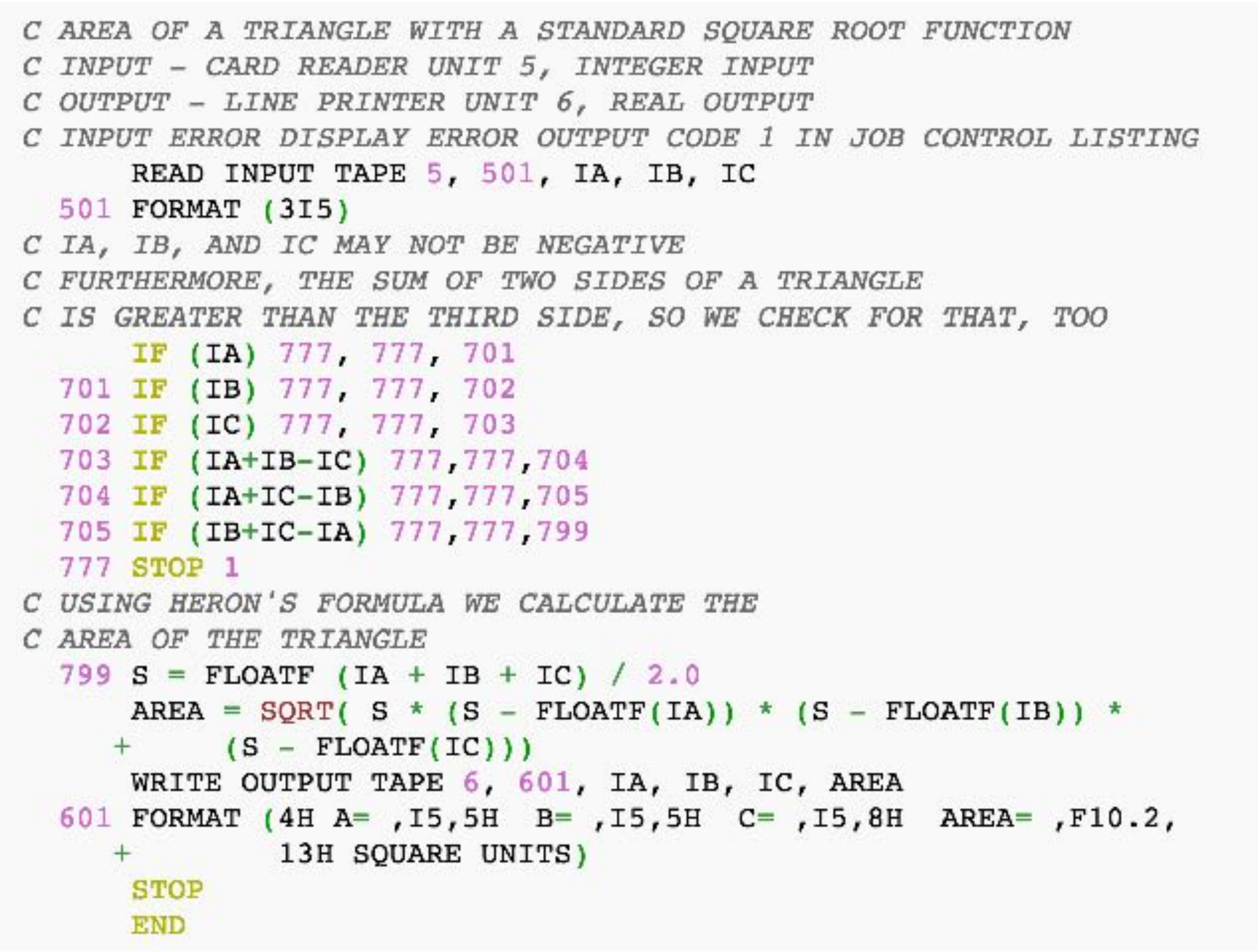


```
           begin
           real   a, b, c, Sx, Sy, Sxx, Sxy, Syy, nextx, nexty
           integer n
           read (nextx)
2:         Sx = 0; Sy = 0; Sxx = 0; Sxy = 0; Syy = 0
           n = 0
1:         read (nexty) ; n = n + 1
           Sx = Sx + nextx; Sy = Sy + nexty
           Sxx = Sxx + nextx² ; Syy = Syy + nexty²
           Sxy = Sxy + nextx*nexty
3:         read (nextx) ; ->1 unless nextx = 999 999
           a = (n*Sxy - Sx*Sy)/(n*Sxx - Sx²)
           b = (Sy - a*Sx)/n
           c = Syy - 2(a*Sxy + b*Sy) + a²*Sxx - 2a*b*Sx + n*b²
           newline
           print fl(a,3) ; space ; print fl(b,3) ; space ; print fl(c,3)
           read (nextx) ; ->2 unless nextx = 999 999
           stop
           end of program
```

From David Hartley's article "CPL: Failed Venture or Noble Ancestor?" (2013)
http://history.dcs.ed.ac.uk/archive/docs/atlasautocode.html

Fortran

(appeared 1957, designed by John Backus)



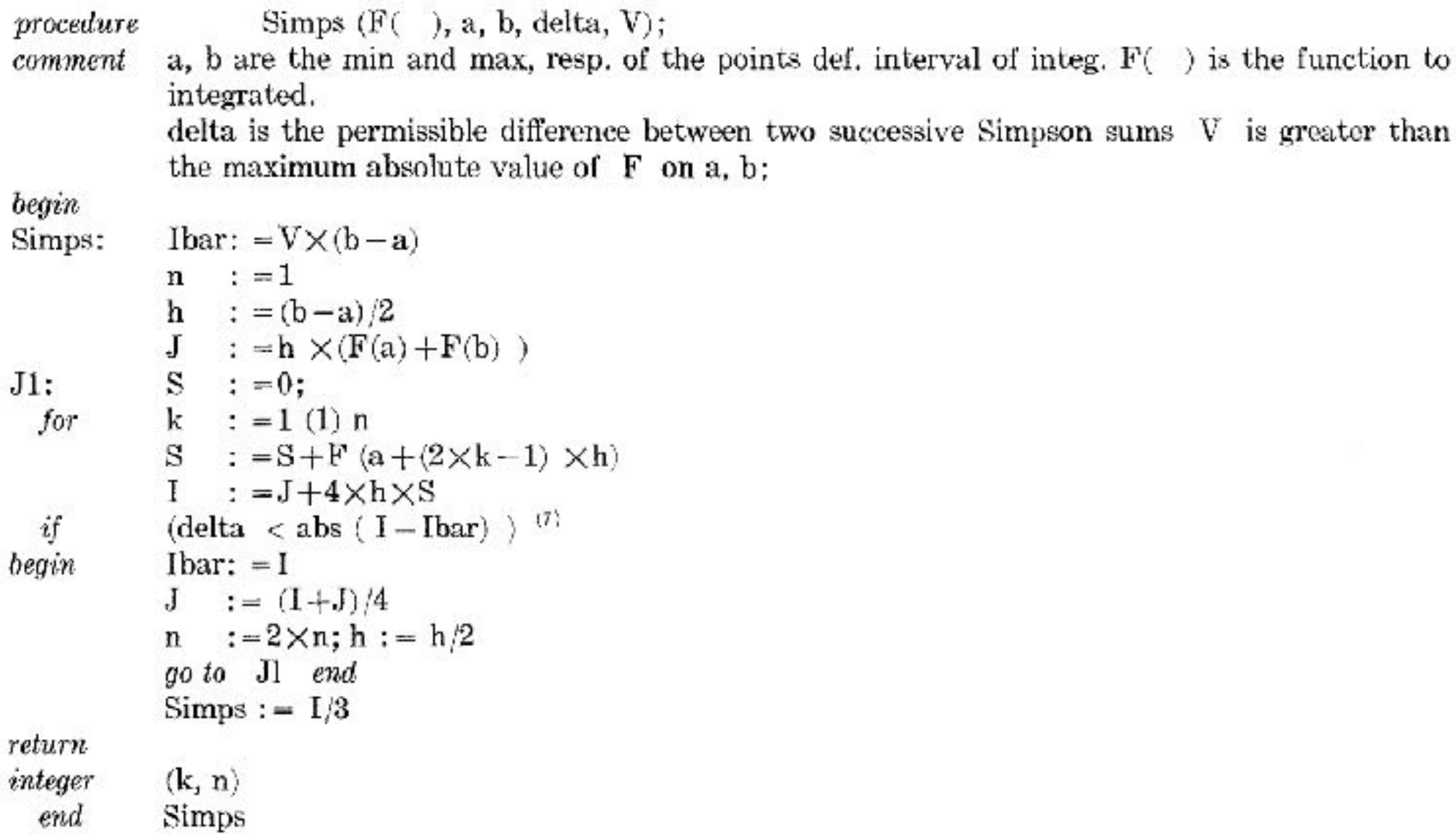
```
C AREA OF A TRIANGLE WITH A STANDARD SQUARE ROOT FUNCTION
C INPUT - CARD READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPLAY ERROR OUTPUT CODE 1 IN JOB CONTROL LISTING
      READ INPUT TAPE 5, 501, IA, IB, IC
  501 FORMAT (3I5)
C IA, IB, AND IC MAY NOT BE NEGATIVE
C FURTHERMORE, THE SUM OF TWO SIDES OF A TRIANGLE
C IS GREATER THAN THE THIRD SIDE, SO WE CHECK FOR THAT, TOO
      IF (IA) 777, 777, 701
  701 IF (IB) 777, 777, 702
  702 IF (IC) 777, 777, 703
  703 IF (IA+IB-IC) 777,777,704
  704 IF (IA+IC-IB) 777,777,705
  705 IF (IB+IC-IA) 777,777,799
  777 STOP 1
C USING HERON'S FORMULA WE CALCULATE THE
C AREA OF THE TRIANGLE
  799 S = FLOATF (IA + IB + IC) / 2.0
      AREA = SQRT( S * (S - FLOATF(IA)) * (S - FLOATF(IB)) *
     +     (S - FLOATF(IC)))
      WRITE OUTPUT TAPE 6, 601, IA, IB, IC, AREA
  601 FORMAT (4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,F10.2,
     +        13H SQUARE UNITS)
      STOP
      END
```

Simple FORTRAN II program

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

Algol

(aka IAL, designed by Friedrich L. Bauer, Hermann Bottenbruch, Heinz Rutishauser, Klaus Samelson, John Backus, Charles Katz, Alan Perlis, Joseph Henry Wegstein)



```
procedure        Simps (F(   ), a, b, delta, V);
comment    a, b are the min and max, resp. of the points def. interval of integ. F(   ) is the function to
           integrated.
           delta is the permissible difference between two successive Simpson sums  V  is greater than
           the maximum absolute value of  F  on a, b;
begin
Simps:     Ibar: =V×(b−a)
           n    : =1
           h    : =(b−a)/2
           J    : =h ×(F(a)+F(b) )
J1:        S    : =0;
  for      k    : =1 (1) n
           S    : =S+F (a+(2×k−1) ×h)
           I    : =J+4×h×S
  if       (delta  < abs ( I−Ibar) )  (7)
begin      Ibar: =I
           J    := (I+J)/4
           n    :=2×n; h := h/2
           go to  J1   end
           Simps := I/3
return
integer    (k, n)
  end      Simps
```

http://en.wikipedia.org/wiki/ALGOL_58

http://www.softwarepreservation.org/projects/ALGOL/report/Algol58_preliminary_report_CACM.pdf/

Autocode? Fortran? Algol? other languages?

But, hey….

In the early 1960's, it was common to think "we are building a new computer, so we need a new programming language."

(David Hartley, in 2013 article)

From David Hartley's article "CPL: Failed Venture or Noble Ancestor?" (2013)

CPL

~~Cambridge Programming Language~~

~~Cambridge Plus London~~

Combined Programming Language

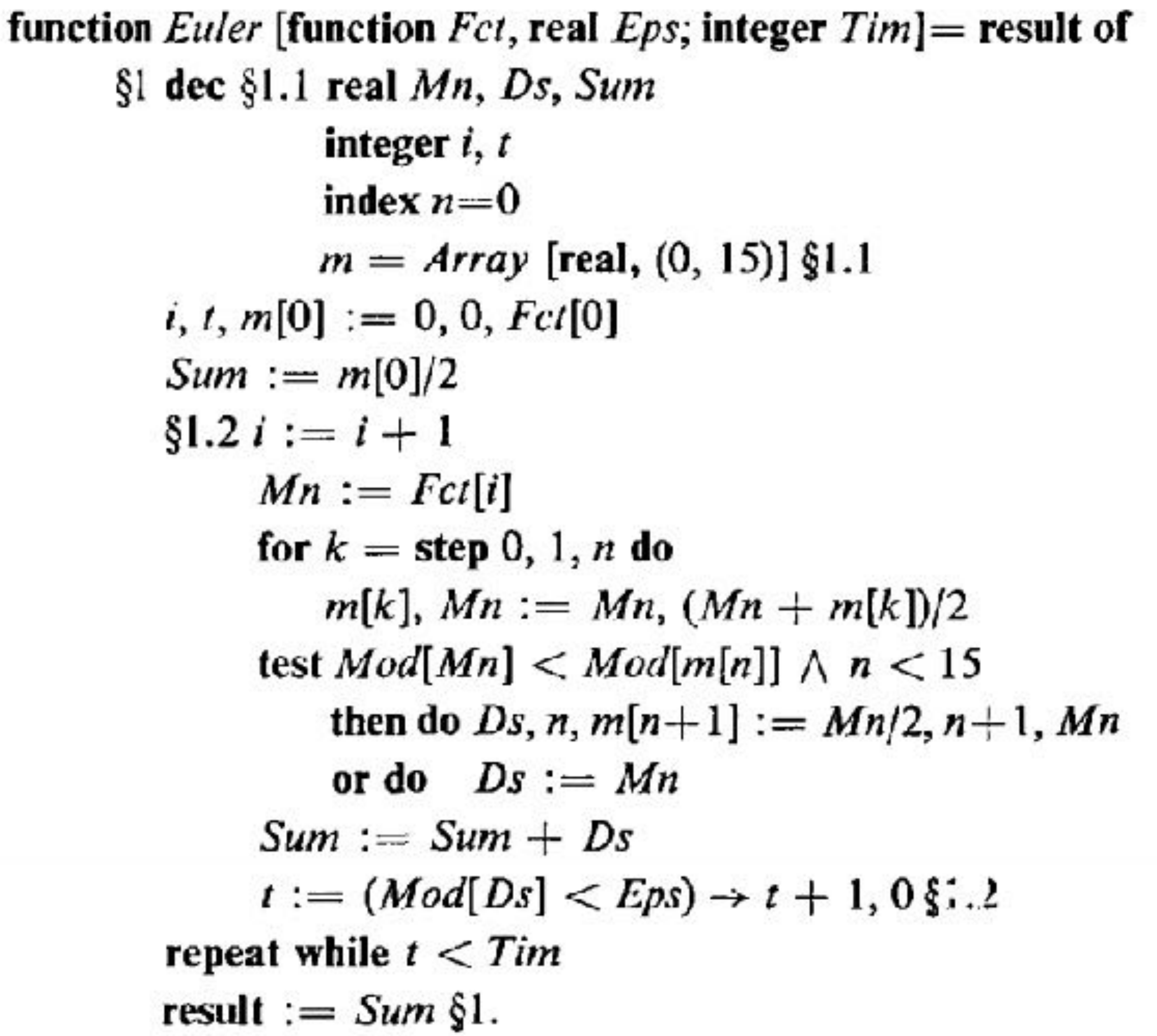
(Cristophers’ Programming Language)

