History and Spirit of C and C++

Olve Maudal


https://c1.staticflickr.com/1/118/300053732_0b20ed7e73.jpg

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 60 minute session at NDC 2015, June 17, Oslo, Norway

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 60 minute session at NDC 2015, June 17, Oslo, Norway

Part I

History and spirit of C

- The short version
- Before C
- Early C and K&R
- ANSI C
- C99
- Modern C (aka C11)

Part II

History and spirit of C++

- Birth of C++
- Evolution of C++ by example
- C with Classes
- C++ in the 80's
- C++ in the 90's (aka ARM C++)
- Standard C++ (aka C++98)
- Modern C++ (aka C11/14)

C


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 60

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

The history of C in 60 seconds

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


The history of C
in ~~60~~ seconds
http://thesourceny.com/welcome/wp-content/uploads/2014/02/stage-red-curtain.jpg

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

At Bell Labs.

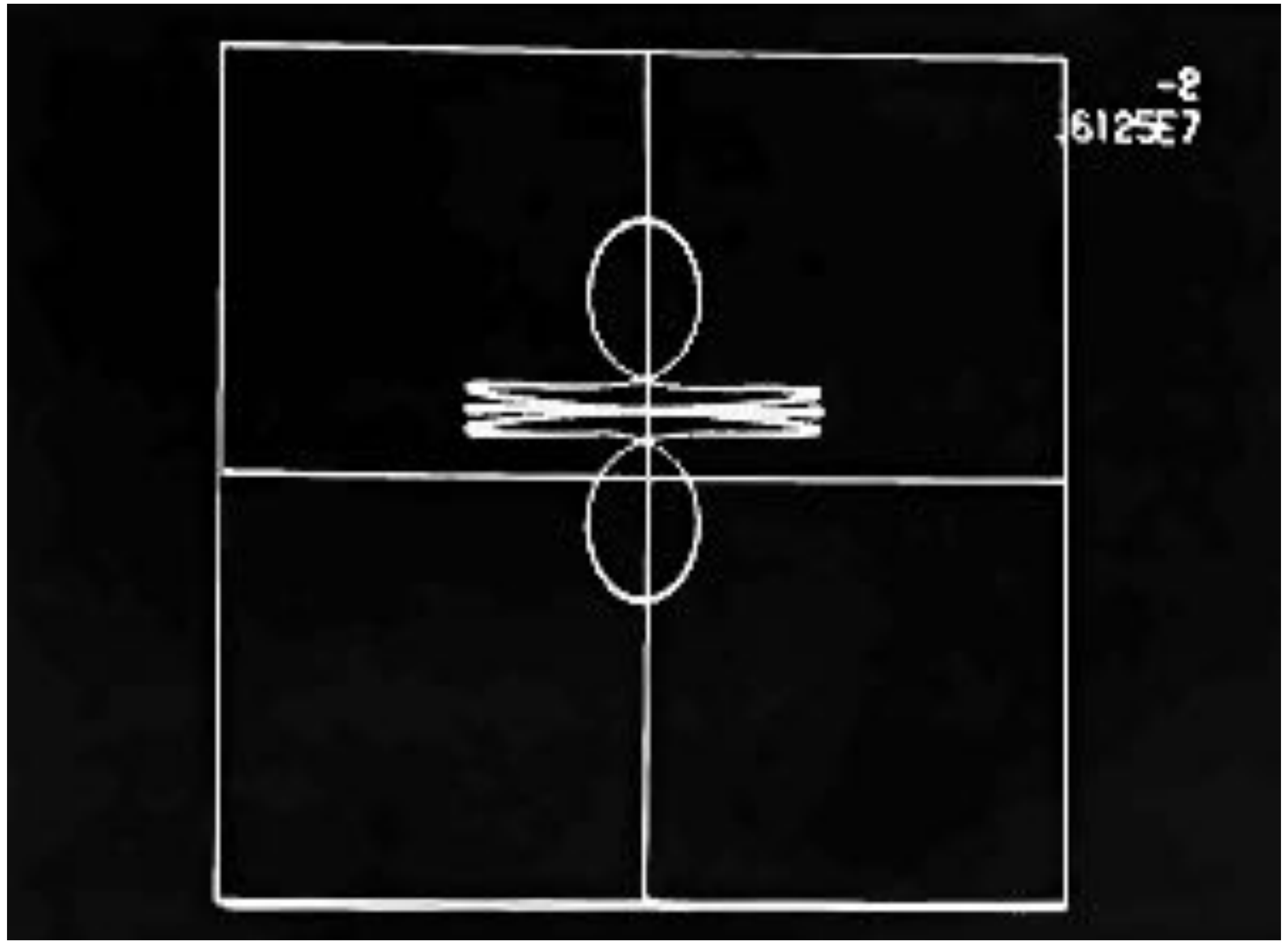
http://www3.nd.edu/~atrozzol/BellLabs1959.jpg

Back in 1969.

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

Ken Thompson wanted to play.

Ken Thompson wanted to play.

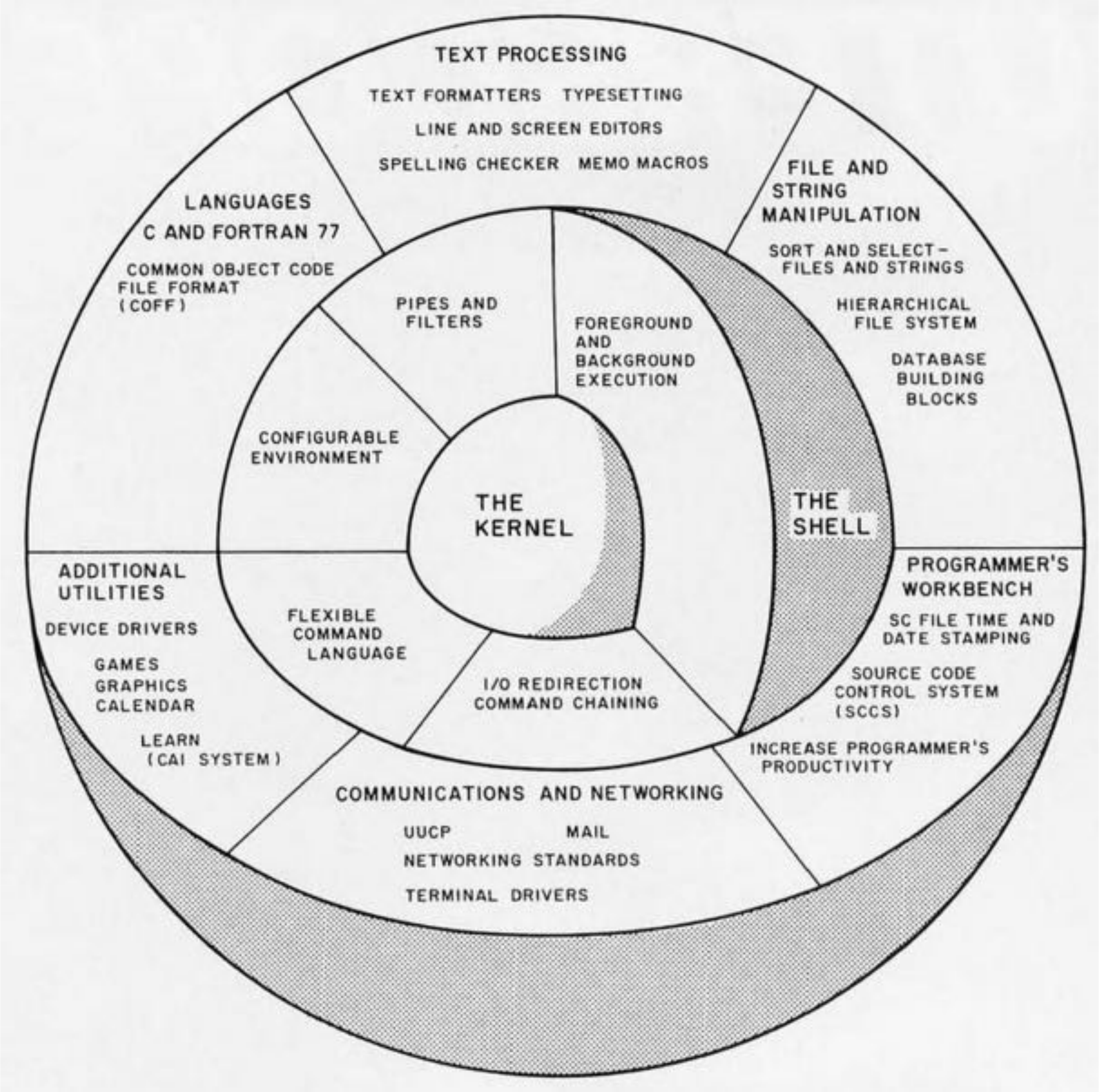


http://upload.wikimedia.org/wikipedia/commons/3/36/Ken_n_dennis.jpg

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.



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

Dennis

Ken

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

they created C,

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

http://cm.bell-labs.com/cm/cs/who/dmr/ctut.pdf heavily inspired by Martin Richards' portable systems programming language BCPL.

Martin Richards, Dec 2014

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

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

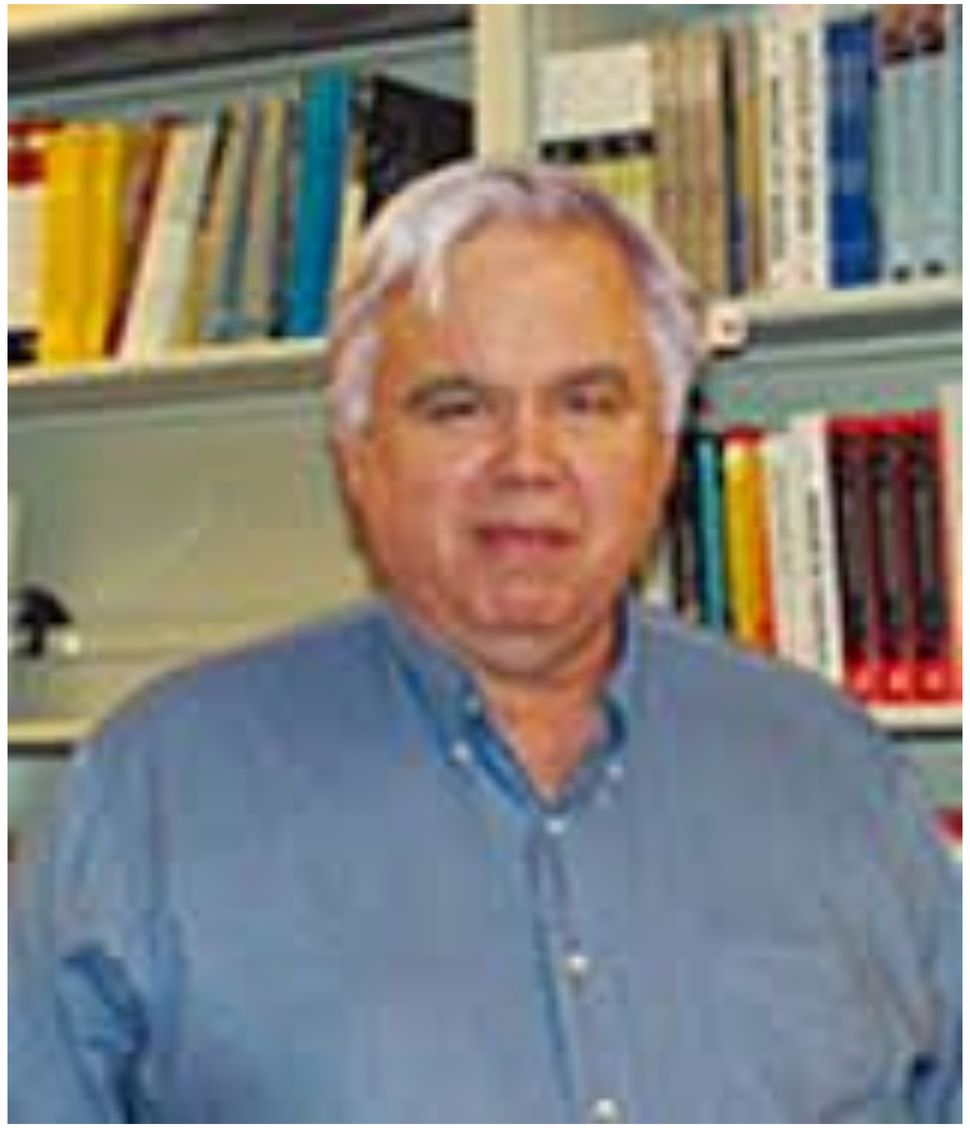
In 1972 Unix was rewritten in C,

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

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

	case 'd': /* decimal */
	case 'o': /* octal */
		if(x < 0) {
			x = -x;
			if(x<0)  {      /* - infinity */
				if(c=='o')
					printf("100000");
				else
					printf("-32767");
				goto loop;
			}
			putchar('-');
		}
		printn(x, c=='o'?8:10);
		goto loop;

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

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

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

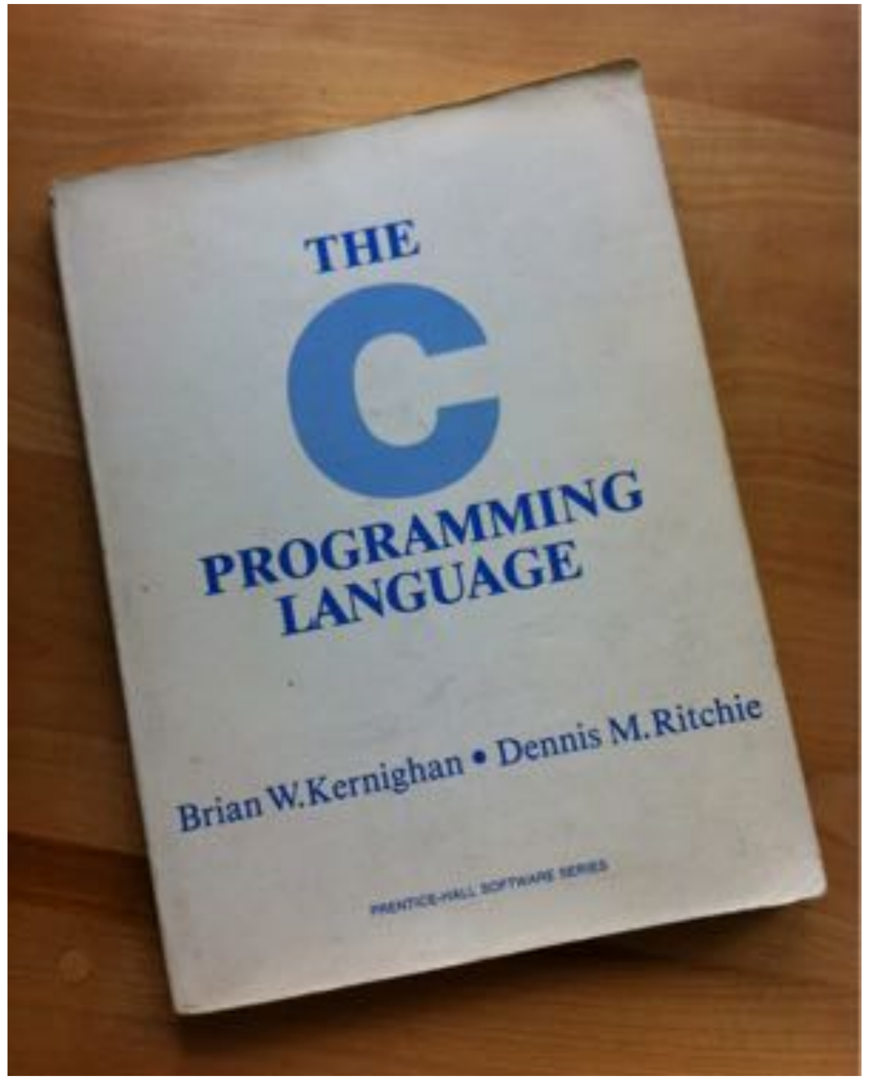
and later ported to many other machines

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

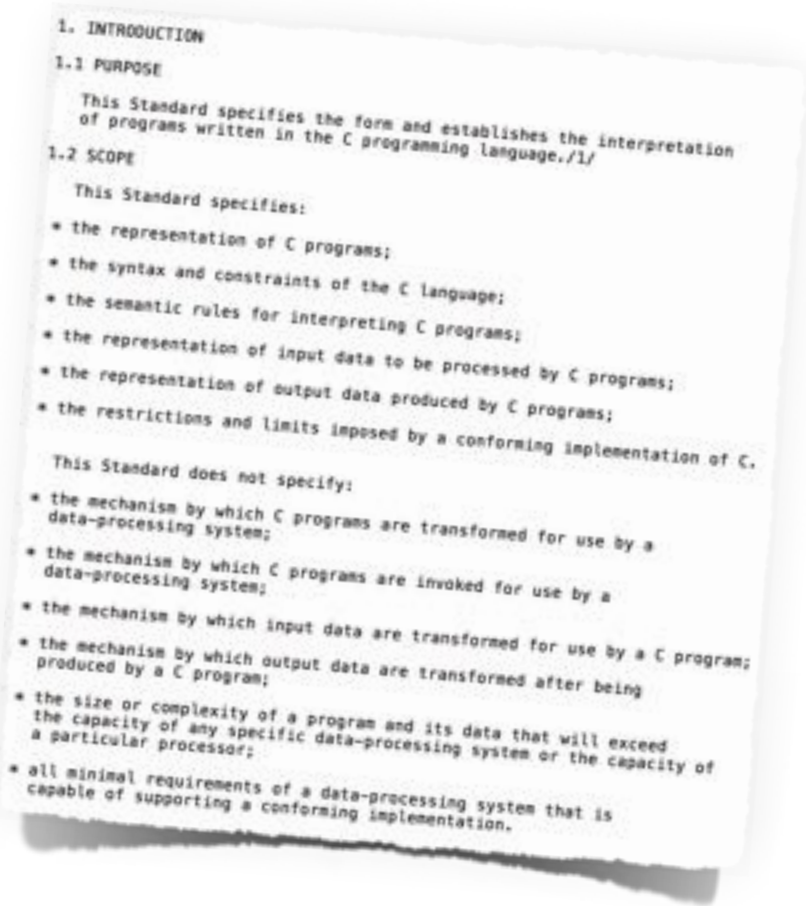
C also gained popularity outside the realm of PDP-11 and Unix.



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.

