WIKIPEDIA

COBOL

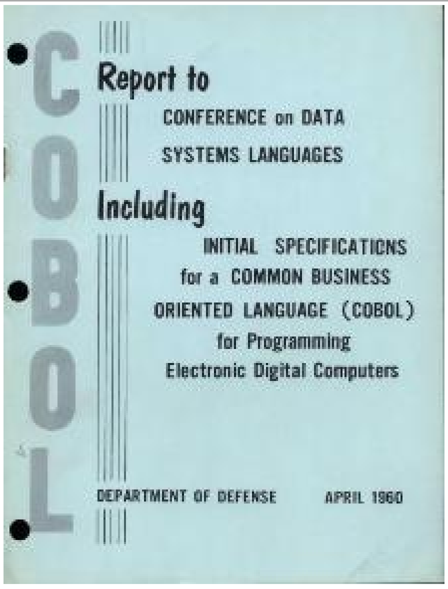
COBOL (/ˈkoʊbɒl, -bɔːl/; an acronym for "common business-oriented language") is a compiled English-like computer programming language designed for business use. It is an imperative, procedural and, since 2002, object-oriented language. COBOL is primarily used in business, finance, and administrative systems for companies and governments. COBOL is still widely used in applications deployed on mainframe computers, such as large-scale batch and transaction processing jobs. However, due to its declining popularity and the retirement of experienced COBOL programmers, programs are being migrated to new platforms, rewritten in modern languages or replaced with software packages.[8] Most programming in COBOL is now purely to maintain existing applications; however, many large financial institutions were still developing new systems in COBOL as late as 2006.[9]

COBOL was designed in 1959 by CODASYL and was partly based on the programming language FLOW-MATIC designed by Grace Hopper. It was created as part of a US Department of Defense effort to create a portable programming language for data processing. It was originally seen as a stopgap, but the Department of Defense promptly forced computer manufacturers to provide it, resulting in its widespread adoption.[10] It was standardized in 1968 and has since been revised four times. Expansions include support for structured and object-oriented programming. The current standard is *ISO/IEC 1989:2014*.[11]

COBOL statements have an English-like syntax, which was designed to be self-documenting and highly readable. However, it is verbose and uses over 300 reserved words. In contrast with modern, succinct syntax like `y = x;`, COBOL has a more English-like syntax (in this case, `MOVE x TO y`). COBOL code is split into four *divisions* (identification, environment, data, and procedure) containing a rigid hierarchy of sections, paragraphs and sentences. Lacking a large standard library, the standard specifies 43 statements, 87 functions and just one class.

Academic computer scientists were generally uninterested in business applications when COBOL was created and were not involved in its design; it was (effectively) designed from the ground up as a computer language for business, with an emphasis on inputs and outputs, whose only data types were numbers and strings of text.[12] COBOL has been criticized throughout its life for its verbosity, design process, and poor support for structured

COBOL



COBOL

Report to
CONFERENCE on DATA
SYSTEMS LANGUAGES
Including
INITIAL SPECIFICATIONS
for a COMMON BUSINESS
ORIENTED LANGUAGE (COBOL)
for Programming
Electronic Digital Computers

DEPARTMENT OF DEFENSE APRIL 1960

The cover of the *COBOL 60* report to CODASYL (April 1960)

Paradigm	Procedural, imperative, object-oriented
Designed by	Howard Bromberg, Norman Discount, Vernon Reeves, Jean E. Sammet, William Selden, Gertrude Tierney, with indirect influence from Grace Hopper[1]
Developers	CODASYL, ANSI, ISO
First appeared	1959
Stable release	ISO/IEC 1989:2014 / 2014
Typing discipline	Weak, static
Filename extensions	`.cbl`, `.cob`, `.cpy`

Major implementations

programming. These weaknesses result in monolithic, verbose (intended to be English-like) programs that are not easily comprehensible.

Contents

GnuCOBOL, IBM COBOL, Micro Focus Visual COBOL
Dialects
COBOL/2, DEC COBOL-10, DEC VAX COBOL, DOSVS COBOL, Envyr ICOBOL, Fujitsu COBOL, Hitachi COBOL2002, HP3000 COBOL/II, IBM COBOL SAA, IBM COBOL/400, IBM COBOL/II, IBM Enterprise COBOL, IBM ILE COBOL, IBM OS/VS COBOL, ICL COBOL (VME), Micro Focus ACUCOBOL-GT, Micro Focus COBOL-IT, Micro Focus RM/COBOL, Micro Focus Visual COBOL, Microsoft COBOL, Raincode COBOL, Realia COBOL, Ryan McFarland RM/COBOL, Ryan McFarland RM/COBOL-85, Tandem (NonStop) COBOL85, Tandem (NonStop) SCOBOL, UNIVAC COBOL, Unisys MCP COBOL74, Unisys MCP COBOL85, Unix COBOL X/Open, Veryant isCOBOL, Wang VS COBOL
Influenced by
Initial: AIMACO, COMTRAN, FACT, FLOW-MATIC COBOL 2002:[a] C++, Eiffel, Smalltalk
Influenced
CobolScript,[5] EGL,[6] PL/I,[7] PL/B
COBOL at Wikibooks

History and specification

Background

In the late 1950s, computer users and manufacturers were becoming concerned about the rising cost of programming. A 1959 survey had found that in any data processing installation, the programming cost US$800,000 on average and that translating programs to run on new hardware would cost $600,000. At a time when new programming languages were proliferating at an ever-increasing rate, the same survey suggested that if a common business-oriented language were used, conversion would be far cheaper and faster.[13]

On 8 April 1959, Mary K. Hawes, a computer scientist at Burroughs Corporation, called a meeting of representatives from academia, computer users, and manufacturers at the University of Pennsylvania to organize a formal meeting on common business languages.[14] Representatives included Grace Hopper (inventor of the English-like data processing language FLOW-MATIC), Jean Sammet and Saul Gorn.[15][16]

At the April meeting, the group asked the Department of Defense (DoD) to sponsor an effort to create a common business language. The delegation impressed Charles A. Phillips, director of the Data System Research Staff at the DoD,[17] who thought that they "thoroughly understood" the DoD's problems. The DoD operated 225 computers, had a further 175 on order and had spent over $200 million on implementing programs to run on them. Portable programs would save time, reduce costs and ease modernization.[18]

Charles Phillips agreed to sponsor the meeting and tasked the delegation with drafting the agenda.[19]

COBOL 60

On 28 and 29 May 1959 (exactly one year after the Zürich ALGOL 58 meeting), a meeting was held at the Pentagon to discuss the creation of a common programming language for business. It was attended by 41 people and was chaired by Phillips.[20] The Department of Defense was concerned about whether it could run the same data processing programs on different computers. FORTRAN, the only mainstream language at the time, lacked the features needed to write such programs.[21]

Representatives enthusiastically described a language that could work in a wide variety of environments, from banking and insurance to utilities and inventory control. They agreed unanimously that more people should be able to program and that the new language should not be restricted by the limitations of contemporary technology. A majority agreed that the language should make maximal use of English, be capable of change, be machine-independent and be easy to use, even at the expense of power.[22]

The meeting resulted in the creation of a steering committee and short, intermediate and long-range committees. The short-range committee was given to September (three months) to produce specifications for an interim language, which would then be improved upon by the other committees.[23][24] Their official mission, however, was to identify the strengths and weaknesses of existing programming languages and did not explicitly direct them to create a new language.[21] The deadline was met with disbelief by the short-range committee.[25] One member, Betty Holberton, described the three-month deadline as "gross optimism" and doubted that the language really would be a stopgap.[26]

The steering committee met on 4 June and agreed to name the entire activity as the *Committee on Data Systems Languages*, or CODASYL, and to form an executive committee.[27]

The short-range committee was made up of members representing six computer manufacturers and three government agencies. The six computer manufacturers were Burroughs Corporation, IBM, Minneapolis-Honeywell (Honeywell Labs), RCA, Sperry Rand, and Sylvania Electric Products. The three government agencies were the US Air Force, the Navy's David Taylor Model Basin, and the National Bureau of Standards (now the National Institute of Standards and Technology).[28] The committee was chaired by Joseph Wegstein of the US National Bureau of Standards. Work began by investigating data description, statements, existing applications and user experiences.[29]

The committee mainly examined the FLOW-MATIC, AIMACO and COMTRAN programming languages.[21][30] The FLOW-MATIC language was particularly influential because it had been implemented and because AIMACO was a derivative of it with only minor changes.[31][32] FLOW-MATIC's inventor, Grace Hopper, also served as a technical adviser to the committee.[25] FLOW-MATIC's major contributions to COBOL were long variable names, English words for commands and the separation of data descriptions and instructions.[33] Hopper is sometimes referred to as "the mother of COBOL" or "the grandmother of COBOL",[34][35][36] although Jean Sammet, a lead designer of COBOL, stated that Hopper "was not the mother, creator or developer of Cobol".[37][1]

IBM's COMTRAN language, invented by Bob Bemer, was regarded as a competitor to FLOW-MATIC[38][39] by a short-range committee made up of colleagues of Grace Hopper.[40] Some of its features were not incorporated into COBOL so that it would not look like IBM had dominated the design process,[23] and Jean Sammet said in 1981 that there had been a "strong anti-IBM bias" from some committee members (herself included).[41] In one case, after Roy Goldfinger, author of the COMTRAN manual and intermediate-range committee member, attended a subcommittee meeting to support his language and encourage the use of algebraic expressions, Grace Hopper sent a memo to the short-range committee reiterating Sperry Rand's efforts to create a language based on English.[42] In 1980, Grace Hopper commented that "COBOL 60 is 95% FLOW-MATIC" and that COMTRAN had had an "extremely small" influence. Furthermore, she said that she would claim that work was influenced by both FLOW-MATIC and COMTRAN only to "keep other people happy [so they] wouldn't try to knock us out".[43] Features from COMTRAN incorporated into COBOL included formulas,[44] the `PICTURE` clause,[45] an improved `IF` statement, which obviated the need for GO TOs, and a more robust file management system.[38]

The usefulness of the committee's work was subject of great debate. While some members thought the language had too many compromises and was the result of design by committee, others felt it was better than the three languages examined. Some felt the language was too complex; others, too simple.[46] Controversial features included those some considered useless or too advanced for data processing users. Such features included boolean expressions, formulas and table *subscripts* (indices).[47][48] Another point of controversy was whether to make keywords context-sensitive and the effect that would have on readability.[47] Although context-sensitive keywords were rejected, the approach was later used in PL/I and partially in COBOL from 2002.[49] Little consideration was given to interactivity, interaction with operating systems (few existed at that time) and functions (thought of as purely mathematical and of no use in data processing).[50][51]

The specifications were presented to the Executive Committee on 4 September. They fell short of expectations: Joseph Wegstein noted that "it contains rough spots and requires some additions", and Bob Bemer later described them as a "hodgepodge". The subcommittee was given until December to improve it.[25]

At a mid-September meeting, the committee discussed the new language's name. Suggestions included "BUSY" (Business System), "INFOSYL" (Information System Language) and "COCOSYL" (Common Computer Systems Language).[52] It is unclear who coined the name "COBOL",[53][54] although Bob Bemer later claimed it had been his suggestion.[55][56][57]

In October, the intermediate-range committee received copies of the FACT language specification created by Roy Nutt. Its features impressed the committee so much that they passed a resolution to base COBOL on it.[58] This was a blow to the short-range committee, who had made good progress on the specification. Despite being technically superior, FACT had not been created with portability in mind or through manufacturer and user consensus. It also lacked a demonstrable implementation,[25] allowing supporters of a FLOW-MATIC-based COBOL to overturn the resolution. RCA representative Howard Bromberg also blocked FACT, so that RCA's work on a COBOL implementation would not go to waste.[59]

'And what name do you want inscribed?'
I said, 'I'll write it for you.' I wrote the name down: COBOL.
'What kind of name is that?'
'Well it's a Polish name. We shortened it and got rid of a lot of unnecessary notation.'

Howard Bromberg on how he bought the COBOL tombstone[60]

It soon became apparent that the committee was too large for any further progress to be made quickly. A frustrated Howard Bromberg bought a $15 tombstone with "COBOL" engraved on it and sent it to Charles Phillips to demonstrate his displeasure.[b][60][62] A sub-committee was formed to analyze existing languages and was made up of six individuals:[21][63]

- William Selden and Gertrude Tierney of IBM,
- Howard Bromberg and Howard Discount of RCA,
- Vernon Reeves and Jean E. Sammet of Sylvania Electric Products.