"anything not explicity allowed should be forbidden ... nothing should be left undefined"

"It was envisagd that [the language] would be sufficiently general and versatile to dispense with machine-code programming as far as possible"

From David Hartley's article "CPL: Failed Venture or Noble Ancestor?" (2013)

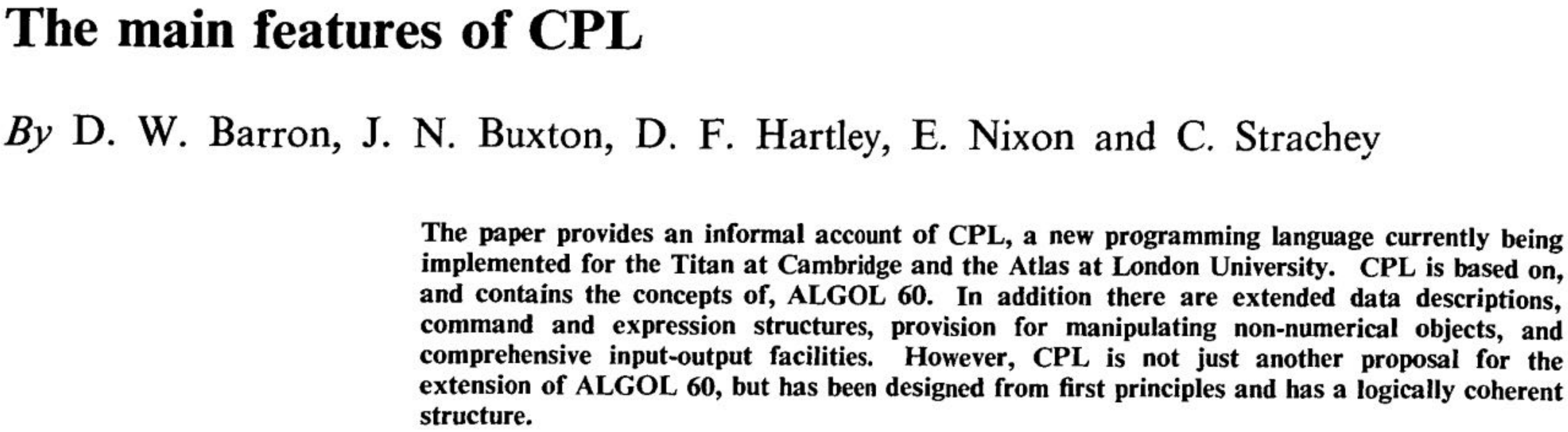
Example of CPL



```
function Euler [function Fct, real Eps; integer Tim]= result of
      §1 dec §1.1 real Mn, Ds, Sum
                  integer i, t
                  index n=0
                  m = Array [real, (0, 15)] §1.1
         i, t, m[0] := 0, 0, Fct[0]
         Sum := m[0]/2
         §1.2 i := i + 1
               Mn := Fct[i]
               for k = step 0, 1, n do
                   m[k], Mn := Mn, (Mn + m[k])/2
               test Mod[Mn] < Mod[m[n]] ∧ n < 15
                   then do Ds, n, m[n+1] := Mn/2, n+1, Mn
                   or do   Ds := Mn
               Sum := Sum + Ds
               t := (Mod[Ds] < Eps) → t + 1, 0 §1.2
         repeat while t < Tim
         result := Sum §1.
```

http://www.math.bas.bg/~bantchev/place/cpl/features.pdf

CPL as described in 1963



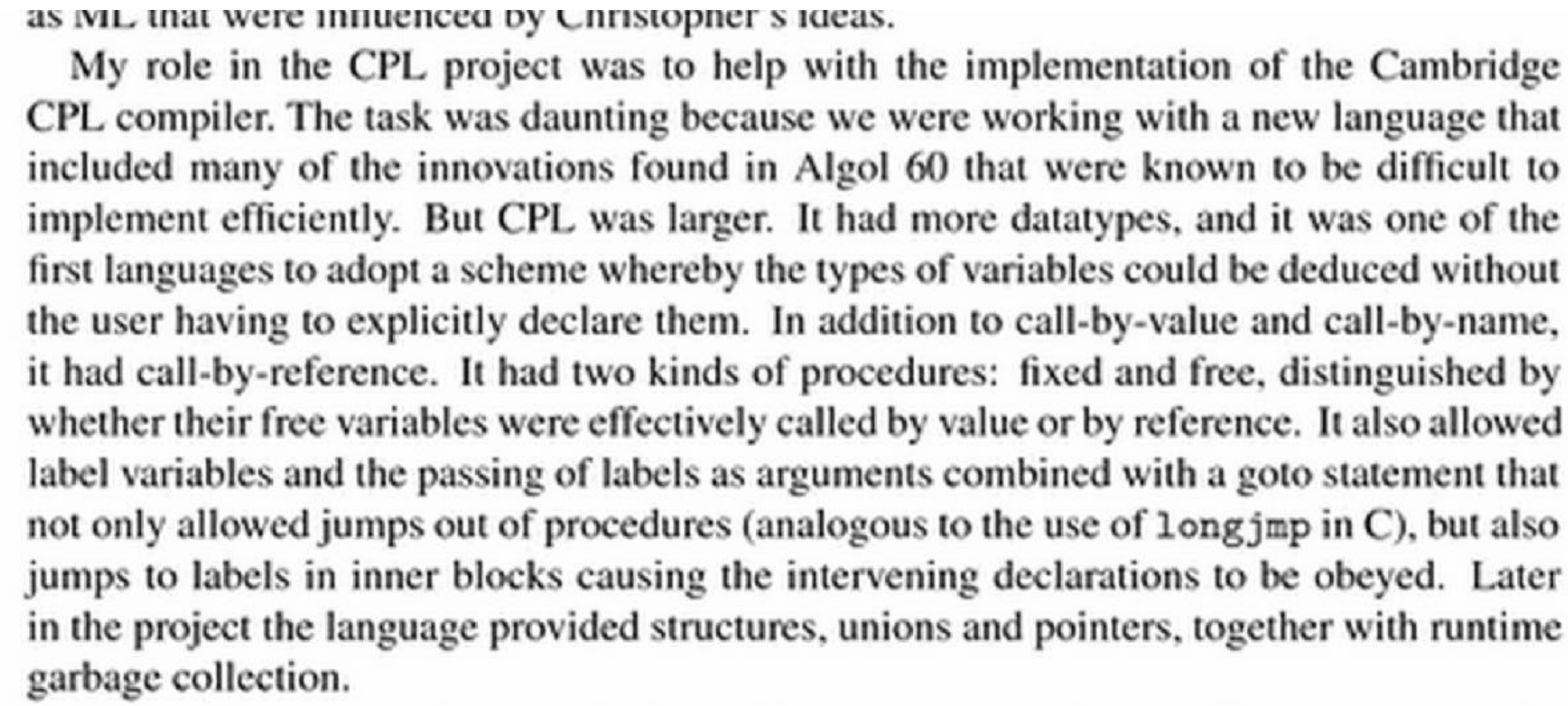
The main features of CPL

By D. W. Barron, J. N. Buxton, D. F. Hartley, E. Nixon and C. Strachey

The paper provides an informal account of CPL, a new programming language currently being implemented for the Titan at Cambridge and the Atlas at London University. CPL is based on, and contains the concepts of, ALGOL 60. In addition there are extended data descriptions, command and expression structures, provision for manipulating non-numerical objects, and comprehensive input-output facilities. However, CPL is not just another proposal for the extension of ALGOL 60, but has been designed from first principles and has a logically coherent structure.

http://comjnl.oxfordjournals.org/content/6/2/134.full.pdf+html

Martin Richards started as a research student in 1963



as ML that were influenced by Christopher's ideas.

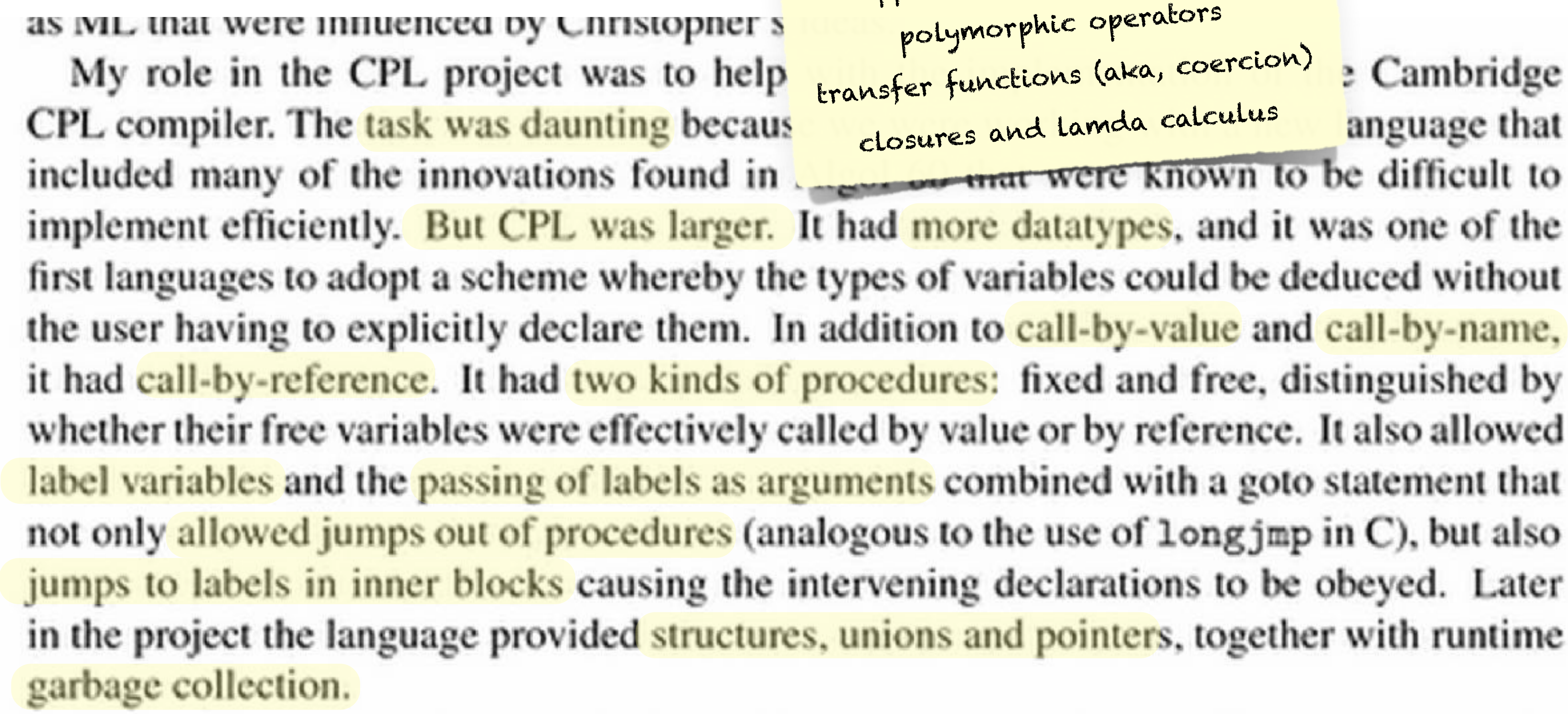
My role in the CPL project was to help with the implementation of the Cambridge CPL compiler. The task was daunting because we were working with a new language that included many of the innovations found in Algol 60 that were known to be difficult to implement efficiently. But CPL was larger. It had more datatypes, and it was one of the first languages to adopt a scheme whereby the types of variables could be deduced without the user having to explicitly declare them. In addition to call-by-value and call-by-name, it had call-by-reference. It had two kinds of procedures: fixed and free, distinguished by whether their free variables were effectively called by value or by reference. It also allowed label variables and the passing of labels as arguments combined with a goto statement that not only allowed jumps out of procedures (analogous to the use of `longjmp` in C), but also jumps to labels in inner blocks causing the intervening declarations to be obeyed. Later in the project the language provided structures, unions and pointers, together with runtime garbage collection.

"Christopher Strachey and the Cambridge CPL Compiler", Martin Richards

From David Hartley's article "CPL: Failed Venture or Noble Ancestor?" (2013)

Martin Richards started as a research student in 1963

double floating point precision
support for complex numbers
polymorphic operators
transfer functions (aka, coercion)
closures and lamda calculus



as ML that were influenced by Christopher S

My role in the CPL project was to help e Cambridge CPL compiler. The task was daunting becaus language that included many of the innovations found in Algol 60 that were known to be difficult to implement efficiently. But CPL was larger. It had more datatypes, and it was one of the first languages to adopt a scheme whereby the types of variables could be deduced without the user having to explicitly declare them. In addition to call-by-value and call-by-name, it had call-by-reference. It had two kinds of procedures: fixed and free, distinguished by whether their free variables were effectively called by value or by reference. It also allowed label variables and the passing of labels as arguments combined with a goto statement that not only allowed jumps out of procedures (analogous to the use of `longjmp` in C), but also jumps to labels in inner blocks causing the intervening declarations to be obeyed. Later in the project the language provided structures, unions and pointers, together with runtime garbage collection.

"Christopher Strachey and the Cambridge CPL Compiler", Martin Richards

From David Hartley's article "CPL: Failed Venture or Noble Ancestor?" (2013)

CPL was once compared to the invention of a pill that could cure every type of ill.

http://s3.amazonaws.com/rapgenius/BIg-Pill.jpg

From David Hartley's article "CPL: Failed Venture or Noble Ancestor?" (2013)

Writing a compiler for CPL was too difficult.

Cambridge never succeeded writing a working CPL compiler.

