History and Spirit of C and C++

Olve Maudal

To get a deep understanding of C and C++, it is useful to know the history of these wonderful programming languages. It is perhaps even more important to appreciate the driving forces, motivation and the spirit that has shaped these languages into what we have today.

In the first half of this talk we go back to the early days of programmable digital computers. We will take a brief look at really old machine code, assembler, Fortran, IAL, Algol 60 and CPL, before we discuss the motivations behind BCPL, B and then early C. We will also discuss influential hardware architectures represented by EDSAC, Atlas, PDP-7, PDP-11 and Interdata 8/32. From there we quickly move through the newer language versions such as K&R C, C89, C99 and C11.

In the second half we backtrack into the history again, now including Simula, Algol 68, Ada, ML, Clu into the equation. We will discuss the motivation for creating C++, and with live coding we will demonstrate by example how it has evolved from the rather primitive "C with Classes" into a supermodern and capable programming language as we now have with C++11/14 and soon with C++17.

A 90 minute session at ACCU 2015, April 23, Bristol, UK

Part I

History and spirit of C

- The short version
- Before C
- Early C and K&R
- ANSI C
- Modern C
- Q&A

Part II

History and spirit of C++

- Before C++
- Developing the initial versions of C++ (pre-1985)
- Development of C++ (after-1985)
- Evolution of C++ by examples

Part I

History and spirit of C

- The short version
- Before C
- Early C and K&R
- ANSI C
- Modern C
- Q&A

(~90 minutes)

Part II

History and spirit of C++

- Before C++
- Developing the initial versions of C++ (pre-1985)
- Development of C++ (after-1985)
- Evolution of C++ by examples

(a few minutes)

Part I

History and spirit of C

- The short version
- Before C
- Early C and K&R
- ANSI C
- Modern C
- Q&A

(~90 minutes)

Part II

History and spirit of C++

- Before C++
- Developing the initial versions of C++ (pre-1985)
- Development of C++ (after-1985)
- ~~Evolution of C++ by examples~~

(a few minutes)

C

History and Spirit of C

Olve Maudal

To get a deep understanding of C, it is useful to know the history of this wonderful programming language. It is perhaps even more important to appreciate the driving forces, motivation and the spirit that has shaped the language into what we have today.

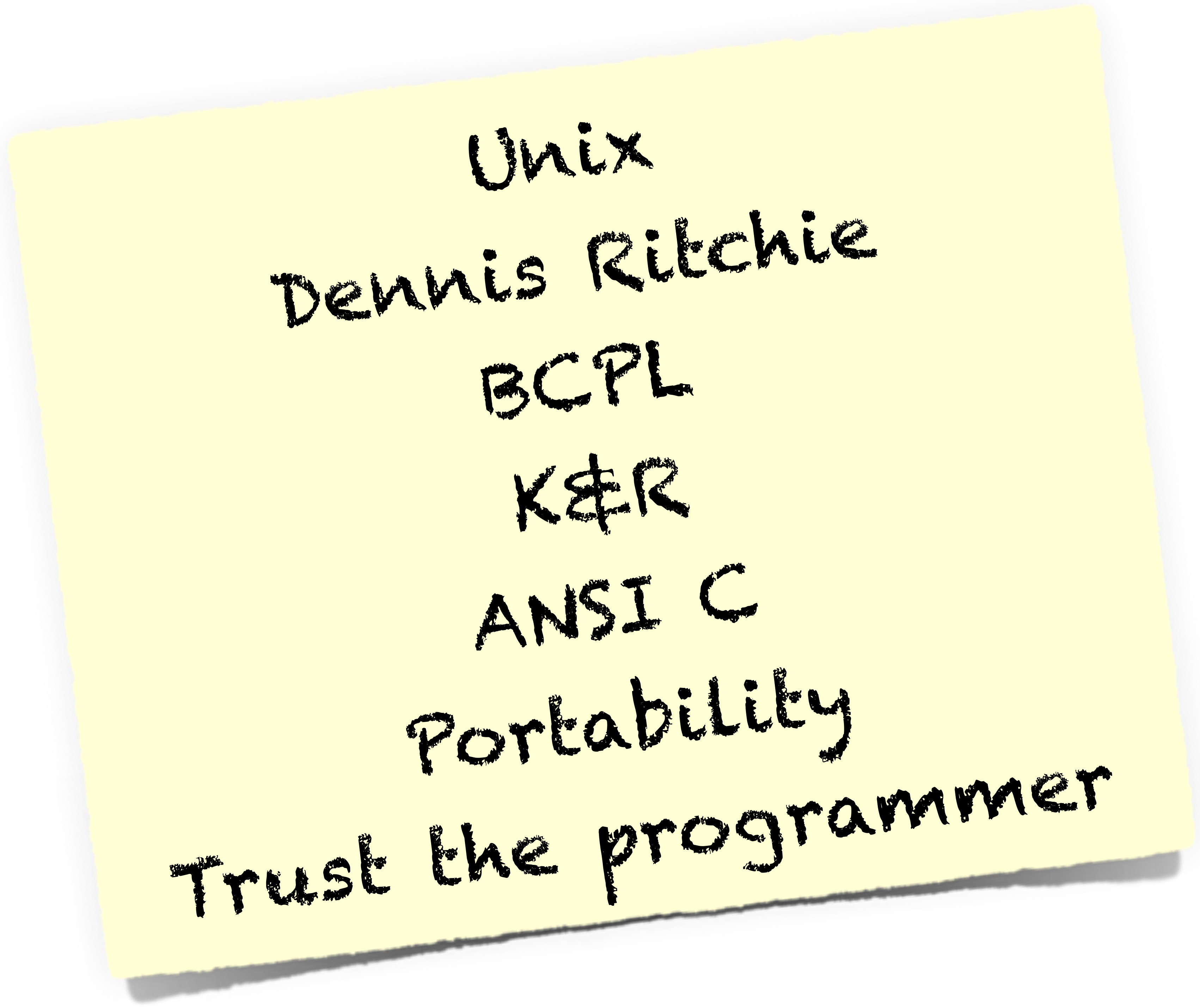
In this talk we go back to the early days of programmable digital computers. We will take a brief look at really old machine code, assembler, Fortran, IAL, Algol 60 and CPL, before we discuss the motivations behind BCPL, B and then early C. We will also discuss influential hardware architectures represented by EDSAC, Atlas, PDP-7, PDP-11 and Interdata 8/32. From there we quickly move through the newer language versions such as K&R C, C89, C99 and C11.

A ~90 minute session at ACCU 2015, April 23, Bristol, UK

This is based on research partly done together with Jon Jagger

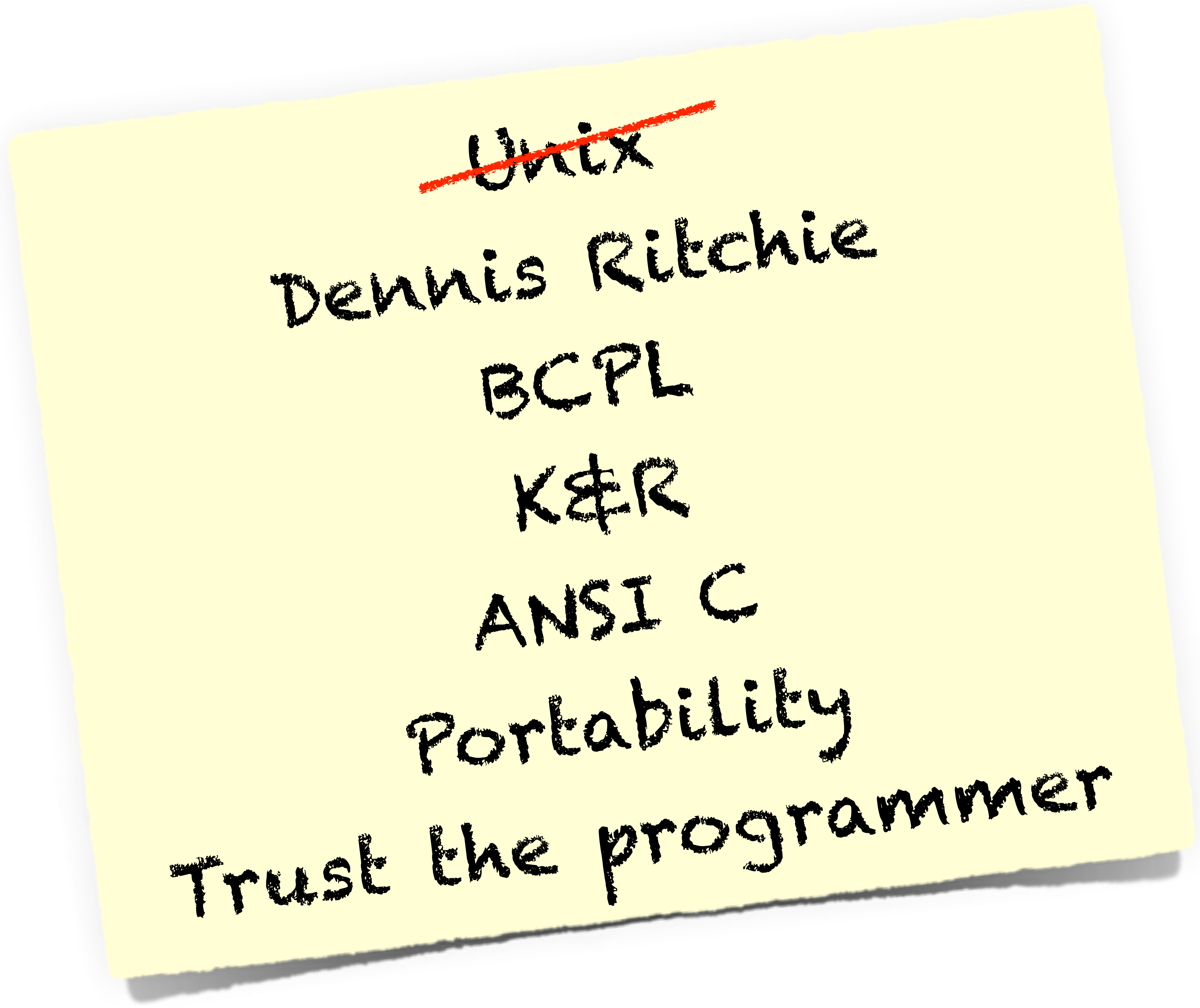
Let’s play bingo! Write down 7 words/names/concepts/whatever that
you expect/hope to be discussed/mentioned in a talk titled “History and Spirit of C”.

Let's play bingo! Write down 7 words/names/concepts/whatever that you expect/hope to be discussed/mentioned in a talk titled "History and Spirit of C".



Unix
Dennis Ritchie
BCPL
K&R
ANSI C
Portability
Trust the programmer

Let's play bingo! Write down 7 words/names/concepts/whatever that you expect/hope to be discussed/mentioned in a talk titled "History and Spirit of C".



~~Unix~~

Dennis Ritchie

BCPL

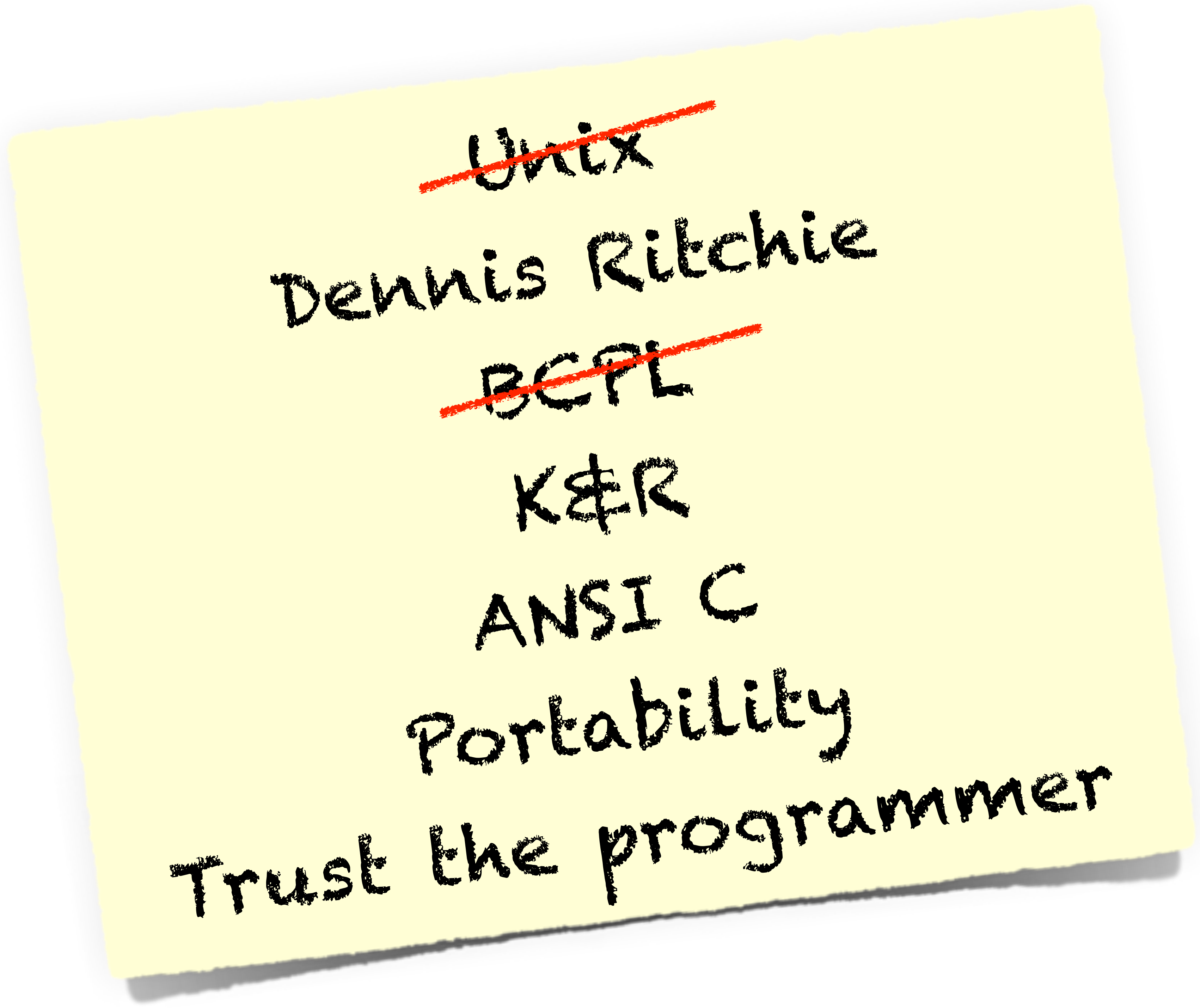
K&R

ANSI C

Portability

Trust the programmer

Let's play bingo! Write down 7 words/names/concepts/whatever that you expect/hope to be discussed/mentioned in a talk titled "History and Spirit of C".



~~Unix~~

Dennis Ritchie

~~BCPL~~

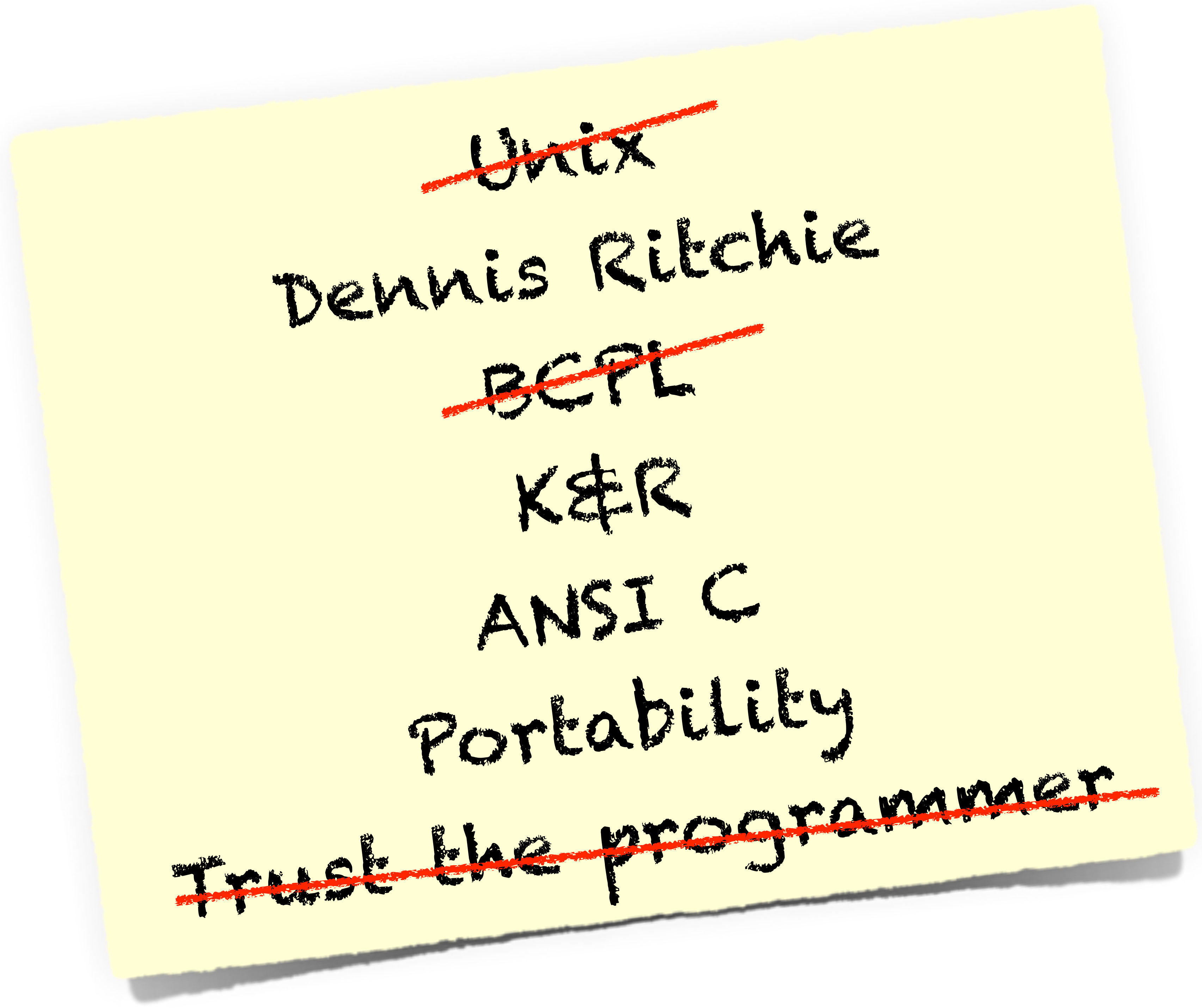
K&R

ANSI C

Portability

Trust the programmer

Let’s play bingo! Write down 7 words/names/concepts/whatever that you expect/hope to be discussed/mentioned in a talk titled “History and Spirit of C”.



~~Unix~~

Dennis Ritchie

~~BCPL~~

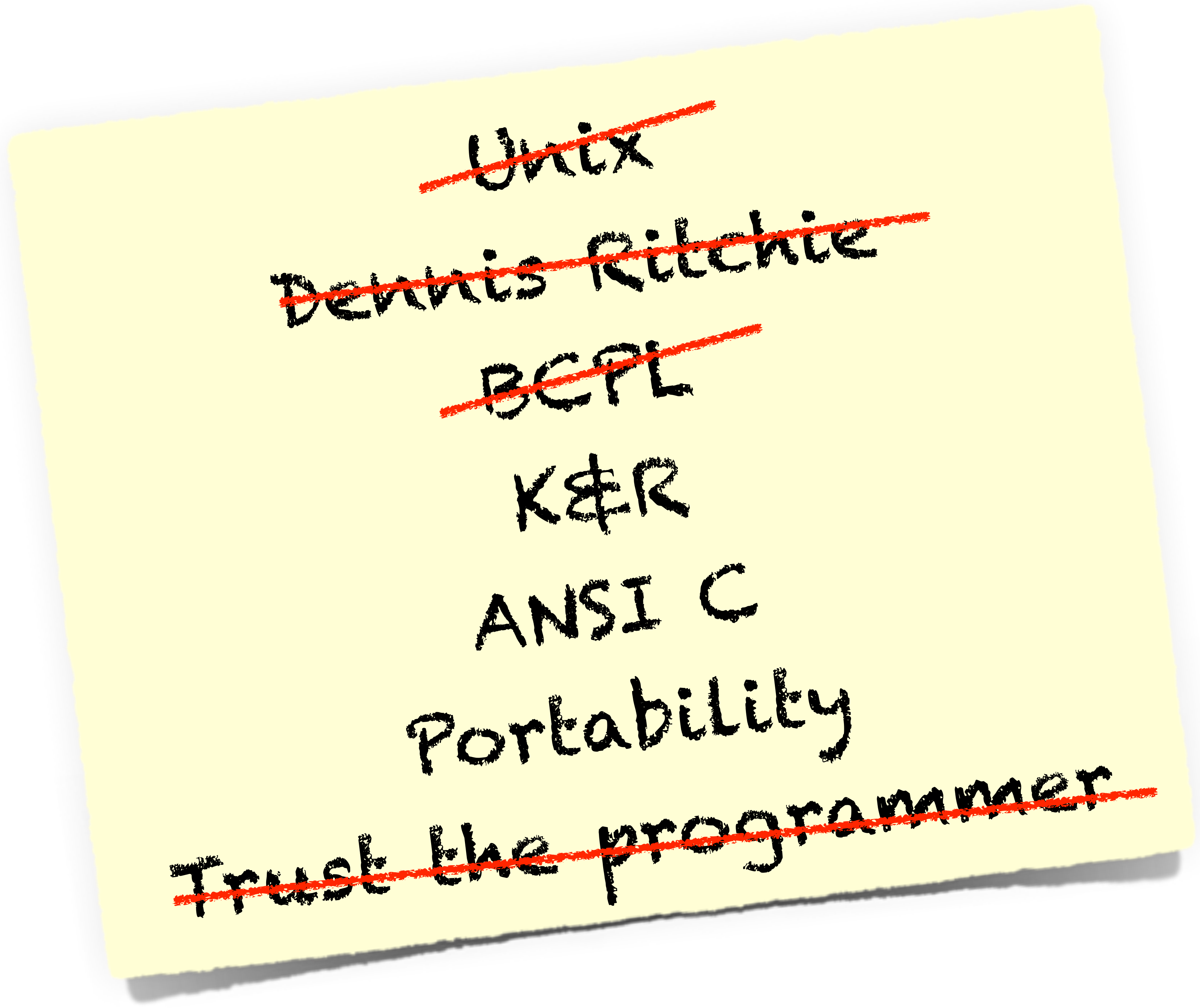
K&R

ANSI C

Portability

~~Trust the programmer~~

Let’s play bingo! Write down 7 words/names/concepts/whatever that you expect/hope to be discussed/mentioned in a talk titled “History and Spirit of C”.



~~Unix~~

~~Dennis Ritchie~~

~~BCPL~~

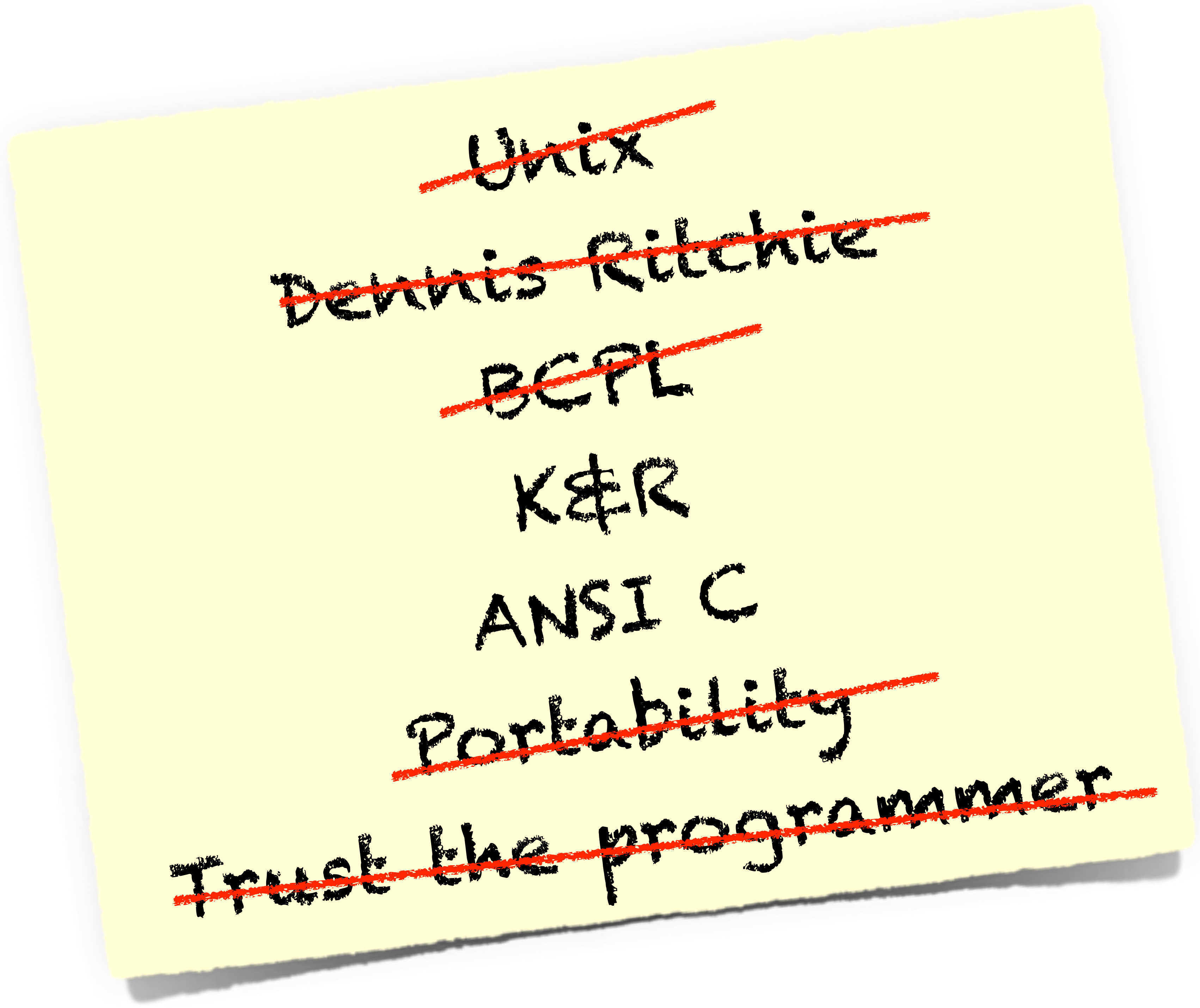
K&R

ANSI C

Portability

~~Trust the programmer~~

Let’s play bingo! Write down 7 words/names/concepts/whatever that you expect/hope to be discussed/mentioned in a talk titled “History and Spirit of C”.



~~Unix~~

~~Dennis Ritchie~~

~~BCPL~~

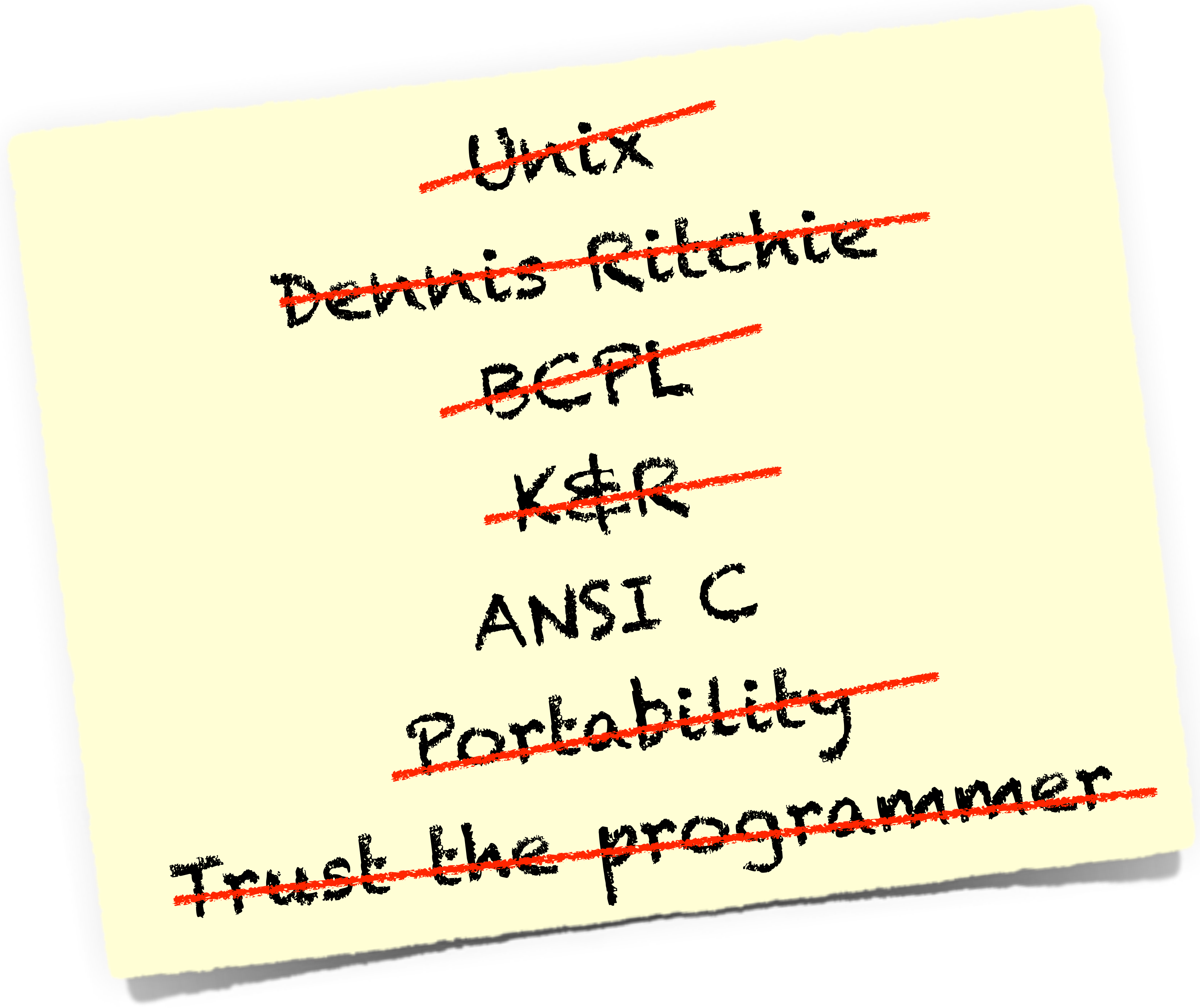
K&R

ANSI C

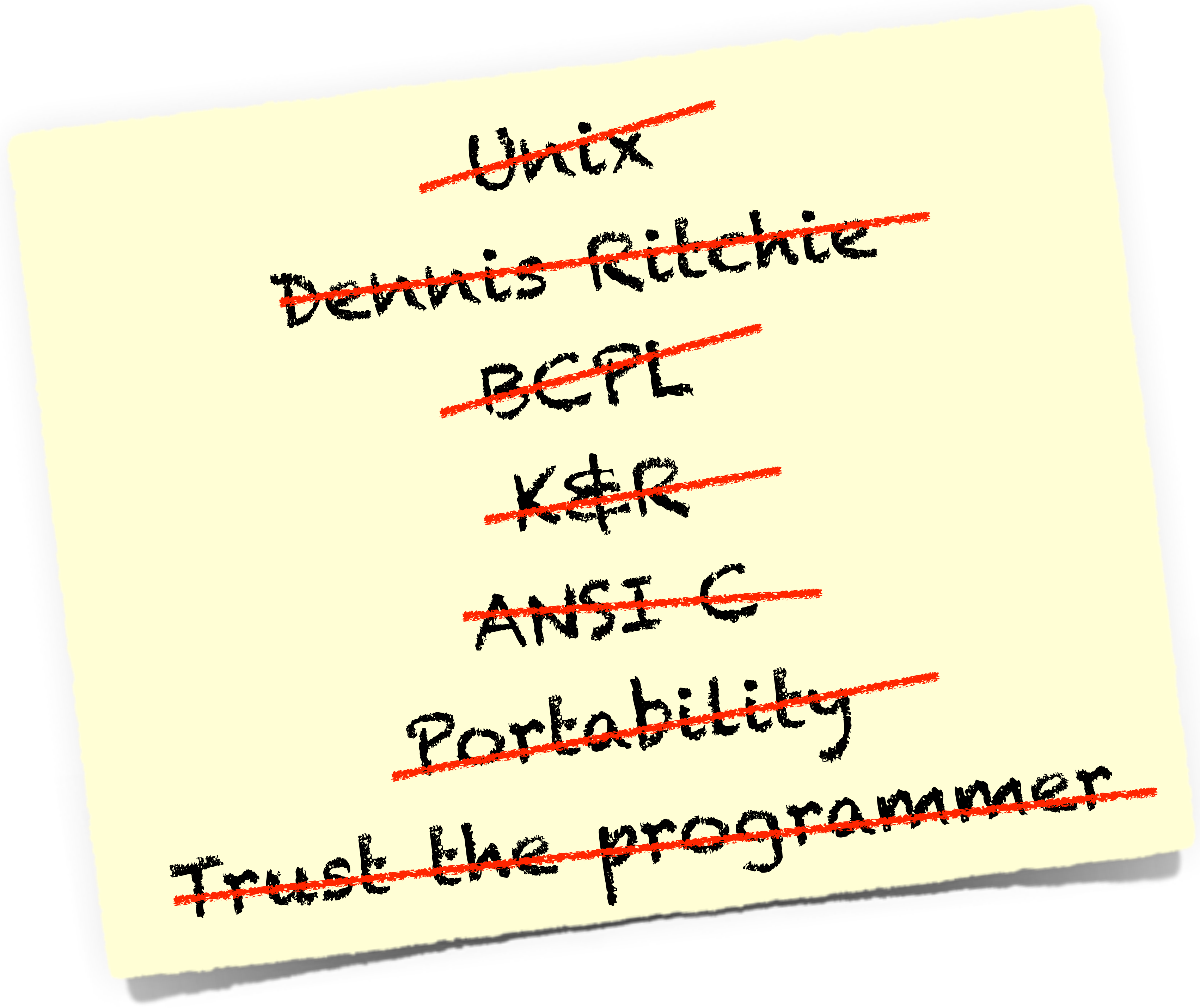
~~Portability~~

~~Trust the programmer~~

Let's play bingo! Write down 7 words/names/concepts/whatever that you expect/hope to be discussed/mentioned in a talk titled "History and Spirit of C".


~~Unix~~
~~Dennis Ritchie~~
~~BCPL~~
~~K&R~~
ANSI C
~~Portability~~
~~Trust the programmer~~

Let’s play bingo! Write down 7 words/names/concepts/whatever that you expect/hope to be discussed/mentioned in a talk titled “History and Spirit of C”.



~~Unix~~

~~Dennis Ritchie~~

~~BCPL~~

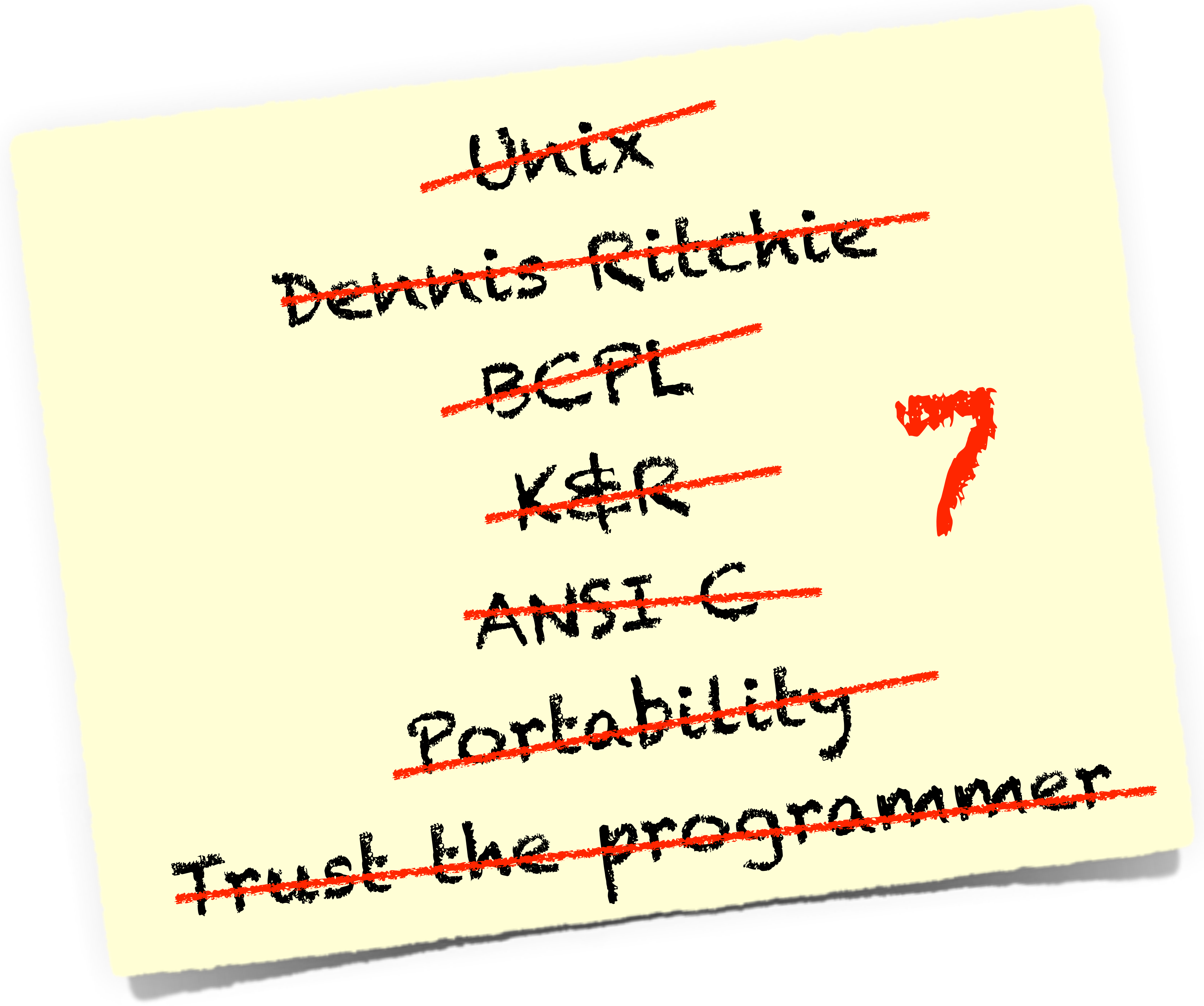
~~K&R~~

~~ANSI C~~

~~Portability~~

~~Trust the programmer~~

Let's play bingo! Write down 7 words/names/concepts/whatever that you expect/hope to be discussed/mentioned in a talk titled "History and Spirit of C".


~~Unix~~

~~Dennis Ritchie~~

~~BCPL~~

~~K&R~~

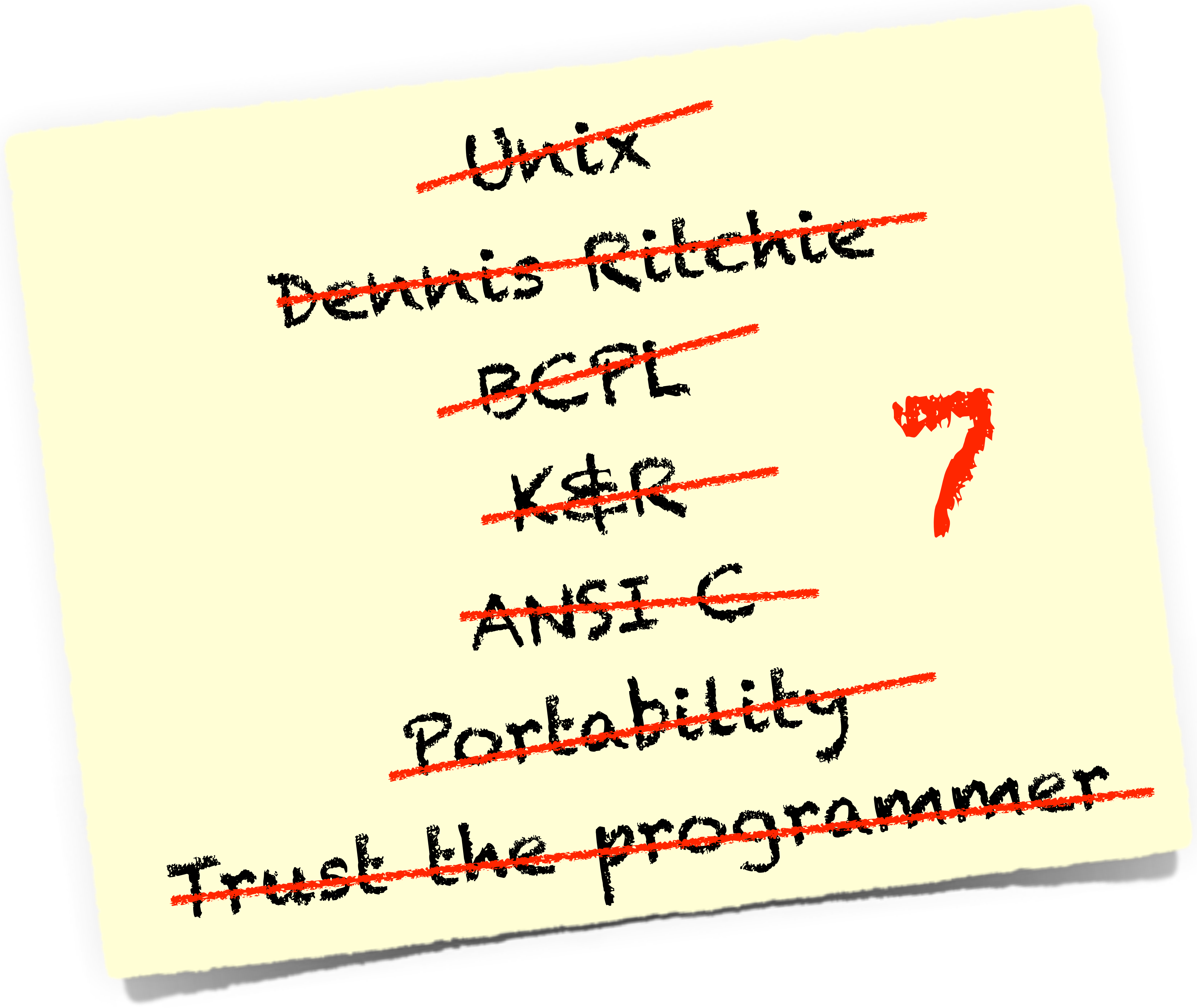
~~ANSI C~~

~~Portability~~

~~Trust the programmer~~

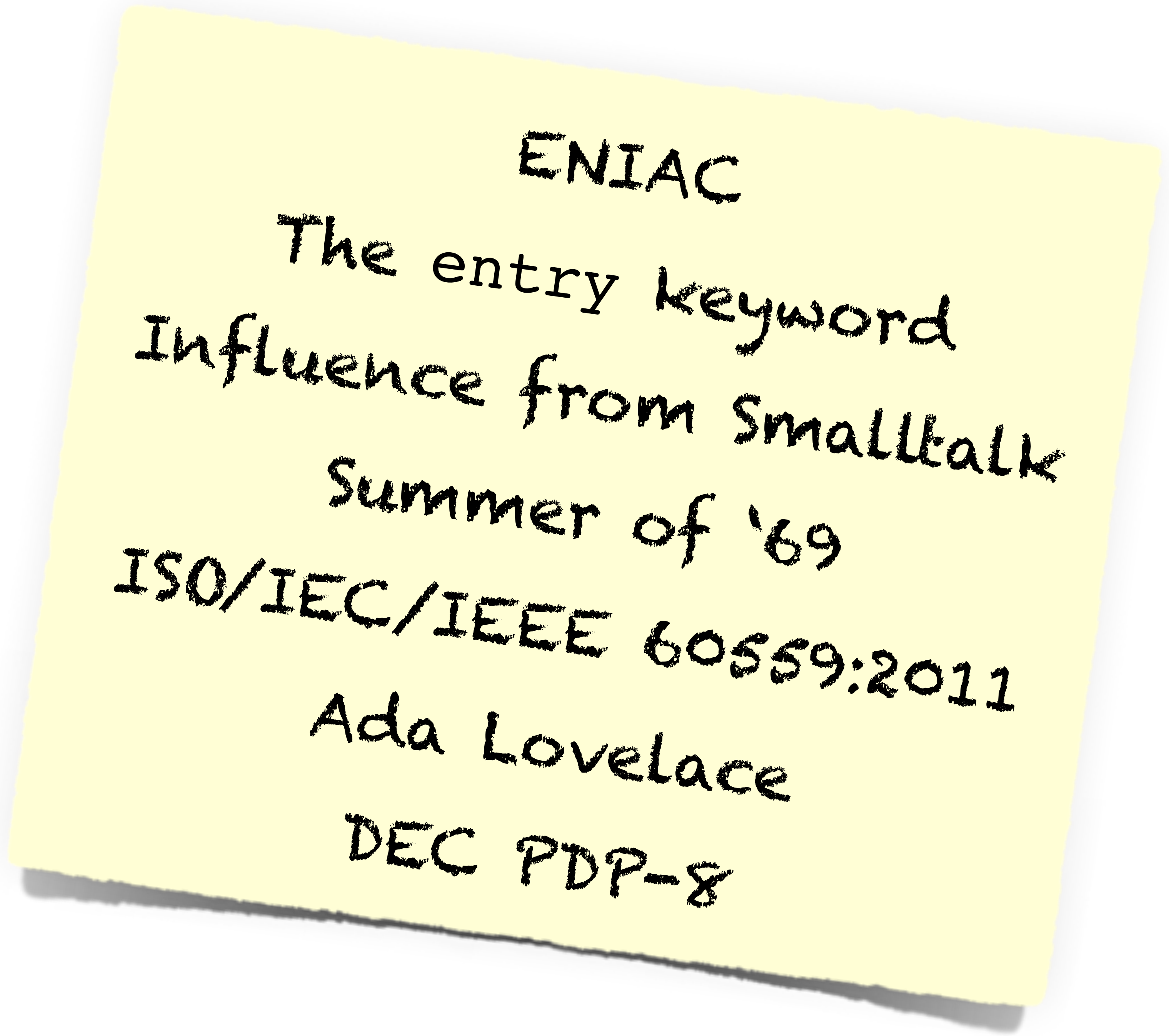
7

Let’s play bingo! Write down 7 words/names/concepts/whatever that you expect/hope to be discussed/mentioned in a talk titled “History and Spirit of C”.



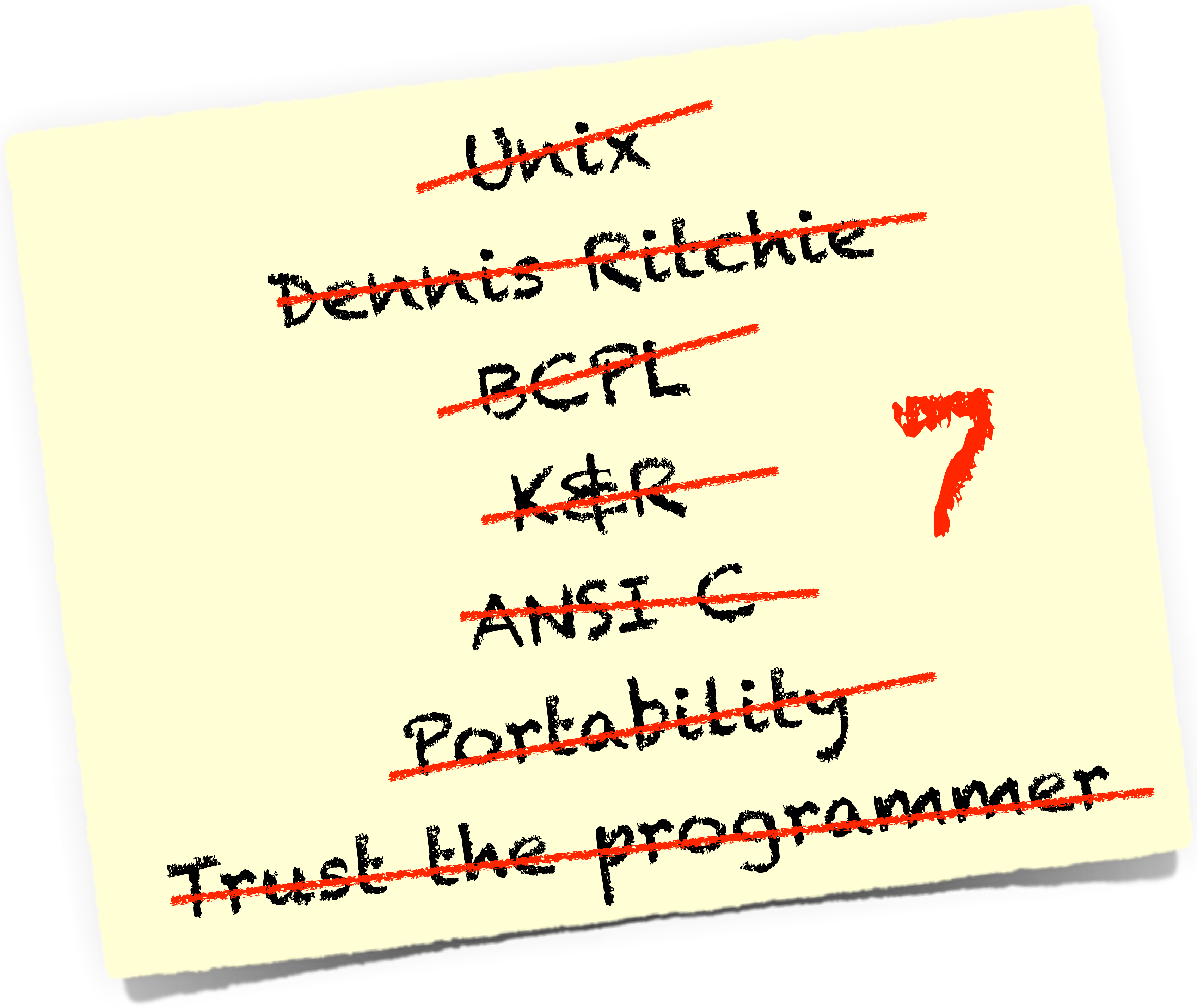
~~Unix~~
~~Dennis Ritchie~~
~~BCPL~~
~~K&R~~
~~ANSI C~~
~~Portability~~
~~Trust the programmer~~

7



ENIAC
The entry keyword
Influence from Smalltalk
Summer of '69
ISO/IEC/IEEE 60559:2011
Ada Lovelace
DEC PDP-8

Let's play bingo! Write down 7 words/names/concepts/whatever that you expect/hope to be discussed/mentioned in a talk titled "History and Spirit of C".



