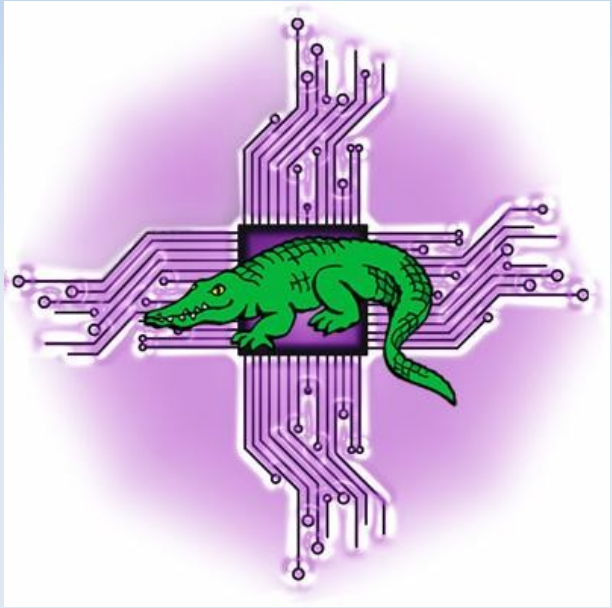
GnuCOBOL: A Gnu Life for an Old Workhorse



Linux Users of Victoria, July 16, 2016

lev@levlafayette.com

A Brief History of COBOL

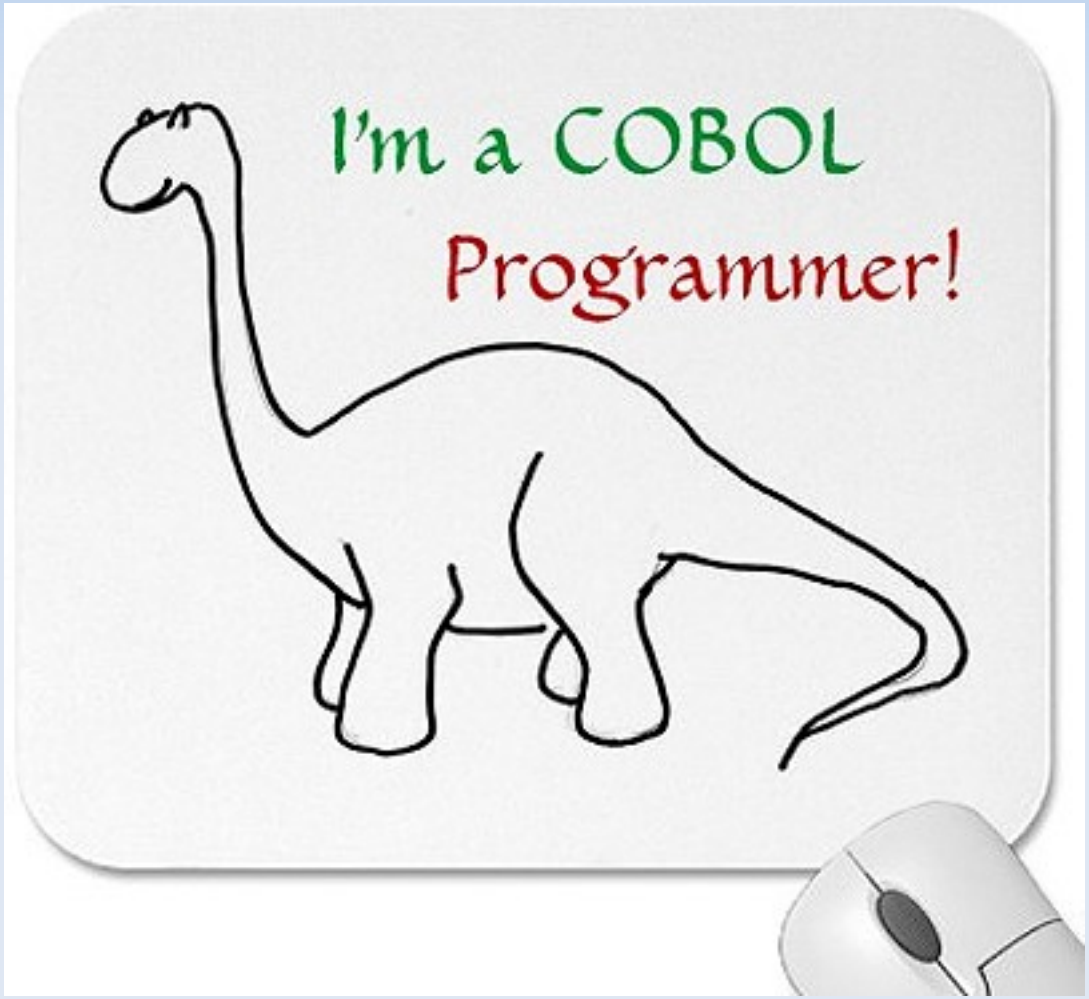
COBOL (COmmon Business-Oriented Language) is a compiled, imperative, procedural, and (in recent versions) object-orientated language. It was first designed in 1959 by the near-legendary Rear Admiral Grace Hooper (first programmer of the Harvard Mark I in 1944, who invented the first compiler (for the A-0 programming language) in 1951, developed first machine-independent program, original developer of COBOL).