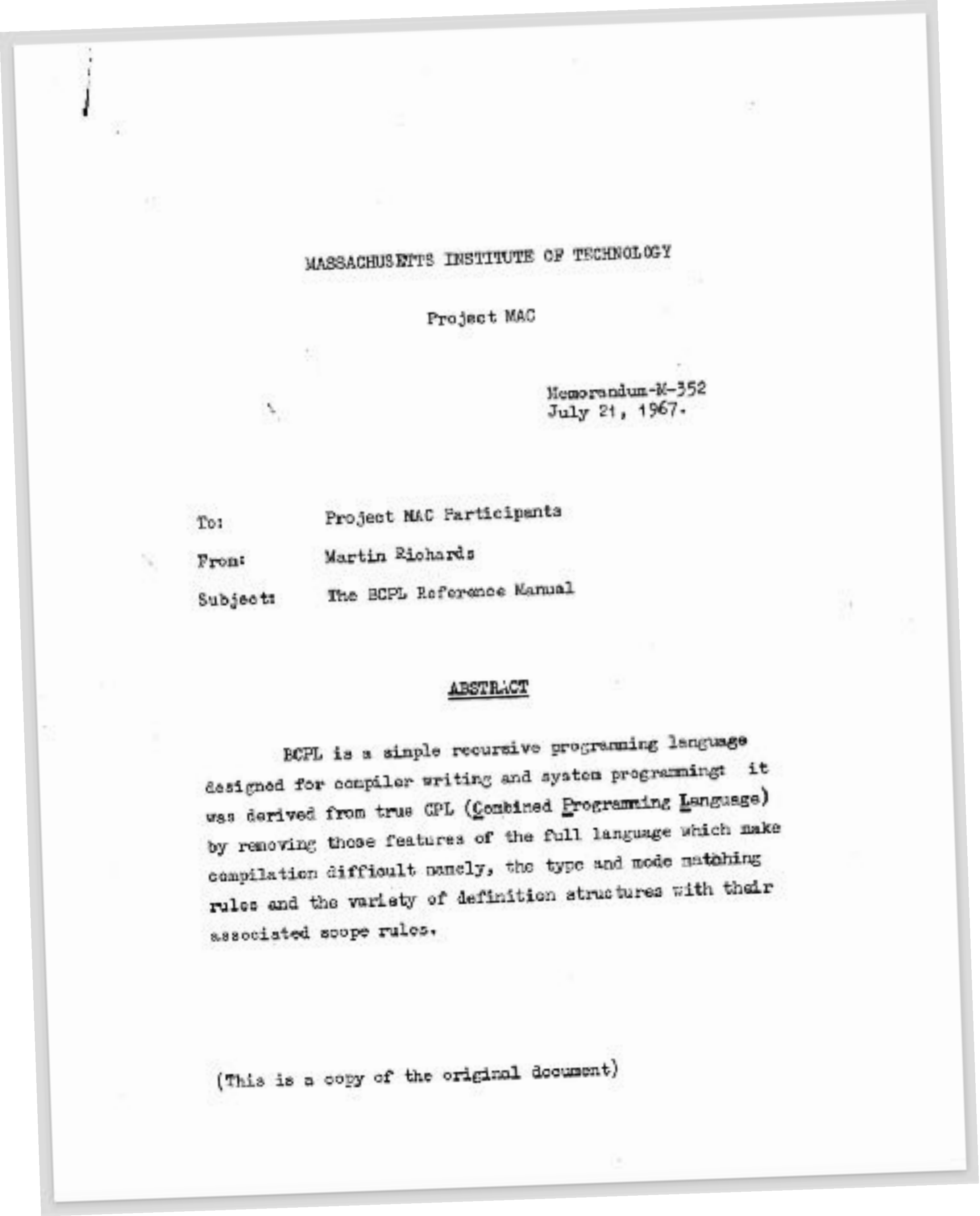
Development on CPL ended December 1966.

Inspired by his work on CPL, Martin Richards wanted to create a language:

- that was simple to compile
- with direct mapping to machine code
- that assumes the programmer know what he is doing

"The philosophy of BCPL is not one of the tyrant who thinks he knows best and lay down the law on what is and what is not allowed; rather, BCPL acts more as a servant offering his services to the best of his ability without complaint, even when confronted with apparent nonsense. The programmer is always assumed to know what he is doing and is not hemmed in by petty restrictions."

The BCPL Reference Manual, Martin Richards, July 1967



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Memorandum-M-352
July 21, 1967.

To: Project MAC Participants

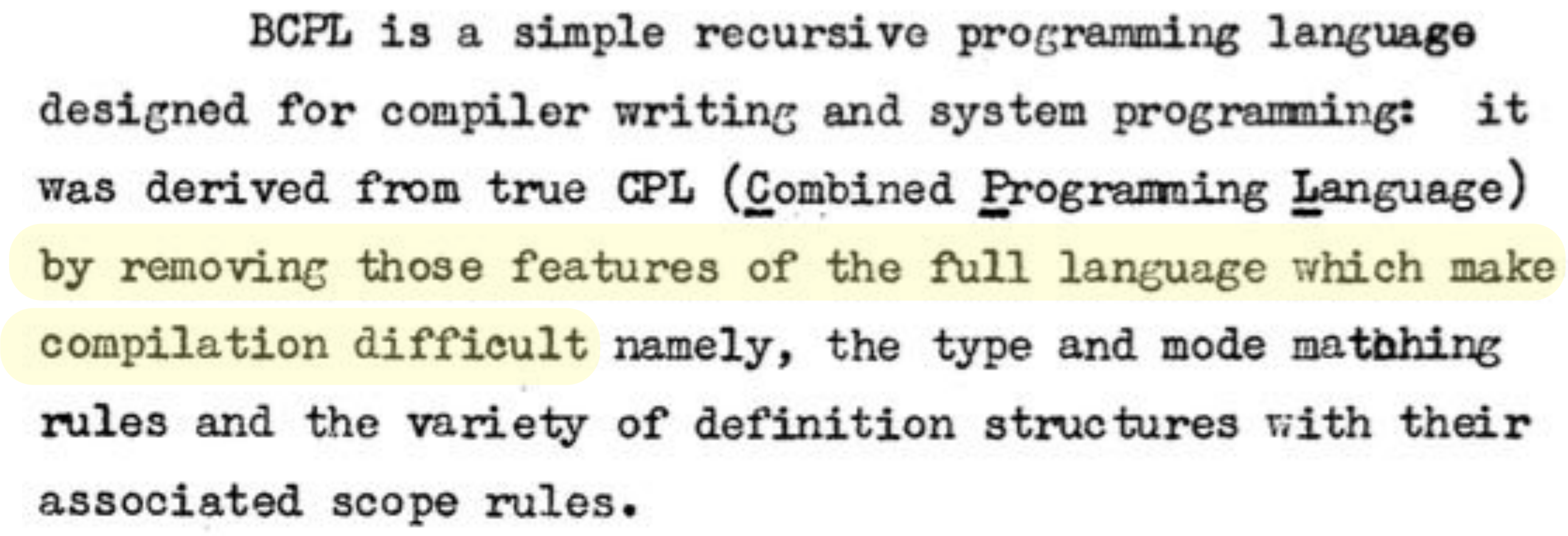
From: Martin Richards

Subject: The BCPL Reference Manual

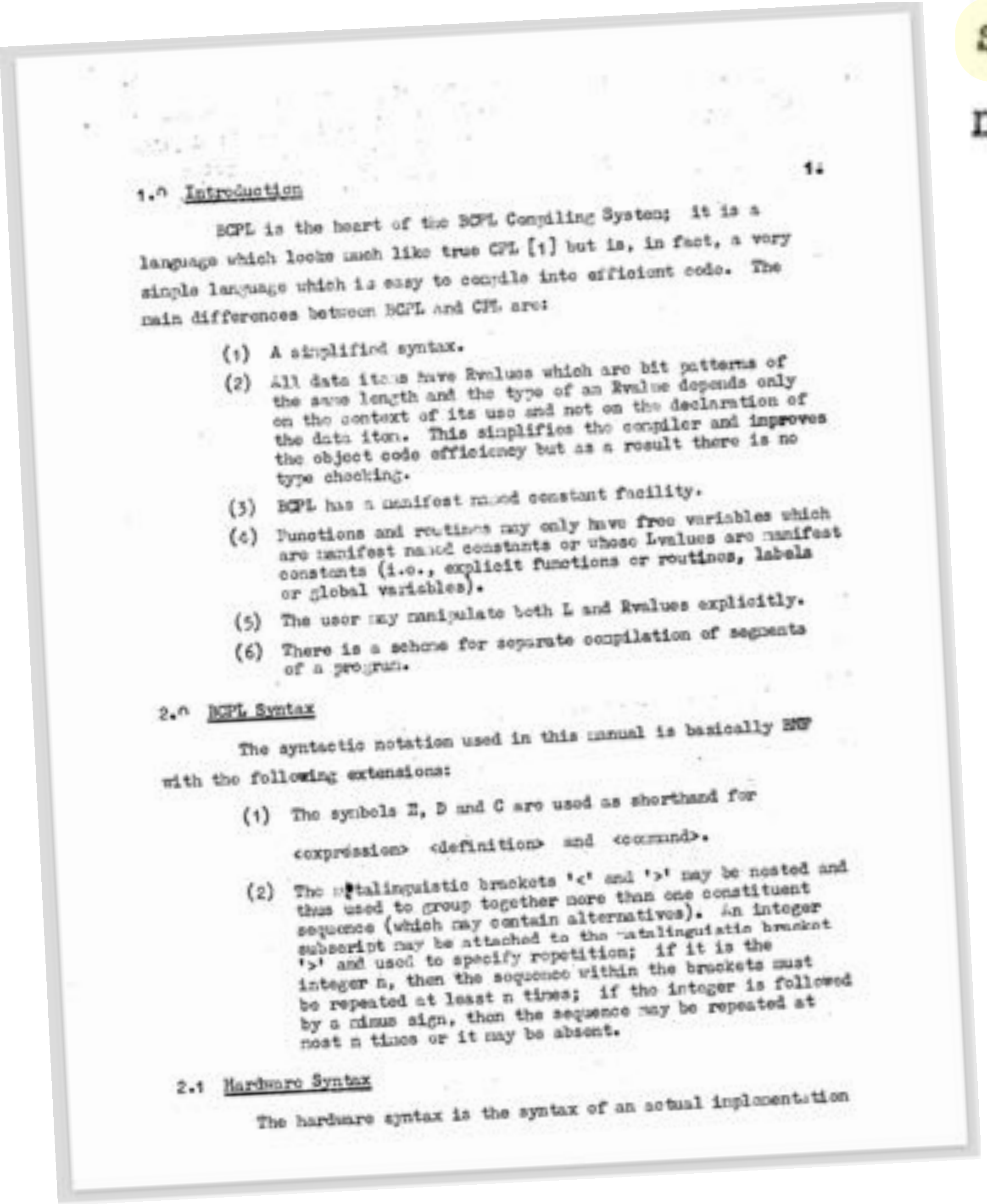
ABSTRACT

BCPL is a simple recursive programming language designed for compiler writing and system programming: it was derived from true CPL (Combined Programming Language) by removing those features of the full language which make compilation difficult namely, the type and mode matbhing rules and the variety of definition structures with their associated scope rules.

(This is a copy of the original document)



BCPL is a simple recursive programming language designed for compiler writing and system programming: it was derived from true CPL (Combined Programming Language) by removing those features of the full language which make compilation difficult namely, the type and mode matbhing rules and the variety of definition structures with their associated scope rules.



1.0 Introduction

1.

BCPL is the heart of the BCPL Compiling System; it is a language which looke much like true CPL [1] but is, in fact, a very simple language which is easy to compile into efficient code. The main differences between BCPL and CPL are:

(1) A simplified syntax.

(2) All data items have Rvalues which are bit patterns of the same length and the type of an Rvalue depends only on the context of its use and not on the declaration of the data item. This simplifies the compiler and improves the object code efficiency but as a result there is no type checking.

(3) BCPL has a manifest named constant facility.

(4) Functions and routines may only have free variables which are manifest named constants or whose Lvalues are manifest constants (i.e., explicit functions or routines, labels or global variables).

(5) The user may manipulate both L and Rvalues explicitly.

(6) There is a scheme for separate compilation of segments of a program.

2.0 BCPL Syntax

The syntactic notation used in this manual is basically BNF with the following extensions:

(1) The symbols E, D and C are used as shorthand for <expression> <definition> and <command>.

(2) The metalinguistic brackets '<' and '>' may be nested and thus used to group together more than one constituent sequence (which may contain alternatives). An integer subscript may be attached to the metalinguistic bracket '>' and used to specify repetition; if it is the integer n, then the sequence within the brackets must be repeated at least n times; if the integer is followed by a minus sign, then the sequence may be repeated at most n times or it may be absent.

2.1 Hardware Syntax

The hardware syntax is the syntax of an actual implementation

BCPL is the heart of the BCPL Compiling System; it is language which looke nuch like true CPL [1] but is, in fact, a sinple language which is easy to compile into efficient code. The nain differences between BCPL and CPL are:

(1) A sinplified syntax.

(2) All data itens have Rvalues which are bit patterns of the same length and the type of an Rvalue depends only on the context of its use and not on the declaration of the data iten. This sinplifies the compiler and improves the object code efficiency but as a rosult there is no type checking.

(3) BCPL has a nanifest naned constant facility.

(4) Functions and routines may only have free variables which are nanifest naned constants or whose Lvalues are nanifest constants (i.e., explicit functions or routines, labels or global variables).

(5) The user nay manipulate both L and Rvalues explicitly.

(6) There is a schene for separate compilation of segnents of a program.

Martin Richards joined MIT's Project MAC

and through the MULTICS project the Bell Labs people learned about this beautiful language called BCPL - a language exactly to the taste of Ken and Dennis.

MIT

GE

Bell Labs

Humble fans meet Martin Richards, the inventor of BCPL

Jon Jagger, Martin Richards, Olve Maudal

Computer Laboratory, Cambridge, December 2014

B was the link between BCPL and C

From an interview with Ken Thompson in 1989

Interviewer: Did you develop B?

Thompson: I did B.

Interviewer: As a subset of BCPL?

Thompson: It wasn't a subset. It was almost exactly the same.

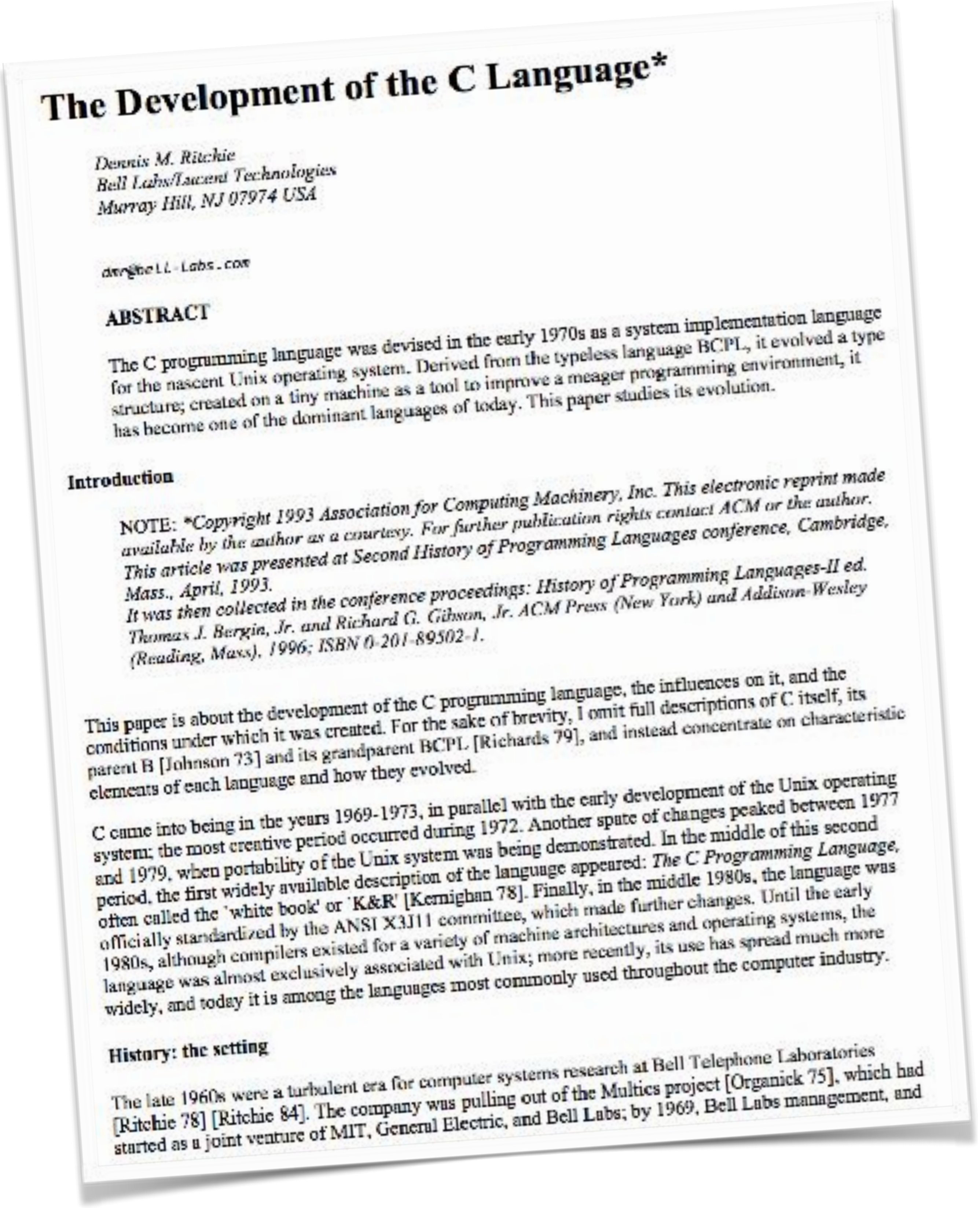
...