~~Unix~~

~~Dennis Ritchie~~

~~BCPL~~

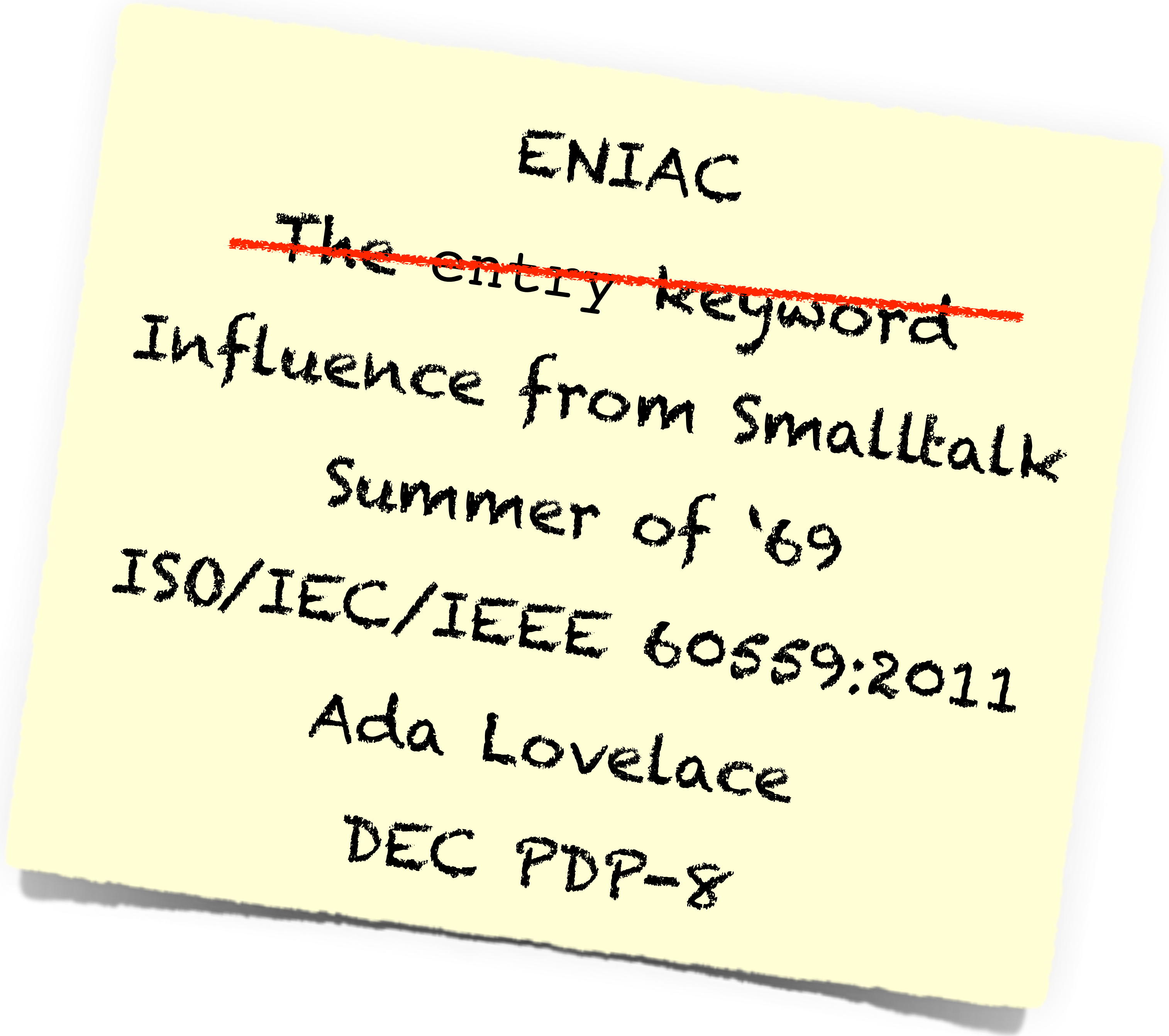
~~K&R~~

~~ANSI C~~

~~Portability~~

~~Trust the programmer~~

7



ENIAC

~~The entry keyword~~

Influence from Smalltalk

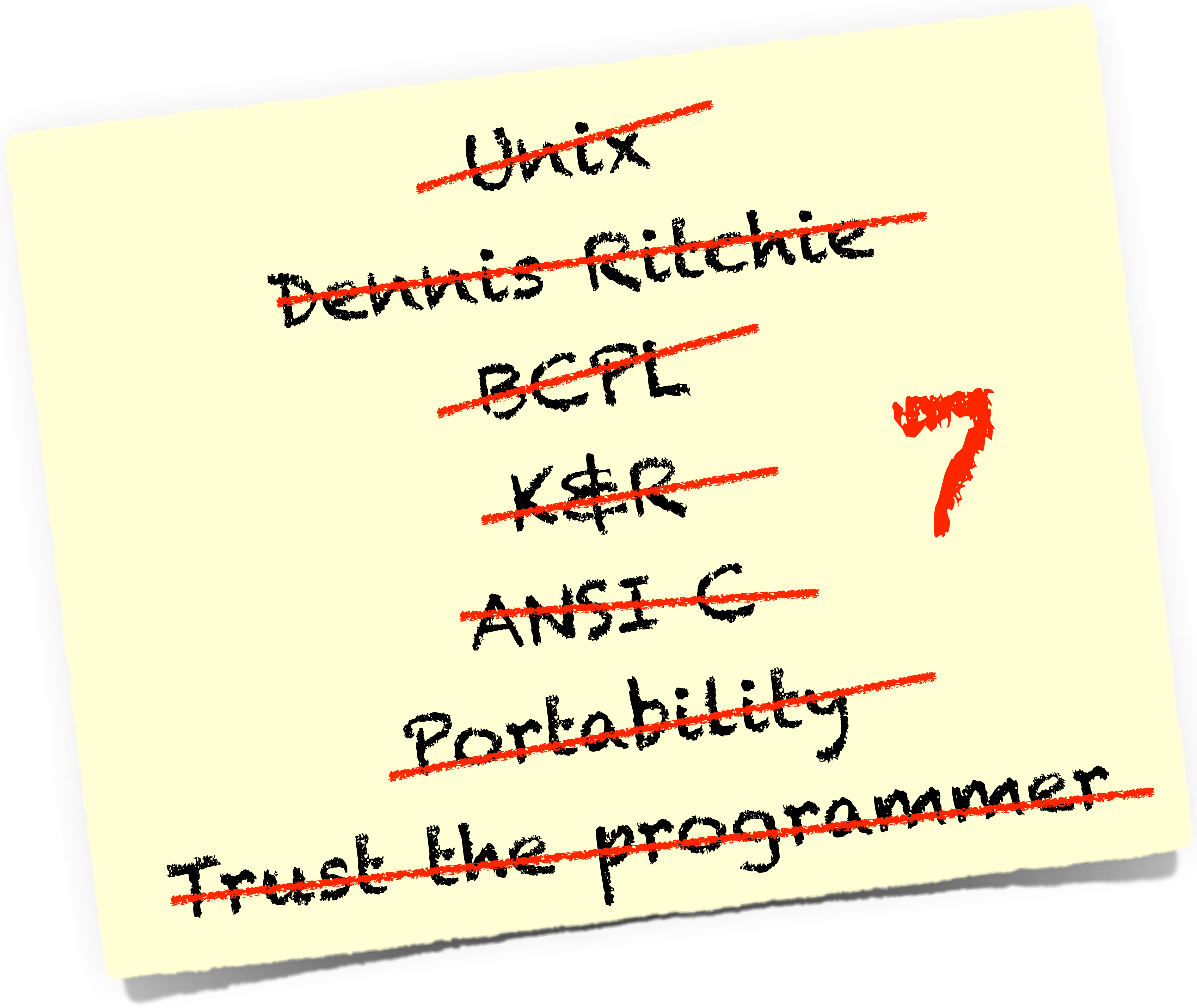
Summer of '69

ISO/IEC/IEEE 60559:2011

Ada Lovelace

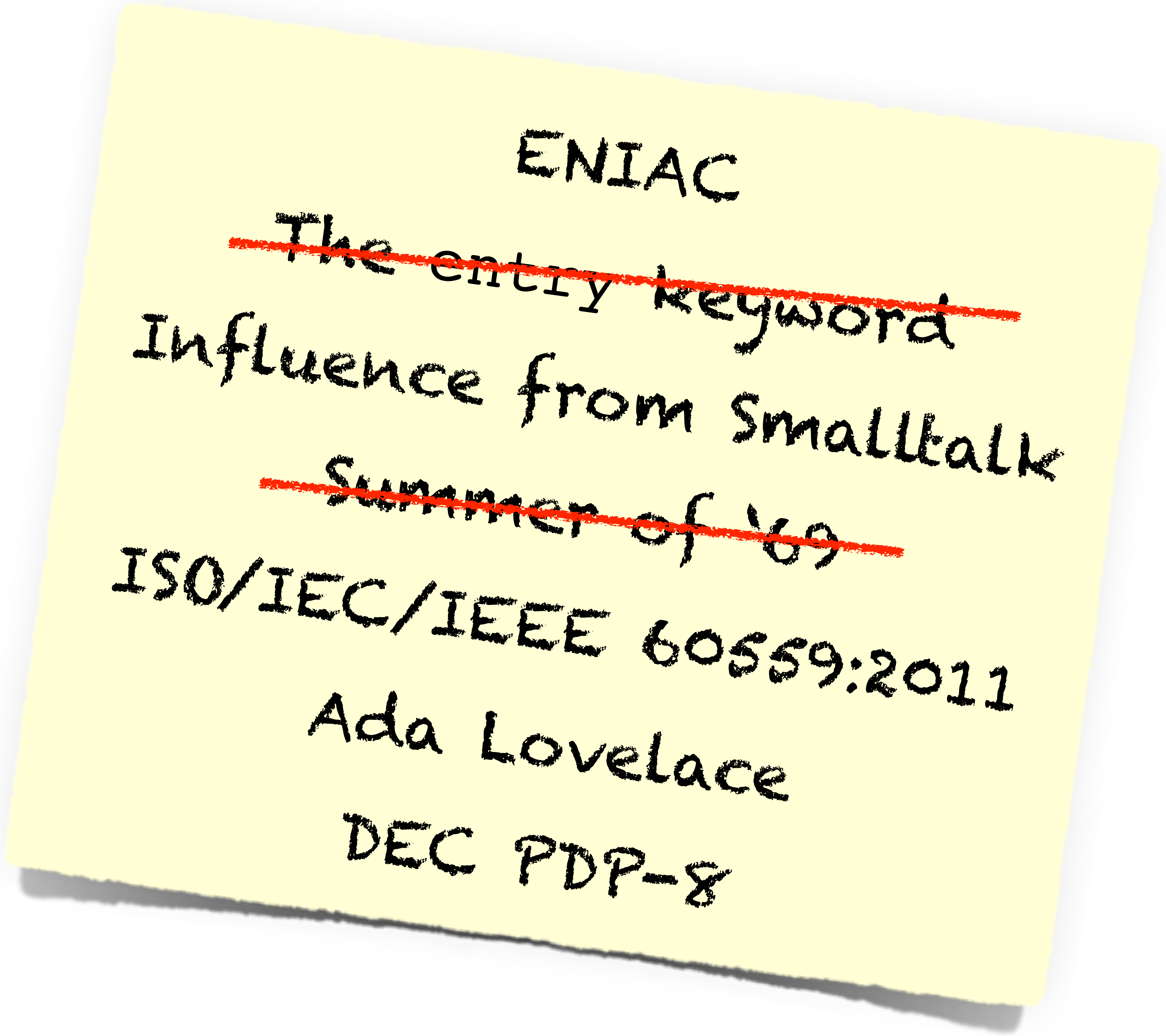
DEC PDP-8

Let's play bingo! Write down 7 words/names/concepts/whatever that you expect/hope to be discussed/mentioned in a talk titled "History and Spirit of C".



~~Unix~~
~~Dennis Ritchie~~
~~BCPL~~
~~K&R~~
~~ANSI C~~
~~Portability~~
~~Trust the programmer~~

7



ENIAC
~~The entry keyword~~
Influence from Smalltalk
~~Summer of '69~~
ISO/IEC/IEEE 60559:2011
Ada Lovelace
DEC PDP-8

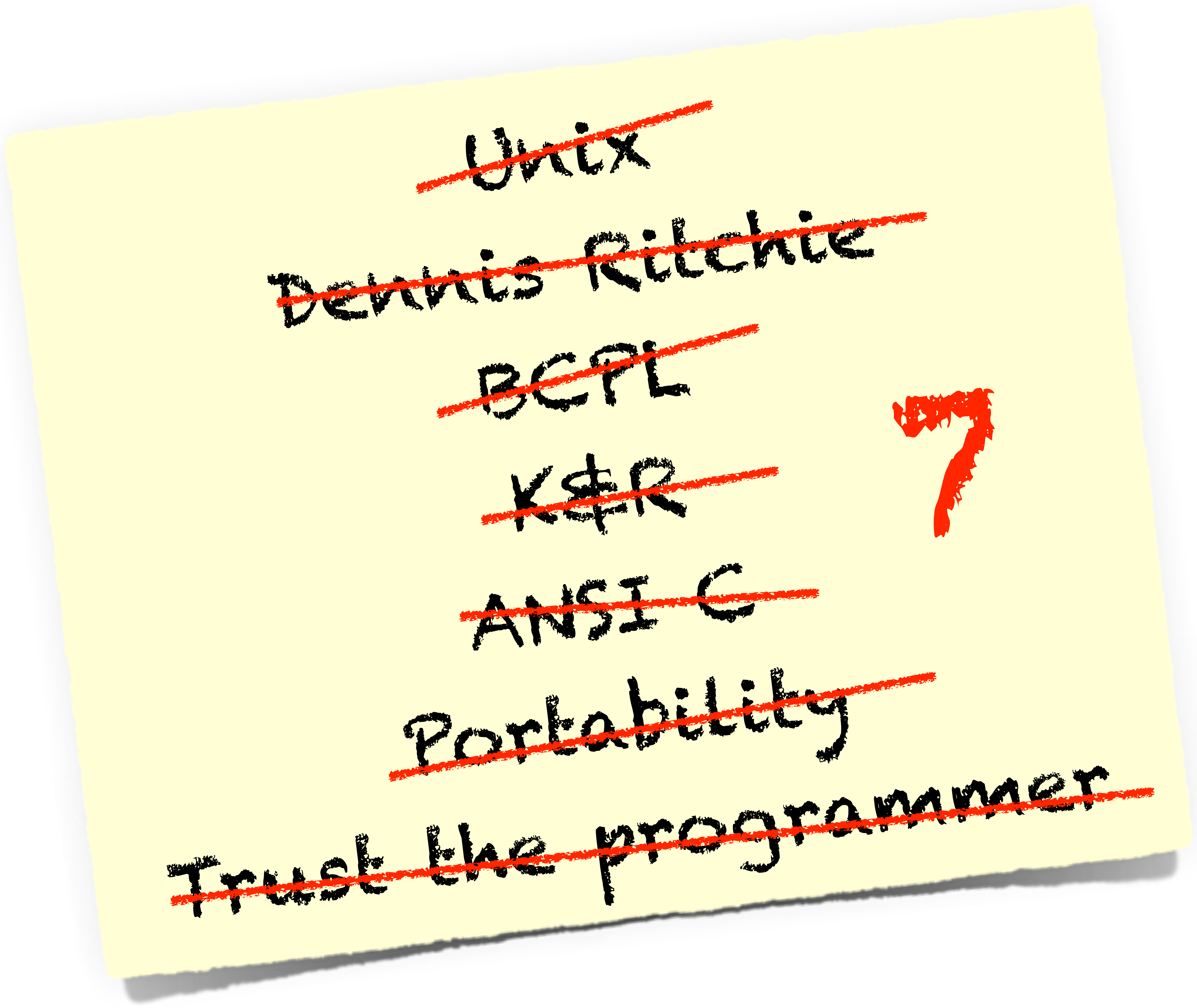
Let's play bingo! Write down 7 words/names/concepts/whatever that you expect/hope to be discussed/mentioned in a talk titled "History and Spirit of C".

~~Unix~~
~~Dennis Ritchie~~
~~BCPL~~
~~K&R~~
~~ANSI C~~
~~Portability~~
~~Trust the programmer~~

7

ENIAC
~~The entry keyword~~
Influence from Smalltalk
~~Summer of '69~~
~~ISO/IEC/IEEE 60559:2011~~
Ada Lovelace
DEC PDP-8

Let's play bingo! Write down 7 words/names/concepts/whatever that you expect/hope to be discussed/mentioned in a talk titled "History and Spirit of C".



~~Unix~~

~~Dennis Ritchie~~

~~BCPL~~

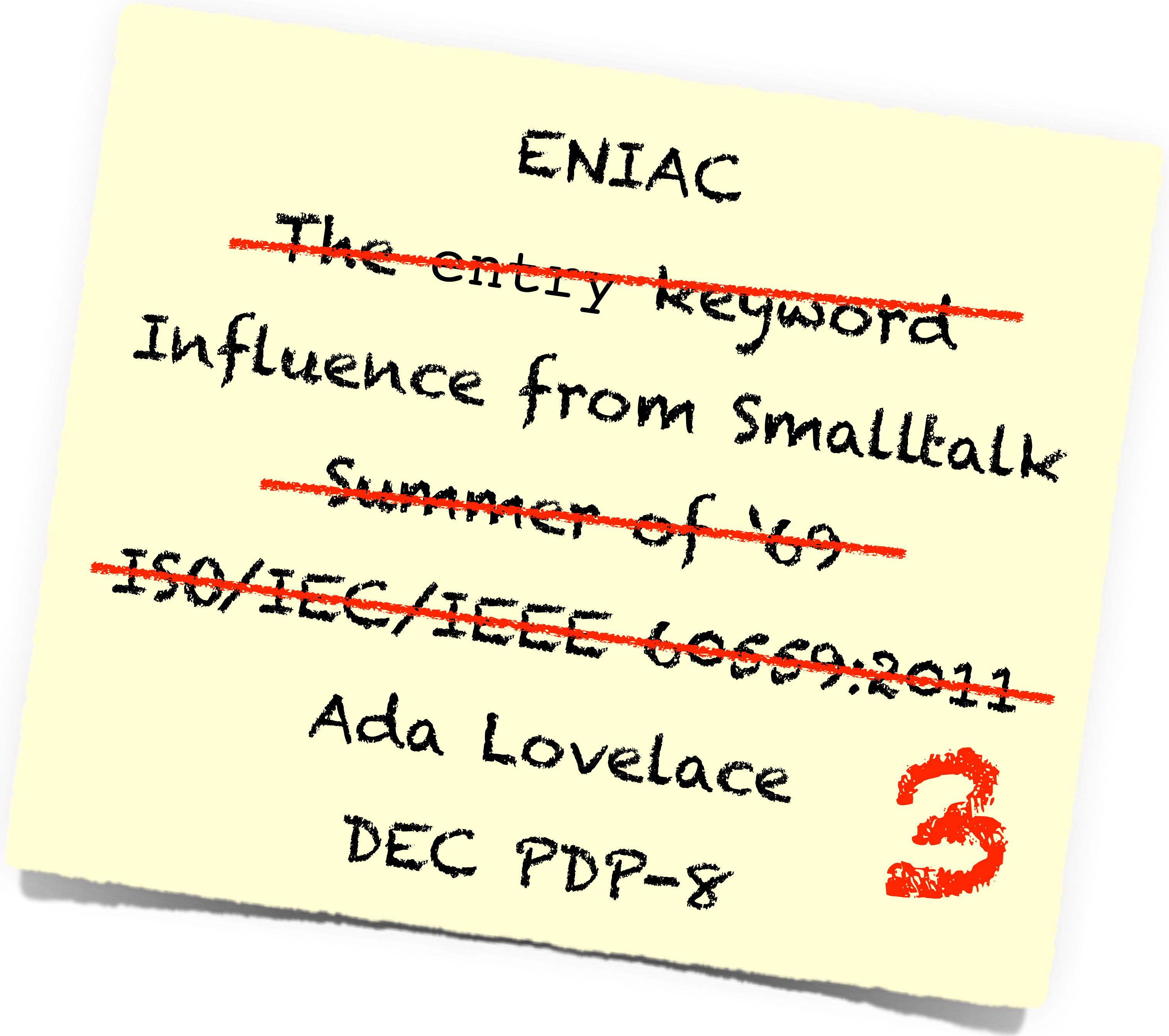
~~K&R~~

~~ANSI C~~

~~Portability~~

~~Trust the programmer~~

7



ENIAC

~~The entry keyword~~

Influence from Smalltalk

~~Summer of '69~~

~~ISO/IEC/IEEE 60559:2011~~

Ada Lovelace

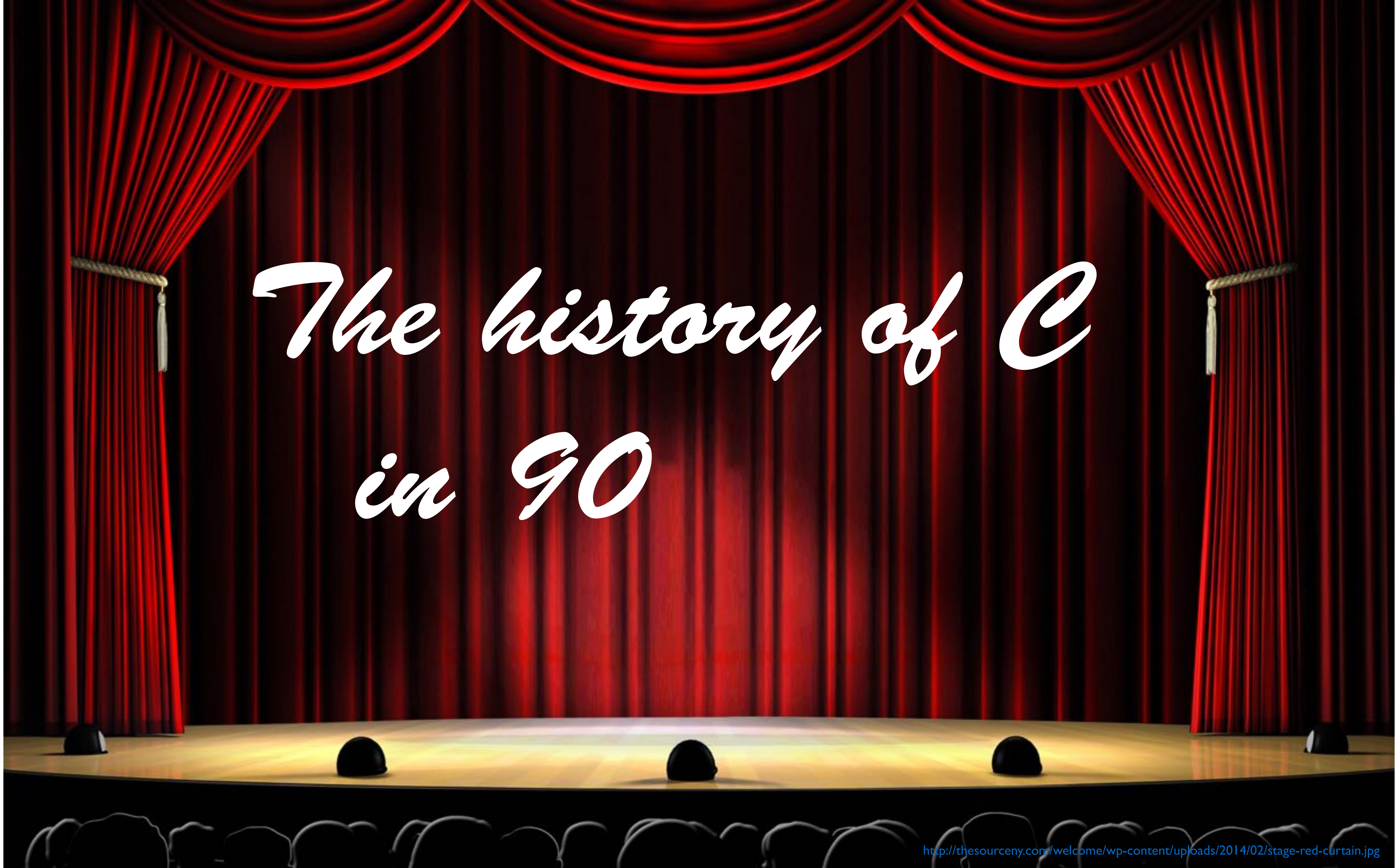
DEC PDP-8

3

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

The history of C

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



The history of C in 90

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

The history of C in 90 seconds

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

At Bell Labs.

NB
PE
NS
ME
MI
NH
N
MA
IN
OH
PA
WV
KY
VA
TN
NC
SC
AL
GA
FL

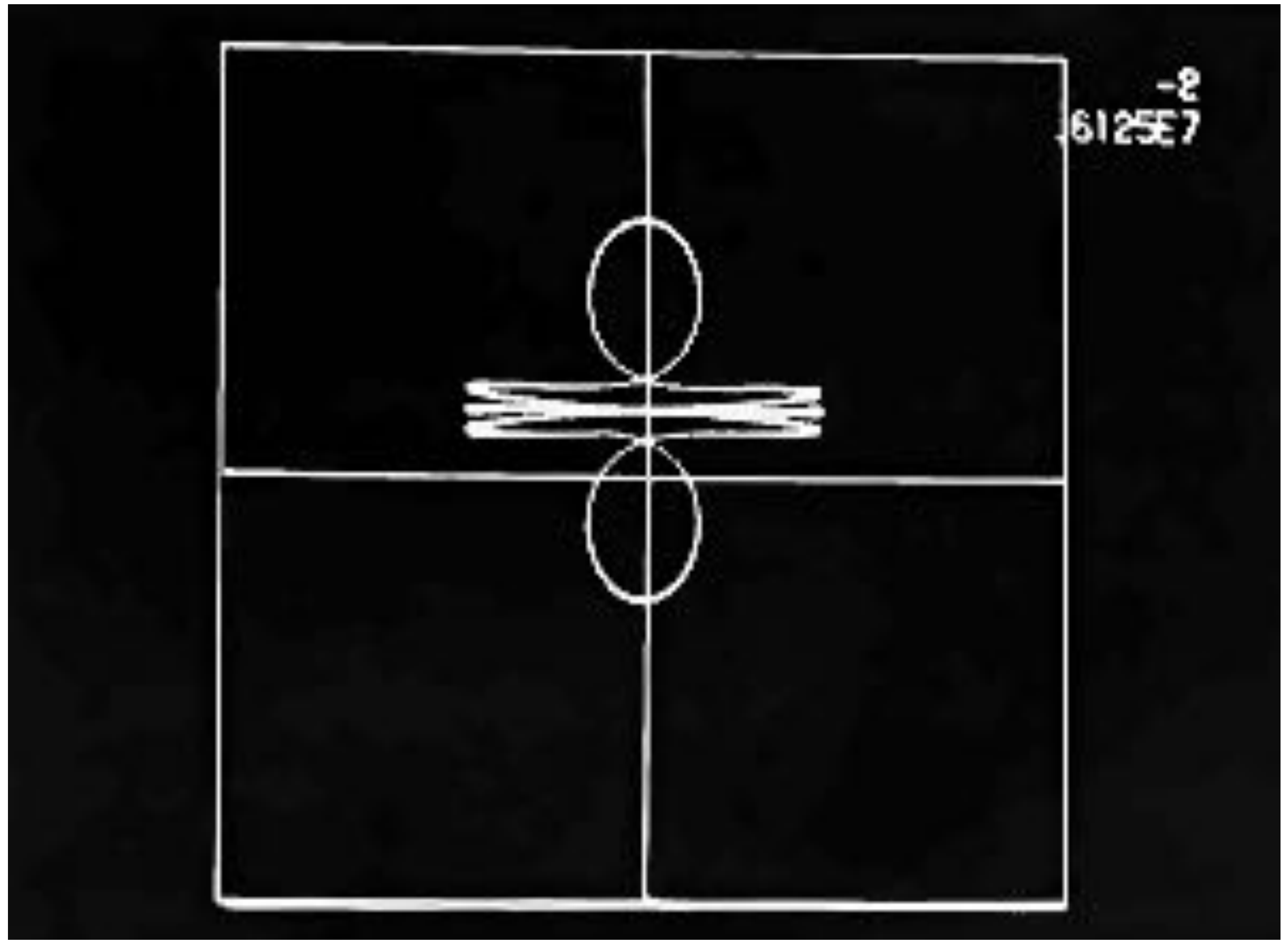
http://www3.nd.edu/~atrozzol/BellLabs1959.jpg

Back in 1969.

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

Ken Thompson wanted to play.

Ken Thompson wanted to play.

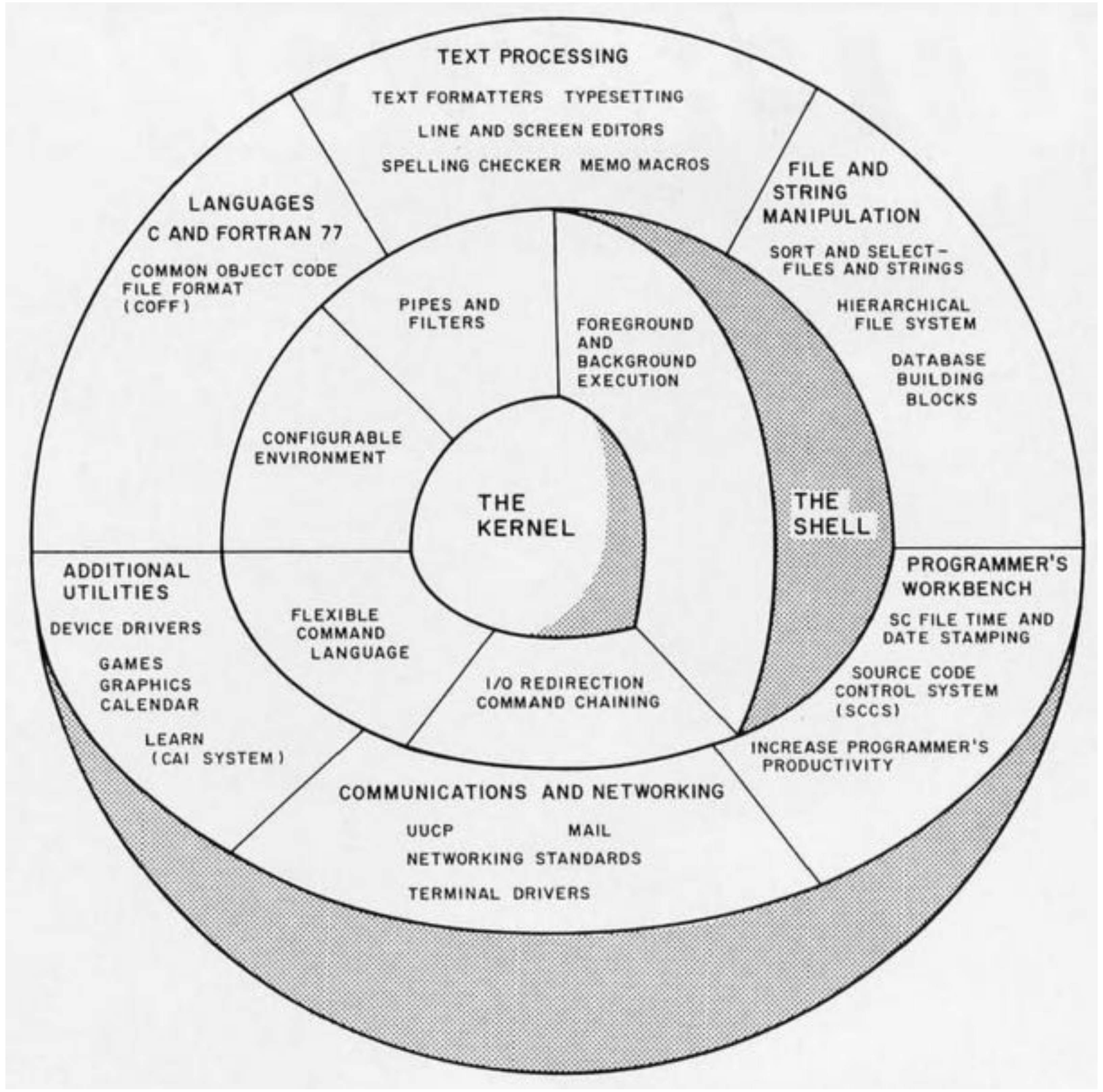


http://upload.wikimedia.org/wikipedia/commons/3/36/Ken_n_dennis.jpg

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

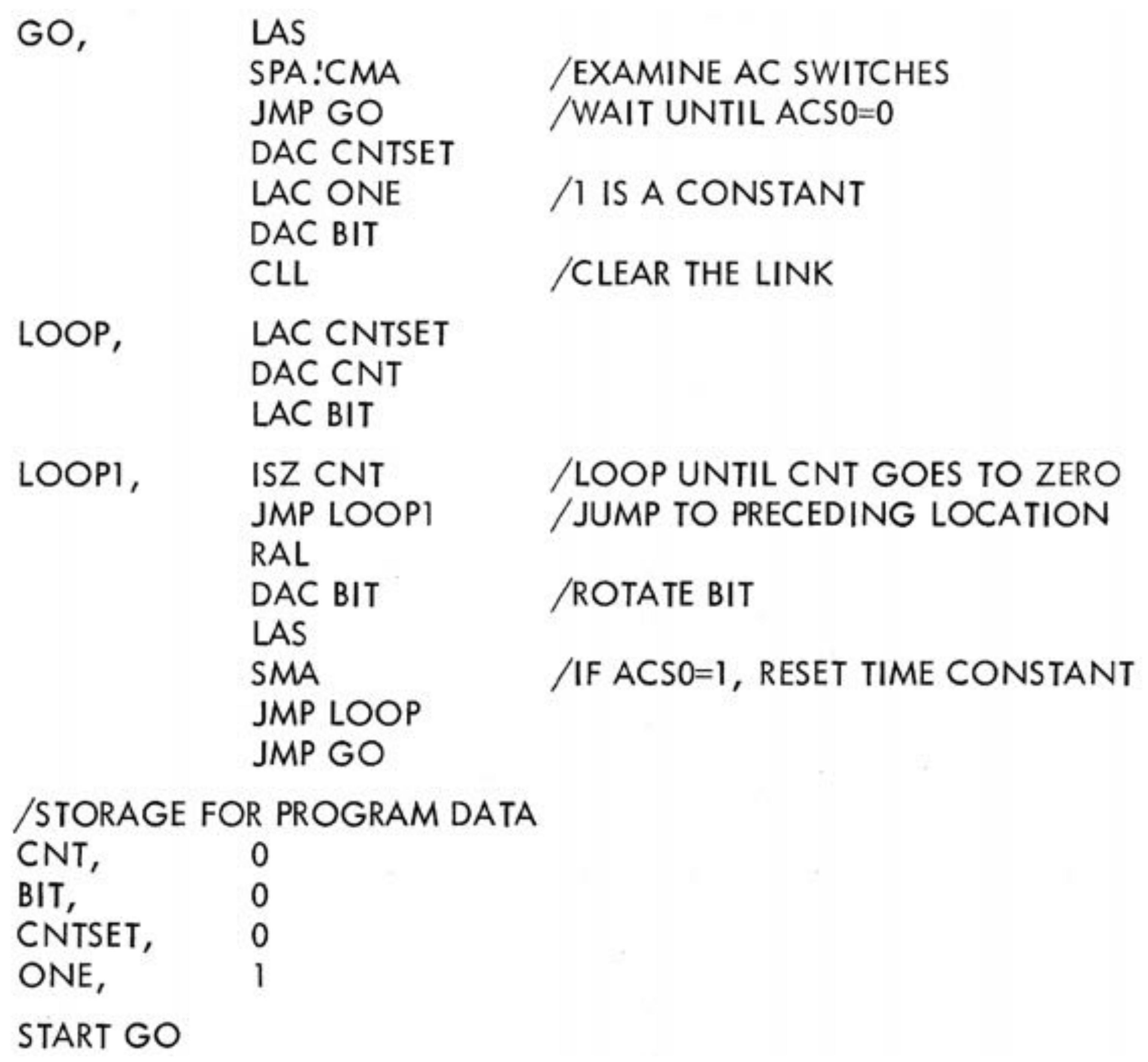
https://archive.org/stream/byte-magazine-1983-08/1983_08_BYTE_08-08_The_C_Language#page/n190/mode/1up

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

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

While porting Unix to a PDP-11

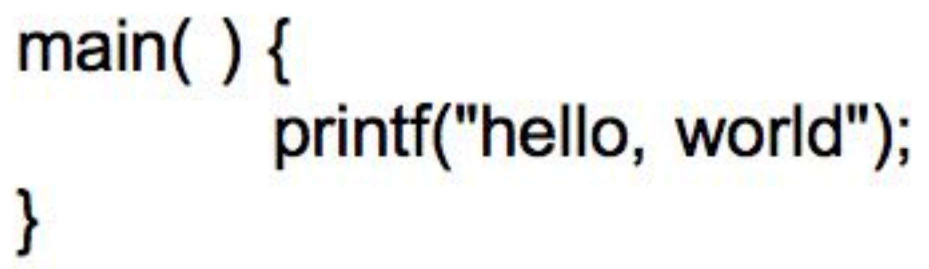
Ken

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

While porting Unix to a PDP-11

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

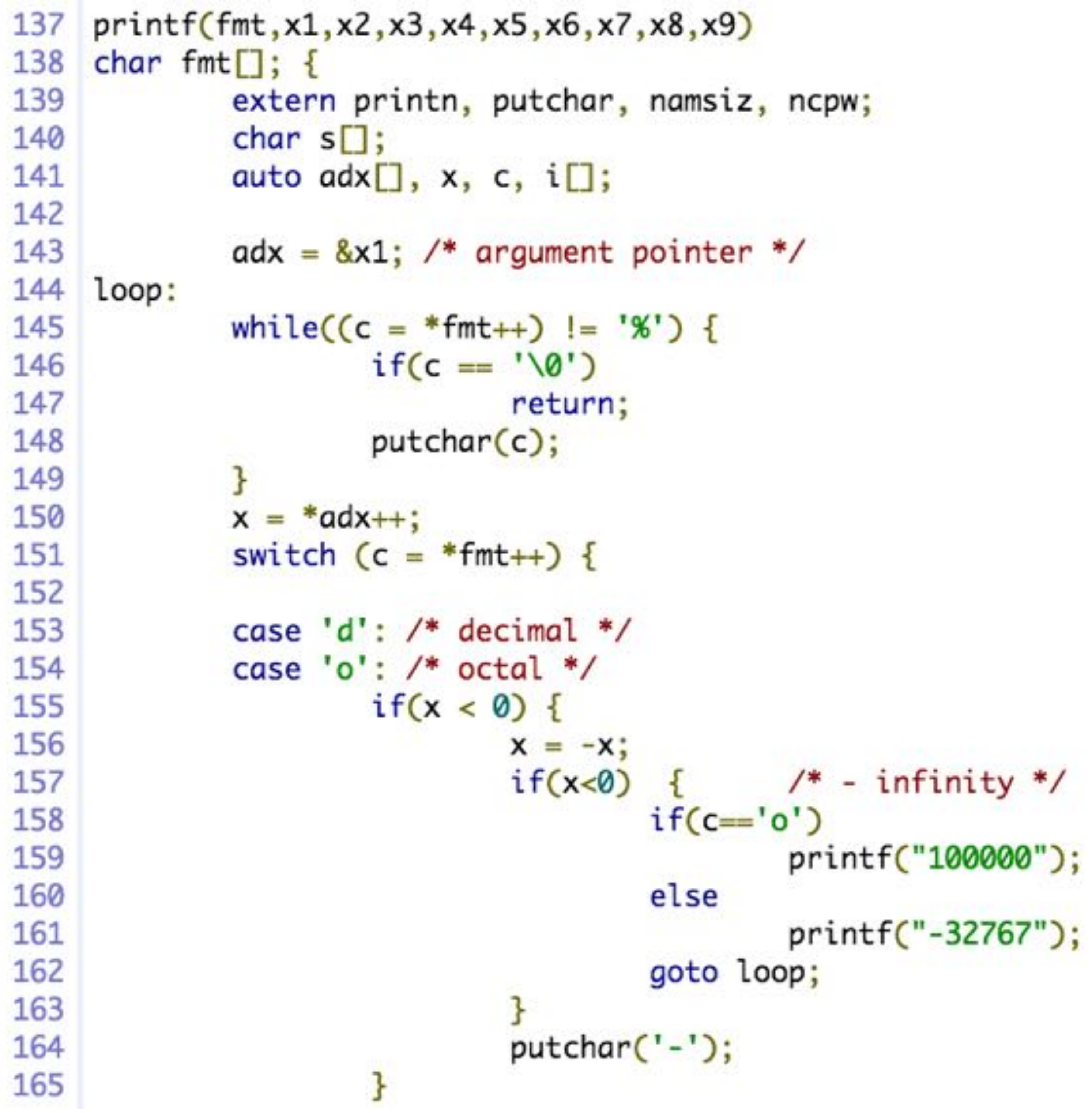
heavily inspired by Martin Richards' portable systems programming language BCPL.

Martin Richards, Dec 2014

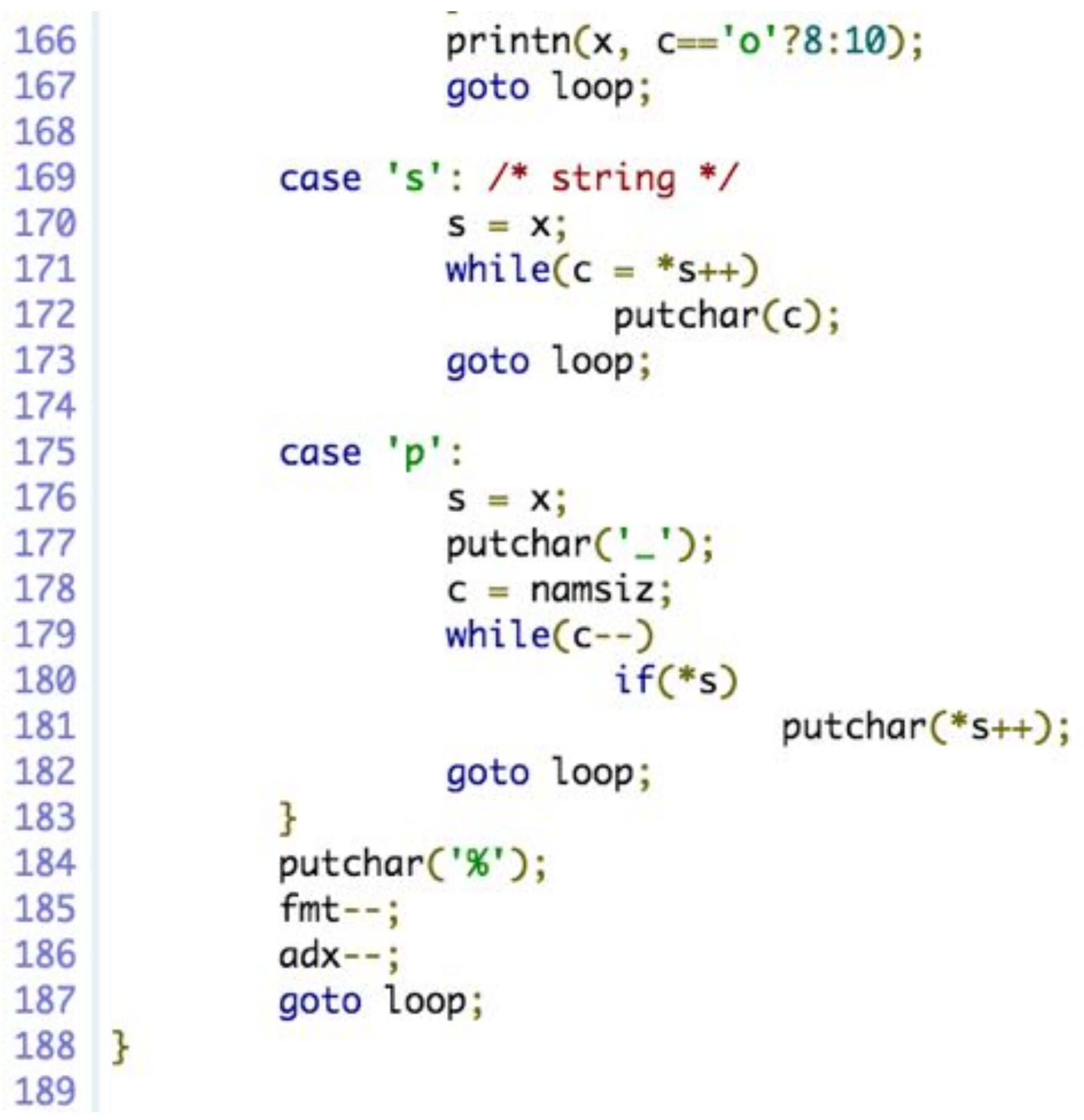
```
GET "LIBHDR"
LET START() BE WRITES("Hello, World")
```

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

In 1972 Unix was rewritten in C,



```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('-');
                }
```



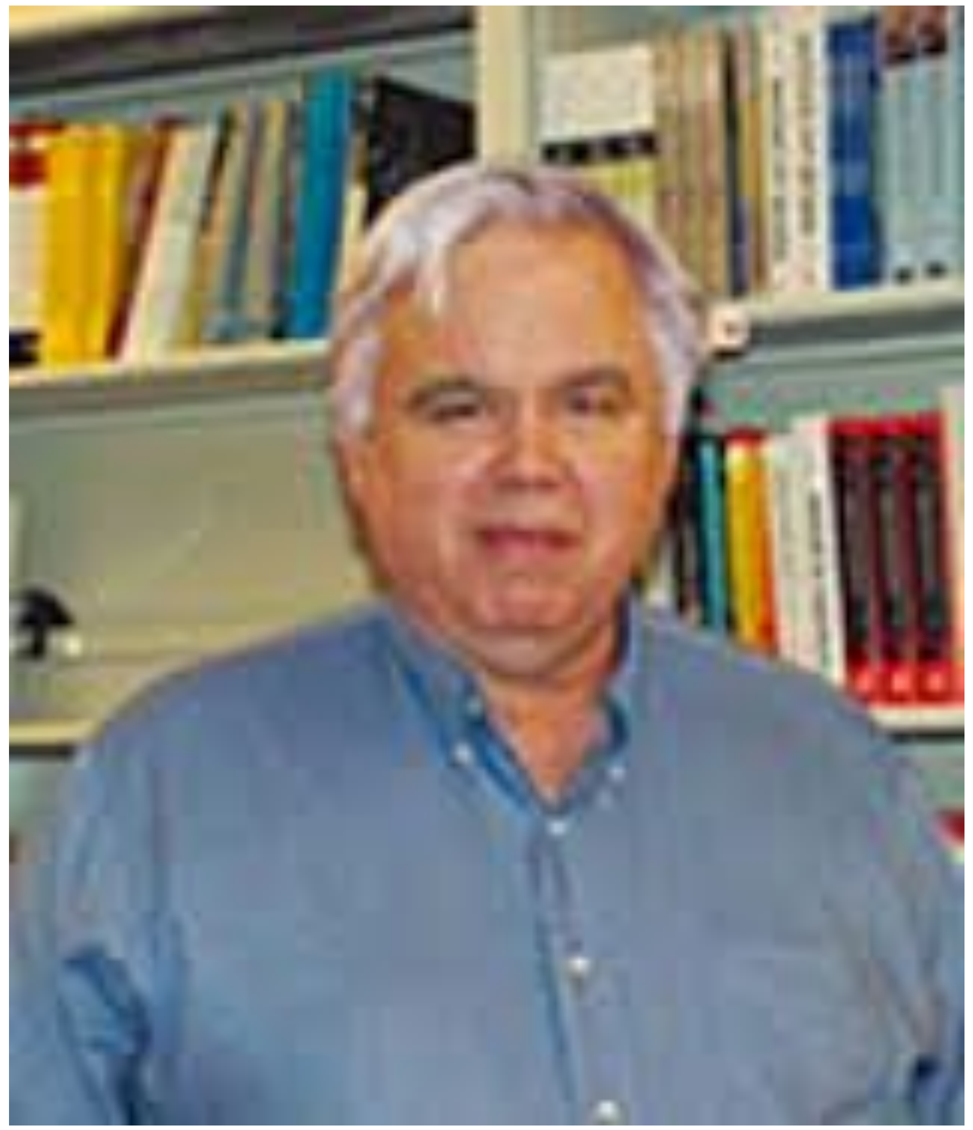
```c
                printn(x, c=='o'?8:10);
                goto loop;

        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

and later ported to many other machines

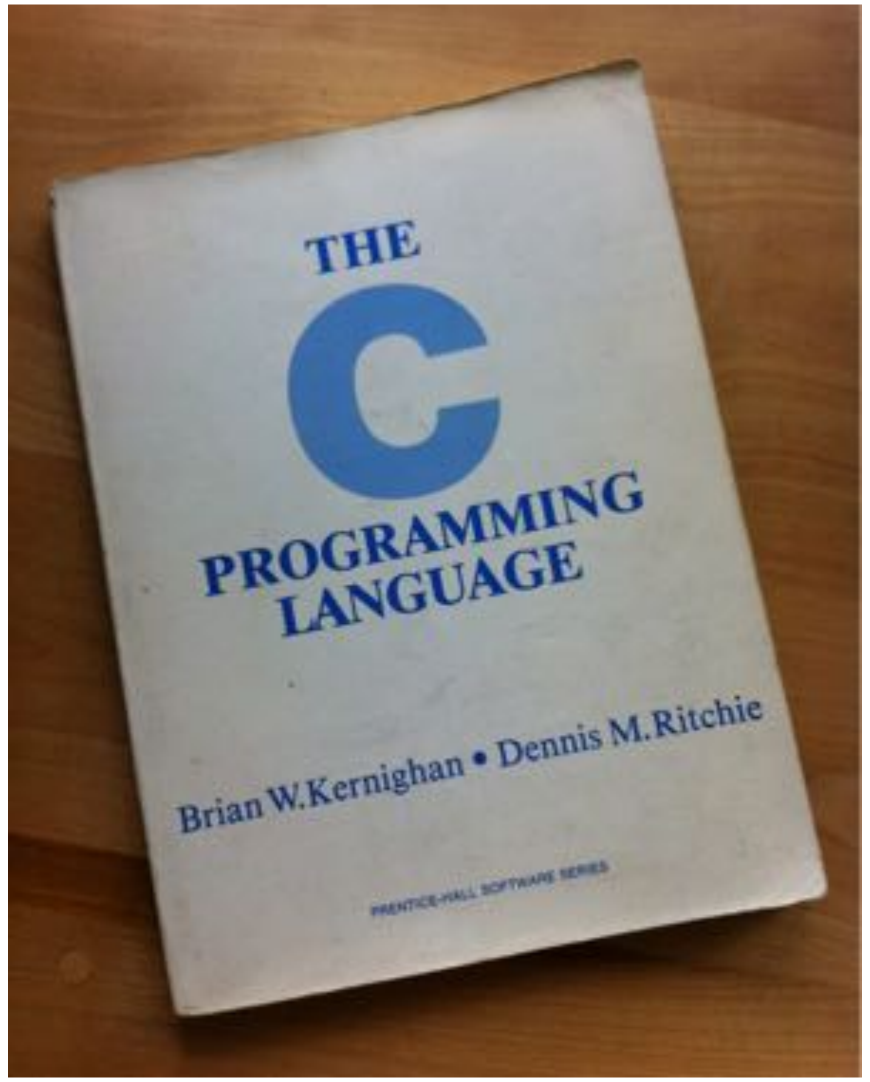
http://www.computerhistory.org/collections/catalog/102691249 http://www.technikum29.de/en/computer/early-computers http://en.wikipedia.org/wiki/IBM_System/370 http://alegion63.tripod.com/bob/id6.html

aided by Steve Johnsons Portable C Compiler.



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

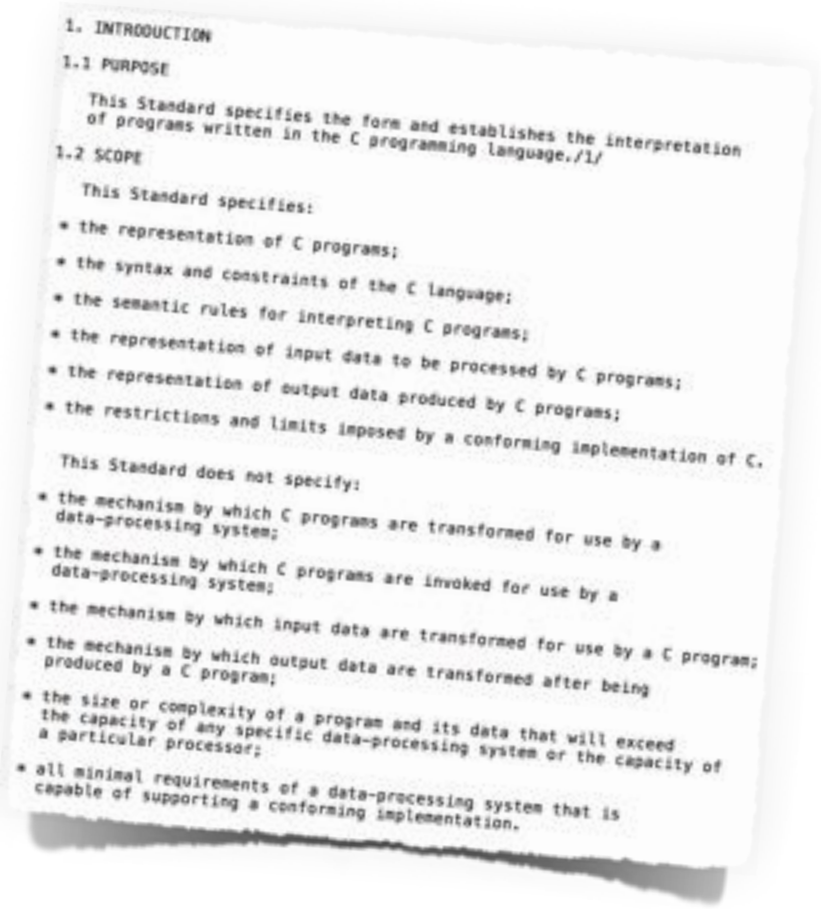
C also gained popularity outside the realm of PDP-11 and Unix.


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

K&R (1978)

http://blogs.jpmsonline.com/wp-content/uploads/2014/11/Success.jpg

Initially K&R was the definitive reference until the language was standardized by ANSI and ISO 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

[illegible]

Programming languages — C

Langages de programmation — C

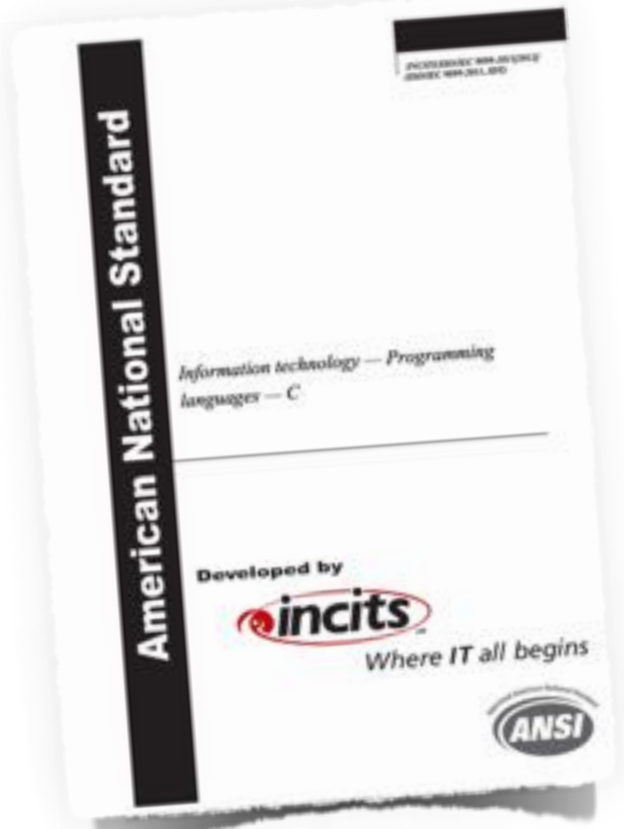
Adopted by INCITS (International Committee for Information Technology Standards) as an American National Standard. [illegible]

Printed in the United States of America

ISO IEC

[illegible]

C99



[illegible]

American National Standard

Information technology — Programming languages — C

Developed by

incits

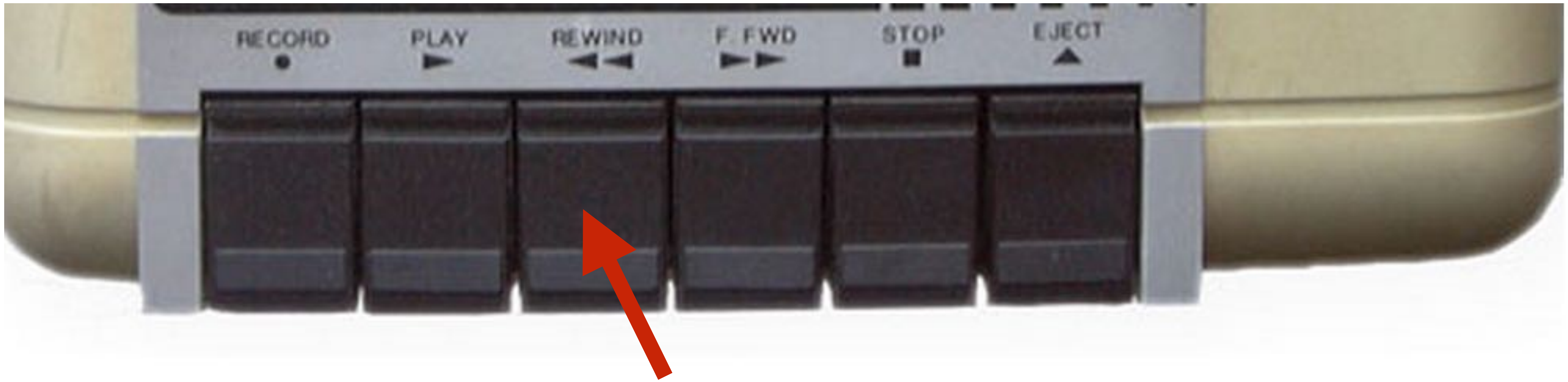
Where IT all begins

ANSI

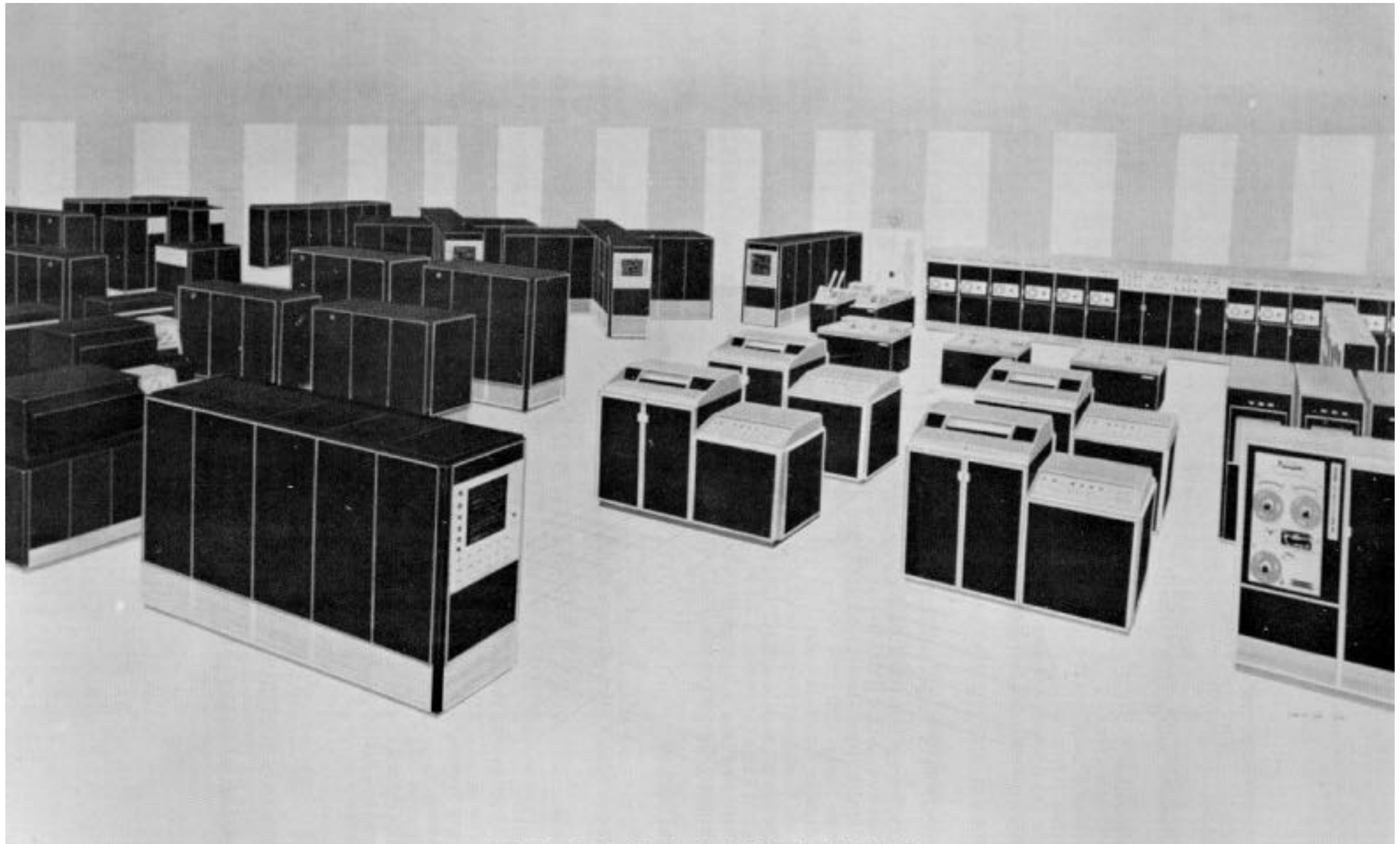
C11

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, and later ported to many other machines aided by Steve Johnsons Portable C Compiler. C gained popularity outside the realm of PDP-11 and Unix. Initially the K&R was the definitive reference until the language was standardized by ANSI and ISO 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.

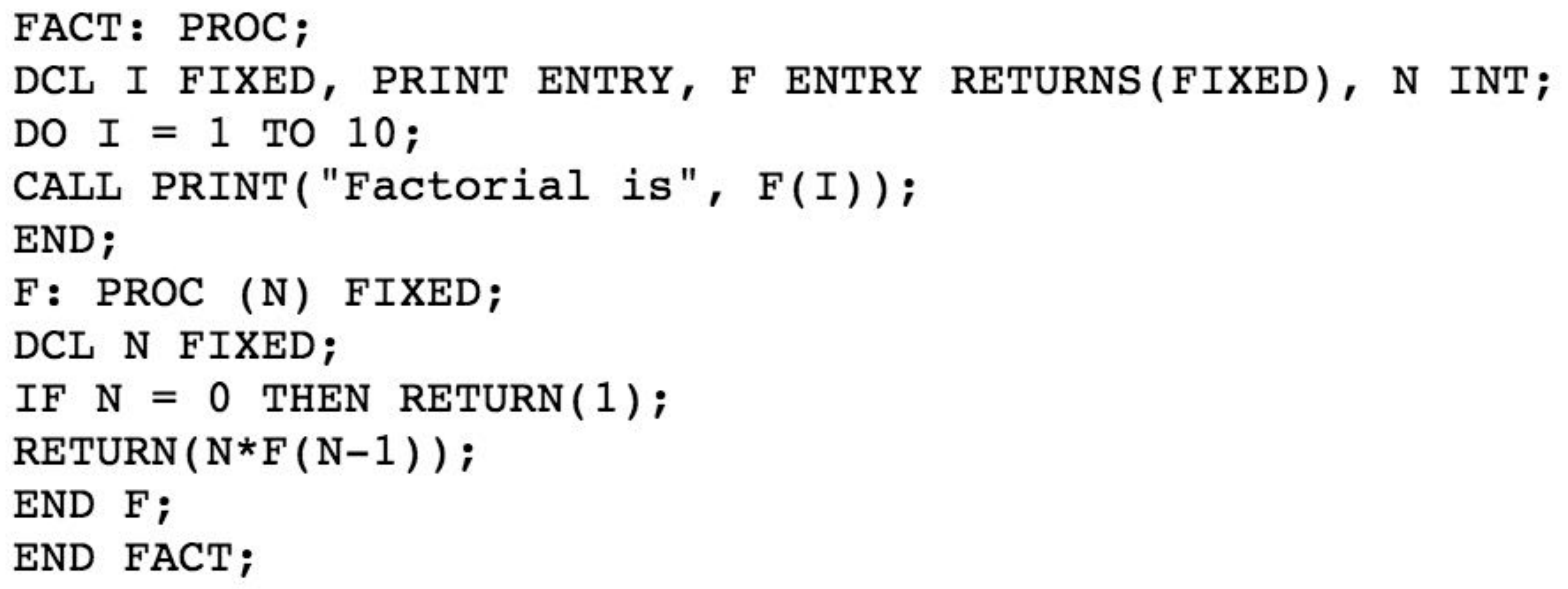

GE-645 SYSTEM

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

The MULTICS ("Multiplexed Information and Computing Service) was started in 1964, as a cooperative project led by MIT's Project MAC (Multiple Access Computing), General Electric and Bell Labs.

Bell Labs pulled out of the project in 1969.

Multics was a huge project, with great ambitions. It was a secure time-sharing system with lots of advanced features, and it was one of the few operating systems at the time written in a high level language, PL/1.