The sub-committee did most of the work creating the specification, leaving the short-range committee to review and modify their work before producing the finished specification.[21]

The specifications were approved by the Executive Committee on 8 January 1960, and sent to the government printing office, which printed these as *COBOL 60*. The language's stated objectives were to allow efficient, portable programs to be easily written, to allow users to move to new systems with minimal effort and cost, and to be suitable for inexperienced programmers.[64] The CODASYL Executive Committee later created the COBOL Maintenance Committee to answer questions from users and vendors and to improve and expand the specifications.[65]

During 1960, the list of manufacturers planning to build COBOL compilers grew. By September, five more manufacturers had joined CODASYL (Bendix, Control Data Corporation, General Electric (GE), National Cash Register and Philco), and all represented manufacturers had announced COBOL compilers. GE and IBM planned to integrate COBOL into their own languages, GECOM and COMTRAN, respectively. In contrast, International Computers and Tabulators planned to replace their language, CODEL, with COBOL.[66]

Meanwhile, RCA and Sperry Rand worked on creating COBOL compilers. The first COBOL program ran on 17 August on an RCA 501.[67] On 6 and 7 December, the same COBOL program (albeit with minor changes) ran on an RCA computer and a Remington-Rand Univac computer, demonstrating that compatibility could be achieved.[68]

The relative influences of which languages were used continues to this day in the recommended advisory printed in all COBOL reference manuals:

> COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.
>
> No warranty, expressed or implied, is made by any contributor or by the CODASYL COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith. The authors and copyright holders of the copyrighted material used herein are as follows:
>
> > FLOW-MATIC (trademark of Unisys Corporation), Programming for the UNIVAC (R) I and II, Data Automation Systems, copyrighted 1958, 1959, by Unisys Corporation; IBM Commercial Translator Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.
>
> They have specifically authorized the use of this material, in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.[69]

COBOL-61 to COBOL-65

> It is rather unlikely that Cobol will be around by the end of the decade.
>
> Anonymous, June 1960[70]

Many logical flaws were found in *COBOL 60*, leading GE's Charles Katz to warn that it could not be interpreted unambiguously. A reluctant short-term committee enacted a total cleanup and, by March 1963, it was reported that COBOL's syntax was as definable as ALGOL's, although semantic ambiguities remained.[66]

Early COBOL compilers were primitive and slow. A 1962 US Navy evaluation found compilation speeds of 3–11 statements per minute. By mid-1964, they had increased to 11–1000 statements per minute. It was observed that increasing memory would drastically increase speed and that compilation costs varied wildly: costs per statement were between $0.23 and $18.91.[71]

In late 1962, IBM announced that COBOL would be their primary development language and that development of COMTRAN would cease.[71]

The COBOL specification was revised three times in the five years after its publication. COBOL-60 was replaced in 1961 by COBOL-61. This was then replaced by the COBOL-61 Extended specifications in 1963, which introduced the sort and report writer facilities.[72] The added facilities corrected flaws identified by Honeywell in late 1959 in a letter to the short-range committee.[67] COBOL Edition 1965 brought further clarifications to the specifications and introduced facilities for handling mass storage files and tables.[73]

COBOL-68

Efforts began to standardize COBOL to overcome incompatibilities between versions. In late 1962, both ISO and the United States of America Standards Institute (now ANSI) formed groups to create standards. ANSI produced *USA Standard COBOL X3.23* in August 1968, which became the cornerstone for later versions.[74] This version was known as American National Standard (ANS) COBOL and was adopted by ISO in 1972.[75]

COBOL-74

By 1970, COBOL had become the most widely used programming language in the world.[76]

Independently of the ANSI committee, the CODASYL Programming Language Committee was working on improving the language. They described new versions in 1968, 1969, 1970 and 1973, including changes such as new inter-program communication, debugging and file merging facilities as well as improved string-handling and library inclusion features.[77] Although CODASYL was independent of the ANSI committee, the *CODASYL Journal of Development* was used by ANSI to identify features that were popular enough to warrant implementing.[78] The Programming Language Committee also liaised with ECMA and the Japanese COBOL Standard committee.[77]

The Programming Language Committee was not well-known, however. The vice-president, William Rinehuls, complained that two-thirds of the COBOL community did not know of the committee's existence. It was also poor, lacking the funds to make public documents, such as minutes of meetings and change proposals, freely available.[79]

In 1974, ANSI published a revised version of (ANS) COBOL, containing new features such as file organizations, the `DELETE` statement[80] and the segmentation module.[81] Deleted features included the `NOTE` statement, the `EXAMINE` statement (which was replaced by `INSPECT`) and the implementer-defined random access module (which was superseded by the new sequential and relative I/O modules). These made up 44 changes, which rendered existing statements incompatible with the new standard.[82] The report writer was slated to be removed from COBOL, but was reinstated before the standard was published.[83][84] ISO later adopted the updated standard in 1978.[75]

COBOL-85

In June 1978, work began on revising COBOL-74. The proposed standard (commonly called COBOL-80) differed significantly from the previous one, causing concerns about incompatibility and conversion costs. In January 1981, Joseph T. Brophy, Senior Vice-President of Travelers Insurance, threatened to sue the standard committee because it was not upwards compatible with COBOL-74. Mr. Brophy described previous conversions of their 40-million-line code base as "non-productive" and a "complete waste of our programmer resources".[85] Later that year, the Data Processing Management Association (DPMA) said it was "strongly opposed" to the new standard, citing "prohibitive" conversion costs and enhancements that were "forced on the user".[86][87]

During the first public review period, the committee received 2,200 responses, of which 1,700 were negative form letters.[88] Other responses were detailed analyses of the effect COBOL-80 would have on their systems; conversion costs were predicted to be at least 50 cents per line of code. Fewer than a dozen of the responses were in favor of the proposed standard.[89]

ISO TC97-SC5 installed in 1979 the international COBOL Experts Group, on initiative of Wim Ebbinkhuijsen. The group consisted of COBOL experts from many countries, including the United States. Its goal was to achieve mutual understanding and respect between ANSI and the rest of the world with regard to the need of new COBOL features. After three years, ISO changed the status of the group to a formal Working Group: WG 4 COBOL. The group took primary ownership and development of the COBOL standard, where ANSI did most of the proposals.

In 1983, the DPMA withdrew its opposition to the standard, citing the responsiveness of the committee to public concerns. In the same year, a National Bureau of Standards study concluded that the proposed standard would present few problems.[87][90] A year later, DEC released a VAX/VMS COBOL-80, who noted that conversion of COBOL-74 programs posed few problems. The new `EVALUATE` statement and inline `PERFORM` were particularly well received and improved productivity, thanks to simplified control flow and debugging.[91]

The second public review drew another 1,000 (mainly negative) responses, while the last drew just 25, by which time many concerns had been addressed.[87]

In 1985, the ISO Working Group 4 accepted the then-version of the ANSI proposed standard, made several changes and set it as the new ISO standard COBOL 85. It was published in late 1985.

Sixty features were changed or deprecated and many were added, such as:[92][93]

- Scope terminators (`END-IF`, `END-PERFORM`, `END-READ`, etc.)
- Nested subprograms
- `CONTINUE`, a no-operation statement
- `EVALUATE`, a switch statement
- `INITIALIZE`, a statement that can set groups of data to their default values
- Inline `PERFORM` loop bodies – previously, loop bodies had to be specified in a separate procedure
- Reference modification, which allows access to substrings
- I/O status codes.

The new standard was adopted by all national standard bodies, including ANSI.[75]

Two amendments followed in 1989 and 1993, the first introducing intrinsic functions and the other providing corrections.[75]

COBOL 2002 and object-oriented COBOL

In 1997, Gartner Group estimated that there were a total of 200 billion lines of COBOL in existence, which ran 80% of all business programs.[94]

In the early 1990s, work began on adding object-orientation in the next full revision of COBOL. Object-oriented features were taken from C++ and Smalltalk.[2][3] The initial estimate was to have this revision completed by 1997, and an ISO Committee Draft (CD) was available by 1997. Some vendors (including Micro Focus, Fujitsu, and IBM) introduced object-oriented syntax based on drafts of the full revision. The final approved ISO standard was approved and published in late 2002.[95]

Fujitsu/GTSoftware,[96] Micro Focus and RainCode introduced object-oriented COBOL compilers targeting the .NET Framework.

There were many other new features, many of which had been in the *CODASYL COBOL Journal of Development* since 1978 and had missed the opportunity to be included in COBOL-85.[97] These other features included:[98][99]

- Free-form code
- User-defined functions
- Recursion
- Locale-based processing
- Support for extended character sets such as Unicode
- Floating-point and binary data types (until then, binary items were truncated based on their declaration's base-10 specification)
- Portable arithmetic results
- Bit and boolean data types
- Pointers and syntax for getting and freeing storage
- The `SCREEN SECTION` for text-based user interfaces
- The `VALIDATE` facility
- Improved interoperability with other programming languages and framework environments such as .NET and Java.

Three corrigenda were published for the standard: two in 2006 and one in 2009.[100]

COBOL 2014

Between 2003 and 2009, three technical reports were produced describing object finalization, XML processing and collection classes for COBOL.[100]

COBOL 2002 suffered from poor support: no compilers completely supported the standard. Micro Focus found that it was due to a lack of user demand for the new features and due to the abolition of the NIST test suite, which had been used to test compiler conformance. The standardization process was also found to be slow and under-resourced.[101]

COBOL 2014 includes the following changes:[102]

- Portable arithmetic results have been replaced by IEEE 754 data types
- Major features have been made optional, such as the `VALIDATE` facility, the report writer and the screen-handling facility
- Method overloading
- Dynamic capacity tables (a feature dropped from the draft of COBOL 2002)[103]

Legacy

COBOL programs are used globally in governments and businesses and are running on diverse operating systems such as z/OS, z/VSE, VME, Unix, OpenVMS and Windows. In 1997, the Gartner Group reported that 80% of the world's business ran on COBOL with over 200 billion lines of code and 5 billion lines more being written annually.[104]

Near the end of the 20th century, the year 2000 problem (Y2K) was the focus of significant COBOL programming effort, sometimes by the same programmers who had designed the systems decades before. The particular level of effort required to correct COBOL code has been attributed to the large amount of business-oriented COBOL, as business applications use dates heavily, and to fixed-length data fields. After the clean-up effort put into these programs for Y2K, a 2003 survey found that many remained in use.[105] The authors said that the survey data suggest "a gradual decline in the importance of Cobol in application development over the [following] 10 years unless ... integration with other languages and technologies can be adopted".[106]

In 2006 and 2012, *Computerworld* surveys found that over 60% of organizations used COBOL (more than C++ and Visual Basic .NET) and that for half of those, COBOL was used for the majority of their internal software.[9][107] 36% of managers said they planned to migrate from COBOL, and 25% said they would like to if it was cheaper. Instead, some businesses have migrated their systems from expensive mainframes to cheaper, more modern systems, while maintaining their COBOL programs.[9]

Testimony before the House of Representatives in 2016 indicated that COBOL is still in use by many federal agencies.[108] Reuters reported in 2017 that 43% of banking systems still used COBOL with over 220 billion lines of COBOL code in use.[109]

By 2019, the number of COBOL programmers was shrinking fast due to retirements, leading to an impending skills gap in business and government organizations which still use mainframe systems for high-volume transaction processing. Efforts to rewrite systems in newer languages have proven expensive and problematic, as has the outsourcing of code maintenance, thus proposals to train more people in COBOL are advocated.[110]

During the COVID-19 pandemic and the ensuing surge of unemployment, several US states reported a shortage of skilled COBOL programmers to support the legacy systems used for unemployment benefit management. Many of these systems had been in the process of conversion to more modern programming languages prior to the pandemic, but the process had to be put on hold.[111] Similarly, the US Internal Revenue Service rushed to patch its COBOL-based Individual Master File in order to disburse the tens of millions of payments mandated by the Coronavirus Aid, Relief, and Economic Security Act.[112]

Features

Syntax

COBOL has an English-like syntax, which is used to describe nearly everything in a program. For example, a condition can be expressed as `x IS GREATER THAN y` or more concisely as `x GREATER y` or `x > y`. More complex conditions can be "abbreviated" by removing repeated conditions and variables. For example, `a > b AND a > c OR a = d` can be shortened to `a > b AND c OR = d`. To support this English-like syntax, COBOL has over 300 keywords.[113][c] Some of the keywords are simple alternative or pluralized spellings of the same word, which provides for more English-like statements and clauses; e.g., the `IN` and `OF` keywords can be used interchangeably, as can `IS` and `ARE`, and `VALUE` and `VALUES`.

Each COBOL program is made up of four basic lexical items: words, literals, picture character-strings (see § PICTURE clause) and separators. Words include reserved words and user-defined identifiers. They are up to 31 characters long and may include letters, digits, hyphens and underscores. Literals include numerals (e.g. `12`) and strings (e.g. `'Hello!'`).[115] Separators include the space character and commas and semi-colons followed by a space.[116]

A COBOL program is split into four divisions: the identification division, the environment division, the data division and the procedure division. The identification division specifies the name and type of the source element and is where classes and interfaces are specified. The environment division specifies any program features that depend on the system running it, such as files and character sets. The data division is used to declare variables and parameters. The procedure division contains the program's statements. Each division is sub-divided into sections, which are made up of paragraphs.