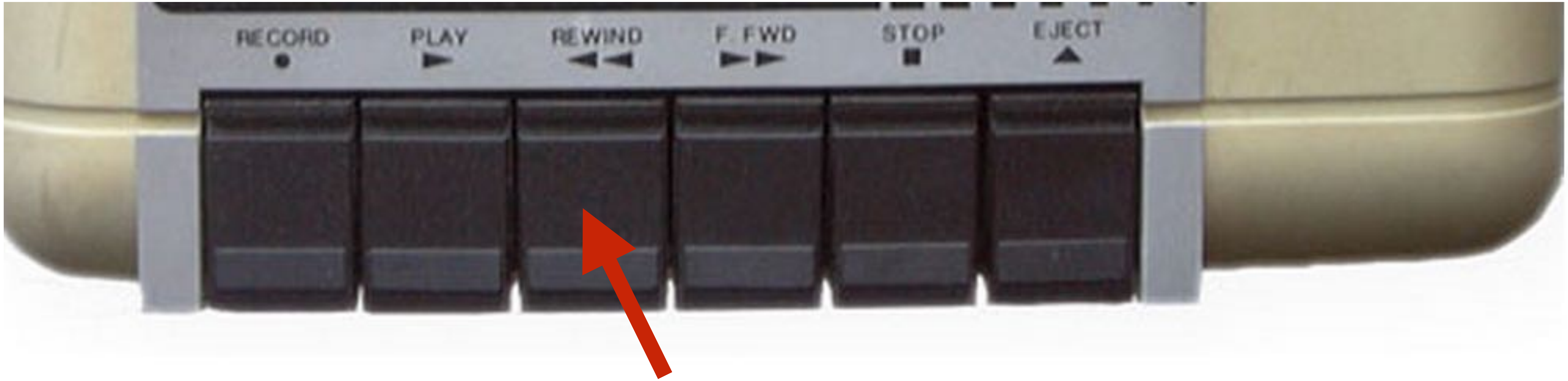
ANSI/ISO C (C89/C90)

C99

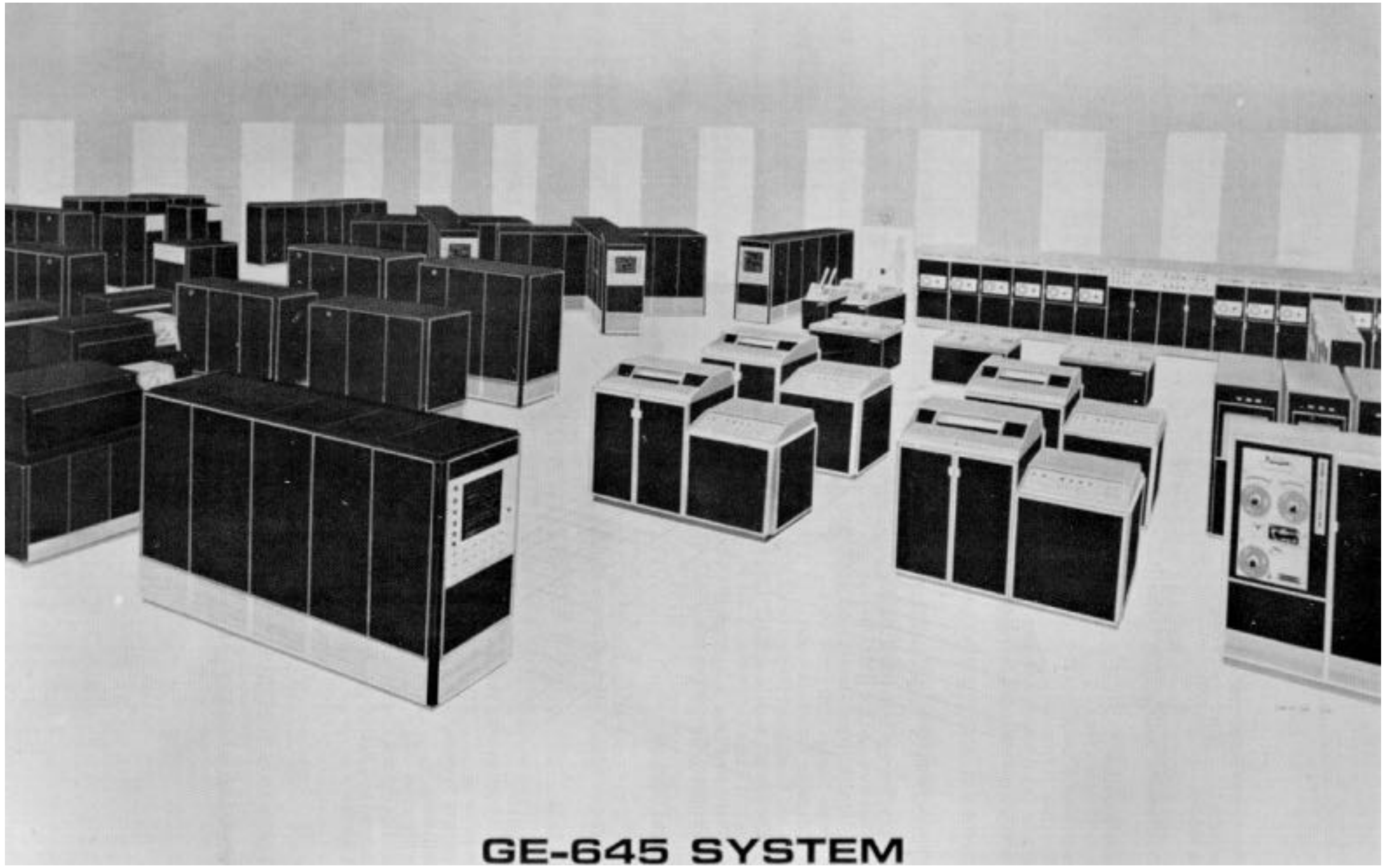
C11

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 created 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.



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)

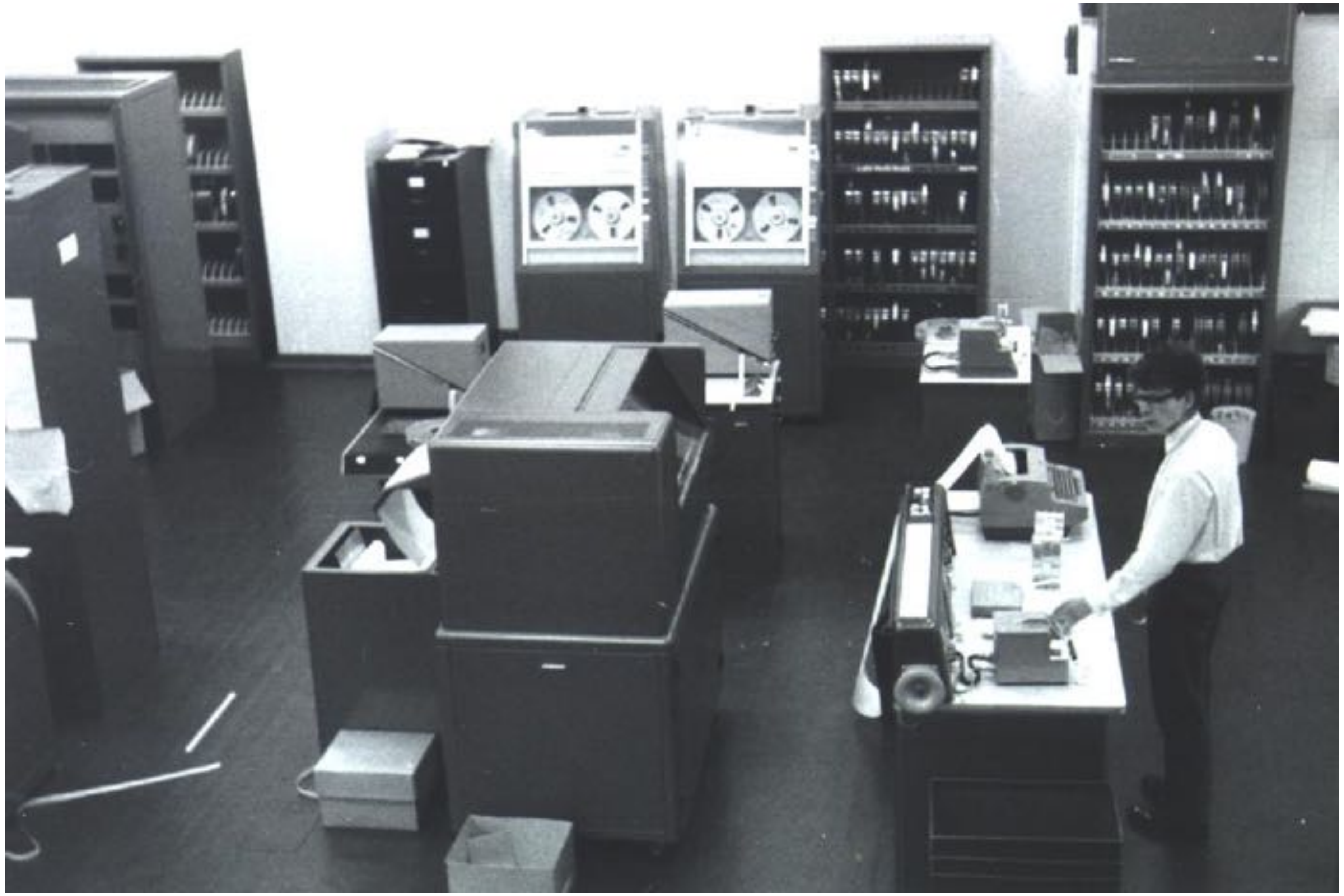
BCPL is a simple recursive programming language designed for compiler writing and system programming: it was derived from true CPL (Combined Programming Language) by removing those features of the full language which make compilation difficult namely, the type and mode matbhing rules and the variety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

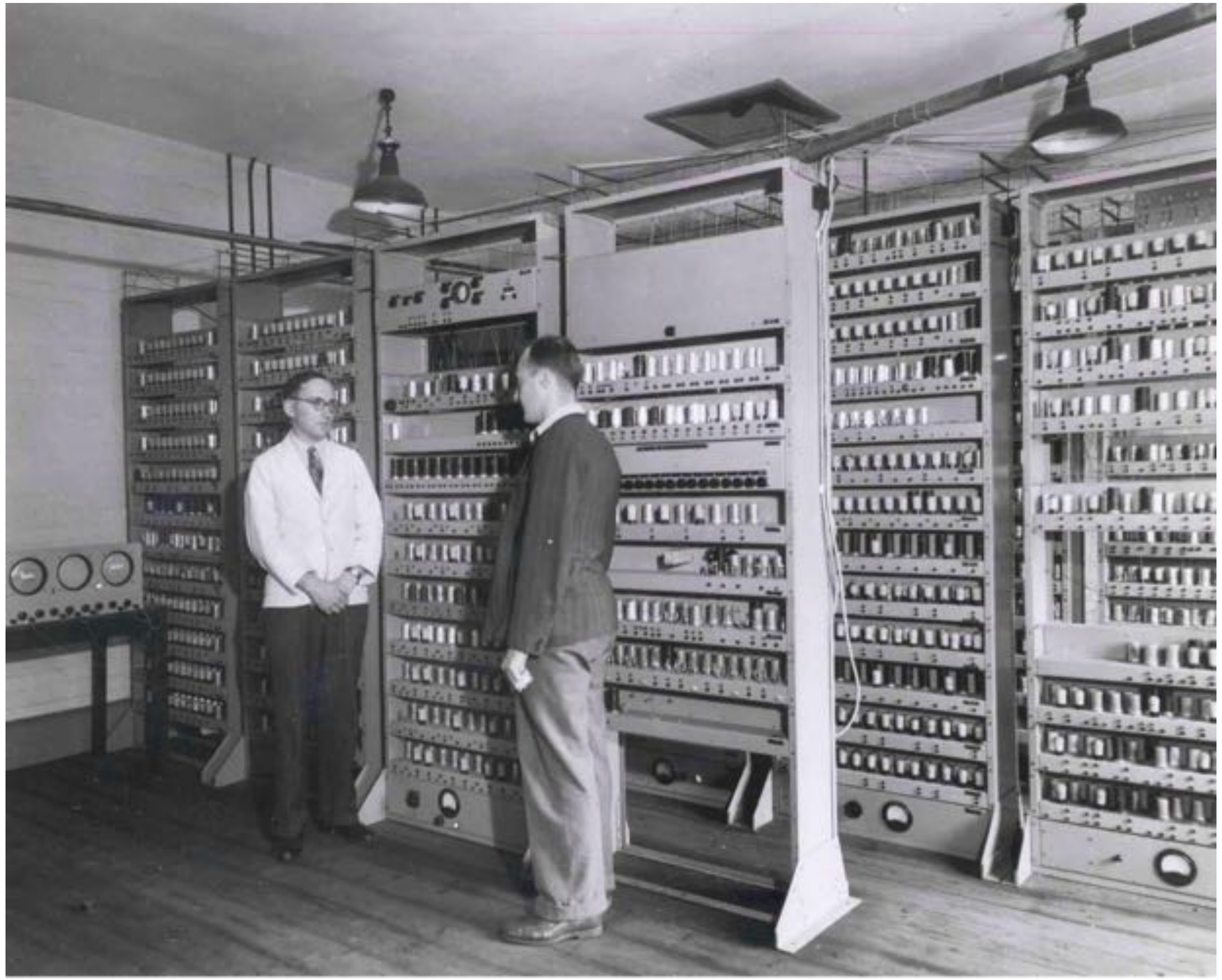


which was a replacement for the very busy EDSAC 2

EDSAC 2 users in 1960

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

Lets take a closer look at the EDSAC 1 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