Thompson: It was the same language as BCPL, it looked completely different, syntactically it was, you know, a redo. The semantics was exactly the same as BCPL. And in fact the syntax of it was, if you looked at, you didn't look too close, you would say it was C. Because in fact it was C, without types.

...

http://www.princeton.edu/~hos/mike/transcripts/thompson.htm

From the HOPL article by Dennis Ritchie in 1993



The Development of the C Language*

Dennis M. Ritchie
Bell Labs/Lucent Technologies
Murray Hill, NJ 07974 USA

dmr@bell-labs.com

ABSTRACT

The C programming language was devised in the early 1970s as a system implementation language for the nascent Unix operating system. Derived from the typeless language BCPL, it evolved a type structure; created on a tiny machine as a tool to improve a meager programming environment, it has become one of the dominant languages of today. This paper studies its evolution.

Introduction

NOTE: **Copyright 1993 Association for Computing Machinery, Inc. This electronic reprint made available by the author as a courtesy. For further publication rights contact ACM or the author. This article was presented at Second History of Programming Languages conference, Cambridge, Mass., April, 1993.*
It was then collected in the conference proceedings: History of Programming Languages-II ed. Thomas J. Bergin, Jr. and Richard G. Gibson, Jr. ACM Press (New York) and Addison-Wesley (Reading, Mass), 1996; ISBN 0-201-89502-1.

This paper is about the development of the C programming language, the influences on it, and the conditions under which it was created. For the sake of brevity, I omit full descriptions of C itself, its parent B [Johnson 73] and its grandparent BCPL [Richards 79], and instead concentrate on characteristic elements of each language and how they evolved.

C came into being in the years 1969-1973, in parallel with the early development of the Unix operating system; the most creative period occurred during 1972. Another spate of changes peaked between 1977 and 1979, when portability of the Unix system was being demonstrated. In the middle of this second period, the first widely available description of the language appeared: *The C Programming Language*, often called the `white book' or `K&R' [Kernighan 78]. Finally, in the middle 1980s, the language was officially standardized by the ANSI X3J11 committee, which made further changes. Until the early 1980s, although compilers existed for a variety of machine architectures and operating systems, the language was almost exclusively associated with Unix; more recently, its use has spread much more widely, and today it is among the languages most commonly used throughout the computer industry.

History: the setting

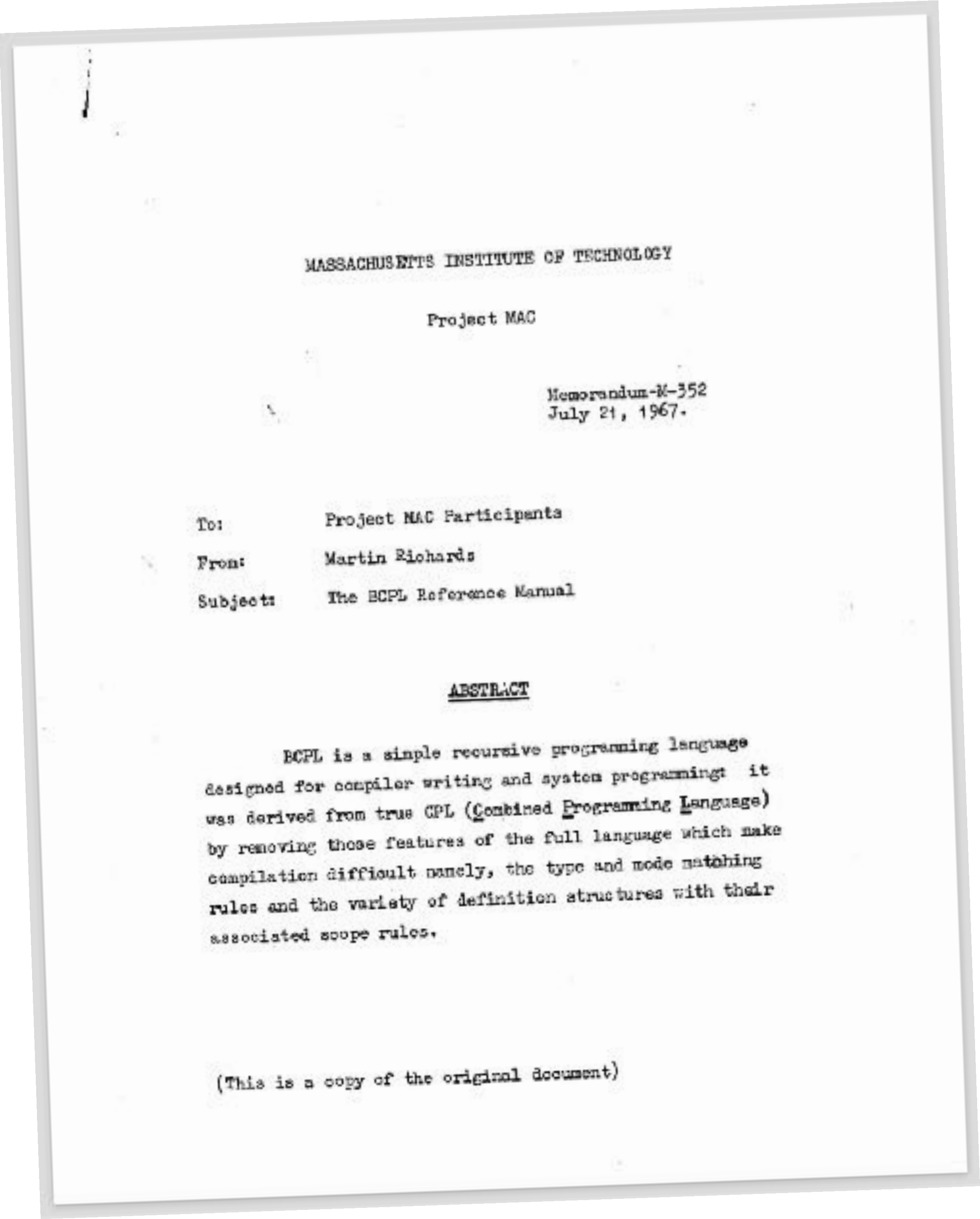
The late 1960s were a turbulent era for computer systems research at Bell Telephone Laboratories [Ritchie 78] [Ritchie 84]. The company was pulling out of the Multics project [Organick 75], which had started as a joint venture of MIT, General Electric, and Bell Labs; by 1969, Bell Labs management, and

The C programming language was devised in the early 1970s as a system implementation language for the nascent Unix operating system. Derived from the typeless language BCPL, it evolved a type structure; created on a tiny machine as a tool to improve a meager programming environment, it has become one of the dominant languages of today.

…

BCPL, B and C differ syntactically in many details, but broadly they are similar.

The BCPL Reference Manual, Martin Richards, July 1967



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Memorandum-M-352
July 21, 1967.

To: Project MAC Participants

From: Martin Richards

Subject: The BCPL Reference Manual

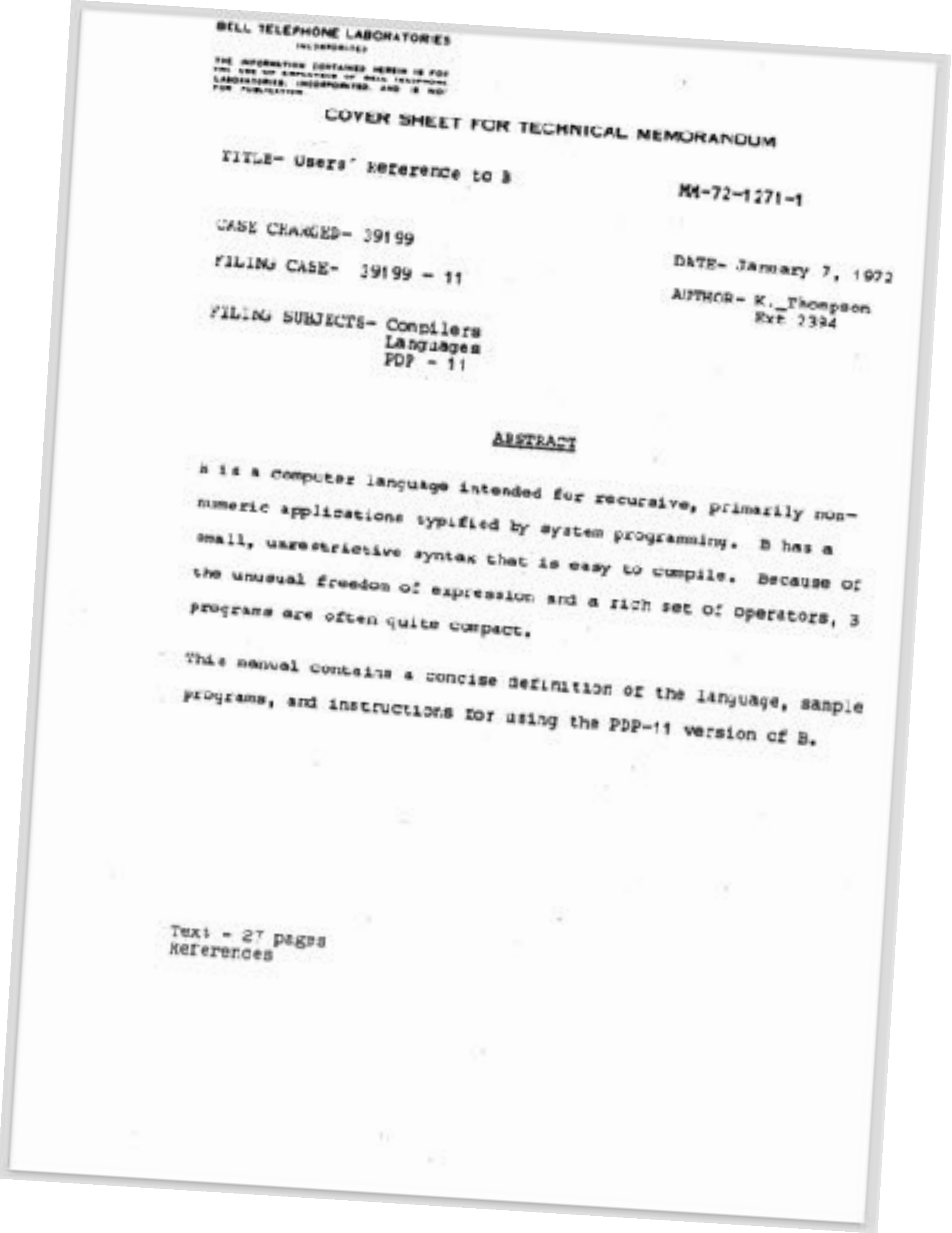
ABSTRACT

BCPL is a simple recursive programming language designed for compiler writing and system programming: it was derived from true CPL (Combined Programming Language) by removing those features of the full language which make compilation difficult namely, the type and mode matching rules and the variety of definition structures with their associated scope rules.

(This is a copy of the original document)

vs

Users' Reference to B, Ken Thompson, January 1972



BELL TELEPHONE LABORATORIES
INCORPORATED

THE INFORMATION CONTAINED HEREIN IS FOR THE USE OF EMPLOYEES OF BELL TELEPHONE LABORATORIES, INCORPORATED, AND IS NOT FOR PUBLICATION

COVER SHEET FOR TECHNICAL MEMORANDUM

TITLE- Users' Reference to B

MM-72-1271-1

CASE CHARGED- 39199

FILING CASE- 39199 - 11

DATE- January 7, 1972

AUTHOR- K._Thompson
Ext 2394

FILING SUBJECTS- Compilers
Languages
PDP - 11

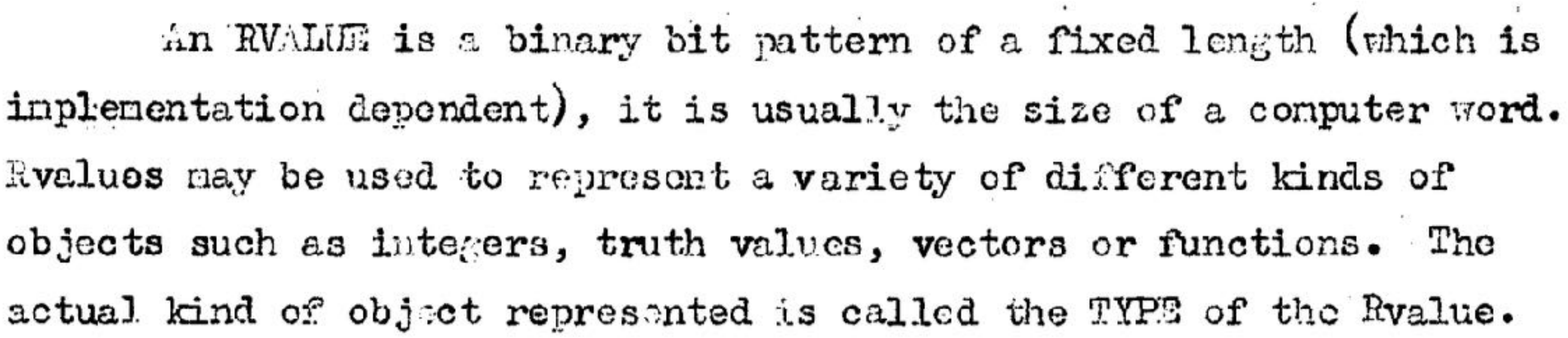
ABSTRACT

B is a computer language intended for recursive, primarily non-numeric applications typified by system programming. B has a small, unrestrictive syntax that is easy to compile. Because of the unusual freedom of expression and a rich set of operators, B programs are often quite compact.

This manual contains a concise definition of the language, sample programs, and instructions for using the PDP-11 version of B.

