NAME

gcobol — GCC COBOL Front-end

SYNOPSIS

gcobol [−D*name*[=*value*]] [−E] [−fdefaultbyte=*value*] [−fsyntax-only]
[−I*copybook-path*] [−fmax-errors=*nerror*] [−nomain | −main *filename* |
−main=*filename* −main=*filename:program-id*]
[−fcobol-exceptions *exception*[,*exception*...]] [−copyext *ext*]
[−ffixed-form | −ffree-form] [−findicator-column]
[−finternal-ebcdic] [−dialect *dialect-name*] [−include *filename*]
[−preprocess *preprocess-filter*] [−fflex-debug] [−fyacc-debug]
filename [...]

DESCRIPTION

gcobol compiles COBOL source code to object code, and optionally produces an executable binary or shared object. As a GCC component, it accepts all options that affect code-generation and linking. Options specific to COBOL are listed below.

−main *filename*
: **gcobol** will generate a **main**() function as an entry point calling the first PROGRAM-ID in *filename*.

  −main is the default. When none of−nomain, −c, or −shared, is present, an implicit −main is inserted into the command line ahead of the first source file name.

−main=*filename*
: The .o object module for *filename* will include a **main**() entry point calling the first PROGRAM-ID in *filename*

−main=*filename:program-id*
: The .o object module for *filename* will include a **main**() entry point that calls the *program-id* entry point

−nomain
: No **main**() entry point will be generated by this compilation. The −nomain option is incompatible with −main, and is implied by −shared. It is also implied by−c when there is no −main present.

  See below for examples showing the use of −main and −nomain.

−D *name*[=*expr*]
: Define a CDF name (for use with >>**IF**) to have the value of *expr*.

−E
: Write the CDF-processed COBOL input to standard output in *free-form reference format*. Certain non-COBOL markers are included in the output to indicate where copybook files were included. For line-number consistency with the input, blank lines are retained.

  Unlike the C compiler, This option does not prevent compilation. To prevent compilation, use the option

  −fsyntax-only

  also.

−fdefaultbyte=*value*
: Use *value*, a number between 0 and 255, as the default value for all WORKING-STORAGE data items that have no VALUE clause. By default, alphanumeric data items are initialized with blanks, and numeric data items are initialized to zero. This option overrides the default with *value*.

−fsyntax-only
: Invoke only the parser. Check the code for syntax errors, but don't do anything beyond that.

`-copyext` *`ext`*

For the CDF directive

COPY *`name`*

if *`name`* is unquoted, several varieties of *`name`* are tried, as described below under *Copybooks*. The `-copyext` option extends the names searched to include *`ext`*. If*`ext`* is all uppercase or all lowercase, both forms are tried, with preference given to the one supplied. If *`ext`* is mixed-case, only that version is tried. For example, with

`-copyext .abc`

given the CDF directive

COPY name

gcobol will add to possible names searched `name.abc` and `name.ABC` in that order.

`-ffixed-form`

Use strict *fixed-form reference format* in reading the COBOL input: 72-character lines, with a 6-character sequence area, and an indicator column. Data past column 72 are ignored.

`-ffree-form`

Force the COBOL input to be interpreted as *free-form reference format*. Line breaks are insignificant, except that '*' at the start of a line acts as a comment marker. Equivalent to `-indicator-column` *`0`*.

`-findicator-column`

describes the location of the Indicator Area in a COBOL file in *Reference Format*, where the first 6 columns — known as the "Sequence Number Area" — are ignored, and the 7th column — the Indicator Area — may hold a character of significance to the compiler.

Although *reference format*, strictly speaking, ignores data after column 72, with this option **gcobol** accepts long COBOL lines, sometimes known as *extended source format*. Text past column 72 is treated as ordinary COBOL text. (Line continuation remains in effect, however, provided no text appears *past* column 72.)

There is no maximum line length. Regardless of source code format, the entire program could appear on one line.

By default, **gcobol** auto-detects the source code format by examining the line that contains the text "program-id". When there are characters on past column 72 on that line, the file is assumed to be in *extended source format*, with the indicator area in column 7. Otherwise, columns 1-6 are examined. If those characters are all digits or blanks, the file is assumed to be in *fixed-form reference format*, also with the indicator in column 7. If not auto-detected as *fixed-form reference format* or *extended source format*, the file is assumed to be in *free-form reference format*.

`-fcobol-exceptions` *`exception`* [,*`exception`*...]

By default, no exception condition is enabled (including fatal ones), and by the ISO standard exception conditions are enabled only via the CDF **TURN** directive. This option enables one or more exception conditions by default, as though **TURN** had appeared at the top of the first source code file. This option may also appear more than once on the command line.

The value of *`exception`* is a Level 1, 2, or 3 exception condition name, as described by ISO/IEC 1989:2023. `EC-ALL` means enable all exceptions.

The `-fno-cobol-exceptions` form turns off *`exception`*, just as though

>>TURN *`exception CHECKING OFF`*

had appeared.