Metalanguage

COBOL's syntax is usually described with a unique metalanguage using braces, brackets, bars and underlining. The metalanguage was developed for the original COBOL specifications. Although Backus–Naur form did exist at the time, the committee had not heard of it.[117]

Elements of COBOL's metalanguage

Element	Appearance	Function
All capitals	EXAMPLE	Reserved word
Underlining	<u>EXAMPLE</u>	The reserved word is compulsory
Braces	{ }	Only one option may be selected
Brackets	[]	Zero or one options may be selected
Ellipsis	...	The preceding element may be repeated
Bars	{| |}	One or more options may be selected. Any option may only be selected once.
	[| |]	Zero or more options may be selected. Any option may only be selected once.

As an example, consider the following description of an `ADD` statement:

$$
\begin{array}{l}
\underline{\text{ADD}} \begin{Bmatrix} \text{identifier-1} \\ \text{literal-1} \end{Bmatrix} \dots \ \underline{\text{TO}} \ \{\text{identifier-2} \ [\,\underline{\text{ROUNDED}}\,]\} \dots \\
\quad \left[\left| \begin{array}{l} \text{ON } \underline{\text{SIZE}} \ \underline{\text{ERROR}} \text{ imperative-statement-1} \\ \underline{\text{NOT}} \text{ ON } \underline{\text{SIZE}} \ \underline{\text{ERROR}} \text{ imperative-statement-2} \end{array} \right|\right] \\
\quad [\,\underline{\text{END-ADD}}\,]
\end{array}
$$

This description permits the following variants:

```
ADD 1 TO x
ADD 1, a, b TO x ROUNDED, y, z ROUNDED

ADD a, b TO c
    ON SIZE ERROR
        DISPLAY "Error"
END-ADD

ADD a TO b
    NOT SIZE ERROR
        DISPLAY "No error"
    ON SIZE ERROR
        DISPLAY "Error"
```

Code format

COBOL can be written in two formats: fixed (the default) or free. In fixed-format, code must be aligned to fit in certain areas (a hold-over from using punched cards). Until COBOL 2002, these were:

Name	Column(s)	Usage
Sequence number area	1–6	Originally used for card/line numbers (facilitating mechanical punched card sorting to assure intended program code sequence after manual editing/handling), this area is ignored by the compiler
Indicator area	7	The following characters are allowed here: ▪ `*` – Comment line ▪ `/` – Comment line that will be printed on a new page of a source listing ▪ `-` – Continuation line, where words or literals from the previous line are continued ▪ `D` – Line enabled in debugging mode, which is otherwise ignored
Area A	8–11	This contains: `DIVISION`, `SECTION` and procedure headers; 01 and 77 level numbers and file/report descriptors
Area B	12–72	Any other code not allowed in Area A
Program name area	73–	Historically up to column 80 for punched cards, it is used to identify the program or sequence the card belongs to

In COBOL 2002, Areas A and B were merged to form the program-text area, which now ends at an implementor-defined column.[118]

COBOL 2002 also introduced free-format code. Free-format code can be placed in any column of the file, as in newer programming languages. Comments are specified using `*>`, which can be placed anywhere and can also be used in fixed-format source code. Continuation lines are not present, and the `>>PAGE` directive replaces the `/` indicator.[118]

Identification division

The identification division identifies the following code entity and contains the definition of a class or interface.

Object-oriented programming

Classes and interfaces have been in COBOL since 2002. Classes have factory objects, containing class methods and variables, and instance objects, containing instance methods and variables.[119] Inheritance and interfaces provide polymorphism. Support for generic programming is provided through parameterized classes, which can be instantiated to use any class or interface. Objects are stored as references which may be restricted to a certain type. There are two ways of calling a method: the `INVOKE` statement, which acts similarly to `CALL`, or through inline method invocation, which is analogous to using functions.[120]

```
*> These are equivalent.
INVOKE my-class "foo" RETURNING var
```

```
MOVE my-class::"foo" TO var *> Inline method invocation
```

COBOL does not provide a way to hide methods. Class data can be hidden, however, by declaring it without a `PROPERTY` clause, which leaves the user with no way to access it.[121] Method overloading was added in COBOL 2014.[122]

Environment division

The environment division contains the configuration section and the input-output section. The configuration section is used to specify variable features such as currency signs, locales and character sets. The input-output section contains file-related information.

Files

COBOL supports three file formats, or *organizations*: sequential, indexed and relative. In sequential files, records are contiguous and must be traversed sequentially, similarly to a linked list. Indexed files have one or more indexes which allow records to be randomly accessed and which can be sorted on them. Each record must have a unique key, but other, *alternate*, record keys need not be unique. Implementations of indexed files vary between vendors, although common implementations, such as C-ISAM and VSAM, are based on IBM's ISAM. Relative files, like indexed files, have a unique record key, but they do not have alternate keys. A relative record's key is its ordinal position; for example, the 10th record has a key of 10. This means that creating a record with a key of 5 may require the creation of (empty) preceding records. Relative files also allow for both sequential and random access.[123]

A common non-standard extension is the *line sequential* organization, used to process text files. Records in a file are terminated by a newline and may be of varying length.[124]

Data division

The data division is split into six sections which declare different items: the file section, for file records; the working-storage section, for static variables; the local-storage section, for automatic variables; the linkage section, for parameters and the return value; the report section and the screen section, for text-based user interfaces.

Aggregated data

Data items in COBOL are declared hierarchically through the use of level-numbers which indicate if a data item is part of another. An item with a higher level-number is subordinate to an item with a lower one. Top-level data items, with a level-number of 1, are called *records*. Items that have subordinate aggregate data are called *group items*; those that do not are called *elementary items*. Level-numbers used to describe standard data items are between 1 and 49.[125][126]

```
       01  some-record.                   *> Aggregate group record item
           05  num            PIC 9(10).  *> Elementary item
           05  the-date.                  *> Aggregate (sub)group record item
               10  the-year   PIC 9(4).   *> Elementary item
               10  the-month  PIC 99.     *> Elementary item
               10  the-day    PIC 99.     *> Elementary item
```

In the above example, elementary item `num` and group item `the-date` are subordinate to the record `some-record`, while elementary items `the-year`, `the-month`, and `the-day` are part of the group item `the-date`.

Subordinate items can be disambiguated with the `IN` (or `OF`) keyword. For example, consider the example code above along with the following example:

```
       01  sale-date.
           05  the-year       PIC 9(4).
           05  the-month      PIC 99.
           05  the-day        PIC 99.
```

The names `the-year`, `the-month`, and `the-day` are ambiguous by themselves, since more than one data item is defined with those names. To specify a particular data item, for instance one of the items contained within the `sale-date` group, the programmer would use `the-year IN sale-date` (or the equivalent `the-year OF sale-date`). (This syntax is similar to the "dot notation" supported by most contemporary languages.)

Other data levels

A level-number of 66 is used to declare a re-grouping of previously defined items, irrespective of how those items are structured. This data level, also referred to by the associated `RENAMES` clause, is rarely used[127] and, circa 1988, was usually found in old programs. Its ability to ignore the hierarchical and logical structure data meant its use was not recommended and many installations forbade its use.[128]

```
       01  customer-record.
           05  cust-key            PIC X(10).
           05  cust-name.
               10  cust-first-name PIC X(30).
               10  cust-last-name  PIC X(30).
           05  cust-dob            PIC 9(8).
           05  cust-balance        PIC 9(7)V99.

       66  cust-personal-details   RENAMES cust-name THRU cust-dob.
       66  cust-all-details        RENAMES cust-name THRU cust-balance.
```

A 77 level-number indicates the item is stand-alone, and in such situations is equivalent to the level-number 01. For example, the following code declares two 77-level data items, `property-name` and `sales-region`, which are non-group data items that are independent of (not subordinate to) any other data items:

```
       77  property-name      PIC X(80).
       77  sales-region       PIC 9(5).
```

An 88 level-number declares a *condition name* (a so-called 88-level) which is true when its parent data item contains one of the values specified in its `VALUE` clause.[129] For example, the following code defines two 88-level condition-name items that are true or false depending on the current character data value of the `wage-type` data item. When the data item contains a value of `'H'`, the condition-name `wage-is-hourly` is true, whereas when it contains a value of `'S'` or `'Y'`, the condition-name `wage-is-yearly` is true. If the data item contains some other value, both of the condition-names are false.

```
       01  wage-type          PIC X.
           88  wage-is-hourly VALUE "H".
           88  wage-is-yearly VALUE "S", "Y".
```

Data types

Standard COBOL provides the following data types:[130]

Data type	Sample declaration	Notes
Alphabetic	`PIC A(30)`	May only contain letters or spaces
Alphanumeric	`PIC X(30)`	May contain any characters
Boolean	`PIC 1 USAGE BIT`	Data stored in the form of 0s and 1s, as a binary number
Index	`USAGE INDEX`	Used to reference table elements
National	`PIC N(30)`	Similar to alphanumeric, but using an extended character set, e.g. UTF-8
Numeric	`PIC 9(5)V9(5)`	May contain only numbers
Object	`USAGE OBJECT REFERENCE`	May reference either an object or `NULL`
Pointer	`USAGE POINTER`	

Type safety is variable in COBOL. Numeric data is converted between different representations and sizes silently and alphanumeric data can be placed in any data item that can be stored as a string, including numeric and group data.[131] In contrast, object references and pointers may only be assigned from items of the same type and their values may be restricted to a certain type.[132]

PICTURE clause

A `PICTURE` (or `PIC`) clause is a string of characters, each of which represents a portion of the data item and what it may contain. Some picture characters specify the type of the item and how many characters or digits it occupies in memory. For example, a `9` indicates a decimal digit, and an `S` indicates that the item is signed. Other picture characters (called *insertion* and *editing* characters) specify how an item should be formatted. For example, a series of `+` characters define character positions as well as how a leading sign character is to be positioned within the final character data; the rightmost non-numeric character will contain the item's sign, while other character positions corresponding to a `+` to the left of this position will contain a space. Repeated characters can be specified more concisely by specifying a number in parentheses after a picture character; for example, `9(7)` is equivalent to `9999999`. Picture specifications containing only digit (`9`) and sign (`S`) characters define purely *numeric* data items, while picture specifications containing alphabetic (`A`) or alphanumeric (`X`) characters define *alphanumeric* data items. The presence of other formatting characters define *edited numeric* or *edited alphanumeric* data items.[133]

Examples

`PICTURE` clause	Value in	Value out
`PIC 9(5)`	`100`	`00100`
	`"Hello"`	`"Hello"` (this is legal, but results in undefined behavior)[131]
`PIC +++++`	`-10`	`"  -10"` (note leading spaces)
`PIC 99/99/9(4)`	`31042003`	`"31/04/2003"`
`PIC *(4)9.99`	`100.50`	`"**100.50"`
	`0`	`"****0.00"`
`PIC X(3)BX(3)BX(3)`	`"ABCDEFGHI"`	`"ABC DEF GHI"`

USAGE clause

The `USAGE` clause declares the format data is stored in. Depending on the data type, it can either complement or be used instead of a `PICTURE` clause. While it can be used to declare pointers and object references, it is mostly geared towards specifying numeric types. These numeric formats are:[134]

- Binary, where a minimum size is either specified by the `PICTURE` clause or by a `USAGE` clause such as `BINARY-LONG`.
- `USAGE COMPUTATIONAL`, where data may be stored in whatever format the implementation provides; often equivalent to `USAGE BINARY`
- `USAGE DISPLAY`, the default format, where data is stored as a string
- Floating-point, in either an implementation-dependent format or according to IEEE 754.
- `USAGE NATIONAL`, where data is stored as a string using an extended character set
- `USAGE PACKED-DECIMAL`, where data is stored in the smallest possible decimal format (typically packed binary-coded decimal)

Report writer

The report writer is a declarative facility for creating reports. The programmer need only specify the report layout and the data required to produce it, freeing them from having to write code to handle things like page breaks, data formatting, and headings and footings.[135]

Reports are associated with report files, which are files which may only be written to through report writer statements.

```
       FD  report-out REPORT sales-report.
```

Each report is defined in the report section of the data division. A report is split into report groups which define the report's headings, footings and details. Reports work around hierarchical *control breaks*. Control breaks occur when a key variable changes it value; for example, when creating a report detailing customers' orders, a control break could occur when the program reaches a different customer's orders. Here is an example report description for a report which gives a salesperson's sales and which warns of any invalid records:

```
       RD  sales-report
           PAGE LIMITS 60 LINES
```

```
            FIRST DETAIL 3
            CONTROLS seller-name.

        01  TYPE PAGE HEADING.
            03  COL 1                    VALUE "Sales Report".
            03  COL 74                   VALUE "Page".
            03  COL 79                   PIC Z9 SOURCE PAGE-COUNTER.

        01  sales-on-day TYPE DETAIL, LINE + 1.
            03  COL 3                    VALUE "Sales on".
            03  COL 12                   PIC 99/99/9999 SOURCE sales-date.
            03  COL 21                   VALUE "were".
            03  COL 26                   PIC $$$$9.99 SOURCE sales-amount.

        01  invalid-sales TYPE DETAIL, LINE + 1.
            03  COL 3                    VALUE "INVALID RECORD:".
            03  COL 19                   PIC X(34) SOURCE sales-record.

        01  TYPE CONTROL HEADING seller-name, LINE + 2.
            03  COL 1                    VALUE "Seller:".
            03  COL 9                    PIC X(30) SOURCE seller-name.
```