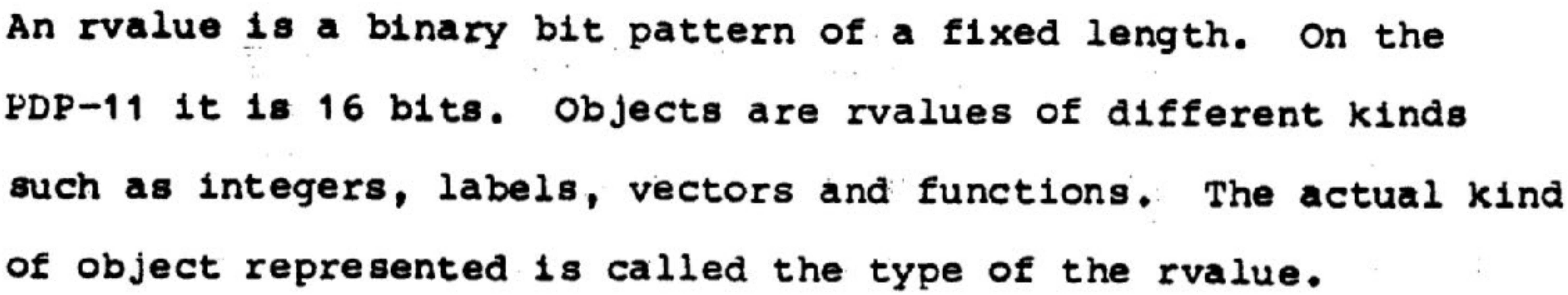
Text - 27 pages
References

excerpt from the BCPL reference manual (Richards, 1967), page 6



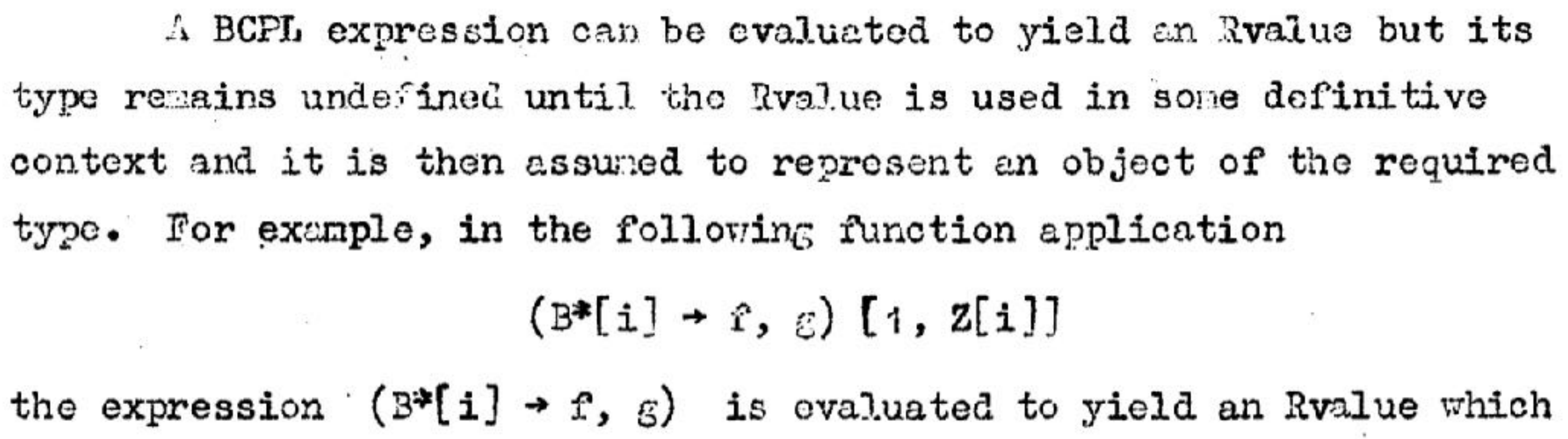
An RVALUE is a binary bit pattern of a fixed length (which is implementation dependent), it is usually the size of a computer word. Rvalues may be used to represent a variety of different kinds of objects such as integers, truth values, vectors or functions. The actual kind of object represented is called the TYPE of the Rvalue.

excerpt from the B reference manual (Thompson, 1972), page 6



An rvalue is a binary bit pattern of a fixed length. On the PDP-11 it is 16 bits. Objects are rvalues of different kinds such as integers, labels, vectors and functions. The actual kind of object represented is called the type of the rvalue.

excerpt from the BCPL reference manual (Richards, 1967), page 6

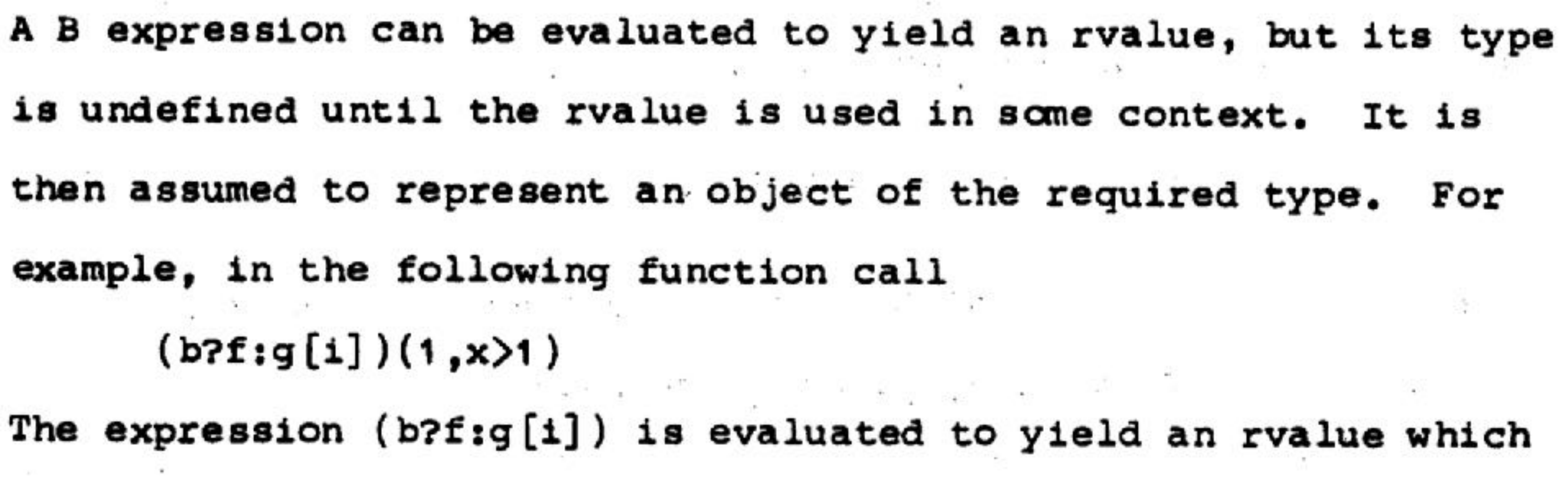


A BCPL expression can be evaluated to yield an Rvalue but its type remains undefined until the Rvalue is used in some definitive context and it is then assumed to represent an object of the required type. For example, in the following function application

```
(B*[i] → f, g) [1, Z[i]]
```

the expression (B*[i] → f, g) is evaluated to yield an Rvalue which

excerpt from the B reference manual (Thompson, 1972), page 6

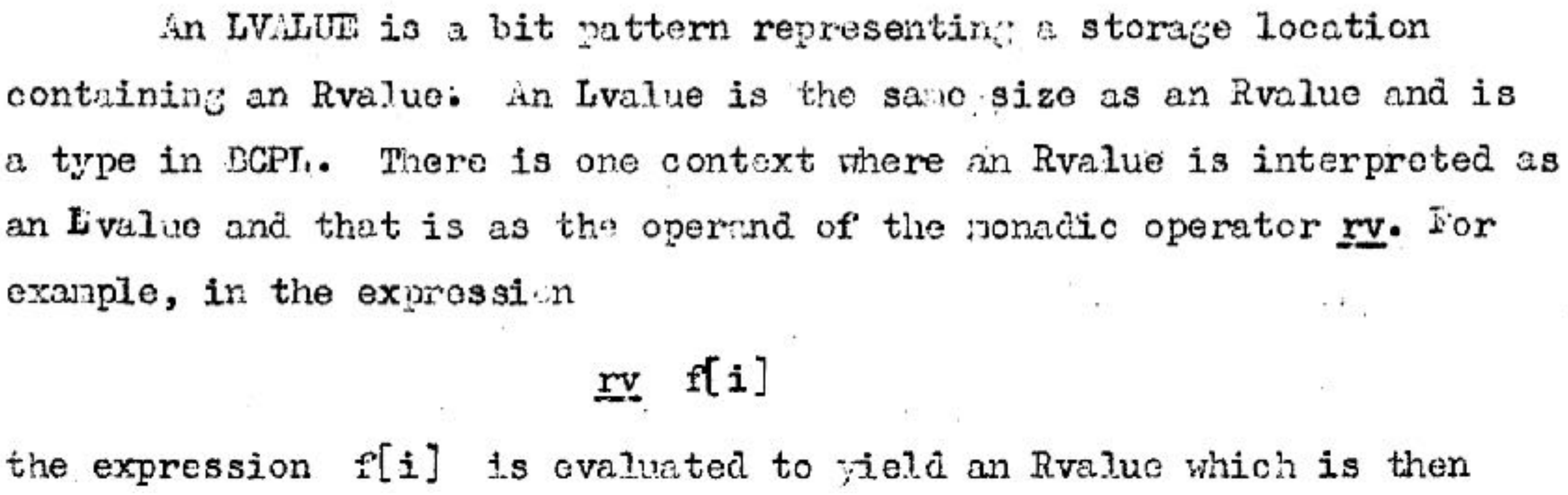


A B expression can be evaluated to yield an rvalue, but its type is undefined until the rvalue is used in some context. It is then assumed to represent an object of the required type. For example, in the following function call

```
(b?f:g[i])(1,x>1)
```

The expression (b?f:g[i]) is evaluated to yield an rvalue which

excerpt from the BCPL reference manual (Richards, 1967), page 6

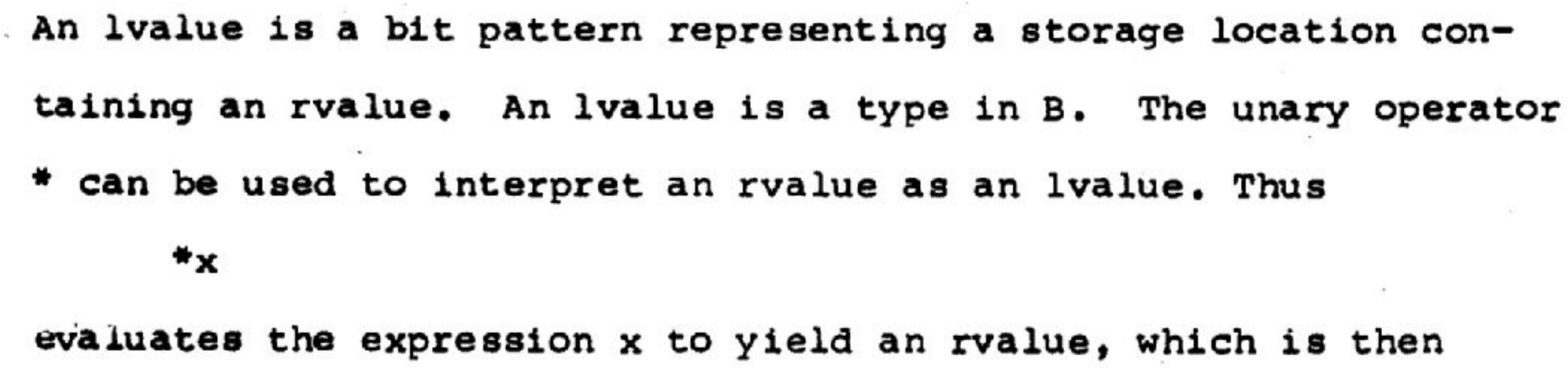


An LVALUE is a bit pattern representing a storage location containing an Rvalue. An Lvalue is the same size as an Rvalue and is a type in BCPL. There is one context where an Rvalue is interpreted as an Lvalue and that is as the operand of the monadic operator rv. For example, in the expression

```
rv f[i]
```

the expression f[i] is evaluated to yield an Rvalue which is then

excerpt from the B reference manual (Thompson, 1972), page 6



An lvalue is a bit pattern representing a storage location containing an rvalue. An lvalue is a type in B. The unary operator * can be used to interpret an rvalue as an lvalue. Thus

```
*x
```

evaluates the expression x to yield an rvalue, which is then

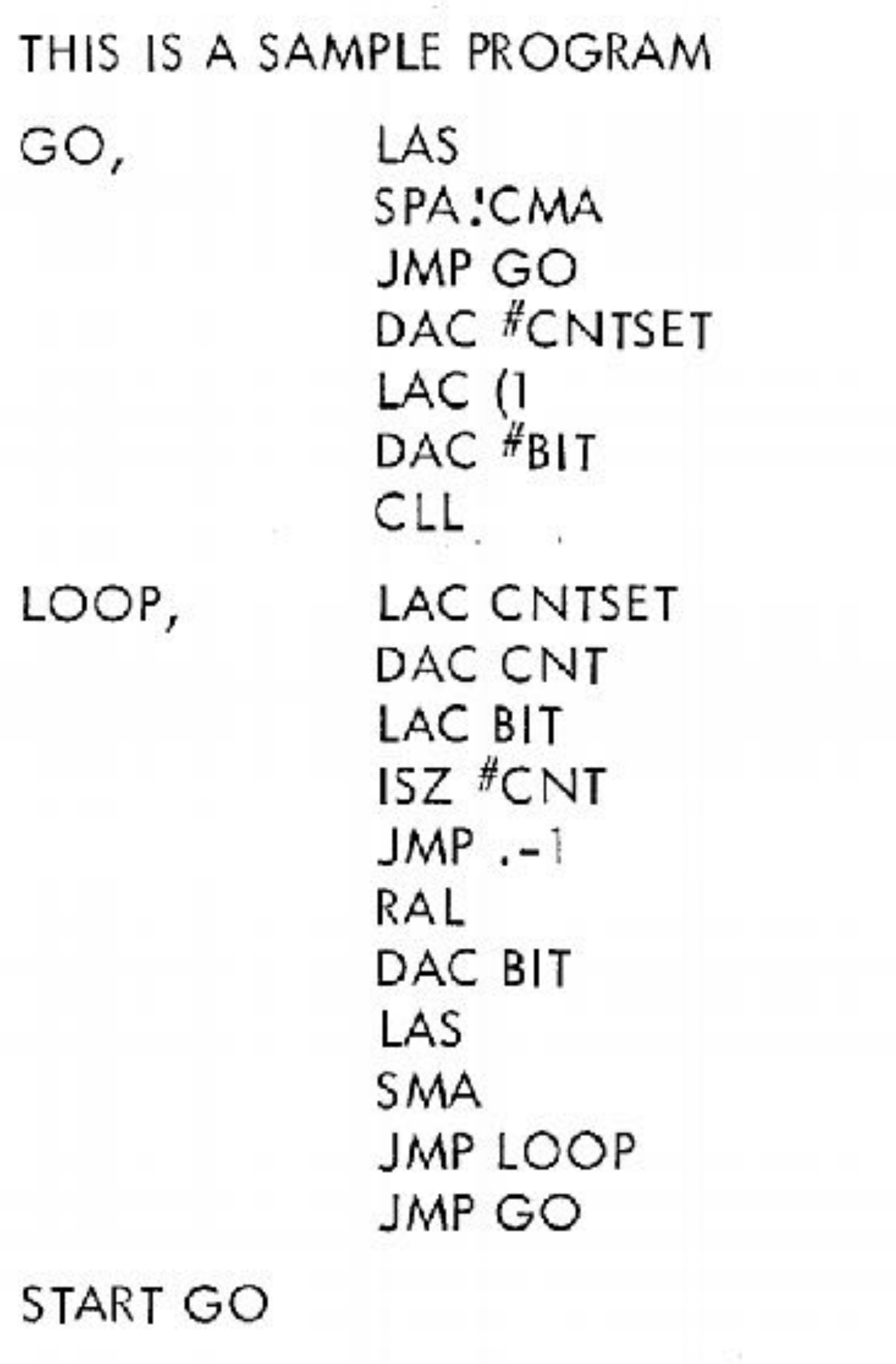
BCPL

- Designed by Martin Richards, appeared in 1966, typeless (everything is a word)
- Influenced by Fortran and Algol
- Intended for writing compilers for other languages
- Simplified version of CPL by "removing those features of the full language which make compilation difficult"

```
GET "LIBHDR"

GLOBAL $(
        COUNT: 200
        ALL: 201
$)

LET TRY(LD, ROW, RD) BE
        TEST ROW = ALL THEN
                COUNT := COUNT + 1
        ELSE $(
                LET POSS = ALL & ~(LD | ROW | RD)
                UNTIL POSS = 0 DO $(
                        LET P = POSS & -POSS
                        POSS := POSS - P
                        TRY(LD + P << 1, ROW + P, RD + P >> 1)
                $)
        $)

LET START() = VALOF $(
        ALL := 1
        FOR I = 1 TO 12 DO $(
                COUNT := 0
                TRY(0, 0, 0)
                WRITEF("%I2-QUEENS PROBLEM HAS %I5 SOLUTIONS*N", I, COUNT)
                ALL := 2 * ALL + 1
        $)
        RESULTIS 0
$)
```