“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

“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
```

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

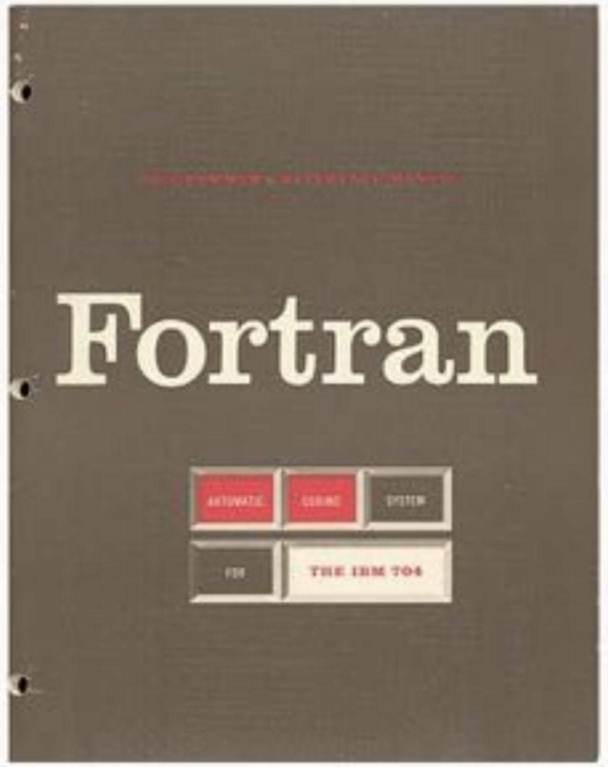
IBM 701 processor frame

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

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

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

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

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)

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

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 + nextx^2 ; Syy = Syy + nexty^2
          Sxy = Sxy + nextx*nexty
3:        read (nextx) ; ->1 unless nextx = 999 999
          a = (n*Sxy - Sx*Sy)/(n*Sxx - Sx^2)
          b = (Sy - a*Sx)/n
          c = Syy - 2(a*Sxy + b*Sy) + a^2*Sxx - 2a*b*Sx + n*b^2
          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 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 … Cambridge CPL compiler. The task was daunting becaus… language 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)

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 … 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)

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 … Cambridge CPL compiler. The task was daunting becaus … language 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)


WENGER

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

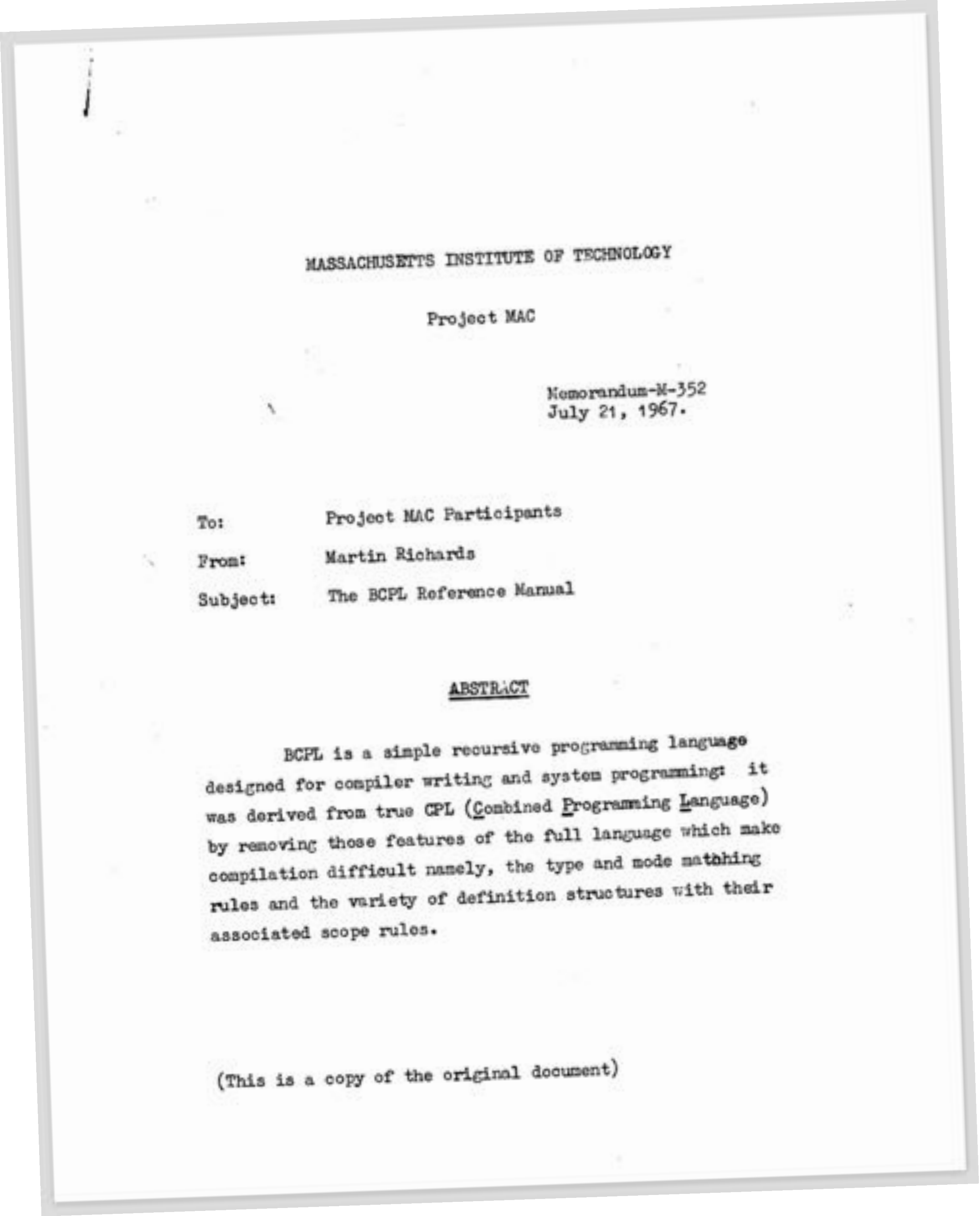
"The philosophy of BCPL is not one of the tyrant who thinks he knows best and lay down the law on what is and what is not allowed; rather, BCPL acts more as a servant offering his services to the best of his ability without complaint, even when confronted with apparent nonsense. The programmer is always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

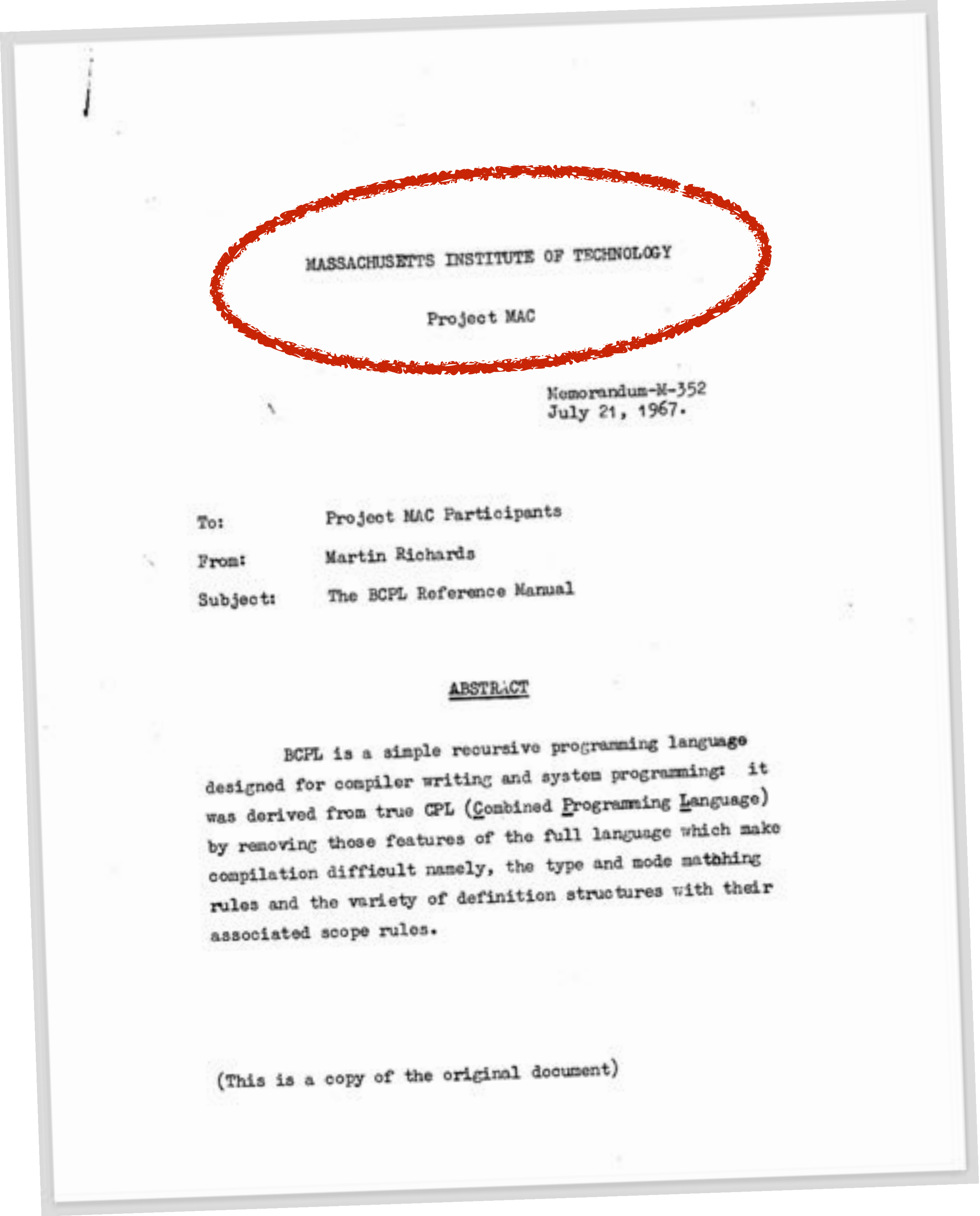
ABSTRACT

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

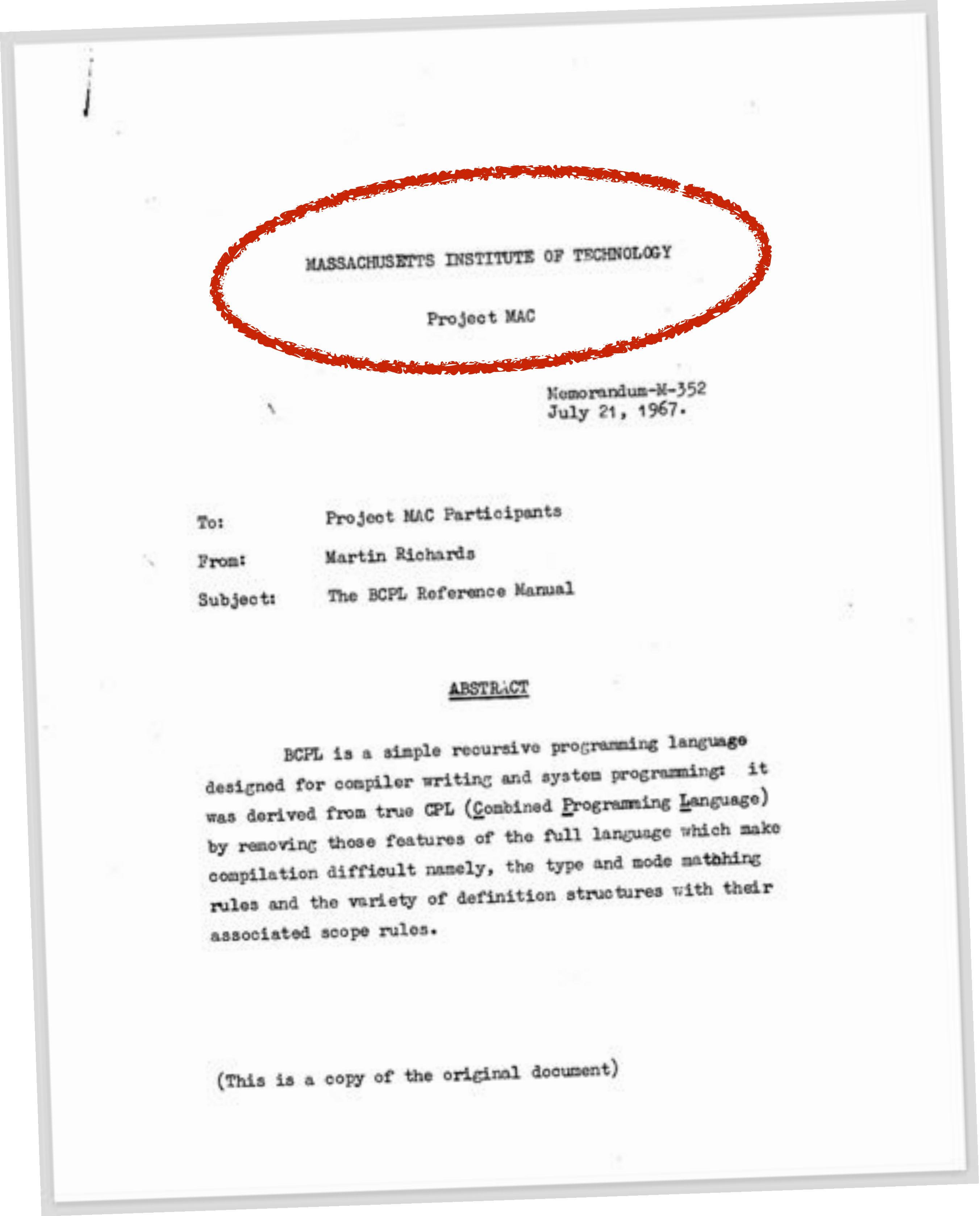
ABSTRACT

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.

The BCPL Reference Manual, Martin Richards, July 1967



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Memorandum-M-352
July 21, 1967.

To: Project MAC Participants

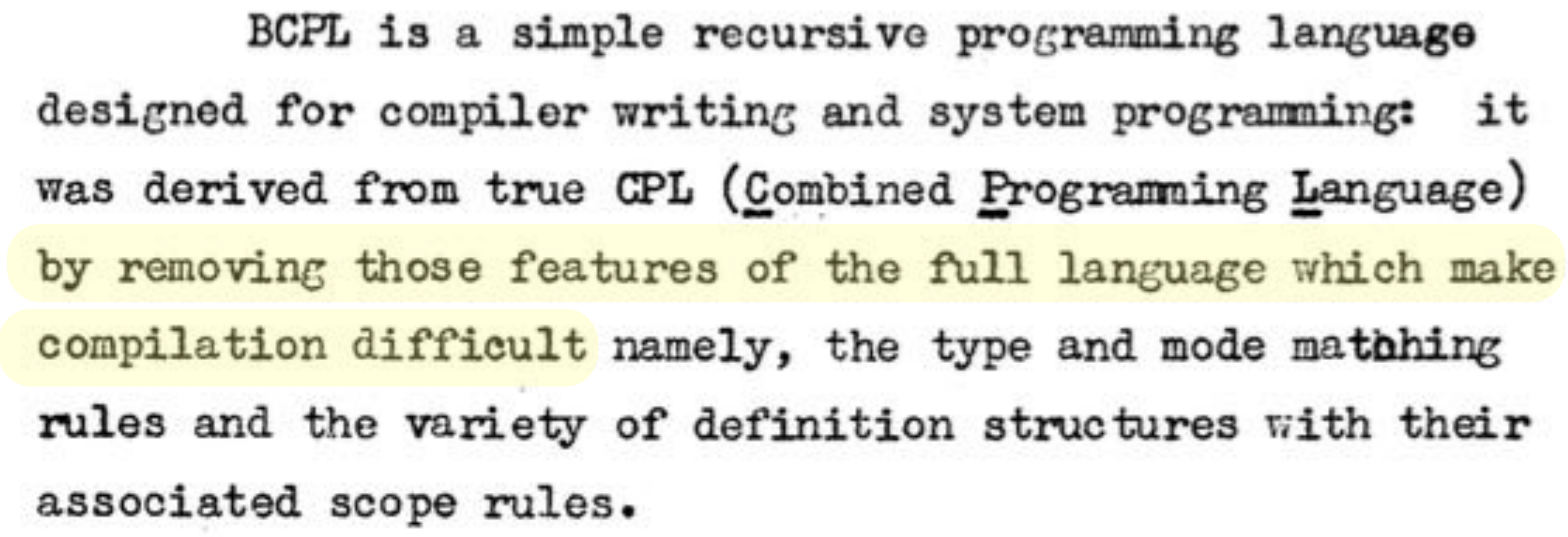
From: Martin Richards

Subject: The BCPL Reference Manual

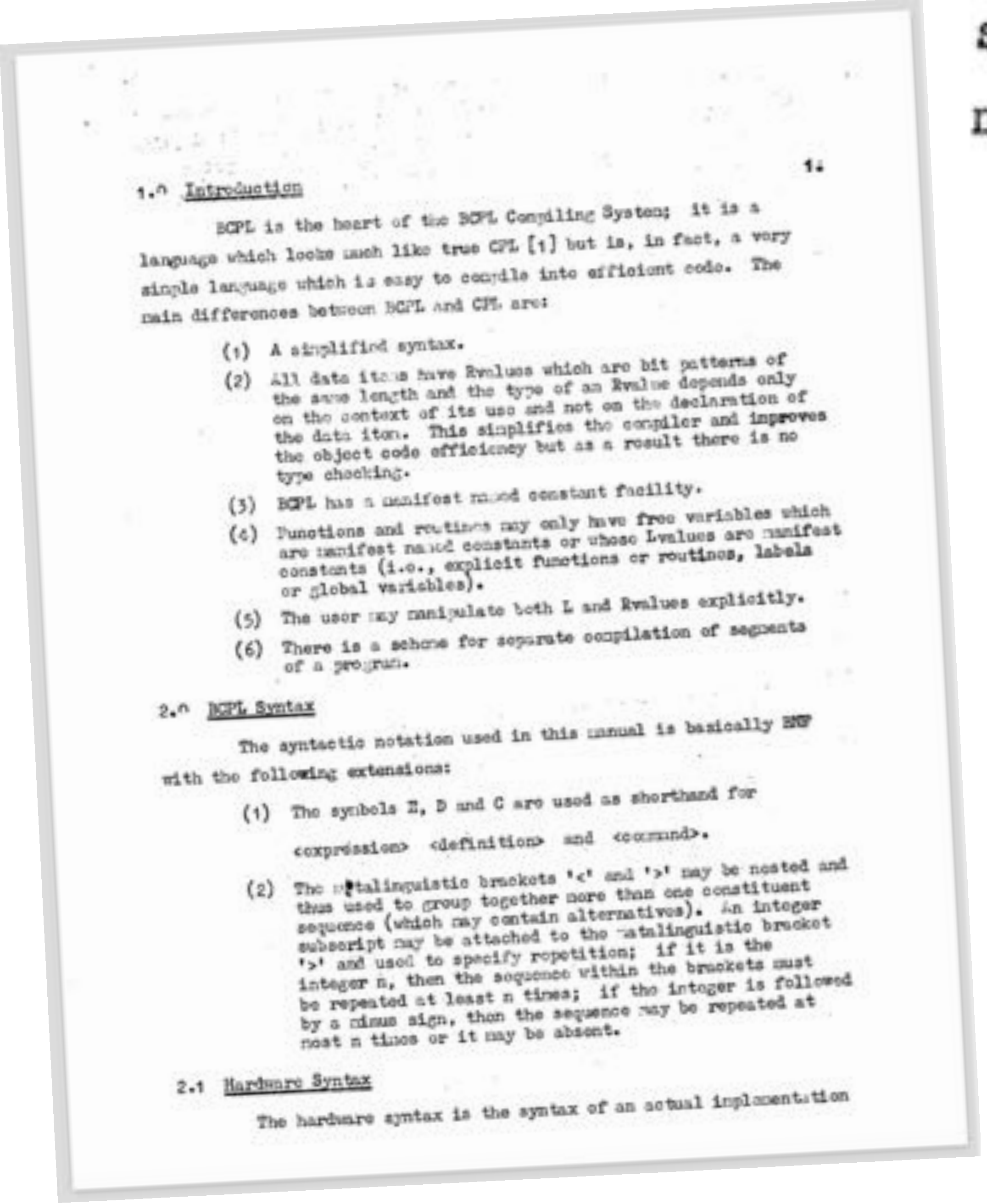
ABSTRACT

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.



1.0 Introduction

1.

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

(1) A simplified syntax.

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

(3) BCPL has a manifest named constant facility.

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

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

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

2.0 BCPL Syntax

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

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

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

2.1 Hardware Syntax

The hardware syntax is the syntax of an actual implementation

BCPL is the heart of the BCPL Compiling System; it is 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.

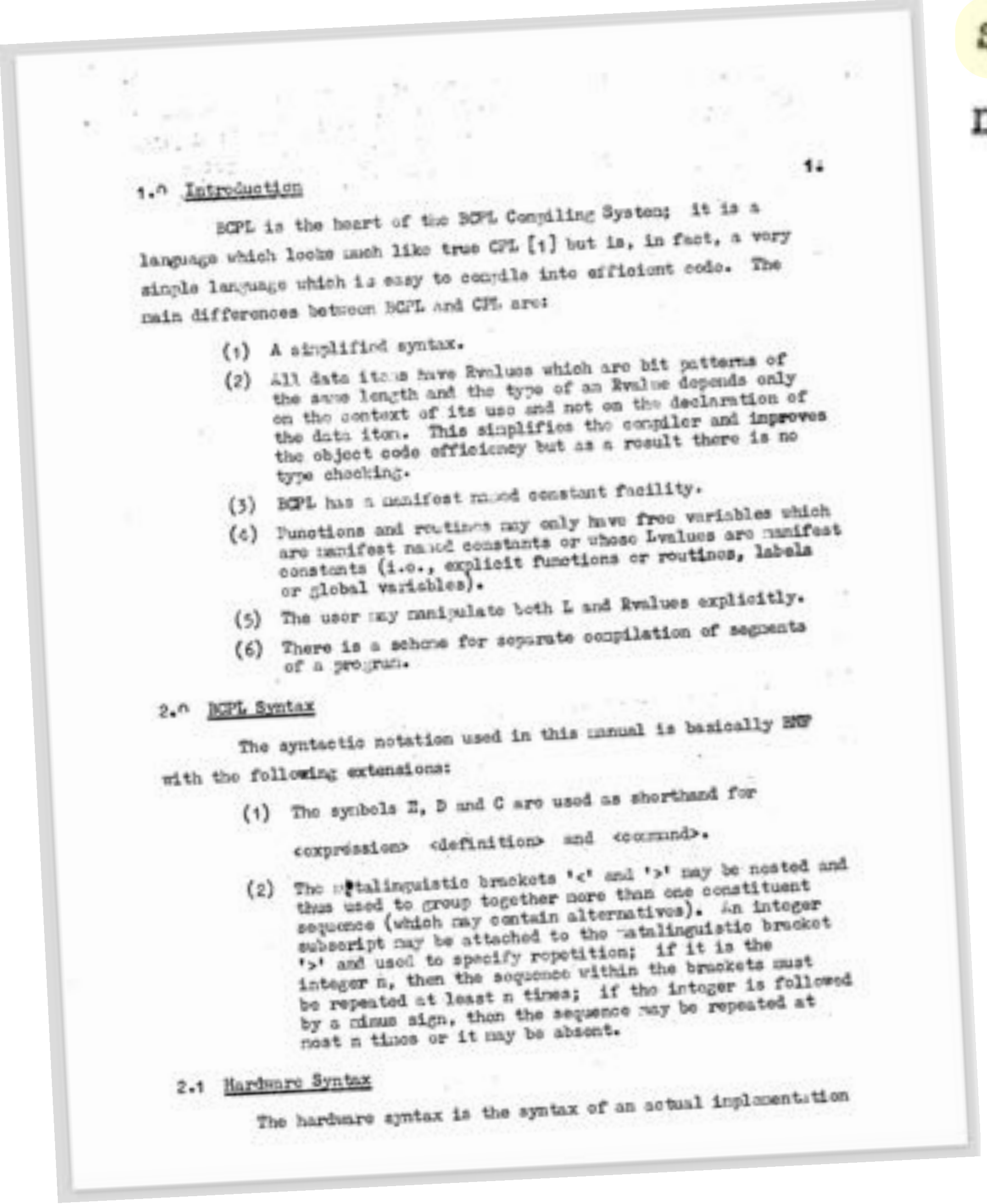
(2) All data items have Rvalues which are bit patterns of the same length and the type of an Rvalue depends only on the context of its use and not on the declaration of the data item. This simplifies the compiler and improves the object code efficiency but as a result there is no type checking.

(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.

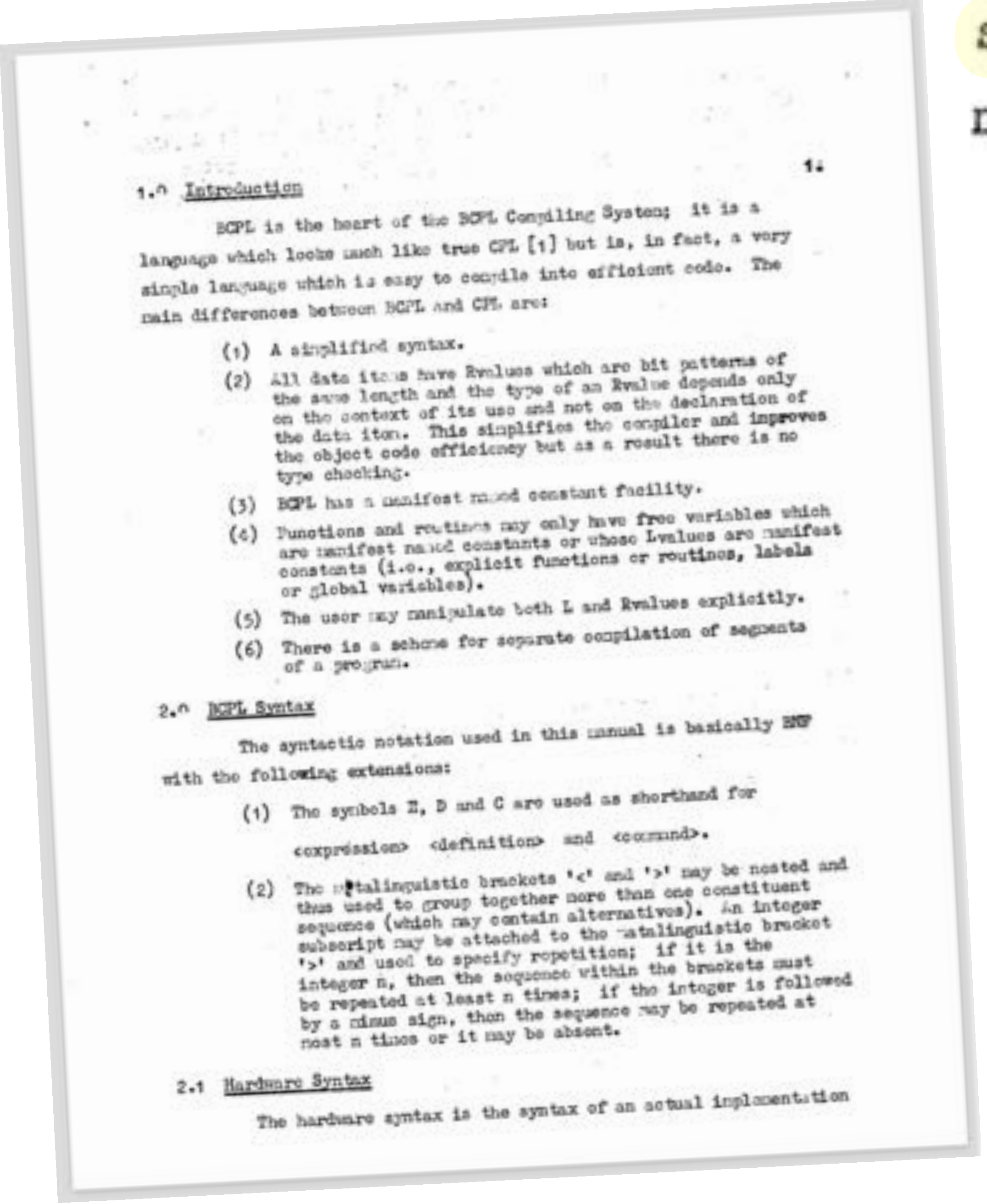
(2) All data items have Rvalues which are bit patterns of the same length and the type of an Rvalue depends only on the context of its use and not on the declaration of the data item. This simplifies the compiler and improves the object code efficiency but as a result there is no type checking.

(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:

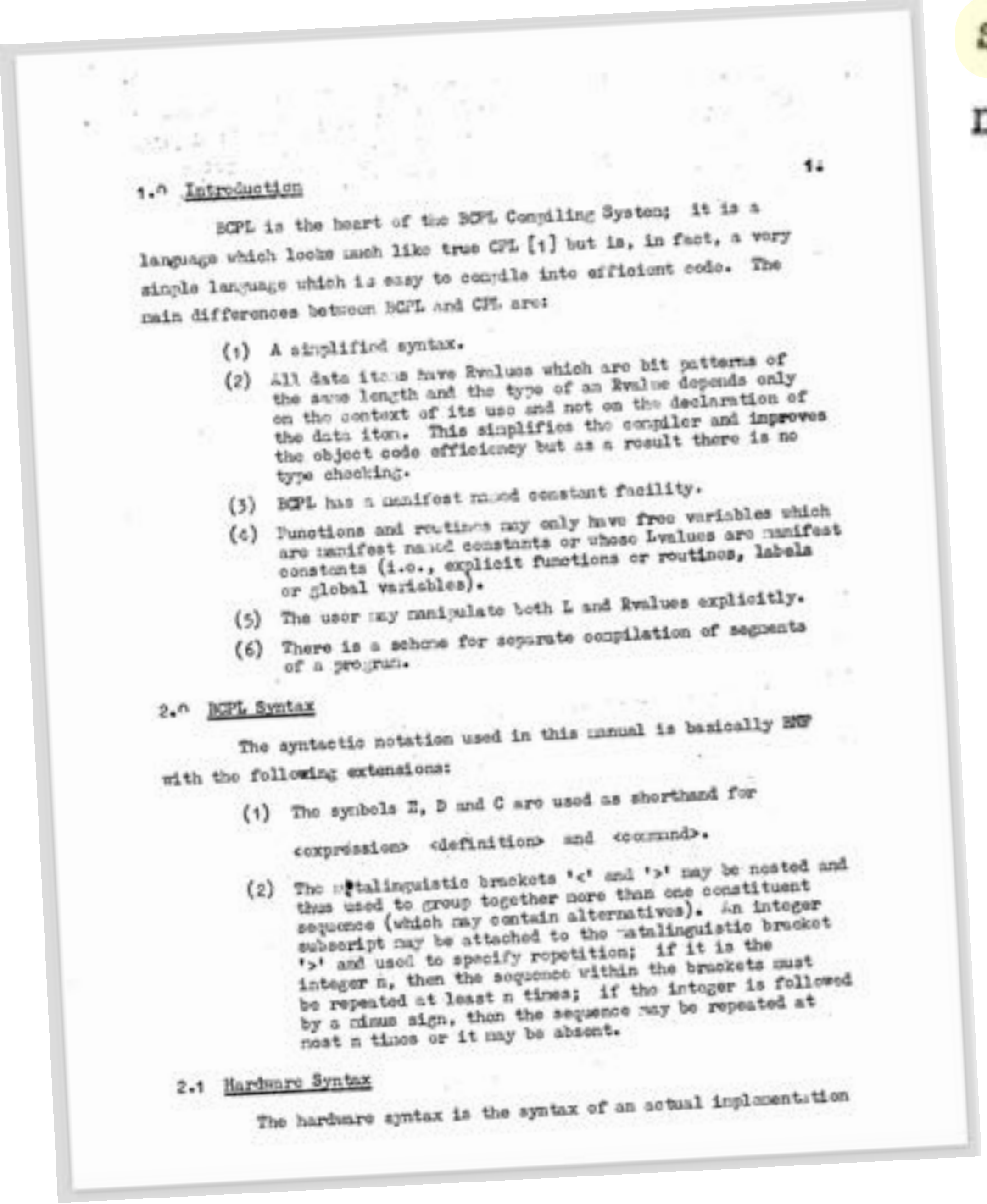
(1) The symbols E, D and C are used as shorthand for <expression> <definition> and <command>.
(2) The metalinguistic brackets '<' and '>' may be nested and thus used to group together more than one constituent sequence (which may contain alternatives). An integer subscript may be attached to the metalinguistic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



1.0 Introduction 1.

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

(1) A simplified syntax.

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

(3) BCPL has a manifest named constant facility.

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

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

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

2.0 BCPL Syntax

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

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

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

2.1 Hardware Syntax

The hardware syntax is the syntax of an actual implementation

BCPL is the heart of the BCPL Compiling System; it is 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.

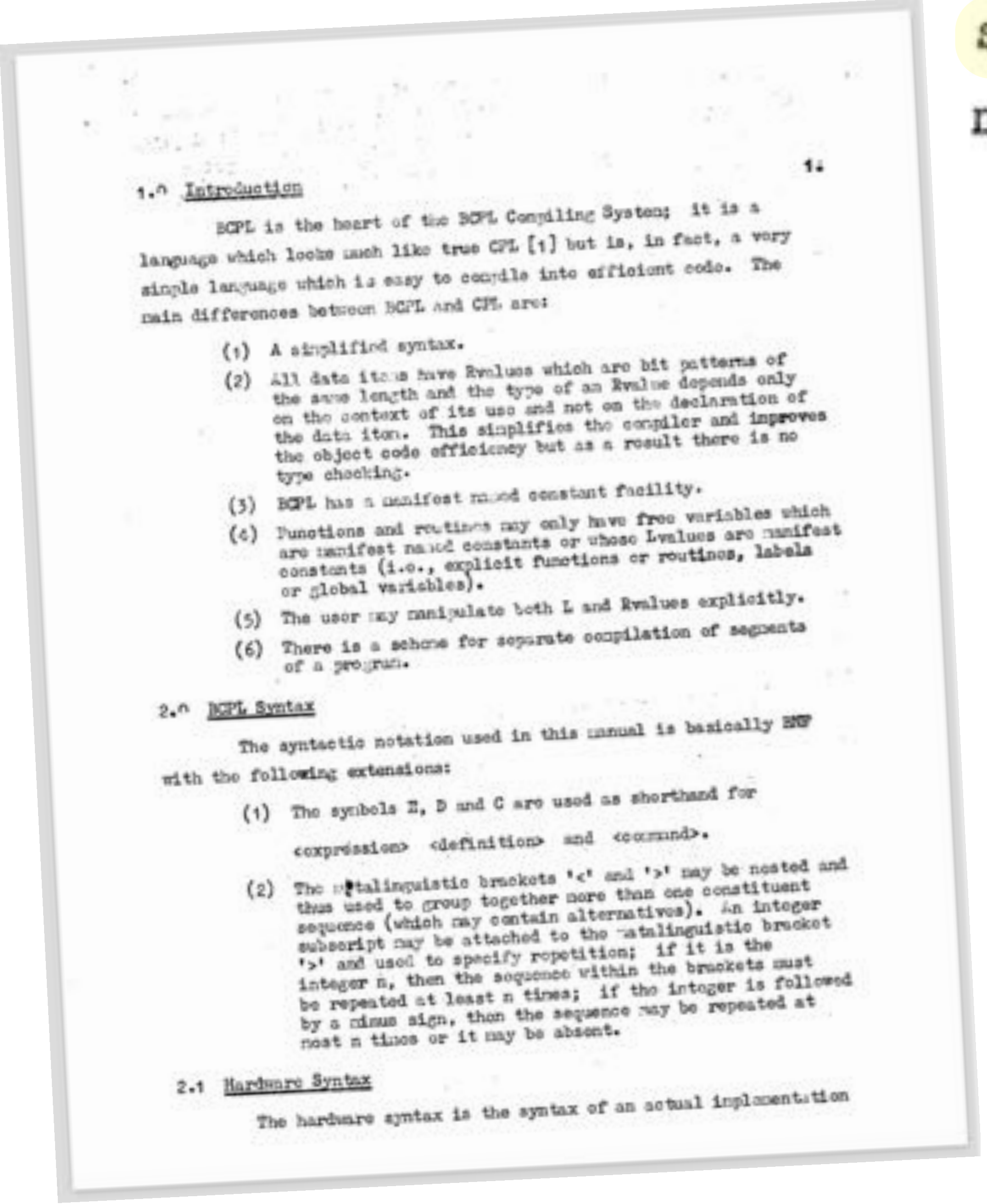
(2) All data items have Rvalues which are bit patterns of the same length and the type of an Rvalue depends only on the context of its use and not on the declaration of the data item. This simplifies the compiler and improves the object code efficiency but as a result there is no type checking.

(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

14

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.

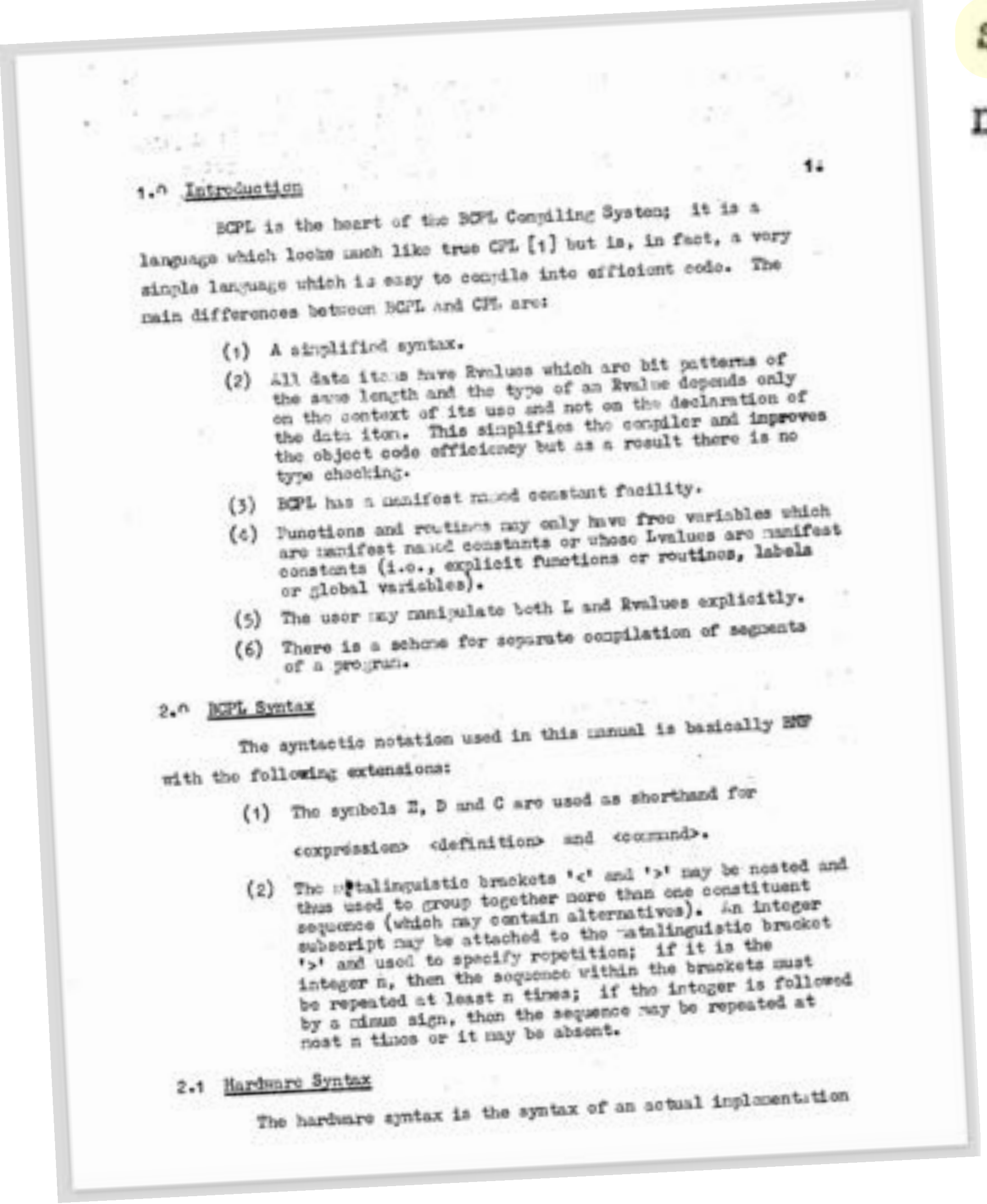
(2) All data items have Rvalues which are bit patterns of the same length and the type of an Rvalue depends only on the context of its use and not on the declaration of the data item. This simplifies the compiler and improves the object code efficiency but as a result there is no type checking.

(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

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.

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.

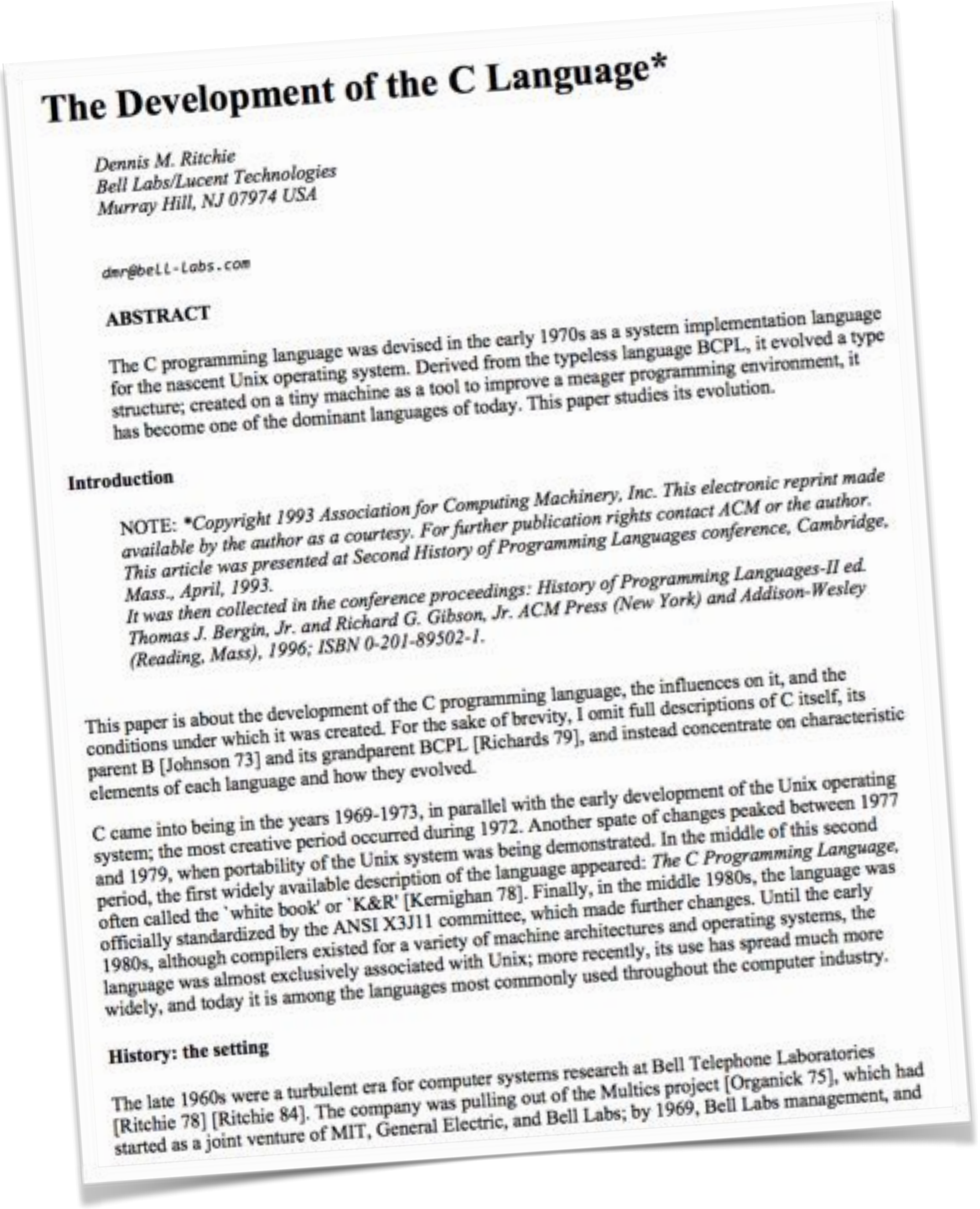
...

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

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

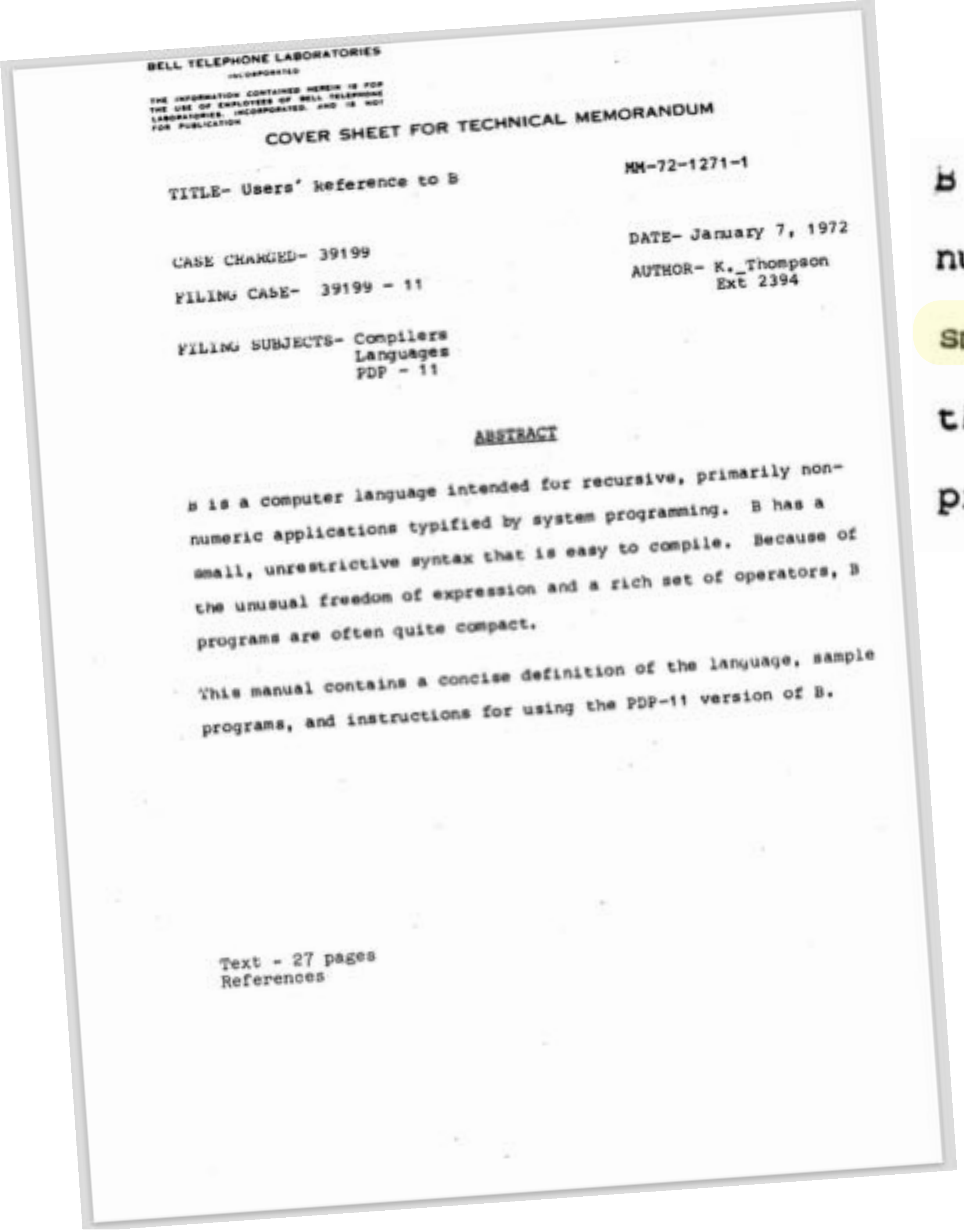
The late 1960s were a turbulent era for computer systems research at Bell Telephone Laboratories [Ritchie 78] [Ritchie 84]. The company was pulling out of the Multics project [Organick 75], which had started as a joint venture of MIT, General Electric, and Bell Labs; by 1969, Bell Labs management, and

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

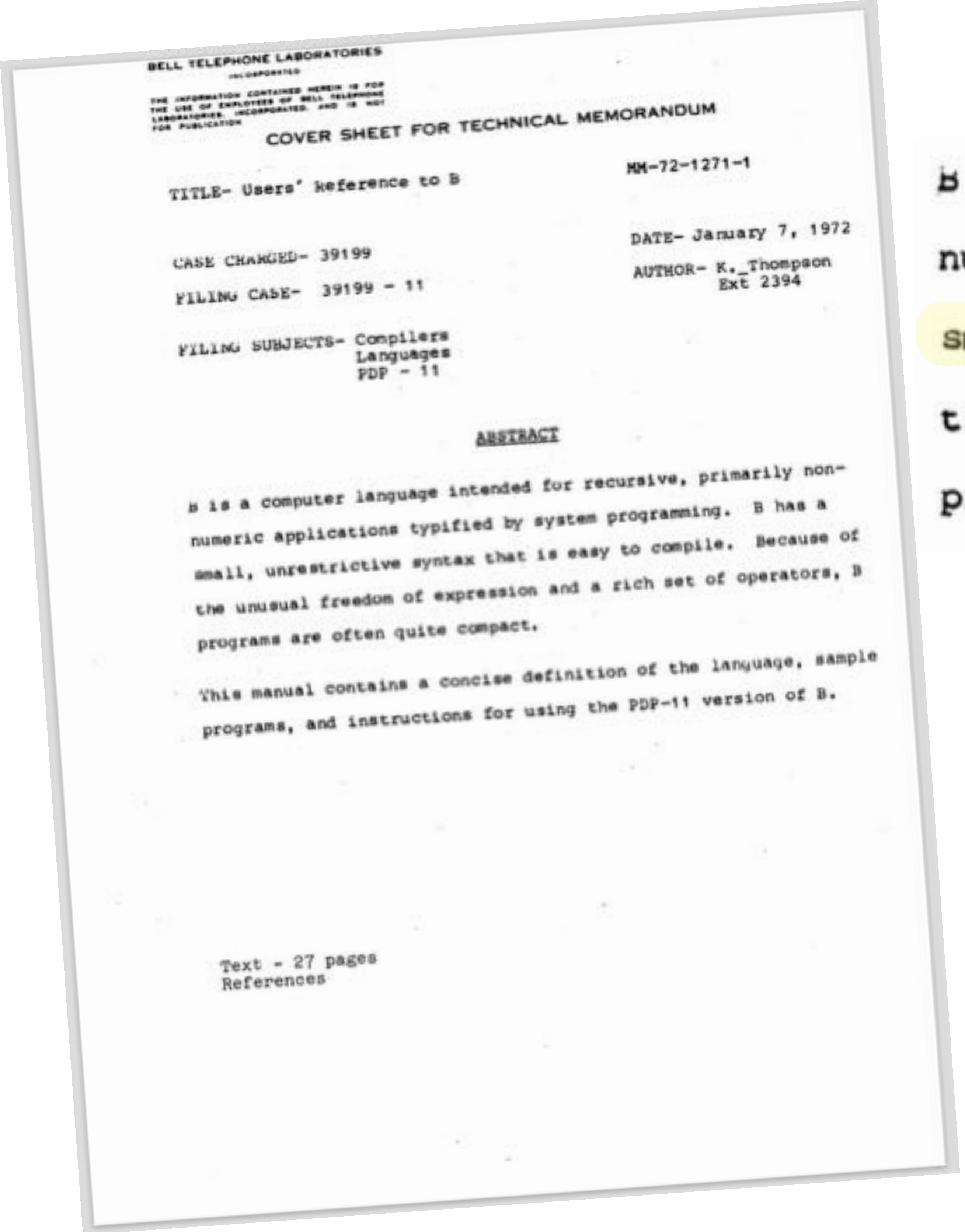
B is a computer language intended for recursive, primarily non-numeric applications typified by system programming. B has a small, unrestrictive syntax that is easy to compile. Because of the unusual freedom of expression and a rich set of operators, B programs are often quite compact.

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.

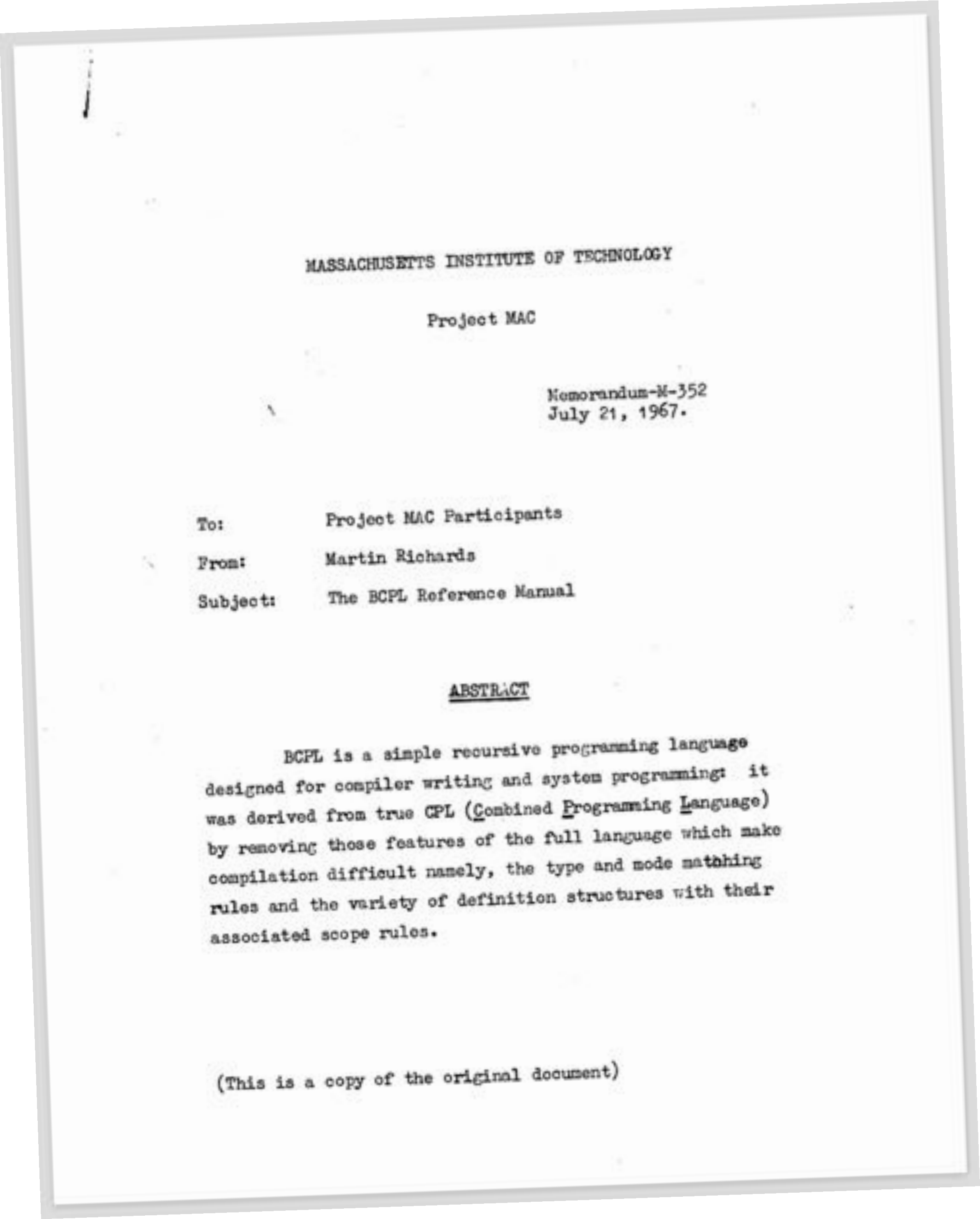
This manual contains a concise definition of the language, sample programs, and instructions for using the PDP-11 version of B.

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

Users' Reference to B, Ken Thompson, January 1972



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Memorandum-M-352
July 21, 1967.

To: Project MAC Participants

From: Martin Richards

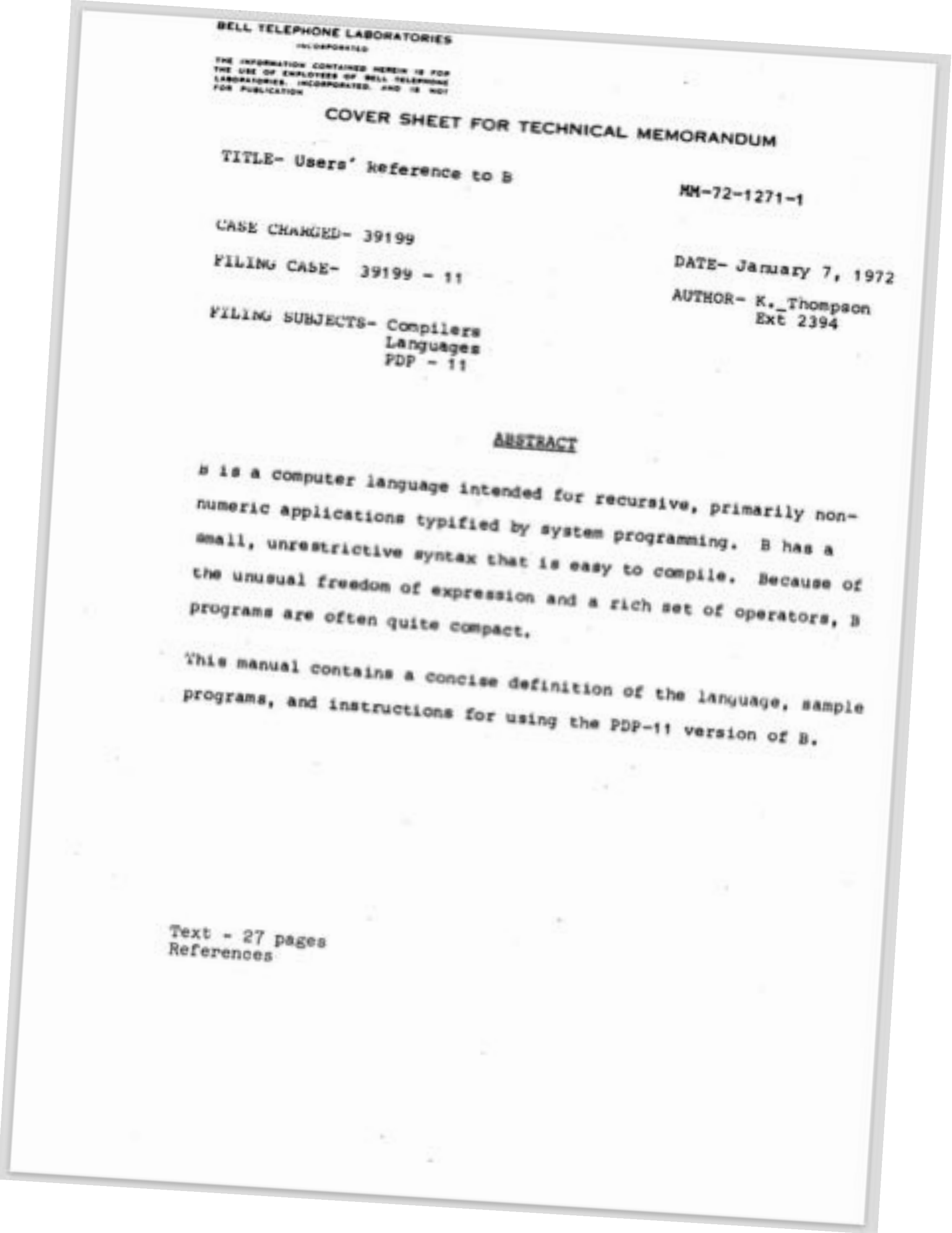
Subject: The BCPL Reference Manual

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



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."

“Good artists copy. Great artists steal.”

Picasso?

"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

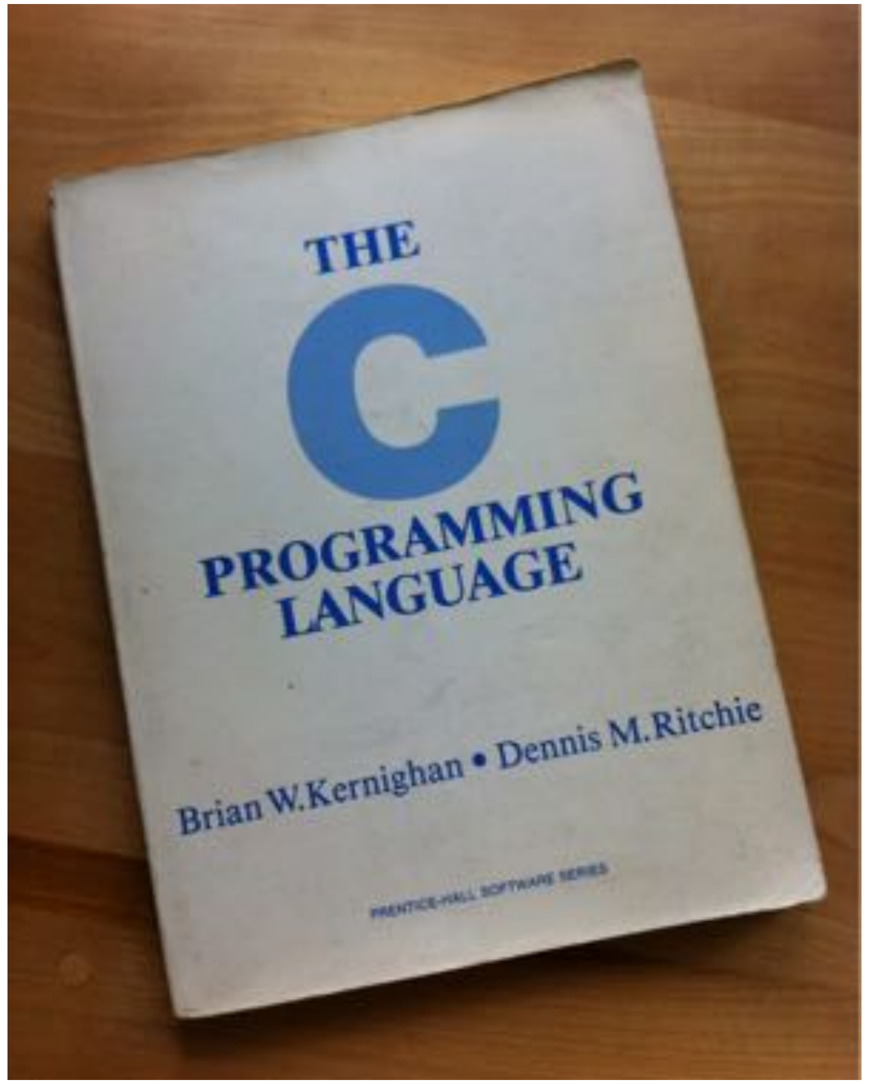
- 16-bit computer
- introduced 1970
- orthogonal instruction set
- byte-oriented

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.