```
FACT: PROC;
DCL I FIXED, PRINT ENTRY, F ENTRY RETURNS(FIXED), N INT;
DO I = 1 TO 10;
CALL PRINT("Factorial is", F(I));
END;
F: PROC (N) FIXED;
DCL N FIXED;
IF N = 0 THEN RETURN(1);
RETURN(N*F(N-1));
END F;
END FACT;
```

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

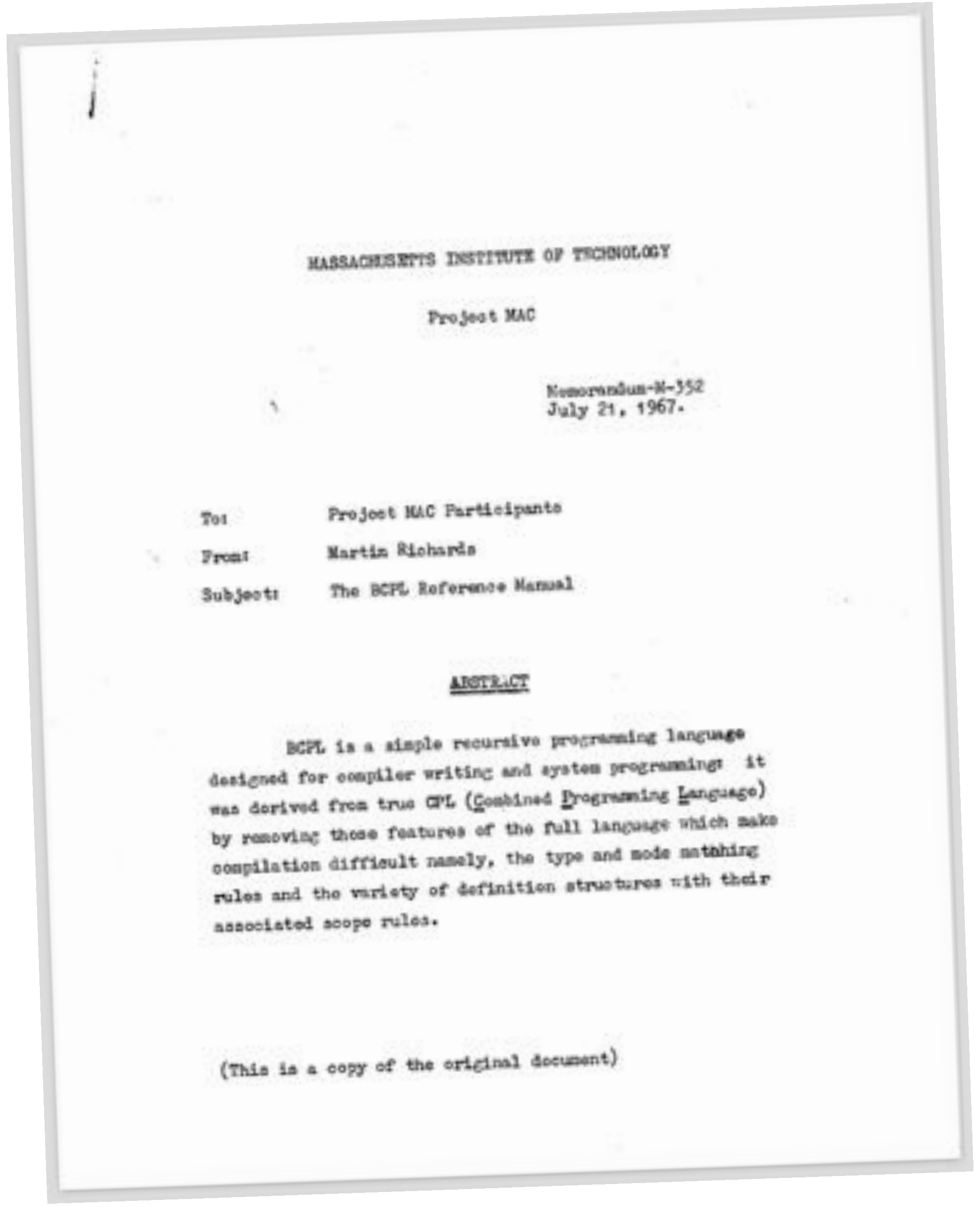
While working on the Multics projects, Dennis and Ken had also been exposed to the very portable language systems programming language BCPL.

```
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, Basic CPL, had been described and implemented for the Project MAC in 1967 by a visiting researcher, Martin Richards from Cambridge University.



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Memorandum-M-352
July 21, 1967.

To: Project MAC Participants

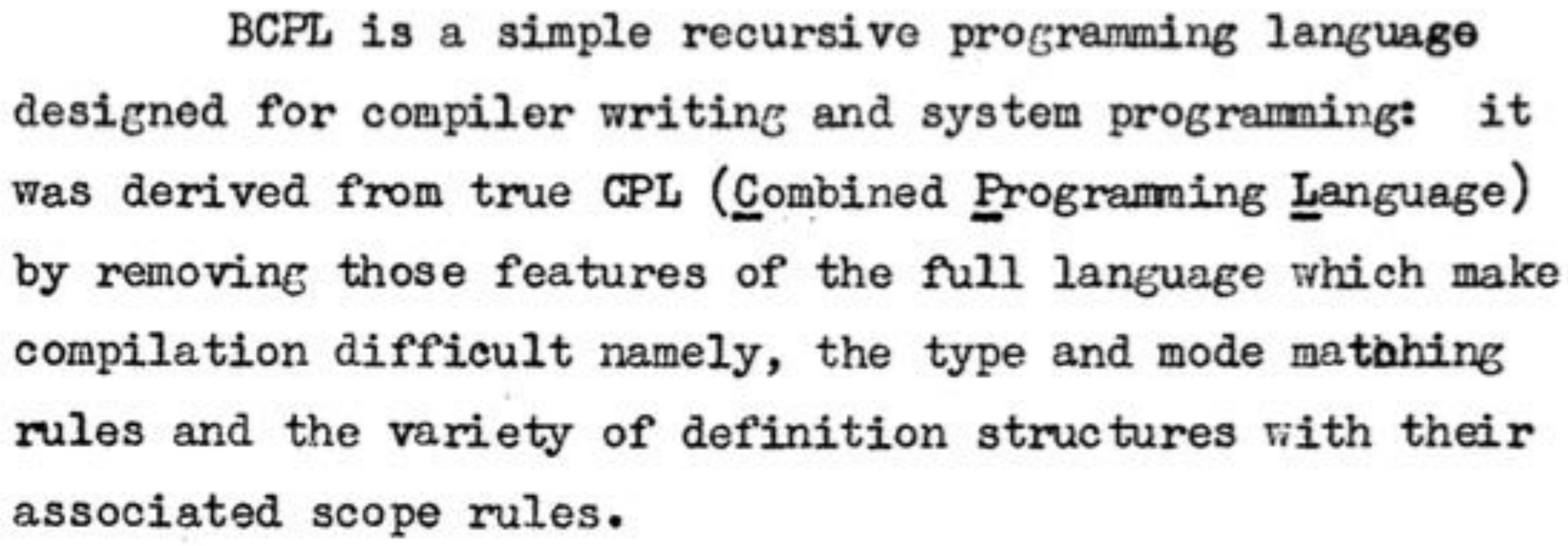
From: Martin Richards

Subject: The BCPL Reference Manual

ABSTRACT

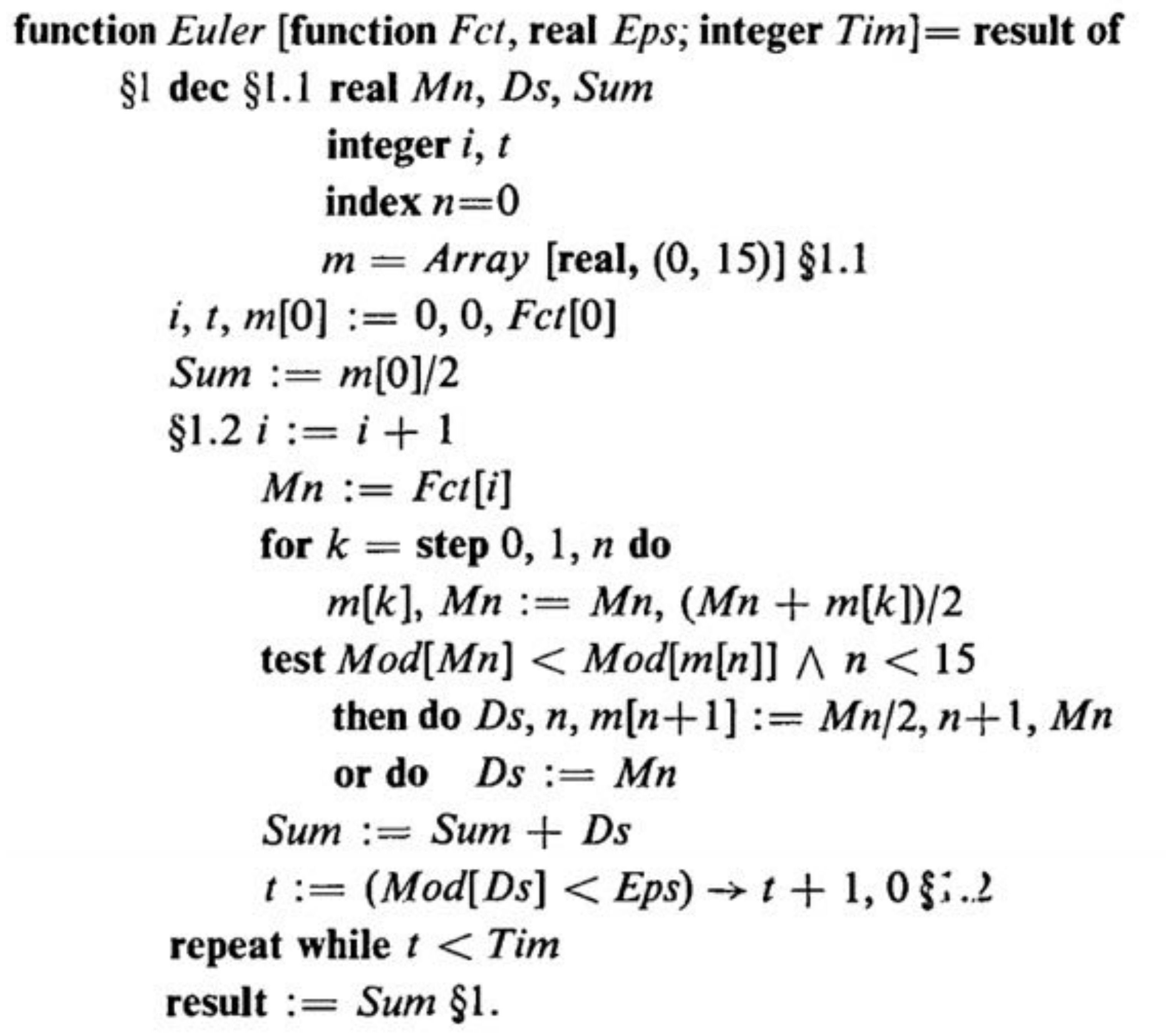
BCPL is a simple recursive programming language designed for compiler writing and system programming: it was derived from true CPL (Combined Programming Language) by removing those features of the full language which make compilation difficult namely, the type and mode matbhing rules and the variety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

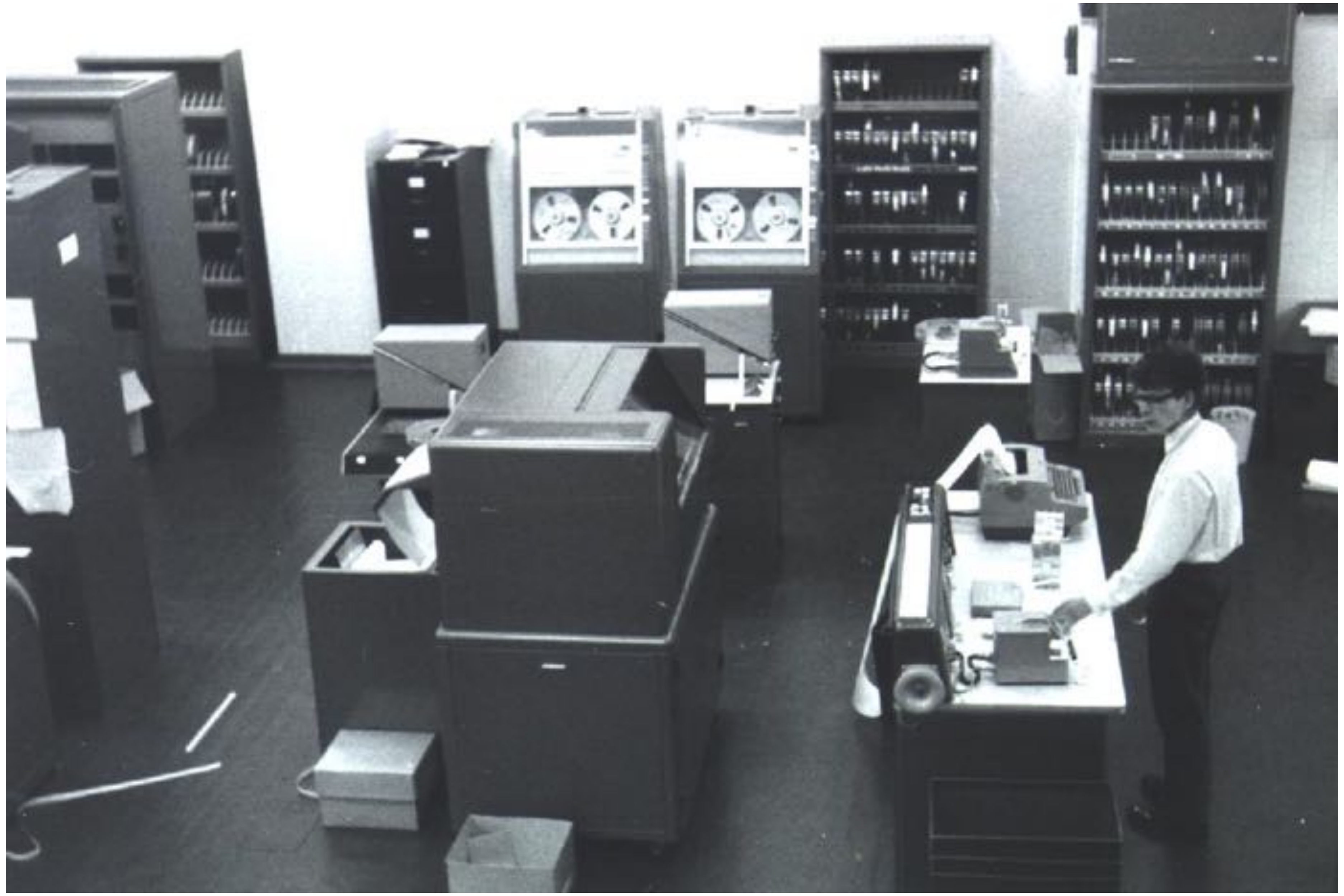
Before visiting MIT, Martin Richards had been actively involved in developing a compiler for a very ambitious programming language - CPL.



```
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.
```

Designed jointly by the Mathematical Laboratory at the University of Cambridge and the University of London Computer Unit

for the Atlas computer (ordered in 1961, operational in 1964)

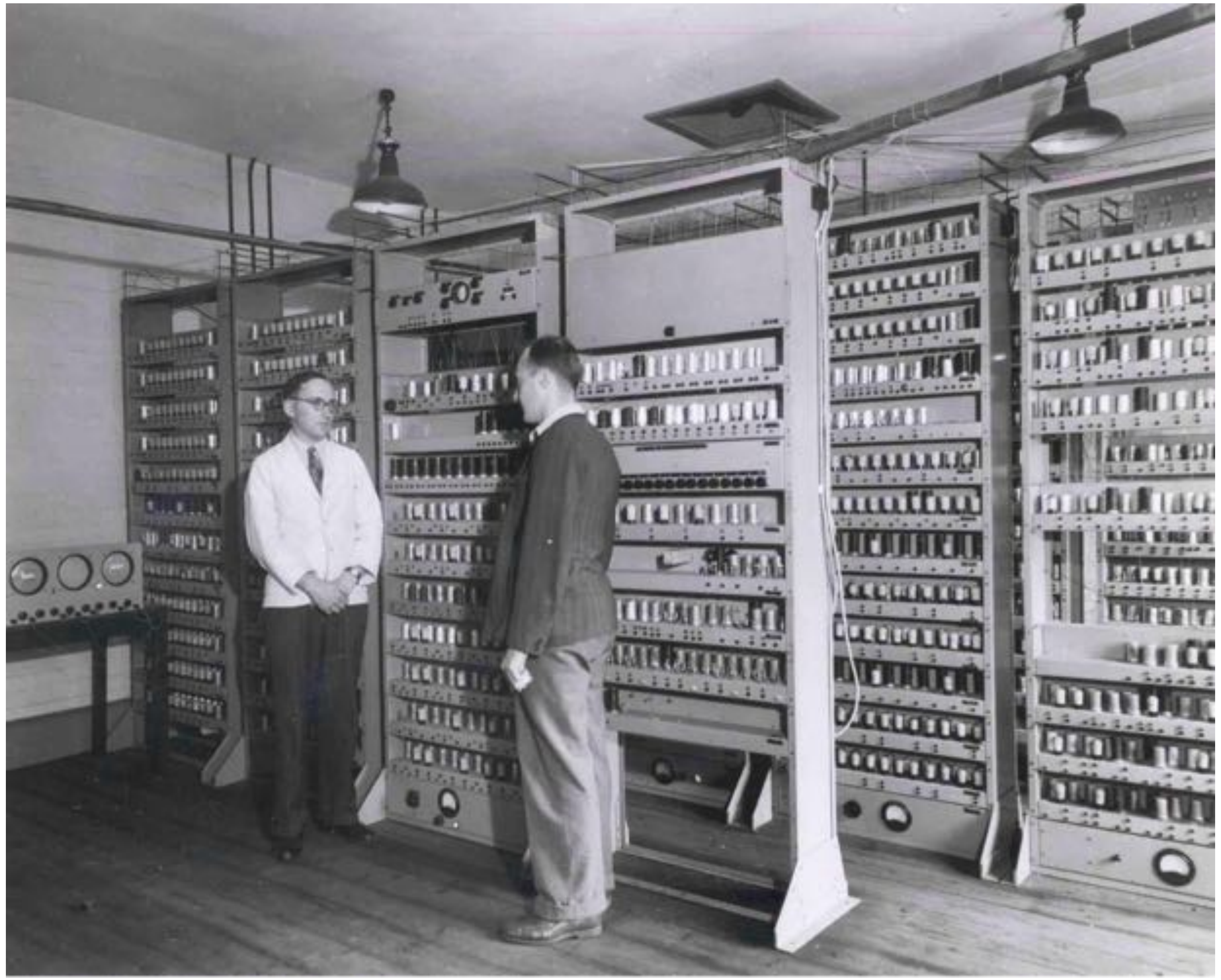


CPL was designed and partly implemented before the Atlas computer was operational. Martin Richard and the others had to work on the EDSAC 2 computer.

EDSAC 2 users in 1960

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

Which was an upgrade of the EDSAC computer. 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

The EDSAC 1951 film
abridged version

Commentary by
M. V. Wilkes

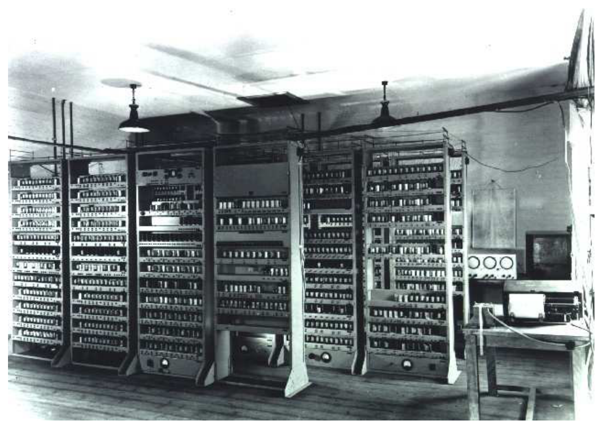
EDSAC Initial Orders and Squares Program

Martin Richards

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

EDSAC

EDSAC (Electronic Delay Storage Automatic Computer), pictured below, was the world's first stored-program computer to operate a regular computing service. Maurice Wilkes lead the team responsible for its design and construction. It ran its first program successfully on May 6, 1949.



EDSAC's main memory used mercury delay lines to hold 512 words of 35 bits. We will use the notation: $w[0]$, $w[2]$,...,$w[1022]$ to refer to these words of memory. Each word could be split into two 17-bit halves, separated by a padding bit. We will use the notation $m[a]$, $a = 0, 1, .., 1023$ to represent these 17-bit memory locations. The word at address $2n$, namely $w[2n]$, consisted of the concatenation of $m[2n+1]$, a padding bit, and $m[2n]$. Note that $m[1]$ is the senior half of $w[0]$.

```
                       17                   1                 17
w[2n]:  | * * * * * * * * * * * * * * * * * | * | * * * * * * * * * * * * * * * * * |
                    m[2n+1]                               m[2n]
```

The machine had two central registers visible to the user: the 71-bit accumulator and the 35-bit multiplier register. We will use the notation ABC to represent the whole accumulator, and A and AB to represent its senior 17 and 35 bits, respectively. We will use RS to represent the whole multiplier register and R to represent its senior 17 bits. The leftmost bit of each register was the sign bit and the remaining bits form a binary fraction.

EDSAC's machine instructions (also called orders) occupied 17 bits. The leftmost 5 bits was the operation code, the next bit was unused, the following 10 bits was the address field and the last bit specified (where appropriate) whether the order used 17 or 35-bit operands.

```
                   5        1          10           1
Order format:  | * * * * * | * | * * * * * * * * * * | * |
                   Op     Unused     Address         S/L
```

Orders were punched on paper tape and consisted of: a character that directly gave the 5-bit operation code, followed by zero or more decimal digits giving the address, and terminated by S or L specifying the operand length bit. For example, R16S assembled to `00100 0 0000010000 0` and T11L to `00101 0 0000001011 1`. Note that the characters R and T had codes 4 and 5, respectively.

The Character Set

EDSAC used 5-bit integers (0 to 31) to represent characters using two shifts: letters and figures. In letter shift the codes 0 to 31 respectively represented: P, Q, W, E, R, T, Y, U, I, O, J, *figs*, S, Z, K, *lets*, *null*, F, *cr*, D, *sp*, H, N, M, *lf*, L, X, G, A, B, C and V. In figure shift the encoding was as follows: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ?, *figs*, ", +, (, *lets*, *null*, $, *cr*, ;, *sp*, £, ,, ., *lf*,), /, #, -, ?, : and =. In these tables, *figs*, *cr*, *sp* and *lf* denote figure shift, carriage return, space and line feed, and on the paper tape perforator their keys were labelled π, θ, ϕ and Δ, respectively. In this document, these codes correspond to the ASCII characters #, @, ! and &. The paper tape reader complemented the high order bit of each 5-bit character, so the rows ● ·, ● ·●●● and ●· ●● are read as codes 0(P), 7(U) and 27(G), respectively. The machine could read paper tape at a rate of 50 characters per second and output to a Creed teleprinter at nearly 7 characters per second.

The 1949 Instruction set

EDSAC's instructions in 1949 was very simple and were executed at a rate of about 600 per second. They were as follows:

AnS:	A += $m[n]$	AnL:	AB += $w[n]$
SnS:	A -= $m[n]$	SnL:	AB -= $w[n]$
HnS:	R += $m[n]$	HnL:	RS += $w[n]$
VnS:	AB += $m[n]$ * R	VnL:	ABC += $w[n]$ * RS
NnS:	AB -= $m[n]$ * R	NnL:	ABC -= $w[n]$ * RS
TnS:	$m[n]$ = A; ABC = 0	TnL:	$w[n]$ = AB; ABC = 0
UnS:	$m[n]$ = A	UnL:	$w[n]$ = AB
CnS:	AB += $m[n]$ & R	CnL:	ABC += $w[n]$ & RS
RnS, RnL:	Shift ABC right arithmetically by the number of places corresponding to the position of the least significant one in the shift instruction. For example, R0L, R1S, R16S and R0S shift by 1, 2, 6 and 15 places, respectively.		
LnS, LnL:	Shift ABC left arithmetically by the number of places corresponding to the position of the least significant one in the shift instruction. For example, L0L, L1S, L16S, L64S and L0S shift by 1, 2, 6, 8 and 13 places, respectively.		
EnS:	if A >= 0 goto n		
GnS:	if A < 0 goto n		
InS:	Place the next paper tape character in the least significant 5 bits of $m[n]$.		
OnS:	Output the character in the most significant 5 bits of $m[n]$.		
FnS:	Verify the last character output.		
XnS:	No operation.		
YnS:	Add a one to bit position 35 of ABC, counting the sign bit as bit zero. This effectively rounds ABC up to 34 fractional bits.		
ZnS:	Stop the machine and ring a bell.		

The numerical values in the accumulator and multiplier registers are normally thought of as signed binary fractions, but integer operations could also be done easily. For example, the order V1S can be interpreted as adding the product of the 17-bit signed integer in m[1] and to the 17-bit integer in RS and adding the result into bits 0 to 32 of the ABC. With a suitable shift, the integer result can be placed in the senior 17 bits of A ready for storing in memory.

Initial Orders

The four glass panels on your right contain 20 segments of 5 track paper tape. Reading from right to left and from top to bottom, the first five segments correspond to the initial orders, and the remaining 15 to a program to compute squares. The glass panels contain errors so a corrected version of the panels are given below.

The initial orders were written by David Wheeler in May 1949 to load and enter a paper tape represention of a program. When EDSAC was started, these initial orders were placed in memory locations 0 to 30 by a mechanism involving uniselectors before execution stared from location 0.

The glass panels give a paper tape representation of these orders even though no such paper tape ever existed. The following is an annotated listing of this program.

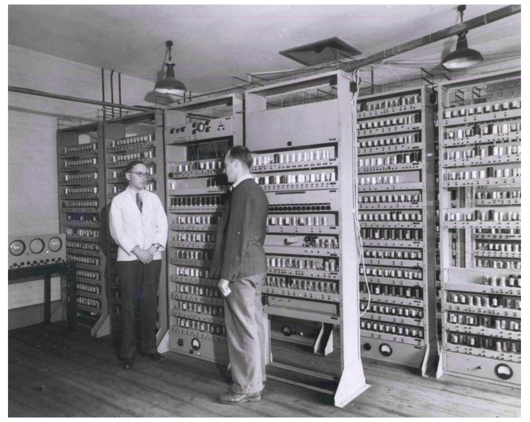
Order bit pattern	Loc	Order	Meaning	Comment
00101 0 0000000000 0	0:	T0S	$m[0]$=A; ABC=0	
10101 0 0000000010 0	1:	H2S	R=$m[2]$	Put 10<<11 in R
00101 0 0000000000 0	2:	T0S	$m[0]$=A; ABC=0	
00011 0 0000000110 0	3:	E6S	goto 6	Jump to main loop
00000 0 0000000001 0	4:	P1S	data 2	The constant 2
00000 0 0000000101 0	5:	P5S	data 10	The constant 10
00101 0 0000000000 0	6:	T0S	$m[0]$=A; ABC=0	Start of the main loop
01000 0 0000000000 0	7:	I0S	$m[0]$=rdch()	Get operation code
11100 0 0000000000 0	8:	A0S	A+=$m[0]$	Put it in A
00100 0 0000010000 0	9:	R16S	ABC>>=6	Shift and store it
00101 0 0000000000 1	10:	T0L	$w[0]$=AB; ABC=0	so that it becomes the senior 5 bits of $m[0]$ $m[1]$ is now zero
01000 0 0000000010 0	11:	I2S	$m[2]$=rdch()	Put next ch in $m[2]$
11100 0 0000000010 0	12:	A2S	A+=$m[2]$	Put ch in A
01100 0 0000000101 0	13:	S5S	A-=$m[5]$	A=ch-10
00011 0 0000010101 0	14:	E21S	if A>=0 goto 21	Jump to 21, if ch>=10
00101 0 0000000011 0	15:	T3S	$m[3]$=A; ABC=0	Clear A, $m[3]$ is junk
11111 0 0000000001 0	16:	V1S	AB+=$m[1]$*R	A = $m[1]$*(10<<11)
11001 0 0000001000 0	17:	L8S	A<<=5	Shift 5 more places
11100 0 0000000010 0	18:	A2S	A+=$m[2]$	Add the new digit
00101 0 0000000001 0	19:	T1S	$m[1]$=A; ABC=0	Store back in $m[1]$
00011 0 0000001011 0	20:	E11S	goto 11	Repeat from 11
00100 0 0000000100 0	21:	R4S	ABC>>=4	A=2, if ch='S'(=12) A=15, if ch='L'(=25) lenbit=0, if ch='S' lenbit=1, if ch='L'
11100 0 0000000001 0	22:	A1S	A+=$m[1]$	Add in the address
11001 0 0000000000 1	23:	L0L	ABC<<=1	Shift to correct position
11100 0 0000000000 0	24:	A0S	A+=$m[0]$	Add in the operation field
00101 0 0000011111 0	25:	T31S	$m[31]$= A; ABC=0	Store the order in next location
11100 0 0000011001 0	26:	A25S	A+=$m[25]$	Increment the address field of $m[25]$
11100 0 0000000100 0	27:	A4S	A+=$m[4]$	$m[4]$ holds 2
00111 0 0000011001 0	28:	U25S	$m[25]$=A	Update $m[25]$
01100 0 0000011111 0	29:	S31S	A-=$m[31]$	Jump to 6, if there are
11011 0 0000000110 0	30:	G6S	if A<0 goto 6	more orders to load

The instruction at location 0 does nothing useful, but the instruction at 1 loads the multiplier register R with a 17-bit pattern 00101000000000000 which is also 10 shifted left 11 places. The instruction instruction at 2 (T0S) assembles into exactly this bit pattern, so is used both as data and as an instruction to clear $m[0]$. The instruction at 3 skips to location 6 over the instructions at 4 and 5 that assemble as the 17-bit constants 2 and 10, respectively.

The main assembly loop starts at 6, leaving locations $m[0]$ to $m[5]$ available as variables and constants in the program. They are used as follows:

$m[0]$	uses include holding the first character of an order,
$m[1]$	used to hold the address field of the current order,
$m[2]$	initially 001010...0 as discussed above but also used for characters other than the first of an order,
$m[3]$	used as a junk register when the instruction at 15 clears ABC,
$m[4]$	the constant 2 used at 27 to add one to an address field,
$m[5]$	the constant 10 used to check for the end of address digits.

The order at 25 is of the form TnS, initially T31S. It is used to store an order at location n. This instruction is modified by the code in locations 26 to 28 which adds one to its address field, so the next time it is executed it will update the next location. Location 31 is the first order to be loaded and must be of the form TnS where n-1 is the address of last instruction of the program. It is used by the code in locations 29 and 30 which compares it with the current version of TnS in 25. If loading is not yet complete execution jumps to 11, otherwise it fall through to 31. Note that the instruction at 31 will do no damage, since it just writes a value to the first location following the loaded program. The first real instruction of the program is in $m[32]$.



M.V Wilkes and W.A. Renwick

The Squares Program

This program, written by Maurice Wilkes in June 1949, outputs the following table of squares and differences of the numbers 1 to 100.

1	1	1
2	4	3
3	9	5
....	.....	.....
98	9604	195
99	9801	197
100	10000	199

The following is an annotated listing of the program.

Order bit pattern	Loc	Order	Meaning	Comment
00101 0 0001111011 0	31:	T123S	m[123]=A; ABC=0	The required first word
00011 0 0001010100 0	32:	E84S	goto 84	Jump to start
00000 0 0000000000 0	33:	PS	data 0	For the next decimal digit
00000 0 0000000000 0	34:	PS	data 0	For the current power of ten
00100 1 1100010000 0	35:	P10000S	data 10000<<1	The table of 16-bit powers of ten
00000 0 1111101000 0	36:	P1000S	data 1000<<1	
00000 0 0001100100 0	37:	P100S	data 100<<1	
00000 0 0000001010 0	38:	P10S	data 10<<1	
00000 0 0000000001 0	39:	P1S	data 1<<1	
00001 0 0000000000 0	40:	QS	data 1<<12	00001 in MS 5 bits, used to form digits
01011 0 0000000000 0	41:	#S	data 11<<12	Figure shift character
11100 0 0000101000 0	42:	A40S		End limit for values placed in $m[52]$
10100 0 0000000000 0	43:	!S	data 20<<12	Space character
11000 0 0000000000 0	44:	&S	data 24<<12	Line feed character
10010 0 0000000000 0	45:	@S	data 18<<12	Carriage return character
01001 0 0000101011 0	46:	O43S	wr(m[43])	Write a space
01001 0 0000100001 0	47:	O33S	wr(m[33])	Write a digit
00000 0 0000000000 0	48:	PS	data 0	The number to print
11100 0 0000101110 0	49:	A46S	A+=m[46]	Print subroutine entry point
00101 0 0001000001 0	50:	T65S	$m[65]$=A; ABC=0	Put O43S in $m[65]$
00101 0 0010000001 0	51:	T129S	$m[129]$=A; ABC=0	Clear A
11100 0 0000100011 0	52:	A35S	A+=$m[35]$	A is next power of ten. $m[52]$ cycles through A35S, A36S, A37S, A38S and A39S
00101 0 0000100010 0	53:	T34S	$m[34]$=A; ABC=0	Store it in $m[34]$
00011 0 0000111101 0	54:	E61S	goto 61	
00101 0 0000110000 0	55:	T48S	$m[48]$=A; ABC=0	Store value to be printed
11100 0 0000101111 0	56:	A47S	A+=$m[47]$	Store instruction O33S
00101 0 0001000001 0	57:	T65S	$m[65]$=A; ABC=0	in $m[65]$
11100 0 0000100001 0	58:	A33S	A+=$m[33]$	Increment the decimal digit
11100 0 0000101000 0	59:	A40S	A+=$m[40]$	held in the MS 5 bits
00101 0 0000100001 0	60:	T33S	$m[33]$=A; ABC=0	of $m[33]$
11100 0 0000110000 0	61:	A48S	A+=$m[48]$; ABC=0	Get value to print
11100 0 0000100010 0	62:	S34S	A-=$m[34]$	Subtract a power of 10
00011 0 0000110111 0	63:	E55S	if A>=0 goto 55	Repeat, if positive
11100 0 0000100010 0	64:	A34S	A+=$m[34]$	Add back the power of 10
00000 0 0000000000 0	65:	PS	data 0	This is replaced by either O43S to write a space, or O33S to write a digit
00101 0 0000110000 0	66:	T48S	$m[48]$=A; ABC=0	Set the value to print
00101 0 0000100001 0	67:	T33S	$m[33]$=A; ABC=0	Set digit to 0
11100 0 0000110100 0	68:	A52S	A+=$m[52]$	Increment the address field
11100 0 0000000100 0	69:	A4S	A+=$m[4]$	of the instruction
00111 0 0000110100 0	70:	U52S	$m[52]$=A	in $m[52]$
01100 0 0000101010 0	71:	S42S	A-=$m[42]$	Compare with A40S and
11011 0 0000110011 0	72:	G51S	if A<0 goto 51	Repeat, if more digits
11100 0 0001110101 0	73:	A117S	A+=$m[117]$	Put A35S back
00101 0 0000110100 0	74:	T52S	$m[52]$=A; ABC=0	in $m[52]$
00000 0 0000000000 0	75:	PS	data 0	To hold the return jump instruction which is E95S, E110S or E118S
00000 0 0000000000 0	76:	PS	data 0	Holds x
00000 0 0000000000 0	77:	PS	data 0	Holds x^2
00000 0 0000000000 0	78:	PS	data 0	Holds previous x^2
00000 0 0000000000 0	79:	PS	data 0	Holds Δx^2
00011 0 0001101110 0	80:	E110S	goto 110	Order to place in $m[52]$
00011 0 0001110110 0	81:	E118S	goto 118	Order to place in $m[52]$
00000 0 0001100100 0	82:	P100S	data 100<<1	End limit for x
00011 0 0001011111 0	83:	E95S	goto 95	Order to place in $m[52]$
01001 0 0000101001 0	84:	O41S	wr($m[41]$)	Write figure shift
00101 0 0010000001 0	85:	T129S	$m[129]$=A; ABC=0	Start of main loop
01001 0 0000101100 0	86:	O44S	wr($m[44]$)	Write line feed
01001 0 0000101101 0	87:	O45S	wr($m[45]$)	Write carriage return
11100 0 0001001100 0	88:	A76S	A+=$m[76]$; ABC=0	Get x
11100 0 0000000100 0	89:	A4S	A+=$m[4]$	Increment it
00111 0 0001001100 0	90:	U76S	$m[76]$=A	and store it back in x
00101 0 0000110000 0	91:	T48S	$m[48]$=A; ABC=0	Put it also in $m[48]$ for printing
11100 0 0001010011 0	92:	A83S	A+=$m[83]$	Put return jump E95S
00101 0 0001001011 0	93:	T75S	$m[75]$=A; ABC=0	into $m[75]$
00011 0 0000110001 0	94:	E49S	goto 49	Enter the print subroutine
01001 0 0000101011 0	95:	O43S	wr($m[43]$)	Write a space
01001 0 0000101011 0	96:	O43S	wr($m[43]$)	Write a space
10101 0 0001001100 0	97:	H76S	R=$m[76]$	Multiply x by
11111 0 0001001100 0	98:	V76S	ABC+=$m[76]$*RS	itself and
11001 0 0001000000 0	99:	L64S	ABC<<8	re-position
11001 0 0000100000 0	100:	L32S	ABC<<7	the result
00111 0 0001001101 0	101:	U77S	$m[77]$=A	Store in location for x^2
01100 0 0001001110 0	102:	S78S	A-=$m[78]$	Subtract the previous value
00101 0 0001001111 0	103:	T79S	$m[79]$=A; ABC=0	and store the new Δx^2
11100 0 0001001101 0	104:	A77S	A+=$m[77]$	Update variable holding
00111 0 0001001110 0	105:	U78S	$m[78]$=A	the previous x^2
00101 0 0000110000 0	106:	T48S	$m[48]$=A; ABC=0	Put x^2 in $m[48]$ for printing
11100 0 0001010000 0	107:	A80S	A+=$m[80]$	Put return jump E110S
00101 0 0001001011 0	108:	T75S	$m[75]$=A; ABC=0	into $m[75]$
00011 0 0000110001 0	109:	E49S	goto 49	Enter the print subroutine
01001 0 0000101011 0	110:	O43S	wr($m[43]$)	Write a space
01001 0 0000101011 0	111:	O43S	mr($m[43]$)	Write a space
11100 0 0001001111 0	112:	A79S	A+=$m[79]$	Get Δx^2
00101 0 0000110000 0	113:	T48S	$m[48]$=A; ABC=0	Put it in $m[48]$ for printing
11100 0 0001010001 0	114:	A81S	A+=$m[81]$	Put return jump E118S
00101 0 0001001011 0	115:	T75S	$m[75]$=A; ABC=0	into $m[75]$
00011 0 0000110001 0	116:	E49S	goto 49	Enter the print subroutine
11100 0 0000100011 0	117:	A35S	A+=$m[35]$	Order to place in $m[52]$
11100 0 0001001100 0	118:	A76S	A+=$m[76]$	Get x
01100 0 0001010010 0	119:	S82S	A-=$m[82]$	Subtract the end limit (=100)
11011 0 0001010101 0	120:	G85S	if A<0 goto 85	Repeat, if more to do
01001 0 0000101001 0	121:	O41S	wr($m[41]$)	Write figure shift
01101 0 0000000000 0	122:	ZS	Stop	Stop the machine

The Green Door

The green door on your left was the Corn Exchange Street entrance to the Mathematical Laboratory where EDSAC was built. By convention, the brass plaque on this door holds the engraved names of those retired members of the Laboratory who used the door in its original location.

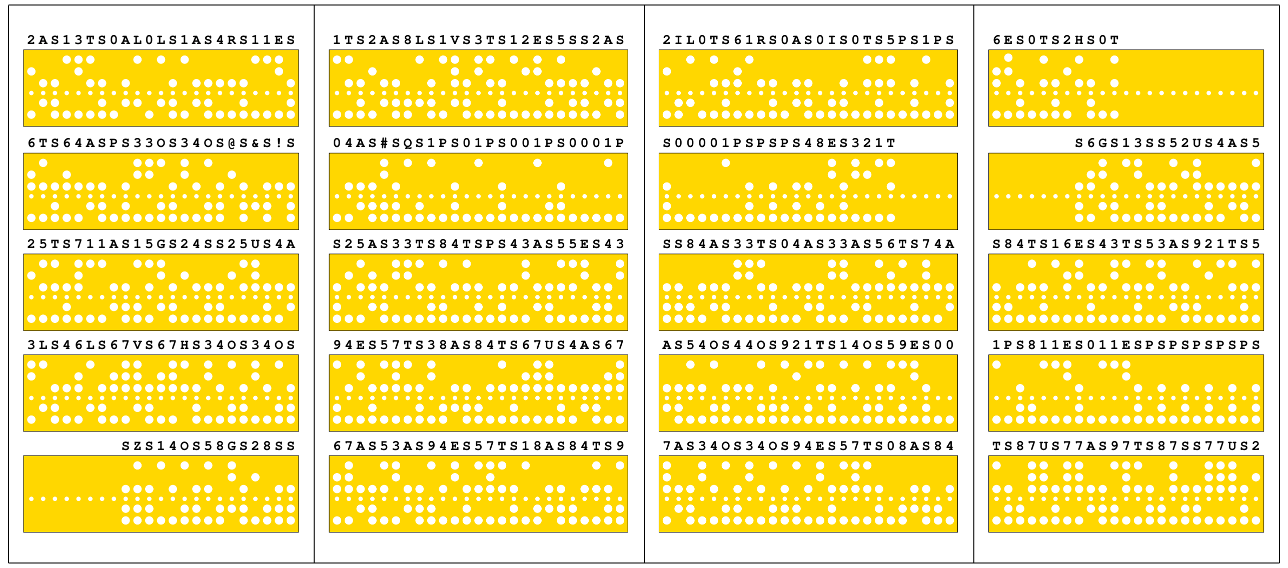
Links

`http://www.dcs.warwick.ac.uk/~edsac/`
This links to Martin Campbell-Kelly's excellent EDSAC simulator and related documents.

`http://www.cl.cam.ac.uk/UoCCL/misc/EDSAC99`
This links to pages relating to the celebration, held in Cambridge in April 1999, of the 50th anniversary of the EDSAC 1 Computer.

`http://www.cl.cam.ac.uk/~mr/Edsac.html`
This links to a shell based EDSAC simulator that runs on Pentium based Linux systems. It was designed to be educational having a built-in interactive debugger allowing single step execution, the setting of breakpoints and convenient inspection and setting of memory and register values. It can be used to explore the execution of the programs described in this poster. This simulator also appears as a demonstration program in the Cintcode BCPL system (`http://www.cl.cam.ac.uk/~mr/BCPL.html`).

`http://www.cl.cam.ac.uk/~mr/edsacposter.pdf`
This is a PDF version of this poster on two A4 pages.