Many implementations of COBOL, and several standards; ANSI COBOL 1968 (became ISO in 1972), 1974 (file organisation, segmentation module), 1980 (abandoned after extensive criticism), 1985, 2002 (with object-orientated support), and 2014 (adopted IEEE 754 standard for floating point computation).

Features, Issues, and Opportunities

COBOL has an English-like syntax, making it easy to learn, but it appears verbose. Program structures are expressed in statements rather than blocks, it has no error checking on subroutine calls. It lacks standard libraries (resulting in c300 reserved words), and poor consistency among its many dialects. However it has excellent screen support, decimal arithmetic, and good performance. Modern versions do have built-in XML support and support for OOP.

COBOL is disliked by many because it's a domain-specific language (business, finance, and administration for large companies and government = lots of data crunching), and it was relatively isolated from computer science as a discipline.



Features, Issues, and Opportunities

Not taught extensively at universities. *"The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offense."* (Edsger Dijkstra, 1975). Arguably is the lack of teaching that has generated a lot of poor COBOL code.

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. In 2006 and 2012, Computerworld surveys found that over 60% of organisations used COBOL. Often used as the backend system (e.g., PeopleSoft). It is *still* in the top 20 programming languages on TIOBE. COBOL programmers (mainly from the 60s to the 90s) are retiring and there's a growing skills gap.

The GnuCOBOL Initiative

In January 2002 Keisuke Nishida decided to make a COBOL compiler suitable for integration with gcc; this became OpenCOBOL. Inl 2005 when Roger While took over the project, releasing version 1.0 in December 2007. In September 2013, OpenCOBOL was accepted as a GNU Project, renamed to GNU Cobol, and then finally to GnuCOBOL in September of 2014. Transfer of copyright to the Free Software Foundation over GnuCOBOL source code (including versions with GNU Cobol and OpenCOBOL spellings) was finalized on June 17th, 2015. GnuCOBOL v2.0 is in active development.

GnuCOBOL implements nearly all of the COBOL 85 standards and many of the COBOL 2002 advances. The (US) National Institute of Standards and Technology, NIST, maintains a COBOL 85 implementation verification suite. GnuCOBOL passes over 9000 all of them; but it does not include complete support for report and communication section tests, segment tests, degbugging facilities tests, and obsolete facilities tests.

GnuCOBOL is a strictly speaking a "transpiler"; it converts COBOL statements into C, which is then compiled with the GNU C compiler on a particular system. This also means that GnuCOBOL is works on systems where the GNU C is available; Unix or Linux system, Windows, and even Android and iOS. It also allows the combination of COBOL and C code (examples follow).

Installation

GnuCOBOL requires the following external libraries to be installed: GNU MP (libgmp) 4.1.2+ for decimal arithmetic. Source installation follows a standard `./configure`, `make`, `make check`, `make install`, `ldconfig` process with the usual options (e.g., `./configure --prefix=/usr/local/$(basename $(pwd) | sed 's#-#/#')`). The canonical source is located at `ftp://ftp.gnu.org/gnu/gnucobol/`. On some Red Hat derived installations a `/usr/local/lib` is not automatically searched at runtime. Edit `/etc/ld.so.conf` and add `/usr/local/lib` to the file and rerun `ldconfig`. A clean installation status can be set by running `make distclean`.