```
/* 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
- A lot of the new features and syntax in ANSI C came from C++

Private conversation with Tom Plum, April 2015

Standardization of C

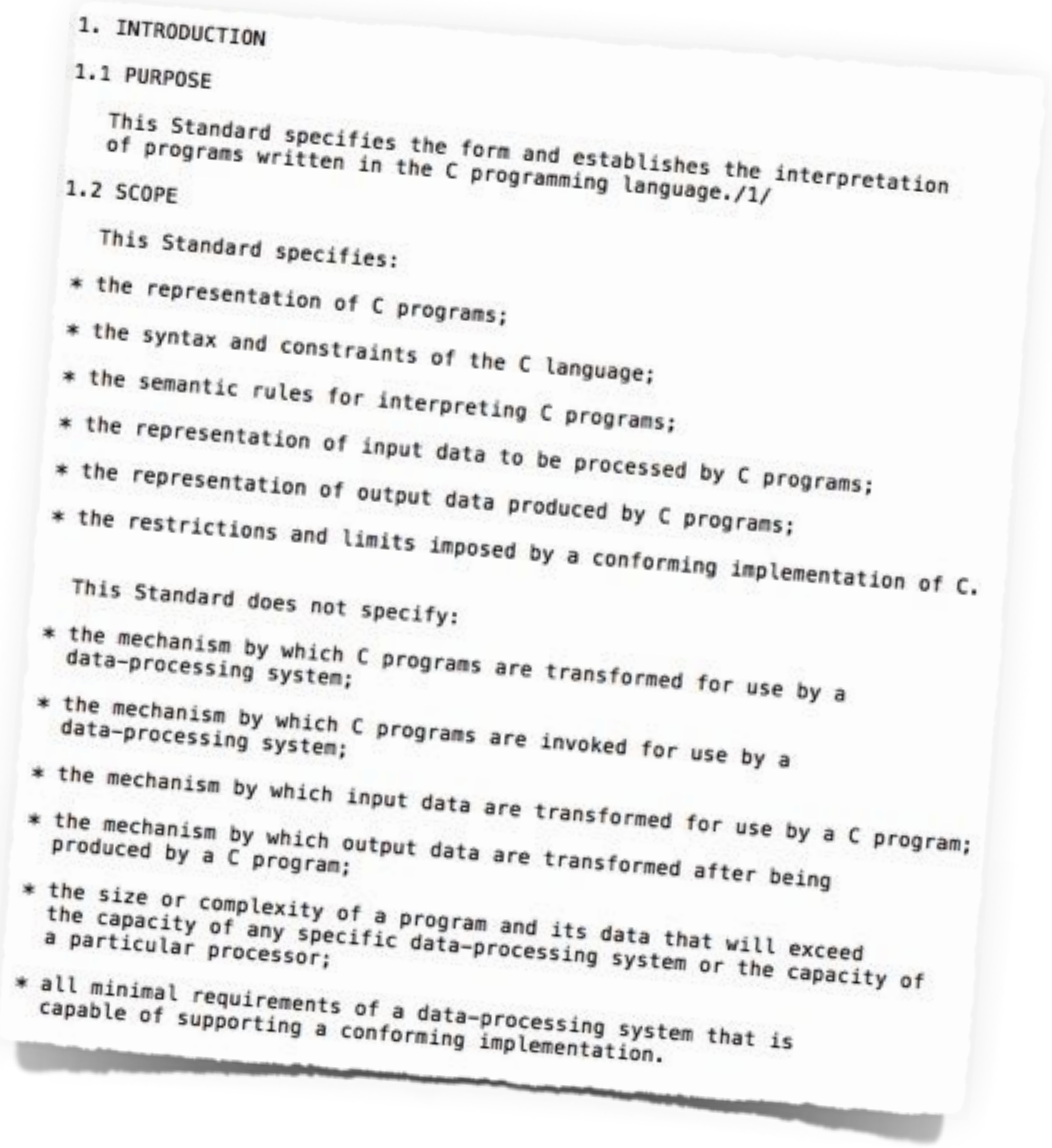
- Dennis Ritchie not involved(except for the "noalias must go" article)
- Committee met four times a year, from 83 til publication
- All meetings in the US (due to political issues between ANSI and ISO)
- The committee avoided inventing features
- All features had to be demonstrated by one or more existing 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
- A lot of the new features and syntax in ANSI C came from C++

It is common to say that C++ is a superset of C. However, it is an interesting perspective to think about ANSI C as a subset of C++, "everything" in ANSI C could be compiled by a proper C++ compiler back in 1989.

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.


```
// 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]);
}
```

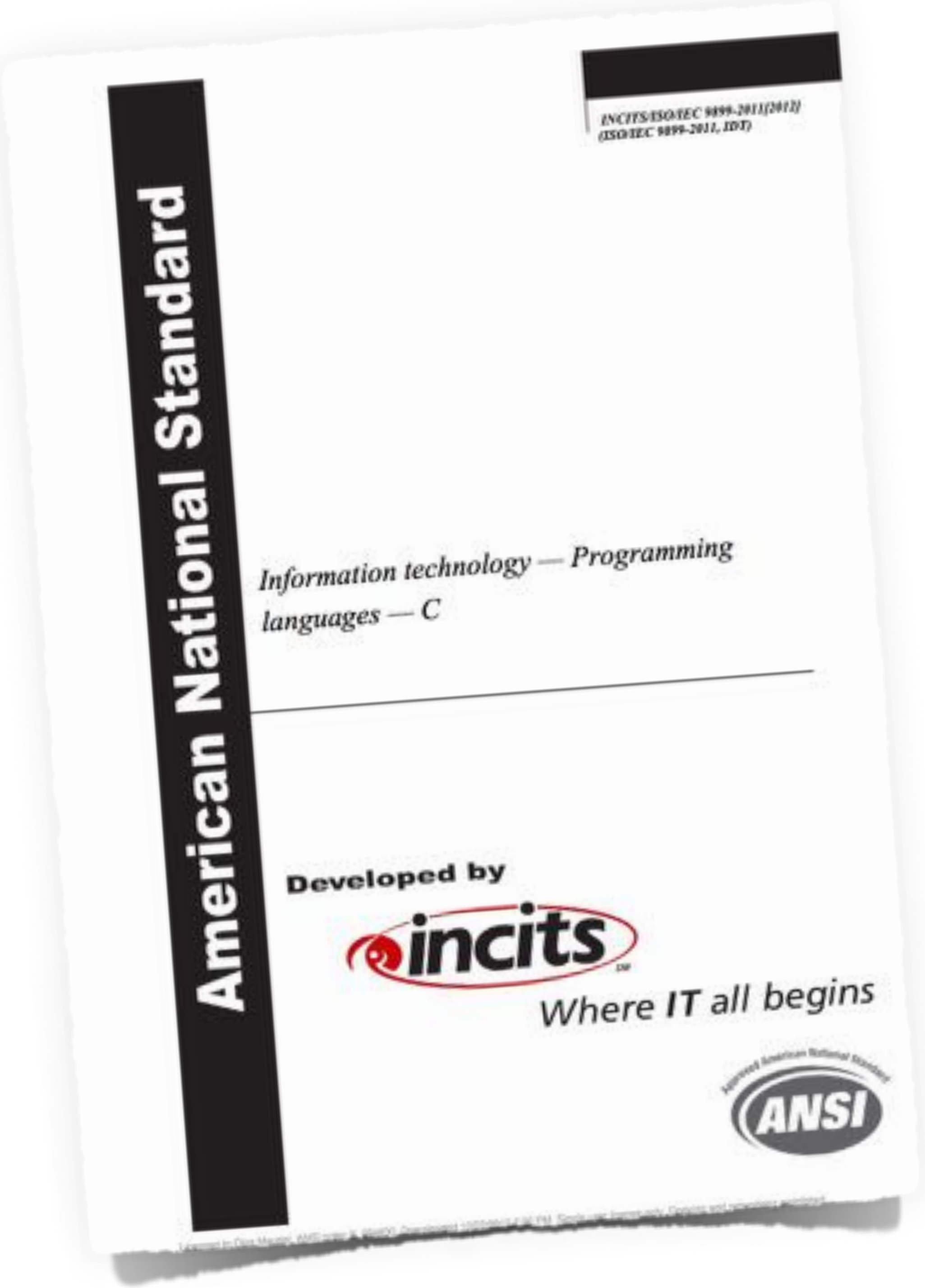
C11

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.



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

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

C++

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

History of C++

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

with words copied from Bjarne Stroustrups book
"The Design and Evolution of C++” from 1994

Bjarne was working on his PhD thesis

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

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

Bjarne was working on his PhD thesis

Bjarne

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

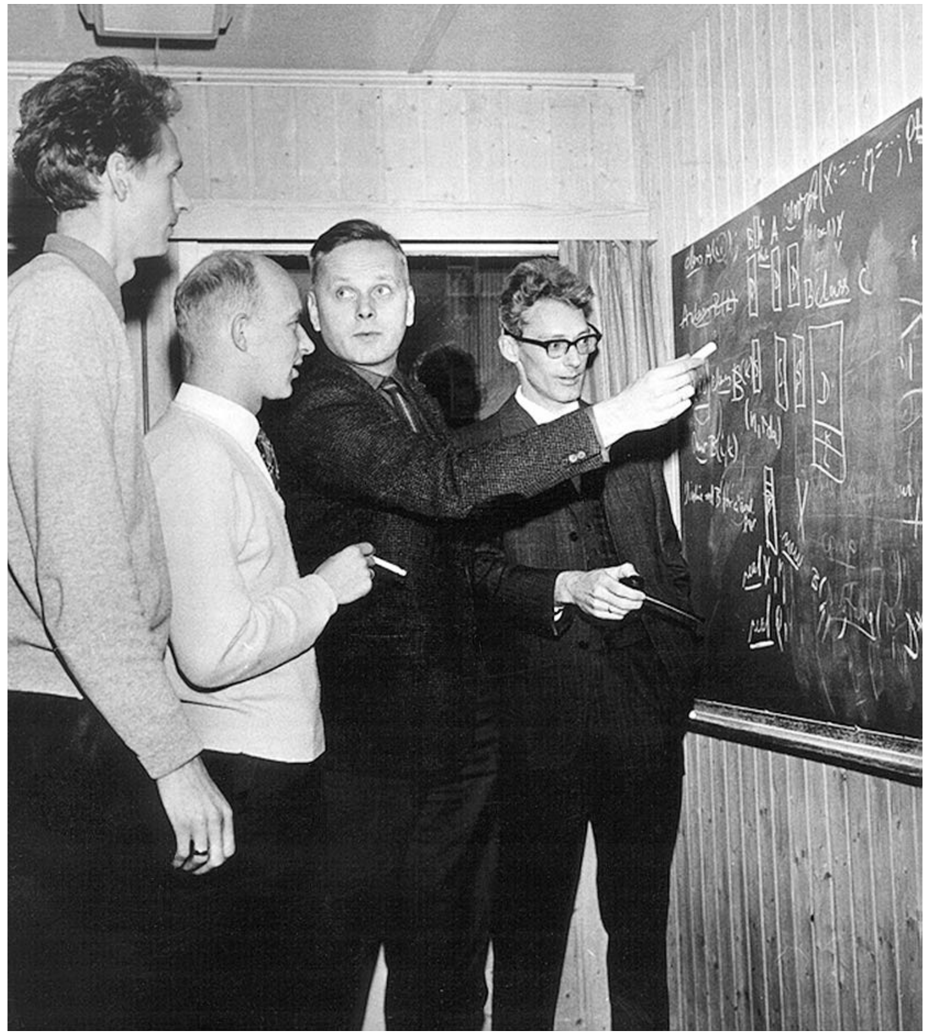
http://computersweden.idg.se/polopoly_fs/1.346563!imageManager/1326219611.jpg in the Computing Laboratory at

in the Computing Laboratory at University of Cambridge.