Not all exception conditions are implemented. Any that are not produce a warning message.

`-fmax-errors=`*`nerror`*

`nerror` represents the number of error messages produced. Without this option, **gcobol** attempts to recover from a syntax error by resuming compilation at the next statement, continuing until end-of-file. With it, **gcobol** counts the messages as they're produced, and stops when

nerror is reached.

`-fstatic-call`, `-fno-static-call`

With `-fno-static-call`, **gcobol** never uses static linking for

CALL *program*

By default, or with `-fstatic-call`, if *program* is an alphanumeric literal, **gcobol** uses static linkage, meaning the compiler produces an external symbol *program* for the linker to resolve. (In the future, that will work with **CONSTANT** data items, too.) With static linkage, if *program* is not supplied by the source code module or another object file or library at build time, the linker will produce an "unresolved symbol" error. With `-fno-static-call`, **gcobol** always uses dynamic linking.

This option affects the **CALL** statement for literals only. If*program* is a non-constant data item, it is always resolved using dynamic linking, with *dlsym*(3), because its value is determined at run time.

`-dialect` *dialect-name*

By default, **gcobol** accepts COBOL syntax as defined by ISO/IEC 1989:2023, with some extensions for backward compatibility with COBOL-85. To make the compiler more generally useful, some additional syntax is supported by this option.

The value of *dialect-name* may be

ibm to indicate IBM COBOL 6.3 syntax, specifically
STOP <number>.

gnu to indicate GnuCOBOL syntax

mf to indicate MicroFocus syntax, specifically **LEVEL 78** constants.

Only a few such non-standard constructs are accepted, and **gcobol** makes no claim to emulate other compilers. But to the extent that a feature is popular but nonstandard, this option provides a way to support it, or add it.

`-include` *filename*

Process *filename* as if

COPY "*filename*"

appeared as the first line of the primary source file. If *filename* is not an absolute path, the directory searched is the current working directory, not the directory containing the main source file. The name is used verbatim. No permutations are applied, and no directories searched.

If multiple `-include` options are given, the files are included in the order they appear on the command line.

`-preprocess` *preprocess-filter*

After all CDF text-manipulation has been applied, and before the prepared COBOL is sent to the **cobol1** compiler, the input may be further altered by one or more filters. In the tradition of *sed*(1), each *preprocess-filter* reads from standard input and writes to standard output.

To supply options to *preprocess-filter*, use a comma-separated string, similar to how linker options are supplied to **-Wl**. (Do not put any spaces after the commas, because the shell will treat it as an option separator.) **gcobol** replaces each comma with a space when *preprocess-filter* is invoked. For example,

```
-preprocess tee,output.cbl
```

invokes *tee*(1) with the output filename argument *output.cbl*, causing a copy of the input to be written to the file.

gcobol searches the current working directory and the PATH environment variable directories for an executable file whose name matches *preprocess-filter*. The first one found is used. If none is found, an error is reported and the compiler is not invoked.

The `-preprocess` option may appear more than once on the command line. Each *preprocess-filter* is applied in turn, in order of appearance.

The *preprocess-filter* should return a zero exit status, indicating success. If it returns a nonzero exit status, an error is reported and the compiler is not invoked.

`-fflex-debug`, `-fyacc-debug`
produce messages useful for compiler development. The `-fflex-debug` option prints the tokenized input stream. The `-fyacc-debug` option shows the shift and reduce actions taken by the parser.

COMPILATION SCENARIOS

gcobol *xyz.cob*
gcobol -main *xyz.cob*
gcobol -main=*xyz.cob xyz.cob*

These are equivalent. The *xyz.cob* code is compiled and a **main**() function is inserted that calls the first PROGRAM-ID in the *xyz.cob* source file.

gcobol -nomain *xyz.cob elsewhere.o*

The `-nomain` option prevents a **main**() function from being generated by the gcobol compiler. A **main**() entry point must be present in the *elsewhere.o* module; without it the linker will report a "missing main" error.

gcobol *aaa.cob bbb.cob ccc.cob*
gcobol -main *aaa.cob bbb.cob ccc.cob*

The two commands are equivalent. The three source code modules are compiled and linked together along with a generated **main**() function that calls the first PROGRAM-ID in the *aaa.cob* module.

gcobol *aaa.cob bbb.cob* `-main` *ccc.cob*
gcobol -main=*ccc.cob aaa.cob bbb.cob ccc.cob*

These two commands have the same result: An *a.out* executable is created that starts executing at the first PROGRAM-ID in *ccc.cob*.

gcobol -main=*bbb.cob:b-entry aaa.cob bbb.cob ccc.cob*

An *a.out* executable is created that starts executing at the PROGRAM-ID *b-entry*.

gcobol -c *aaa.cob*
gcobol -c -main *bbb.cob*
gcobol -c *ccc.cob*
gcobol *aaa.o bbb.o ccc.o*

The first three commands each create a .o file. The *bbb.o* file will contain a **main**() entry point that calls the first PROGRAM-ID in *bbb*. The fourth links the three .o files into an*a.out*.