2AS13TS0AL0LS1AS4RS11ES
1TS2AS8LS1VS3TS12ES5SS2AS
2IL0TS61RS0AS0IS0TS5PS1PS
6ES0TS2HS0T
6TS64ASPS33OS34OS@S&S!S
04AS#SQS1PS01PS001PS0001P
S00001PSPSPS48ES321T
S6GS13SS52US4AS5
25TS711AS15GS24SS25US4A
S25AS33TS84TSPS43AS55ES43
SS84AS33TS04AS33AS56TS74A
S84TS16ES43TS53AS921TS5
3LS46LS67VS67HS34OS34OS
94ES57TS38AS84TS67US4AS67
AS54OS44OS921TS14OS59ES00
1PS811ES011ESPSPSPSPSPS
SZS14OS58GS28SS
67AS53AS94ES57TS18AS84TS9
7AS34OS34OS94ES57TS08AS84
TS87US77AS97TS87SS77US2

The corrected tape segments etched on the Tea Room glass panels

http://www.cl.cam.ac.uk/~mr10/edsacposter.pdf

“Hi” on the EDSAC / Initial Orders 1

```
T44S            31          T _end+1        mark end of program
E38S            32          E _start        jump to beginning of program
*S              33  lshift  *               letter shift
HS              34  _H      H               letter H
IS              35  _I      I               letter I
&S              36  lf      &               LF - line feed character
@S              37  cr      @               CR - carriage return character
O33S            38  _start  O lshift        prepare for printing lettersn
O34S            39          O _H            print H
O35S            40          O _I            print I
O36S            41          O lf            print lf
O37S            42          O cr            print cr
ZS              43  _end    Z               end of program
```

```
T44SE38S*SHSIS&S@SO33SO34SO35SO36SO37SZS
```

“Count to 10” on the EDSAC / Initial Orders 1

```
T62S        31          T _end+1     mark end of program
E43S        32          E _start     jump to beginning of program
#S          33 fshift   #            figure shift
&S          34 lf       &            LF - line feed character
@S          35 cr       @            CR - carriage return character
PS          36 dummy    P            dummy (used to reset Acc)
P0S         37 first    P 0          first value
P9S         38 last     P 9          last value
P1S         39 incr     P 1          increment
PS          40 cur      P            current value
PS          41 d        P            d - digit to be printed
XS          42 _start   X            nop
O33S        43          O fshift     prepare for printing digits
T36S        44          T dummy      reset Acc
A37S        45          A first      load first
T40S        46          T cur        store to cur
XS          47 _loop    X            nop
T36S        48          T dummy      reset Acc
A40S        49          A cur        load current value
L512S       50          L 2^(11-2)   Acc << 11, create a digit
T41S        51          T d          store digit to be printed
O41S        52          O d          print digit
A40S        53          A cur        load current value
A39S        54          A incr       acc += 1
T40S        55          T cur        store current value
A38S        56          A last       load last value
S40S        57          S cur        last - cur < 0, should we break?
E48S        58          E _loop      if no, jump to loop
O34S        59          O lf         print line feed
O35S        60          O cr         print carriage return
ZS          61 _end     Z            stop program
```

“FizzBuzz” on the EDSAC / Initial Orders 1

written in a “primitive” 1949-like style
by Olve Maudal, Monday, April 20, 2015

I pretended I was a student, who had won a **single** chance to run my program on this precious computer.

The program did actually ran on the very first attempt!

“FizzBuzz” on the EDSAC / Initial Orders 1

```
T123S    31              T L_end         mark end of program
E60S     32              E L_start       jump to the beginning of program
#S       33 _FS          #               figure shift
*S       34 _LS          *               letter shift
&S       35 _LF          &               linefeed character
@S       36 _CR          @               carriage return character
P100S    37 _100         P 100           constant 100
P10S     38 _10          P 10            constant 10
P5S      39 _5           P 5             constant 5
P3S      40 _3           P 3             constant 3
P1S      41 _1           P 1             constant 1
QS       42 _'1'         Q               constant figure 1
PS       43 _'0'         P               constant figure 0
BS       44 _B           B               constant letter B
FS       45 _F           F               constant letter F
IS       46 _I           I               constant letter I
US       47 _U           U               constant letter U
ZS       48 _Z           Z               constant letter Z
PS       49 _dummy       P               used to flush and reset the accumulator
P1S      50 _cnt         P 1             counter, current number to be considered, will be increased
PS       51 _num         P               number to be printed, negative if counter is mod 3 or mod 5
PS       52 _d           P               digit to be printed
O34S     53 L_next       O _LS           output LS, prepare for printing letters
O35S     54              O _LF           output LF, linefeed
O36S     55              O _CR           output CR, carriage return
T49S     56              T _dummy        reset Acc
A50S     57              A _cnt          load Acc with _cnt
A41S     58              A _1            increase Acc
T50S     59              T _cnt          store Acc into _cnt, reset Acc
A50S     60 L_start      A _cnt          load Acc with _cnt (we know that Acc initially is 0)
U51S     61              U _num          tentatively set number to be printed
S40S     62 L_tryFizz    S _3            subtract 3
E62S     63              E L_tryFizz     loop until Acc < 0
A40S     64              A _3            add 3, restore previous value
S41S     65              S _1            subtract 1, to check if Acc was 0
E73S     66              E L_notFizz     jump if Acc was not 0, ie number was not divisable by 3
T51S     67              T _num          set _num to negative value, flag that no value should be printed
O34S     68              O _LS           prepare printing letters
O45S     69              O _F            output F
O46S     70              O _I            output I
O48S     71              O _Z            output Z
O48S     72              O _Z            output Z
T49S     73 L_notFizz    T _dummy        reset Acc
A50S     74              A _cnt          load Acc with _cnt
S39S     75 L_Buzz       S _5            subtract 5
E75S     76              E L_Buzz        loop until Acc < 0
A39S     77              A _5            add 5, restore previous value
S41S     78              S _1            subtract 1, to check if Acc was 0
E86S     79              E L_notBuzz     jump if Acc was not 0, ie number was not divisable by 5
T51S     80              T _num          set _num to negative value, flag that no value should be printed
O34S     81              O _LS           prepare printing letters
O44S     82              O _B            output B
O47S     83              O _U            output U
O48S     84              O _Z            output Z
O48S     85              O _Z            output Z
T49S     86 L_notBuzz    T _dummy        reset Acc
A51S     87              A _num          load _num to check number to be printed
G53S     88              G L_next        goto next iteration if _num is negative
O33S     89 L_printNum   O _FS           prepare for printing numbers
T49S     90              T _dummy        reset Acc
A50S     91              A _cnt          load counter
S37S     92              S _100          subtract 100, check if we should stop
G98S     93              G L_not100      jump if not 100 yet
O42S     94              O _'1'          output 1
O43S     95              O _'0'          output 0
O43S     96              O _'0'          output 0
ZS       97              Z               end the program
T49S     98 L_not100     T _dummy        reset Acc
T52S     99              T _d            reset digit
A50S    100              A _cnt          load counter
S38S    101 L_count10s   S _10           subtract 10
G109S   102              G L_print10s    goto print 10s if Acc < 0
T51S    103              T _num          store number
A52S    104              A _d            load digit
A41S    105              A _1            increase digit
T52S    106              T _d            store digit
A51S    107              A _num          load number
E101S   108              E L_count10s    loop unconditionally
T49S    109 L_print10s   T _dummy        reset Acc
A52S    110              A _d            load digit
S41S    111              S _1            decrease digit by 1
G117S   112              G L_1           if negative (digit was 0), skip printing of tens digits
A41S    113              A _1            restore digit, by increasing with 1
L512S   114              L 2^(11-2)      Acc << 11, create a printable figure
T52S    115              T _d            save printable figure
O52S    116              O _d            print figure / digit
T49S    117 L_1:         T _dummy        reset Acc
A51S    118              A _num          load number
L512S   119              L 2^(11-2)      Acc << 11, create a printable figure
T52S    120              T _d            save printable figure
O52S    121              O _d            print figure / digit
E53S    122              E L_next        unconditional jump
XS      123 L_end        X
```

```
T123S    31              T L_end         mark end of program
E60S     32              E L_start       jump to the beginning of program
#S       33 _FS          #               figure shift
*S       34 _LS          *               letter shift
&S       35 _LF          &               linefeed character
@S       36 _CR          @               carriage return character
P100S    37 _100         P 100           constant 100
P10S     38 _10          P 10            constant 10
P5S      39 _5           P 5             constant 5
P3S      40 _3           P 3             constant 3
P1S      41 _1           P 1             constant 1
QS       42 _'1'         Q               constant figure 1
PS       43 _'0'         P               constant figure 0
BS       44 _B           B               constant letter B
FS       45 _F           F               constant letter F
IS       46 _I           I               constant letter I
US       47 _U           U               constant letter U
ZS       48 _Z           Z               constant letter Z
PS       49 _dummy       P               used to flush and reset the accumulator
P1S      50 _cnt         P 1             counter, current number to be considered, will be increased
PS       51 _num         P               number to be printed, negative if counter is mod 3 or mod 5
PS       52 _d           P               digit to be printed
```

“FizzBuzz” on the EDSAC / Initial Orders 1

```
T123S   31              T L_end        mark end of program
E60S    32              E L_start      jump to the beginning of program
#S      33 _FS          #              figure shift
*S      34 _LS          *              letter shift
&S      35 _LF          &              linefeed character
@S      36 _CR          @              carriage return character
P100S   37 _100         P 100          constant 100
P10S    38 _10          P 10           constant 10
P5S     39 _5           P 5            constant 5
P3S     40 _3           P 3            constant 3
P1S     41 _1           P 1            constant 1
QS      42 _'1'         Q              constant figure 1
PS      43 _'0'         P              constant figure 0
BS      44 _B           B              constant letter B
FS      45 _F           F              constant letter F
IS      46 _I           I              constant letter I
US      47 _U           U              constant letter U
ZS      48 _Z           Z              constant letter Z
PS      49 _dummy       P              used to flush and reset the accumulator
P1S     50 _cnt         P 1            counter, current number to be considered, will be increased
PS      51 _num         P              number to be printed, negative if counter is mod 3 or mod 5
PS      52 _d           P              digit to be printed
O34S    53 L_next       O _LS          output LS, prepare for printing letters
O35S    54              O _LF          output LF, linefeed
O36S    55              O _CR          output CR, carriage return
T49S    56              T _dummy       reset Acc
A50S    57              A _cnt         load Acc with _cnt
A41S    58              A _1           increase Acc
T50S    59              T _cnt         store Acc into _cnt, reset Acc
A50S    60 L_start      A _cnt         load Acc with _cnt (we know that Acc initially is 0)
U51S    61              U _num         tentatively set number to be printed
S40S    62 L_tryFizz    S _3           subtract 3
E62S    63              E L_tryFizz    loop until Acc < 0
A40S    64              A _3           add 3, restore previous value
S41S    65              S _1           subtract 1, to check if Acc was 0
E73S    66              E L_notFizz    jump if Acc was not 0, ie number was not divisable by 3
T51S    67              T _num         set _num to negative value, flag that no value should be printed
O34S    68              O _LS          prepare printing letters
O45S    69              O _F           output F
O46S    70              O _I           output I
O48S    71              O _Z           output Z
O48S    72              O _Z           output Z
T49S    73 L_notFizz    T _dummy       reset Acc
A50S    74              A _cnt         load Acc with _cnt
S39S    75 L_Buzz       S _5           subtract 5
E75S    76              E L_Buzz       loop until Acc < 0
A39S    77              A _5           add 5, restore previous value
S41S    78              S _1           subtract 1, to check if Acc was 0
E86S    79              E L_notBuzz    jump if Acc was not 0, ie number was not divisable by 5
T51S    80              T _num         set _num to negative value, flag that no value should be printed
O34S    81              O _LS          prepare printing letters
O44S    82              O _B           output B
O47S    83              O _U           output U
O48S    84              O _Z           output Z
O48S    85              O _Z           output Z
T49S    86 L_notBuzz    T _dummy       reset Acc
A51S    87              A _num         load _num to check number to be printed
G53S    88              G L_next       goto next iteration if _num is negative
O33S    89 L_printNum   O _FS          prepare for printing numbers
T49S    90              T _dummy       reset Acc
A50S    91              A _cnt         load counter
S37S    92              S _100         subtract 100, check if we should stop
G98S    93              G L_not100     jump if not 100 yet
O42S    94              O _'1'         output 1
O43S    95              O _'0'         output 0
O43S    96              O _'0'         output 0
ZS      97              Z              end the program
T49S    98 L_not100     T _dummy       reset Acc
T52S    99              T _d           reset digit
A50S   100              A _cnt         load counter
S38S   101 L_count10s   S _10          subtract 10
G109S  102              G L_print10s   goto print 10s if Acc < 0
T51S   103              T _num         store number
A52S   104              A _d           load digit
A41S   105              A _1           increase digit
T52S   106              T _d           store digit
A51S   107              A _num         load number
E101S  108              E L_count10s   loop unconditionally
T49S   109 L_print10s   T _dummy       reset Acc
A52S   110              A _d           load digit
S41S   111              S _1           decrease digit by 1
G117S  112              G L_1          if negative (digit was 0), skip printing of tens digits
A41S   113              A _1           restore digit, by increasing with 1
L512S  114              L 2^(11-2)     Acc << 11, create a printable figure
T52S   115              T _d           save printable figure
O52S   116              O _d           print figure / digit
T49S   117 L_1:         T _dummy       reset Acc
A51S   118              A _num         load number
L512S  119              L 2^(11-2)     Acc << 11, create a printable figure
T52S   120              T _d           save printable figure
O52S   121              O _d           print figure / digit
E53S   122              E L_next       unconditional jump
XS     123 L_end        X
```

```
O34S    53 L_next       O _LS          output LS, prepare for printing letters
O35S    54              O _LF          output LF, linefeed
O36S    55              O _CR          output CR, carriage return
T49S    56              T _dummy       reset Acc
A50S    57              A _cnt         load Acc with _cnt
A41S    58              A _1           increase Acc
T50S    59              T _cnt         store Acc into _cnt, reset Acc
A50S    60 L_start      A _cnt         load Acc with _cnt (we know that Acc initially is 0)
U51S    61              U _num         tentatively set number to be printed
S40S    62 L_tryFizz    S _3           subtract 3
E62S    63              E L_tryFizz    loop until Acc < 0
A40S    64              A _3           add 3, restore previous value
S41S    65              S _1           subtract 1, to check if Acc was 0
E73S    66              E L_notFizz    jump if Acc was not 0, ie number was not divisable by 3
T51S    67              T _num         set _num to negative value, flag that no value should be printed
O34S    68              O _LS          prepare printing letters
O45S    69              O _F           output F
O46S    70              O _I           output I
O48S    71              O _Z           output Z
O48S    72              O _Z           output Z
```

“FizzBuzz” on the EDSAC / Initial Orders 1

```
T123S   31              T L_end         mark end of program
E60S    32              E L_start       jump to the beginning of program
#S      33 _FS          #               figure shift
*S      34 _LS          *               letter shift
&S      35 _LF          &               linefeed character
@S      36 _CR          @               carriage return character
P100S   37 _100         P 100           constant 100
P10S    38 _10          P 10            constant 10
P5S     39 _5           P 5             constant 5
P3S     40 _3           P 3             constant 3
P1S     41 _1           P 1             constant 1
QS      42 _'1'         Q               constant figure 1
PS      43 _'0'         P               constant figure 0
BS      44 _B           B               constant letter B
FS      45 _F           F               constant letter F
IS      46 _I           I               constant letter I
US      47 _U           U               constant letter U
ZS      48 _Z           Z               constant letter Z
PS      49 _dummy       P               used to flush and reset the accumulator
P1S     50 _cnt         P 1             counter, current number to be considered, will be increased
PS      51 _num         P               number to be printed, negative if counter is mod 3 or mod 5
PS      52 _d           P               digit to be printed
O34S    53 L_next       O _LS           output LS, prepare for printing letters
O35S    54              O _LF           output LF, linefeed
O36S    55              O _CR           output CR, carriage return
T49S    56              T _dummy        reset Acc
A50S    57              A _cnt          load Acc with _cnt
A41S    58              A _1            increase Acc
T50S    59              T _cnt          store Acc into _cnt, reset Acc
A50S    60 L_start      A _cnt          load Acc with _cnt (we know that Acc initially is 0)
U51S    61              U _num          tentatively set number to be printed
S40S    62 L_tryFizz    S _3            subtract 3
E62S    63              E L_tryFizz     loop until Acc < 0
A40S    64              A _3            add 3, restore previous value
S41S    65              S _1            subtract 1, to check if Acc was 0
E73S    66              E L_notFizz     jump if Acc was not 0, ie number was not divisable by 3
T51S    67              T _num          set _num to negative value, flag that no value should be printed
O34S    68              O _LS           prepare printing letters
O45S    69              O _F            output F
O46S    70              O _I            output I
O48S    71              O _Z            output Z
O48S    72              O _Z            output Z
T49S    73 L_notFizz    T _dummy        reset Acc
A50S    74              A _cnt          load Acc with _cnt
S39S    75 L_Buzz       S _5            subtract 5
E75S    76              E L_Buzz        loop until Acc < 0
A39S    77              A _5            add 5, restore previous value
S41S    78              S _1            subtract 1, to check if Acc was 0
E86S    79              E L_notBuzz     jump if Acc was not 0, ie number was not divisable by 5
T51S    80              T _num          set _num to negative value, flag that no value should be printed
O34S    81              O _LS           prepare printing letters
O44S    82              O _B            output B
O47S    83              O _U            output U
O48S    84              O _Z            output Z
O48S    85              O _Z            output Z
T49S    86 L_notBuzz    T _dummy        reset Acc
A51S    87              A _num          load _num to check number to be printed
G53S    88              G L_next        goto next iteration if _num is negative
O33S    89 L_printNum   O _FS           prepare for printing numbers
T49S    90              T _dummy        reset Acc
A50S    91              A _cnt          load counter
S37S    92              S _100          subtract 100, check if we should stop
G98S    93              G L_not100      jump if not 100 yet
O42S    94              O _'1'          output 1
O43S    95              O _'0'          output 0
O43S    96              O _'0'          output 0
ZS      97              Z               end the program
T49S    98 L_not100     T _dummy        reset Acc
T52S    99              T _d            reset digit
A50S   100              A _cnt          load counter
S38S   101 L_count10s   S _10           subtract 10
G109S  102              G L_print10s    goto print 10s if Acc < 0
T51S   103              T _num          store number
A52S   104              A _d            load digit
A41S   105              A _1            increase digit
T52S   106              T _d            store digit
A51S   107              A _num          load number
E101S  108              E L_count10s    loop unconditionally
T49S   109 L_print10s   T _dummy        reset Acc
A52S   110              A _d            load digit
S41S   111              S _1            decrease digit by 1
G117S  112              G L_1           if negative (digit was 0), skip printing of tens digits
A41S   113              A _1            restore digit, by increasing with 1
L512S  114              L 2^(11-2)      Acc << 11, create a printable figure
T52S   115              T _d            save printable figure
O52S   116              O _d            print figure / digit
T49S   117 L_1:         T _dummy        reset Acc
A51S   118              A _num          load number
L512S  119              L 2^(11-2)      Acc << 11, create a printable figure
T52S   120              T _d            save printable figure
O52S   121              O _d            print figure / digit
E53S   122              E L_next        unconditional jump
XS     123 L_end        X
```

```
T49S    73 L_notFizz    T _dummy        reset Acc
A50S    74              A _cnt          load Acc with _cnt
S39S    75 L_Buzz       S _5            subtract 5
E75S    76              E L_Buzz        loop until Acc < 0
A39S    77              A _5            add 5, restore previous value
S41S    78              S _1            subtract 1, to check if Acc was 0
E86S    79              E L_notBuzz     jump if Acc was not 0, ie number was not divisable by 5
T51S    80              T _num          set _num to negative value, flag that no value should be printed
O34S    81              O _LS           prepare printing letters
O44S    82              O _B            output B
O47S    83              O _U            output U
O48S    84              O _Z            output Z
O48S    85              O _Z            output Z
T49S    86 L_notBuzz    T _dummy        reset Acc
A51S    87              A _num          load _num to check number to be printed
G53S    88              G L_next        goto next iteration if _num is negative
O33S    89 L_printNum   O _FS           prepare for printing numbers
T49S    90              T _dummy        reset Acc
A50S    91              A _cnt          load counter
S37S    92              S _100          subtract 100, check if we should stop
G98S    93              G L_not100      jump if not 100 yet
O42S    94              O _'1'          output 1
O43S    95              O _'0'          output 0
O43S    96              O _'0'          output 0
ZS      97              Z               end the program
```

“FizzBuzz” on the EDSAC / Initial Orders 1

```
T123S    31              T L_end        mark end of program
E60S     32              E L_start      jump to the beginning of program
#S       33 _FS          #              figure shift
*S       34 _LS          *              letter shift
&S       35 _LF          &              linefeed character
@S       36 _CR          @              carriage return character
P100S    37 _100         P 100          constant 100
P10S     38 _10          P 10           constant 10
P5S      39 _5           P 5            constant 5
P3S      40 _3           P 3            constant 3
P1S      41 _1           P 1            constant 1
QS       42 _'1'         Q              constant figure 1
PS       43 _'0'         P              constant figure 0
BS       44 _B           B              constant letter B
FS       45 _F           F              constant letter F
IS       46 _I           I              constant letter I
US       47 _U           U              constant letter U
ZS       48 _Z           Z              constant letter Z
PS       49 _dummy       P              used to flush and reset the accumulator
P1S      50 _cnt         P 1            counter, current number to be considered, will be increased
PS       51 _num         P              number to be printed, negative if counter is mod 3 or mod 5
PS       52 _d           P              digit to be printed
O34S     53 L_next       O _LS          output LS, prepare for printing letters
O35S     54              O _LF          output LF, linefeed
O36S     55              O _CR          output CR, carriage return
T49S     56              T _dummy       reset Acc
A50S     57              A _cnt         load Acc with _cnt
A41S     58              A _1           increase Acc
T50S     59              T _cnt         store Acc into _cnt, reset Acc
A50S     60 L_start      A _cnt         load Acc with _cnt (we know that Acc initially is 0)
U51S     61              U _num         tentatively set number to be printed
S40S     62 L_tryFizz    S _3           subtract 3
E62S     63              E L_tryFizz    loop until Acc < 0
A40S     64              A _3           add 3, restore previous value
S41S     65              S _1           subtract 1, to check if Acc was 0
E73S     66              E L_notFizz    jump if Acc was not 0, ie number was not divisable by 3
T51S     67              T _num         set _num to negative value, flag that no value should be printed
O34S     68              O _LS          prepare printing letters
O45S     69              O _F           output F
O46S     70              O _I           output I
O48S     71              O _Z           output Z
O48S     72              O _Z           output Z
T49S     73 L_notFizz    T _dummy       reset Acc
A50S     74              A _cnt         load Acc with _cnt
S39S     75 L_Buzz       S _5           subtract 5
E75S     76              E L_Buzz       loop until Acc < 0
A39S     77              A _5           add 5, restore previous value
S41S     78              S _1           subtract 1, to check if Acc was 0
E86S     79              E L_notBuzz    jump if Acc was not 0, ie number was not divisable by 5
T51S     80              T _num         set _num to negative value, flag that no value should be printed
O34S     81              O _LS          prepare printing letters
O44S     82              O _B           output B
O47S     83              O _U           output U
O48S     84              O _Z           output Z
O48S     85              O _Z           output Z
T49S     86 L_notBuzz    T _dummy       reset Acc
A51S     87              A _num         load _num to check number to be printed
G53S     88              G L_next       goto next iteration if _num is negative
O33S     89 L_printNum   O _FS          prepare for printing numbers
T49S     90              T _dummy       reset Acc
A50S     91              A _cnt         load counter
S37S     92              S _100         subtract 100, check if we should stop
G98S     93              G L_not100     jump if not 100 yet
O42S     94              O _'1'         output 1
O43S     95              O _'0'         output 0
O43S     96              O _'0'         output 0
ZS       97              Z              end the program
T49S     98 L_not100     T _dummy       reset Acc
T52S     99              T _d           reset digit
A50S    100              A _cnt         load counter
S38S    101 L_count10s   S _10          subtract 10
G109S   102              G L_print10s   goto print 10s if Acc < 0
T51S    103              T _num         store number
A52S    104              A _d           load digit
A41S    105              A _1           increase digit
T52S    106              T _d           store digit
A51S    107              A _num         load number
E101S   108              E L_count10s   loop unconditionally
T49S    109 L_print10s   T _dummy       reset Acc
A52S    110              A _d           load digit
S41S    111              S _1           decrease digit by 1
G117S   112              G L_1          if negative (digit was 0), skip printing of tens digits
A41S    113              A _1           restore digit, by increasing with 1
L512S   114              L 2^(11-2)     Acc << 11, create a printable figure
T52S    115              T _d           save printable figure
O52S    116              O _d           print figure / digit
T49S    117 L_1:         T _dummy       reset Acc
A51S    118              A _num         load number
L512S   119              L 2^(11-2)     Acc << 11, create a printable figure
T52S    120              T _d           save printable figure
O52S    121              O _d           print figure / digit
E53S    122              E L_next       unconditional jump
XS      123 L_end        X
```

```
T49S     98 L_not100     T _dummy       reset Acc
T52S     99              T _d           reset digit
A50S    100              A _cnt         load counter
S38S    101 L_count10s   S _10          subtract 10
G109S   102              G L_print10s   goto print 10s if Acc < 0
T51S    103              T _num         store number
A52S    104              A _d           load digit
A41S    105              A _1           increase digit
T52S    106              T _d           store digit
A51S    107              A _num         load number
E101S   108              E L_count10s   loop unconditionally
T49S    109 L_print10s   T _dummy       reset Acc
A52S    110              A _d           load digit
S41S    111              S _1           decrease digit by 1
G117S   112              G L_1          if negative (digit was 0), skip printing of tens digits
A41S    113              A _1           restore digit, by increasing with 1
L512S   114              L 2^(11-2)     Acc << 11, create a printable figure
T52S    115              T _d           save printable figure
O52S    116              O _d           print figure / digit
T49S    117 L_1:         T _dummy       reset Acc
A51S    118              A _num         load number
L512S   119              L 2^(11-2)     Acc << 11, create a printable figure
T52S    120              T _d           save printable figure
O52S    121              O _d           print figure / digit
E53S    122              E L_next       unconditional jump
XS      123 L_end        X
```

“FizzBuzz” on the EDSAC / Initial Orders 1

```
T123SE60S#S*S&S@SP100SP10SP5SP3SP1SQSPSBSFSISU
SZSPSP1SPSPSO34SO35SO36ST49SA50SA41ST50SA50SU5
1SS40SE62SA40SS41SE73ST51SO34SO45SO46SO48SO48S
T49SA50SS39SE75SA39SS41SE86ST51SO34SO44SO47SO4
8SO48ST49SA51SG53SO33ST49SA50SS37SG98SO42SO43S
O43SZST49ST52SA50SS38SG109ST51SA52SA41ST52SA51
SE101ST49SA52SS41SG117SA41SL512ST52SO52ST49SA5
1SL512ST52SO52SE53SXS
```

Try this program on NISHIO Hirokazu’s EDSAC Simulator
http://nhiro.org/learn_language/repos/EDSAC-on-browser/index.html

“FizzBuzz” on the EDSAC / Initial Orders 1

```
T123SE60S#S*S&S@SP100SP10SP5SP3SP1SQSPSBSFSISU
SZSPSP1SPSPSO34SO35SO36ST49SA50SA41ST50SA50SU5
1SS40SE62SA40SS41SE73ST51SO34SO45SO46SO48SO48S
T49SA50SS39SE75SA39SS41SE86ST51SO34SO44SO47SO4
8SO48ST49SA51SG53SO33ST49SA50SS37SG98SO42SO43S
O43SZST49ST52SA50SS38SG109ST51SA52SA41ST52SA51
SE101ST49SA52SS41SG117SA41SL512ST52SO52ST49SA5
1SL512ST52SO52SE53SXS
```

Try this program on NISHIO Hirokazu’s EDSAC Simulator
http://nhiro.org/learn_language/repos/EDSAC-on-browser/index.html

There is a small bug in the program. Did you notice?

“FizzBuzz” on the EDSAC / Initial Orders 1

```
T123SE60S#S*S&S@SP100SP10SP5SP3SP1SQSPSBSFSISU
SZSPSP1SPSPSO34SO35SO36ST49SA50SA41ST50SA50SU5
1SS40SE62SA40SS41SE73ST51SO34SO45SO46SO48SO48S
T49SA50SS39SE75SA39SS41SE86ST51SO34SO44SO47SO4
8SO48ST49SA51SG53SO33ST49SA50SS37SG98SO42SO43S
O43SZST49ST52SA50SS38SG109ST51SA52SA41ST52SA51
SE101ST49SA52SS41SG117SA41SL512ST52SO52ST49SA5
1SL512ST52SO52SE53SXS
```

Try this program on NISHIO Hirokazu’s EDSAC Simulator
http://nhiro.org/learn_language/repos/EDSAC-on-browser/index.html

“FizzBuzz” on the EDSAC / Initial Orders 1

```
T123SE60S#S*S&S@SP100SP10SP5SP3SP1SQSPSBSFSISU
SZSPSP1SPSPSO34SO35SO36ST49SA50SA41ST50SA50SU5
1SS40SE62SA40SS41SE73ST51SO34SO45SO46SO48SO48S
T49SA50SS39SE75SA39SS41SE86ST51SO34SO44SO47SO4
8SO48ST49SA51SG53SO33ST49SA50SS37SG98SO42SO43S
O43SZST49ST52SA50SS38SG109ST51SA52SA41ST52SA51
SE101ST49SA52SS41SG117SA41SL512ST52SO52ST49SA5
1SL512ST52SO52SE53SXS
```

Here is a quick and dirty fix!

Try this program on NISHIO Hirokazu’s EDSAC Simulator
http://nhiro.org/learn_language/repos/EDSAC-on-browser/index.html

“FizzBuzz” on the EDSAC / Initial Orders 1

```
T123SE60S#S*S&S@SP100SP10SP5SP3SP1SQSPSBSFSISU
SZSPSP1SPSPS034S035S036ST49SA50SA41ST50SA50SU5
1SS40SE62SA40SS41SE73ST51S034S045S046S048S048S
T49SA50SS39SE75SA39SS41SE86ST51S034S044S047S04
8S048ST49SA51SG53S033ST49SA50SS37SA41SG98SZS04
3S043ST49ST52SA50SS38SG109ST51SA52SA41ST52SA51
SE101ST49SA52SS41SG117SA41SL512ST52S052ST49SA5
1SL512ST52S052SE53SXS
```

Try this program on NISHIO Hirokazu’s EDSAC Simulator
http://nhiro.org/learn_language/repos/EDSAC-on-browser/index.html

“FizzBuzz” on the EDSAC / Initial Orders 1

```
T123SE60S#S*S&S@SP100SP10SP5SP3SP1SQSPSBSFSISU
SZSPSP1SPSPSO34SO35SO36ST49SA50SA41ST50SA50SU5
1SS40SE62SA40SS41SE73ST51SO34SO45SO46SO48SO48S
T49SA50SS39SE75SA39SS41SE86ST51SO34SO44SO47SO4
8SO48ST49SA51SG53SO33ST49SA50SS37SA41SG98SZSO4
3SO43ST49ST52SA50SS38SG109ST51SA52SA41ST52SA51
SE101ST49SA52SS41SG117SA41SL512ST52SO52ST49SA5
1SL512ST52SO52SE53SXS
```

Try this program on NISHIO Hirokazu’s EDSAC Simulator
http://nhiro.org/learn_language/repos/EDSAC-on-browser/index.html

“FizzBuzz” on the EDSAC / Initial Orders 1

```
T123SE60S#S*S&S@SP100SP10SP5SP3SP1SQSPSBSFSISU
SZSPSP1SPSPSO34SO35SO36ST49SA50SA41ST50SA50SU5
1SS40SE62SA40SS41SE73ST51SO34SO45SO46SO48SO48S
T49SA50SS39SE75SA39SS41SE86ST51SO34SO44SO47SO4
8SO48ST49SA51SG53SO33ST49SA50SS37SA41SG98SZSO4
3SO43ST49ST52SA50SS38SG109ST51SA52SA41ST52SA51
SE101ST49SA52SS41SG117SA41SL512ST52SO52ST49SA5
1SL512ST52SO52SE53SXS
```

Enjoy!

Try this program on NISHIO Hirokazu’s EDSAC Simulator
http://nhiro.org/learn_language/repos/EDSAC-on-browser/index.html

Speedcoding, John Backus, 1953 on the IBM 701

IBM 701 operator's console

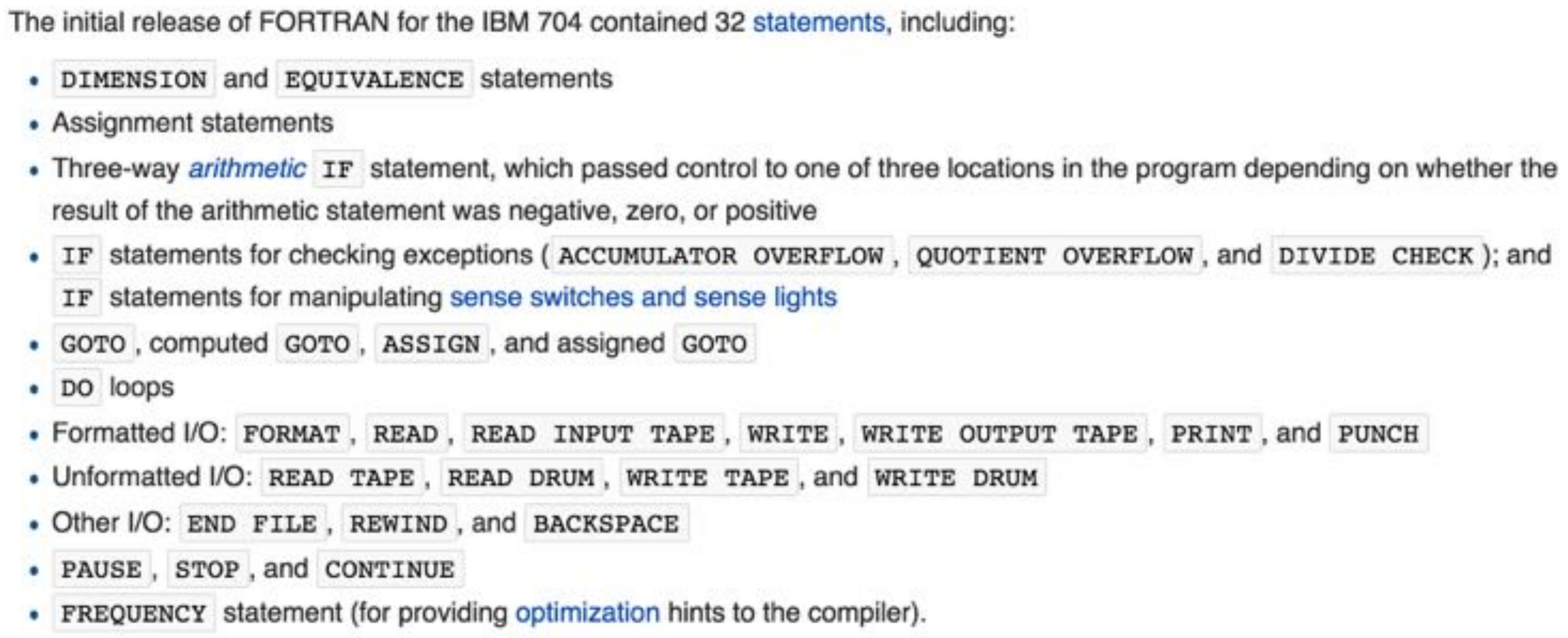
IBM 701 processor frame

http://en.wikipedia.org/wiki/IBM_701

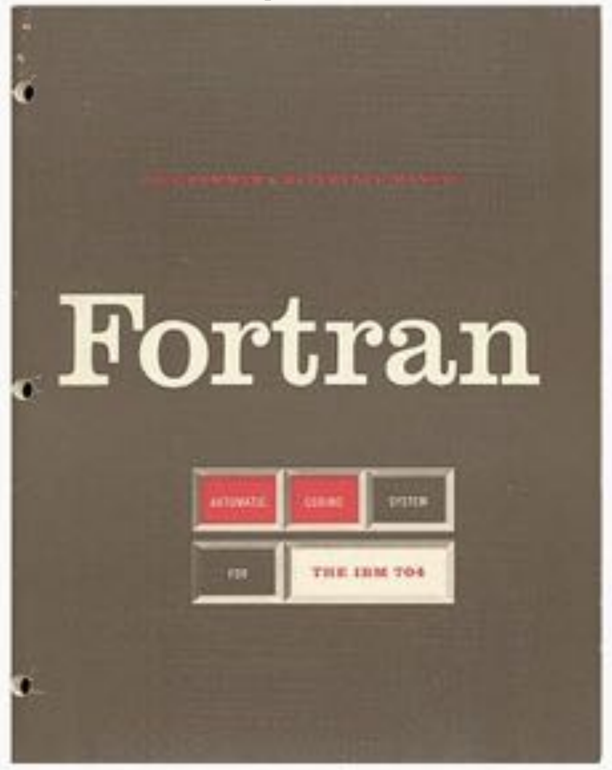
Backus later did work on the IBM 704

http://en.wikipedia.org/wiki/IBM_704

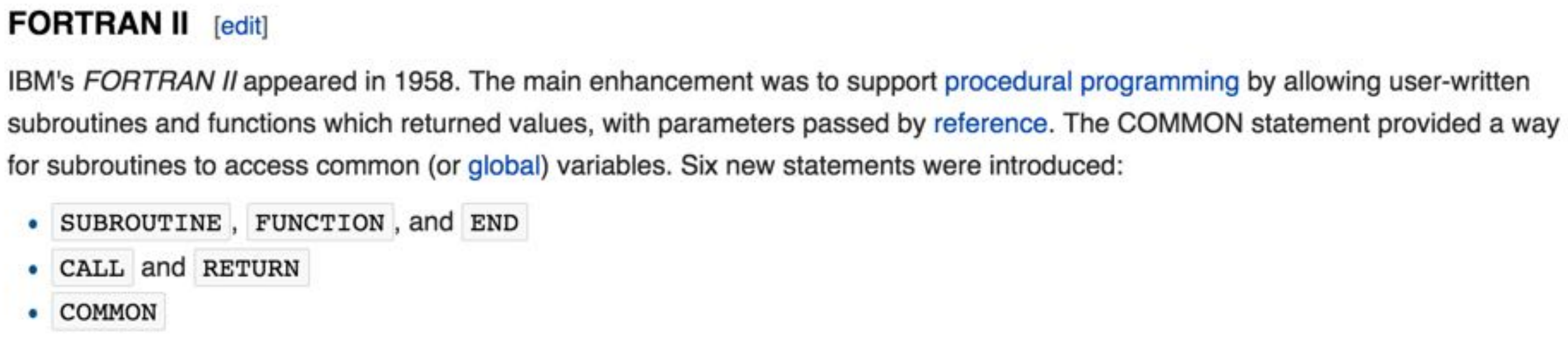
Fortran (appeared 1957, designed by John Backus)


The initial release of FORTRAN for the IBM 704 contained 32 statements, including:

- `DIMENSION` and `EQUIVALENCE` statements
- Assignment statements
- Three-way *arithmetic* `IF` statement, which passed control to one of three locations in the program depending on whether the result of the arithmetic statement was negative, zero, or positive
- `IF` statements for checking exceptions (`ACCUMULATOR OVERFLOW`, `QUOTIENT OVERFLOW`, and `DIVIDE CHECK`); and `IF` statements for manipulating sense switches and sense lights
- `GOTO`, computed `GOTO`, `ASSIGN`, and assigned `GOTO`
- `DO` loops
- Formatted I/O: `FORMAT`, `READ`, `READ INPUT TAPE`, `WRITE`, `WRITE OUTPUT TAPE`, `PRINT`, and `PUNCH`
- Unformatted I/O: `READ TAPE`, `READ DRUM`, `WRITE TAPE`, and `WRITE DRUM`
- Other I/O: `END FILE`, `REWIND`, and `BACKSPACE`
- `PAUSE`, `STOP`, and `CONTINUE`
- `FREQUENCY` statement (for providing optimization hints to the compiler).

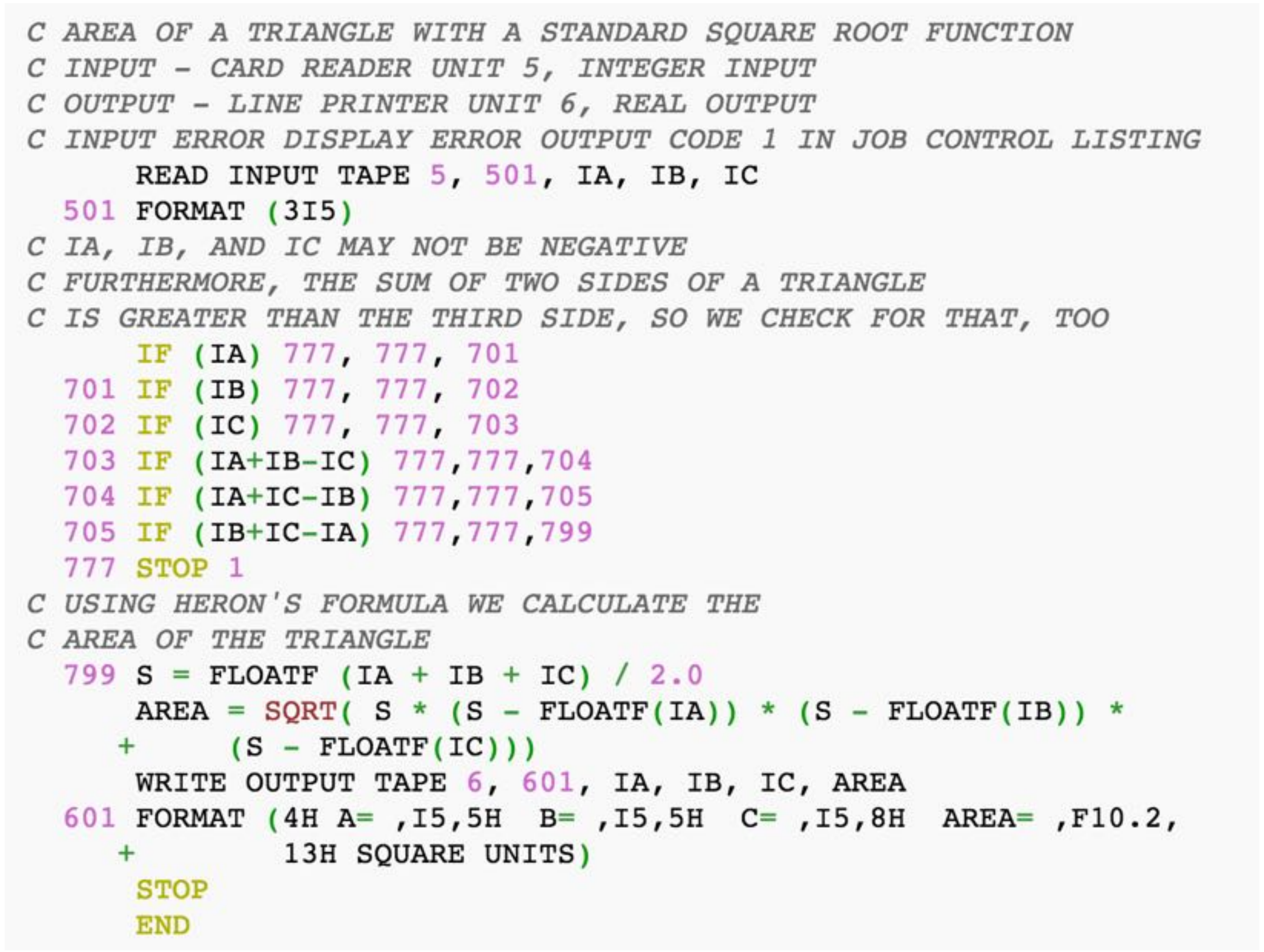


The Fortran Automatic Coding System for the IBM 704 (15 October 1956), the first Programmer's Reference Manual for Fortran


FORTRAN II [edit]

IBM's *FORTRAN II* appeared in 1958. The main enhancement was to support procedural programming by allowing user-written subroutines and functions which returned values, with parameters passed by reference. The COMMON statement provided a way for subroutines to access common (or global) variables. Six new statements were introduced:

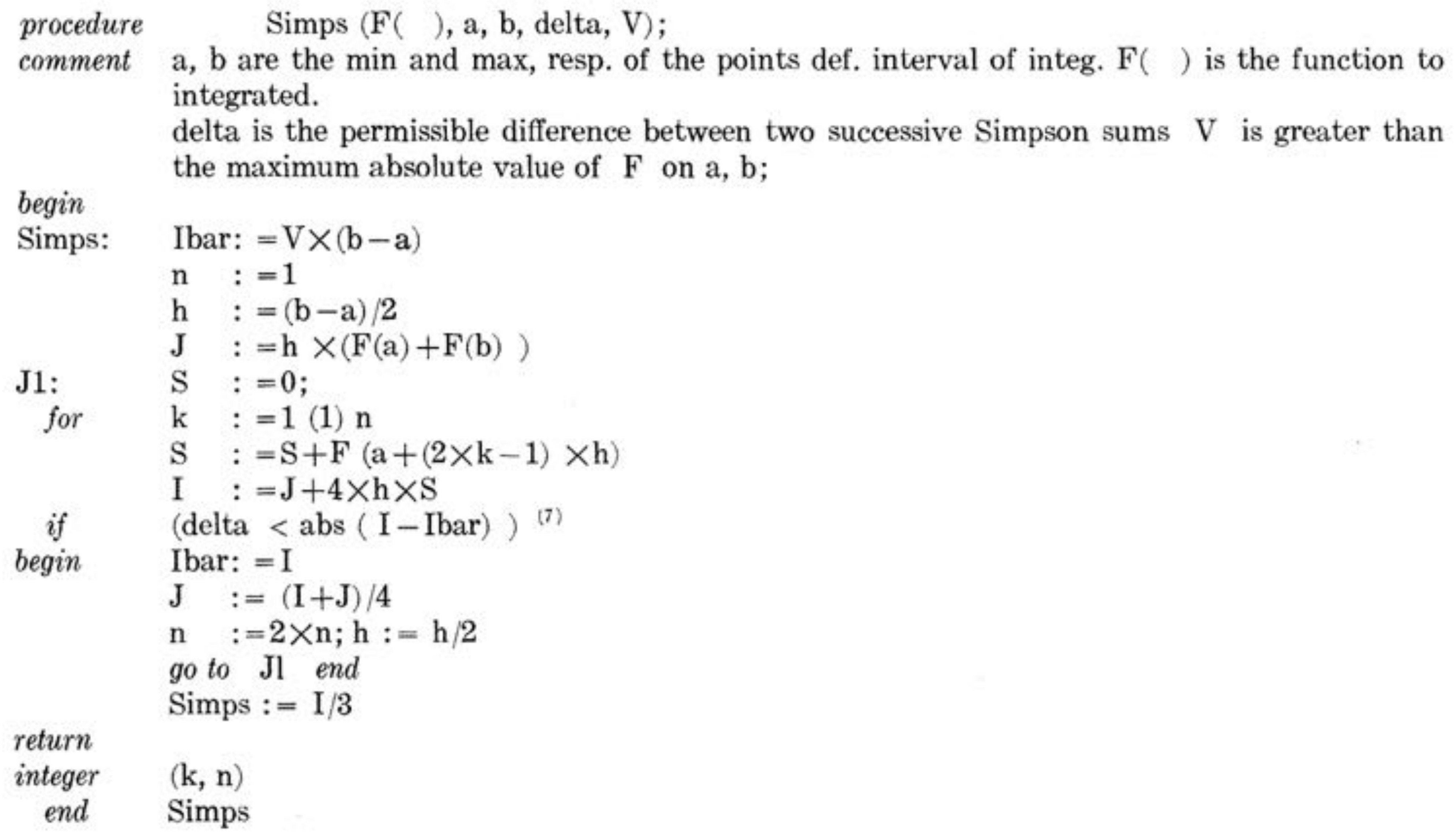
- `SUBROUTINE`, `FUNCTION`, and `END`
- `CALL` and `RETURN`
- `COMMON`



```
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

IAL (aka Algol 58) (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  Jl   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/

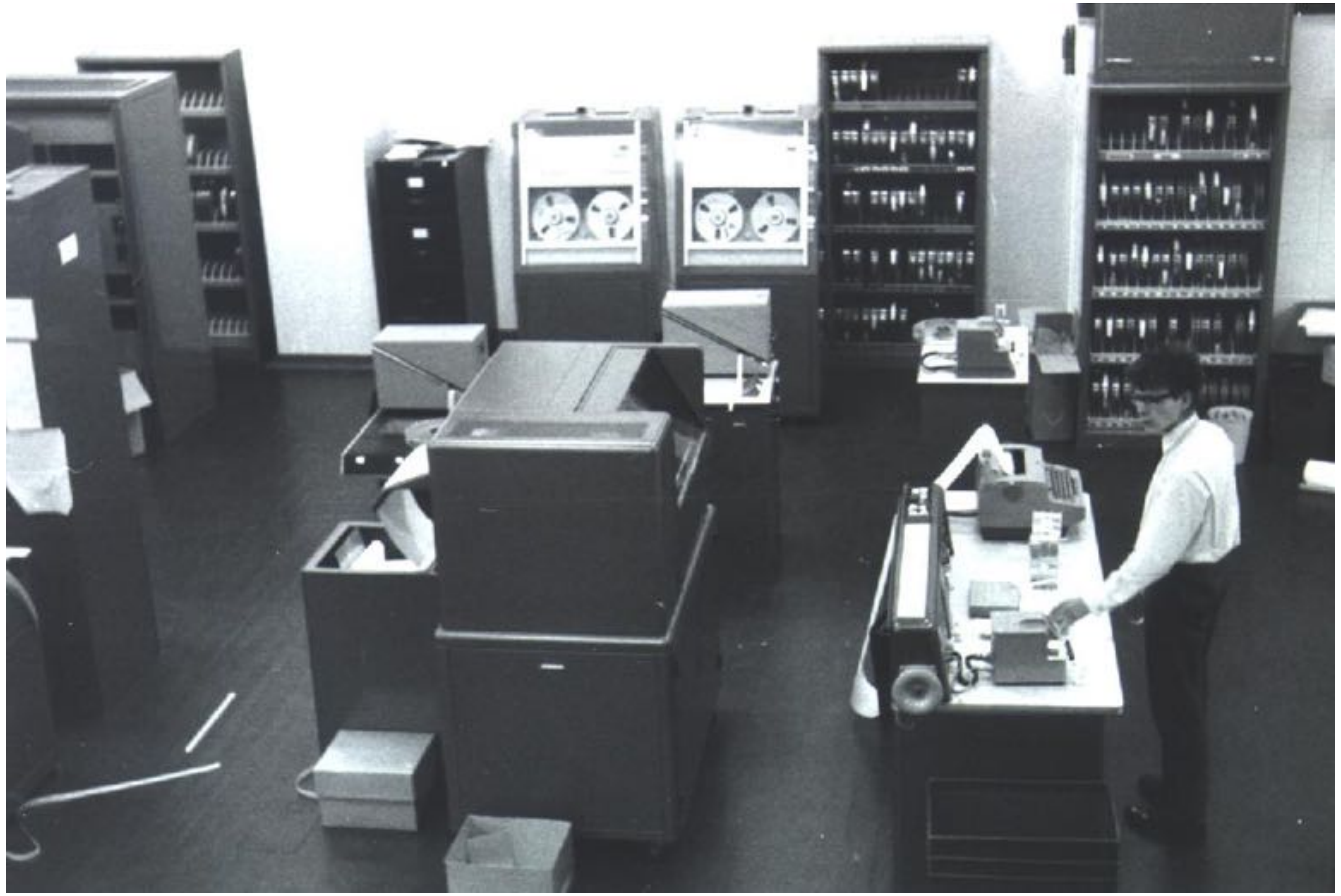
Cambridge

http://i.telegraph.co.uk/multimedia/archive/02012/oxbridge-1_2012609b.jpg

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, but….

ALGOL 60 was just *“a language, not a programming system”*