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

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

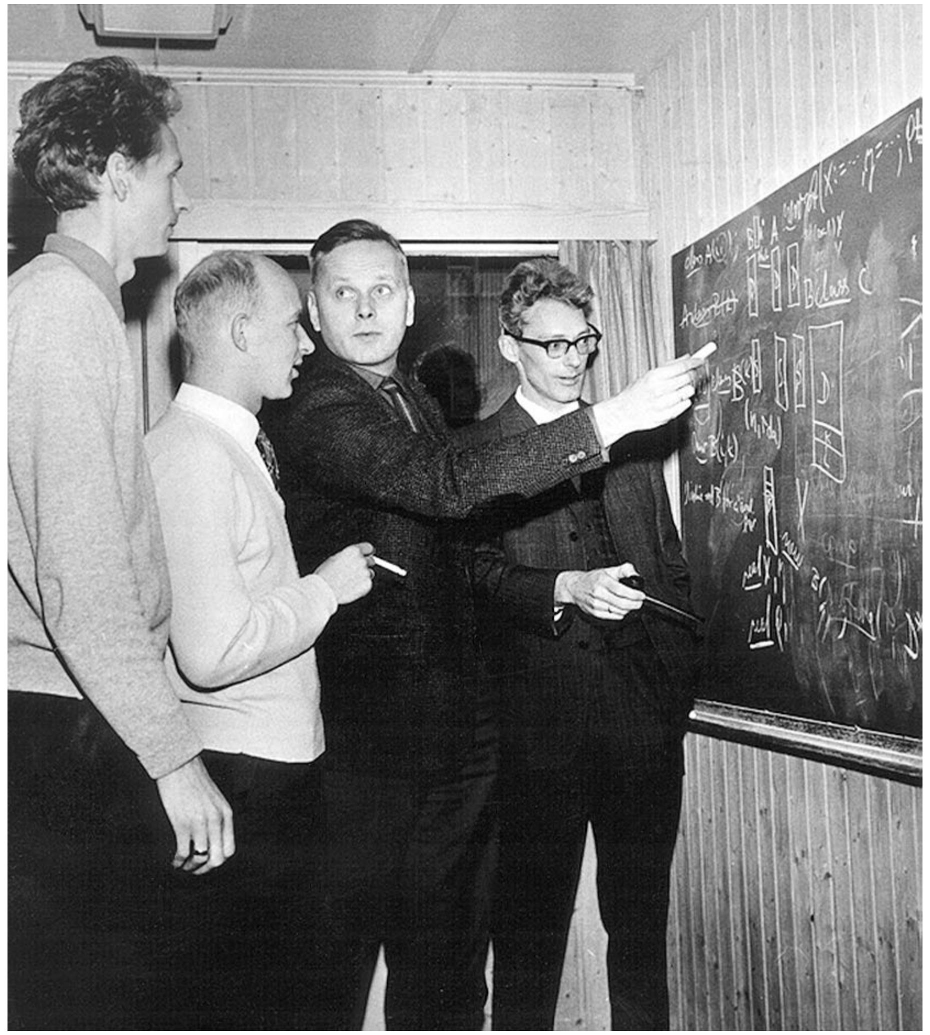
He was working on a simulator to study alternatives for the organization of system software for distributed systems.
The initial version of this simulator was written in Simula



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

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

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



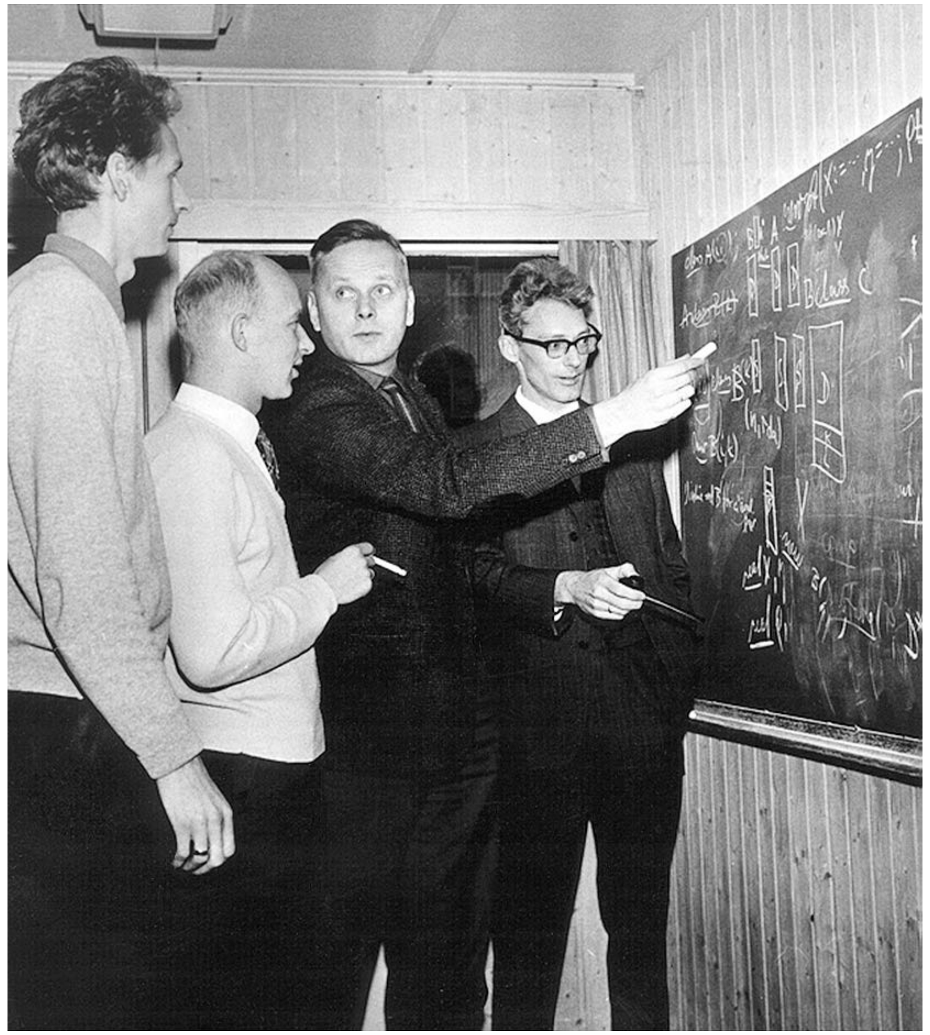
Bjørn Myrhaug, Sigurd Kubosh, Kristen Nygard and Ole Johan Dahl by the “Simula blackboard”

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

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

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

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

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

object oriented programming

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

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



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

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

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

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

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

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

object oriented programming

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

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

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

   Integer u;
   Ref (FittingRoom) fittingRoom1;

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

multitasking

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

and ran on the IBM 360/165 mainframe.

System/370 model 165

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

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

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

Eventually, he 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.”

“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 his PhD Thesis in Cambridge he was hired by

After finishing his PhD Thesis in Cambridge he was hired by Bell Labs.

After finishing his PhD Thesis in Cambridge he was hired by Bell Labs.

In April 1979

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

Proper tools was needed so Bjarne started to write a preprocessor to C that added Simula like classes to C.

The main motivation and aim for "C with Classes" was to create better support for modularity and concurrency, heavily inspired by Simula.

Evolution of C++ by example

```
#define MAXVAL 4

int sp = 0;
double val[MAXVAL];

double err(msg)
    char *msg;
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}

double push(f)
    double f;
{
    if (sp < MAXVAL)
        return val[sp++] = f;
    else
        return err("stack full");
}

double pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

```
#define MAXVAL 4

int sp = 0;
double val[MAXVAL];

double err(msg)
    char *msg;
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}

double push(f)
    double f;
{
    if (sp < MAXVAL)
        return val[sp++] = f;
    else
        return err("stack full");
}

double pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

This is an example of a program written in old style K&R (as of 1978)

This is an example of a program written in old style K&R (as of 1978)

```
#define MAXVAL 4

int sp = 0;
double val[MAXVAL];

double err(msg)
    char *msg;
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}

double push(f)
    double f;
{
    if (sp < MAXVAL)
        return val[sp++] = f;
    else
        return err("stack full");
}

double pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

```
#define MAXVAL 4

struct stack {
    int sp;
    double val[MAXVAL];
};

double err(s, msg)
    struct stack *s;
    char *msg;
{
    printf("error: %s\n", msg);
    s->sp = 0;
    return 0;
}

double push(s,f)
    struct stack *s;
    double f;
{
    if (s->sp < MAXVAL)
        return s->val[s->sp++] = f;
    else
        return err(s,"stack full");
}

double pop(s)
    struct stack *s;
{
    if (s->sp > 0)
        return s->val[--s->sp];
    else
        return err(s,"stack empty");
}
```

```c
#define MAXVAL 4

int sp = 0;
double val[MAXVAL];

double err(msg)
    char *msg;
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}

double push(f)
    double f;
{
    if (sp < MAXVAL)
        return val[sp++] = f;
    else
        return err("stack full");
}

double pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

This is an example of a program written in old style K&R (as of 1978)

```c
#define MAXVAL 4

struct stack {
    int sp;
    double val[MAXVAL];
};

double err(s, msg)
    struct stack *s;
    char *msg;
{
    printf("error: %s\n", msg);
    s->sp = 0;
    return 0;
}

double push(s,f)
    struct stack *s;
    double f;
{
    if (s->sp < MAXVAL)
        return s->val[s->sp++] = f;
    else
        return err(s,"stack full");
}

double pop(s)
    struct stack *s;
{
    if (s->sp > 0)
        return s->val[--s->sp];
    else
        return err(s,"stack empty");
}
```

Here is a “naive” ADT version of the same program.

```c
#define MAXVAL 4

int sp = 0;
double val[MAXVAL];

double err(msg)
    char *msg;
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}

double push(f)
    double f;
{
    if (sp < MAXVAL)
        return val[sp++] = f;
    else
        return err("stack full");
}

double pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

This is an example of a program written in old style K&R (as of 1978)

```c
#define MAXVAL 4

struct stack {
    int sp;
    double val[MAXVAL];
};

double err(s, msg)
    struct stack *s;
    char *msg;
{
    printf("error: %s\n", msg);
    s->sp = 0;
    return 0;
}

double push(s,f)
    struct stack *s;
    double f;
{
    if (s->sp < MAXVAL)
        return s->val[s->sp++] = f;
    else
        return err(s,"stack full");
}

double pop(s)
    struct stack *s;
{
    if (s->sp > 0)
        return s->val[--s->sp];
    else
        return err(s,"stack empty");
}
```

Here is a “naive” ADT version of the same program.

```c
#define MAXVAL 4

int sp = 0;
double val[MAXVAL];

double err(msg)
    char *msg;
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}

double push(f)
    double f;
{
    if (sp < MAXVAL)
        return val[sp++] = f;
    else
        return err("stack full");
}

double pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

This is an example of a program written in old style K&R (as of 1978)

```c
#define MAXVAL 4

struct stack {
    int sp;
    double val[MAXVAL];
};

double err(s, msg)
    struct stack *s;
    char *msg;
{
    printf("error: %s\n", msg);
    s->sp = 0;
    return 0;
}

double push(s,f)
    struct stack *s;
    double f;
{
    if (s->sp < MAXVAL)
        return s->val[s->sp++] = f;
    else
        return err(s,"stack full");
}

double pop(s)
    struct stack *s;
{
    if (s->sp > 0)
        return s->val[--s->sp];
    else
        return err(s,"stack empty");
}
```

Here is a “naive” ADT version of the same program.

```c
#define MAXVAL 4

int sp = 0;
double val[MAXVAL];

double err(msg)
    char *msg;
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}

double push(f)
    double f;
{
    if (sp < MAXVAL)
        return val[sp++] = f;
    else
        return err("stack full");
}

double pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

This is an example of a program written in old style K&R (as of 1978)

Here is a “naive” ADT version of the same program.

```c
#define MAXVAL 4

struct stack {
    int sp;
    double val[MAXVAL];
};

double err(s, msg)
    struct stack *s;
    char *msg;
{
    printf("error: %s\n", msg);
    s->sp = 0;
    return 0;
}

double push(s,f)
    struct stack *s;
    double f;
{
    if (s->sp < MAXVAL)
        return s->val[s->sp++] = f;
    else
        return err(s,"stack full");
}

double pop(s)
    struct stack *s;
{
    if (s->sp > 0)
        return s->val[--s->sp];
    else
        return err(s,"stack empty");
}
```

```c
struct stack {
    int sp;
    int max;
    double *val;
};

struct stack *new_stack(size)
    int size;
{
    struct stack *s = malloc(sizeof(struct stack));
    s->max = size;
    s->val = malloc(sizeof(double) * size);
    s->sp = 0;
    return s;
}

void delete_stack(s)
    struct stack *s;
{
    free(s->val);
    free(s);
}
```

```c
double err(s, msg)
    struct stack *s;
    char *msg;
{
    printf("error: %s\n", msg);
    s->sp = 0;
    return 0;
}

double push(s,f)
    struct stack *s;
    double f;
{
    if (s->sp < s->max)
        return s->val[s->sp++] = f;
    else
        return err(s,"stack full");
}

double pop(s)
    struct stack *s;
{
    if (s->sp > 0)
        return s->val[--s->sp];
    else
        return err(s,"stack empty");
}
```

```c
struct stack {
    int sp;
    int max;
    double *val;
};

struct stack *new_stack(size)
    int size;
{
    struct stack *s = malloc(sizeof(struct stack));
    s->max = size;
    s->val = malloc(sizeof(double) * size);
    s->sp = 0;
    return s;
}

void delete_stack(s)
    struct stack *s;
{
    free(s->val);
    free(s);
}
```

```c
double err(s, msg)
    struct stack *s;
    char *msg;
{
    printf("error: %s\n", msg);
    s->sp = 0;
    return 0;
}

double push(s,f)
    struct stack *s;
    double f;
{
    if (s->sp < s->max)
        return s->val[s->sp++] = f;
    else
        return err(s,"stack full");
}

double pop(s)
    struct stack *s;
{
    if (s->sp > 0)
        return s->val[--s->sp];
    else
        return err(s,"stack empty");
}
```

This is a “proper” ADT version of a stack written in old K&R style

```c
struct stack {
    int sp;
    int max;
    double *val;
};

struct stack *new_stack(size)
    int size;
{
    struct stack *s = malloc(sizeof(struct stack));
    s->max = size;
    s->val = malloc(sizeof(double) * size);
    s->sp = 0;
    return s;
}

void delete_stack(s)
    struct stack *s;
{
    free(s->val);
    free(s);
}
```

```c
double err(s, msg)
    struct stack *s;
    char *msg;
{
    printf("error: %s\n", msg);
    s->sp = 0;
    return 0;
}

double push(s,f)
    struct stack *s;
    double f;
{
    if (s->sp < s->max)
        return s->val[s->sp++] = f;
    else
        return err(s,"stack full");
}

double pop(s)
    struct stack *s;
{
    if (s->sp > 0)
        return s->val[--s->sp];
    else
        return err(s,"stack empty");
}
```

This is a "proper" ADT version of a stack written in old K&R style

```c
struct stack {
    int sp;
    int max;
    double *val;
};

struct stack *new_stack(size)
    int size;
{
    struct stack *s = malloc(sizeof(struct stack));
    s->max = size;
    s->val = malloc(sizeof(double) * size);
    s->sp = 0;
    return s;
}

void delete_stack(s)
    struct stack *s;
{
    free(s->val);
    free(s);
}
```

?

```c
double err(s, msg)
    struct stack *s;
    char *msg;
{
    printf("error: %s\n", msg);
    s->sp = 0;
    return 0;
}

double push(s,f)
    struct stack *s;
    double f;
{
    if (s->sp < s->max)
        return s->val[s->sp++] = f;
    else
        return err(s,"stack full");
}

double pop(s)
    struct stack *s;
{
    if (s->sp > 0)
        return s->val[--s->sp];
    else
        return err(s,"stack empty");
}
```

This is a “proper” ADT version of a stack written in old K&R style

There was a need for something more like Simula.

“C with Classes” was conceived and born.
(October 1979 - March 1980)

Bjarnes primary goal was to add support for
modularity and concurrency in C

```
class stack {
private:
    int sp;
    int max;
    double * val;
    // void call()
    // void return()
public:
    new(int);
    delete();
    double err(char *);
    double push(double);
    double pop();
};

stack.new(size)
    int size;
{
    max = size;
    val = malloc(sizeof(double) * size);
    sp = 0;
}

stack.delete()
{
    free(val);
}
```

```
double stack.err(msg)
    char * msg;
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}

double stack.push(f)
    double f;
{
    if (sp < max)
        return val[sp++] = f;
    else
        return err("stack full");
}

double stack.pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

```
class stack {
private:
    int sp;
    int max;
    double * val;
    // void call()
    // void return()
public:
    new(int);
    delete();
    double err(char *);
    double push(double);
    double pop();
};

stack.new(size)
    int size;
{
    max = size;
    val = malloc(sizeof(double) * size);
    sp = 0;
}

stack.delete()
{
    free(val);
}
```

```
double stack.err(msg)
    char * msg;
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}

double stack.push(f)
    double f;
{
    if (sp < max)
        return val[sp++] = f;
    else
        return err("stack full");
}

double stack.pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

An example of how a "C with Classes" implementation of stack might have looked like back in 1980.

```
class stack {
private:
    int sp;
    int max;
    double * val;
    // void call()
    // void return()
public:
    new(int);
    delete();
    double err(char *);
    double push(double);
    double pop();
};

stack.new(size)
    int size;
{
    max = size;
    val = malloc(sizeof(double) * size);
    sp = 0;
}

stack.delete()
{
    free(val);
}
```

```
double stack.err(msg)
    char * msg;
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}

double stack.push(f)
    double f;
{
    if (sp < max)
        return val[sp++] = f;
    else
        return err("stack full");
}

double stack.pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

```
class stack {
private:
    int sp;
    int max;
    double * val;
    // void call()
    // void return()
public:
    new(int);
    delete();
    double err(char *);
    double push(double);
    double pop();
};

stack.new(size)
    int size;
{
    max = size;
    val = malloc(sizeof(double) * size);
    sp = 0;
}

stack.delete()
{
    free(val);
}
```

```
double stack.err(msg)
    char * msg;
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}

double stack.push(f)
    double f;
{
    if (sp < max)
        return val[sp++] = f;
    else
        return err("stack full");
}

double stack.pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

Notice the syntax for constructor and destructor

```
class stack {
private:
    int sp;
    int max;
    double * val;
    // void call()
    // void return()
public:
    new(int);
    delete();
    double err(char *);
    double push(double);
    double pop();
};

stack.new(size)
    int size;
{
    max = size;
    val = malloc(sizeof(double) * size);
    sp = 0;
}

stack.delete()
{
    free(val);
}
```

```
double stack.err(msg)
    char * msg;
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}

double stack.push(f)
    double f;
{
    if (sp < max)
        return val[sp++] = f;
    else
        return err("stack full");
}

double stack.pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

```
class stack {
private:
    int sp;
    int max;
    double * val;
    // void call()
    // void return()
public:
    new(int);
    delete();
    double err(char *);
    double push(double);
    double pop();
};

stack.new(size)
    int size;
{
    max = size;
    val = malloc(sizeof(double) * size);
    sp = 0;
}

stack.delete()
{
    free(val);
}
```

```
double stack.err(msg)
    char * msg;
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}

double stack.push(f)
    double f;
{
    if (sp < max)
        return val[sp++] = f;
    else
        return err("stack full");
}

double stack.pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

Eearly version of "C with Classes" also had "magic" call() and return() that could be invoked before and after every function call.