EBCDIC

The `-finternal-ebcdic` option is useful when working with mainframe COBOL programs intended for EBCDIC-encoded files. With this option, while the COBOL text remains in ASCII, the character literals and field initial values produce EBCDIC strings in the compiled binary, and any character data read from a file are interpreted as EBCDIC data. The file data are not *converted*; rather, the file is assumed to use EBCDIC representation. String literals in the COBOL text *are* converted, so that they can be compared meaningfully with data in the file.

Only file data and character literals are affected. Data read from and written to the environment, or taken from the command line, are interpreted according the *locale*(7) in force during execution. The same is true of **ACCEPT** and **DISPLAY**. Names known to the operating system, such as file names and the names of environment variables, are processed verbatim.

At the present time, this is an all-or-nothing setting. Support for **USAGE** and **CODESET**, which would allow conversion between encodings, remains a future goal.

See also "Feature-set Variables", below.

REDEFINES ... USAGE POINTER

Per ISO, an item that **REDEFINES** another may not be larger than the item it redefines, unless that item has LEVEL 01 and is not EXTERNAL. In **gcobol**, using `-dialect` *ibm*, this rule is relaxed for **REDEFINES** with **USAGE POINTER** whose redefined member is a 4-byte **USAGE COMP-5** (usually

PIC S9(8)), or vice-versa. In that case, the redefined member is re-sized to be 8 bytes, to accommodate the pointer. This feature allows pointer arithmetic on a 64-bit system with source code targeted at a 32-bit system.

See also "Feature-set Variables", below.

IMPLEMENTATION NOTES

gcobol is a gcc compiler, and follows gcc conventions where applicable. Sometimes those conventions (and user expectations) conflict with common Mainframe practice. Unless required of the compiler by the ISO specification, any such conflicts are resolved in favor of gcc.

Linking

Unlike, C, the COBOL **CALL** statement implies dynamic linking, because for

CALL *program*

program can be a variable whose value is determined at runtime. However, the parameter may also be compile-time constant, either an alphanumeric literal, or a **CONSTANT** data item.

gcobol supports static linking where possible, unless defeated by `-fno-static-call`. If the parameter value is known at compile time, the compiler produces an external reference to be resolved by the linker. The referenced program is normally supplied via an object module, a static library, or a shared object. If it is not supplied, the linker will report an "unresolved symbol" error, either at build time or, if using a shared object, when the program is executed. This feature informs the programmer of the error at the earliest opportunity.

Programs that are expected to execute correctly in the presence of an unresolved symbol (perhaps because the program logic won't require that particular **CALL**) can use the `-no-static-call` option. That forces all **CALL** statements to be resolved dynamically, at runtime.

Implemented Exception Conditions

By default, per ISO, no EC is enabled. Implemented ECs may be enabled on the command line or via the **TURN** directive. Any attempt to enable an EC that is not implemented is treated as an error.

An enabled EC not handled by a **DECLARATIVE** is written to the system log and to standard error. (The authors intend to make that an option.) A fatal EC not handled with **RESUME** ends with a call to *abort*(3) and process termination.

Not all Exception Conditions are implemented. Any attempt to enable an EC that that is not implemented produces a warning message. The following are implemented:

EC-FUNCTION-ARGUMENT
: for the following functions:
  ACOS
  ANNUITY
  ASIN
  LOG
  LOG10
  PRESENT-VALUE
  SQRT

EC-SORT-MERGE-FILE-OPEN

EC-BOUND-SUBSCRIPT
: subscript not an integer, less than 1, or greater than occurs

EC-BOUND-REF-MOD
: refmod start not an integer, start less than 1, start greater than variable size, length not an integer, length less than 1, and start+length exceeds variable size

EC-BOUND-ODO
: DEPENDING not an integer, greater than occurs upper limit, less than occurs lower limit, and subscript greater than DEPENDING for sending item

EC-SIZE-ZERO-DIVIDE
for both fixed-point and floating-point division
EC-SIZE-TRUNCATION
EC-SIZE-EXPONENTIATION

As of this writing, no COBOL compiler documents a complete implementation of ISO/IEC 1989:2023 Exception Conditions. **gcobol** will give priority to those ECs that the user community deems most valuable.

EXTENSIONS TO ISO COBOL

Standard COBOL has no provision for environment variables as defined by Unix and Windows, or command-line arguments. **gcobol** supports them using syntax similar to that of GnuCOBOL. ISO and IBM also define incompatible ways to return the program's exit status to the operating system. **gcobol** supports IBM syntax.

Environment Variables

To read an environment variable:

ACCEPT *target* FROM ENVIRONMENT *envar*

where *target* is a data item defined in **DATA DIVISION**, and *envar* names an environment variable. *envar* may be a string literal or alphanumeric data item whose value is the name of an environment variable. The value of the named environment variable is moved to *target*. The rules are the same as for **MOVE**.