PDP-7

(18-bit computer, introduced 1965)



```
THIS IS A SAMPLE PROGRAM

GO,             LAS
                SPA!CMA
                JMP GO
                DAC #CNTSET
                LAC (1
                DAC #BIT
                CLL

LOOP,           LAC CNTSET
                DAC CNT
                LAC BIT
                ISZ #CNT
                JMP .-1
                RAL
                DAC BIT
                LAS
                SMA
                JMP LOOP
                JMP GO

START GO
```

B

Designed by Ken Thompson, appeared in ~1969, typeless (everything is a word)
"BCPL squeezed into 8K words of memory and filtered through Thompson's brain"

```
/* The following program will calculate the constant e-2 to about
   4000 decimal digits, and print it 50 characters to the line in
   groups of 5 characters. */

main() {
    extrn putchar, n, v;
    auto i, c, col, a;

    i = col = 0;
    while(i<n)
        v[i++] = 1;
    while(col<2*n) {
        a = n+1 ;
        c = i = 0;
        while (i<n) {
            c =+ v[i] *10;
            v[i++]  = c%a;
            c =/ a--;
        }

        putchar(c+'0');
        if(!(++col%5))
            putchar(col%50?' ': '*n');
    }
    putchar('*n*n');
}

v[2000];
n 2000;
```

```
if
else
while
switch
case

goto
return

auto
extrn
```

PDP-11

- 16-bit computer
- introduced 1970
- orthogonal instruction set
- byte-oriented

C

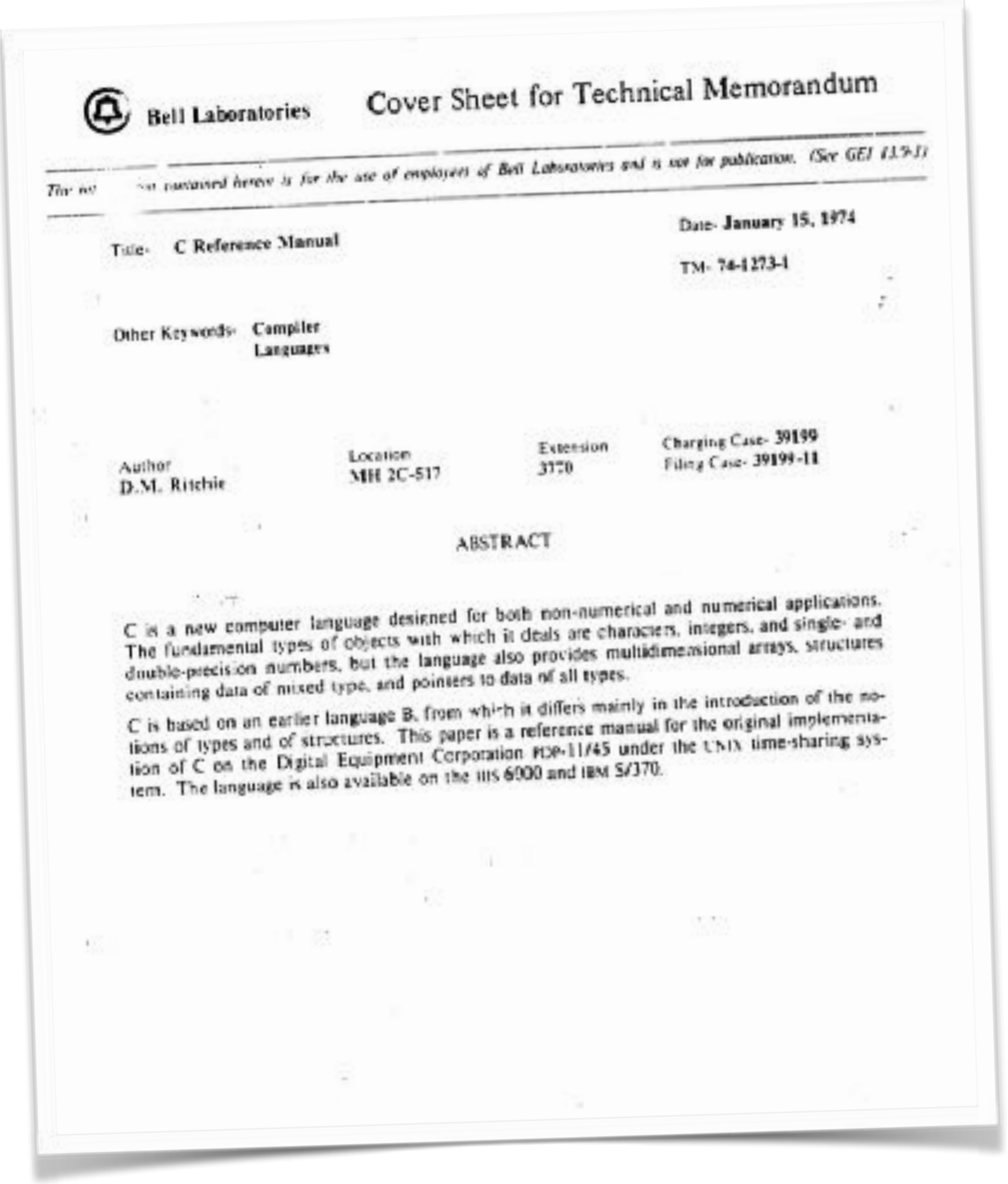
Designed by Dennis Ritchie and Ken Thompson
Developed during 1969-1972 in parallel with Unix
Data types added to the language to support the PDP-11

```
/* Early C example */

mystrcpy(s,t)
char *s;
char *t;
{
    int i;

    for (i = 0; (*s++ = *t++) != '\0'; i++)
        ;
    return(i);
}

main()
{
    char str1[10];
    char str2[] = "Hello!";
    int len = mystrcpy(str1, str2);
    int i;
    for (i = 0; i < len; i++)
        putchar(str1[i]);
    exit(0);
}
```

```
if
else
while
switch
case
default
do
for

goto
return
break
continue
entry

auto
```
~~extrn~~
```
extern
static
register
```

```
int
char
float
double
struct
sizeof
```

The C Reference Manual, Dennis Ritchie, Jan 1974 (aka C74)



Bell Laboratories

Cover Sheet for Technical Memorandum

The [illegible] contained herein is for the use of employees of Bell Laboratories and is not for publication. (See GEI 13.9-3)

Title- C Reference Manual

Date- January 15, 1974

TM- 74-1273-1

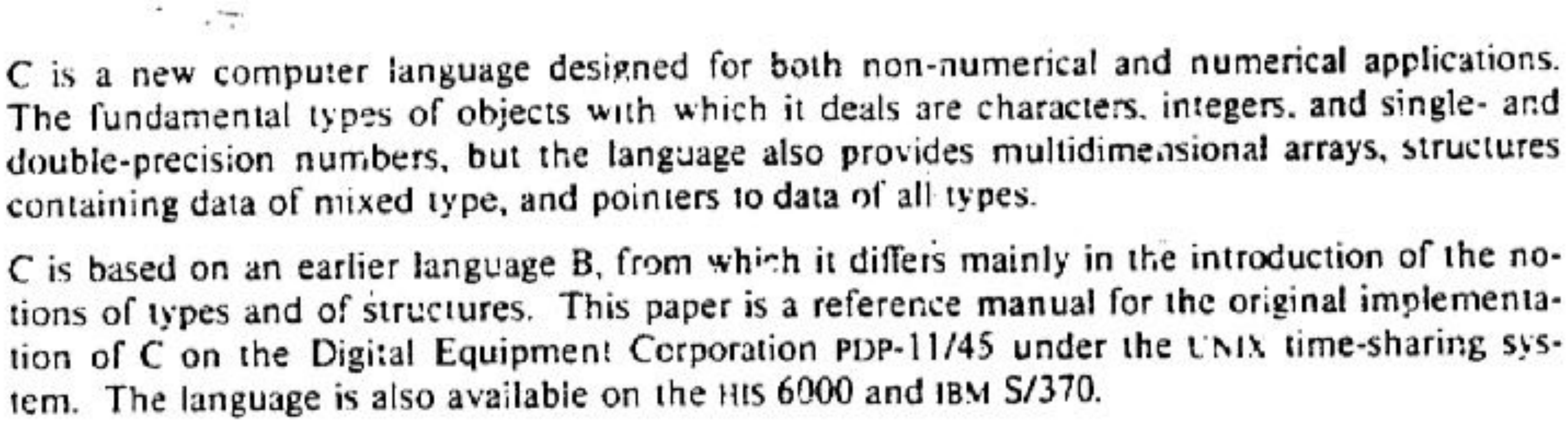
Other Keywords- Compiler
Languages

Author	Location	Extension	
D.M. Ritchie	MH 2C-517	3370	Charging Case- 39199 Filing Case- 39199-11

ABSTRACT

C is a new computer language designed for both non-numerical and numerical applications. The fundamental types of objects with which it deals are characters, integers, and single- and double-precision numbers, but the language also provides multidimensional arrays, structures containing data of mixed type, and pointers to data of all types.

C is based on an earlier language B, from which it differs mainly in the introduction of the notions of types and of structures. This paper is a reference manual for the original implementation of C on the Digital Equipment Corporation PDP-11/45 under the UNIX time-sharing system. The language is also available on the HIS 6000 and IBM S/370.



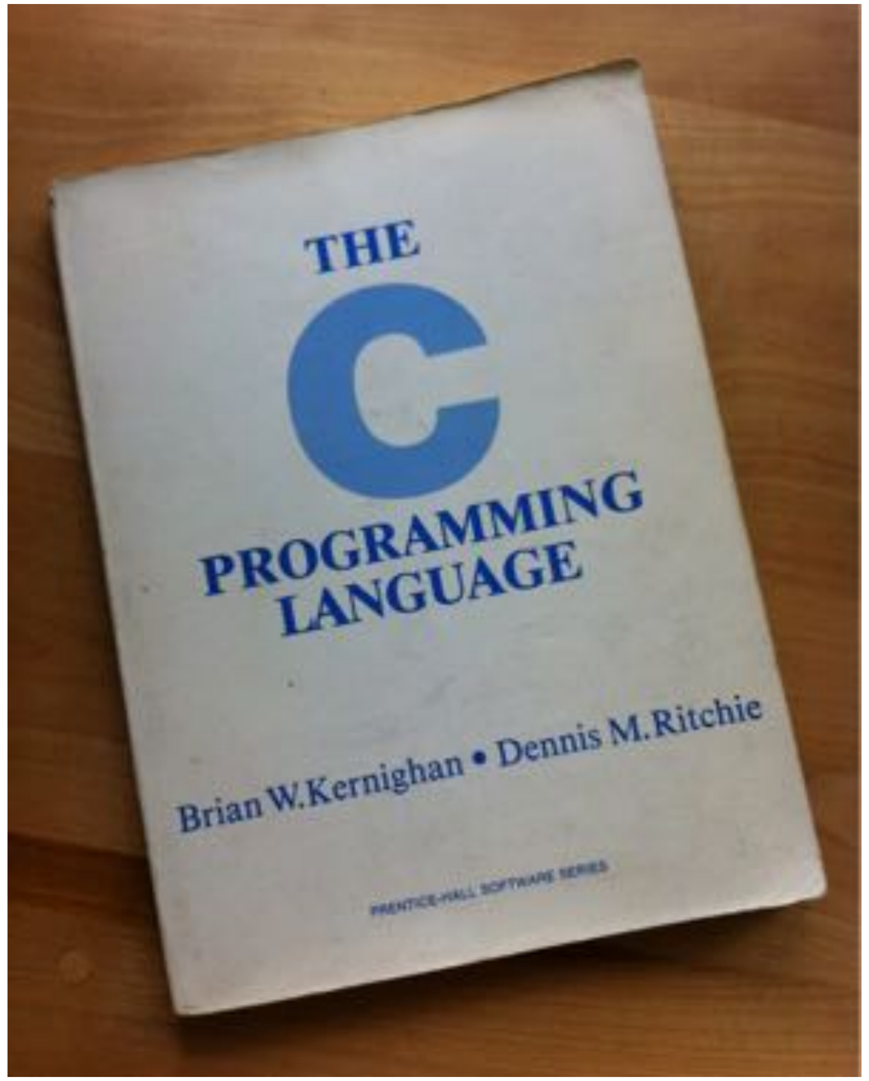
C is a new computer language designed for both non-numerical and numerical applications. The fundamental types of objects with which it deals are characters, integers, and single- and double-precision numbers, but the language also provides multidimensional arrays, structures containing data of mixed type, and pointers to data of all types.

C is based on an earlier language B, from which it differs mainly in the introduction of the notions of types and of structures. This paper is a reference manual for the original implementation of C on the Digital Equipment Corporation PDP-11/45 under the UNIX time-sharing system. The language is also available on the HIS 6000 and IBM S/370.

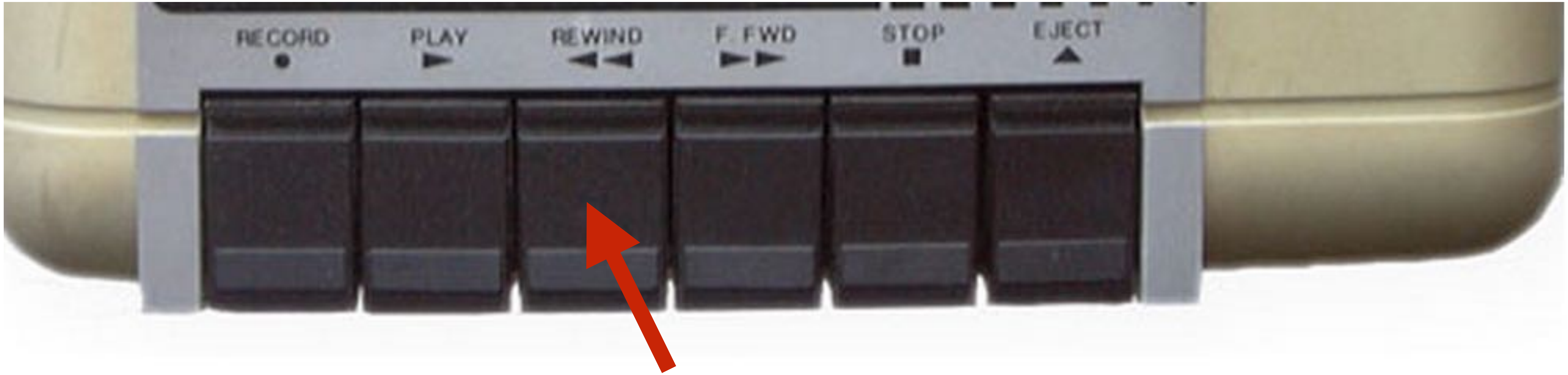
Fun fact: The C74 reference manual does not mention BCPL at all.
It does not even mention the B reference manual by Ken Thompson.