```
class stack {
private:
    int sp;
    int max;
    double * val;
    // void call()
    // void return()
public:
    new(int);
    delete();
    double err(char *);
    double push(double);
    double pop();
};

stack.new(size)
    int size;
{
    max = size;
    val = malloc(sizeof(double) * size);
    sp = 0;
}

stack.delete()
{
    free(val);
}
```

```
double stack.err(msg)
    char * msg;
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}

double stack.push(f)
    double f;
{
    if (sp < max)
        return val[sp++] = f;
    else
        return err("stack full");
}

double stack.pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

```
class stack {
private:
    int sp;
    int max;
    double * val;
    // void call()
    // void return()
public:
    new(int);
    delete();
    double err(char *);
    double push(double);
    double pop();
};

stack.new(size)
    int size;
{
    max = size;
    val = malloc(sizeof(double) * size);
    sp = 0;
}

stack.delete()
{
    free(val);
}
```

```
double stack.err(msg)
    char * msg;
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}

double stack.push(f)
    double f;
{
    if (sp < max)
        return val[sp++] = f;
    else
        return err("stack full");
}

double stack.pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

```
class stack {
private:
    int sp;
    int max;
    double * val;
    // void call()
    // void return()
public:
    new(int);
    delete();
    double err(char *);
    double push(double);
    double pop();
};

stack.new(size)
    int size;
{
    max = size;
    val = malloc(sizeof(double) * size);
    sp = 0;
}

stack.delete()
{
    free(val);
}
```

```
double stack.err(msg)
    char * msg;
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}

double stack.push(f)
    double f;
{
    if (sp < max)
        return val[sp++] = f;
    else
        return err("stack full");
}

double stack.pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

The syntax for accessing members of a class was not optimal.

```cpp
class stack {
private:
    int sp;
    int max;
    double * val;


public:
    stack(int size);
    ~stack();
    double err(char * msg);
    double push(double f);
    double pop();
};

stack::stack(int size) : sp(0), max(size),
                         val(new double[size]) {
}


stack::~stack()
{
    delete[] val;
}
```

```cpp
double stack::err(char * msg)
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}


double stack::push(double f)
{
    if (sp < max)
        return val[sp++] = f;
    else
        return err("stack full");
}


double stack::pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

```
class stack {
private:
    int sp;
    int max;
    double * val;


public:
    stack(int size);
    ~stack();
    double err(char * msg);
    double push(double f);
    double pop();
};

stack::stack(int size) : sp(0), max(size),
                         val(new double[size]) {
}


stack::~stack()
{
    delete[] val;
}
```

```
double stack::err(char * msg)
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}


double stack::push(double f)
{
    if (sp < max)
        return val[sp++] = f;
    else
        return err("stack full");
}


double stack::pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

After some iterations on the syntax of the language, it ended up like this.

```
class stack {
private:
    int sp;
    int max;
    double * val;


public:
    stack(int size);
    ~stack();
    double err(char * msg);
    double push(double f);
    double pop();
};

stack::stack(int size) : sp(0), max(size),
                         val(new double[size]) {
}


stack::~stack()
{
    delete[] val;
}
```

```
double stack::err(char * msg)
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}


double stack::push(double f)
{
    if (sp < max)
        return val[sp++] = f;
    else
        return err("stack full");
}


double stack::pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

```
class stack {
private:
    int sp;
    int max;
    double * val;


public:
    stack(int size);
    ~stack();
    double err(char * msg);
    double push(double f);
    double pop();
};

stack::stack(int size) : sp(0), max(size),
                         val(new double[size]) {
}


stack::~stack()
{
    delete[] val;
}
```

```
double stack::err(char * msg)
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}


double stack::push(double f)
{
    if (sp < max)
        return val[sp++] = f;
    else
        return err("stack full");
}


double stack::pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

This version will compile fine with modern C++ compilers as well.

```
class stack {
private:
    int sp;
    int max;
    double * val;


public:
    stack(int size);
    ~stack();
    double err(char * msg);
    double push(double f);
    double pop();
};

stack::stack(int size) : sp(0), max(size),
                         val(new double[size]) {
}


stack::~stack()
{
    delete[] val;
}
```

```
double stack::err(char * msg)
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}


double stack::push(double f)
{
    if (sp < max)
        return val[sp++] = f;
    else
        return err("stack full");
}


double stack::pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

```
class stack {
private:
    int sp;
    int max;
    double * val;


public:
    stack(int size);
    ~stack();
    double err(char * msg);
    double push(double f);
    double pop();
};

stack::stack(int size) : sp(0), max(size),
                         val(new double[size]) {
}


stack::~stack()
{
    delete[] val;
}
```

```
double stack::err(char * msg)
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}


double stack::push(double f)
{
    if (sp < max)
        return val[sp++] = f;
    else
        return err("stack full");
}


double stack::pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

However, once again we have a performance tradeoff. The cost of a function call is significant so there was a need for inlining the member functions

```
class stack {
private:
    int sp;
    int max;
    double * val;


public:
    stack(int size);
    ~stack();
    double err(char * msg);
    double push(double f);
    double pop();
};

stack::stack(int size) : sp(0), max(size),
                         val(new double[size]) {
}


stack::~stack()
{
    delete[] val;
}
```

```
double stack::err(char * msg)
{
    printf("error: %s\n", msg);
    sp = 0;
    return 0;
}


double stack::push(double f)
{
    if (sp < max)
        return val[sp++] = f;
    else
        return err("stack full");
}


double stack::pop()
{
    if (sp > 0)
        return val[--sp];
    else
        return err("stack empty");
}
```

However, once again we have a performance tradeoff. The cost of a function call is significant so there was a need for inlining the member functions

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
};
```

```
class stack {
public:
    int sp;
private:
    int max;
    double * val;
public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
};
```

```
class mystack : public stack {
public:
    mystack(int size=4) : stack(4) {}
    int size() {
        return sp;
    }
};
```

And of course, inheritance was also useful.

```
class stack {
public:
    int sp;
private:
    int max;
    double * val;
public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
};
```

```
class mystack : public stack {
public:
    mystack(int size=4) : stack(4) {}
    int size() {
        return sp;
    }
};
```

```
class stack {
public:
    int sp;
private:
    int max;
    double * val;
public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
};
```

```
class mystack : public stack {
public:
    mystack(int size=4) : stack(4) {}
    int size() {
        return sp;
    }
};
```

```
class stack {
public:
    int sp;
private:
    int max;
    double * val;
public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
};
```

```
class mystack : public stack {
public:
    mystack(int size=4) : stack(4) {}
    int size() {
        return sp;
    }
};
```

Notice here that sp member needs to be public for the derived class to access it. Protected members was introduced in CFront 1.2 (1987)

```
class stack {
protected:
    int sp;
private:
    int max;
    double * val;
public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
};
```

```
class mystack : public stack {
public:
    mystack(int size=4) : stack(4) {}
    int size() {
        return sp;
    }
};
```

```
class stack {
protected:
    int sp;
private:
    int max;
    double * val;
public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    virtual ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    virtual char * name() {
        return "stack";
    }
};
```

```
class mystack : public stack {
public:
    mystack(int size=4) : stack(4) {}
    int size() {
        return sp;
    }
    virtual char * name() {
        return "mystack";
    }
};
```

C with Classes did not have virtual functions, but it was one of the first features to be introduced in C++ (~1984). Bjarne did not claim support for object-orientation until C++ had virtual functions.

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
};
```

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(char * valuestr) {
        double f = atof(valuestr);
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
};
```

Function name overloading was also introduced…

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(char * valuestr) {
        double f = atof(valuestr);
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
};
```

Function name overloading was also introduced…

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(char * valuestr) {
        double f = atof(valuestr);
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double operator+(double f) {
        return push(f);
    }
};
```

Function name overloading was also introduced…

and operator overloading.

```cpp
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(char * valuestr) {
        double f = atof(valuestr);
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double operator+(double f) {
        return push(f);
    }
};
```

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(char * valuestr) {
        double f = atof(valuestr);
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double operator+(double f) {
        return push(f);
    }
};
```

huh?

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(char * valuestr) {
        double f = atof(valuestr);
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double operator+(double f) {
        return push(f);
    }
};
```

huh?

```
stack s;
s+42; // equivalent to s.push(42);
…
```

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(char * valuestr) {
        double f = atof(valuestr);
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double operator+(double f) {
        return push(f);
    }
};
```

huh?

```
stack s;
s+42; // equivalent to s.push(42);
…
```

- You can write bad programs in any language.

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
```

```
    double push(char * valuestr) {
        double f = atof(valuestr);
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double operator+(double f) {
        return push(f);
    }
};
```

huh?

```
stack s;
s+42; // equivalent to s.push(42);
…
```

- You can write bad programs in any language.
- Trust the programmer.

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(char * valuestr) {
        double f = atof(valuestr);
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double operator+(double f) {
        return push(f);
    }
};
```

huh?

```
stack s;
s+42; // equivalent to s.push(42);
…
```

- You can write bad programs in any language.
- Trust the programmer.
- Don't try to force people.

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(char * valuestr) {
        double f = atof(valuestr);
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double operator+(double f) {
        return push(f);
    }
};
```

huh?

```
stack s;
s+42; // equivalent to s.push(42);
…
```

- You can write bad programs in any language.
- Trust the programmer.
- Don't try to force people.
- It is more important to allow a useful feature than to prevent every misuse.

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(char * valuestr) {
        double f = atof(valuestr);
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double operator+(double f) {
        return push(f);
    }
};
```

huh?

```
stack s;
s+42; // equivalent to s.push(42);
…
```

- You can write bad programs in any language.
- Trust the programmer.
- Don't try to force people.
- It is more important to allow a useful feature than to prevent every misuse.

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(char * valuestr) {
        double f = atof(valuestr);
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }

};
```

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
```

```
    double push(mystring valuestr) {
        double f = atof(valuestr);
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
};
```

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(mystring & valuestr) {
        double f = atof(valuestr);
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
};
```

references was motivated by overloading.

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(const mystring & valuestr) {
        double f = atof(valuestr);
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
};
```

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(const mystring & valuestr) {
        double f = valuestr;
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
};
```

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        printf("error: %s\n", msg);
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(const mystring & valuestr) {
        double f = valuestr;
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
};
```

```
stack s;
s.push("3.14");
…
```

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        cout << "error: “ << msg << ‘\n’;
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(const mystring & valuestr) {
        double f = valuestr;
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
};
```

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        cout << "error: " << msg << '\n';
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
```

```
    double push(const mystring & valuestr) {
        double f = valuestr;
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
};
```

With operator overloading and references in place, streams was introduced as type-safe way to do I/O.

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        cout << "error: " << msg << '\n';
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(const mystring & valuestr) {
        double f = valuestr;
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
};
```

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        cout << "error: “ << msg << ‘\n’;
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
    double push(const mystring & valuestr) {
        double f = valuestr;
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
};
```

This is C++ as of 1986

- 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

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        cout << "error: “ << msg << ‘\n’;
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
};
```

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    double err(char * msg) {
        cout << "error: “ << msg << ‘\n’;
        sp = 0;
        return 0;
    }
    double push(double f) {
        if (sp < max)
            return val[sp++] = f;
        else
            return err("stack full");
    }
    double pop() {
        if (sp > 0)
            return val[--sp];
        else
            return err("stack empty");
    }
};
```

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    void err(char * msg) {
        cout << "error: " << msg << '\n';
        exit(-1);
    }

    void push(double f) {
        if (sp >= max)
            err("stack full");
        val[sp++] = f;
    }

    double pop() {
        if (sp <= 0)
            err("stack empty");
        return val[--sp];
    }

};
```

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    void err(char * msg) {
        cout << "error: " << msg << '\n';
        exit(-1);
    }

    void push(double f) {
        if (sp >= max)
            err("stack full");
        val[sp++] = f;
    }

    double pop() {
        if (sp <= 0)
            err("stack empty");
        return val[--sp];
    }

};
```

This is a big decision for a utility class. Really exit the program?

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    void err(char * msg) {
        cout << "error: " << msg << '\n';
        exit(-1);
    }

    void push(double f) {
        if (sp >= max)
            err("stack full");
        val[sp++] = f;
    }

    double pop() {
        if (sp <= 0)
            err("stack empty");
        return val[--sp];
    }

};
```

This is a big decision for a utility class. Really exit the program?

It is better to let the user decide what should happen when an error occurs.

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }





    void push(double f) {
        if (sp >= max)
            throw "stack full";
        val[sp++] = f;
    }

    double pop() {
        if (sp <= 0)
            throw "stack empty";
        return val[--sp];
    }

};
```

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }




    void push(double f) {
        if (sp >= max)
            throw "stack full";
        val[sp++] = f;
    }

    double pop() {
        if (sp <= 0)
            throw "stack empty";
        return val[--sp];
    }

};
```

Exceptions! (defined in 1990, first implemented in 1992)

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    void push(double f) {
        if (sp >= max)
            throw "stack full";
        val[sp++] = f;
    }
    double pop() {
        if (sp <= 0)
            throw "stack empty";
        return val[--sp];
    }
};
```

```
class stack {
private:
    int sp;
    int max;
    double * val;

public:
    stack(int size = 4) : sp(0), max(size),
                          val(new double[size]) {
    }
    ~stack() {
        delete[] val;
    }
    void push(double f) {
        if (sp >= max)
            throw "stack full";
        val[sp++] = f;
    }
    double pop() {
        if (sp <= 0)
            throw "stack empty";
        return val[--sp];
    }
};
```

```cpp
class stack_double {
private:
    int sp;
    int max;
    double * val;

public:
    stack_double(int size = 4) : sp(0), max(size),
                                 val(new double[size]) {
    }
    ~stack_double() {
        delete[] val;
    }
    void push(double v) {
        if (sp >= max)
            throw "stack full";
        val[sp++] = v;
    }
    double pop() {
        if (sp <= 0)
            throw "stack empty";
        return val[--sp];
    }
};
```

```cpp
class stack_int {
private:
    int sp;
    int max;
    int * val;

public:
    stack_int(int size = 4) : sp(0), max(size),
                              val(new int[size]) {
    }
    ~stack_int() {
        delete[] val;
    }
    void push(int v) {
        if (sp >= max)
            throw "stack full";
        val[sp++] = v;
    }
    int pop() {
        if (sp <= 0)
            throw "stack empty";
        return val[--sp];
    }
};
```

```cpp
class stack_double {
private:
    int sp;
    int max;
    double * val;

public:
    stack_double(int size = 4) : sp(0), max(size),
                                 val(new double[size]) {
    }
    ~stack_double() {
        delete[] val;
    }
    void push(double v) {
        if (sp >= max)
            throw "stack full";
        val[sp++] = v;
    }
    double pop() {
        if (sp <= 0)
            throw "stack empty";
        return val[--sp];
    }
};
```

```cpp
class stack_int {
private:
    int sp;
    int max;
    int * val;

public:
    stack_int(int size = 4) : sp(0), max(size),
                              val(new int[size]) {
    }
    ~stack_int() {
        delete[] val;
    }
    void push(int v) {
        if (sp >= max)
            throw "stack full";
        val[sp++] = v;
    }
    int pop() {
        if (sp <= 0)
            throw "stack empty";
        return val[--sp];
    }
};
```

```cpp
template <typename T>
class stack {
private:
    int sp;
    int max;
    T * val;
public:
    stack(int size = 4) : sp(0), max(size),
                          val(new T[size]) {
    }
    ~stack() {
        delete[] val;
    }
    void push(T v) {
        if (sp >= max)
            throw "stack full";
        val[sp++] = v;
    }
    T pop() {
        if (sp <= 0)
            throw "stack empty";
        return val[--sp];
    }
};
```

```
template <typename T>
class stack {
private:
    int sp;
    int max;
    T * val;
public:
    stack(int size = 4) : sp(0), max(size),
                          val(new T[size]) {
    }
    ~stack() {
        delete[] val;
    }
    void push(T v) {
        if (sp >= max)
            throw "stack full";
        val[sp++] = v;
    }
    T pop() {
        if (sp <= 0)
            throw "stack empty";
        return val[--sp];
    }
};
```

Templates
(appeared in CFront 3.0, 1991)

```
namespace mystuff {

template <typename T>
class stack {
private:
    int sp;
    int max;
    T * val;
public:
    stack(int size = 4) : sp(0), max(size),
                          val(new T[size]) {
    }
    ~stack() {
        delete[] val;
    }
    void push(T v) {
        if (sp >= max)
            throw "stack full";
        val[sp++] = v;
    }
    T pop() {
        if (sp <= 0)
            throw "stack empty";
        return val[--sp];
    }
};

}
```

```
namespace mystuff {

template <typename T>
class stack {
private:
    int sp;
    int max;
    T * val;
public:
    stack(int size = 4) : sp(0), max(size),
                          val(new T[size]) {
    }
    ~stack() {
        delete[] val;
    }
    void push(T v) {
        if (sp >= max)
            throw "stack full";
        val[sp++] = v;
    }
    T pop() {
        if (sp <= 0)
            throw "stack empty";
        return val[--sp];
    }
};

}
```

Namespaces
(~1993)

```
#include <vector>

namespace mystuff {
  class stack {
  private:
      size_t max;
      std::vector<double> vec;
  public:
      stack(size_t size = 4) : max(size), vec() {
      }
      void push(double v) {
          if (vec.size() >= max)
              throw "stack full";
          vec.push_back(v);
      }
      double pop() {
          if (vec.size() == 0)
              throw "stack empty";
          double v = vec.back();
          vec.pop_back();
          return v;
      }
  };
}
```

```
#include <vector>

namespace mystuff {
  class stack {
  private:
      size_t max;
      std::vector<double> vec;
  public:
      stack(size_t size = 4) : max(size), vec() {
      }
      void push(double v) {
          if (vec.size() >= max)
              throw "stack full";
          vec.push_back(v);
      }
      double pop() {
          if (vec.size() == 0)
              throw "stack empty";
          double v = vec.back();
          vec.pop_back();
          return v;
      }
  };
}
```

Namespaces and templates pawed the way for the Standard Template Library (included in C++98)

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

```
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;
```

Modern C++ by Example

unless specified otherwise, all these code snippets should compile cleanly with a modern C++ compiler

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (std::vector<int>::const_iterator it = log.cbegin();
         it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

Consider this small toy program...

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (std::vector<int>::const_iterator it = log.cbegin();
         it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (std::vector<int>::const_iterator it = log.cbegin();
         it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

Consider this small toy program...

```
$ g++-4.9 -std=c++1y -Wall -Wextra -pedantic -Werror foo.cpp && ./a.out
20
24
37
42
23
45
37
$
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (std::vector<int>::const_iterator it = log.cbegin();
         it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (std::vector<int>::const_iterator it = log.cbegin();
         it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

This shows a "traditional" way of looping through a collection of objects.

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (std::vector<int>::const_iterator it = log.cbegin();
         it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (std::vector<int>::const_iterator it = log.cbegin();
         it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

But why do we have to write all this stuff? In this case, wouldn't it be nice if the compiler could just figure out which type we need to store the return value from `log.cbegin()`?

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (std::vector<int>::const_iterator it = log.cbegin();
         it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (std::vector<int>::const_iterator it = log.cbegin();
         it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (decltype(log.cbegin()) it = log.cbegin();
         it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (decltype(log.cbegin()) it = log.cbegin();
         it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

`decltype` gives us type deduction in C++. Or even better...

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (auto it = log.cbegin();
         it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (auto it = log.cbegin();
         it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

We can just use the new meaning of the keyword `auto`

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (auto it = log.cbegin();
         it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (auto it = log.cbegin();
         it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (auto it = log.cbegin(); it != log.cend(); ++it)

        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (auto it = log.cbegin(); it != log.cend(); ++it)

        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (auto it = log.cbegin(); it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```cpp
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (auto it = log.cbegin(); it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (auto it = log.cbegin(); it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (auto it = log.cbegin(); it != log.cend(); ++it)
        transmit_item(*it);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

Looping through an array like this is something C++ programmers often do. So the language now provides a new way of looping through ranges of objects.

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (auto i : log)
        transmit_item(i);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (auto i : log)
        transmit_item(i);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

Introducing:
range based for-loop.

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (auto i : log)
        transmit_item(i);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (auto i : log)
        transmit_item(i);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

Sometimes we might want to save some object copies by writing...

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (auto i : log)
        transmit_item(i);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

Sometimes we might want to save some object copies by writing...

```cpp
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (const auto & i : log)
        transmit_item(i);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (const auto & i : log)
        transmit_item(i);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (const auto & i : log)
        transmit_item(i);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

But even for simple loops like this you will often see that **STL algorithms** are used instead.

```
#include <iostream>
#include <vector>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (const auto & i : log)
        transmit_item(i);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (const auto & i : log)
        transmit_item(i);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    for (const auto & i : log)
        transmit_item(i);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);

}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

Suppose we would like to sort the array before transmitting the items...

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

Suppose we would like to sort the array before transmitting the items...

First we make a local copy of the log through a **pass-by-value**

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

Suppose we would like to sort the array before transmitting the items...

First we make a local copy of the log through a **pass-by-value**

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

But wait! What if the log has million of entries? Perhaps we should do **pass-by-reference** instead?

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

But wait! What if the log has million of entries? Perhaps we should do **pass-by-reference** instead?

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int>& log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> & log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> & log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

This works. But, in this case, it would be even better if we had an option to pass the **ownership** of the log to `transmit_log` by reference so it can do whatever it wants.

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> & log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

This works. But, in this case, it would be even better if we had an option to pass the **ownership** of the log to `transmit_log` by reference so it can do whatever it wants.

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> & log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(log);
}
```

This works. But, in this case, it would be even better if we had an option to pass the **ownership** of the log to `transmit_log` by reference so it can do whatever it wants.