To write an environment variable:

SET ENVIRONMENT *envar* TO *source*

where *source* is a data item defined in **DATA DIVISION**, and *envar* names an environment variable. *envar* again may be a string literal or alphanumeric data item whose value is the name of an environment variable. The value of the named environment variable is set to the value of *source*.

Command-line Arguments

To read command-line arguments, use the registers **COMMAND-LINE** and **COMMAND-LINE-COUNT** in an **ACCEPT** statement (only). Used without a subscript, **COMMAND-LINE** returns the whole command line as a single string. With a subscript, **COMMAND-LINE** is a table of command-line arguments. For example, if the program is invoked as

./program −i *input output*

then

ACCEPT target FROM COMMAND-LINE(3)

moves *input* into *target*. The program name is the first thing in the whole command line and is found in COMMAND-LINE(1) **COMMAND-LINE** table.

To discover how many arguments were provided on the command line, use

ACCEPT *target* FROM COMMAND-LINE-COUNT

If **ACCEPT** refers to a nonexistent environment variable or command-line argument, the target is set to **LOW-VALUES**.

The system command line parameters can also be accessed through the LINKAGE SECTION in the program where execution starts. The data structure looks like this:

```
linkage         section.
01   argc       pic 999.
01   argv.
 02  argv-table   occurs 1 to 100 times depending on argc.
  03 argv-element pointer.
```

```
01   argv-string  pic x(100) .
```

and the code to access the third parameter looks like this

```
procedure division using by value argc by reference argv.
set address of argv-string to argv-element(3)
display argv-string
```

#line directive

The parser accepts lines in the form

#line *lineno* "*filename*".

The effect is to set the current line number to *lineno* and the current input filename to *filename*. Preprocessors may use this directive to control the filename and line numbers reported in error messages and in the debugger.

SELECT ... ASSIGN TO

In the phrase

ASSIGN TO *filename*

filename may appear in quotes or not. If quoted, it represents a filename as known to the operating system. If unquoted, it names either a data element or an environment variable containing the name of a file. If *filename* matches the name of a data element, that element is used. If not, resolution of *filename* is deferred until runtime, when the name must appear in the program's environment.

ISO COBOL Implementation Status

USAGE Data Types

gcobol supports the following **USAGE IS** clauses:

INDEX for use as an index in a table.

POINTER for variables whose value is the address of an external function, **PROGRAM-ID**, or data item. Assignment is via the **SET** statement.

BINARY, COMP, **COMPUTATIONAL, COMP-4, COMPUTATIONAL-4**
big-endian integer, 1 to 16 bytes, per PICTURE.

COMP-1, **COMPUTATIONAL-1**, **FLOAT-BINARY-32**
IEEE 754 single-precision (4-byte) floating point, as provided by the hardware.

COMP-2, **COMPUTATIONAL-2**, **FLOAT-BINARY-64**
IEEE 754 double-precision (8-byte) floating point, as provided by the hardware.

COMP-3, **COMPUTATIONAL-3, PACKED-DECIMAL**
currently unimplemented.

COMP-5, **COMPUTATIONAL-5**
little-endian integer, 1 to 16 bytes, per **PICTURE.**

FLOAT-BINARY-128, **FLOAT-EXTENDED**
implements 128-bit floating point, per IEEE 754.

gcobol supports ISO integer **BINARY-<type>** types, most of which alias **COMP-5.**

LB LB LB LB LB LB LB LB L L L L . COMP-5 Compatible Picture BINARY Type Bytes Value T{ BINARY-CHAR [UNSIGNED] T} 1 0 — 256 S9(1...4) T{ BINARY-CHAR SIGNED T} 1 -128 — +127 9(1...4) T{ BINARY-SHORT [UNSIGNED] T} 2 0 — 65535 S9(1...4) T{ BINARY-SHORT SIGNED T} 2 -32768 — +32767 9(5...9) T{ BINARY-LONG [UNSIGNED] T} 4 0 — 4,294,967,295 S9(5...9) T{ BINARY-LONG SIGNED T} 4 T{ -2,147,483,648 — +2,147,483,647 T} 9(10...18) T{ BINARY-LONG-LONG [UNSIGNED] T} 8 T{ 0 — 18,446,744,073,709,551,615 T} S9(10...18) T{ BINARY-LONG-LONG SIGNED T} 8 T{ -9,223,372,036,854,775,808 — +9,223,372,036,854,775,807 T}

These define a size (in bytes) and cannot be used with a **PICTURE** clause. Per the ISO standard, **SIGNED** is the default for the **BINARY-***type* aliases.

All computation — both integer and floating point — is done using 128-bit intermediate forms.

Environment Names

In **gcobol**

```
DISPLAY UPON
```

maps **SYSOUT** and **STDOUT** to standard output, and **SYSPUNCH**, **SYSPCH** and **STDERR** to standard error.