```
procedure Absmax(a) Size:(n, m) Result:(y) Subscripts:(i, k);
    value n, m; array a; integer n, m, i, k; real y;
comment The absolute greatest element of the matrix a, of size n by m,
    is transferred to y, and the subscripts of this element to i and k;
begin
    integer p, q;
    y := 0; i := k := 1;
    for p := 1 step 1 until n do
        for q := 1 step 1 until m do
            if abs(a[p, q]) > y then
                begin y := abs(a[p, q]);
                    i := p; k := q
                end
end Absmax
```

Algol 60 was criticized as not enabling efficient compilation, call by name being cited as a main cause. A second area of concern was the side effects of procedures necessitating a strict left-to-right rule for the evaluation of expressions.

From David Hartley's article "CPL: Failed Venture or Noble Ancestor?" (2013)
wikipedia on Algol 60

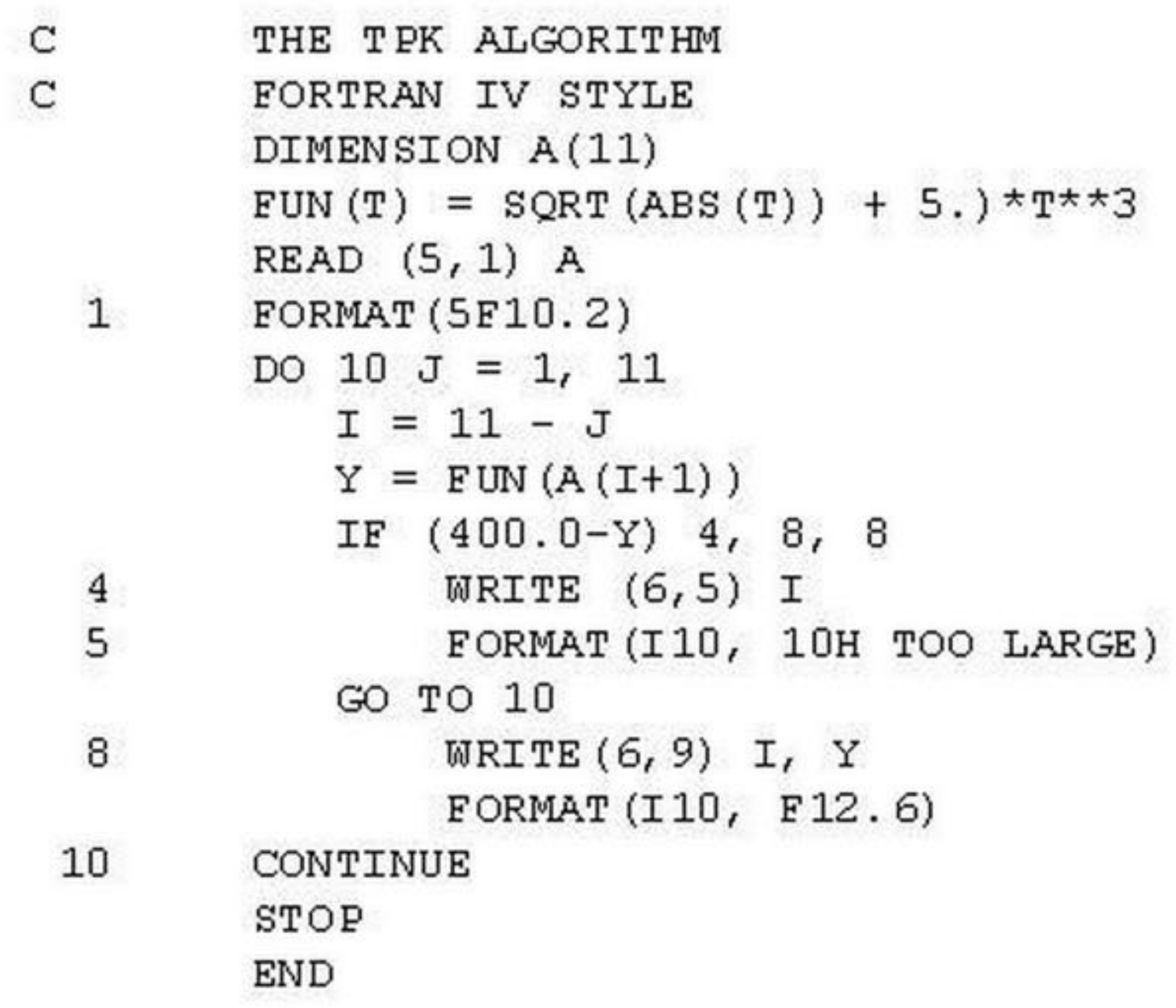
ALGOL 60 was just *"a language, not a programming system"*

```
procedure Absmax(a) Size:(n, m) Result:(y) Subscripts:(i, k);
    value n, m; array a; integer n, m, i, k; real y;
comment The absolute greatest element of the matrix a, of size n by m,
    is transferred to y, and the subscripts of this element to i and k;
begin
    integer p, q;
    y := 0; i := k := 1;
    for p := 1 step 1 until n do
        for q := 1 step 1 until m do
            if abs(a[p, q]) > y then
                begin y := abs(a[p, q]);
                    i := p; k := q
                end
end Absmax
```

Algol 60 was criticized as not enabling efficient compilation, call by name being cited as a main cause. A second area of concern was the side effects of procedures necessitating a strict left-to-right rule for the evaluation of expressions.

From David Hartley's article "CPL: Failed Venture or Noble Ancestor?" (2013)
wikipedia on Algol 60

Fortran IV was too tied up to IBM 709/7090

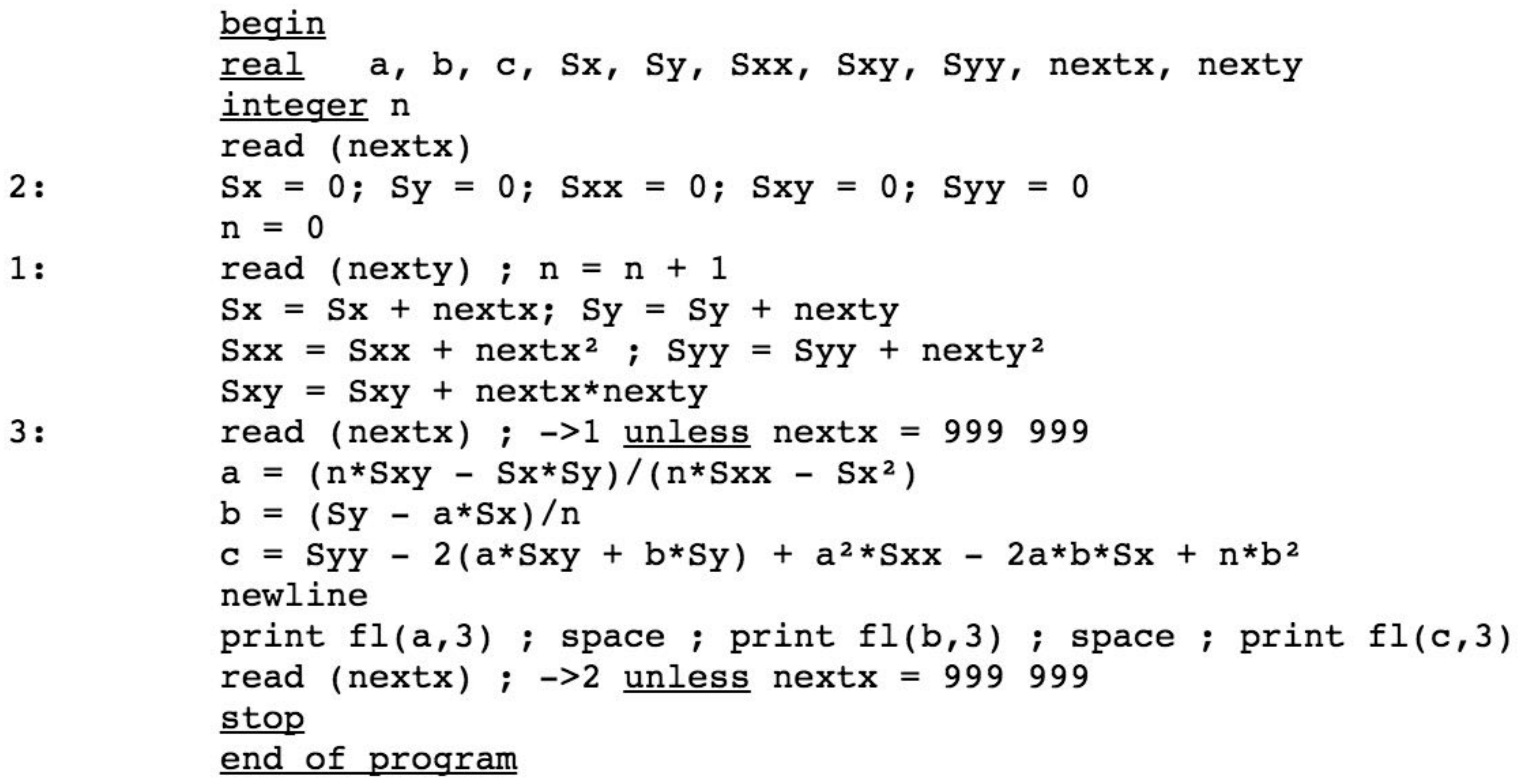


```
C       THE TPK ALGORITHM
C       FORTRAN IV STYLE
        DIMENSION A(11)
        FUN(T) = SQRT(ABS(T)) + 5.)*T**3
        READ (5,1) A
  1     FORMAT(5F10.2)
        DO 10 J = 1, 11
           I = 11 - J
           Y = FUN(A(I+1))
           IF (400.0-Y) 4, 8, 8
  4           WRITE (6,5) I
  5           FORMAT(I10, 10H TOO LARGE)
           GO TO 10
  8           WRITE(6,9) I, Y
              FORMAT(I10, F12.6)
 10     CONTINUE
        STOP
        END
```

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

http://www.fortran.bcs.org/2005/fortran/img10.jpg

Example of 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
```

“the use of compiler-compiler technology frightened us”

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

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

CPL

~~Cambridge Programming Language~~

CPL

~~Cambridge Programming Language~~
Cambridge Plus London

CPL

~~Cambridge Programming Language~~

~~Cambridge Plus London~~

CPL

~~Cambridge Programming Language~~

~~Cambridge Plus London~~

Combined Programming Language

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, as occurs in ALGOL 60"

"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)

"anything not explicity allowed should be forbidden ... nothing should be left undefined, as occurs in ALGOL 60"

"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)

Advanced were made in understanding the evaluation of expressions so as to recognize not just the value of data but also its location. Taking terminology related to the assignment statement, we developed the concept of left-hand and right-hand values ... this enabled an assignment statement to have the generalized form

```
  <expression> := <expression>
```

the first being evaluated in left-hand mode to reveal a location and the second in right-hand mode to obtain a value to be assigned to that location.

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

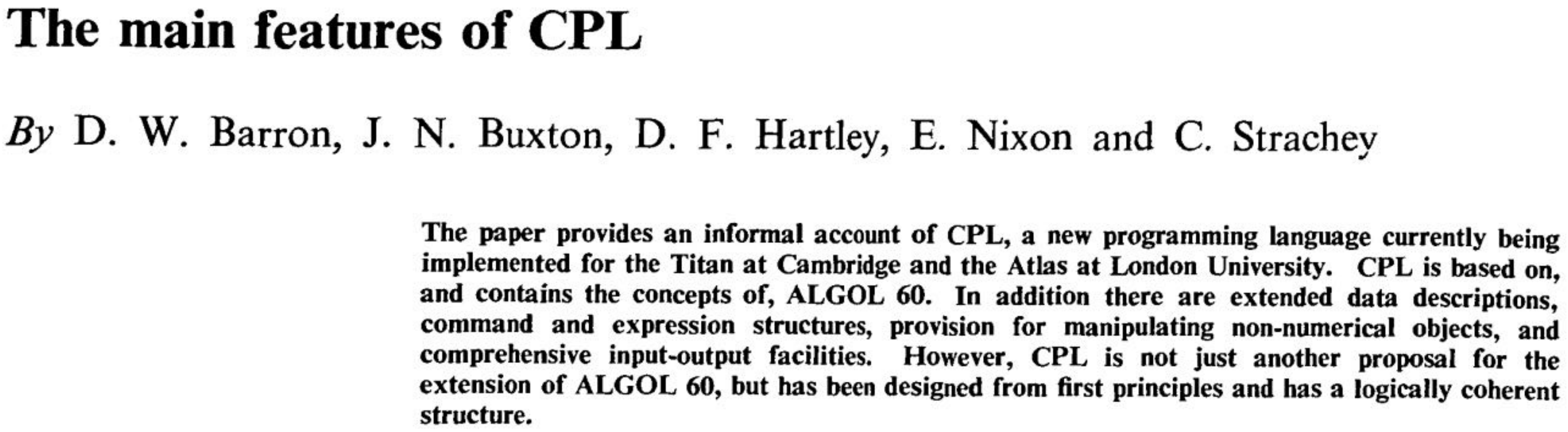
Advanced were made in understanding the evaluation of expressions so as to recognize not just the value of data but also its location. Taking terminology related to the assignment statement, we developed the concept of left-hand and right-hand values ... this enabled an assignment statement to have the generalized form

```
<expression> := <expression>
```

the first being evaluated in left-hand mode to reveal a location and the second in right-hand mode to obtain a value to be assigned to that location.

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

CPL as described in 1963



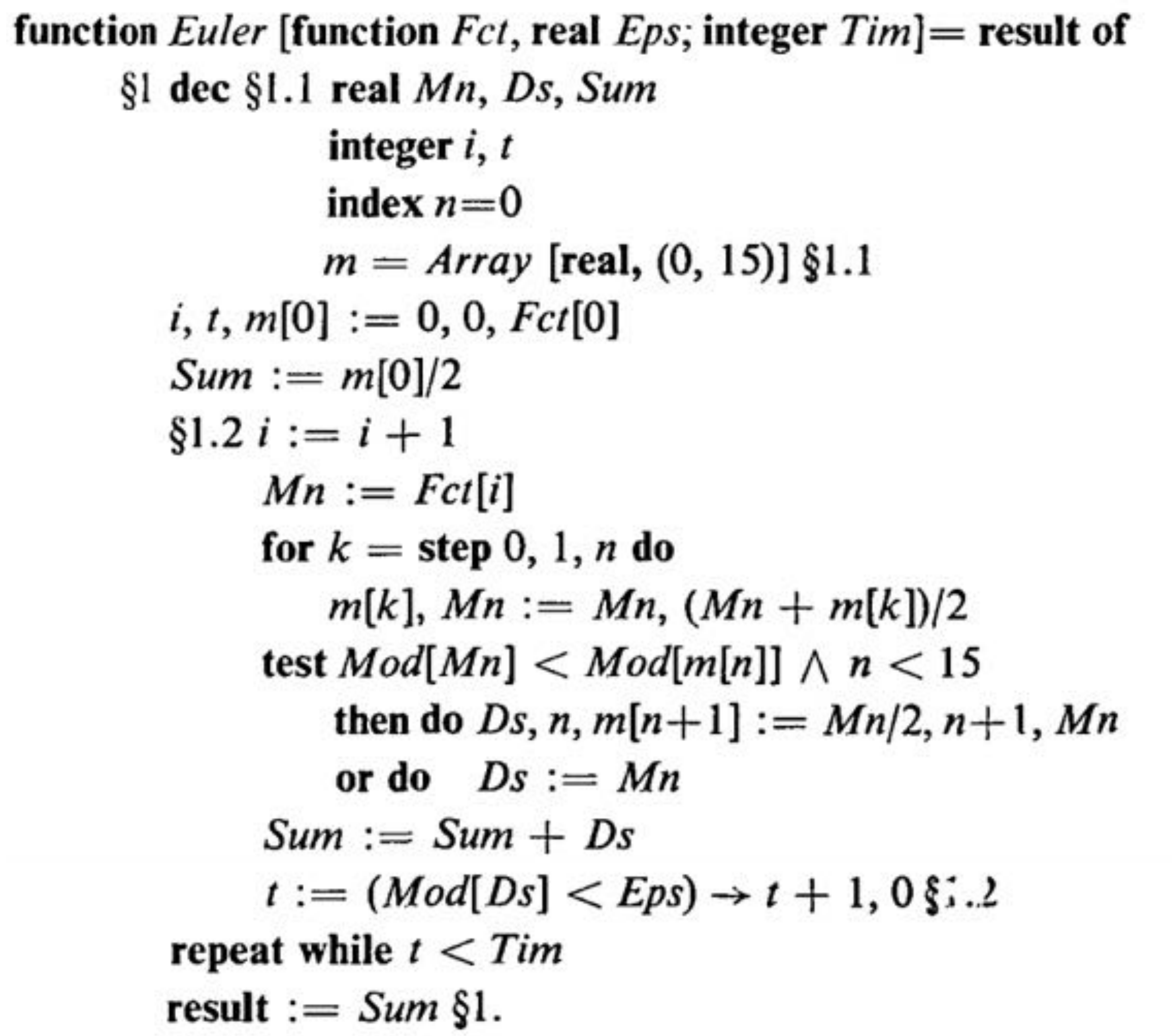
The main features of CPL

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

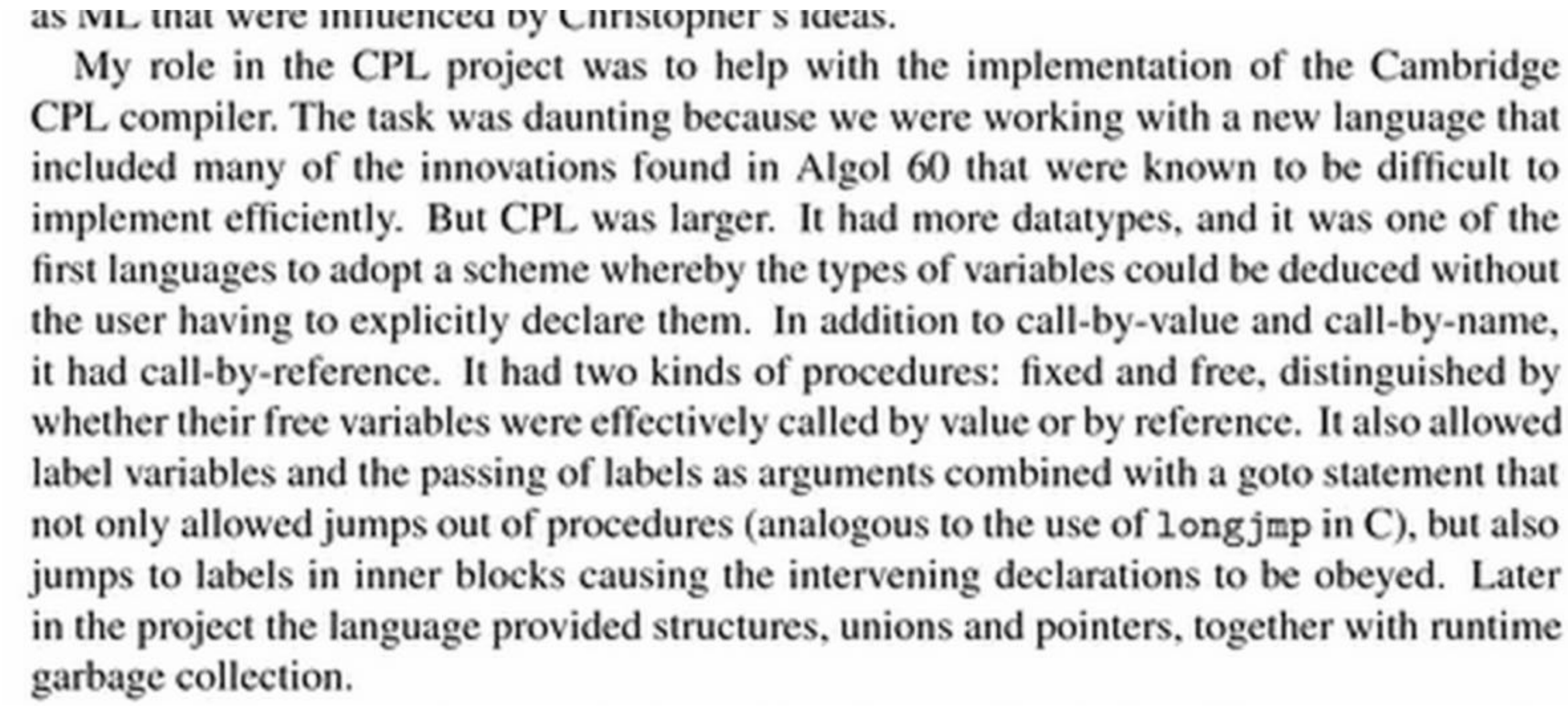
Example of CPL from 1963



```
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.
```

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

Martin Richards started as a research student in 1963



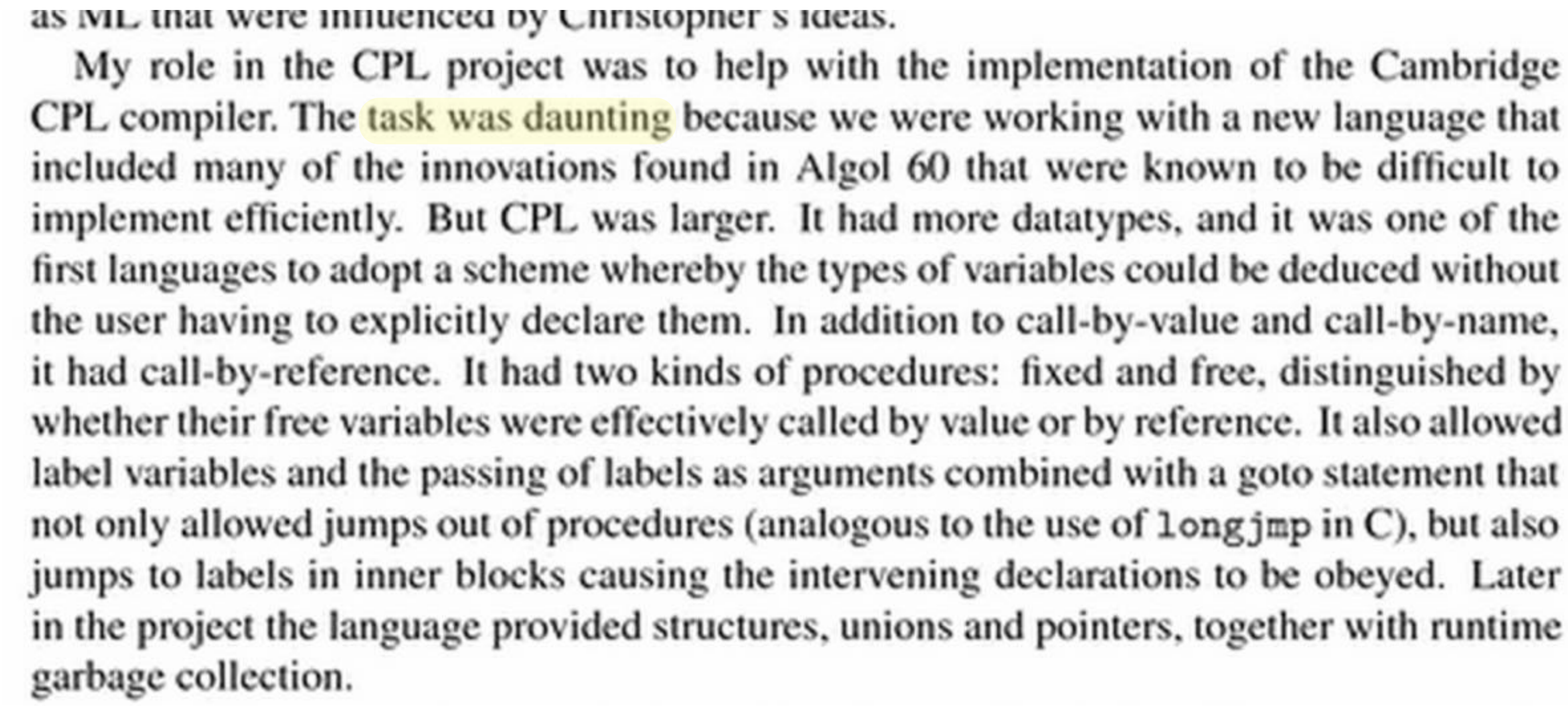
as ML that were influenced by Christopher's ideas.

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



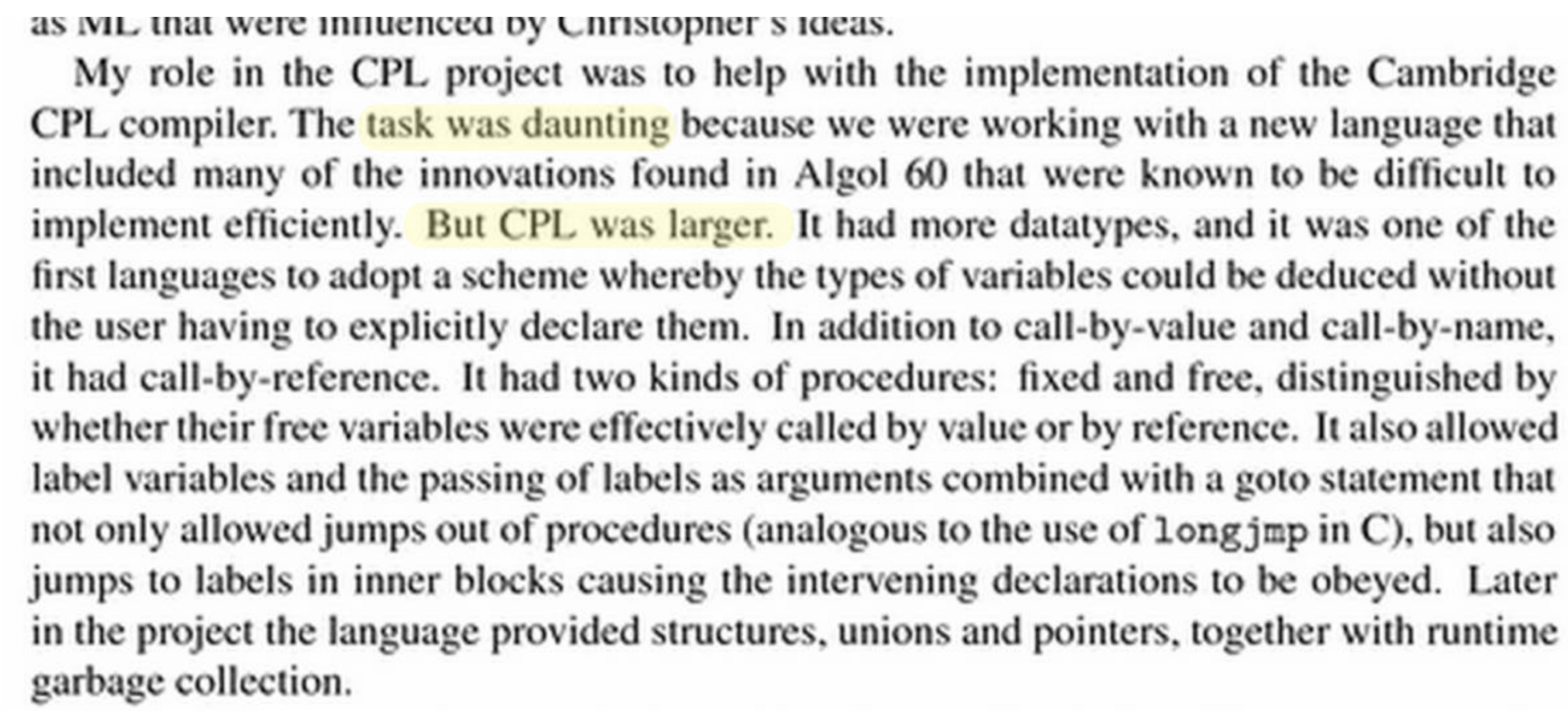
as ML that were influenced by Christopher's ideas.

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



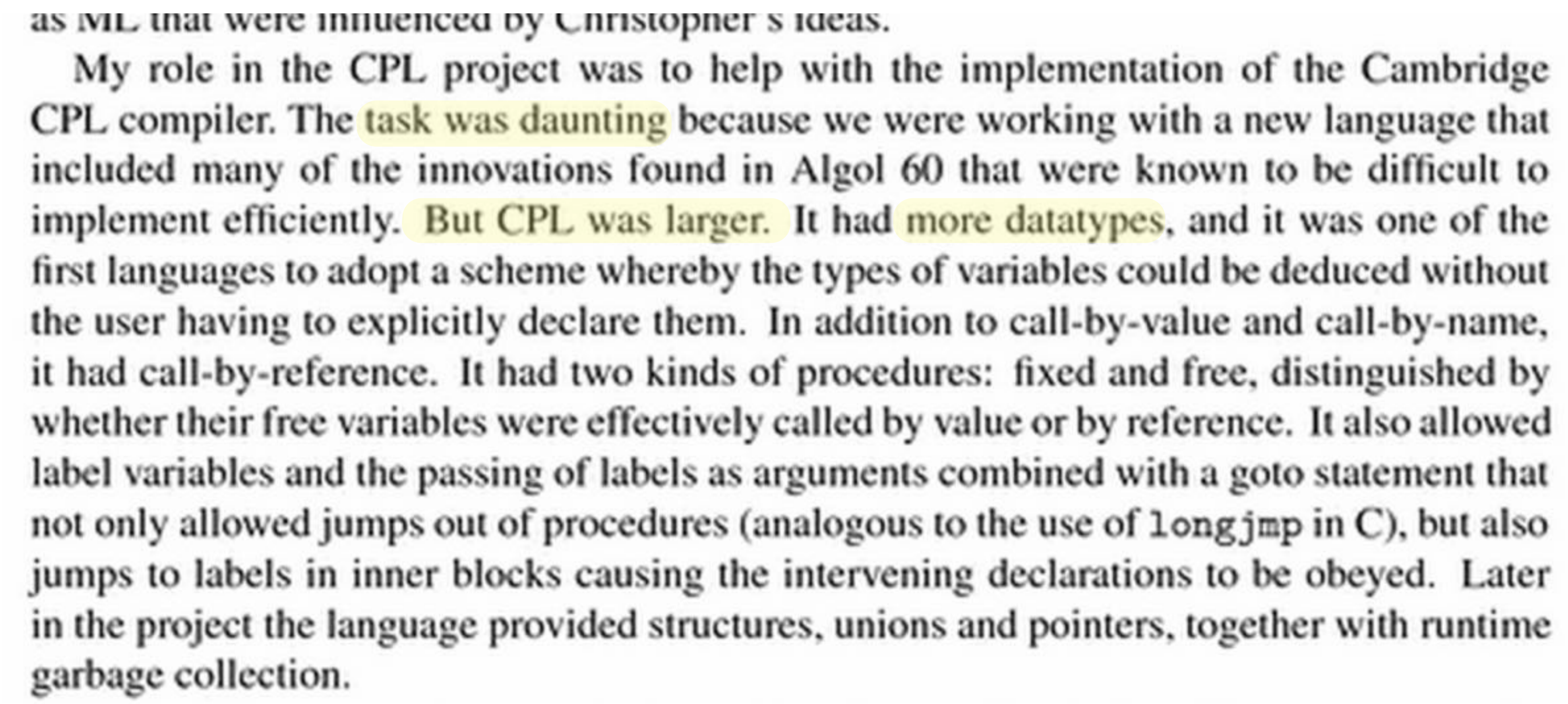
as ML that were influenced by Christopher's ideas.

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



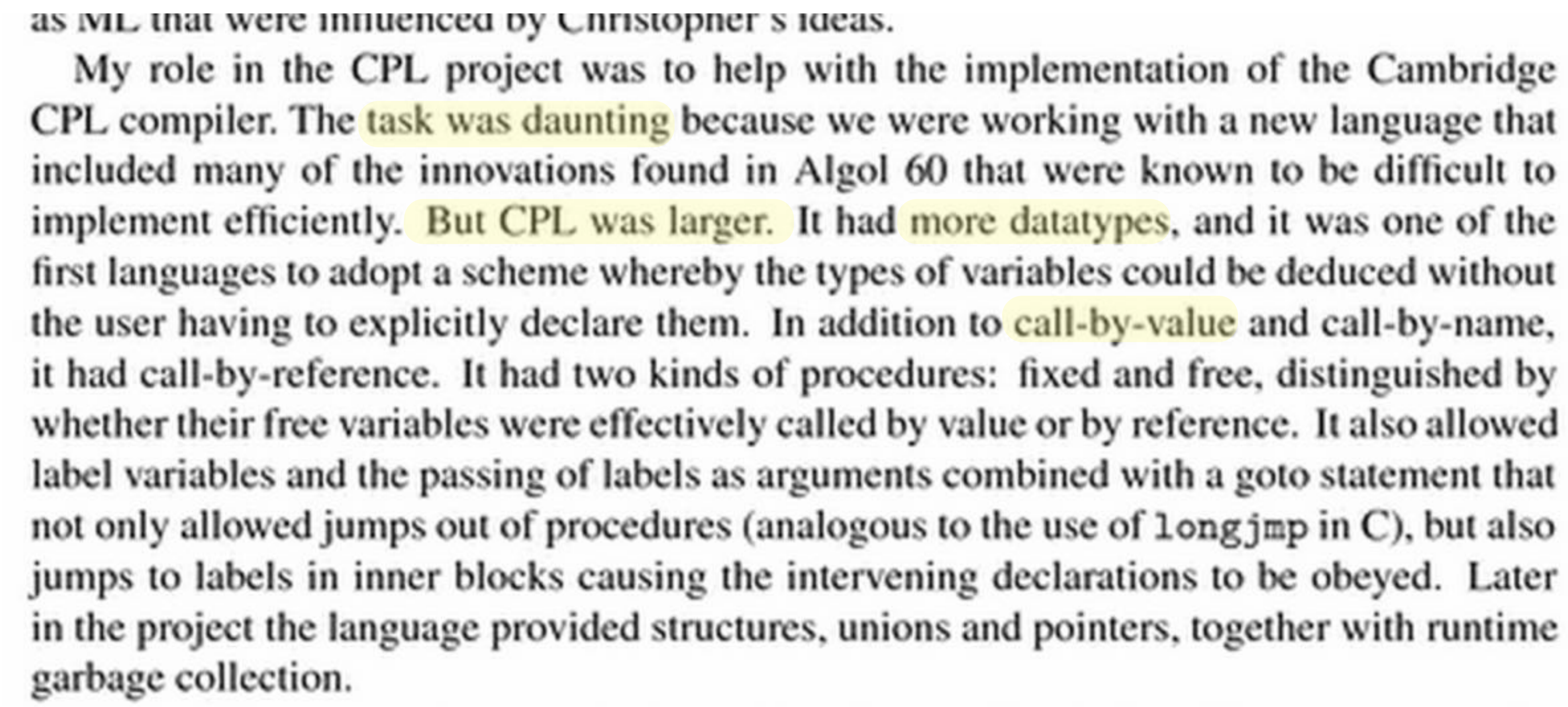
as ML that were influenced by Christopher's ideas.

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



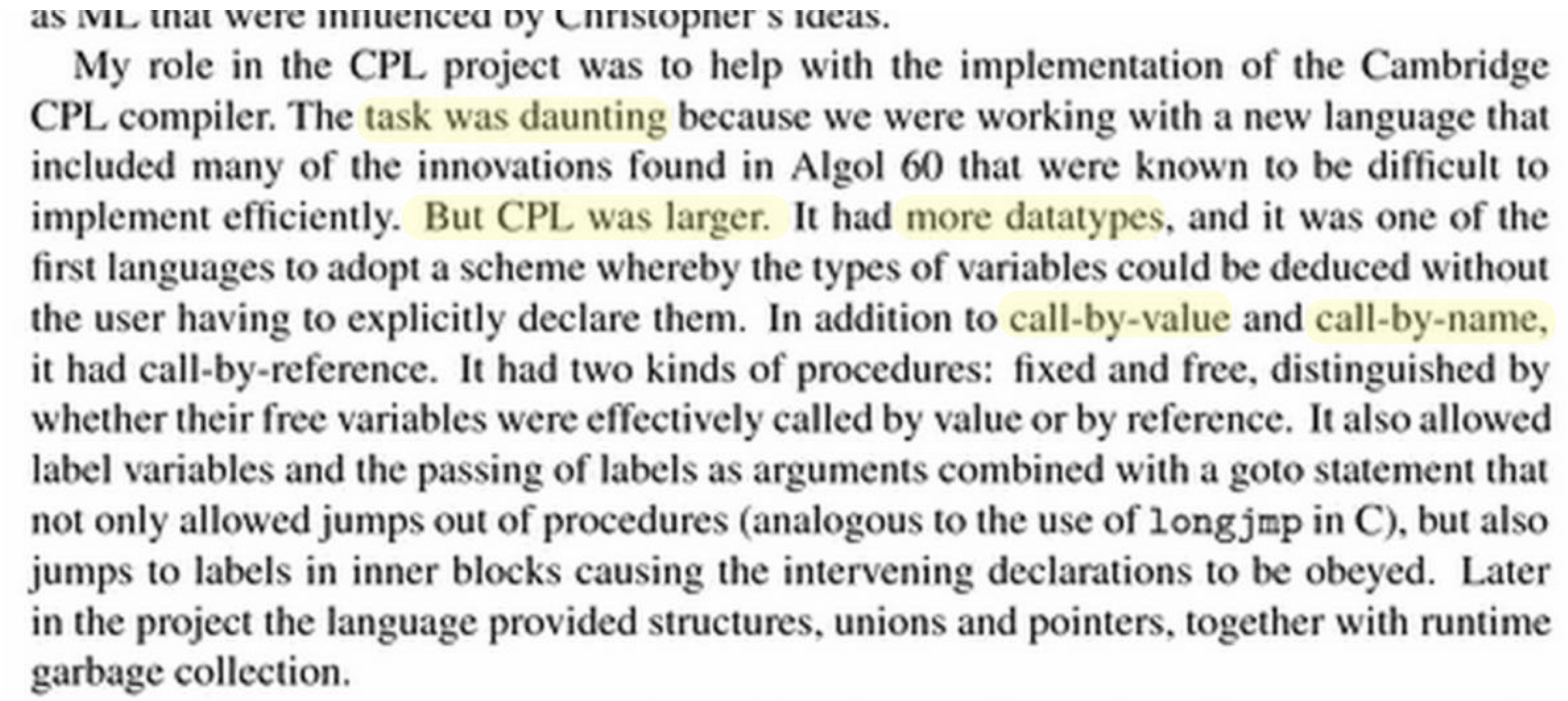
as ML that were influenced by Christopher's ideas.

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



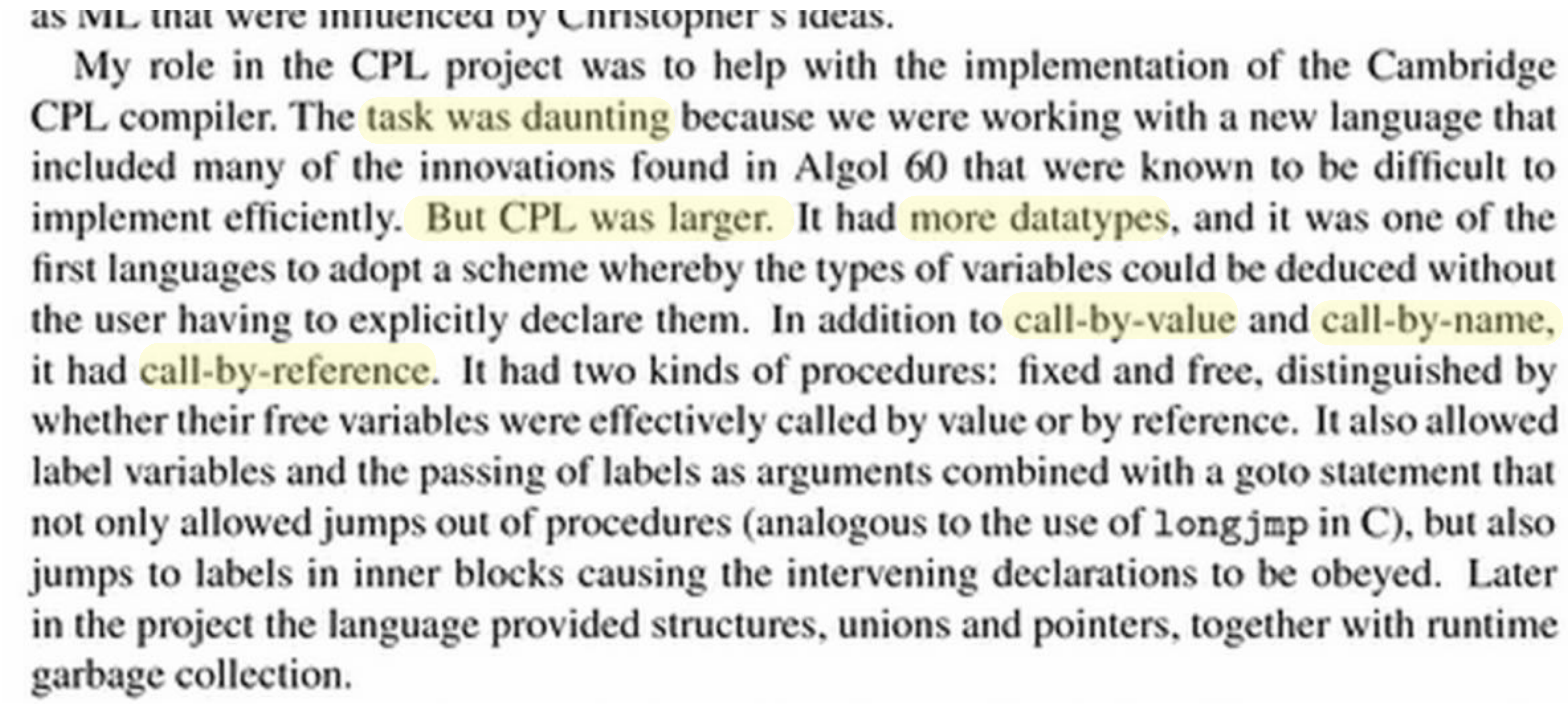
as ML that were influenced by Christopher's ideas.

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



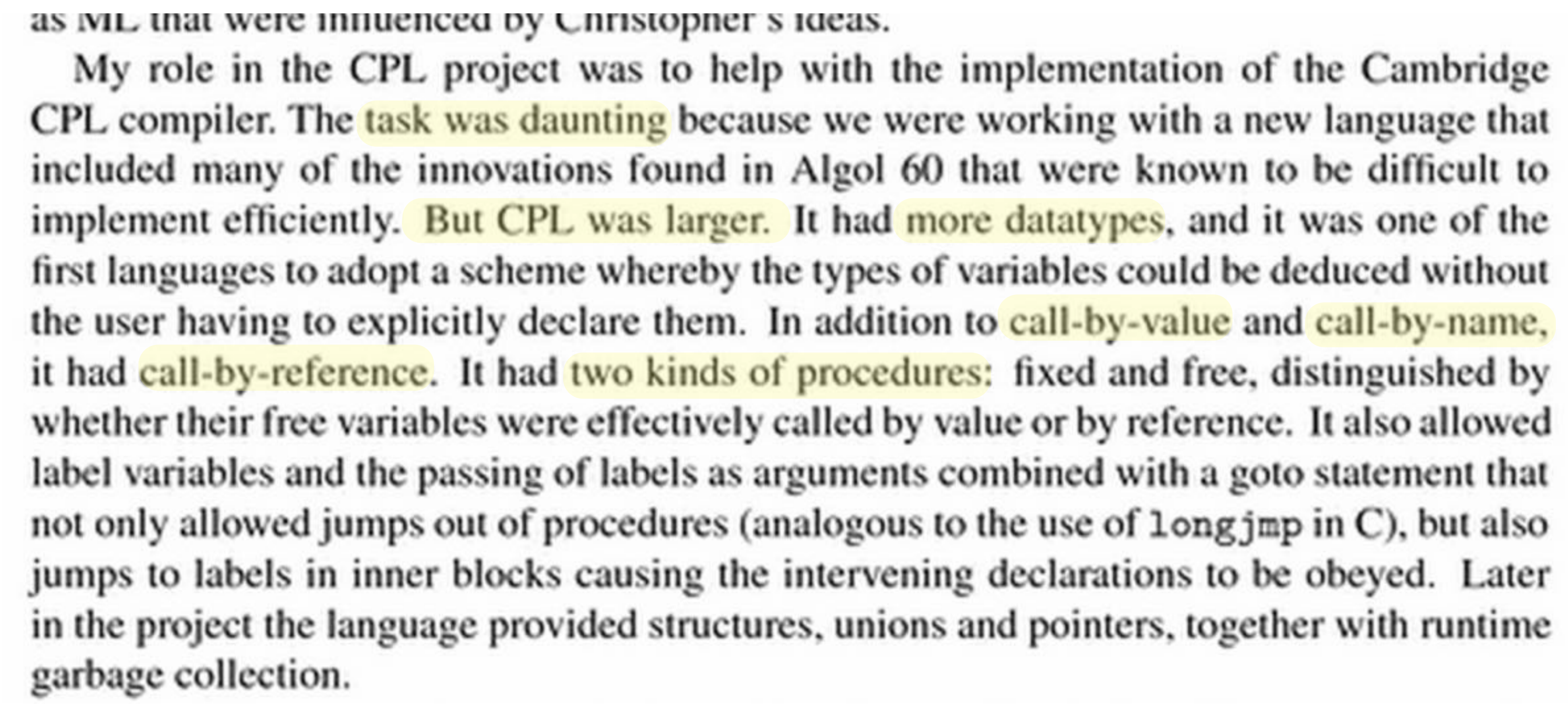
as ML that were influenced by Christopher's ideas.

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



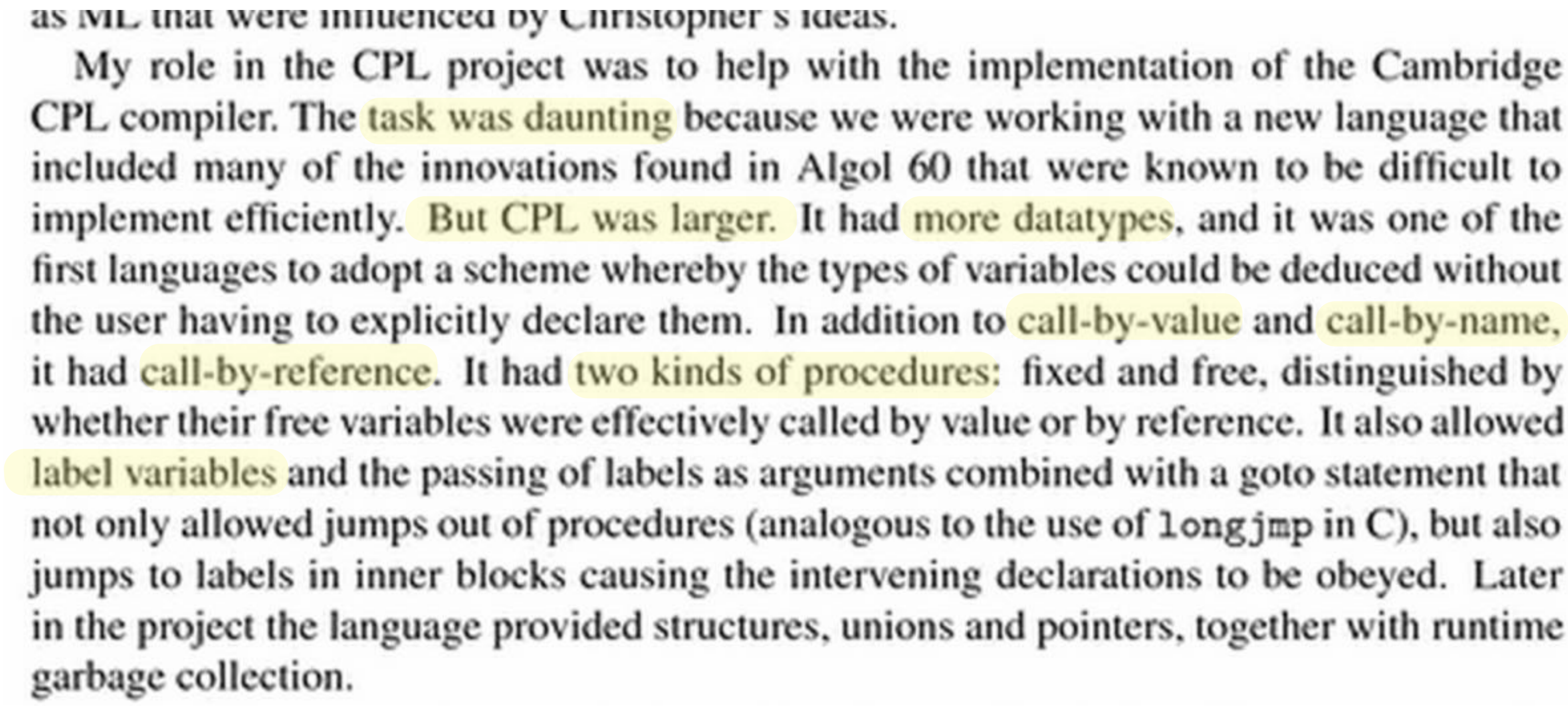
as ML that were influenced by Christopher's ideas.

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



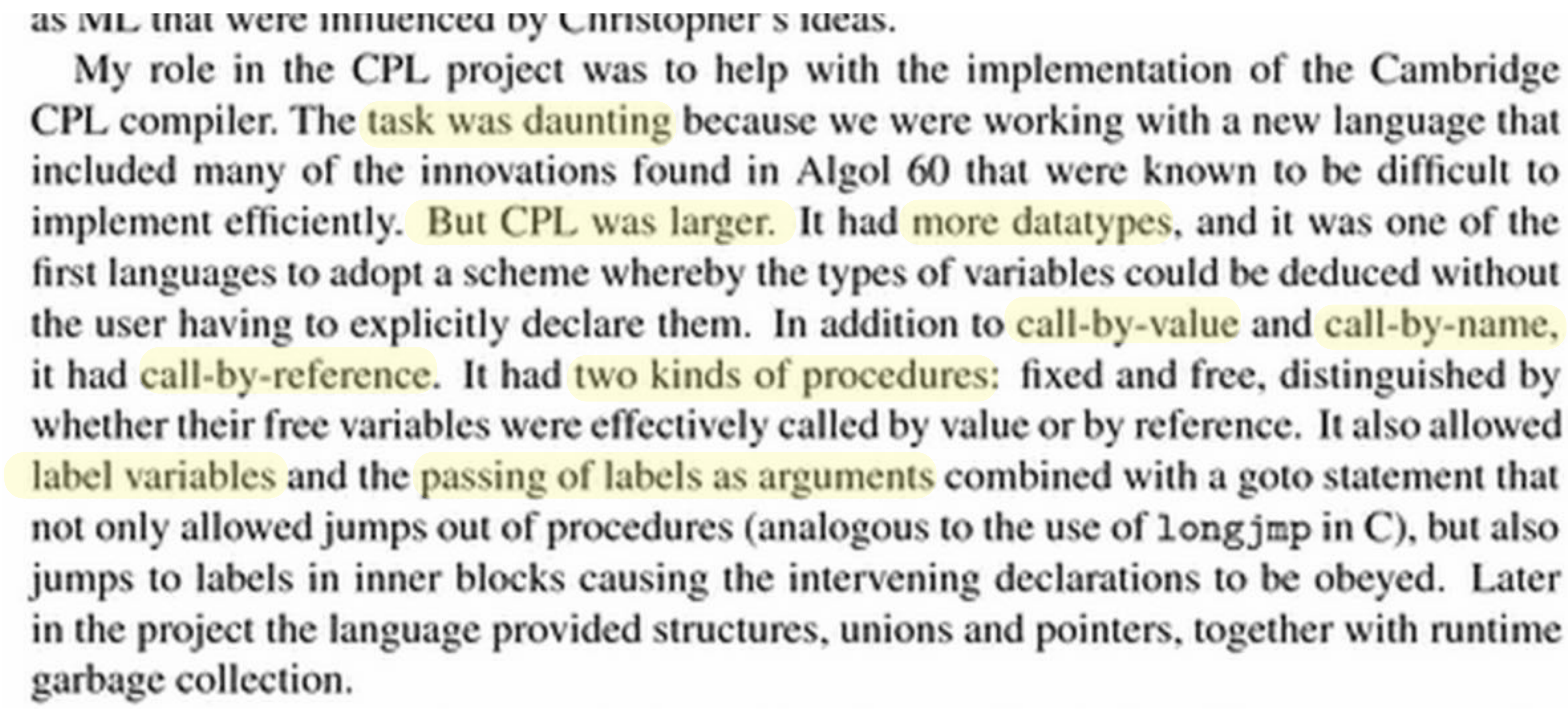
as ML that were influenced by Christopher's ideas.

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



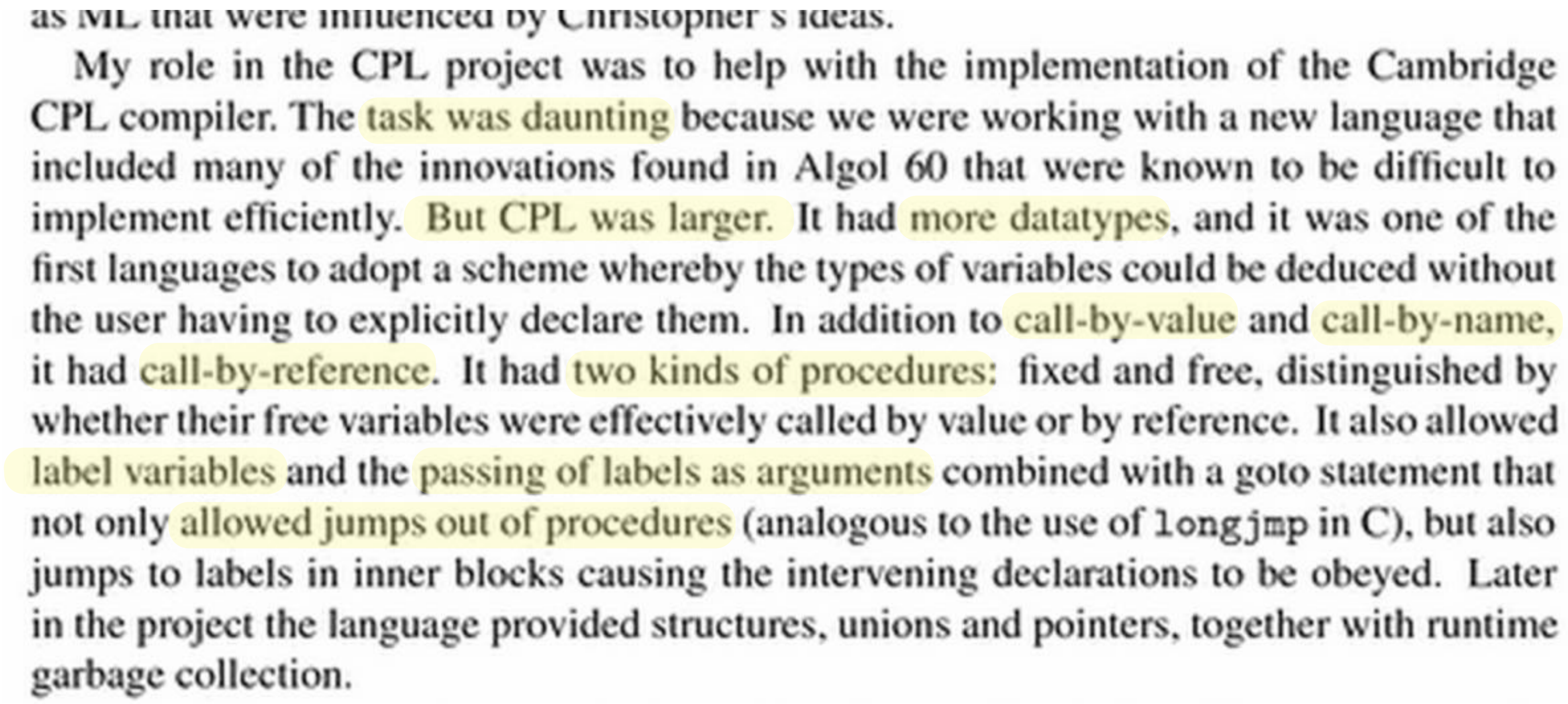
as ML that were influenced by Christopher's ideas.