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

This is an **rvalue reference**.

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

This is an **rvalue reference**.

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

This is an **rvalue reference**.

And here we basically say: Just take this data object, it is yours, do whatever you want with it. I promise to never assume anything about it’s content after you are done.

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

This is an **rvalue reference**.

And here we basically say: Just take this data object, it is yours, do whatever you want with it. I promise to never assume anything about it's content after you are done.

rvalue references and the corresponding **move semantics** are very important contributions to modern C++. It reduces the need to create copies of objects while still being able to use **value semantics** as a programming style (ie, avoiding the need to use pointers for everything).

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

Typical for most algorithms in the C++ library is that you can adapt them to your own needs. Let's try to change the sorting order by writing our own comparator function.

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

Typical for most algorithms in the C++ library is that you can adapt them to your own needs. Let's try to change the sorting order by writing our own comparator function.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

Typical for most algorithms in the C++ library is that you can adapt them to your own needs. Let's try to change the sorting order by writing our own comparator function.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static bool mycomp(int lhs, int rhs)
{
    return lhs > rhs;
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log), mycomp);
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static bool mycomp(int lhs, int rhs)
{
    return lhs > rhs;
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log), mycomp);
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static bool mycomp(int lhs, int rhs)
{
    return lhs > rhs;
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log), mycomp);
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

This is a typical way to introduce a **strategy** into an existing algorithm. (Here you could have used `std::greater` as well, but if you want something more complex you need to write it yourself.)

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static bool mycomp(int lhs, int rhs)
{
    return lhs > rhs;
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log), mycomp);
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

This is a typical way to introduce a **strategy** into an existing algorithm. (Here you could have used `std::greater` as well, but if you want something more complex you need to write it yourself.)

It depends on the context, but in this case you might want to allow the caller to pass the strategy in into `transmit_log()`

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static bool mycomp(int lhs, int rhs)
{
    return lhs > rhs;
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log), mycomp);
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

This is a typical way to introduce a **strategy** into an existing algorithm. (Here you could have used `std::greater` as well, but if you want something more complex you need to write it yourself.)

It depends on the context, but in this case you might want to allow the caller to pass the strategy in into `transmit_log()`

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static bool mycomp(int lhs, int rhs)
{
    return lhs > rhs;
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log), mycomp);
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

This is a typical way to introduce a **strategy** into an existing algorithm. (Here you could have used `std::greater` as well, but if you want something more complex you need to write it yourself.)

It depends on the context, but in this case you might want to allow the caller to pass the strategy in into `transmit_log()`

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static bool mycomp(int lhs, int rhs)
{
    return lhs > rhs;
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log), mycomp);
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

This is a typical way to introduce a **strategy** into an existing algorithm. (Here you could have used `std::greater` as well, but if you want something more complex you need to write it yourself.)

It depends on the context, but in this case you might want to allow the caller to pass the strategy in into `transmit_log()`

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static bool mycomp(int lhs, int rhs)
{
    return lhs > rhs;
}

static void transmit_log(std::vector<int> && log, bool comp(int, int))
{
    std::sort(std::begin(log), std::end(log), comp);
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), mycomp);
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static bool mycomp(int lhs, int rhs)
{
    return lhs > rhs;
}

static void transmit_log(std::vector<int> && log, bool comp(int, int))
{
    std::sort(std::begin(log), std::end(log), comp);
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), mycomp);
}
```

This is an example of
parameterize from above

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static bool mycomp(int lhs, int rhs)
{
    return lhs > rhs;
}

static void transmit_log(std::vector<int> && log, bool comp(int, int))
{
    std::sort(std::begin(log), std::end(log), comp);
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), mycomp);
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static bool mycomp(int lhs, int rhs)
{
    return lhs > rhs;
}

static void transmit_log(std::vector<int> && log, bool comp(int, int))
{
    std::sort(std::begin(log), std::end(log), comp);
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), mycomp);
}
```

I am now going to introduce function objects and lambdas. Let's first simplify the code, before introducing algorithms for filtering out and removing log values.

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static bool mycomp(int lhs, int rhs)
{
    return lhs > rhs;
}

static void transmit_log(std::vector<int> && log, bool comp(int, int))
{
    std::sort(std::begin(log), std::end(log), comp);
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), mycomp);
}
```

I am now going to introduce function objects and lambdas. Let's first simplify the code, before introducing algorithms for filtering out and removing log values.

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static bool mycomp(int lhs, int rhs)
{
    return lhs > rhs;
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static bool mycomp(int lhs, int rhs)
{
    return lhs > rhs;
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    struct filter {
        filter(int limit) : lim(limit) {}
        bool operator()(int i) { return i <= lim; };
        int lim;
    } myfilter(23);
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    struct filter {
        filter(int limit) : lim(limit) {}
        bool operator()(int i) { return i <= lim; };
        int lim;
    } myfilter(23);
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

Here we have created code to remove all log items that are 23 or below.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    struct filter {
        filter(int limit) : lim(limit) {}
        bool operator()(int i) { return i <= lim; };
        int lim;
    } myfilter(23);
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    struct filter {
        filter(int limit) : lim(limit) {}
        bool operator()(int i) { return i <= lim; };
        int lim;
    } myfilter(23);
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

Notice how we have created a "function" on the fly by overloading the call operator on an object. This is an example of a **function object**, sometimes called a **functor**.

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    struct filter {
        filter(int limit) : lim(limit) {}
        bool operator()(int i) { return i <= lim; };
        int lim;
    } myfilter(23);
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    struct filter {
        filter(int limit) : lim(limit) {}
        bool operator()(int i) { return i <= lim; };
        int lim;
    } myfilter(23);
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

Suppose we want to parameterize from above again, by passing in the limit.

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    struct filter {
        filter(int limit) : lim(limit) {}
        bool operator()(int i) { return i <= lim; };
        int lim;
    } myfilter(23);
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

Suppose we want to parameterize from above again, by passing in the limit.

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    struct filter {
        filter(int limit) : lim(limit) {}
        bool operator()(int i) { return i <= lim; };
        int lim;
    } myfilter(23);
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

Suppose we want to parameterize from above again, by passing in the limit.

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log)
{
    struct filter {
        filter(int limit) : lim(limit) {}
        bool operator()(int i) { return i <= lim; };
        int lim;
    } myfilter(23);
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log));
}
```

Suppose we want to parameterize from above again, by passing in the limit.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, int limit)
{
    struct filter {
        filter(int limit) : lim(limit) {}
        bool operator()(int i) { return i <= lim; };
        int lim;
    } myfilter(limit);
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), 23);
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, int limit)
{
    struct filter {
        filter(int limit) : lim(limit) {}
        bool operator()(int i) { return i <= lim; };
        int lim;
    } myfilter(limit);
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), 23);
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, int limit)
{
    struct filter {
        filter(int limit) : lim(limit) {}
        bool operator()(int i) { return i <= lim; };
        int lim;
    } myfilter(limit);
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), 23);
}
```

Such function objects are sometimes very useful. New in C++11 is a convenient syntax for creating these functions.

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, int limit)
{
    struct filter {
        filter(int limit) : lim(limit) {}
        bool operator()(int i) { return i <= lim; };
        int lim;
    } myfilter(limit);
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), 23);
}
```

Such function objects are sometimes very useful. New in C++11 is a convenient syntax for creating these functions.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, int limit)
{
    struct filter {
        filter(int limit) : lim(limit) {}
        bool operator()(int i) { return i <= lim; };
        int lim;
    } myfilter(limit);
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), 23);
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, int limit)
{
    struct filter {
        filter(int limit) : lim(limit) {}
        bool operator()(int i) { return i <= lim; };
        int lim;
    } myfilter(limit);
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), 23);
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, int limit)
{
    auto myfilter = [limit](int i) { return i <= limit; };
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), 23);
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, int limit)
{
    auto myfilter = [limit](int i) { return i <= limit; };
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), 23);
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, int limit)
{
    auto myfilter = [limit](int i) { return i <= limit; };
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), 23);
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, int limit)
{
    auto myfilter = [limit](int i) { return i <= limit; };
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), 23);
}
```

This is a lambda expression that creates a function object on the "fly". We are **capturing** the value of the variable `limit` and using it to initialize the function object.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, int limit)
{
    auto myfilter = [limit](int i) { return i <= limit; };
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), 23);
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, int limit)
{
    auto myfilter = [limit](int i) { return i <= limit; };
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), 23);
}
```

You can of course also pass function objects around as any other objects.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, int limit)
{
    auto myfilter = [limit](int i) { return i <= limit; };
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), 23);
}
```

You can of course also pass function objects around as any other objects.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, int limit)
{
    std::function<bool (int)> myfilter = [limit](int i) { return i <= limit; };
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), 23);
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, int limit)
{
    std::function<bool (int)> myfilter = [limit](int i) { return i <= limit; };
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), 23);
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, std::function<bool (int)> myfilter)
{

    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, std::function<bool (int)> myfilter)
{

    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, std::function<bool (int)> myfilter)
{

    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, std::function<bool (int)> myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, std::function<bool (int)> myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, std::function<bool (int)> myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

Basically anything that can be called with an `int` and returning a `bool` is OK. We can generalize the code with a template.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static void transmit_log(std::vector<int> && log, std::function<bool (int)> myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

Basically anything that can be called with an `int` and returning a `bool` is OK. We can generalize the code with a template.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Filt>
static void transmit_log(std::vector<int> && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Filt>
static void transmit_log(std::vector<int> && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Filt>
static void transmit_log(std::vector<int> && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

... and the same is true for the log. Anything that we can iterate over, and that contains some items that we can transmit should be fine.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Filt>
static void transmit_log(std::vector<int> && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

... and the same is true for the log. Anything that we can iterate over, and that contains some items that we can transmit should be fine.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

... and the same is true for the log. Anything that we can iterate over, and that contains some items that we can transmit should be fine.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

And while we are at it, let's generalize the code for `transmit_item` as well

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

And while we are at it, let's generalize the code for `transmit_item` as well

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item<int>);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log), transmit_item<int>);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<int>);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<int>);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

`transmit_log` and `transmit_item` are now **type independent code.** This is a fine example of **generic programming.** Notice how we can change both the type of the log items and the container and it should still work (given some restrictions)

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

`transmit_log` and `transmit_item` are now **type independent code**. This is a fine example of **generic programming**. Notice how we can change both the type of the log items and the container and it should still work (given some restrictions)

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

`transmit_log` and `transmit_item` are now **type independent code**. This is a fine example of **generic programming**. Notice how we can change both the type of the log items and the container and it should still work (given some restrictions)

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](int i) { return i <= 23; });
}
```

`transmit_log` and `transmit_item` are now **type independent code**. This is a fine example of **generic programming**. Notice how we can change both the type of the log items and the container and it should still work (given some restrictions)

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

```
#include <iostream>
#include <deque>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::deque<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

It would be nice to specify exactly what expectations we have to the types and objects that are passed into our generic code. A "poor man" solution is to use **type traits** and `static_assert`.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

It would be nice to specify exactly what expectations we have to the types and objects that are passed into our generic code. A "poor man" solution is to use **type traits** and `static_assert`.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

template <typename T>
static void transmit_item(T i)
{
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

It would be nice to specify exactly what expectations we have to the types and objects that are passed into our generic code. A "poor man" solution is to use **type traits** and `static_assert`.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>

template <typename T>
static void transmit_item(T i)
{
    static_assert(std::is_integral<T>::value, "integral type expected");
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>

template <typename T>
static void transmit_item(T i)
{
    static_assert(std::is_integral<T>::value, "integral type expected");
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>

template <typename T>
static void transmit_item(T i)
{
    static_assert(std::is_integral<T>::value, "integral type expected");
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

Here we will get an understandable compile error if the type of the log items are not of integral type. However, you can, with some work, define your own traits and constraints. Eg, something like this:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>

template <typename T>
static void transmit_item(T i)
{
    static_assert(std::is_integral<T>::value, "integral type expected");
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

Here we will get an understandable compile error if the type of the log items are not of integral type. However, you can, with some work, define your own traits and constraints. Eg, something like this:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>

template <typename T>
static void transmit_item(T i)
{
    static_assert(std::is_integral<T>::value, "integral type expected");
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

Here we will get an understandable compile error if the type of the log items are not of integral type. However, you can, with some work, define your own traits and constraints. Eg, something like this:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>
#include "mystuff"

template <typename T>
static void transmit_item(T i)
{
    static_assert(my::is_transmittable<T>::value, "transmittable type expected");
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

this is just an example that does not compile

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>
#include "mystuff"

template <typename T>
static void transmit_item(T i)
{
    static_assert(my::is_transmittable<T>::value, "transmittable type expected");
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

this is just an example that does not compile

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>
#include "mystuff"

template <typename T>
static void transmit_item(T i)
{
    static_assert(my::is_transmittable<T>::value, "transmittable type expected");
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

There are some proposals for the next versions of C++ to include better syntax for such constraints.

this is just an example that does not compile

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>
#include "mystuff"

template <typename T>
static void transmit_item(T i)
{
    static_assert(my::is_transmittable<T>::value, "transmittable type expected");
    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

There are some proposals for the next versions of C++ to include better syntax for such constraints.

this is just an example that does not compile

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>
#include "mystuff"

template <typename T> require Transmittable<T>
static void transmit_item(T i)
{

    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>
#include "mystuff"

template <typename T> require Transmittable<T>
static void transmit_item(T i)
{

    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>
#include "mystuff"

template <typename T> require Transmittable<T>
static void transmit_item(T i)
{

    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

X

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>
#include "mystuff"

template <Transmittable T>
static void transmit_item(T i)
{

    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

X

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>
#include "mystuff"

template <Transmittable T>
static void transmit_item(T i)
{

    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

X

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>
#include "mystuff"

template <Transmittable T>
static void transmit_item(T i)
{

    std::cout << i << std::endl;
    // ...
}

template <typename Log, typename Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

X

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>
#include "mystuff"

template <Transmittable T>
static void transmit_item(T i)
{

    std::cout << i << std::endl;
    // ...
}

template <Iterable Log, UnaryFunctionPredicate Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

X

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>
#include "mystuff"

template <Transmittable T>
static void transmit_item(T i)
{

    std::cout << i << std::endl;
    // ...
}

template <Iterable Log, UnaryFunctionPredicate Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>
#include "mystuff"

template <Transmittable T>
static void transmit_item(T i)
{

    std::cout << i << std::endl;
    // ...
}

template <Iterable Log, UnaryFunctionPredicate Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

This proposal is a step towards something called Concepts. I am not going to explain that, so let's clean up the code so I can show one final thing.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>
#include "mystuff"

template <Transmittable T>
static void transmit_item(T i)
{

    std::cout << i << std::endl;
    // ...
}

template <Iterable Log, UnaryFunctionPredicate Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <type_traits>
#include "mystuff"

template <Transmittable T>
static void transmit_item(T i)
{

    std::cout << i << std::endl;
    // ...
}

template <Iterable Log, UnaryFunctionPredicate Filt>
static void transmit_log(Log && log, Filt myfilter)
{
    log.erase(std::remove_if(std::begin(log), std::end(log), myfilter),
              std::end(log));
    std::sort(std::begin(log), std::end(log));
    std::for_each(std::begin(log), std::end(log),
                  transmit_item<typename Log::value_type>);
}

int main()
{
    using log_item_type = long;
    std::vector<log_item_type> log{20,24,37,42,23,45,37};
    transmit_log(std::move(log), [](log_item_type i) { return i <= 23; });
}
```

X

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    size_t items = transmit_log(log);
    std::cout << "# " << items << std::endl;
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    size_t items = transmit_log(log);
    std::cout << "# " << items << std::endl;
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    size_t items = transmit_log(log);
    std::cout << "# " << items << std::endl;
}
```

Transmitting the data probably takes some time, and we might want to do something else while waiting for the log to be transmitted. Let's simulate that, and show an example of how concurrency is supported in modern C++.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    size_t items = transmit_log(log);
    std::cout << "# " << items << std::endl;
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    size_t items = transmit_log(log);
    std::cout << "# " << items << std::endl;
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    size_t items = transmit_log(log);
    std::cout << "# " << items << std::endl;
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <thread>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(200));
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    size_t items = transmit_log(log);
    std::cout << "# " << items << std::endl;
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <thread>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(200));
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    size_t items = transmit_log(log);
    std::cout << "# " << items << std::endl;
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <thread>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(200));
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    size_t items = transmit_log(log);
    std::cout << "# " << items << std::endl;
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <thread>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(200));
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    size_t items = transmit_log(log);
    std::cout << "# " << items << std::endl;
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <thread>
#include <future>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(200));
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    auto res = std::async(std::launch::async, transmit_log, log);
    size_t items = res.get();
    std::cout << "# " << items << std::endl;
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <thread>
#include <future>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(200));
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    auto res = std::async(std::launch::async, transmit_log, log);
    size_t items = res.get();
    std::cout << "# " << items << std::endl;
}
```

```
$ g++-4.9 -std=c++1y -Wall -Wextra -pedantic -Werror -pthread foo.cpp
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <thread>
#include <future>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(200));
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    auto res = std::async(std::launch::async, transmit_log, log);
    size_t items = res.get();
    std::cout << "# " << items << std::endl;
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <thread>
#include <future>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(200));
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    auto res = std::async(std::launch::async, transmit_log, log);
    size_t items = res.get();
    std::cout << "# " << items << std::endl;
}
```

... and now we can do some stuff between calling transmit_log until we need the result from calling that function.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <thread>
#include <future>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(200));
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    auto res = std::async(std::launch::async, transmit_log, log);
    size_t items = res.get();
    std::cout << "# " << items << std::endl;
}
```

... and now we can do some stuff between calling transmit_log until we need the result from calling that function.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <thread>
#include <future>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(200));
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    auto res = std::async(std::launch::async, transmit_log, log);
    for (int i=0; i<5; i++) {
        std::this_thread::sleep_for(std::chrono::milliseconds(77));
        std::cout << "do something else..." << std::endl;
    }
    size_t items = res.get();
    std::cout << "# " << items << std::endl;
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <thread>
#include <future>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(200));
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    auto res = std::async(std::launch::async, transmit_log, log);
    for (int i=0; i<5; i++) {
        std::this_thread::sleep_for(std::chrono::milliseconds(123));
        std::cout << "do something else..." << std::endl;
    }
    size_t items = res.get();
    std::cout << "# " << items << std::endl;
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <thread>
#include <future>

static void transmit_item(int i)
{
    std::cout << i << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(200));
    // ...
}

static size_t transmit_log(const std::vector<int> & log)
{
    std::for_each(std::begin(log), std::end(log), transmit_item);
    return log.size();
}

int main()
{
    std::vector<int> log{20,24,37,42,23,45,37};
    auto res = std::async(std::launch::async, transmit_log, log);
    for (int i=0; i<5; i++) {
        std::this_thread::sleep_for(std::chrono::milliseconds(123));
        std::cout << "do something else..." << std::endl;
    }
    size_t items = res.get();
    std::cout << "# " << items << std::endl;
}
```

```
20
do something else...
24
do something else...
do something else...
37
do something else...
42
do something else...
23
45
37
# 7
```

Modern C++

- move semantics (rvalue references, value semantics)
- type deduction (decltype, auto)
- better support for OOP (attributes, member initialization, delegation)
- compile time computation (templates, static_assert, constexpr)
- template metaprogramming (traits, constraints, concepts)
- robust resource management (RAII, unique, shared)
- high-order parallelism (atomic, mutex, async, promises and futures)
- functional programming (algorithms, lamdas, closures, lazy evaluation)
- misc (chrono, user-defined literals, regex, uniform initialization)

!

History of C

- CPL (1963)
- BCPL (1966)
- B (1969)
- Unix (1969-1973)
- Early C (1972)
- K&R C (1978)
- X3J11 established (1983)
- ANSI X3.159-1989 (C89 / ANSI C)
- ISO/IEC 9899:1990 (C90, same as C89)
- WG14 first meeting in 1994
- ISO/IEC 9899/AMD1:1995 (C95, minor update)
- ISO/IEC 9899:1999 (C99)
- ISO/IEC 9899:2011 (C11, current version)

History of C++

- PhD, Simula, BCPL (Cambridge)
- C with Classes (Cpre, 1979)
- First external paper (1981)
- C++ named (1983)
- CFront 1.0 (1985)
- TC++PL, Ed1 (1985)
- ANSI X3J16 meeting (1989)
- The Annotated C++ Reference Manual (1990)
- First WG21 meeting (1991)
- The Design and Evolution of C++ (1994)
- ISO/IEC 14882:1998 (C++98)
- ISO/IEC 14882:2003 (C++03)
- ISO/IEC TR 19768:2007 (C++TR1)
- ISO/IEC 14882:2011 (C++11)
- soon ISO/IEC 14882:2014 (C++14)

c

C++ is not like this

C++