Exit Status

gcobol supports the ISO syntax for returning an exit status to the operating system,

STOP RUN [WITH] {NORMAL | ERROR} [STATUS] *status*

In addition, **gcobol** also supports the IBM syntax for returning an exit status to the operating system. Use the **RETURN-CODE** register:

```
MOVE ZERO TO RETURN-CODE.
GOBACK.
```

The **RETURN-CODE** register is defined as a 4-byte binary integer.

COMPILER-DIRECTING FACILITY

The CDF should be used with caution because no comprehensive test suite has been identified.

CDF Text Manipulation

COPY *copybook* [OF|BY *library*] [**REPLACING ...**]

If *copybook* is a literal, it treated a literal filename, which either does or does not exist. If *copybook* is a COBOL word, **gcobol** looks first for an environment variable named *copybook* and, if found, uses the contents of that variable as the name of the copybook file. If that file does not exist, it continues looking for a file named one of:

- *copybook* (literally)
- *copybook.cpy*
- *copybook.CPY*
- *copybook.cbl*
- *copybook.CBL*
- *copybook.cob*
- *copybook.COB*

in that order. It looks first in the same directory as the source code file, and then in any *copybook-path* named with the `-I` option. *copybook-path* may (like the shell's `PATH` variable) be a colon-separated list. The `-I` option may occur multiple times on the command line. Each successive *copybook-path* is concatenated to previous ones. Relative paths (having no leading '`/`') are searched relative to the compiler's current working directory.

For example,

```
-I /usr/local/include:include
```

searches first the directory where the COBOL program is found, next in */usr/local/include*, and finally in an *include* subdirectory of the directory from which **gcobol** was invoked.

For the **REPLACING** phrase, both the modern pseudo-text and the COBOL/85 forms are recognized. (The older forms are used in the NIST CCVS/85 test suite.)

REPLACE ...

gcobol supports the full ISO **REPLACE** syntax.

CDF Directives

>>**DEFINE** *name* **AS** {*expression* | **PARAMETER**} [**OVERRIDE**]

Define *name* as a compilation variable to have the value *expression*. If*name* w as previously defined, **OVERRIDE** is required, else the directive is invalid. **AS PARAMETER** is accepted, but has no effect in **gcobol**.

>>**DEFINE** *name AS* **OFF**

releases the definition *name*, making it subsequently invalid for use.

>>**IF** *cce text* [>>**ELSE** *alt-text*] **>>END-IF**

evaluates *cce*, a *constant conditional expression*, for conditional compilation. If a name, *cce* may be defined with the **-D** command-line parameter. If true, the COBOL text *text* is compiled. If false, *else-text*, if present, is compiled. [**IS** [**NOT**]] **DEFINED** is supported. Boolean literals are not supported.

>>**EVALUATE**

Not implemented.

>>**CALL-CONVENTION** *convention*

convention may be one of:

COBOL

Use standard COBOL case-insensitive symbol-name matching. For **CALL** "*name*", *name* is rendered by the compiler in lowercase.

C Use case-sensitive symbol-name matching. The **CALL** target is not changed in any way; it is used verbatim.

VERBATIM

An alias for >>**CALL-CONVENTION C**.

>>**COBOL-WORDS EQUATE** *keyword* **WITH** *alias*

makes *alias* a synonym for *keyword*.

>>**COBOL-WORDS UNDEFINE** *keyword*

keyword is removed from the COBOL grammar. Use of it in a program will provoke a syntax error from the compiler.

>>**COBOL-WORDS SUBSTITUTE** *keyword* **BY** *new-word*

keyword is deleted as a keyword from the grammar, replaced by *new-word*. *keyword* may thereafter be used as a user-defined word.

>>**COBOL-WORDS RESERVE** *new-word*

Treat *new-word* as a COBOL keyword. It cannot be used by the program, either as a keyword or as a user-defined word.

>>**DISPLAY** *string ...*

Write *string* to standard error as a warning message.

>>**SOURCE** *format*

format may be one of:

FIXED

Source conforms to COBOL *fixed-form reference format* with unlimited line length.

FREE Source conforms to COBOL *free-form reference format*. '*' at the beginning of a line is recognized as a comment.

>>**FLAG-02** Not implemented.

>>**FLAG-85** Not implemented.

>>**FLAG-NATIVE-ARITHMETIC**
Not implemented.
>>**LEAP-SECOND**
Not implemented.
>>**LISTING** Not implemented.
>>**PAGE** Not implemented.
>>**PROPAGATE**
Not implemented.
>>**PUSH** *directive*
>>**POP** *directive*
With **PUSH**, push CDF state onto a stack. With **POP**, return to the prior pushed state. *directive* may be one of
CALL-CONVENTION
COBOL-WORDS
DEFINE
SOURCE FORMAT
TURN
>>**TURN** [*ec* [*file* . . .] …]**CHECKING** {[**ON**] [[**WITH**] **LOCA TION**] | **OFF**}
Enable (or, with **OFF**, disable) exception condition *ec* optionally associated with the file connectors *file*. If**LOCA TION** is specified, **gcobol** reports at runtime the source filename and line number of the statement that triggered the exception condition.