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



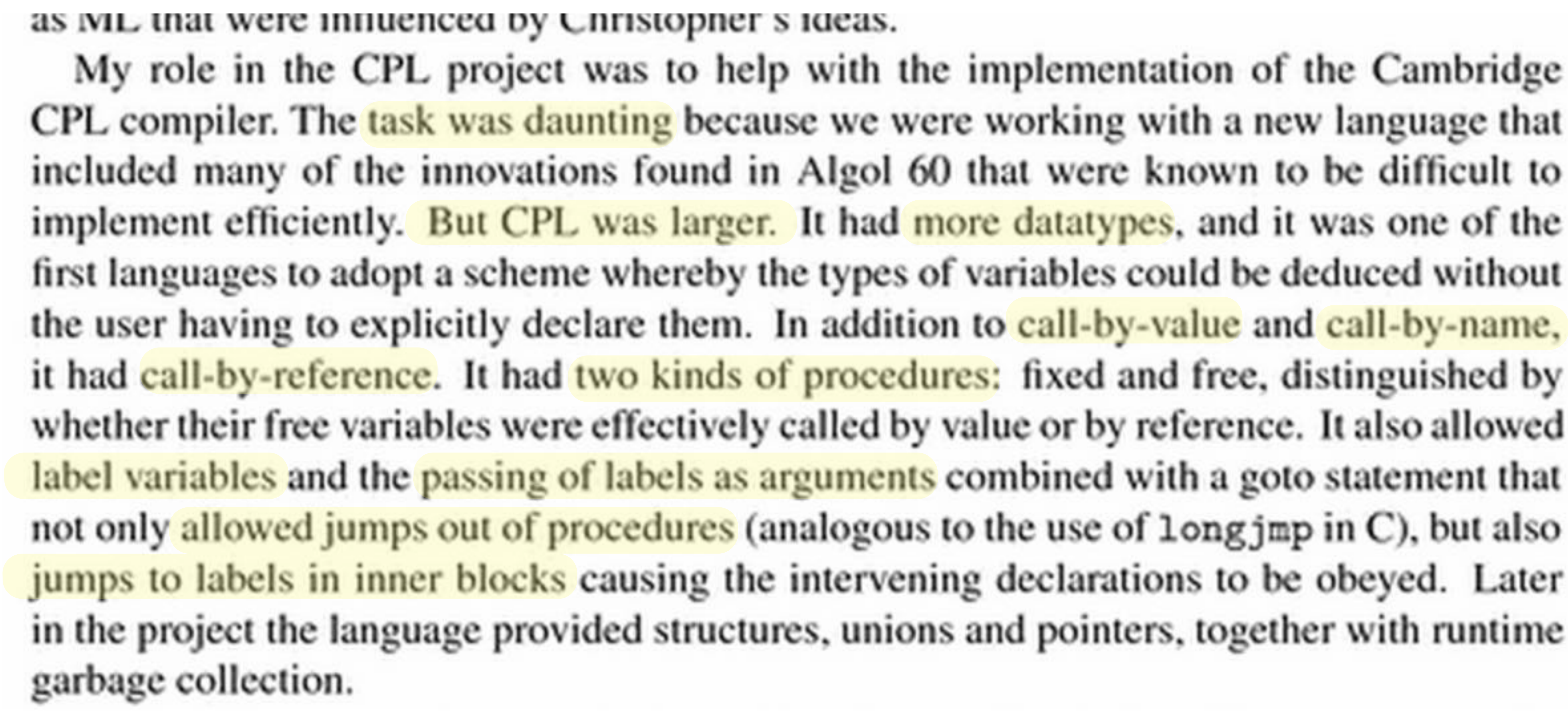
as ML that were influenced by Christopher's ideas.

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



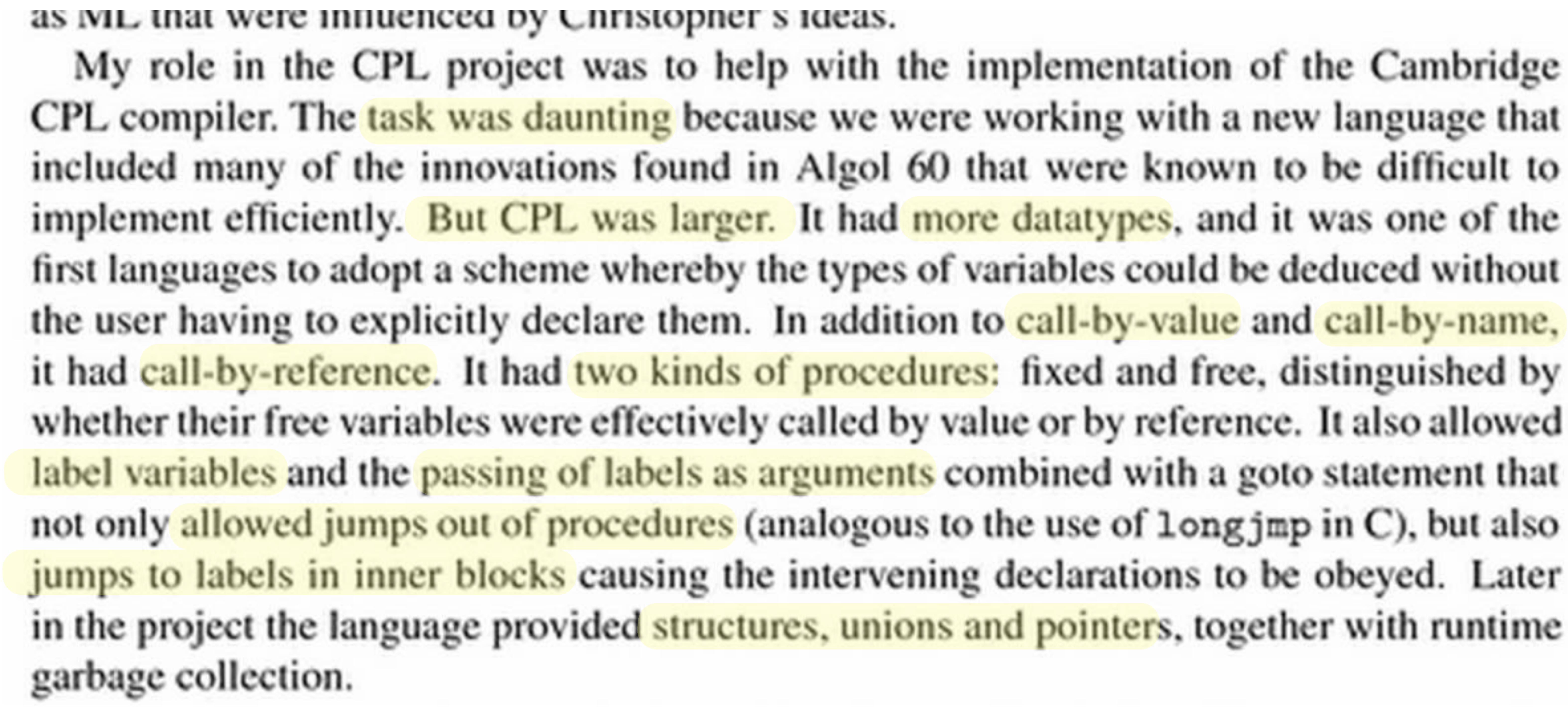
as ML that were influenced by Christopher's ideas.

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



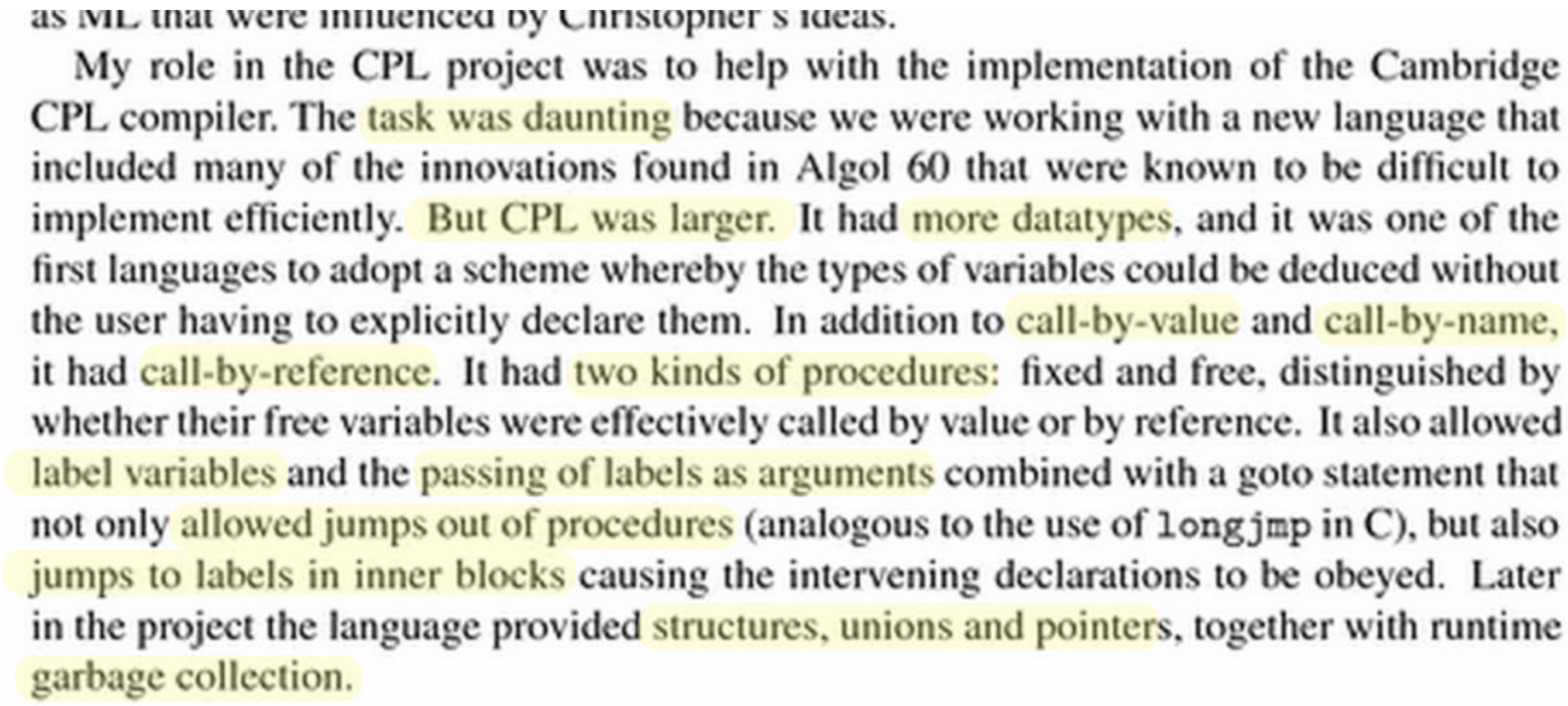
as ML that were influenced by Christopher's ideas.

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



as ML that were influenced by Christopher's ideas.

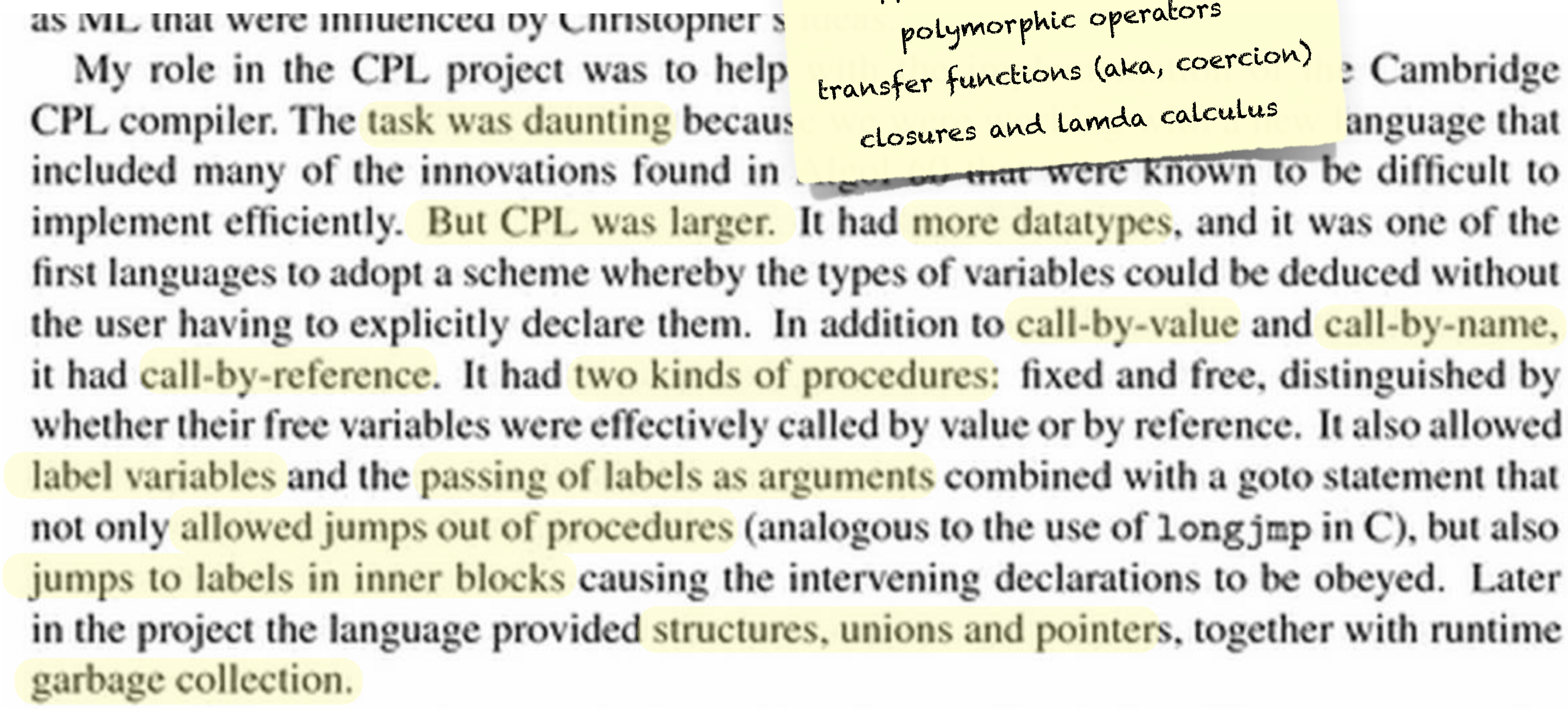
My role in the CPL project was to help with the implementation of the Cambridge CPL compiler. The task was daunting because we were working with a new language that included many of the innovations found in Algol 60 that were known to be difficult to implement efficiently. But CPL was larger. It had 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 [illegible] 963

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

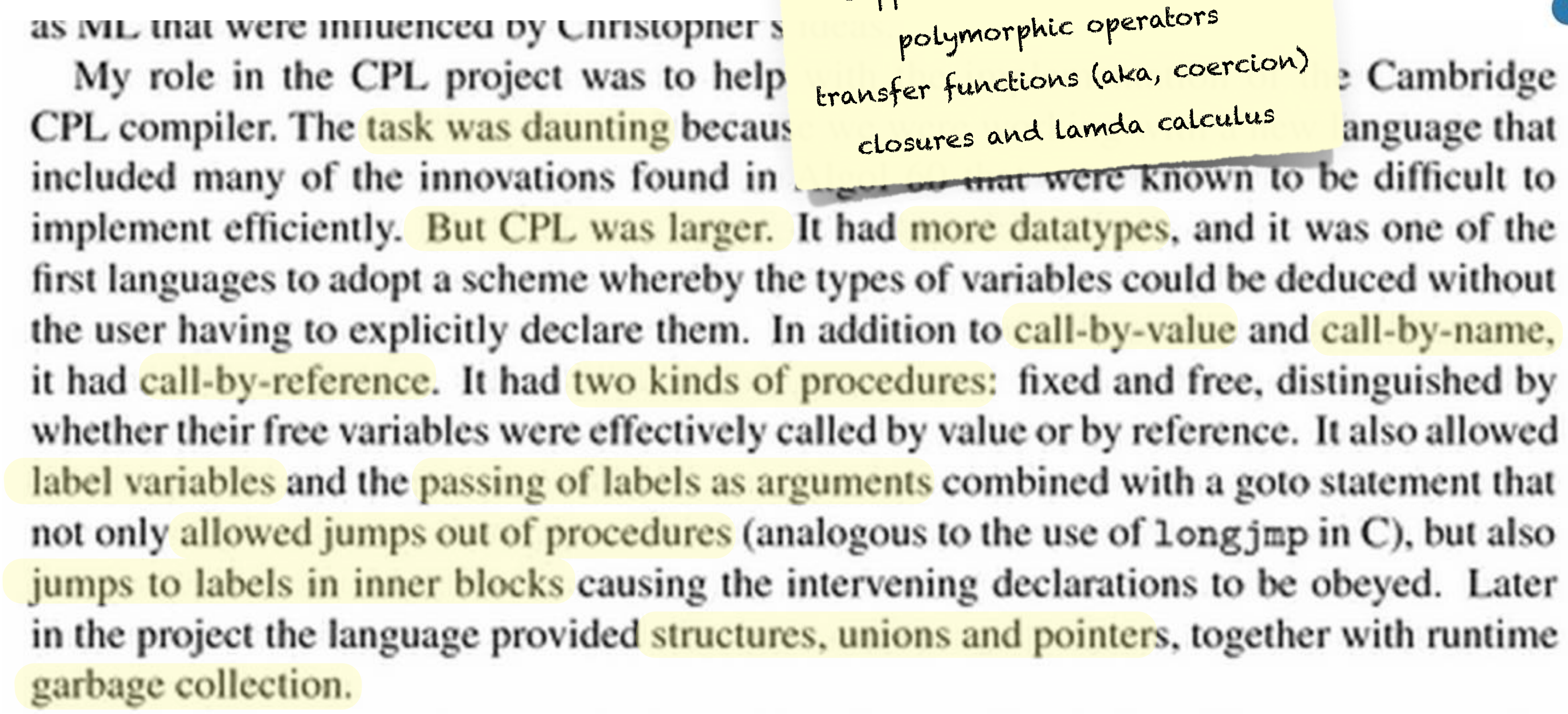

as ML that were influenced by Christopher S

My role in the CPL project was to help [illegible] e Cambridge CPL compiler. The task was daunting becaus [illegible] language that included many of the innovations found in [illegible] 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



as ML that were influenced by Christopher S

My role in the CPL project was to help 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.

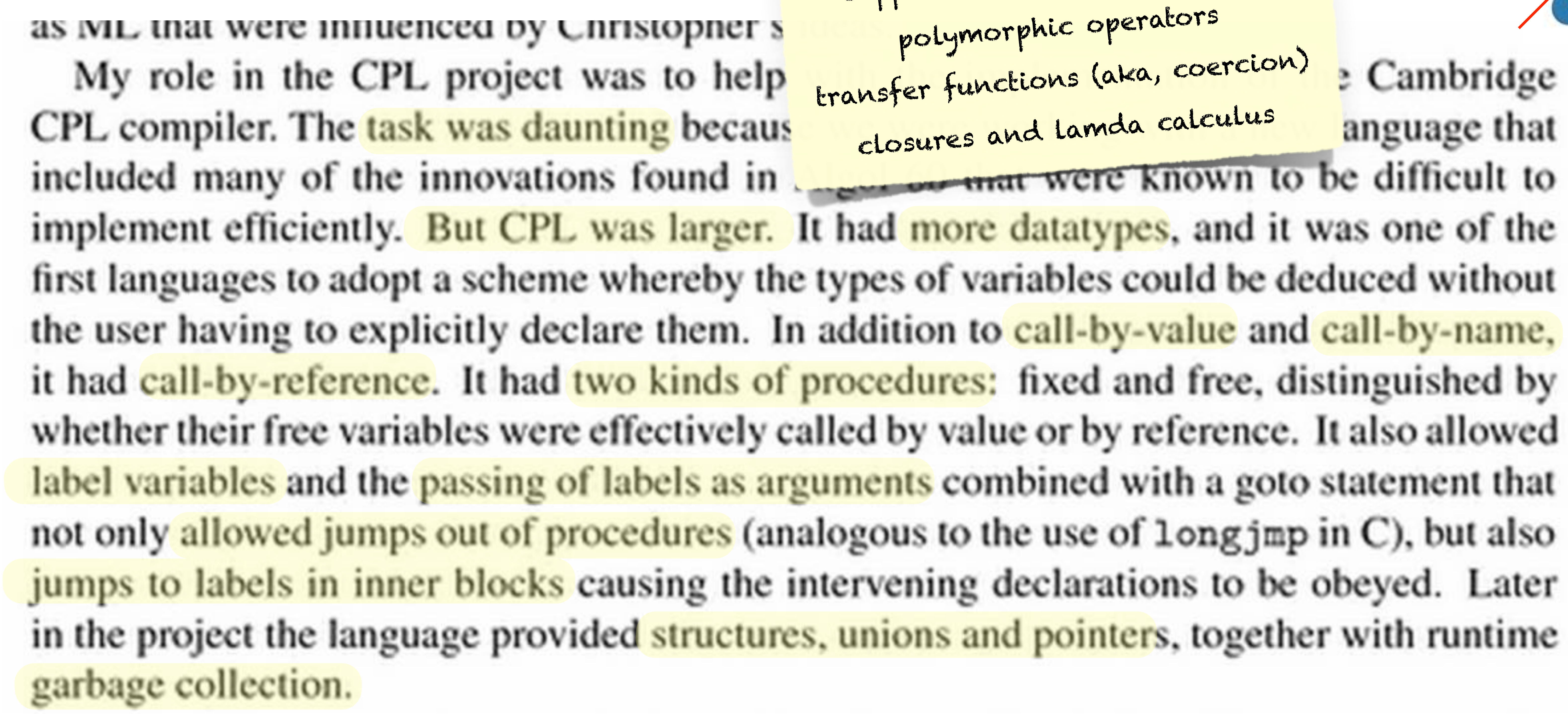
double floating point precision
support for complex numbers
polymorphic operators
transfer functions (aka, coercion)
closures and lamda calculus

"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 anguage that included many of the innovations found in 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.

Writing a compiler for CPL was too difficult.

Cambridge never succeeded writing a working CPL compiler.

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:

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

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 book, 1979)

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

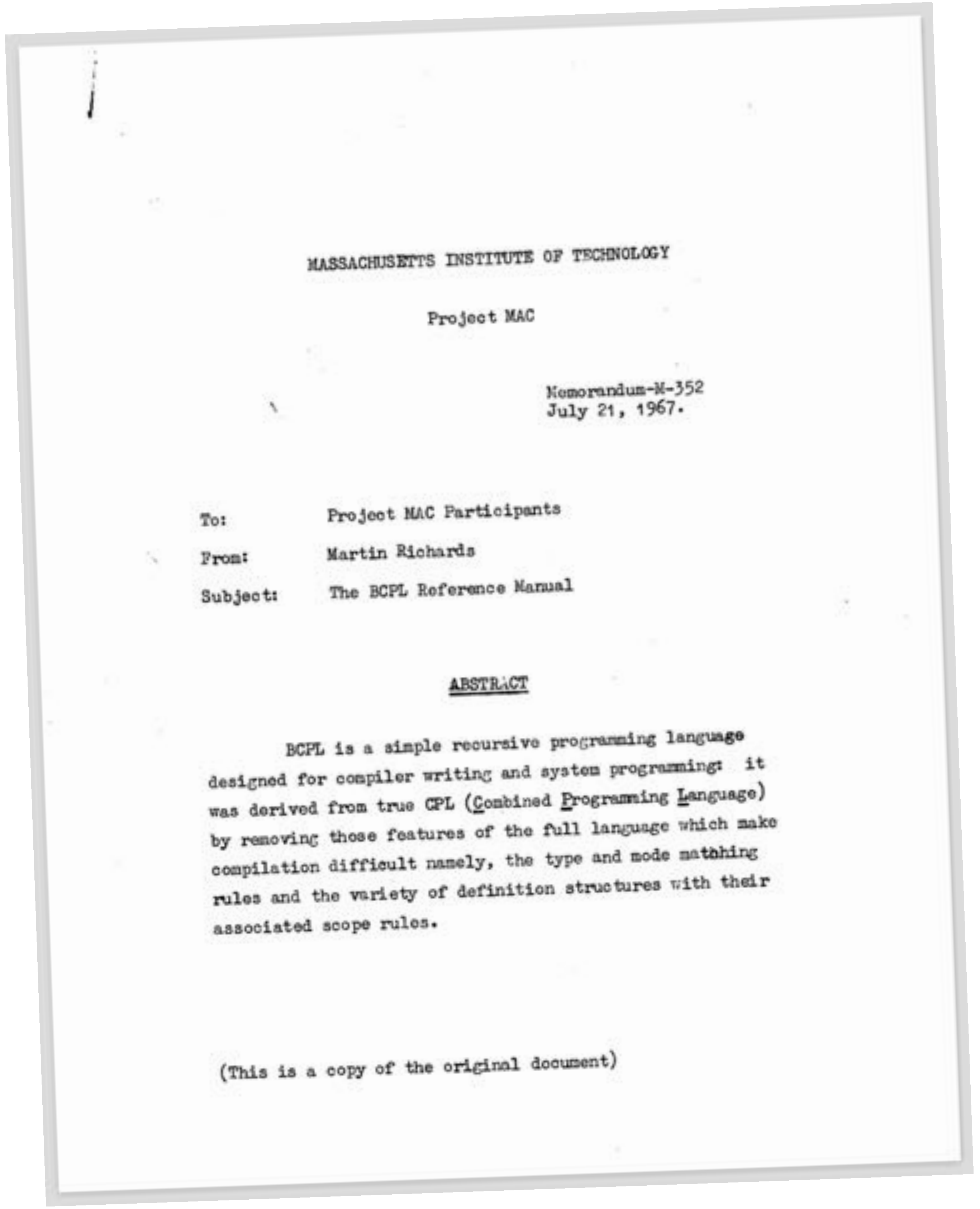
"The philosophy of BCPL is not one of the tyrant who thinks he knows best and lay down the law on what is and what is not allowed; rather, BCPL acts more as a servant offering his services to the best of his ability without complaint, even when confronted with apparent nonsense. The programmer is always assumed to know what he is doing and is not hemmed in by petty restrictions." (The BCPL book, 1979)

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 book, 1979)

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

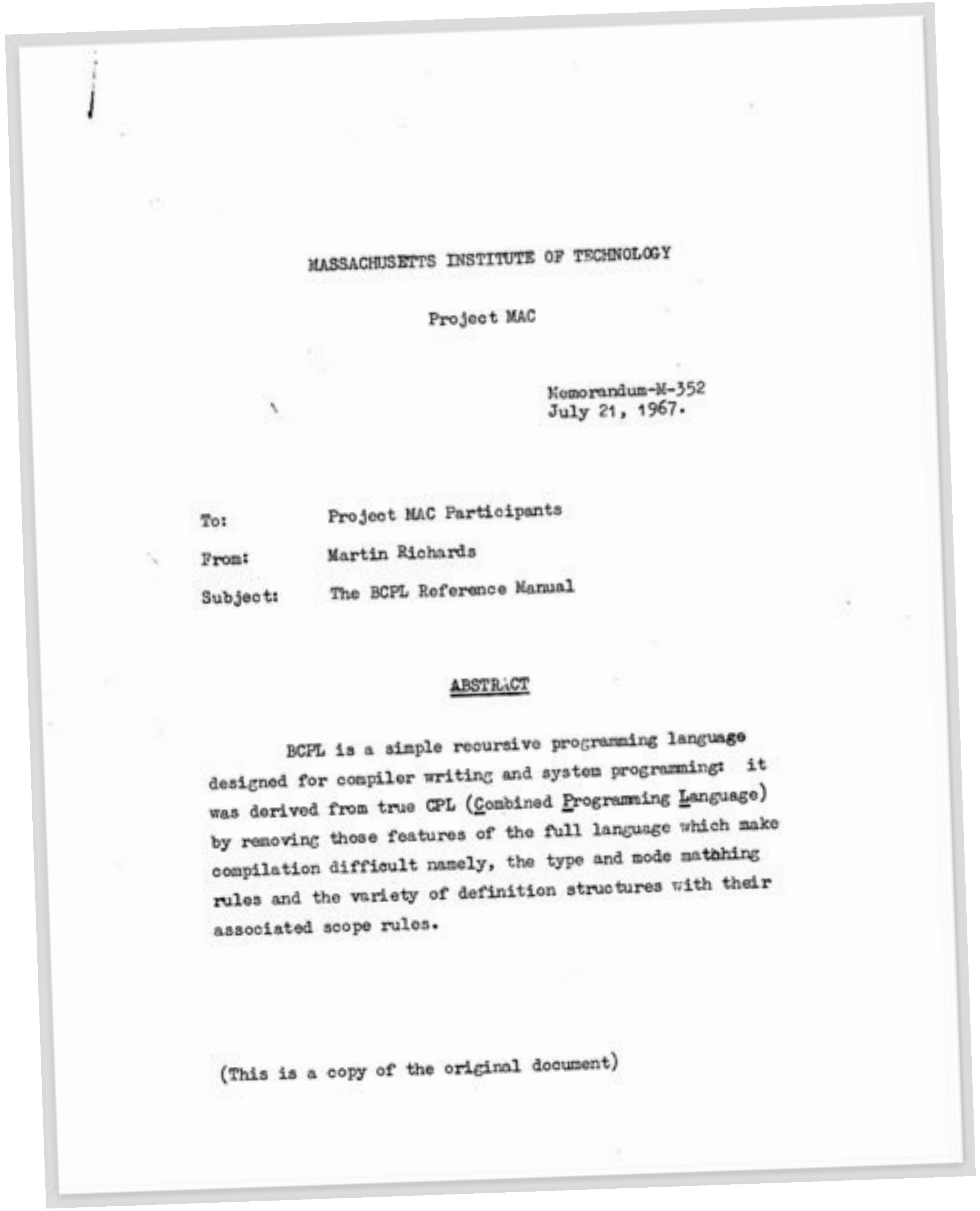
ABSTRACT

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)

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.

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

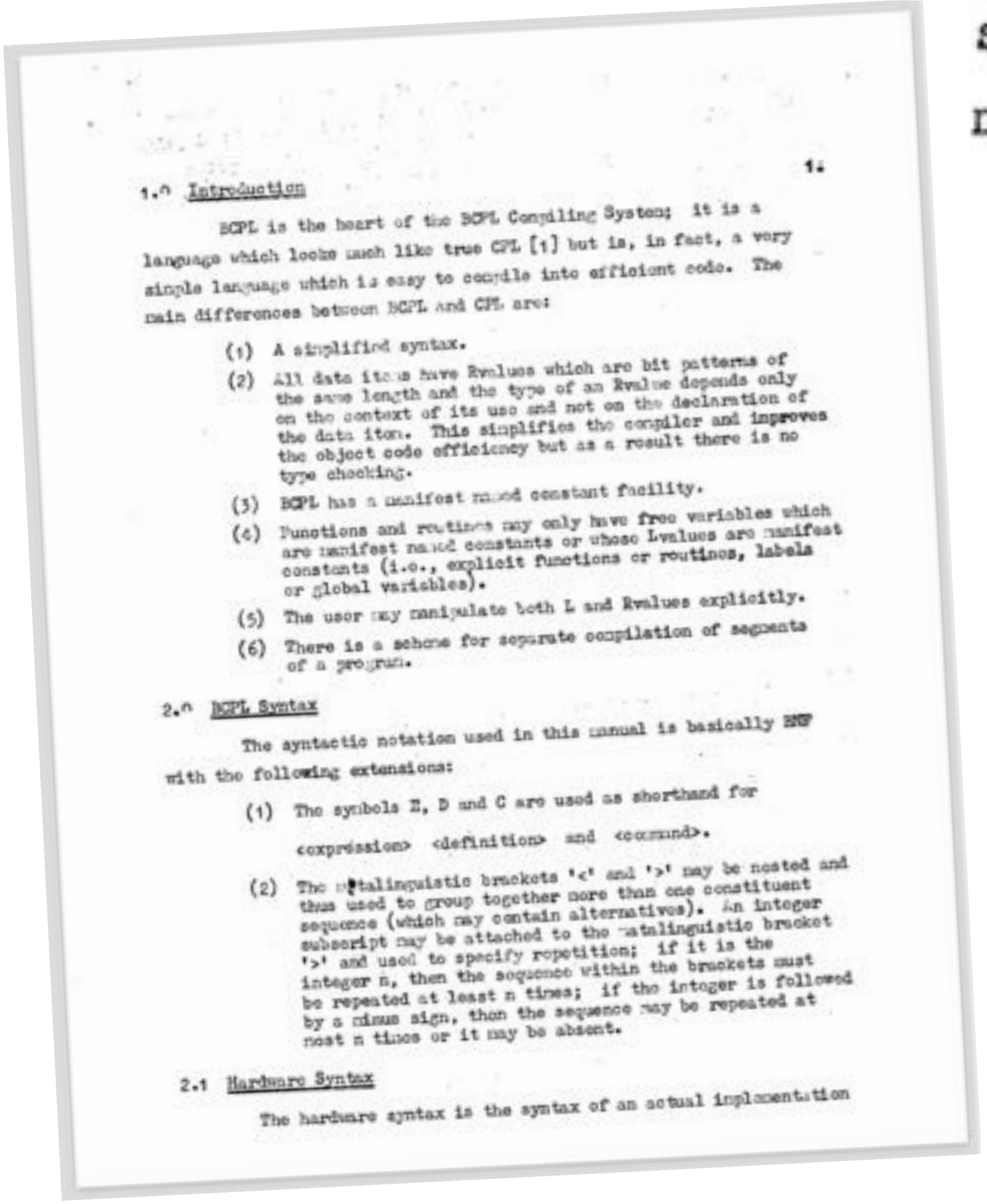
Subject: The BCPL Reference Manual

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)

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.



1.

1.0 Introduction

BCPL is the heart of the BCPL Compiling System; it is a language which looks 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 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.

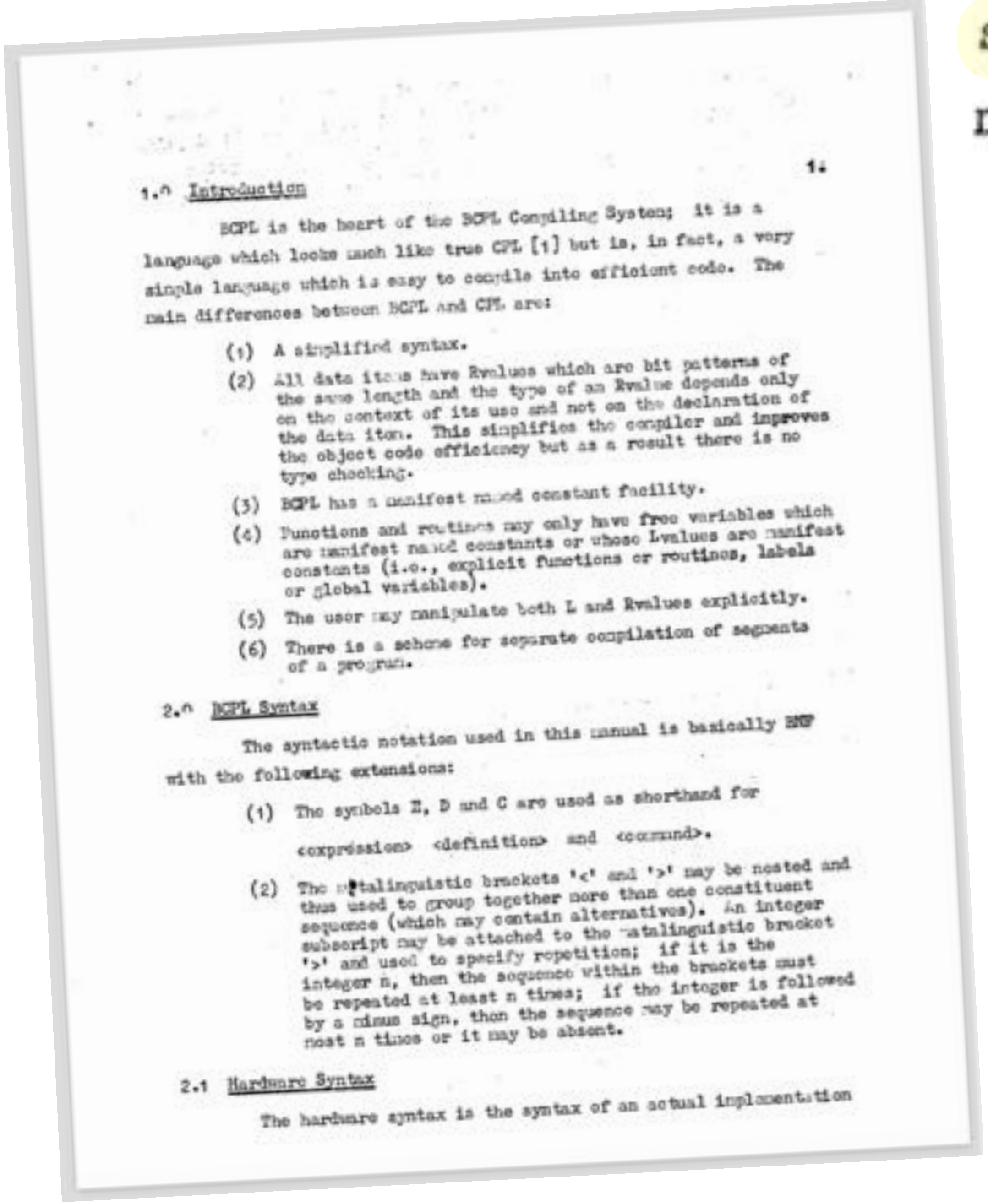
(2) All data items have Rvalues which are bit patterns of the same length and the type of an Rvalue depends only on the context of its use and not on the declaration of the data item. This simplifies the compiler and improves the object code efficiency but as a result there is no type checking.

(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.



1.0 Introduction

1.

BCPL is the heart of the BCPL Compiling System; it is a language which looks 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 a language which looks 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.

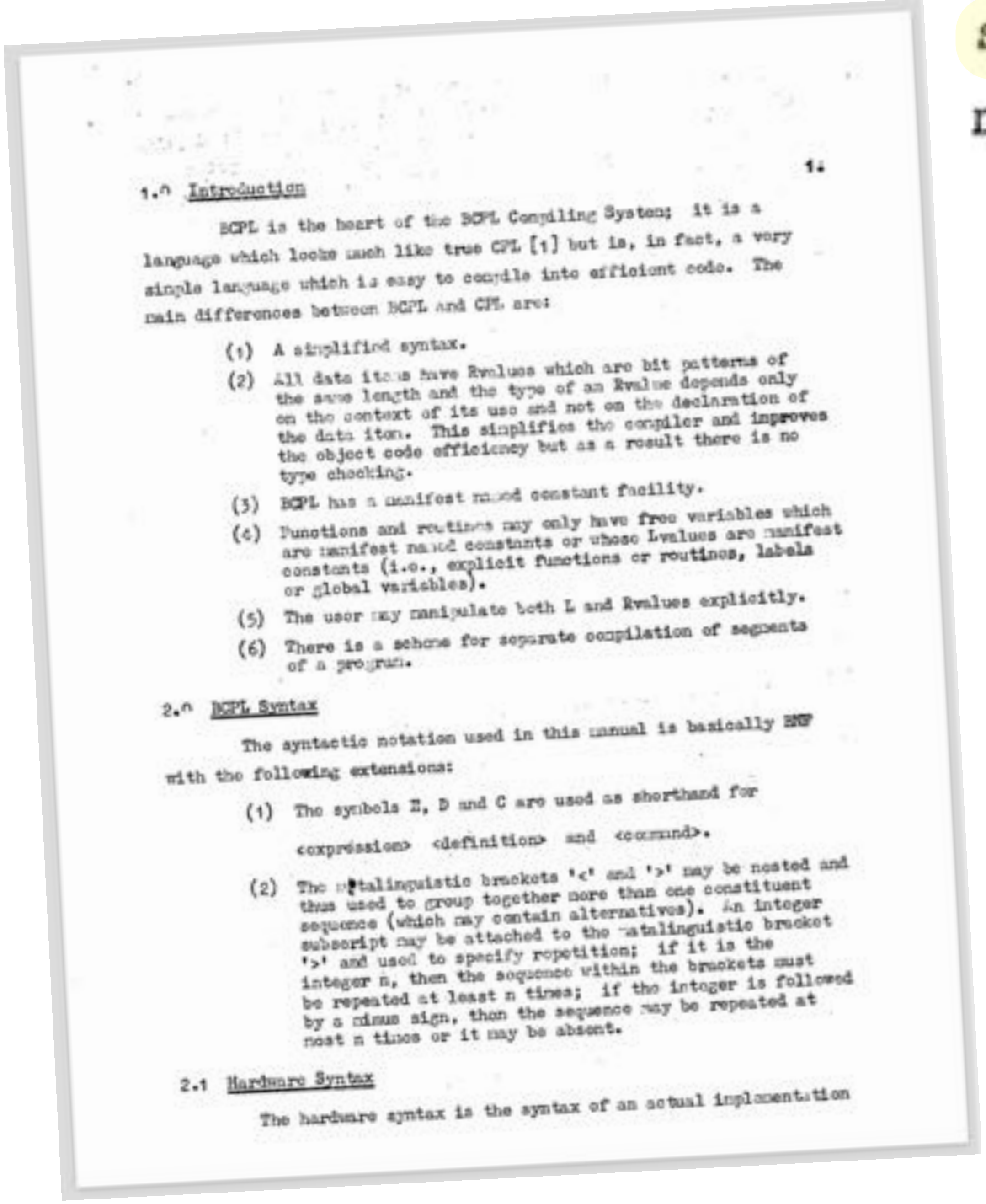
(2) All data items have Rvalues which are bit patterns of the same length and the type of an Rvalue depends only on the context of its use and not on the declaration of the data item. This simplifies the compiler and improves the object code efficiency but as a result there is no type checking.

(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.



1.0 Introduction

1.

BCPL is the heart of the BCPL Compiling System; it is a language which looks 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:

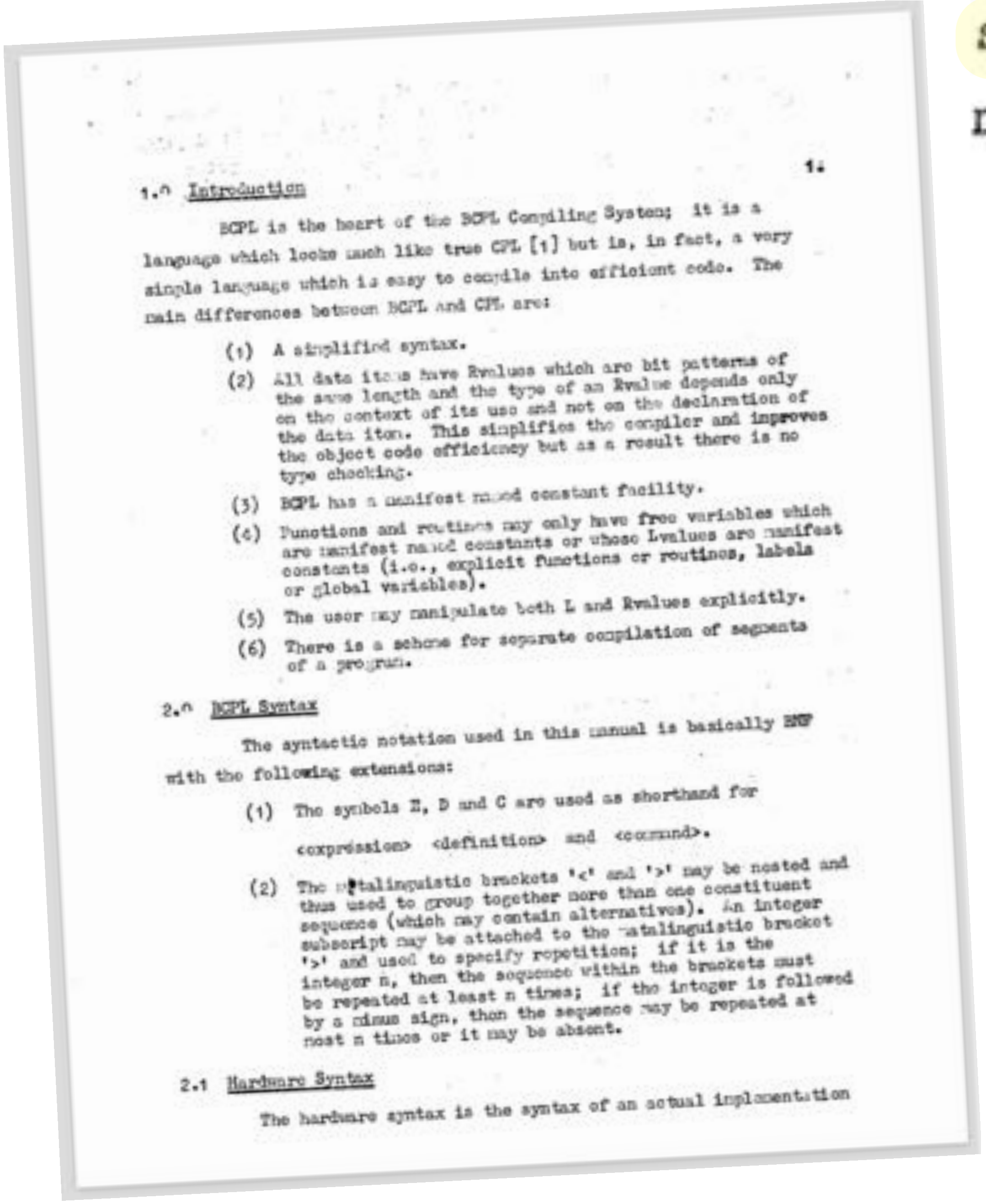
(1) The symbols E, D and C are used as shorthand for <expression> <definition> and <command>.
(2) The metalinguistic brackets '<' and '>' may be nested and thus used to group together more than one constituent sequence (which may contain alternatives). An integer subscript may be attached to the metalinguistic 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 a language which looks 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.



1.

1.0 Introduction

BCPL is the heart of the BCPL Compiling System; it is a language which looks 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 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.

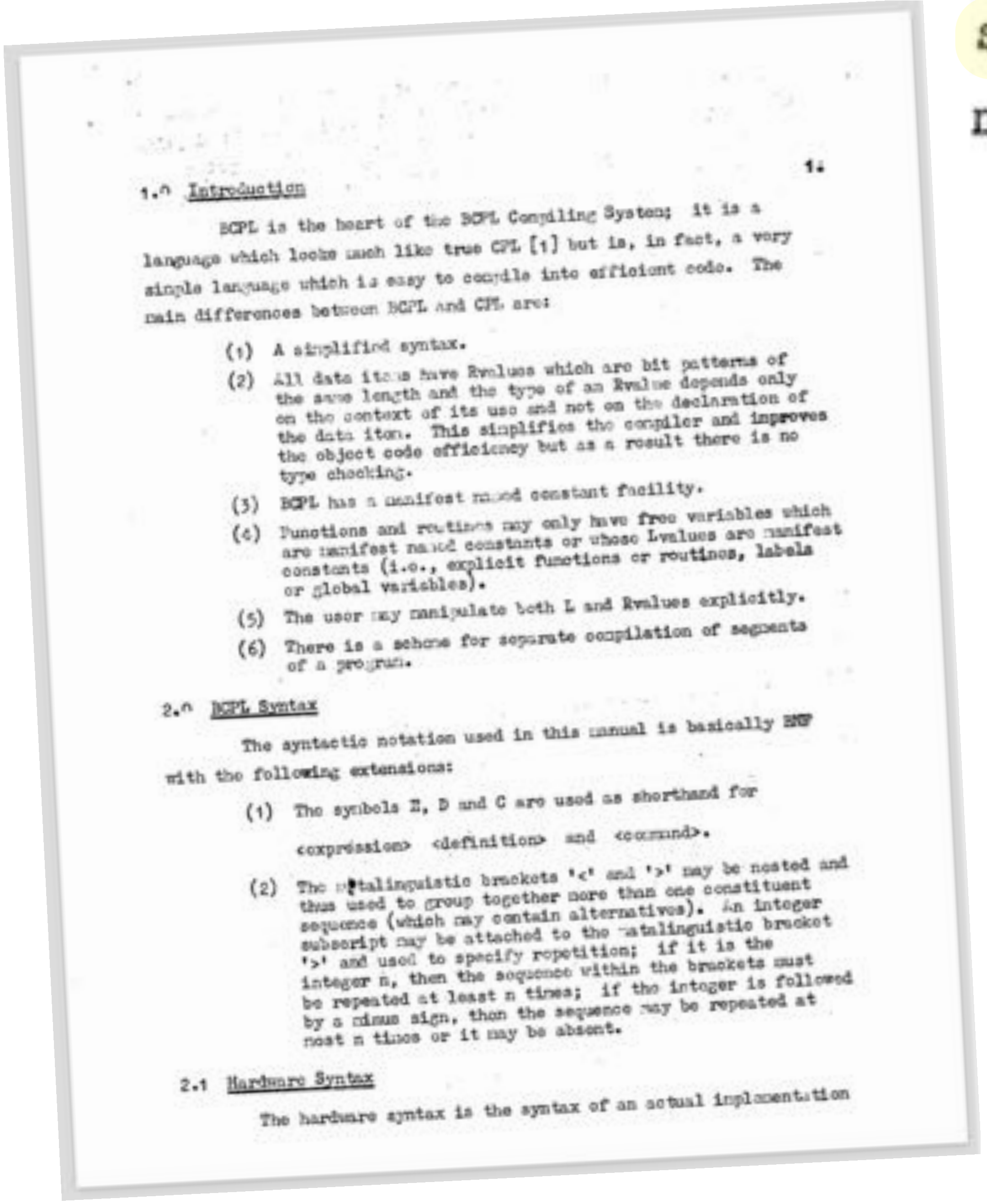
(2) All data items have Rvalues which are bit patterns of the same length and the type of an Rvalue depends only on the context of its use and not on the declaration of the data item. This simplifies the compiler and improves the object code efficiency but as a result there is no type checking.

(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.



1 .

1.0 Introduction

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 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.

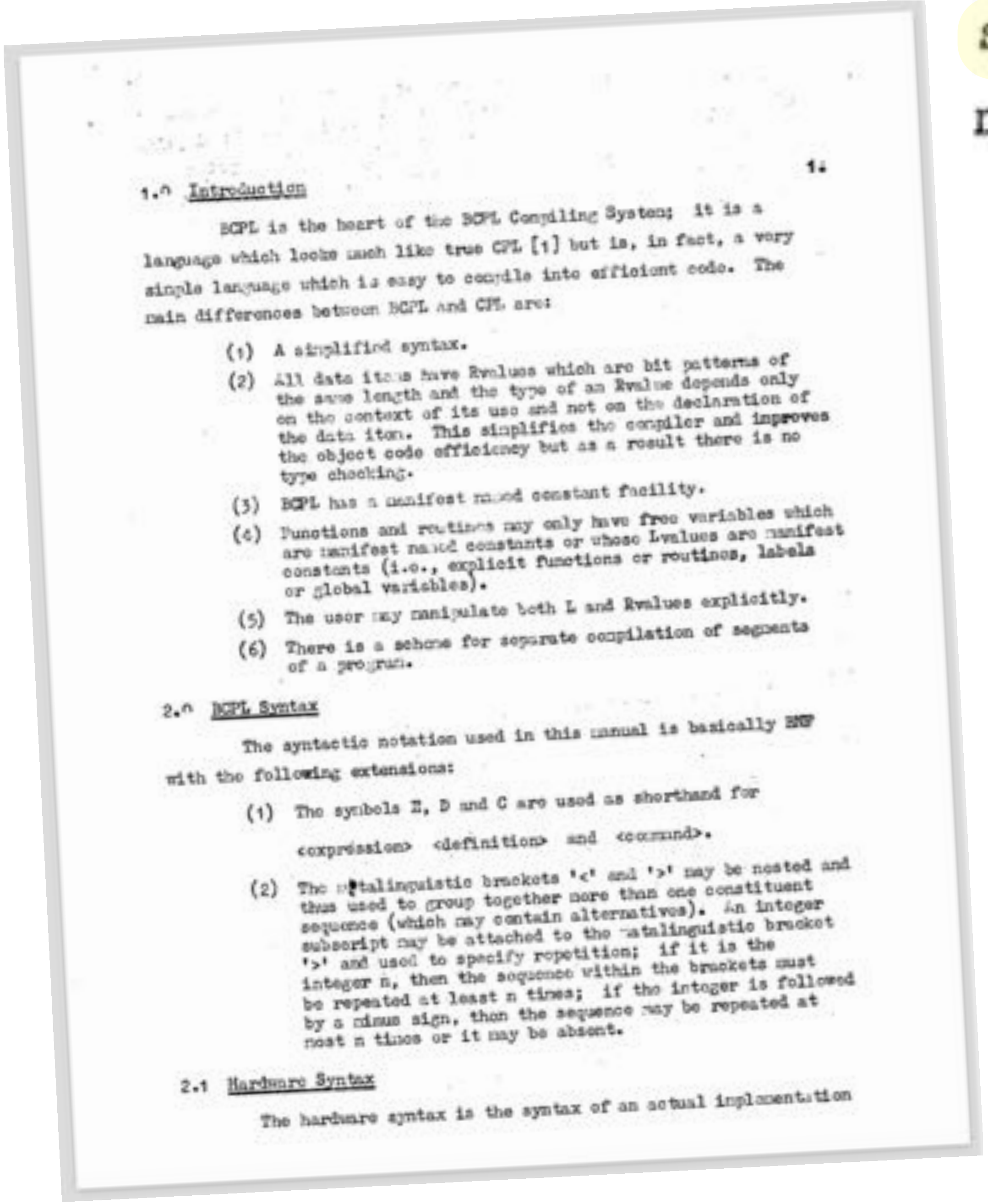
(2) All data items have Rvalues which are bit patterns of the same length and the type of an Rvalue depends only on the context of its use and not on the declaration of the data item. This simplifies the compiler and improves the object code efficiency but as a result there is no type checking.