The above report description describes the following layout:

```
Sales Report                                                          Page  1

Seller: Howard Bromberg
  Sales on 10/12/2008 were $1000.00
  Sales on 12/12/2008 were    $0.00
  Sales on 13/12/2008 were   $31.47
  INVALID RECORD: Howard Bromberg             XXXXYY

Seller: Howard Discount
...
Sales Report                                                         Page 12

  Sales on 08/05/2014 were  $543.98
  INVALID RECORD: William Selden      12O52014FOOFOO
  Sales on 30/05/2014 were    $0.00
```

Four statements control the report writer: `INITIATE`, which prepares the report writer for printing; `GENERATE`, which prints a report group; `SUPPRESS`, which suppresses the printing of a report group; and `TERMINATE`, which terminates report processing. For the above sales report example, the procedure division might look like this:

```
            OPEN INPUT sales, OUTPUT report-out
            INITIATE sales-report

            PERFORM UNTIL 1 <> 1
                READ sales
                    AT END
                        EXIT PERFORM
                END-READ

                VALIDATE sales-record
                IF valid-record
                    GENERATE sales-on-day
                ELSE
                    GENERATE invalid-sales
                END-IF
            END-PERFORM

            TERMINATE sales-report
            CLOSE sales, report-out
            .
```

Use of the Report Writer facility tended to vary considerably; some organizations used it extensively and some not at all.[136] In addition, implementations of Report Writer ranged in quality, with those at the lower end sometimes using excessive amounts of memory at runtime.[136]

Procedure division

Procedures

The sections and paragraphs in the procedure division (collectively called procedures) can be used as labels and as simple subroutines. Unlike in other divisions, paragraphs do not need to be in sections.[137] Execution goes down through the procedures of a program until it is terminated.[138] To use procedures as subroutines, the `PERFORM` verb is used.

A `PERFORM` statement somewhat resembles a procedure call in a modern language in the sense that execution returns to the code following the `PERFORM` statement at the end of the called code; however, it does not provide any mechanism for parameter passing or for returning a result value. If a subroutine is invoked using a simple statement like `PERFORM subroutine`, then control returns at the end of the called procedure. However, `PERFORM` is unusual in that it may be used to call a range spanning a sequence of several adjacent procedures. This is done with the `PERFORM sub-1 THRU sub-n` construct:

```
PROCEDURE so-and-so.
    PERFORM ALPHA
    PERFORM ALPHA THRU GAMMA
    STOP RUN.
ALPHA.
    DISPLAY 'A'.
BETA.
    DISPLAY 'B'.
GAMMA.
    DISPLAY 'C'.
```

The output of this program will be: "A A B C".

`PERFORM` also differs from conventional procedure calls in that there is, at least traditionally, no notion of a call stack. As a consequence, nested invocations are possible (a sequence of code being `PERFORM`'ed may execute a `PERFORM` statement itself), but require extra care if parts of the same code are executed by both invocations. The problem arises when the code in the inner invocation reaches the exit point of the outer invocation. More formally, if control passes through the exit point of a `PERFORM` invocation that was called earlier but has not completed yet, the COBOL 2002 standard officially stipulates that the behaviour is undefined.

The reason is that COBOL, rather than a "return address", operates with what may be called a continuation address. When control flow reaches the end of any procedure, the continuation address is looked up and control is transferred to that address. Before the program runs, the continuation address for every procedure is initialised to the start address of the procedure that comes next in the program text so that, if no `PERFORM` statements happen, control flows from top to bottom through the program. But when a `PERFORM` statement executes, it modifies the continuation address of the called procedure (or the last procedure of the called range, if `PERFORM THRU` was used), so that control will return to the call site at the end. The original value is saved and is restored afterwards, but there is only one storage position. If two nested invocations operate on overlapping code, they may interfere which each other's management of the continuation address in several ways.[139][140]

The following example (taken from Veerman & Verhoeven 2006) illustrates the problem:

```
LABEL1.
    DISPLAY '1'
    PERFORM LABEL2 THRU LABEL3
    STOP RUN.
LABEL2.
    DISPLAY '2'
    PERFORM LABEL3 THRU LABEL4.
LABEL3.
    DISPLAY '3'.
LABEL4.
    DISPLAY '4'.
```

One might expect that the output of this program would be "1 2 3 4 3": After displaying "2", the second `PERFORM` causes "3" and "4" to be displayed, and then the first invocation continues on with "3". In traditional COBOL implementations, this is not the case. Rather, the first `PERFORM` statement sets the continuation address at the end of `LABEL3` so that it will jump back to the call site inside `LABEL1`. The second `PERFORM` statement sets the return at the end of `LABEL4` but does not modify the continuation address of `LABEL3`, expecting it to be the default continuation. Thus, when the inner invocation arrives at the end of `LABEL3`, it jumps back to the outer `PERFORM` statement, and the program stops having printed just "1 2 3". On the other hand, in some COBOL implementations like the open-source TinyCOBOL compiler, the two `PERFORM` statements do not interfere with each other and the output is indeed "1 2 3 4 3". Therefore, the behaviour in such cases is not only (perhaps) surprising, it is also not portable.[140]

A special consequence of this limitation is that `PERFORM` cannot be used to write recursive code. Another simple example to illustrate this (slightly simplified from Veerman & Verhoeven 2006):

```
    MOVE 1 TO A
    PERFORM LABEL
    STOP RUN.
LABEL.
    DISPLAY A
    IF A < 3
        ADD 1 TO A
        PERFORM LABEL
    END-IF
    DISPLAY 'END'.
```

One might expect that the output is "1 2 3 END END END", and in fact that is what some COBOL compilers will produce. But some compilers, like IBM COBOL, will produce code that prints "1 2 3 END END END END ..." and so on, printing "END" over and over in an endless loop. Since there is limited space to store backup continuation addresses, the backups get overwritten in the course of recursive invocations, and all that can be restored is the jump back to `DISPLAY 'END'`.[140]

Statements

COBOL 2014 has 47 statements (also called *verbs*),[141] which can be grouped into the following broad categories: control flow, I/O, data manipulation and the report writer. The report writer statements are covered in the report writer section.

Control flow

COBOL's conditional statements are IF and EVALUATE. EVALUATE is a switch-like statement with the added capability of evaluating multiple values and conditions. This can be used to implement decision tables. For example, the following might be used to control a CNC lathe:

```
EVALUATE TRUE ALSO desired-speed ALSO current-speed
    WHEN lid-closed ALSO min-speed THRU max-speed ALSO LESS THAN desired-speed
        PERFORM speed-up-machine
    WHEN lid-closed ALSO min-speed THRU max-speed ALSO GREATER THAN desired-speed
        PERFORM slow-down-machine
    WHEN lid-open ALSO ANY ALSO NOT ZERO
        PERFORM emergency-stop
    WHEN OTHER
        CONTINUE
END-EVALUATE
```

The PERFORM statement is used to define loops which are executed *until* a condition is true (not *while* true, which is more common in other languages). It is also used to call procedures or ranges of procedures (see the procedures section for more details). CALL and INVOKE call subprograms and methods, respectively. The name of the subprogram/method is contained in a string which may be a literal or a data item.[142] Parameters can be passed by reference, by content (where a copy is passed by reference) or by value (but only if a prototype is available).[143] CANCEL unloads subprograms from memory. GO TO causes the program to jump to a specified procedure.

The GOBACK statement is a return statement and the STOP statement stops the program. The EXIT statement has six different formats: it can be used as a return statement, a break statement, a continue statement, an end marker or to leave a procedure.[144]

Exceptions are raised by a RAISE statement and caught with a handler, or *declarative*, defined in the DECLARATIVES portion of the procedure division. Declaratives are sections beginning with a USE statement which specify the errors to handle. Exceptions can be names or objects. RESUME is used in a declarative to jump to the statement after the one that raised the exception or to a procedure outside the DECLARATIVES. Unlike other languages, uncaught exceptions may not terminate the program and the program can proceed unaffected.

I/O

File I/O is handled by the self-describing OPEN, CLOSE, READ, and WRITE statements along with a further three: REWRITE, which updates a record; START, which selects subsequent records to access by finding a record with a certain key; and UNLOCK, which releases a lock on the last record accessed.

User interaction is done using ACCEPT and DISPLAY.

Data manipulation

The following verbs manipulate data:

- INITIALIZE, which sets data items to their default values.
- MOVE, which assigns values to data items ; *MOVE CORRESPONDING* assigns corresponding like-named fields.
- SET, which has 15 formats: it can modify indices, assign object references and alter table capacities, among other functions.[145]

- ADD, SUBTRACT, MULTIPLY, DIVIDE, and COMPUTE, which handle arithmetic (with COMPUTE assigning the result of a formula to a variable).
- ALLOCATE and FREE, which handle dynamic memory.
- VALIDATE, which validates and distributes data as specified in an item's description in the data division.
- STRING and UNSTRING, which concatenate and split strings, respectively.
- INSPECT, which tallies or replaces instances of specified substrings within a string.
- SEARCH, which searches a table for the first entry satisfying a condition.

Files and tables are sorted using SORT and the MERGE verb merges and sorts files. The RELEASE verb provides records to sort and RETURN retrieves sorted records in order.

Scope termination

Some statements, such as IF and READ, may themselves contain statements. Such statements may be terminated in two ways: by a period (*implicit termination*), which terminates *all* unterminated statements contained, or by a scope terminator, which terminates the nearest matching open statement.

```
*> Terminator period ("implicit termination")
IF invalid-record
    IF no-more-records
        NEXT SENTENCE
    ELSE
        READ record-file
            AT END SET no-more-records TO TRUE.

*> Scope terminators ("explicit termination")
IF invalid-record
    IF no-more-records
        CONTINUE
    ELSE
        READ record-file
            AT END SET no-more-records TO TRUE
        END-READ
    END-IF
END-IF
```

Nested statements terminated with a period are a common source of bugs.[146][147] For example, examine the following code:

```
IF x
    DISPLAY y.
    DISPLAY z.
```

Here, the intent is to display y and z if condition x is true. However, z will be displayed whatever the value of x because the IF statement is terminated by an erroneous period after DISPLAY y.

Another bug is a result of the dangling else problem, when two IF statements can associate with an ELSE.

```
IF x
    IF y
        DISPLAY a
ELSE
    DISPLAY b.
```

In the above fragment, the `ELSE` associates with the `IF y` statement instead of the `IF x` statement, causing a bug. Prior to the introduction of explicit scope terminators, preventing it would require `ELSE NEXT SENTENCE` to be placed after the inner `IF`.[147]

Self-modifying code

The original (1959) COBOL specification supported the infamous `ALTER X TO PROCEED TO Y` statement, for which many compilers generated self-modifying code. `X` and `Y` are procedure labels, and the single `GO TO` statement in procedure `X` executed after such an `ALTER` statement means `GO TO Y` instead. Many compilers still support it,[148] but it was deemed obsolete in the COBOL 1985 standard and deleted in 2002.[149]

The `ALTER` statement was poorly regarded because it undermined "locality of context" and made a program's overall logic difficult to comprehend.[150] As textbook author Daniel D. McCracken wrote in 1976, when "someone who has never seen the program before must become familiar with it as quickly as possible, sometimes under critical time pressure because the program has failed ... the sight of a GO TO statement in a paragraph by itself, signaling as it does the existence of an unknown number of ALTER statements at unknown locations throughout the program, strikes fear in the heart of the bravest programmer."[150]

Hello, world

A "Hello, world" program in COBOL:

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. hello-world.
       PROCEDURE DIVISION.
           DISPLAY "Hello, world!"
           .
```

When the – now famous – "Hello, World!" program example in *The C Programming Language* was first published in 1978 a similar mainframe COBOL program sample would have been submitted through JCL, very likely using a punch card reader, and 80 column punch cards. The listing below, *with an empty DATA DIVISION*, was tested using Linux and the System/370 Hercules emulator running MVS 3.8J. The JCL, written in July 2015, is derived from the Hercules tutorials and samples hosted by Jay Moseley.[151] In keeping with COBOL programming of that era, HELLO, WORLD is displayed in all capital letters.