Feature-set Variables

Some command-line options affect CDF *feature-set* variables that are special to **gcobol**. They can be set and tested using >>**DEFINE** and >>**IF**, and are distinguished by a leading '%' in the name, which is otherwise invalid in a COBOL identifier:

%EBCDIC-MODE
is set by `-finternal-ebcdic`.
%64-BIT-POINTER
is implied by `-dialect ibm`.

To set a feature-set variable, use

```
>>SET feature [AS] {ON | OFF}
```

If *feature* is **%EBCDIC-MODE**, the directive must appear before **PROGRAM-ID**.

To test a feature-set variable, use

```
>>IF feature DEFINED
```

Intrinsic functions

gcobol implements all intrinsic functions defined by ISO/IEC 1989:2023, plus a few others. They are listed alphabetically below.

ABS ACOS ANNUITY ASIN ATAN
BASECONVERT BIT-OF BIT-TO-CHAR BOOLEAN-OF-INTEGER BYTE-LENGTH
CHAR CHAR-NATIONAL COMBINED-DATETIME CONCAT CONVERT COS CURRENT-DATE
DATE-OF-INTEGER DATE-TO-YYYYMMDD DAY-OF-INTEGER DAY-TO-YYYYDDD DISPLAY-OF
E EXCEPTION-FILE EXCEPTION-FILE-N EXCEPTION-LOCATION EXCEPTION-LOCATION-N
EXCEPTION-STATEMENT EXCEPTION-STATUS EXP EXP10
FACTORIAL FIND-STRING FORMATTED-CURRENT-DATE FORMATTED-DATE FORMATTED-
DATETIME FORMATTED-TIME FRACTION-PART
HEX-OF HEX-TO-CHAR HIGHEST-ALGEBRAIC
INTEGER INTEGER-OF-BOOLEAN INTEGER-OF-DATE INTEGER-OF-DAY INTEGER-OF-FOR-
MATTED-DATE INTEGER-PART
LENGTH LOCALE-COMPARE LOCALE-DATE LOCALE-TIME LOCALE-TIME-FROM-SECONDS
LOG LOG10 LOWER-CASE LOWEST-ALGEBRAIC

MAX MEAN MEDIAN MIDRANGE MIN MOD MODULE-NAME
NATIONAL-OF NUMVAL NUMVAL-C NUMVAL-F ORD
ORD-MAX ORD-MIN
PI PRESENT-VALUE
RANDOM RANGE REM REVERSE
SECONDS-FROM-FORMATTED-TIME SECONDS-PAST-MIDNIGHT SIGN SIN SMALLEST-ALGEBRAIC SQRT STANDARD-COMPARE STANDARD-DEVIATION SUBSTITUTE SUM
TAN TEST-DATE-YYYYMMDD TEST-DAY-YYYYDDD TEST-FORMATTED-DATETIME TEST-NUMVAL TEST-NUMVAL-C TEST-NUMVAL-F TRIM
ULENGTH UPOS UPPER-CASE USUBSTR USUPPLEMENTARY UUID4 UVALID UWIDTH
VARIANCE
WHEN-COMPILED
YEAR-TO-YYYY

Binary floating point DISPLAY

How the DISPLAY presents binary floating point numbers depends on the value.

When a value has six or fewer decimal digits to the left of the decimal point, it is expressed as *123456.789....*

When a value is less than 1 and has no more than three zeroes to the right of the decimal point, it is expressed as *0.0001234....*

Otherwise, exponential notation is used: *1.23456E+7*.

In all cases, trailing zeroes on the right of the number are removed from the displayed value.

COMP-1 displayed with 9 decimal digits.
COMP-2 displayed with 17 decimal digits.
FLOAT-EXTENDED
displayed with 36 decimal digits.

Those digit counts are consistent with the IEEE 754 requirements for information interchange. As one example, the description for COMP-2 binary64 values (per Wikipedia).

If an IEEE 754 double-precision number is converted to a decimal string with at least 17 significant digits, and then converted back to double-precision representation, the final result must match the original number.

17 digits was chosen so that the **DISPLAY** statement shows the contents of a COMP-2 variable without hiding any information.

Binary floating point MOVE

During a **MOVE** statement, a floating-point value may be truncated. It will not be unusual for Numeric Display values to be altered when moved through a floating-point value.