(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.



1.0 Introduction

1.

BCPL is the heart of the BCPL Compiling System; it is a language which looks 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 looks much like true CPL [1] but is, in fact, a 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.

Lucky and humble fans meet Martin Richards, the inventor of BCPL

Computer Laboratory, Cambridge, December 2014

So what is the link between BCPL and B 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.

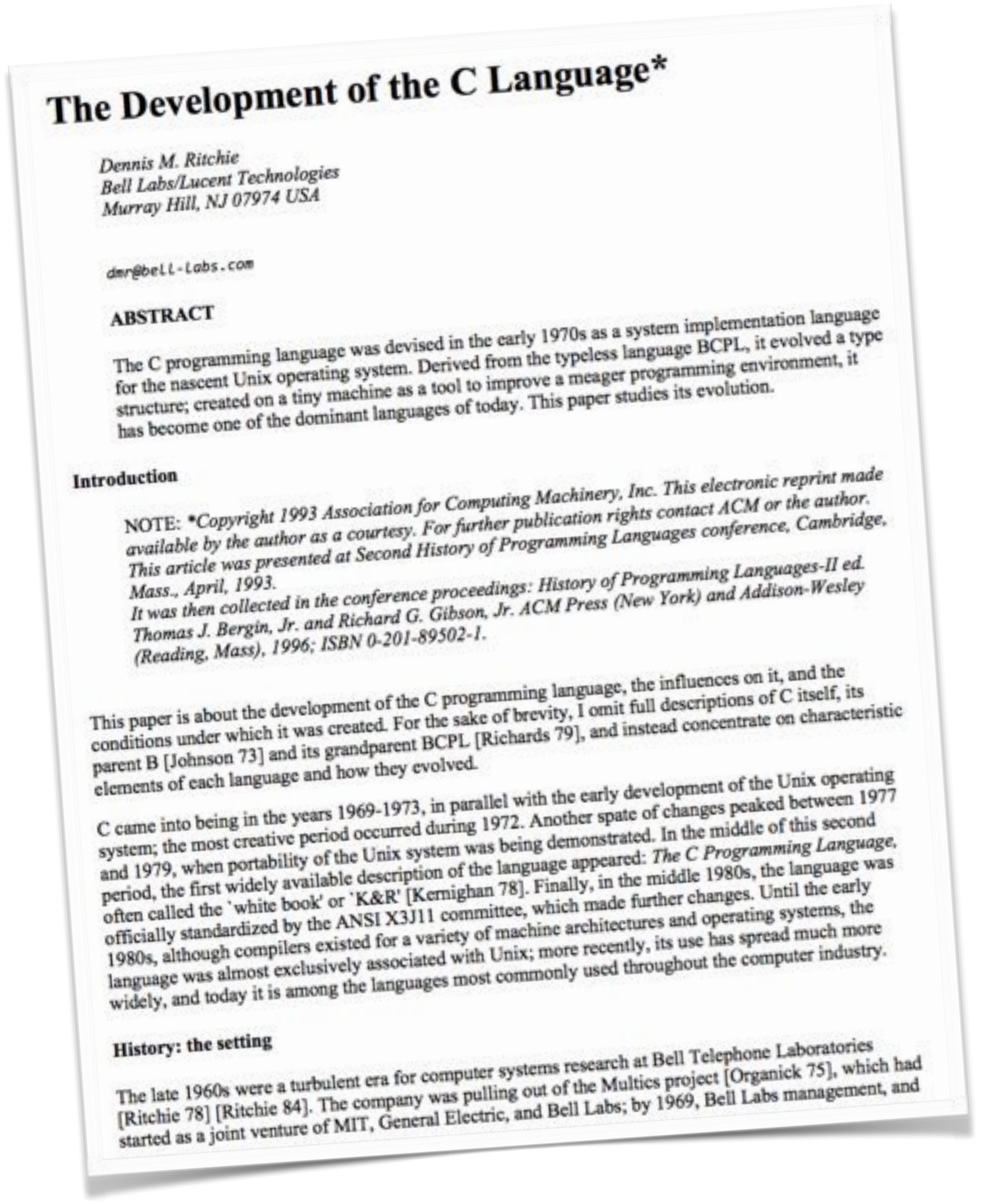
...

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

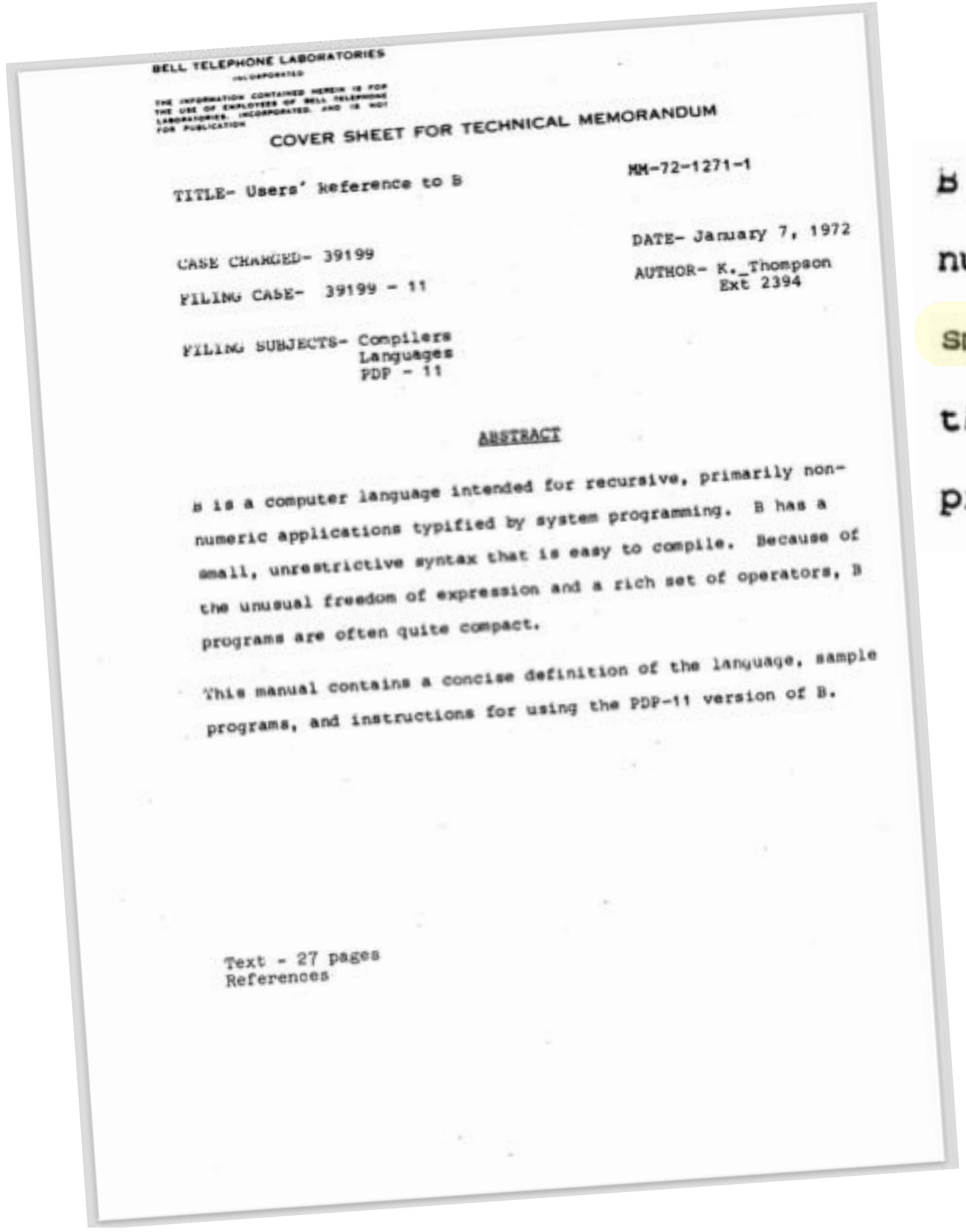
The late 1960s were a turbulent era for computer systems research at Bell Telephone Laboratories [Ritchie 78] [Ritchie 84]. The company was pulling out of the Multics project [Organick 75], which had started as a joint venture of MIT, General Electric, and Bell Labs; by 1969, Bell Labs management, and

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. This paper studies its evolution.

…

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

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

DATE- January 7, 1972

CASE CHARGED- 39199

FILING CASE- 39199 - 11

AUTHOR- K. Thompson
Ext 2394

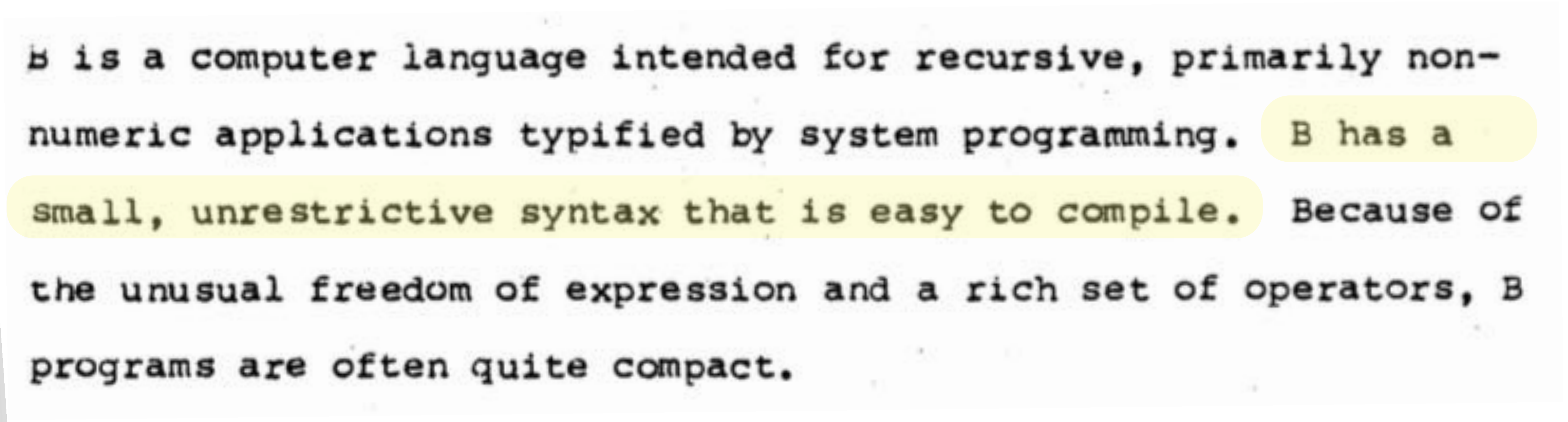
FILING SUBJECTS- Compilers
Languages
PDP - 11

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.

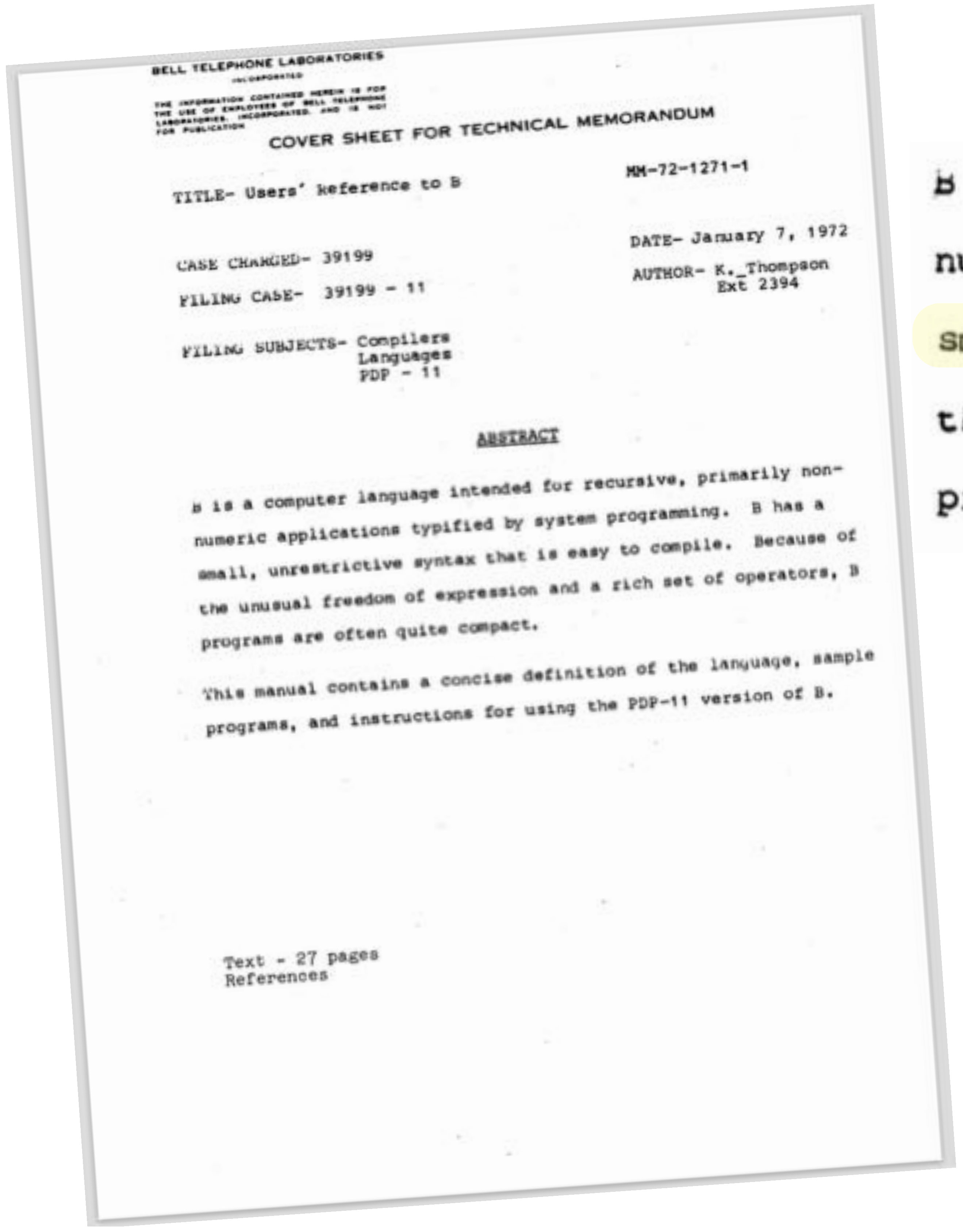
This manual contains a concise definition of the language, sample programs, and instructions for using the PDP-11 version of B.

Text - 27 pages
References



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.

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

DATE- January 7, 1972

CASE CHARGED- 39199

FILING CASE- 39199 - 11

AUTHOR- K._Thompson
Ext 2394

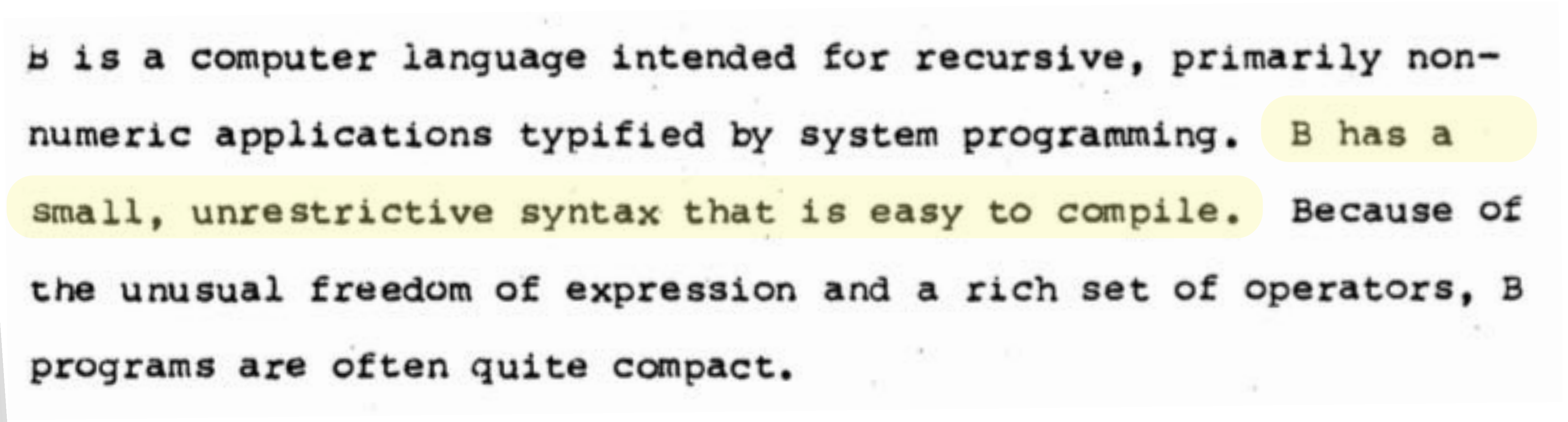
FILING SUBJECTS- Compilers
Languages
PDP - 11

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.

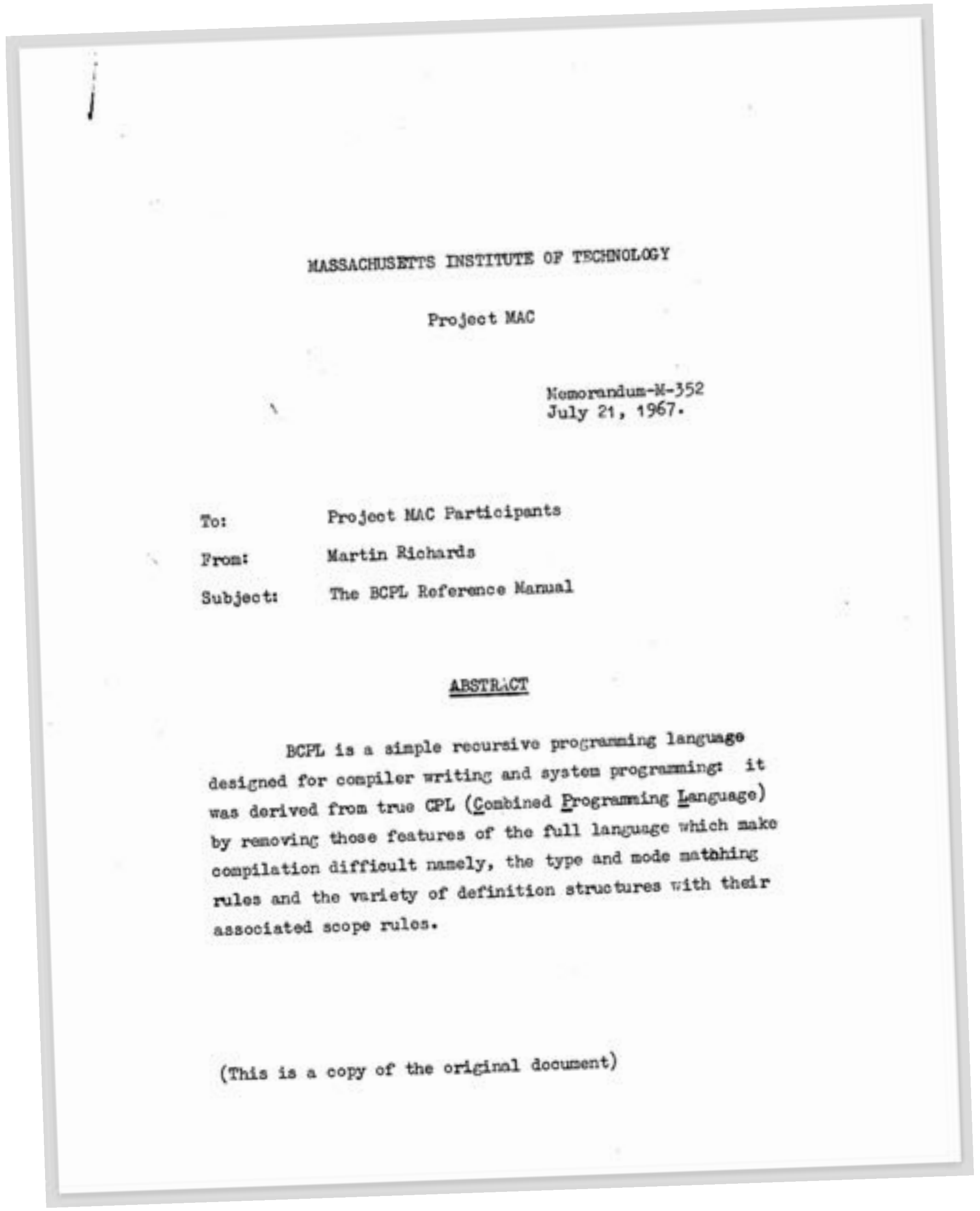
This manual contains a concise definition of the language, sample programs, and instructions for using the PDP-11 version of B.

Text - 27 pages
References



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.

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

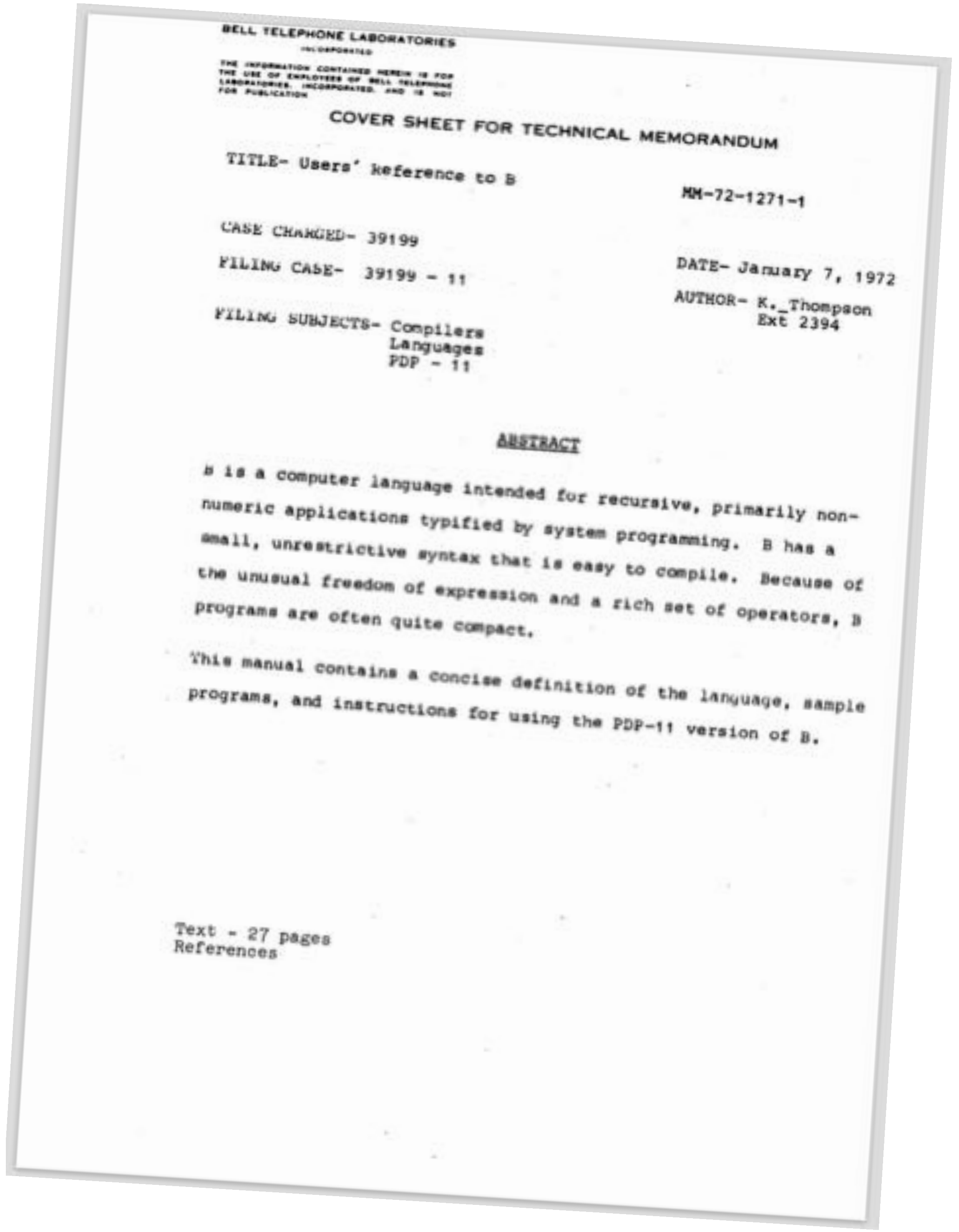
ABSTRACT

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

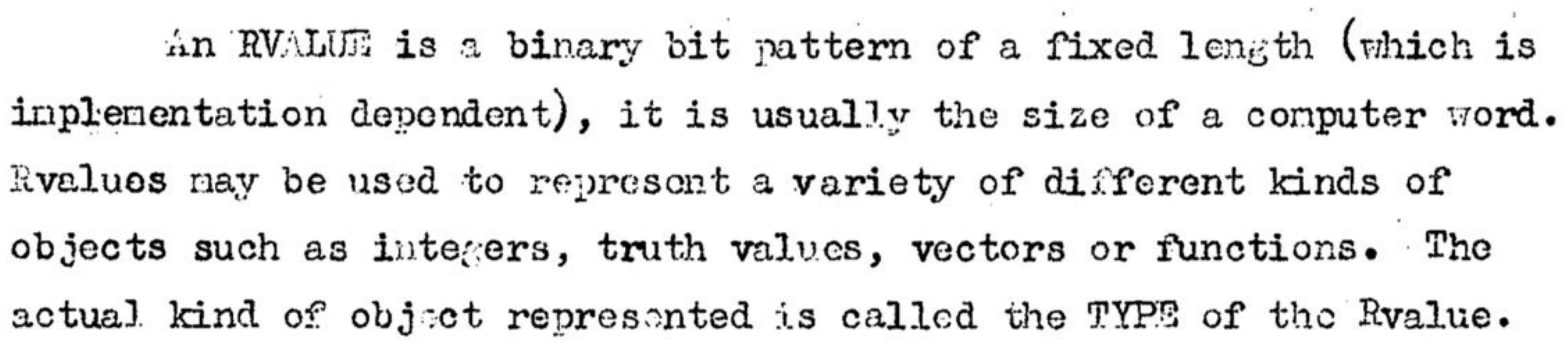
ABSTRACT

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.

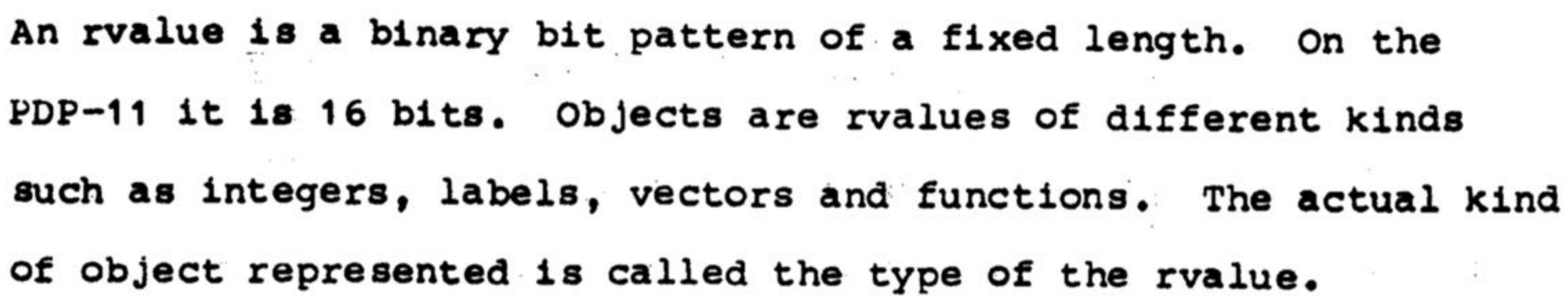
This manual contains a concise definition of the language, sample programs, and instructions for using the PDP-11 version of B.

Text - 27 pages
References

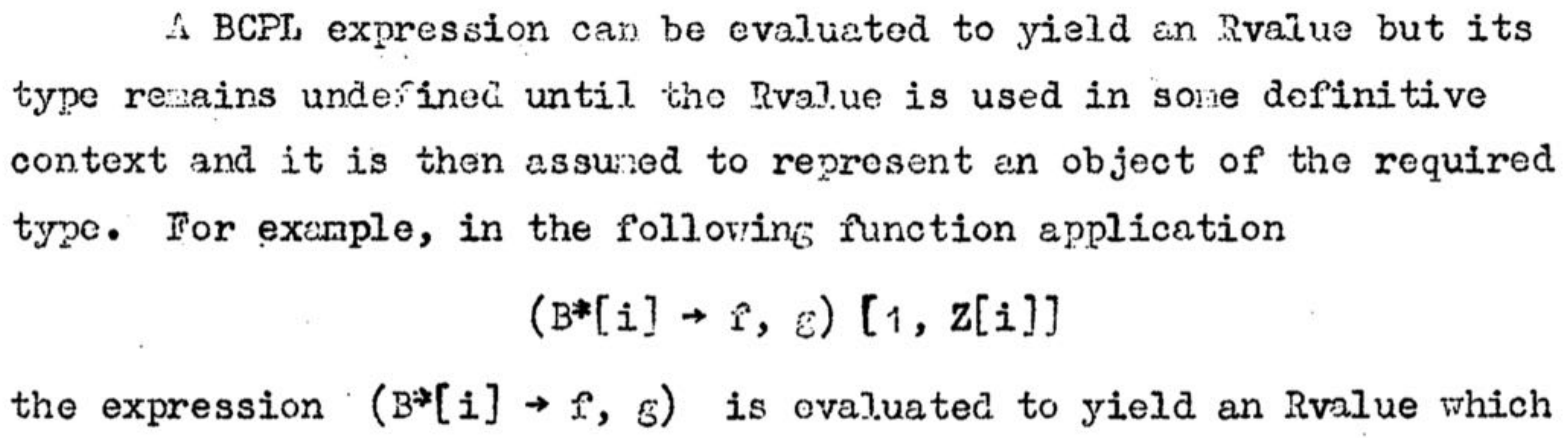
excerpt from the BCPL reference manual (Richards, 1967), page 6


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

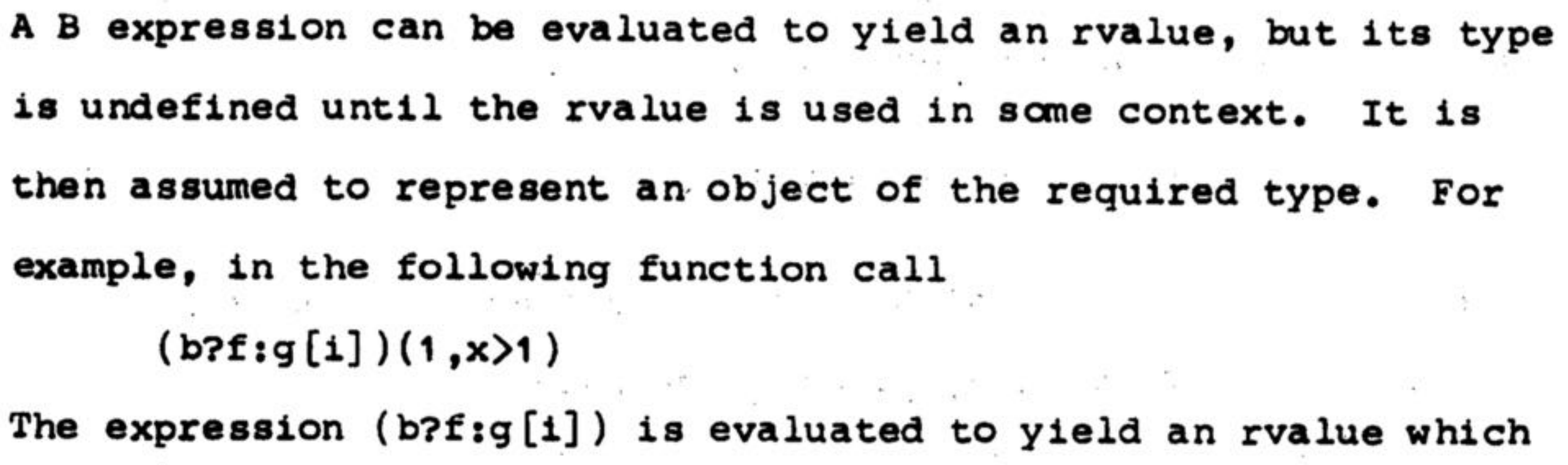


A BCPL expression can be evaluated to yield an Rvalue but its type remains undefined until the Rvalue is used in some definitive context and it is then assumed to represent an object of the required type. For example, in the following function application

```
(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

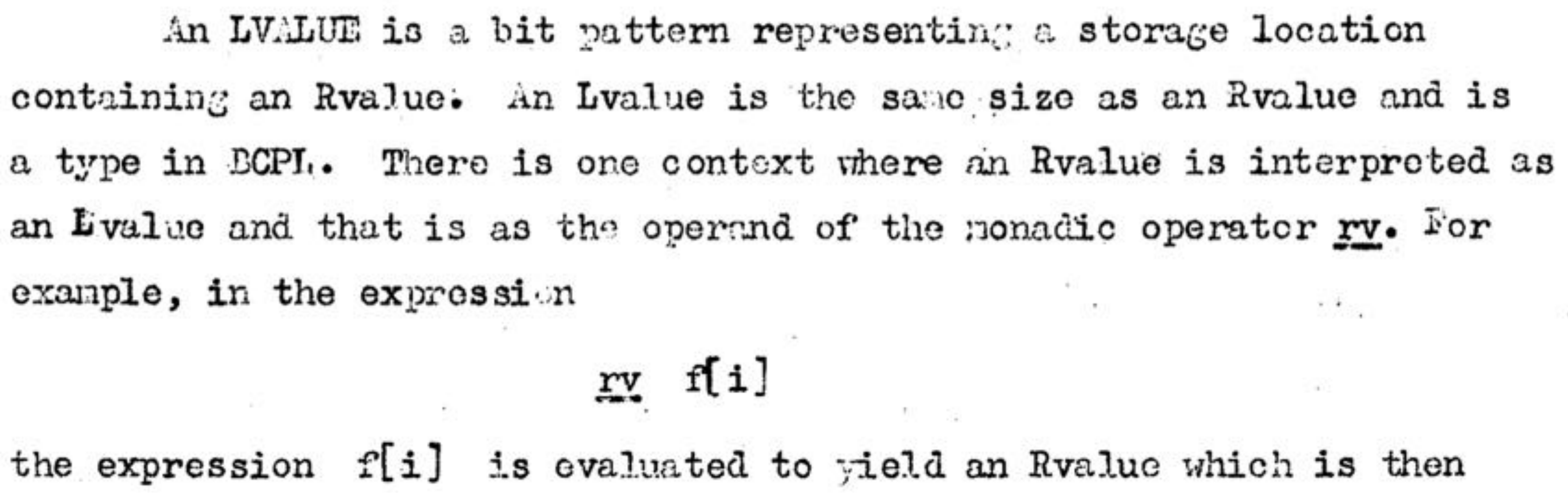


A B expression can be evaluated to yield an rvalue, but its type is undefined until the rvalue is used in some context. It is then assumed to represent an object of the required type. For example, in the following function call

```
(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

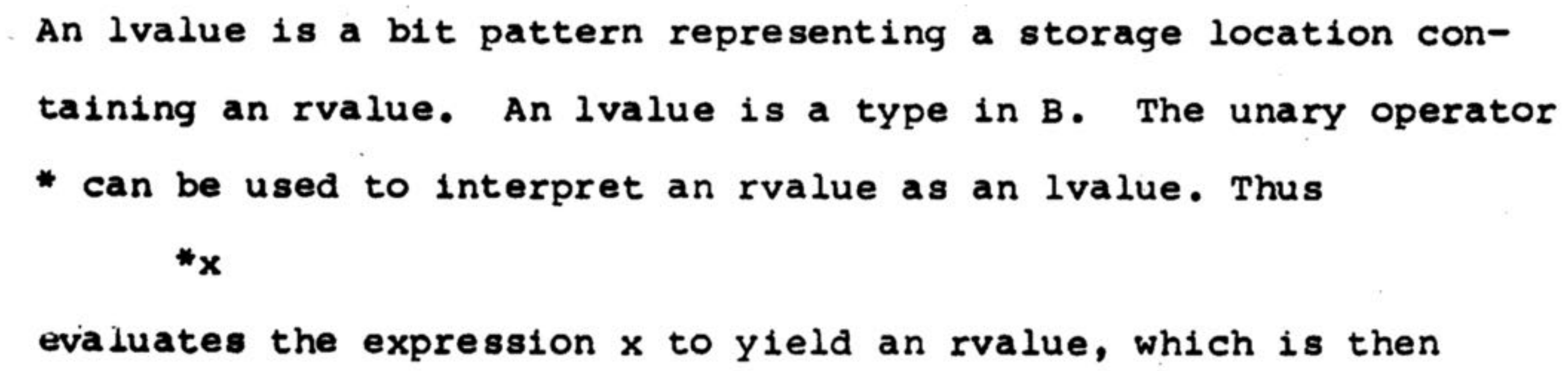


An LVALUE is a bit pattern representing a storage location containing an Rvalue. An Lvalue is the same size as an Rvalue and is a type in BCPL. There is one context where an Rvalue is interpreted as an Lvalue and that is as the operand of the monadic operator rv. For example, in the expression

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

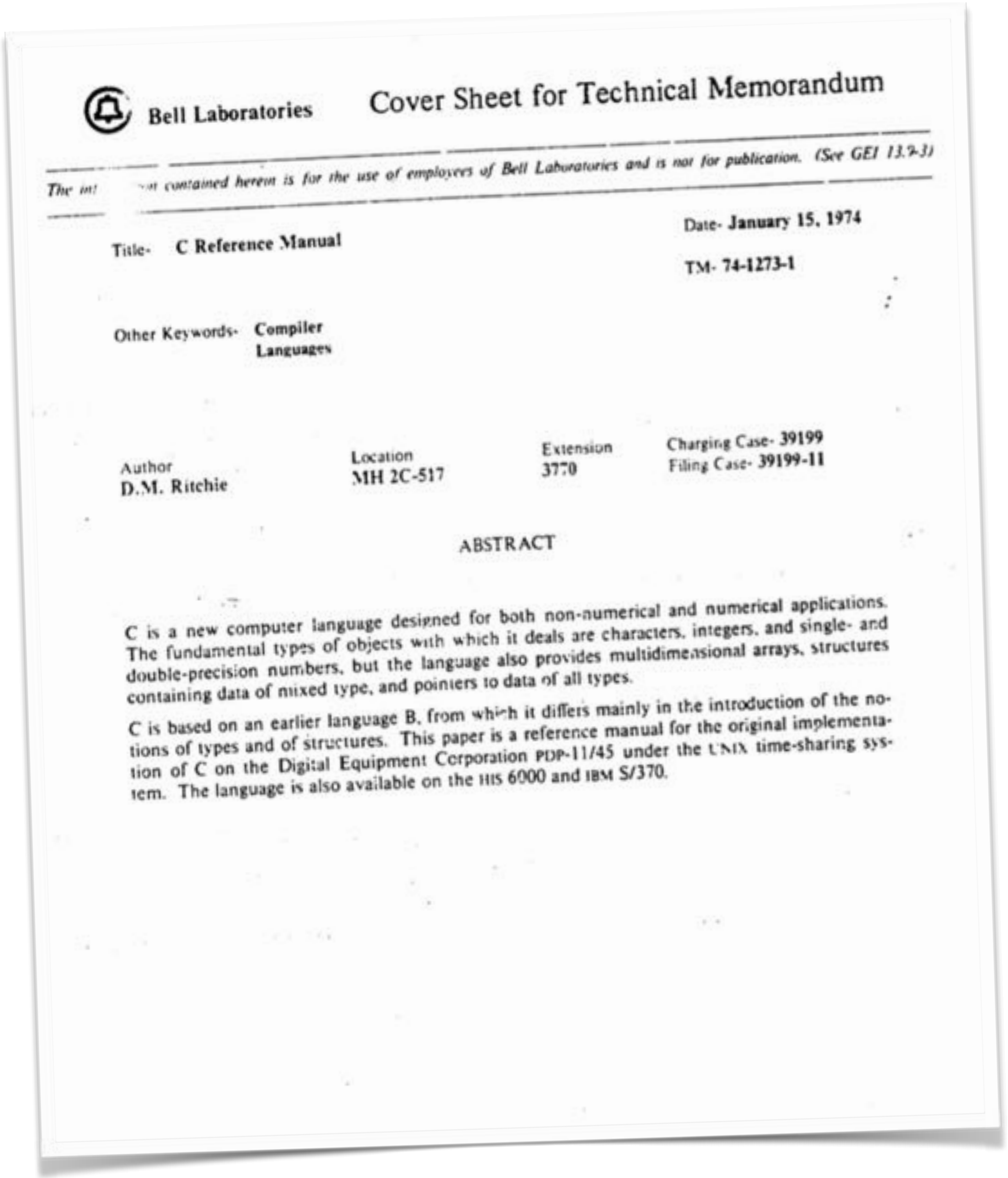


An lvalue is a bit pattern representing a storage location containing an rvalue. An lvalue is a type in B. The unary operator * can be used to interpret an rvalue as an lvalue. Thus

*x

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

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



Bell Laboratories Cover Sheet for Technical Memorandum

The information 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	Charging Case- 39199
D.M. Ritchie	MH 2C-517	3770	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.

Interesting fact:

Interesting fact:

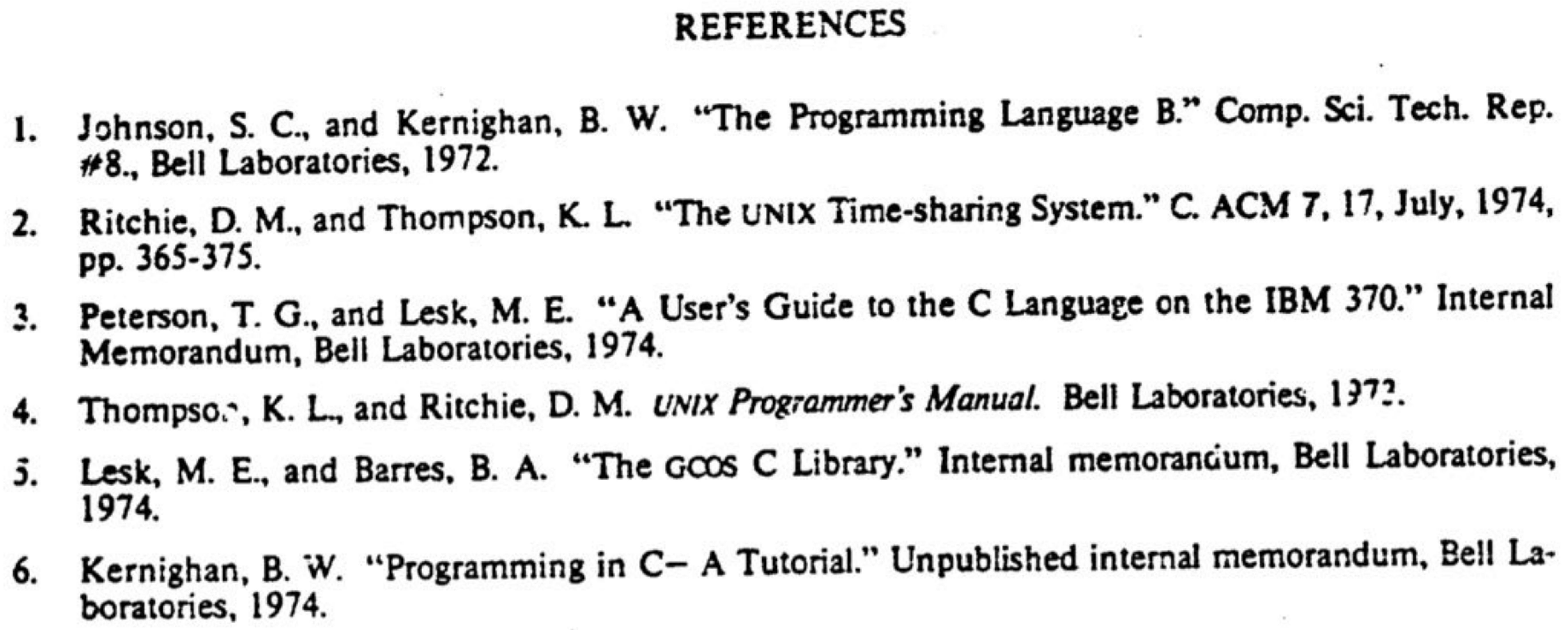
The C74 reference manual does not mention BCPL at all.

Interesting fact:

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

Interesting fact:

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



REFERENCES

1. Johnson, S. C., and Kernighan, B. W. "The Programming Language B." Comp. Sci. Tech. Rep. #8., Bell Laboratories, 1972.
2. Ritchie, D. M., and Thompson, K. L. "The UNIX Time-sharing System." C. ACM 7, 17, July, 1974, pp. 365-375.
3. Peterson, T. G., and Lesk, M. E. "A User's Guide to the C Language on the IBM 370." Internal Memorandum, Bell Laboratories, 1974.
4. Thompson, K. L., and Ritchie, D. M. *UNIX Programmer's Manual.* Bell Laboratories, 1972.
5. Lesk, M. E., and Barres, B. A. "The GCOS C Library." Internal memorandum, Bell Laboratories, 1974.
6. Kernighan, B. W. "Programming in C– A Tutorial." Unpublished internal memorandum, Bell Laboratories, 1974.

“Good artists copy. Great artists steal.”