```
//COBUCLG  JOB (001),'COBOL BASE TEST',                                 00010000
//             CLASS=A,MSGCLASS=A,MSGLEVEL=(1,1)                        00020000
//BASETEST EXEC COBUCLG                                                 00030000
//COB.SYSIN DD *                                                        00040000
 00000* VALIDATION OF BASE COBOL INSTALL                                00050000
 01000 IDENTIFICATION DIVISION.                                         00060000
 01100 PROGRAM-ID. 'HELLO'.                                             00070000
 02000 ENVIRONMENT DIVISION.                                            00080000
 02100 CONFIGURATION SECTION.                                           00090000
 02110 SOURCE-COMPUTER.  GNULINUX.                                      00100000
 02120 OBJECT-COMPUTER.  HERCULES.                                      00110000
 02200 SPECIAL-NAMES.                                                   00120000
 02210     CONSOLE IS CONSL.                                            00130000
 03000 DATA DIVISION.                                                   00140000
 04000 PROCEDURE DIVISION.                                              00150000
 04100 00-MAIN.                                                         00160000
 04110     DISPLAY 'HELLO, WORLD' UPON CONSL.                           00170000
 04900     STOP RUN.                                                    00180000
//LKED.SYSLIB DD DSNAME=SYS1.COBLIB,DISP=SHR                            00190000
//            DD DSNAME=SYS1.LINKLIB,DISP=SHR                           00200000
```

```
//GO.SYSPRINT DD SYSOUT=A                                                00210000
//                                                                       00220000
```

After submitting the JCL, the MVS console displayed:

```
    19.52.48 JOB    3  $HASP100 COBUCLG  ON READER1     COBOL BASE TEST
    19.52.48 JOB    3  IEF677I WARNING MESSAGE(S) FOR JOB COBUCLG  ISSUED
    19.52.48 JOB    3  $HASP373 COBUCLG  STARTED - INIT 1 - CLASS A - SYS BSP1
    19.52.48 JOB    3  IEC130I SYSPUNCH DD STATEMENT MISSING
    19.52.48 JOB    3  IEC130I SYSLIB   DD STATEMENT MISSING
    19.52.48 JOB    3  IEC130I SYSPUNCH DD STATEMENT MISSING
    19.52.48 JOB    3  IEFACTRT - Stepname  Procstep  Program   Retcode
    19.52.48 JOB    3  COBUCLG    BASETEST  COB       IKFCBL00  RC= 0000
    19.52.48 JOB    3  COBUCLG    BASETEST  LKED      IEWL      RC= 0000
    19.52.48 JOB    3  +HELLO, WORLD
    19.52.48 JOB    3  COBUCLG    BASETEST  GO        PGM=*.DD  RC= 0000
    19.52.48 JOB    3  $HASP395 COBUCLG  ENDED
```

Line 10 of the console listing above is highlighted for effect, the highlighting is not part of the actual console output.

The associated compiler listing generated over four pages of technical detail and job run information, for the single line of output from the 14 lines of COBOL.

Reception

Lack of structure

In the 1970s, adoption of the structured programming paradigm was becoming increasingly widespread. Edsger Dijkstra, a preeminent computer scientist, wrote a letter to the editor of Communications of the ACM, published 1975 entitled "How do we tell truths that might hurt?", in which he was critical of COBOL and several other contemporary languages; remarking that "the use of COBOL cripples the mind".[152] In a published dissent to Dijkstra's remarks, the computer scientist Howard E. Tompkins claimed that unstructured COBOL tended to be "written by programmers that have never had the benefit of structured COBOL taught well", arguing that the issue was primarily one of training.[153]

One cause of spaghetti code was the `GO TO` statement. Attempts to remove `GO TO`s from COBOL code, however, resulted in convoluted programs and reduced code quality.[154] `GO TO`s were largely replaced by the `PERFORM` statement and procedures, which promoted modular programming[154] and gave easy access to powerful looping facilities. However, `PERFORM` could only be used with procedures so loop bodies were not located where they were used, making programs harder to understand.[155]

COBOL programs were infamous for being monolithic and lacking modularization.[156] COBOL code could only be modularized through procedures, which were found to be inadequate for large systems. It was impossible to restrict access to data, meaning a procedure could access and modify *any* data item. Furthermore, there was no way to pass parameters to a procedure, an omission Jean Sammet regarded as the committee's biggest mistake.[157] Another complication stemmed from the ability to `PERFORM THRU` a specified sequence of procedures. This meant that control could jump to and return from any procedure, creating convoluted control flow and permitting a programmer to break the single-entry single-exit rule.[158]

This situation improved as COBOL adopted more features. COBOL-74 added subprograms, giving programmers the ability to control the data each part of the program could access. COBOL-85 then added nested subprograms, allowing programmers to hide subprograms.[159] Further control over data and code came in 2002 when object-oriented programming, user-defined functions and user-defined data types were included.

Nevertheless, much important legacy COBOL software uses unstructured code, which has become unmaintainable. It can be too risky and costly to modify even a simple section of code, since it may be used from unknown places in unknown ways.[160]

Compatibility issues

COBOL was intended to be a highly portable, "common" language. However, by 2001, around 300 dialects had been created.[161] One source of dialects was the standard itself: the 1974 standard was composed of one mandatory nucleus and eleven functional modules, each containing two or three levels of support. This permitted 104,976 official variants.[162]

COBOL-85 was not fully compatible with earlier versions, and its development was controversial. Joseph T. Brophy, the CIO of Travelers Insurance, spearheaded an effort to inform COBOL users of the heavy reprogramming costs of implementing the new standard.[163] As a result, the ANSI COBOL Committee received more than 2,200 letters from the public, mostly negative, requiring the committee to make changes. On the other hand, conversion to COBOL-85 was thought to increase productivity in future years, thus justifying the conversion costs.[164]

Verbose syntax

COBOL: /koh′bol/, n.
A weak, verbose, and flabby language used by code grinders to do boring mindless things on dinosaur mainframes. [...] Its very name is seldom uttered without ritual expressions of disgust or horror.

The Jargon File 4.4.8.[165]

COBOL syntax has often been criticized for its verbosity. Proponents say that this was intended to make the code self-documenting, easing program maintenance.[166] COBOL was also intended to be easy for programmers to learn and use,[167] while still being readable to non-technical staff such as managers.[168][169][170][171] The desire for readability led to the use of English-like syntax and structural elements, such as nouns, verbs, clauses, sentences, sections, and divisions. Yet by 1984, maintainers of COBOL programs were struggling to deal with "incomprehensible" code[170] and the main changes in COBOL-85 were there to help ease maintenance.[88]

Jean Sammet, a short-range committee member, noted that "little attempt was made to cater to the professional programmer, in fact people whose main interest is programming tend to be very unhappy with COBOL" which she attributed to COBOL's verbose syntax.[172]

Isolation from the computer science community

The COBOL community has always been isolated from the computer science community. No academic computer scientists participated in the design of COBOL: all of those on the committee came from commerce or government. Computer scientists at the time were more interested in fields like numerical analysis, physics and system programming than the commercial file-processing problems which COBOL development tackled.[173] Jean Sammet attributed COBOL's unpopularity to an initial "snob reaction" due to its inelegance, the lack of influential computer scientists participating in the design process and a disdain for business data processing.[174] The COBOL specification used a unique "notation", or metalanguage, to define its syntax rather than the new Backus–Naur form which the committee did not know of. This resulted in "severe" criticism.[175][176][66]

Later, COBOL suffered from a shortage of material covering it; it took until 1963 for introductory books to appear (with Richard D. Irwin publishing a college textbook on COBOL in 1966).[177] By 1985, there were twice as many books on Fortran and four times as many on BASIC as on COBOL in the Library of Congress.[117] University professors taught more modern, state-of-the-art languages and techniques instead of COBOL which was said to have a "trade school" nature.[178] Donald Nelson, chair of the CODASYL COBOL committee, said in 1984 that "academics ... hate COBOL" and that computer science graduates "had 'hate COBOL' drilled into them".[179] A 2013 poll by Micro Focus found that 20% of university academics thought COBOL was outdated or dead and that 55% believed their students thought COBOL was outdated or dead. The same poll also found that only 25% of academics had COBOL programming on their curriculum even though 60% thought they should teach it.[180] In contrast, in 2003, COBOL featured in 80% of information systems curricula in the United States, the same proportion as C++ and Java.[181]

There was also significant condescension towards COBOL in the business community from users of other languages, for example FORTRAN or assembler, implying that COBOL could be used only for non-challenging problems.

Concerns about the design process

Doubts have been raised about the competence of the standards committee. Short-term committee member Howard Bromberg said that there was "little control" over the development process and that it was "plagued by discontinuity of personnel and ... a lack of talent."[76] Jean Sammet and Jerome Garfunkel also noted that changes introduced in one revision of the standard would be reverted in the next, due as much to changes in who was in the standard committee as to objective evidence.[182]

COBOL standards have repeatedly suffered from delays: COBOL-85 arrived five years later than hoped,[183] COBOL 2002 was five years late,[2] and COBOL 2014 was six years late.[95][184] To combat delays, the standard committee allowed the creation of optional addenda which would add features more quickly than by waiting for the next standard revision. However, some committee members raised concerns about incompatibilities between implementations and frequent modifications of the standard.[185]

Influences on other languages

COBOL's data structures influenced subsequent programming languages. Its record and file structure influenced PL/I and Pascal, and the `REDEFINES` clause was a predecessor to Pascal's variant records. Explicit file structure definitions preceded the development of database management systems and aggregated data was a significant advance over Fortran's arrays.[117] `PICTURE` data declarations were incorporated into PL/I, with minor changes.

COBOL's `COPY` facility, although considered "primitive",[186] influenced the development of include directives.[117]

The focus on portability and standardization meant programs written in COBOL could be portable and facilitated the spread of the language to a wide variety of hardware platforms and operating systems.[187] Additionally, the well-defined division structure restricts the definition of external references to the Environment Division, which simplifies platform changes in particular.[188]

See also

- Alphabetical list of programming languages
- BLIS/COBOL
- COBOL ReSource
- CODASYL
- Comparison of programming languages
- Generational list of programming languages § COBOL based
- List of compilers § COBOL compilers

Computer programming portal

Notes

a. Specifically influenced COBOL 2002's object-oriented features.[2][3][4]

b. The tombstone is currently at the Computer History Museum.[61]

c. Vendor-specific extensions cause many implementations to have far more: one implementation recognizes over 1,100 keywords.[114]

References

Citations

1. Sammet, Jean E. (March 2000). "The real creators of Cobol". *IEEE Software*. **17** (2): 30–32. doi:10.1109/52.841602 (https://doi.org/10.1109%2F52.841602). ISSN 1937-4194 (https://www.worldcat.org/issn/1937-4194). "The Short-Range Committee worked diligently from June 1959 on, but there were great difficulties in having a fairly large committee try to create a programming language. In November, the Short-Range Committee chair appointed six people to develop specifications for consideration: William Selden and Gertrude Tierney (IBM), Howard Bromberg and Norman Discount (RCA), and Vernon Reeves and Jean E. Sammet (Sylvania Electric Products). We worked for two full weeks (including some round-the-clock sessions) in November 1959 and sent the proposed specifications to the full Short-Range Committee, which accepted almost all of them. After some editing (by the same six people), we turned in the specifications as a final report in December to the Executive Committee, which accepted them in January 1960. After some further editing, the Government Printing Office issued Cobol 60. [...] [Grace Hopper] did not participate in its work except through the general guidance she gave to her staff who were direct committee members. Thus, while her indirect influence was very important, regrettably the frequent repeated statements that "Grace Hopper developed Cobol" or "Grace Hopper was a codeveloper of Cobol" or "Grace Hopper is the mother of Cobol" are just not correct."
2. Saade, Henry; Wallace, Ann (October 1995). "COBOL '97: A Status Report" (https://web.archive.org/web/20140422232229/http://collaboration.cmc.ec.gc.ca/science/rpn/biblio/ddj/Website/articles/DDJ/1995/9510/9510e/9510e.htm). *Dr. Dobb's Journal*. Archived from the original (http://collaboration.cmc.ec.gc.ca/science/rpn/biblio/ddj/Website/articles/DDJ/1995/9510/9510e/9510e.htm) on 22 April 2014. Retrieved 21 April 2014.
3. Arranga, Edmund C.; Coyle, Frank P. (February 1998). *Object-Oriented COBOL* (https://books.google.com/books?id=posN1cl6XFUC&pg=PA15). Cambridge University Press. p. 15. ISBN 978-0132611404. "Object-Oriented COBOL's style reflects the influence of Smalltalk and C++."

4. Arranga, Edmund C.; Coyle, Frank P. (March 1997). "Cobol: Perception and Reality". *Computer*. **30** (3): 127. doi:10.1109/2.573683 (https://doi.org/10.1109%2F2.573683). ISSN 0018-9162 (https://www.worldcat.org/issn/0018-9162).
5. Imajo, Tetsuji; et al. (September 2000). *COBOL Script: a business-oriented scripting language*. Enterprise Distributed Object Computing Conference (http://dblp1.uni-trier.de/db/conf/edoc/edoc2000.html). Makuhari, Japan: IEEE. doi:10.1109/EDOC.2000.882363 (https://doi.org/10.1109%2FEDOC.2000.882363). ISBN 0769508650.
6. Ho, Wing Hong (7 May 2007). "Introduction to EGL" (https://www.omg.org/adm/EGLOverviewtoOMG.pdf) (PDF). IBM Software Group.
7. Radin, George (1978). Wexelblat, Richard L. (ed.). *The early history and characteristics of PL/I*. History of Programming Languages. Academic Press (published 1981). p. 572. doi:10.1145/800025.1198410 (https://doi.org/10.1145%2F800025.1198410). ISBN 0127450408.
8. Mitchell, Robert L. (14 March 2012). "Brain drain: Where Cobol systems go from here" (http://www.computerworld.com/article/2502420/data-center/brain-drain-where-cobol-systems-go-from-here.html). *Computerworld*. Retrieved 9 February 2015.
9. Mitchell, Robert L. (4 October 2006). "Cobol: Not Dead Yet" (http://www.computerworld.com/s/article/266156/Cobol_Not_Dead_Yet). *Computerworld*. Retrieved 27 April 2014.
10. Ensmenger, Nathan L. (2009). *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise* (https://books.google.com/books?id=VCcsTPQ738oC&pg=PA100). MIT Press. p. 100. ISBN 978-0262050937. LCCN 2009052638 (https://lccn.loc.gov/2009052638).
11. "ISO/IEC 1989:2014" (http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=51416). ISO. 26 May 2014. Retrieved 7 June 2014.
12. Ferguson, Andrew. "A History of Computer Programming Languages" (http://cs.brown.edu/~adf/programming_languages.html). *cs.brown.edu*.
13. Beyer 2009, p. 282.