This program:

```
 01 PICV999 PIC 9999V999.
 01 COMP2 COMP-2.
PROCEDURE DIVISION.
 MOVE 1.001 to PICV999
 MOVE PICV999 TO COMP2
 DISPLAY "The result of MOVE " PICV999 " TO COMP2 is   " COMP2
 MOVE COMP2 to PICV999
 DISPLAY "The result of MOVE COMP2    TO PICV999 is " PICV999
```

generates this result:

```
The result of MOVE 0001.001 TO COMP2 is   1.00099999999999989
The result of MOVE COMP2    TO PICV999 is 0001.000
```

However, the internal implementation can produce results that might be seem surprising:

```
The result of MOVE 0055.110 TO COMP2 is   55.1099999999999994
The result of MOVE COMP2    TO PICV999 is 0055.110
```

The source of this inconsistency is the way **gcobol** stores and converts numbers. Converting the floating-point value to the numeric display value 0055110 is done by multiplying 55.109999... by 1,000 and then truncating the result to an integer. And it turns out that even though 55.11 can't be represented in floating-point as an exact value, the product of the multiplication, 55110, is an exact value.

In cases where it is important for conversions to have predictable results, we need to be able to apply rounding, which can be done with an arithmetic statement:

```
MOVE 1.001 to PICV999
MOVE PICV999 TO COMP2
DISPLAY "The result of MOVE " PICV999 " TO COMP2 is   " COMP2
MOVE COMP2 to PICV999
DISPLAY "The result of MOVE COMP2    TO PICV999 is " PICV999
ADD COMP2 to ZERO GIVING PICV999 ROUNDED
DISPLAY "The result of ADD COMP2 to ZERO GIVING PICV999 ROUNDED is " PICV999

The result of MOVE 0001.001 TO COMP2 is   1.00099999999999989
The result of MOVE COMP2    TO PICV999 is 0001.000
The result of ADD COMP2 to ZERO GIVING PICV999 ROUNDED is 0001.001
```

Binary floating point computation

gcobol attempts to do internal computations using binary integers when possible. Thus, simple arithmetic between binary values and numeric display values conclude with binary intermediate results.

If a floating-point value gets included in the mix of variables specified for a calculation, then the intermediate result becomes a 128-bit floating-point value.

A warning about binary floating point comparison

The cardinal rule when doing comparisons involving floating-point values: Never, ever, test for equality. It's just not worth the hassle.

For example:

```
WORKING-STORAGE SECTION.
 01 COMP1 COMP-1 VALUE 555.11.
 01 COMP2 COMP-2 VALUE 555.11.
PROCEDURE DIVISION.
 DISPLAY "COMPARE " COMP1 " with " COMP2
 IF COMP1 EQUAL COMP2 DISPLAY "Equal" ELSE DISPLAY "Not equal" END-IF

 MOVE COMP1 to COMP2
 DISPLAY "COMPARE " COMP1 " with " COMP2
 IF COMP1 EQUAL COMP2 DISPLAY "Equal" ELSE DISPLAY "Not equal" END-IF
```

the results:

```
COMPARE 555.1099854 with 555.110000000000014
Not equal
COMPARE 555.1099854 with 555.1099853515625
Equal
```

Why? Again, it has to do with the internals of **gcobol**. When differently sized floating-point values need to be compared, they are first converted to 128-bit floats. And it turns out that when a COMP1 is moved to a COMP2, and they are both converted to FLOAT-EXTENDED, the two resulting values are (probably) equal.

Avoid testing for equality unless you really know what you are doing and you really test the code. And then avoid it anyway.

Finally, it is observably the case that the **gcobol** implementations of floating-point conversions and comparisons don't precisely match the behavior of other COBOL compilers.

You have been warned.

ENVIRONMENT

`COBPATH` If defined, specifies the directory paths to be used by the **gcobol** runtime library, *libgcobol.so*, to locate shared objects. Like `LD_LIBRARY_PATH`, it may contain several directory names separated by a colon ('`:`'). `COBPATH` is searched first, followed by `LD_LIBRARY_PATH`.

Each directory is searched for files whose name ends in `.so`. For each such file, *dlopen*(3) is attempted, and, if successful *dlsym*(3). No relationship is defined between the symbol's name and the filename.

Without `COBPATH`, binaries produced by **gcobol** behave as one might expect of any program compiled with gcc. Any shared objects needed by the program are mentioned on the command line with a `-l`*`library`* option, and are found by following the executable's *RPATH* or otherwise per the configuration of the runtime linker, *ld.so*(8).

`UPSI` COBOL defines a User Programmable Status Indicator (UPSI) switch. In **gcobol**, the settings are denoted **UPSI-0** through **UPSI-7**, where 0-7 indicates a bit position. The value of the UPSI switches is taken from the `UPSI` environment variable, whose value is a string of up to eight 1's and 0's. The first character represents the value of **UPSI-0**, and missing values are assigned 0. For example, **UPSI=1000011** in the environment sets bits 0, 5, and 6 on, which means that **UPSI-0**, **UPSI-5**, and **UPSI-6** are on.

`GCOBOL_TEMPDIR`
causes any temporary files created during CDF processing to be written to a file whose name is specified in the value of `GCOBOL_TEMPDIR`. If the value is just "/", the effect is different: each copybook read is reported on standard error. This feature is meant to help diagnose mysterious copybook errors.