Picasso?

```
good_research_labs(knowledge k);
great_research_labs(knowledge && k);

                                    /* Bell Labs? */
```

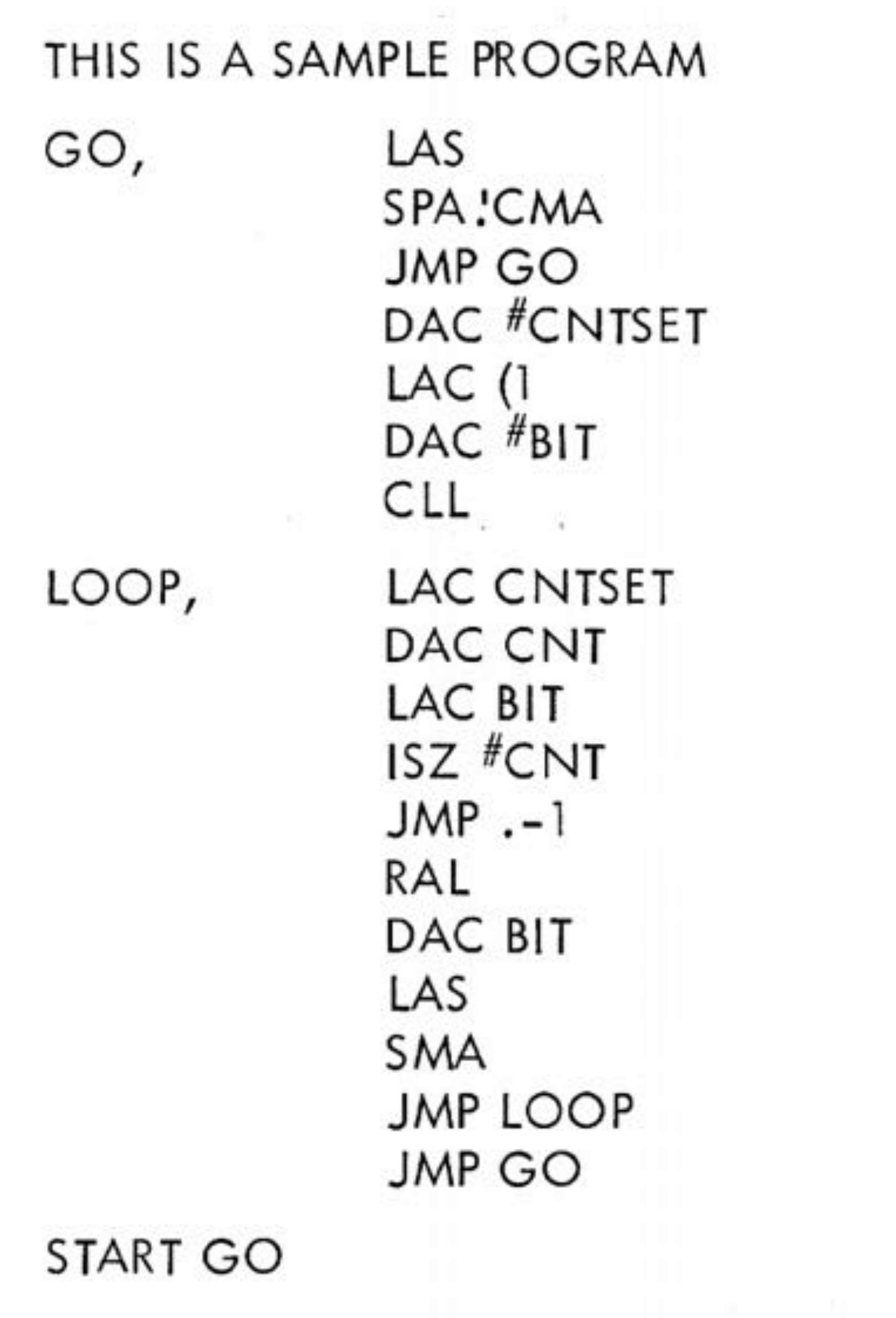
BCPL

- 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;
```

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
```

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
```

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

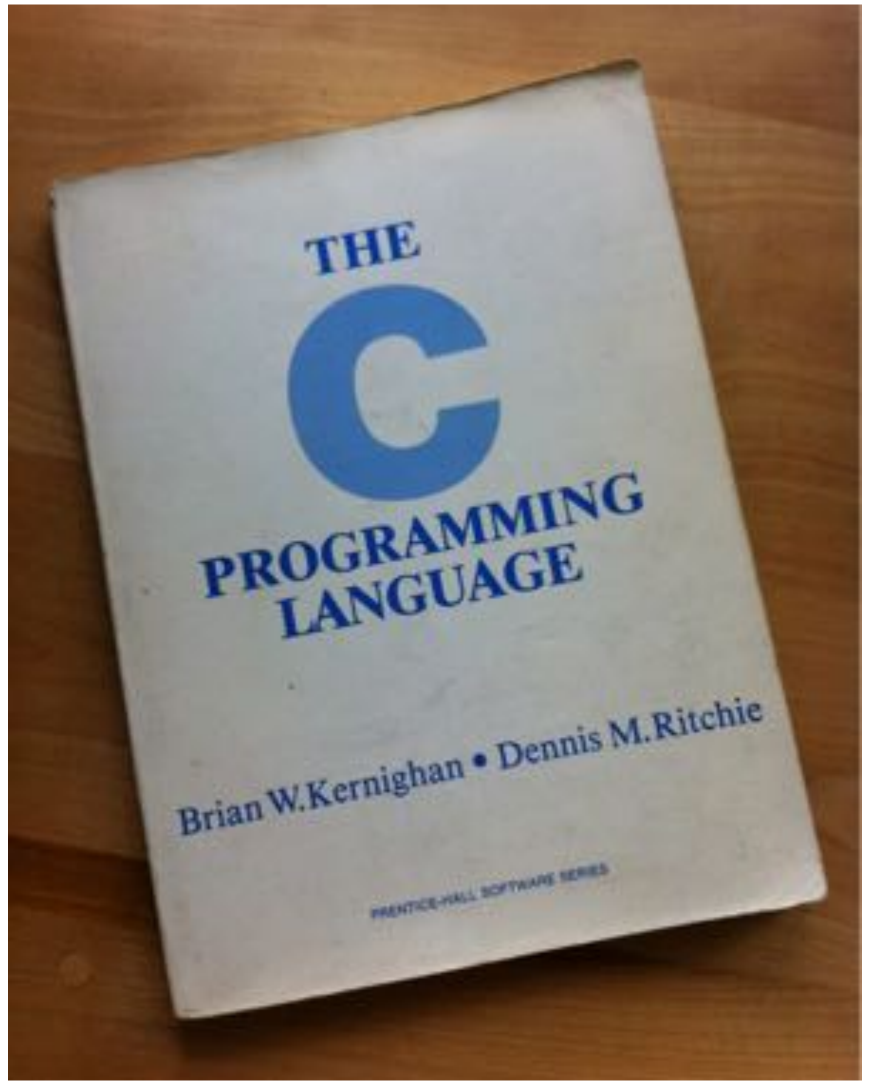
- 16-bit computer
- introduced 1970
- orthogonal instruction set
- byte-oriented

Early C

- Designed by Dennis Ritchie and Ken Thompson
- Developed during 1969-1972 in parallel with Unix
- Developed because of the PDP-11, a 16-bit, byte-oriented machine
- C introduced more types: integer types, characters and floating point types
- A key design principle was to make C amenable to translation by simple compilers
- Storage limitations often demanded a one-pass technique in which output was generated as soon as possible.
- While C had been ported to other architectures, until about 1977 Unix itself had only been running on DEC architectures.
- The PCC (Portable C Compiler, Stephen C. Johnson) was an important reference implementation
- It was not until 1977-1979 that the portability of Unix was demonstrated
- very productive time 1977-1979 for C as Unix was ported to new platforms

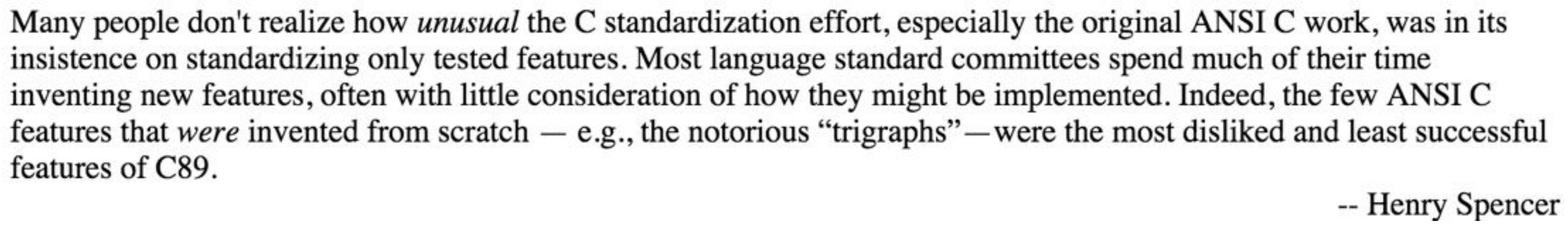
K&R C

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


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

```
/* C78 example, K&R C */

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, C78!";
    int len = mystrcpy(str1, str2);
    int i;
    for (i = 0; i < len; i++)
        putchar(str1[i]);
    exit(0);
}
```

Standardization of C started in 1983

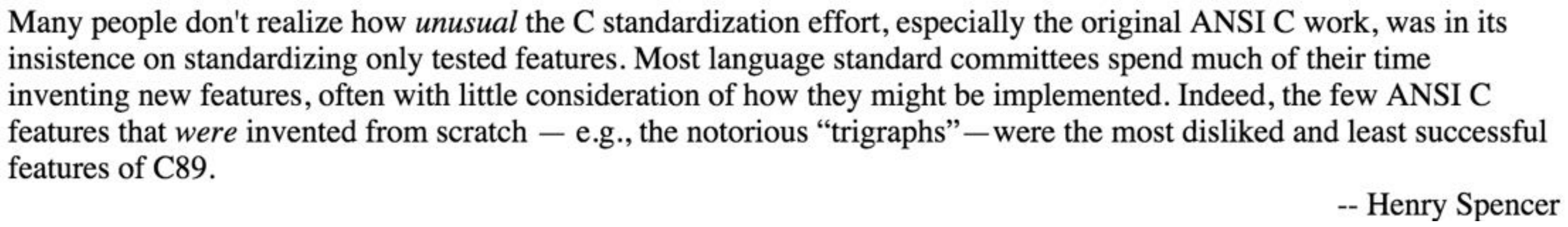


Many people don't realize how *unusual* the C standardization effort, especially the original ANSI C work, was in its insistence on standardizing only tested features. Most language standard committees spend much of their time inventing new features, often with little consideration of how they might be implemented. Indeed, the few ANSI C features that *were* invented from scratch — e.g., the notorious “trigraphs”—were the most disliked and least successful features of C89.

-- Henry Spencer

http://www.faqs.org/docs/artu/c_evolution.html

Standardization of C started in 1983



Many people don't realize how *unusual* the C standardization effort, especially the original ANSI C work, was in its insistence on standardizing only tested features. Most language standard committees spend much of their time inventing new features, often with little consideration of how they might be implemented. Indeed, the few ANSI C features that *were* invented from scratch — e.g., the notorious “trigraphs”—were the most disliked and least successful features of C89.

-- Henry Spencer

http://www.faqs.org/docs/artu/c_evolution.html

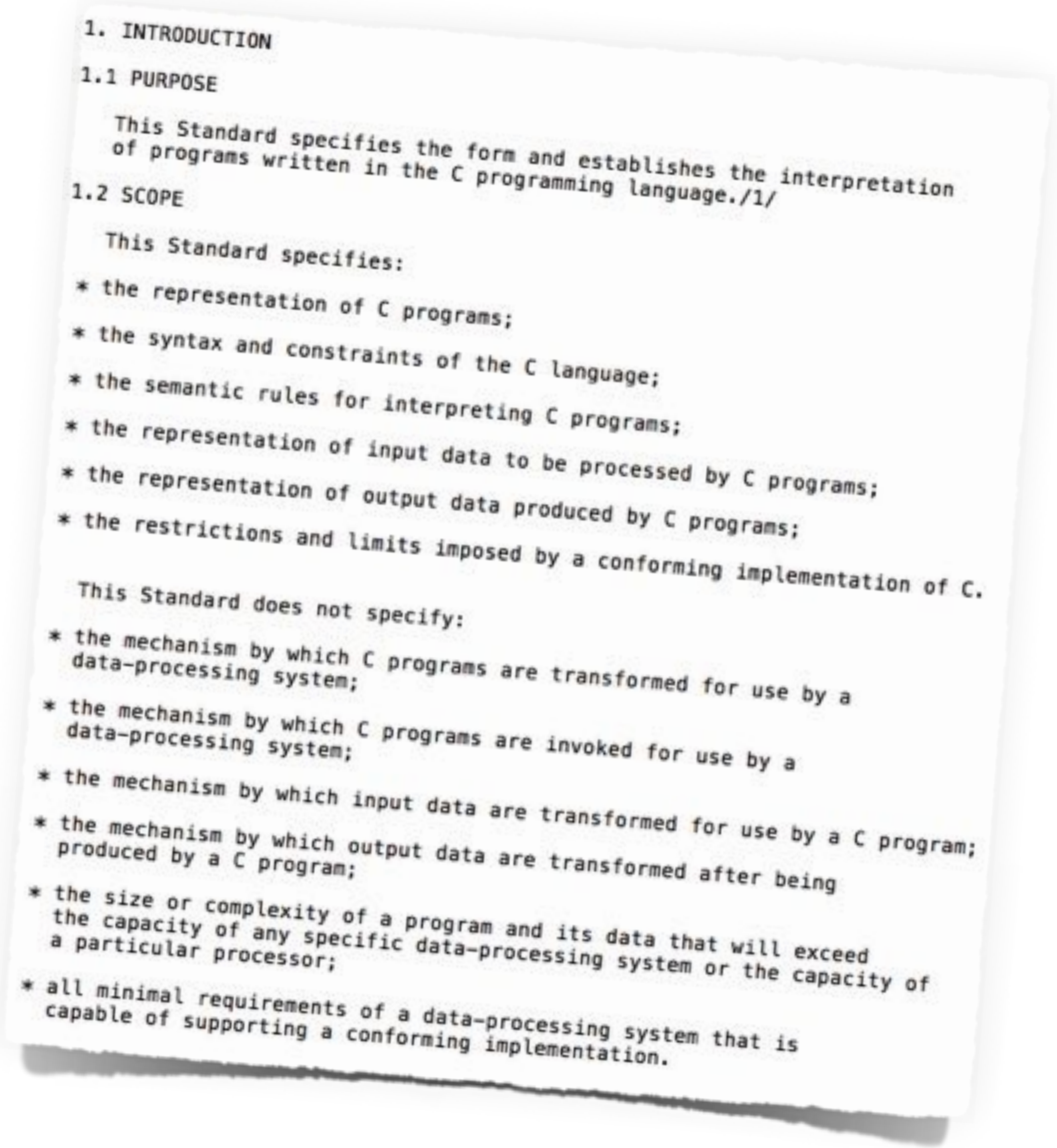
Standardization of C

- Dennis Ritchie not involved(except for the "noalias must go" article)
- Committee met four times a year, from 83 til publication
- All meetings in the US (due to political issues between ANSI and ISO)
- The committee avoided inventing features
- All features had to be demonstrated by one or more existing compilers
- Hot topic: value preserving vs unsigned preserving (value preserving won)
- The idea of text files vs binary files (due to Microsofts CR/NL vs Unix NL)
- The standard was delayed about 2 years due to a US protest

Private conversation with Tom Plum, April 2015

ANSI C / C89 / C90

ANSI published in 1989. ISO adopted in 1990 (but changed the chapter numbers).
Soon after it was all ISO/IEC



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.

```c
/* C89 example, ANSI C */

#include <stdio.h>

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

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

int main(void)
{
    char str1[10];
    char str2[] = "Hello, C89!";
    size_t len = mystrcpy(str1, str2);
    size_t i;
    for (i = 0; i < len; i++)
        putchar(str1[i]);
    return 0;
}
```

ISO/IEC 9899/AMD1:1995, aka "C95"

- Add more extensive support for international character sets (mostly done by Japan)
- Corrected some details

Private conversation with David Keaton, April 2015

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

[illegible]

```
// 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[10];
    char str2[] = "Hello, C99!";
    size_t len = mystrcpy(str1, str2);
    for (size_t i = 0; i < len; i++)
        putchar(str1[i]);
}
```

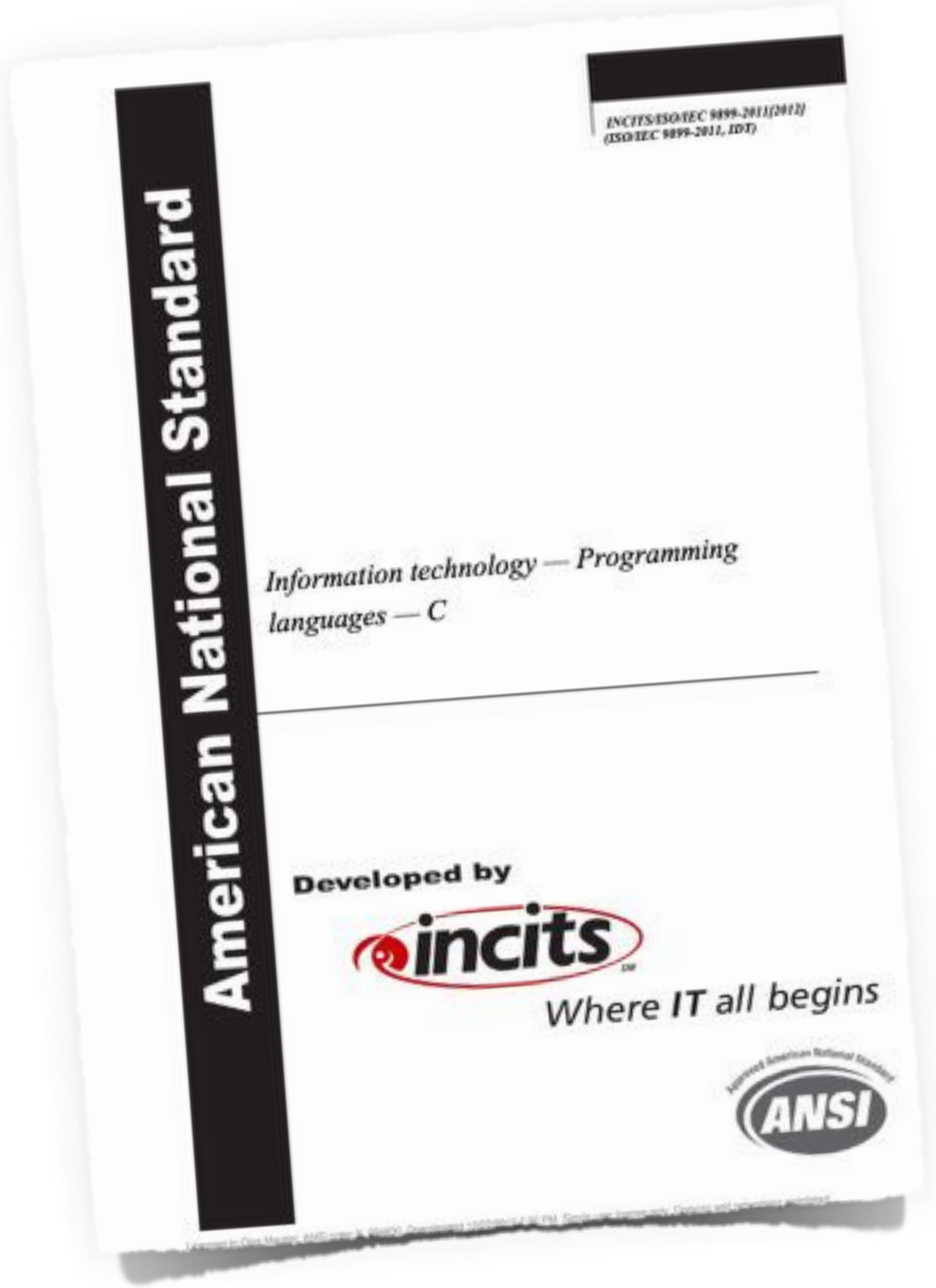
C11

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

ANSI

WG14 meeting at Lysaker, April 2015

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

K&R C

```c
/* C78 example, K&R C */

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, C78!";
    int len = mystrcpy(str1, str2);
    int i;
    for (i = 0; i < len; i++)
        putchar(str1[i]);
    exit(0);
}
```

C89/C90

```c
/* C89 example, ANSI C */

#include <stdio.h>

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

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

int main(void)
{
    char str1[10];
    char str2[] = "Hello, C89!";
    size_t len = mystrcpy(str1, str2);
    size_t i;
    for (i = 0; i < len; i++)
        putchar(str1[i]);
    return 0;
}
```

C99

```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[10];
    char str2[] = "Hello, C99!";
    size_t len = mystrcpy(str1, str2);
    for (size_t i = 0; i < len; i++)
        putchar(str1[i]);
}
```

Evolution of Keywords in C (1972-2011)

B (1972)

auto	goto	if
extrn	return	else
		while
		switch
		case

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

from B to C (1972-1974)

auto	goto	if
extrn	return	else
		while
		switch
		case

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

from B to C (1972-1974)

int	auto	goto	if
char	extrn	return	else
float			while
double			switch
struct			case

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

from B to C (1972-1974)

int	auto	goto	if
char	extrn	return	else
float	static		while
double	register		switch
struct			case

from B to C (1972-1974)

int	auto	goto	if
char	extrn	return	else
float	static	break	while
double	register	continue	switch
struct			case

from B to C (1972-1974)

int	auto	goto	if
char	extrn	return	else
float	static	break	while
double	register	continue	switch
struct			case
			default
			do
			for

from B to C (1972-1974)

int	auto	goto	if	sizeof
char	extrn	return	else	entry
float	static	break	while	
double	register	continue	switch	
struct			case	
			default	
			do	
			for	

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

from B to C (1972-1974)

int	auto	goto	if	sizeof
char	extrn	return	else	entry
float	static	break	while	
double	register	continue	switch	
struct			case	
			default	
			do	
			for	

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

Early C (1974)

int	auto	goto	if	sizeof
char	extern	return	else	entry
float	static	break	while	
double	register	continue	switch	
struct			case	
			default	
			do	
			for	

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

from Early C to K&R C (1974-1978)

int	auto	goto	if	sizeof
char	extern	return	else	entry
float	static	break	while	
double	register	continue	switch	
struct			case	
			default	
			do	
			for	

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

from Early C to K&R C (1974-1978)

int	auto	goto	if	sizeof
char	extern	return	else	entry
float	static	break	while	
double	register	continue	switch	
struct			case	
short			default	
long			do	
union			for	
unsigned				

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

from Early C to K&R C (1974-1978)

int	auto	goto	if	sizeof
char	extern	return	else	entry
float	static	break	while	typedef
double	register	continue	switch	
struct			case	
short			default	
long			do	
union			for	
unsigned				

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

K&R C (1978)

int	auto	goto	if	sizeof
char	extern	return	else	entry
float	static	break	while	typedef
double	register	continue	switch	
struct			case	
short			default	
long			do	
union			for	
unsigned				

K&R

from K&R C to ANSI C (1978-1989)

int	auto	goto	if	sizeof
char	extern	return	else	entry
float	static	break	while	typedef
double	register	continue	switch	
struct			case	
short			default	
long			do	
union			for	
unsigned				

from K&R C to ANSI C (1978-1989)

int	auto	goto	if	sizeof
char	extern	return	else	entry
float	static	break	while	typedef
double	register	continue	switch	
struct			case	
short			default	
long			do	
union			for	
unsigned				
signed				
enum				
void				

from K&R C to ANSI C (1978-1989)

int
char
float
double
struct
short
long
union
unsigned
signed
enum
void

auto
extern
static
register
volatile
const

goto
return
break
continue

if
else
while
switch
case
default
do
for

sizeof
entry
typedef

from K&R C to ANSI C (1978-1989)

int	auto	goto	if	sizeof
char	extern	return	else	~~entry~~
float	static	break	while	typedef
double	register	continue	switch	
struct	volatile		case	
short	const		default	
long			do	
union			for	
unsigned				
signed				
enum				
void				

from K&R C to ANSI C (1978-1989)

int	auto	goto	if	sizeof
char	extern	return	else	~~entry~~
float	static	break	while	typedef
double	register	continue	switch	
struct	volatile		case	
short	const		default	
long			do	
union			for	
unsigned				
signed				
enum				
void				

The `entry` keyword came from PL/I and allowed multiple entry points into a function. The keyword was implemented by some compilers but was never standardized. (stackoverflow.com/questions/254395)

ANSI C (1989)

int
char
float
double
struct
short
long
union
unsigned
signed
enum
void

auto
extern
static
register
volatile
const

goto
return
break
continue

if
else
while
switch
case
default
do
for

sizeof
typedef

from ANSI C to C99 (1989-1999)

int	auto	goto	if	sizeof
char	extern	return	else	typedef
float	static	break	while	
double	register	continue	switch	
struct	volatile		case	
short	const		default	
long			do	
union			for	
unsigned				
signed				
enum				
void				

from ANSI C to C99 (1989-1999)

_Bool				
_Complex				
_Imaginary				
int	auto	goto	if	sizeof
char	extern	return	else	typedef
float	static	break	while	
double	register	continue	switch	
struct	volatile		case	
short	const		default	
long			do	
union			for	
unsigned				
signed				
enum				
void				

from ANSI C to C99 (1989-1999)

_Bool				
_Complex				
_Imaginary				
int	auto	goto	if	sizeof
char	extern	return	else	typedef
float	static	break	while	
double	register	continue	switch	
struct	volatile		case	
short	const		default	
long	restrict		do	
union	inline		for	
unsigned				
signed				
enum				
void				

C99

_Bool
_Complex
_Imaginary
int
char
float
double
struct
short
long
union
unsigned
signed
enum
void

auto
extern
static
register
volatile
const
restrict
inline

goto
return
break
continue

if
else
while
switch
case
default
do
for

sizeof
typedef

from C99 to C11 (1999-2011)

_Bool				
_Complex				
_Imaginary				
int	auto	goto	if	sizeof
char	extern	return	else	typedef
float	static	break	while	
double	register	continue	switch	
struct	volatile		case	
short	const		default	
long	restrict		do	
union	inline		for	
unsigned				
signed				
enum				
void				

from C99 to C11 (1999-2011)

_Bool				
_Complex				
_Imaginary				
int	auto	goto	if	sizeof
char	extern	return	else	typedef
float	static	break	while	
double	register	continue	switch	
struct	volatile		case	
short	const		default	
long	restrict		do	
union	inline		for	
unsigned	_Alignas			
signed	_Atomic			
enum	_Thread_local			
void				

from C99 to C11 (1999-2011)

_Bool				
_Complex				
_Imaginary				
int	auto	goto	if	sizeof
char	extern	return	else	typedef
float	static	break	while	_Noreturn
double	register	continue	switch	_Static_assert
struct	volatile		case	_Alignof
short	const		default	_Generic
long	restrict		do	
union	inline		for	
unsigned	_Alignas			
signed	_Atomic			
enum	_Thread_local			
void				

C11

_Bool				
_Complex				
_Imaginary				
int	auto	goto	if	sizeof
char	extern	return	else	typedef
float	static	break	while	_Noreturn
double	register	continue	switch	_Static_assert
struct	volatile		case	_Alignof
short	const		default	_Generic
long	restrict		do	
union	inline		for	
unsigned	_Alignas			
signed	_Atomic			
enum	_Thread_local			
void				

C11_Standard

The spirit of C

trust the programmer

- let them do what needs to be done
- the programmer is in charge not the compiler

keep the language small and simple

- small amount of code → small amount of assembler
- provide only one way to do an operation
- new inventions are not entertained

make it fast, even if its not portable

- target efficient code generation
- int preference, int promotion rules
- sequence points, maximum leeway to compiler

rich expression support

- lots of operators
- expressions combine into larger expressions

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

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

The history of C

http://thesourceny.com/welcome/wp-content/uploads/2014/02/stage-red-curtain.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 pure assembler of course. In about 4 weeks! 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, and later ported to many other machines aided by Steve Johnsons Portable C Compiler. C gained popularity outside the realm of PDP-11 and Unix. Initially the K&R was the definitive reference until the language was standardized by ANSI and ISO in 1989/1990 and thereafter updated in 1999 and 2011.

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

!

?

C++

History and Spirit of C++

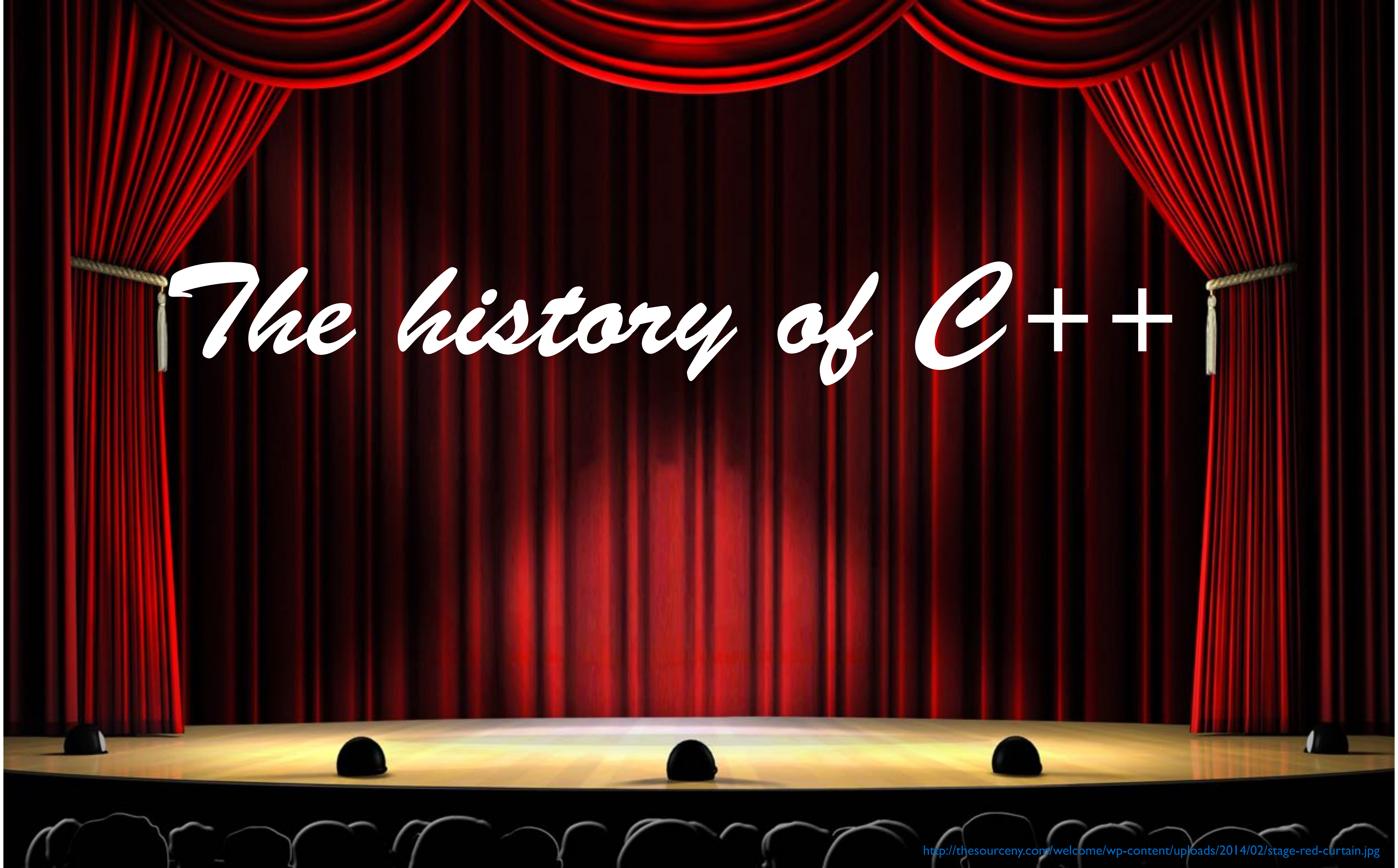
Olve Maudal

To get a deep understanding of C++, it is useful to know the history of this wonderful programming language. It is perhaps even more important to appreciate the driving forces, motivation and the spirit that has shaped this languages into what we have today.

We assume you know the history and spirit of C. We will now include Simula, Algol 68, Ada, ML, Clu into the equation. We will discuss the motivation for creating C++, and with live coding we will demonstrate by example how it has evolved from the rather primitive "C with Classes" into a supermodern and capable programming language as we now have with C++11/14 and soon with C++17.

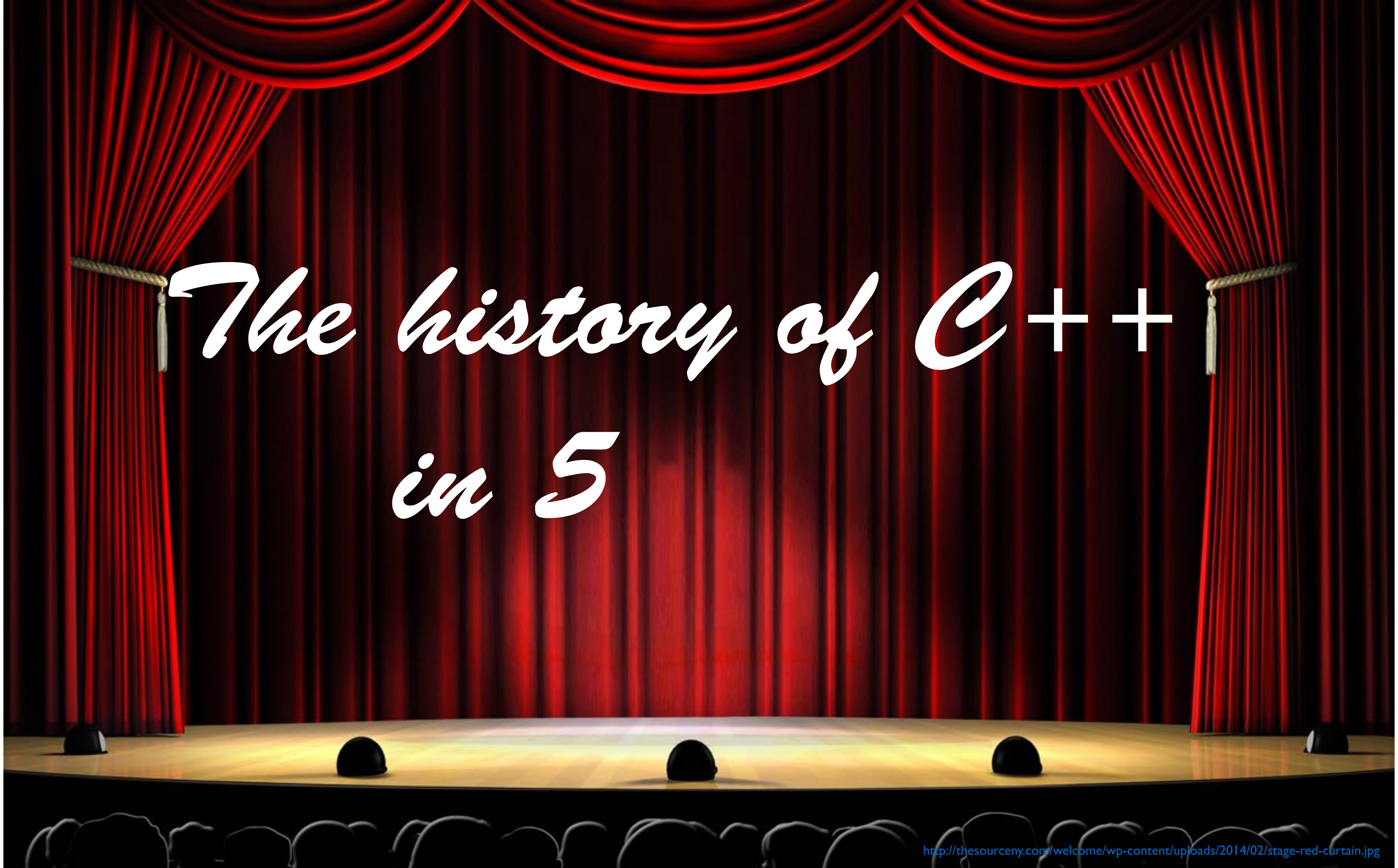
A lightning talk at ACCU 2015, April 23, Bristol, UK

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



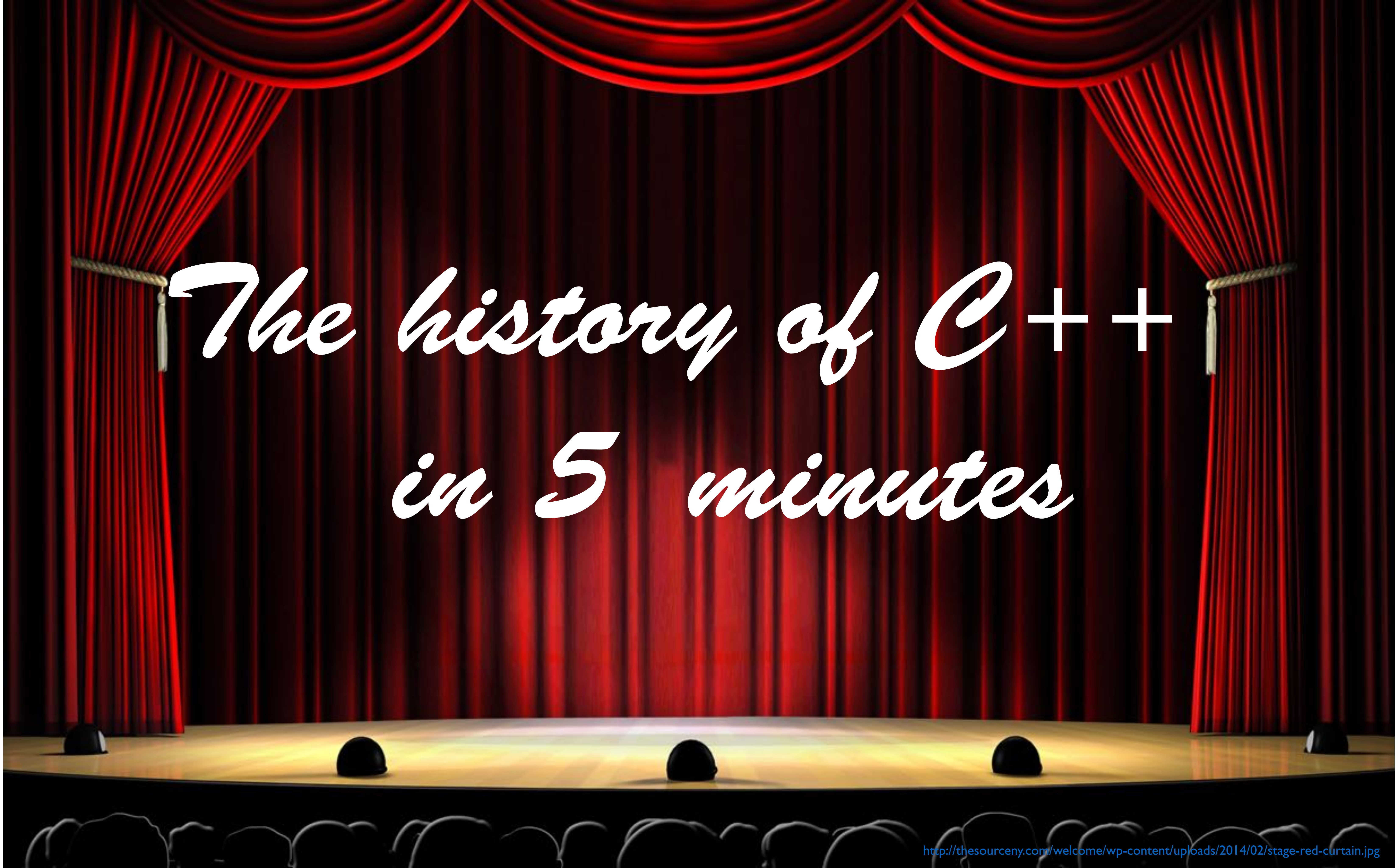
The history of C++

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



The history of C++ in 5

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



The history of C++ in 5 minutes

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

Before C++

with approximately the words of Bjarne Stroustrup himself as copied from "The Design and Evolution of C++", Bjarne Stroustrup, 1994

I was working on my PhD thesis

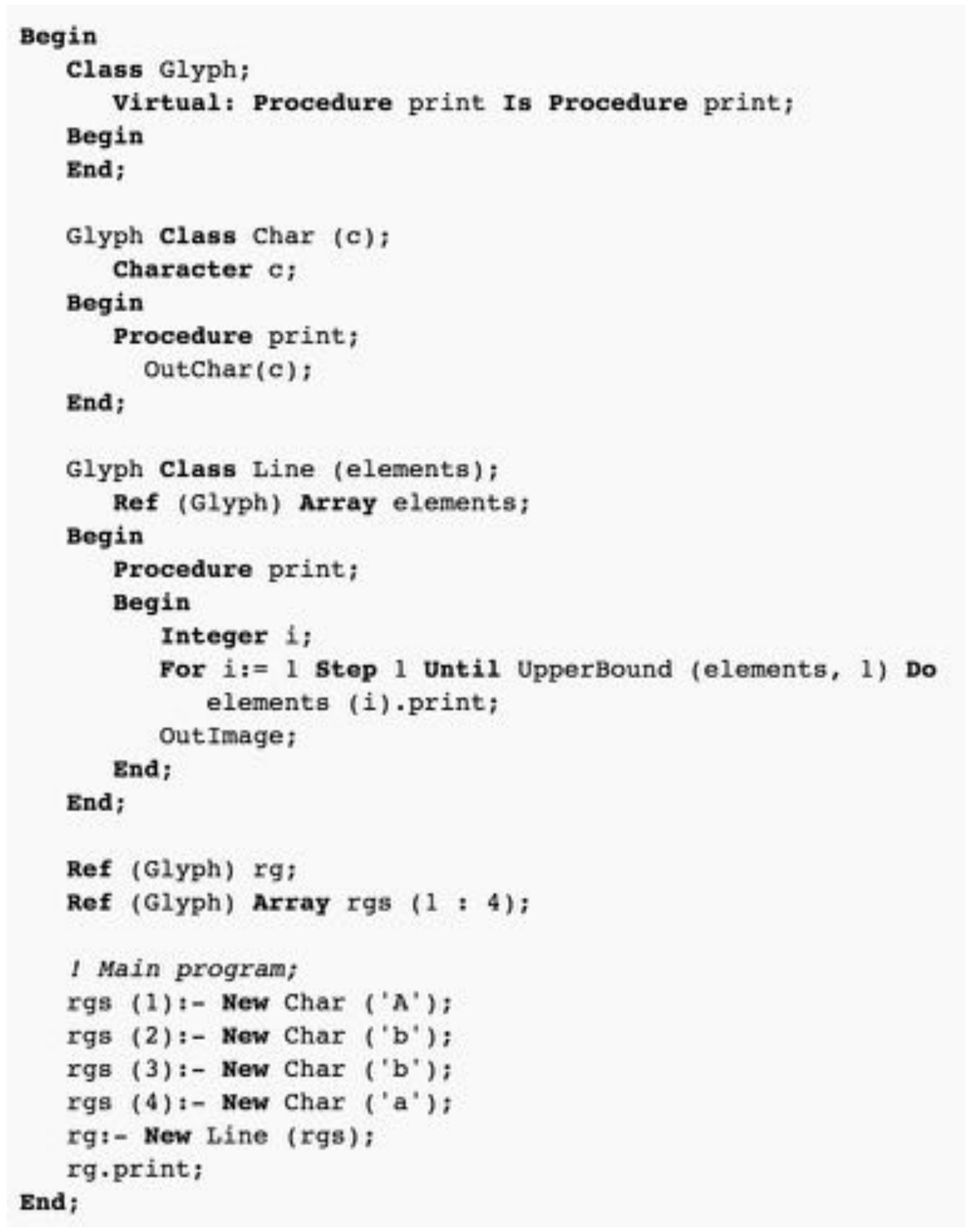
Bjarne

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

http://computersweden.idg.se/polopoly_fs/1.346563!imageManager/1326219611.jpg in the Computing Laboratory at

in the Computing Laboratory at University of Cambridge.

I 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



```
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;
```

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, I had to rewrite the simulator in ? and run it on the experimental CAP computer.

Eventually, I had 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. BCPL makes C look like a very high-level language and provides absolutely no type checking or run-time support.

http://theleafsnation.com/2011/2/7/phil-kessels-frustrated-you-would-be-too

The experience of coding and debugging the simulator in BCPL was horrible. BCPL makes C look like a very high-level language and provides absolutely no type checking or run-time support.

http://theleafsnation.com/2011/2/7/phil-kessels-frustrated-you-would-be-too

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.

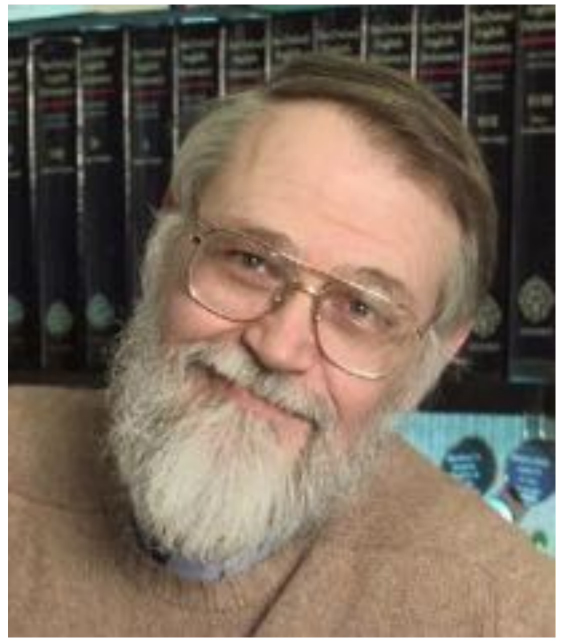
A good tool should:

- have support for program organization, eg classes, concurrency, strong type checking
- produce programs that run as fast as the BCPL programs
- support separately compiled units into a program
- allow for highly portable implementations

After finishing my PhD Thesis in Cambridge I got a job at

After finishing my PhD Thesis in Cambridge I got a job at Bell Labs.

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



Developing the initial version of C++ (pre-1985)

(p44, TDEC++)

- Simula gave classes

- Simula gave classes
- Algol68 gave operator overloading and references

- Simula gave classes
- Algol68 gave operator overloading and references
- Algol68 also gave the ability to declare variables anywhere in a block

- Simula gave classes
- Algol68 gave operator overloading and references
- Algol68 also gave the ability to declare variables anywhere in a block
- The only direct influence from BCPL was

- Simula gave classes
- Algol68 gave operator overloading and references
- Algol68 also gave the ability to declare variables anywhere in a block
- The only direct influence from BCPL was // comments

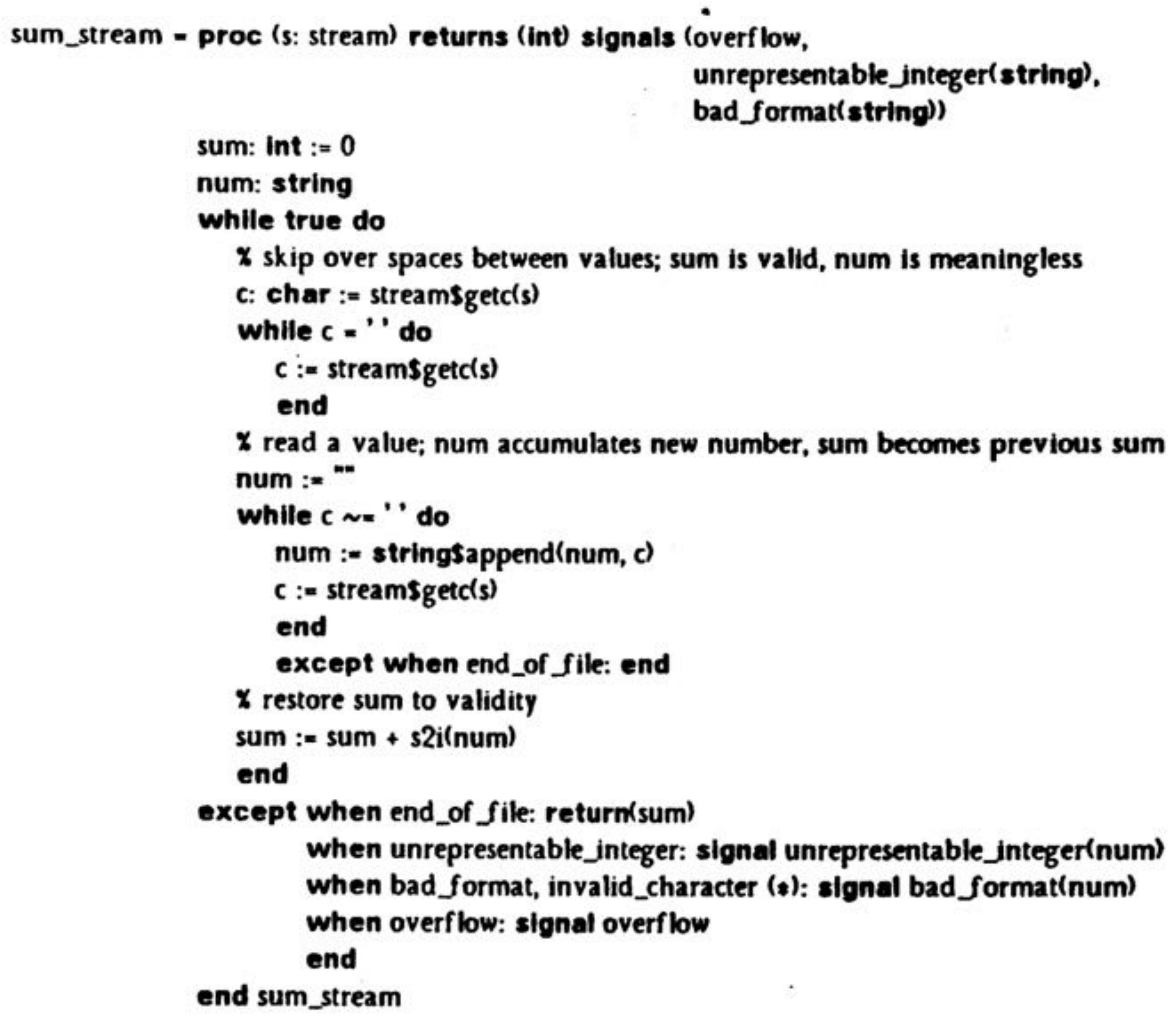
Development of C++ (post-1985)

(p45, TDEC++)

ML (Robin Milner, 1973) influenced exceptions

```
fun factorial n = let
  fun fac (0, acc) = acc
    | fac (n, acc) = fac (n - 1, n * acc)
  in
    if (n < 0) then raise Fail "negative argument"
    else fac (n, 1)
  end
```

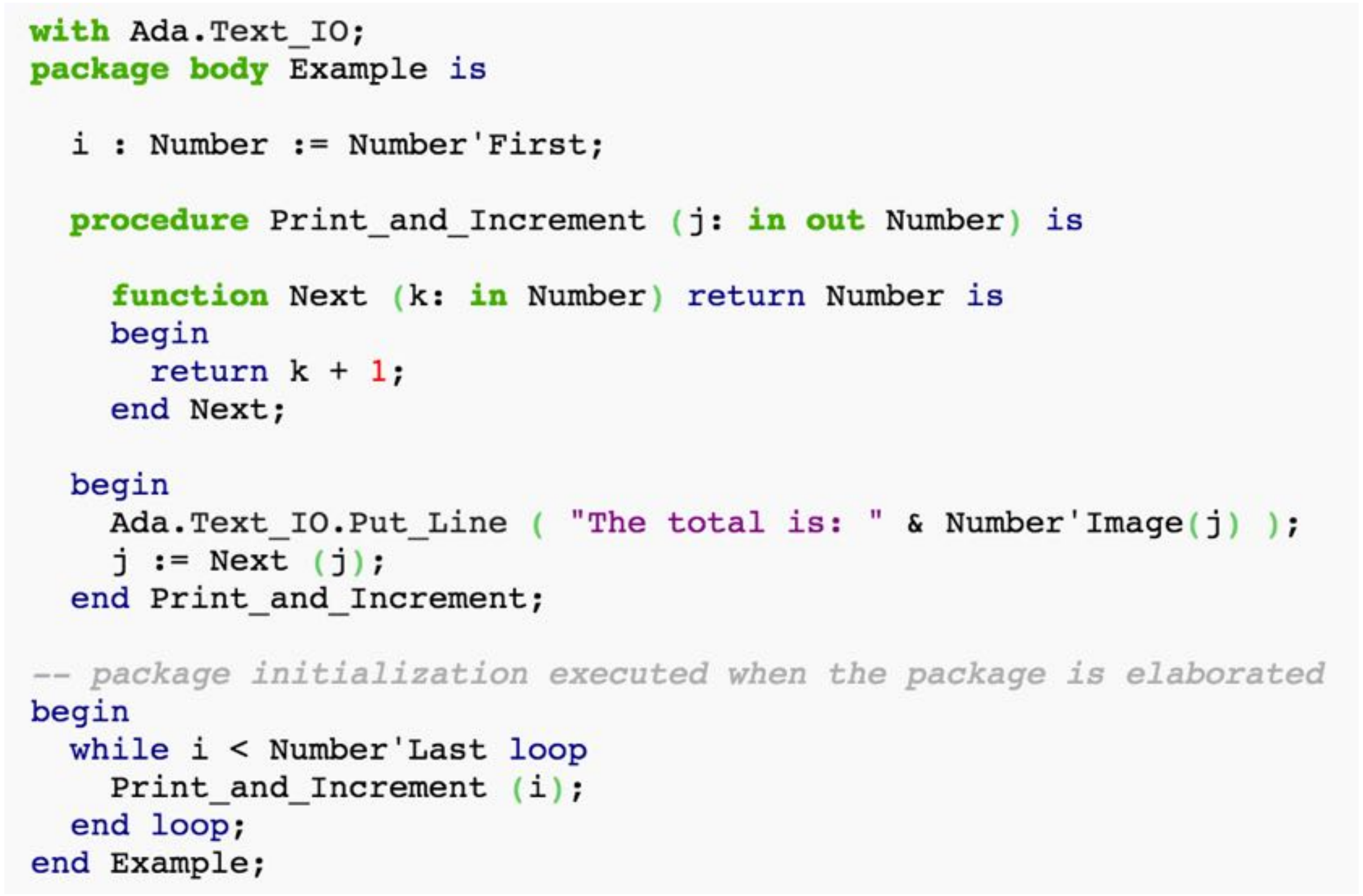
CLU (Barbara Liskov, 1974) also influenced exception



```
sum_stream = proc (s: stream) returns (int) signals (overflow,
                                                     unrepresentable_integer(string),
                                                     bad_format(string))
         sum: int := 0
         num: string
         while true do
             % skip over spaces between values; sum is valid, num is meaningless
             c: char := stream$getc(s)
             while c = ' ' do
                 c := stream$getc(s)
                 end
             % read a value; num accumulates new number, sum becomes previous sum
             num := ""
             while c ~= ' ' do
                 num := string$append(num, c)
                 c := stream$getc(s)
                 end
                 except when end_of_file: end
             % restore sum to validity
             sum := sum + s2i(num)
             end
         except when end_of_file: return(sum)
                when unrepresentable_integer: signal unrepresentable_integer(num)
                when bad_format, invalid_character (*): signal bad_format(num)
                when overflow: signal overflow
                end
         end sum_stream
```

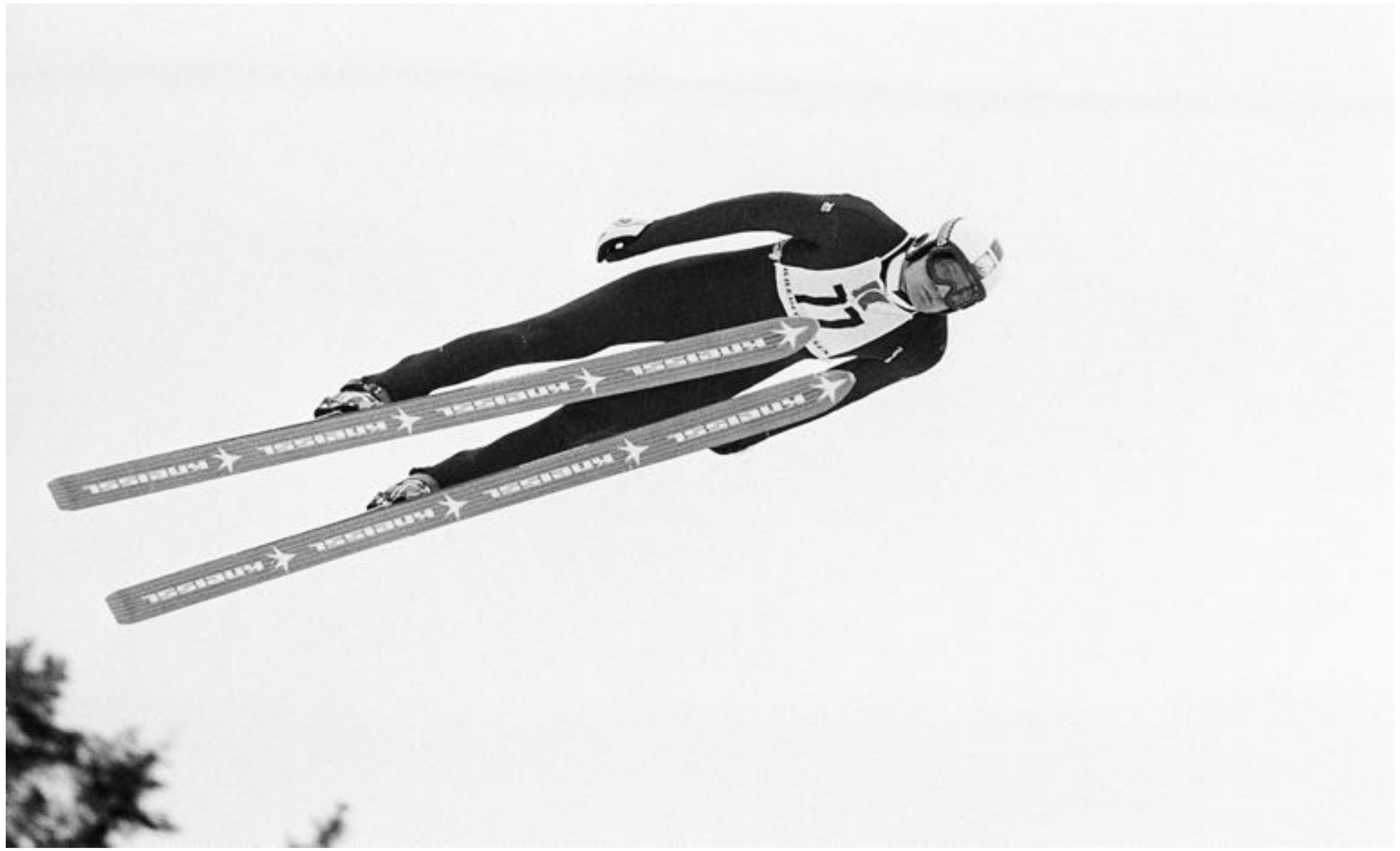
http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-225.pdf

Ada (Jean Ichbiah++, 1980) influenced templates, namespaces and exceptions



```ada
with Ada.Text_IO;
package body Example is

  i : Number := Number'First;

  procedure Print_and_Increment (j: in out Number) is

    function Next (k: in Number) return Number is
    begin
      return k + 1;
    end Next;

  begin
    Ada.Text_IO.Put_Line ( "The total is: " & Number'Image(j) );
    j := Next (j);
  end Print_and_Increment;

-- package initialization executed when the package is elaborated
begin
  while i < Number'Last loop
    Print_and_Increment (i);
  end loop;
end Example;
```

80's
C with classes, C++/CFront, ARM



C++ was improved and became standardized

90's
X3J16, C++arm, WG21, C++98, STL

Ouch... Template Metaprogramming

C++03, TR1, Boost and other external libraries

While the language itself saw some minor improvements after C++98, Boost and other external libraries acted like laboratories for experimenting with potential new C++ features. Resulting in...

C++11/C++14

With the latest version C++ feels like a new language

The future of C++?

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

!

?