14. Gürer, Denise (1 June 2002). "Pioneering Women in Computer Science". *SIGCSE Bull*. **34** (2): 175–180. doi:10.1145/543812.543853 (https://doi.org/10.1145%2F543812.543853). ISSN 0097-8418 (https://www.worldcat.org/issn/0097-8418). S2CID 2577644 (https://api.semanticscholar.org/CorpusID:2577644).
15. Beyer 2009, pp. 281–282.
16. Sammet 1978a, p. 200.
17. Flahive, Paul (24 May 2019). "How COBOL Still Powers The Global Economy At 60 Years Old" (https://web.archive.org/web/20190524035248/https://www.tpr.org/post/how-cobol-still-powers-global-economy-60-years-old). *Texas Public Radio*. Archived from the original (https://www.tpr.org/post/how-cobol-still-powers-global-economy-60-years-old) on 24 May 2019. Retrieved 19 July 2019. "(Grace Hopper) Nicknamed Grandma Cobol, the code was based on some of her earlier work. She said — after hearing the rumors — one of her collaborators went out and bought a granite tombstone. "He had the word COBOL cut in the front of it. Then he shipped it express collect to Mr. Phillips in the pentagon." The prank on Charles Phillips, a leader for the project at the defense department, got the attention of the powers that be and was a turning point she said. COBOL would go on to become the most widely used and longest lasting computer languages in history."
18. Beyer 2009, p. 283.
19. Beyer 2009, p. 284.
20. "Early Meetings of the Conference on Data Systems Languages". *IEEE Annals of the History of Computing*. **7** (4): 316–325. 1985. doi:10.1109/MAHC.1985.10047 (https://doi.org/10.1109%2FMAHC.1985.10047). S2CID 35625728 (https://api.semanticscholar.org/CorpusID:35625728).
21. Sammet 2004, p. 104.

22. Beyer 2009, p. 286.
23. Conner 1984, p. ID/9.
24. Sammet 1978a, p. 201.
25. Bemer 1971, p. 132.
26. Beyer 2009, p. 288.
27. Sammet 1978a, p. 203.
28. CODASYL 1969, § I.2.1.1.
29. Sammet 1978a, p. 204.
30. CODASYL 1969, § I.1.2.
31. Beyer 2009, p. 290.
32. Sammet, Jean (1978). "The Early History of COBOL". *ACM SIGPLAN Notices*. **13** (8): 121–161. doi:10.1145/960118.808378 (https://doi.org/10.1145%2F960118.808378). S2CID 10743643 (https://api.semanticscholar.org/CorpusID:10743643).
33. Sammet 1978a, p. 217.
34. Adams, Vicki Porter (5 October 1981). "Captain Grace M. Hopper: the Mother of COBOL" (https://books.google.com/books?id=JT0EAAAAMBAJ&pg=RA1-PA33). *InfoWorld*. Vol. 3 no. 20. p. 33. ISSN 0199-6649 (https://www.worldcat.org/issn/0199-6649).
35. Betts, Mitch (6 January 1992). "Grace Hopper, mother of Cobol, dies" (https://books.google.com/books?id=J-_T3bxgvMwC&pg=PA14). *Computerworld*. **26** (1): 14.
36. Lohr, Steve (2008). *Go To: The Story of the Math Majors, Bridge Players, Engineers, Chess Wizards, Maverick Scientists, and Iconoclasts—The Programmers Who Created the Software Revolution* (https://books.google.com/books?id=XfPLVx6qS_cC&pg=PA52). Basic Books. p. 52. ISBN 978-0786730766.
37. "Pioneering software engineer and Cobol co-designer" (https://www.irishtimes.com/life-and-style/people/pioneering-software-engineer-and-cobol-co-designer-1.3111332).
38. Beyer 2009, p. 292.
39. Bemer 1971, p. 131.
40. Beyer 2009, p. 296.
41. Sammet 1978a, p. 221.
42. Beyer 2009, p. 291.
43. "Oral History of Captain Grace Hopper" (https://web.archive.org/web/20171225202555/http://archive.computerhistory.org/resources/text/Oral_History/Hopper_Grace/102702026.05.01.pdf) (PDF). Computer History Museum. December 1980. p. 37. Archived from the original (http://archive.computerhistory.org/resources/text/Oral_History/Hopper_Grace/102702026.05.01.pdf) (PDF) on 25 December 2017. Retrieved 28 June 2014.
44. Sammet 1978a, p. 218.
45. Marcotty 1978a, p. 268.
46. Sammet 1978a, pp. 205–206.
47. Sammet 1978a, Figure 8.
48. Sammet 1978a, pp. 230–231.
49. ISO/IEC JTC 1/SC 22/WG 4 2001, p. 846.
50. Sammet 1978a, p. 220.
51. Sammet 1978a, p. 228.
52. Sammet 1978a, p. 210.
53. Bemer 1971, p. 132: *We can't find a single individual who admits coining the acronym "COBOL"*.

54. Sammet 1978a, p. 210: *The next day, the name COBOL was finally agreed to as an acronym for COmmon Business Oriented Language. Unfortunately, my notes do not show who made that suggestion*.
55. Sullivan, Patricia (25 June 2004). "Computer Pioneer Bob Bemer, 84" (https://www.washingtonpost.com/wp-dyn/articles/A4138-2004Jun24.html). *The Washington Post*. p. B06. Retrieved 28 June 2014.
56. "THE COBOL REPORT - Interview with Bob Bemer - the Father of COBOL" (https://web.archive.org/web/20180402200259/http://www.bobbemer.com/arranga.htm). Archived from the original (http://www.bobbemer.com/arranga.htm) on 2 April 2018.
57. "THE COBOL REPORT - Interview with Bob Bemer - the Father of COBOL" (https://web.archive.org/web/20031223115509/http://cobolreport.com/cobolreport/archives/TCR_bemer.htm). Archived from the original (http://cobolreport.com/cobolreport/archives/TCR_bemer.htm) on 23 December 2003.
58. Beyer 2009, p. 293.
59. Beyer 2009, p. 294.
60. "The Story of the COBOL Tombstone" (http://ed-thelen.org/comp-hist/TCMR-V13.pdf) (PDF). *The Computer Museum Report*. **13**: 8–9. Summer 1985. Archived (https://web.archive.org/web/20140403015336/http://ed-thelen.org/comp-hist/TCMR-V13.pdf) (PDF) from the original on 3 April 2014. Retrieved 29 June 2014.
61. *COBOL Tombstone* (http://www.computerhistory.org/collections/catalog/X572.85). Computer History Museum. 1960. Retrieved 29 June 2014.
62. Bemer 1971, p. 130.
63. Beyer 2009, p. 289.
64. CODASYL 1969, § I.1.1.
65. Brown 1976, p. 47.
66. Bemer 1971, p. 133.
67. Beyer 2009, p. 297.
68. Williams, Kathleen Broome (10 November 2012). *Grace Hopper: Admiral of the Cyber Sea* (https://books.google.com/books?id=KKmiw-_2gYIC&pg=PT124). US Naval Institute Press. ISBN 978-1612512655. OCLC 818867202 (https://www.worldcat.org/oclc/818867202).
69. Compaq Computer Corporation: *Compaq COBOL Reference Manual*, Order Number: AA–Q2G0F–TK October 2000, Page xviii; Fujitsu Corporation: *Net Cobol Language Reference*, Version 15, January 2009; IBM Corporation: *Enterprise COBOL for z/OS Language Reference*, Version 4 Release 1, SC23-8528-00, December 2007
70. Garfunkel, Jerome (11 November 1984). "In defense of Cobol" (https://books.google.com/books?id=CLbHc0Acrm4C&pg=RA1-PA67). *Computerworld*. **18** (24): ID/19.
71. Bemer 1971, p. 134.
72. Brown 1976, p. 48.
73. CODASYL 1969, § I.2.2.4.
74. CODASYL 1969, § I.2.3.
75. Follet, Robert H.; Sammet, Jean E. (2003). "Programming language standards" (http://dl.acm.org/citation.cfm?id=1074734). In Ralston, Anthony; Reilly, Edwin D.; Hemmendinger, David (eds.). *Encyclopedia of Computer Science* (4th ed.). Wiley. p. 1467. ISBN 978-0470864128.
76. Beyer 2009, p. 301.
77. Brown 1976, p. 49.
78. Brown 1976, p. 52.
79. Taylor, Alan (2 August 1972). "Few Realise Wasted Resources of Local DP Schools" (https://books.google.com/books?id=JBUJf4n2QxkC&pg=PT10). *Computerworld*. **6** (31): 11.

80. Triance, J. M. (1974). *Programming in COBOL: A Course of Twelve Television Lectures* (https://books.google.com/books?id=Cs1RAQAAIAAJ&pg=PA87). Manchester University Press. p. 87. ISBN 978-0719005923.
81. Klein 2010, p. 16.
82. Baird, George N.; Oliver, Paul (May 1977). "1974 Standard (X3.23–1974)". Programming Language Standards—Who Needs Them? (http://www.dtic.mil/dtic/tr/fulltext/u2/a039740.pdf) (PDF) (Report). Department of the Navy. pp. 19–21. Archived (https://web.archive.org/web/20140107192439/http://www.dtic.mil/dtic/tr/fulltext/u2/a039740.pdf) (PDF) from the original on 7 January 2014. Retrieved 7 January 2014.
83. Culleton, John R., Jr. (23 July 1975). " 'Spotty' Availability A Problem..." (https://books.google.com/books?id=8pMVcgpPyVMC&pg=PA17) *Computerworld*. **9** (30): 17.
84. Simmons, Williams B. (18 June 1975). "Does Cobol's Report Writer Really Miss the Mark?" (https://books.google.com/books?id=X_3_D4RqzvIC&pg=PA20). *Computerworld*. **9** (25): 20.
85. Shoor, Rita (26 January 1981). "User Threatens Suit Over Ansi Cobol-80" (https://books.google.com/books?id=d514ApKzvjYC). *Computerworld*. **15** (4): 1, 8.
86. Shoor, Rita (26 October 1981). "DPMA Takes Stand Against Cobol Draft" (https://books.google.com/books?id=1REkdf3I86oC). *Computerworld*. **15** (43): 1–2.
87. Gallant, John (16 September 1985). "Revised Cobol standard may be ready in late '85" (https://books.google.com/books?id=zrOC44tBR68C&pg=PA8). *Computerworld*. **19** (37): 1, 8.
88. "Expert addresses Cobol 85 standard" (https://books.google.com/books?id=zrOC44tBR68C&pg=PA41). *Computerworld*. **19** (37): 41, 48. 16 September 1985.
89. Paul, Lois (15 March 1982). "Responses to Cobol-80 Overwhelmingly Negative" (https://books.google.com/books?id=Wz-oh7ZQo8MC). *Computerworld*. **16** (11): 1, 5.
90. Paul, Lois (25 April 1983). "Study Sees Few Problems Switching to Cobol-8X" (https://books.google.com/books?id=Of5OA6T_6UIC&pg=PA1). *Computerworld*. **17** (17): 1, 6.
91. Gillin, Paul (19 November 1984). "DEC users get head start implementing Cobol-80" (https://books.google.com/books?id=1qju5_k3q9AC&pg=PA1). *Computerworld*. **18** (47): 1, 6.
92. Garfunkel 1987, p. 150.
93. Roy, M. K.; Dastidar, D. Ghost (1 June 1989). "Features of COBOL-85". *COBOL Programming: Problems and Solutions* (https://books.google.com/books?id=N066w1XgJXcC&pg=PA438) (2nd ed.). McGraw-Hill Education. pp. 438–451. ISBN 978-0074603185.
94. Robinson, Brian (9 July 2009). "Cobol remains old standby at agencies despite showing its age" (http://fcw.com/Articles/2009/07/13/TECH-COBOL-turns-50.aspx). *FCW*. Public Sector Media Group. Retrieved 26 April 2014.
95. "COBOL Standards" (https://web.archive.org/web/20040331054413/http://www.cobolstandards.com/). Micro Focus. Archived from the original (http://www.cobolstandards.com/) on 31 March 2004. Retrieved 2 September 2014.
96. "NetCOBOL for .Net" (https://web.archive.org/web/20140708210107/http://www.netcobol.com/product/netcobol-for-net/). *netcobol.com*. GTSoftware. 2013. Archived from the original (http://www.netcobol.com/product/netcobol-for-net/) on 8 July 2014. Retrieved 29 January 2014.
97. "A list of Codasyl Cobol features" (https://books.google.com/books?id=VQhbdDusHcsC&pg=RA1-PA60). *Computerworld*. **18** (37): ID/28. 10 September 1984. Retrieved 8 June 2014.
98. ISO/IEC JTC 1/SC 22/WG 4 2001, Annex F.
99. Klein 2010, p. 21.
100. "JTC1/SC22/WG4 – COBOL" (https://web.archive.org/web/20140214225220/http://www.cobolstandard.info/wg4/wg4.html). ISO. 30 June 2010. Archived from the original (http://www.cobolstandard.info/wg4/wg4.html) on 14 February 2014. Retrieved 27 April 2014.

101. Billman, John; Klink, Huib (27 February 2008). "Thoughts on the Future of COBOL Standardization" (https://web.archive.org/web/20090711032647/http://www.cobolstandard.info/j4/files/08-0034.pdf) (PDF). Archived from the original (http://www.cobolstandard.info/j4/files/08-0034.pdf) (PDF) on 11 July 2009. Retrieved 14 August 2014.
102. ISO/IEC JTC 1/SC 22/WG 4 2014, Annex E.
103. Schricker, Don (2 December 1998). "J4: COBOL Standardization" (https://web.archive.org/web/19990224043617/http://www.microfocus.com/Standards/). Micro Focus. Archived from the original (http://www.microfocus.com/Standards/) on 24 February 1999. Retrieved 12 July 2014.
104. Kizior, Ronald J.; Carr, Donald; Halpern, Paul. "Does COBOL Have a Future?" (https://web.archive.org/web/20160817115437/http://proc.isecon.org/2000/126/ISECON.2000.Kizior.pdf) (PDF). *The Proceedings of the Information Systems Education Conference 2000*. **17** (126). Archived from the original (http://proc.isecon.org/2000/126/ISECON.2000.Kizior.pdf) (PDF) on 17 August 2016. Retrieved 30 September 2012.
105. Carr & Kizior 2003, p. 16.
106. Carr & Kizior 2003, p. 10.
107. "Cobol brain drain: Survey results" (http://www.computerworld.com/s/article/9225099/Cobol_brain_drain_Survey_results). *Computerworld*. 14 March 2012. Retrieved 27 April 2014.
108. Powner, David A. (25 May 2016). "Federal Agencies Need to Address Aging Legacy Systems" (https://web.archive.org/web/20160615044750/https://www.gao.gov/assets/680/677454.pdf) (PDF). *Government Accountability Office*. p. 18. Archived from the original (https://www.gao.gov/assets/680/677454.pdf) (PDF) on 15 June 2016. Retrieved 19 July 2019. "Several agencies, such as the Department of Agriculture (USDA), DHS, HHS, Justice, Treasury, and VA, reported using Common Business Oriented Language (COBOL)—a programming language developed in the late 1950s and early 1960s—to program their legacy systems. It is widely known that agencies need to move to more modern, maintainable languages, as appropriate and feasible."
109. "COBOL blues" (http://fingfx.thomsonreuters.com/gfx/rngs/USA-BANKS-COBOL/010040KH18J/index.html). Reuters. Retrieved 8 April 2020.
110. Teplitzky, Phil (25 October 2019). "Closing the COBOL Programming Skills Gap" (https://ibmsystemsmag.com/IBM-Z/10/2019/closing-cobol-programming-skills-gap). *IBM Systems Magazine, IBM Z*. Retrieved 11 June 2020.
111. Lee, Alicia (8 April 2020). "Wanted urgently: People who know a half century-old computer language so states can process unemployment claims" (https://www.cnn.com/2020/04/08/business/coronavirus-cobol-programmers-new-jersey-trnd/index.html). *CNN*. Retrieved 8 April 2020.
112. Long, Heather; Stein, Jeff; Rein, Lisa; Romm, Tony (17 April 2020). "Stimulus checks and other coronavirus relief hindered by dated technology and rocky government rollout" (https://www.washingtonpost.com/business/2020/04/17/stimulus-unemployment-checks-delays-government-delays/). *The Washington Post*. Retrieved 19 April 2020.
113. ISO/IEC JTC 1/SC 22/WG 4 2014, § 8.9.
114. "Reserved Words Table" (http://documentation.microfocus.com/help/topic/com.microfocus.eclipse.infocenter.visualcobol.vs2013/HRLHLHARES01U005.html). *Micro Focus Visual COBOL 2.2 COBOL Language Reference*. Micro Focus. Retrieved 3 March 2014.
115. ISO/IEC JTC 1/SC 22/WG 4 2014, § 8.3.1.2.
116. ISO/IEC JTC 1/SC 22/WG 4 2014, § 8.3.2.
117. Shneiderman 1985, p. 349.
118. ISO/IEC JTC 1/SC 22/WG 4 2001, § F.2.
119. ISO/IEC JTC 1/SC 22/WG 4 2014, § D.18.2.
120. ISO/IEC JTC 1/SC 22/WG 4 2014, § D.18.
121. ISO/IEC JTC 1/SC 22/WG 4 2014, p. 108.

122. ISO/IEC JTC 1/SC 22/WG 4 2014, p. 896.
123. ISO/IEC JTC 1/SC 22/WG 4 2014, § D.2.1.
124. "File Organizations" (http://supportline.microfocus.com/documentation/books/nx30books/fhorgs.htm). *File Handling*. Micro Focus. 1998. Retrieved 27 June 2014.
125. ISO/IEC JTC 1/SC 22/WG 4 2014, § 8.5.1.2.
126. Cutler 2014, Appendix A.
127. Hubbell, Thane (1999). *Sams Teach Yourself COBOL in 24 hours*. SAMS Publishing. p. 40. ISBN 978-0672314537. LCCN 98087215 (https://lccn.loc.gov/98087215).
128. McCracken & Golden 1988, § 19.9.
129. Cutler 2014, § 5.8.5.
130. ISO/IEC JTC 1/SC 22/WG 4 2014, § 8.5.2.
131. ISO/IEC JTC 1/SC 22/WG 4 2014, § 14.9.24.
132. ISO/IEC JTC 1/SC 22/WG 4 2014, § 14.9.35.
133. ISO/IEC JTC 1/SC 22/WG 4 2014, § 13.18.40.
134. ISO/IEC JTC 1/SC 22/WG 4 2014, § 13.18.60.3.
135. ISO/IEC JTC 1/SC 22/WG 4 2014, p. 855.