FILES

Executables produced by **gcobol** require the runtime support library *libgcobol*, which is provided both as a static library and as a shared object.

COMPATIBILITY

The ISO standard leaves the default file organization up to the implementation; in **gcobol**, the default is **SEQUENTIAL**.

On-Disk Format

Any ability to use files produced by other COBOL compilers, or for those compilers to use files produced by **gcobol**, is the product of luck and intuition. Various compilers interpret the ISO standard differently, and the standard's text is not always definitive.

For **ORGANIZATION IS LINE SEQUENTIAL** files (explicitly or by default), **gcobol**, absent specific direction, produces an ordinary Linux text file: for each WRITE, the data are written, followed by an ASCII NL (hex 0A) character. On READ, the record is read up to the size of the specified record or NL, whichever comes first. The NL is not included in the data brought into the record buffer; it serves only as an on-disk record-termination marker. Consequently, **SEQUENTIAL** and **LINE SEQUENTIAL** files work the same way: the COBOL program never sees the record terminator.

When **READ** and **WRITE** are used with **ADVANCING**, however, the game changes. If **ADVANCING** is used with **LINE SEQUENTIAL** files, it is honored by **gcobol**.

Other compilers may not do likewise. According to ISO, in**WRITE** (14.9.47.3 General rules) **ADVANCING** is *ignored* for files for which "the physical file does not support vertical positioning". It further states that, in the absence of **ADVANCING**, **WRITE** proceeds as if "as if the user has specified AFTER ADVANCING 1 LINE". Some other implementations interpret that to mean that the first **WRITE**

to a **LINE SEQUENTIAL** file results in a leading NL on the first line, and no trailing NL on the last line. Some furthermore *prohibit* the use of **ADVANCING** with **LINE SEQUENTIAL** files.

STANDARDS

The reference standard for **gcobol** is ISO/IEC 1989:2023.

- If **gcobol** compiles code consistent with that standard, the resulting program should execute correctly; any other result is a bug.
- If **gcobol** compiles code that does not comply with that standard, but runs correctly according to some other specification, that represents a non-standard extension. One day, the `-pedantic` option will produce diagnostic messages for such code.
- If **gcobol** rejects code consistent with that standard, that represents an aspect of COBOL that is (or is not) on the To Do list. If you would like to see it compile, please get in touch with the developers.

Status of NIST COBOL Compiler Verification Suite

NC 100%	Nucleus
SQ 100%	Sequential I/O
RL 100%	Relative I/O
IX 100%	Indexed I/O
IC 100%	Inter-Program Communication
ST 100%	Sort-Merge
SM 100%	Source Text Manipulation RW \n Report Writer
CM	Communication
DB to do?	Debug
SG	Segmentation
IF 100%	Intrinsic Function

Where **gcobol** passes 100% of the tests in a module, we exclude the (few) tests for obsolete features. The authors regard features that were obsolete in 1985 to be well and truly obsolete today, and did not implement them.

Notable deferred features

CCVS-85 modules not marked with above with any status (CM, and SG) are on the “hard maybe” list, meaning they await an interested party with real code using the feature.

gcobol does not implement Report Writer or Screen Section.

Beyond COBOL/85

gcobol increasingly implements ISO/IEC 1989:2023. For example, **DECLARATIVES** is not tested by CCVS-85, but are implemented by **gcobol**. Similarly, Exception Conditions were not defined in 1985, and **gcobol** contains a growing number of them.

The authors are well aware that a complete, pure COBOL-85 compiler won’t compile most existing COBOL code. Every vendor offered (and offers) extensions, and most environments rely on a variety of preprocessors and ancillary systems defined outside the standard. The express goal of adding an ISO COBOL front-end to GCC is to establish a foundation on which any needed extensions can be built.

HISTORY

COBOL, the language, may well be older than the reader. To the author’s knowledge, free COBOL compilers first began to appear in 2000. Around that time an earlier COBOL for GCC project
cobolforgcc: https://cobolforgcc.sourceforge.net/ met with some success, but was never officially merged into GCC.

This compiler, **gcobol**, was begun by *COBOLworx*: https://www.cobolworx.com/ in the fall of 2021. The project announced a complete implementation of the core language features in December 2022.

AUTHORS

James K. Lowden
(jklowden@cobolworx.com) is responsible for the parser.

Robert Dubner
(rdubner@cobolworx.com) is responsible for producing the GIMPLE tree, which is input to the GCC back-end.

CAVEATS

- **gcobol** has been tested only on x64 and Apple M1 processors running Linux in 64-bit mode.
- The I/O support has not been extensively tested, and does not implement or emulate many features related to VSAM and other mainframe subsystems. While LINE-SEQUENTIAL files are ordinary text files that can be manipulated with standard utilities, INDEXED and RELATIVE files produced by **gcobol** are not compatible with that of any other COBOL compiler. Enhancements to the I/O support will be readily available to the paying customer.