K&R C

The seminal book "The C Programming Language" (1978) acted for a long time as the only formal definition of the language. And PCC was the reference implementation for C.


THE
C
PROGRAMMING
LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
PRENTICE-HALL SOFTWARE SERIES

"C became the most successful language ever."


RECORD
●
PLAY
▶
REWIND
◀◀
F. FWD
▶▶
STOP
■
EJECT
▲

in the Computing Laboratory at University of Cambridge.

in the mid/late 70’

Bjarne was working on his PhD thesis

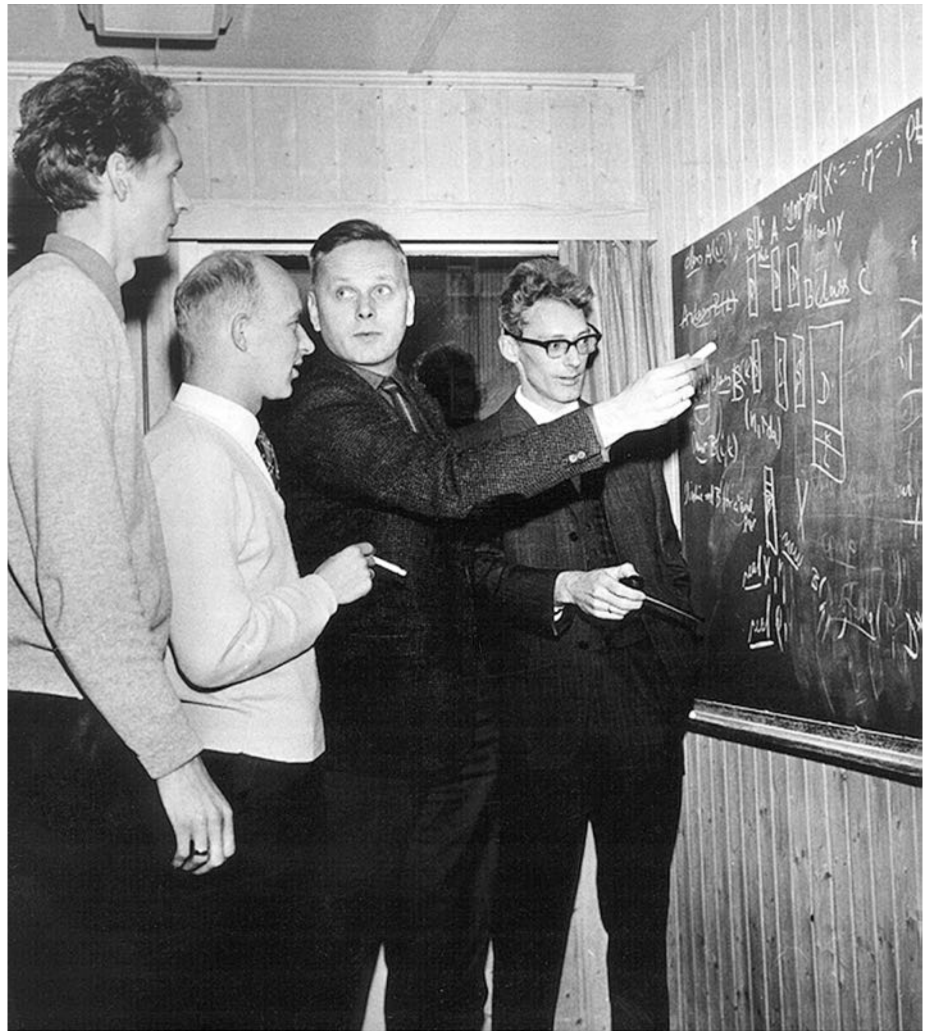
Bjarne

"The Design and Evolution of C++", Bjarne Stroustrup, 1994

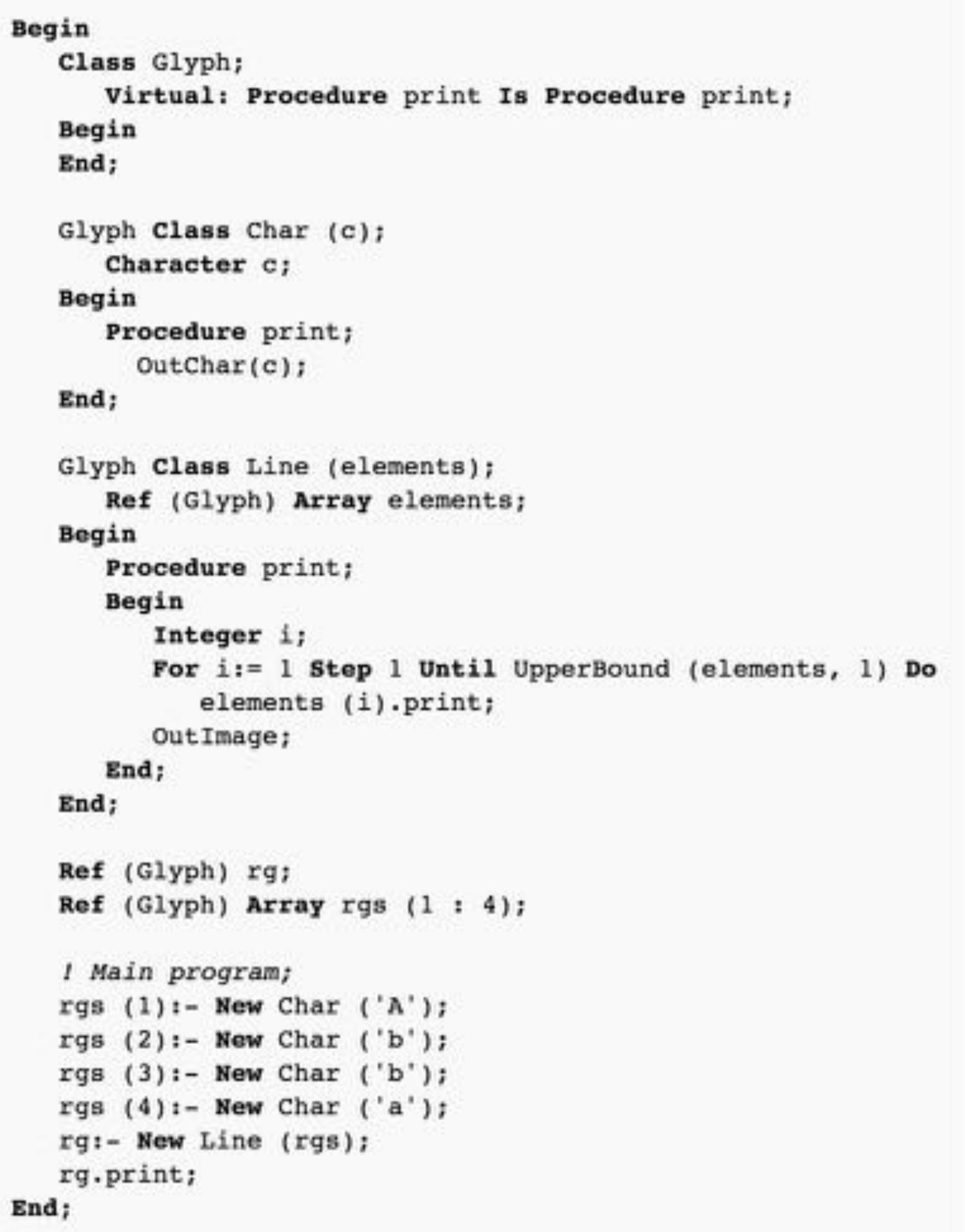
Cambridge Computing, The first 75 years, Haroon Ahmed, 2013

http://computersweden.idg.se/polopoly_fs/1.346563!imageManager/1326219611.jpg

He was working on a simulator to study alternatives for the organization of system software for distributed systems.
The initial version of this simulator was written in Simula

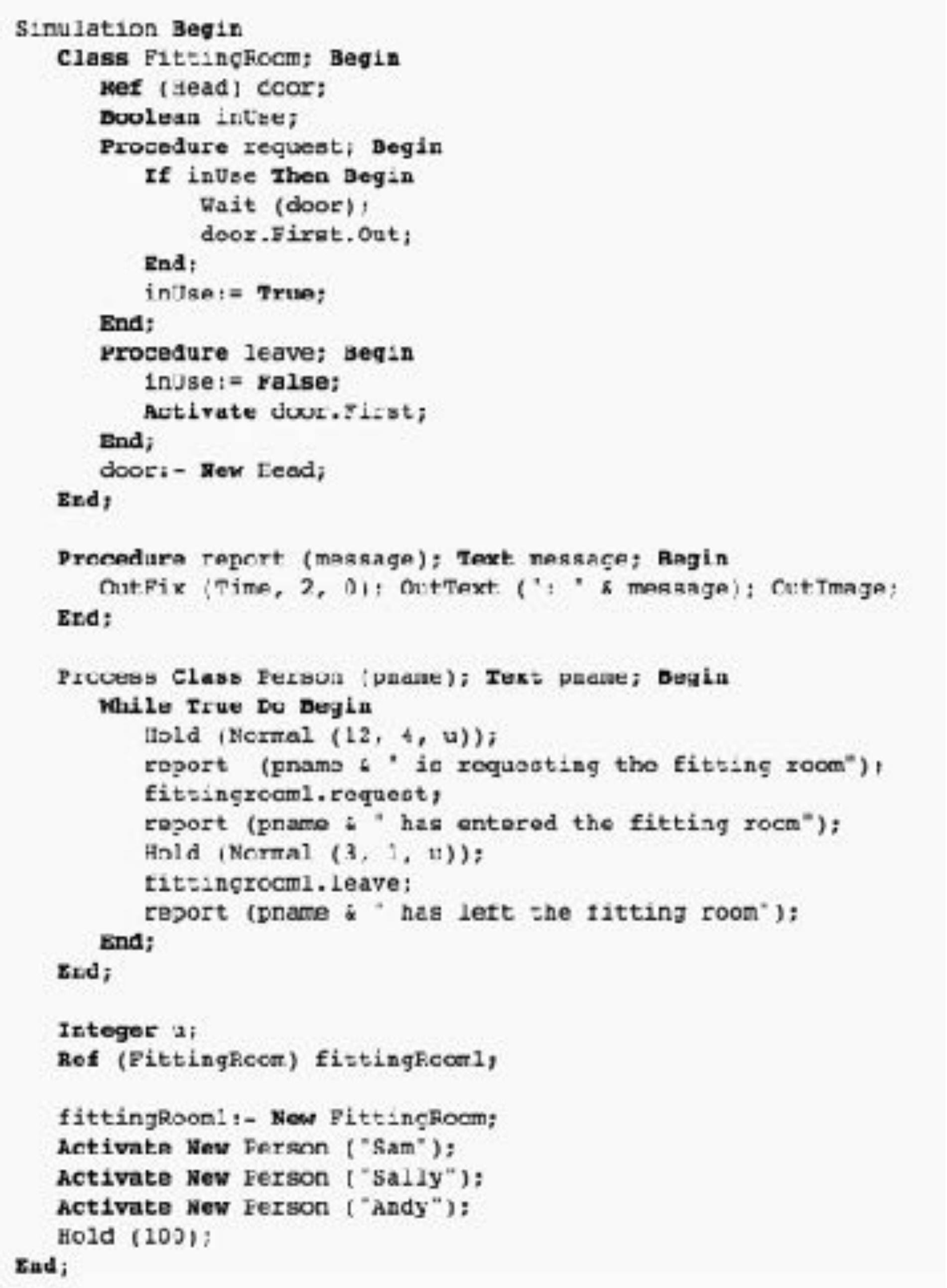


Bjørn Myrhaug, Sigurd Kubosh, Kristen Nygard and Ole Johan Dahl by the "Simula blackboard"



```
Begin
   Class Glyph;
      Virtual: Procedure print Is Procedure print;
   Begin
   End;

   Glyph Class Char (c);
      Character c;
   Begin
      Procedure print;
        OutChar(c);
   End;

   Glyph Class Line (elements);
      Ref (Glyph) Array elements;
   Begin
      Procedure print;
      Begin
         Integer i;
         For i:= 1 Step 1 Until UpperBound (elements, 1) Do
            elements (i).print;
         OutImage;
      End;
   End;

   Ref (Glyph) rg;
   Ref (Glyph) Array rgs (1 : 4);

   ! Main program;
   rgs (1):- New Char ('A');
   rgs (2):- New Char ('b');
   rgs (3):- New Char ('b');
   rgs (4):- New Char ('a');
   rg:- New Line (rgs);
   rg.print;
End;
```

object oriented programming



```
Simulation Begin
   Class FittingRoom; Begin
      Ref (Head) door;
      Boolean inUse;
      Procedure request; Begin
         If inUse Then Begin
             Wait (door);
             door.First.Out;
         End;
         inUse:= True;
      End;
      Procedure leave; Begin
         inUse:= False;
         Activate door.First;
      End;
      door:- New Head;
   End;

   Procedure report (message); Text message; Begin
      OutFix (Time, 2, 0); OutText (": " & message); OutImage;
   End;

   Process Class Person (pname); Text pname; Begin
      While True Do Begin
         Hold (Normal (12, 4, u));
         report  (pname & " is requesting the fitting room");
         fittingroom1.request;
         report (pname & " has entered the fitting room");
         Hold (Normal (3, 1, u));
         fittingroom1.leave;
         report (pname & " has left the fitting room");
      End;
   End;

   Integer u;
   Ref (FittingRoom) fittingRoom1;

   fittingRoom1:- New FittingRoom;
   Activate New Person ("Sam");
   Activate New Person ("Sally");
   Activate New Person ("Andy");
   Hold (100);
End;
```

multitasking

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

and ran on the IBM 360/165 mainframe.

System/370 model 165

https://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP3165.html

The concepts of Simula and object orientation became increasingly helpful as the size of the program increased. Unfortunately, the implementation of Simula did not scale the same way.

Eventually, he was foreced to rewrite the simulator in BCPL and run it on the experimental CAP computer.

