= d can be shortened to a > b AND c OR =
d . As a consequence of this English @-@ like syntax , COBOL has over 300 keywords . 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 ! ') . Separators include the space character and commas and semi @-@ colons followed by a space .

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 .

= = = 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 . Until COBOL 2002 , these were :

In COBOL 2002 , Areas A and B were merged and extended to column 255 , and the program name area was removed .

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 .

= = = 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 . 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 .

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 . Method overloading was added in COBOL 2014 .

= = = 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 .

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 .

= = = 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 .

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 :

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 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 .

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 :

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 it VALUE clause . 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 .

= = = = Data types = = = =

Standard COBOL provides the following data types :
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 . 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 .

= = = = = 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 .

= = = = = 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 :
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 .

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

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 :

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 :

= = = 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 . Execution goes down through the procedures of a program until it is terminated . To use procedures as subroutines , the PERFORM verb is used . This transfers control to the specified range of procedures and returns only upon reaching the end .

Unusual control flow can trigger mines , which cause control in performed procedures to return at unexpected times to unexpected locations . Procedures can be reached in three ways : they can be called with PERFORM , jumped to from a GO TO or through execution " falling through " the bottom of an above paragraph . Combinations of these invoke undefined behavior , creating mines . Specifically , mines occur when execution of a range of procedures would cause control flow to go past the last statement of a range of procedures already being performed .

For example , in the code in the adjacent image , a mine is tripped at the end of update @-@ screen when the screen is invalid . When the screen is invalid , control jumps to the fix @-@ screen section , which , when done , performs update @-@ screen . This recursion triggers undefined behavior as there are now two overlapping ranges of procedures being performed . The mine is then triggered upon reaching the end of update @-@ screen and means control could return to one of two locations :

The first PERFORM statement

The PERFORM statement in fix @-@ screen , where it would then " fall @-@ through " into update @-@ screen and return to the first PERFORM statement upon reaching the end .

= = = = Statements = = = =

COBOL 2014 has 47 statements (also called verbs) , 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 :

The PERFORM statement is used to define loops which are executed until a condition is true (not while , unlike 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 . 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) . 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 .

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 .

SET , which has 15 formats : it can modify indices , assign object references and alter table capacities , among other functions .

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

= = = = 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 , but it was deemed obsolete in the COBOL 1985 standard and deleted in 2002 .

= = = Hello , world = = =

A " Hello , world " program in COBOL :

= = = = 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 GNU / 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 . In keeping with COBOL programming of that era , HELLO , WORLD is displayed in all capital letters .
After submitting the JCL , the MVS console displayed :
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 .

= = Criticism and defense = =

= = = Lack of structure = = =

In the 1970s , programmers began moving away from unstructured spaghetti code to the structured programming paradigm . In his letter to an editor in 1975 entitled " How do we tell truths that might hurt ? " which was critical of several of COBOL 's contemporaries , computer scientist and Turing Award recipient Edsger Dijkstra remarked that " The use of COBOL cripples the mind ; its teaching should , therefore , be regarded as a criminal offense . " In his dissenting response to Dijkstra 's

article and the above " offensive statement , " computer scientist Howard E. Tompkins defended structured COBOL : " COBOL programs with convoluted control flow indeed tend to ' cripple the mind ' , " but this was because " There are too many such business application programs written by programmers that have never had the benefit of structured COBOL taught well ... "

One cause of spaghetti code was the GO TO statement . Attempts to remove GO TOs from COBOL code , however , resulted in convoluted programs and reduced code quality . GO TOs were largely replaced by the PERFORM statement and procedures , which promoted modular programming 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 .

COBOL programs were infamous for being monolithic and lacking modularization . 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 . 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 .

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 . 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 .

= = = Compatibility issues = = =

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

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 . 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 .

= = = Verbose syntax = = =

COBOL syntax has often been criticized for its verbosity . Proponents say that this was intended to make the code self @-@ documenting , easing program maintenance . COBOL was also intended to be easy for programmers to learn and use , while still being readable to non @-@ technical staff such as managers . 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 and the main changes in COBOL @-@ 85 were there to help ease maintenance .

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 .

= = = 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 . 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 . The COBOL specification used a unique " notation " , or metalanguage , to define its syntax rather than the new Backus ? Naur form because few committee members had heard of it . This resulted in " severe " criticism .

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) . 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 . University professors taught more modern , state @-@ of @-@ the @-@ art languages and techniques instead of COBOL which was said to have a " trade school " nature . 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 " . 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 . In contrast , in 2003 , COBOL featured in 80 % of information systems curricula in the United States , the same proportion as C + + and Java .

= = = 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 . " 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 .

COBOL standards have repeatedly suffered from delays : COBOL @-@ 85 arrived five years later than hoped , COBOL 2002 was five years late , and COBOL 2014 was six years late . 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 .

= = = 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 .

COBOL 's COPY facility , although considered " primitive " , influenced the development of include directives .

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 . Additionally , the well @-@ defined division structure restricts the definition of external references to the Environment Division , which simplifies platform changes in particular .