136. McCracken 1976, p. 338.
137. ISO/IEC JTC 1/SC 22/WG 4 2014, § 14.4.
138. ISO/IEC JTC 1/SC 22/WG 4 2014, § 14.6.3.
139. Field, John; Ramalingam, G. (September 1999). *Identifying Procedural Structure in Cobol Programs* (http://pages.cs.wisc.edu/~ramali/Papers/paste99.pdf) (PDF). PASTE '99 (http://cseweb.ucsd.edu/~wgg/paste99.html). doi:10.1145/381788.316163 (https://doi.org/10.1145%2F381788.316163). ISBN 1581131372.
140. Veerman, Niels; Verhoeven, Ernst-Jan (November 2006). "Cobol minefield detection" (https://web.archive.org/web/20070306135410/http://www.cs.vu.nl/~nveerman/research/minefield/minefield.pdf) (PDF). *Software: Practice and Experience*. **36** (14). doi:10.1002/spe.v36:14 (https://doi.org/10.1002%2Fspe.v36%3A14). Archived from the original (http://www.cs.vu.nl/~nveerman/research/minefield/minefield.pdf) (PDF) on 6 March 2007.
141. ISO/IEC JTC 1/SC 22/WG 4 2014, § 14.9.
142. ISO/IEC JTC 1/SC 22/WG 4 2014, §§ 14.9.4, 14.9.22.
143. ISO/IEC JTC 1/SC 22/WG 4 2014, § D.6.5.2.2.
144. ISO/IEC JTC 1/SC 22/WG 4 2014, § 14.9.13.1.
145. ISO/IEC JTC 1/SC 22/WG 4 2014, §14.9.35.1.
146. ISO/IEC JTC 1/SC 22/WG 4 2014, p. 899.
147. McCracken & Golden 1988, § 8.4.

148. Examples of compiler support for `ALTER` can be seen in the following:
    - Tiffin, Brian (18 September 2013). "September 2013" (https://web.archive.org/web/20140505181734/http://sourceforge.net/p/open-cobol/discussion/cobol/thread/7dc2941f/). *GNU Cobol*. Archived from the original (http://sourceforge.net/p/open-cobol/discussion/cobol/thread/7dc2941f/#5ee9) on 5 May 2014. Retrieved 5 January 2014.
    - "The ALTER Statement" (http://documentation.microfocus.com/help/topic/com.microfocus.eclipse.infocenter.visualcobol.vs2013/HRLHLHPDF803.html). *Micro Focus Visual COBOL 2.2 for Visual Studio 2013 COBOL Language Reference*. Micro Focus. Retrieved 5 January 2014.
    - "ALTER Statement (Nucleus)" (https://web.archive.org/web/20140106031540/http://www.csim.scu.edu.tw/~kuo/COBOL/COBOLCompiler/COBOL%E6%89%8B%E5%86%8A/cob_lrf.pdf) (PDF). *COBOL85 Reference Manual*. Fujitsu. November 1996. p. 555. Archived from the original (http://www.csim.scu.edu.tw/~kuo/COBOL/COBOLCompiler/COBOL%E6%89%8B%E5%86%8A/cob_lrf.pdf) (PDF) on 6 January 2014. Retrieved 5 January 2014.
    - "ALTER Statement" (http://pic.dhe.ibm.com/infocenter/pdthelp/v1r1/topic/com.ibm.entcobol.doc_5.1/PGandLR/ref/rlpsalte.html). *Enterprise COBOL for z/OS Language Reference*. IBM. June 2013. Retrieved 5 January 2014.
149. ISO/IEC JTC 1/SC 22/WG 4 2001, § F.1.
150. McCracken 1976, p. 355.
151. Moseley, Jay (17 January 2015). "COBOL Compiler from MVT" (http://www.jaymoseley.com/hercules/compilers/cobol.htm). Retrieved 19 July 2015.
152. Dijkstra, Edsger W. (18 June 1975). "How do we tell truths that might hurt?" (https://web.archive.org/web/20170502143353/http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD498.html). University of Texas at Austin. EWD498. Archived from the original (http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD498.html) on 2 May 2017. Retrieved 29 August 2007.
153. Tompkins, H. E. (1983). "In defense of teaching structured COBOL as computer science". *ACM SIGPLAN Notices*. **18** (4): 86–94. doi:10.1145/948176.948186 (https://doi.org/10.1145%2F948176.948186). S2CID 33803213 (https://api.semanticscholar.org/CorpusID:33803213).
154. Riehle 1992, p. 125.
155. Shneiderman 1985, pp. 349–350.
156. Coughlan, Michael (16 March 2014). *Beginning COBOL for Programmers* (https://books.google.com/books?id=MJmJAwAAQBAJ&pg=PA4). Apress. p. 4. ISBN 978-1430262534. Retrieved 13 August 2014.
157. Sammet 1978b, p. 258.
158. Riehle 1992, p. 126.
159. Riehle 1992, p. 127.
160. "COBOL and Legacy Code as a Systemic Risk | naked capitalism" (http://www.nakedcapitalism.com/2016/07/cobol-and-legacy-code-as-a-systemic-risk.html?imm_mid=0e6043&cmp=em-prog-na-na-newsltr_20160723). 19 July 2016. Retrieved 23 July 2016.
161. Lämmel, Ralf; Verhoef, Chris (November–December 2001). "Cracking the 500-language problem" (https://web.archive.org/web/20140819085841/http://www.cs.vu.nl/grammarware/500/500.pdf) (PDF). *IEEE Software*. **18** (6): 79. doi:10.1109/52.965809 (https://doi.org/10.1109%2F52.965809). hdl:1871/9853 (https://hdl.handle.net/1871%2F9853). Archived from the original (http://www.cs.vu.nl/grammarware/500/500.pdf) (PDF) on 19 August 2014.
162. Howkins, T. J.; Harandi, M. T. (April 1979). "Towards more portable COBOL" (https://doi.org/10.1093%2Fcomjnl%2F22.4.290). *The Computer Journal*. **22** (4): 290. doi:10.1093/comjnl/22.4.290 (https://doi.org/10.1093%2Fcomjnl%2F22.4.290).

163. Garfunkel 1987, p. 11.
164. Garfunkel 1987, p. 15.
165. Raymond, Eric S. (1 October 2004). "COBOL" (http://catb.org/jargon/html/C/COBOL.html). *The Jargon File, version 4.4.8*. Archived (https://web.archive.org/web/20140830184553/http://www.catb.org/jargon/html/C/COBOL.html) from the original on 30 August 2014. Retrieved 13 December 2014.
166. Brown 1976, p. 53.
167. CODASYL 1969, § II.1.1.
168. Shneiderman 1985, p. 350.
169. Sammet 1961, p. 381.
170. Conner 1984, p. ID/10.
171. Marcotty 1978a, p. 263.
172. Conner 1984, p. ID/14.
173. Sammet 1961, p. 380.
174. Marcotty 1978a, p. 266.
175. Sammet 1978b, p. 255.
176. Shneiderman 1985, pp. 348–349.
177. "Archived copy" (https://web.archive.org/web/20160305083547/https://books.bibliopolis.com/main/find/2200821/COBOL-Logic-and-Programming-third-edition-1974-McCameron-Fritz-oldcomputerbooks-com.html). Archived from the original (https://books.bibliopolis.com/main/find/2200821/COBOL-Logic-and-Programming-third-edition-1974-McCameron-Fritz-oldcomputerbooks-com.html) on 5 March 2016. Retrieved 25 February 2016.
178. Shneiderman 1985, p. 351.
179. "An interview: Cobol defender" (https://books.google.com/books?id=VQhbdDusHcsC&pg=RA1-PA61). *Computerworld*. **18** (37): ID/29–ID/32. 10 September 1984. Retrieved 8 June 2014.
180. "Academia needs more support to tackle the IT skills gap" (http://www.microfocus.com/about/press/pressreleases/2013/pr070320131001.aspx) (Press release). Micro Focus. 7 March 2013. Retrieved 4 August 2014.
181. Carr & Kizior 2003, p. 13.
182. Sammet, Jean; Garfunkel, Jerome (October 1985). "Summary of Changes in COBOL, 1960–1985". *Annals of the History of Computing*. **7** (4): 342. doi:10.1109/MAHC.1985.10033 (https://doi.org/10.1109%2FMAHC.1985.10033). S2CID 17940092 (https://api.semanticscholar.org/CorpusID:17940092).
183. Cook, Margaret M. (June 1978). Ghosh, Sakti P.; Liu, Leonard Y. (eds.). *Data Base Facility for COBOL 80* (http://www.computer.org/csdl/proceedings/afips/1978/5086/00/50861107.pdf) (PDF). 1978 National Computer Conference. Anaheim, California: AFIPS Press. pp. 1107–1112. doi:10.1109/AFIPS.1978.63 (https://doi.org/10.1109%2FAFIPS.1978.63). LCCN 55-44701 (https://lccn.loc.gov/55-44701). Retrieved 2 September 2014. "The earliest date that a new COBOL standard could be developed and approved is the year 1980 [...]."
184. "Resolutions from WG4 meeting 24 – June 26–28, 2003 Las Vegas, Nevada, USA" (https://web.archive.org/web/20160308015945/http://www.cobolstandard.info/wg4/open/wg4n0188.doc). 11 July 2003. p. 1. Archived from the original (http://www.cobolstandard.info/wg4/open/wg4n0188.doc) (doc) on 8 March 2016. Retrieved 29 June 2014. "a June 2008 revision of the COBOL standard"
185. Babcock, Charles (14 July 1986). "Cobol standard add-ons flayed" (https://books.google.com/books?id=tQOdtdJmVSsC&pg=PA12). *Computerworld*. **20** (28): 1, 12.
186. Marcotty 1978b, p. 274.

187. This can be seen in:

- "Visual COBOL" (http://www-304.ibm.com/partnerworld/gsd/solutiondetails.do?solution=48040&expand=true&lc=en). *IBM PartnerWorld*. IBM. 21 August 2013. Archived (https://web.archive.org/web/20140712183624/http://www-304.ibm.com/partnerworld/gsd/solutiondetails.do?solution=48040) from the original on 12 July 2014. Retrieved 5 February 2014. "Micro Focus Visual COBOL delivers the next generation of COBOL development and deployment for Linux x86-64, Linux for System z, AIX, HP/UX, Solaris, and Windows."
- "COBOL Compilers family" (https://www-03.ibm.com/software/products/en/cobocompfami). *ibm.com*. IBM. Archived (https://web.archive.org/web/20140223004754/https://www-03.ibm.com/software/products/en/cobocompfami) from the original on 23 February 2014. Retrieved 5 February 2014.
- Tiffin, Brian (4 January 2014). "What platforms are supported by GNU Cobol?" (https://web.archive.org/web/20131214110557/http://opencobol.add1tocobol.com/gnucobol/#what-platforms-are-supported-by-gnu-cobol). Archived from the original (http://opencobol.add1tocobol.com/gnucobol/#what-platforms-are-supported-by-gnucobol) on 14 December 2013. Retrieved 5 February 2014.

188. Coughlan, Michael (2002). "Introduction to COBOL" (http://www.csis.ul.ie/cobol/course/COBOLIntro.htm#part1). Retrieved 3 February 2014.

Sources

- Bemer, Bob (1971). "A View of the History of COBOL" (https://web.archive.org/web/20160122164500/http://archive.computerhistory.org/resources/text/Knuth_Don_X4100/PDF_index/k-8-pdf/k-8-u2776-Honeywell-mag-History-Cobol.pdf) (PDF). *Honeywell Computer Journal*. **5** (3). Archived from the original (http://archive.computerhistory.org/resources/text/Knuth_Don_X4100/PDF_index/k-8-pdf/k-8-u2776-Honeywell-mag-History-Cobol.pdf) (PDF) on 22 January 2016. Retrieved 28 June 2014.
- Beyer, Kurt (2009). *Grace Hopper and the Invention of the Information Age* (https://books.google.com/books?id=u5KKwGjuIEwC&pg=PA282). MIT Press. ISBN 978-0262013109. LCCN 2008044229 (https://lccn.loc.gov/2008044229).
- Brown, William R. (1 December 1976). "COBOL". In Belzer, Jack; Holzman, Albert G.; Kent, Allen (eds.). *Encyclopedia of Computer Science and Technology: Volume 5* (https://books.google.com/books?id=G6a2tSuOoq8C&pg=PA47). CRC Press. ISBN 978-0824722555.
- Carr, Donald E.; Kizior, Ronald J. (31 December 2003). "Continued Relevance of COBOL in Business and Academia: Current Situation and Comparison to the Year 2000 Study" (http://www.isedj.org/1/52/ISEDJ.1(52).Carr.pdf) (PDF). *Information Systems Education Journal*. **1** (52). ISSN 1545-679X (https://www.worldcat.org/issn/1545-679X). Retrieved 4 August 2014.
- CODASYL (July 1969). "CODASYL COBOL Journal of Development 1968" (https://archive.org/details/codasylcoboljour00conf). *Codasyl Cobal Journal of Development*. National Bureau of Standards. ISSN 0591-0218 (https://www.worldcat.org/issn/0591-0218). LCCN 73601243 (https://lccn.loc.gov/73601243).
- Conner, Richard L. (14 May 1984). "Cobol, your age is showing" (https://books.google.com/books?id=BrEo9KtAQH4C&pg=RA1-PA61). *Computerworld*. **18** (20): ID/7–ID/18. ISSN 0010-4841 (https://www.worldcat.org/issn/0010-4841).

- Cutler, Gary (9 April 2014). "GNU COBOL Programmer's Guide" (http://opencobol.add1tocobol.com/guides/GNU%20COBOL%202.1%2023NOV2013%20Programmers%20Guide%20(US%20Letter).pdf) (PDF) (3rd ed.). Retrieved 25 February 2014.
- Garfunkel, Jerome (1987). *The COBOL 85 Example Book*. Wiley. ISBN 978-0471804611.
- ISO/IEC JTC 1/SC 22/WG 4 (4 December 2001). "ISO/IEC IS 1989:2001 – Programming language COBOL" (https://web.archive.org/web/20020124065139/http://www.ncits.org/tc_home/j4htm/cobolv200112.zip). ISO. Archived from the original (http://www.ncits.org/tc_home/j4htm/cobolv200112.zip) (ZIP of PDF) on 24 January 2002. Retrieved 2 September 2014. (Link goes to draft N 0147)
- ISO/IEC JTC 1/SC 22/WG 4 (31 October 2014). *INCITS/ISO/IEC 1989:2014 [2014] – Programming language COBOL*. INCITS.
- Klein, William M. (4 October 2010). "The History of COBOL" (https://web.archive.org/web/20140107192608/http://home.comcast.net/~wmklein/DOX/History.pdf) (PDF). Archived from the original (http://home.comcast.net/~wmklein/DOX/History.pdf) (PDF) on 7 January 2014. Retrieved 7 January 2014.
- Marcotty, Michael (1978a). Wexelblat, Richard L. (ed.). *Transcript of question and answer session* (https://archive.org/details/historyofprogram0000hist/page/263). History of Programming Languages. Academic Press (published 1981). p. 263 (https://archive.org/details/historyofprogram0000hist/page/263). doi:10.1145/800025.1198370 (https://doi.org/10.1145%2F800025.1198370). ISBN 0127450408.
- Marcotty, Michael (1978b). Wexelblat, Richard L. (ed.). *Full text of all questions submitted* (https://archive.org/details/historyofprogram0000hist/page/274). History of Programming Languages. Academic Press (published 1981). p. 274 (https://archive.org/details/historyofprogram0000hist/page/274). doi:10.1145/800025.1198371 (https://doi.org/10.1145%2F800025.1198371). ISBN 0127450408.
- McCracken, Daniel D. (1976). *A Simplified Guide to Structured COBOL Programming* (https://archive.org/details/simplifiedguidet0000mccr). Wiley. ISBN 0-471-58284-0.
- McCracken, Daniel D.; Golden, Donald G. (1988). *A Simplified Guide to Structured COBOL Programming* (2nd ed.). Wiley. ISBN 978-0471610540. LCCN 87034608 (https://lccn.loc.gov/87034608).
- Riehle, Richard L. (August 1992). "PERFORM considered harmful". *Communications of the ACM*. **35** (8): 125–128. doi:10.1145/135226.376106 (https://doi.org/10.1145%2F135226.376106). S2CID 18845698 (https://api.semanticscholar.org/CorpusID:18845698).
- Sammet, Jean E. (May 1961). *A method of combining ALGOL and COBOL*. Papers presented at the 9–11 May 1961, western joint IRE–AIEE–ACM computer conference. ACM. pp. 379–387. doi:10.1145/1460690.1460734 (https://doi.org/10.1145%2F1460690.1460734).
- Sammet, Jean E. (1978a). Wexelblat, Richard L. (ed.). *The early history of COBOL* (https://archive.org/details/historyofprogram0000hist). History of Programming Languages. Academic Press (published 1981). doi:10.1145/800025.1198367 (https://doi.org/10.1145%2F800025.1198367). ISBN 0127450408.
- Sammet, Jean E. (1978b). Wexelblat, Richard L. (ed.). *Transcript of presentation* (https://archive.org/details/historyofprogram0000hist). History of Programming Languages. Academic Press (published 1981). doi:10.1145/800025.1198368 (https://doi.org/10.1145%2F800025.1198368). ISBN 0127450408.
- Sammet, Jean E. (23 July 2004). "COBOL". In Reilly, Edwin D. (ed.). *Concise Encyclopedia of Computer Science* (https://books.google.com/books?id=5Jaa1BVverIC&pg=PA104). Wiley. ISBN 978-0470090954. OCLC 249810423 (https://www.worldcat.org/oclc/249810423).

- Shneiderman, B. (October 1985). "The Relationship Between COBOL and Computer Science". *Annals of the History of Computing*. **7** (4): 348–352. doi:10.1109/MAHC.1985.10041 (https://doi.org/10.1109%2FMAHC.1985.10041). S2CID 1009406 (https://api.semanticscholar.org/CorpusID:1009406).

External links

- COBOL (https://curlie.org/Computers/Programming/Languages/Cobol/) at Curlie
- COBOL Language Standard (https://pubs.opengroup.org/onlinepubs/009680799/toc.pdf) (1991; COBOL-85 with Amendment 1), from The Open Group