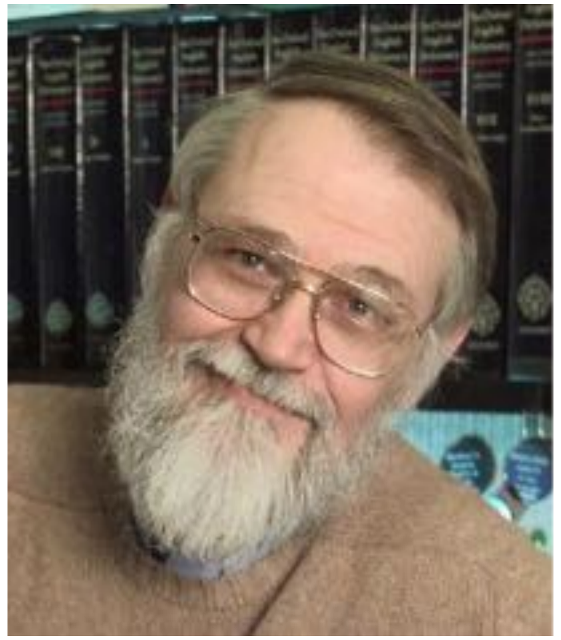
“The experience of coding and debugging the simulator in BCPL was horrible.”

“Upon leaving Cambridge, I swore never again to attack a problem with tools as unsuitable as those I had suffered while designing and implementing the simulator.”

After finishing his PhD Thesis in Cambridge, Bjarne was hired by Bell Labs in April 1979

At Bell Labs, Bjarne started to analyze if the UNIX kernel could be distributed over a network of computers connected by a local area network. Proper tools was needed….

Bjarne started to write a preprocessor to C that added Simula like classes to C.



"I learned C properly from people like Stu Feldman, Steve Johnson, Brian Kernighan, and Dennis Ritchie."

And then Bjarne started to develop "C with Classes". The main motivation was to create better support for modularity and concurrency.

“The first demand from development management was that of 100% compatibility with C.”

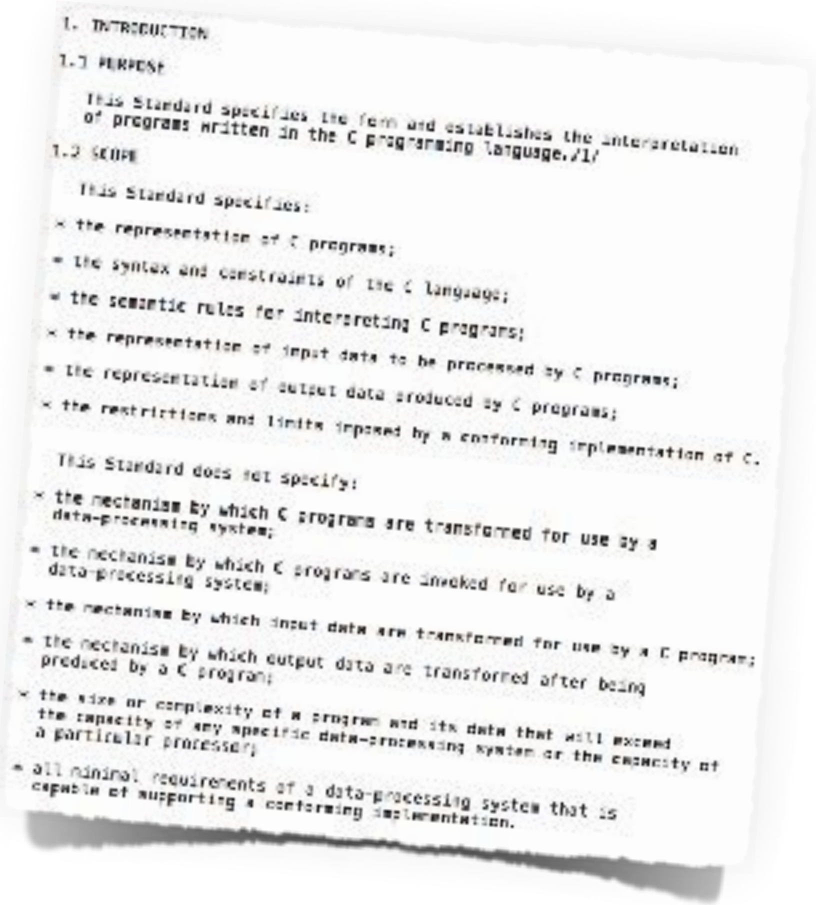
But without a standard, that requirement did not make much sense: compatible with what implementation of C?

The success of C++ added to the motivation for a C standard

C++ was the inspiration for the function prototypes and several other mechanisms stronger type support.

Indeed, while an unusual perspective, it is fair to some extend to view ANSI C as a strict subset of C++ at the time.

Fun fact: All the examples in K&R, 2ed, was compiled with CFront 2.0



1. INTRODUCTION

1.1 PURPOSE

This Standard specifies the form and establishes the interpretation of programs written in the C programming language./1/

1.2 SCOPE

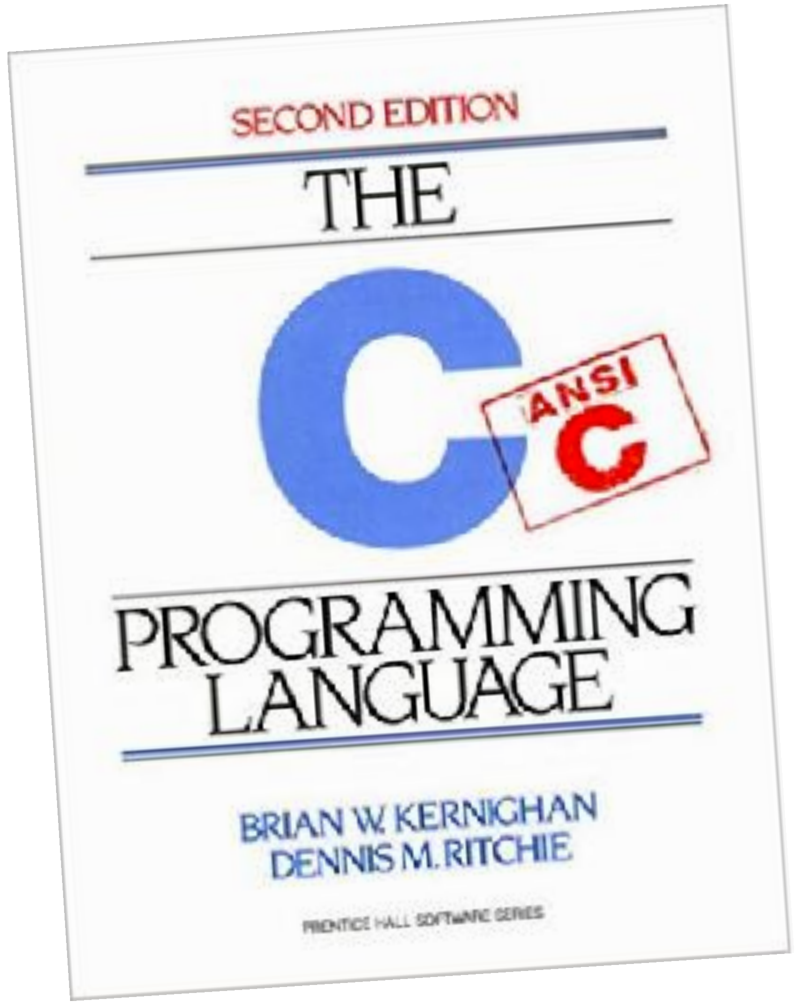
This Standard specifies:

- the representation of C programs;
- the syntax and constraints of the C language;
- the semantic rules for interpreting C programs;
- the representation of input data to be processed by C programs;
- the representation of output data produced by C programs;
- the restrictions and limits imposed by a conforming implementation of C.

This Standard does not specify:

- the mechanism by which C programs are transformed for use by a data-processing system;
- the mechanism by which C programs are invoked for use by a data-processing system;
- the mechanism by which input data are transformed for use by a C program;
- the mechanism by which output data are transformed after being produced by a C program;
- the size or complexity of a program and its data that will exceed the capacity of any specific data-processing system or the capacity of a particular processor;
- all minimal requirements of a data-processing system that is capable of supporting a conforming implementation.

ANSI/ISO C (C89/C90)



K&R, ed 2

C99

C99 added a lot of stuff to C89, perhaps too much. Especially a lot of features for scientific computing was added, but also a few things that made life easier for programmers.

INTERNATIONAL
STANDARD

ISO/IEC
9899

Second edition
1999-12-01

Programming languages — C

Langages de programmation — C

Adopted by INCITS (InterNational Committee for Information Technology Standards) as an American National Standard.

Date of ANSI Approval: 5/22/00

Published by American National Standards Institute,
25 West 43rd Street, New York, New York 10036

Copyright 2002 by Information Technology Industry Council (ITI).
All rights reserved.

These materials are subject to copyright claims of International Standardization Organization (ISO), International Electrotechnical Commission (IEC), American National Standards Institute (ANSI), and Information Technology Industry Council (ITI). Not for resale. No part of this publication may be reproduced in any form, including an electronic retrieval system, without the prior written permission of ITI. All requests pertaining to this standard should be submitted to ITI, 1250 Eye Street NW, Washington, DC 20005.

Printed in the United States of America

Reference number
ISO/IEC 9899:1999(E)

© ISO/IEC 1999

Licensed to [illegible]. ANSI order X [illegible]. Downloaded [illegible] PM. Single user license only. Copying and networking prohibited.

```c
// C99 example, ISO/IEC 9899:1999

#include <stdio.h>

size_t mystrcpy(char *restrict s, const char *restrict t)
{
    size_t i;

    for (i = 0; (*s++ = *t++) != '\0'; i++)
        ;
    return i;
}

int main(void)
{
    char str1[16];
    char str2[] = "Hello, C99!";
    size_t len = mystrcpy(str1, str2);
    for (size_t i = 0; i < len; i++)
        putchar(str1[i]);
}
```

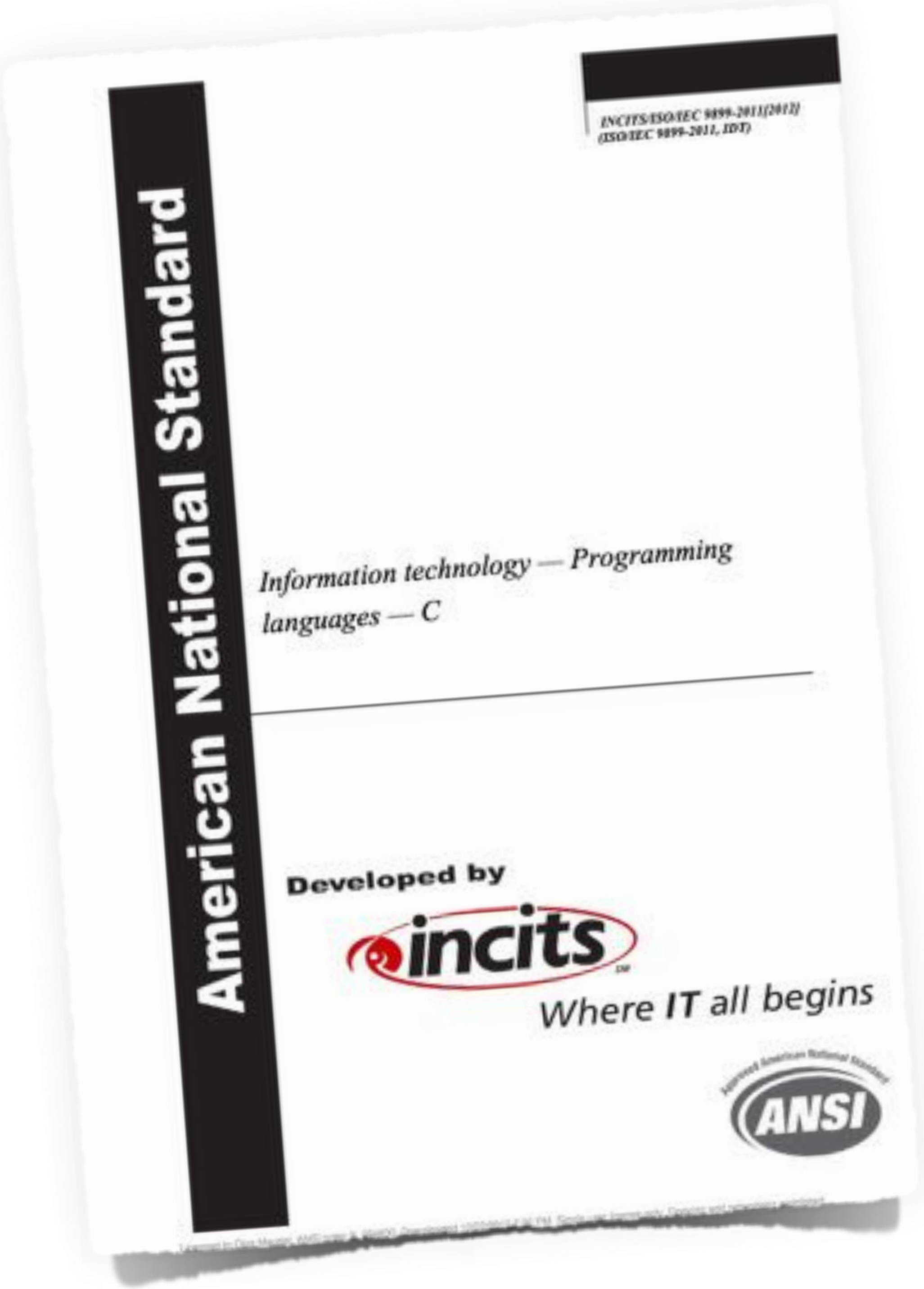
C11

The main focus:
- security, eg Anneks K (the bounds checking library, contributed by Microsoft)
- support for multicore systems (threads from WG14, memory model from WG21)

The most interesting features:

- Type-generic expressions using the _Generic keyword.
- Multi-threading support
- Improved Unicode support
- Removal of the gets() function
- Bounds-checking interfaces
- Anonymous structures and unions
- Static assertions
- Misc library improvements

Made a few C99 features optional.



INCITS/ISO/IEC 9899-2011[2012]
(ISO/IEC 9899-2011, IDT)

American National Standard

Information technology — Programming languages — C

Developed by

incits

Where IT all begins

WG14 meeting at Lysaker, April 2015

Cisco Systems in Norway

Next version of C - C2x?

- Currently working on defect reports
- There are some nasty/interesting differences between C11 and C++11
- IEEE 754 floating point standard updated in 2008
- CPLEX - C parallel language extentions (started after C11)

Private conversation with David Keaton, April 2015

The Spirit of C

- *Trust the programmer.*
- *Don't prevent the programmer from doing what needs to be done.*
- *Keep the language small and simple.*
- *Provide only one way to do an operation.*
- *Make it fast, even if it is not guaranteed to be portable.*

http://www.open-std.org/jtc1/sc22/wg14/www/C99RationaleV5.10.pdf

"The philosophy of BCPL is not one of the tyrant who thinks he knows best and lay down the law on what is and what is not allowed; rather, BCPL acts more as a servant offering his services to the best of his ability without complaint, even when confronted with apparent nonsense. The programmer is always assumed to know what he is doing and is not hemmed in by petty restrictions."

```
s/BCPL/C/g
```

(The BCPL book, 1979)

"The philosophy of C is not one of the tyrant who thinks he knows best and lay down the law on what is and what is not allowed; rather, C acts more as a servant offering his services to the best of his ability without complaint, even when confronted with apparent nonsense. The programmer is always assumed to know what he is doing and is not hemmed in by petty restrictions."

(a rewrite of perhaps the most important paragraph in the BCPL book, 1979)

!

?