Packaged versions are also available. For Debian-based systems use `apt-get install open-cobol`, or Red Hat derived systems `yum install open-cobol` or `dnf install open-cobol`, with plenty of rpms available at `https://www.rpmfind.net/linux/rpm2html/search.php?query=open-cobol`. A Slackware build is available at `http://ftp.slackware.com/pub/slackware/slackware-current/source/d/gnu-cobol/` .

There is also an IDE on Github `https://github.com/OpenCobolIDE/OpenCobolIDE`, which includes syntax highlighting, code completion, folding, configurable margins, auto-indentation, compilation etc.

Program Structure and Example

A COBOL program consists four divisions in sequence: Identification; Environment; Data and Procedure. Each DIVISION may consist of a variety of SECTIONs and each SECTION consists of one or more PARAGRAPHs. A PARARAPH consists of SENTENCEs, each of which consists of one or more STATEMENTs, which consist of CHARACTERS.

Three sample programs that shows the program divisions, comments, followed by compilation in free and fixed formats and traditional style, with all warnings to create the executable, with a specified executable output name, which when run displays "Hello World". A range of compiler options are available (e.g., library creation with -M); one interesting option is to compiler from GnuCOBOL to C. Another example shows the shortest possible fixed format (default) version of the same program:

```
`wget http://levlafayette.com/files/hello.cob`, `cobc -Wall -x -free
hello.cob -o hello-world`, `./hello-world`, `cobc -Wall -m -free
hello.cob`, `cobc -Wall -C -free hello.cob`, `wget
http://levlafayette.com/files/shortest.cob`, `cobc -x shortest.cob`,
`./shortest`, `wget http://levlafayette.com/files/hello-trad.cob`,
`cobc -x hello-trad.cob`, `./hello-trad`
```

Divisions and Variables

The Identification Division is the first and mandatory division of every COBOL program. Within this Division, PROGRAM-ID is the only mandatory paragraph; the ID must be less than 32 characters.

The Environment Division specified input and output to the program, consisting of two sections. First is a Configuration section which consists of two paragraphs; the Source computer (the system used to compile the program) and the Object computer (the system used to execute the program). The Input-Output section contains two paragraphs; File control, which holds information for external datasets used by the program and I-O control, for the files used in the progam.

The Data Division defines the variables in a program. The four sections are a File section, which defines the record structure of file, the Working-Storage section to declare temporary variables, a Local-Storage section which re-initialises variables every time a program is executed, and a Linkage section which describes data names that are received from an external program. The following illustrates the use of variables in GnuCOBOL (the data type will be explained later):

```
`wget http://levlafayette.com/files/luv.cob`, `cobc -Wall -x -free luv.cob`, `./luv`
```

Characters

The lowest level of the COBOL hierarchy are characters, a set of 78 symbols which include the upper and lower case alphabet [a-z, A-Z], numerics [0-9], mathematical and logical operators (+ - / * < > = .), currency ($), and grammatical signs [` " ( ) : ;] and space. Traditional (fixed-form) COBOL was designed for punch-card systems and fixed-terminal width and therefore requires characters to be placed in specific fields: 1-6 for line numbers, 7 for an indicator ('*' indicating comments, '-' indicating continuation, and slash '/' indicating form feed), 8-11, aka Area A for beginning of divisions, sections, paragraphs, 12-72 aka Area B, statements, 73-80, aka Identification Area for programmer's needs.

Strings formed from individual, unseparated, characters. A character string can be a comment, a COBOL reserved word (many of those), user-defined words, or a litreral. Alphanumeric Literals, up to 160 characters, are enclosed in paired quotes or apostrophes. Length can be up to 160 characters. Numeric Literal, up to 18 characters, is a combination of digits from 0 to 9, with a signed +, -, and optional decimal point. Length can be up to 18 characters. Test these literals and try to break it:

```
`wget http://levlafayette.com/files/literals.cob`, `cobc -Wall -free -x literals.cob`, `./literals`
```

Data Types

The Data Division defines varibles. Data consists of the Level Number, the Data Name, a Picture Clause, and Value Clause. To see examples have a look at the `luv.cob` file.

The Level Number specifies the level of data in a record, and is used to differentiate between elementary and group items. The level numbers are 01 (record description), 02 to 49 (group and elementary items), 66 (rename clause items), 77 (items which cannot be sub-divided), and 88 (condition name entry). Group items consist of one or more elementary items; elementary items cannot be subdivided.

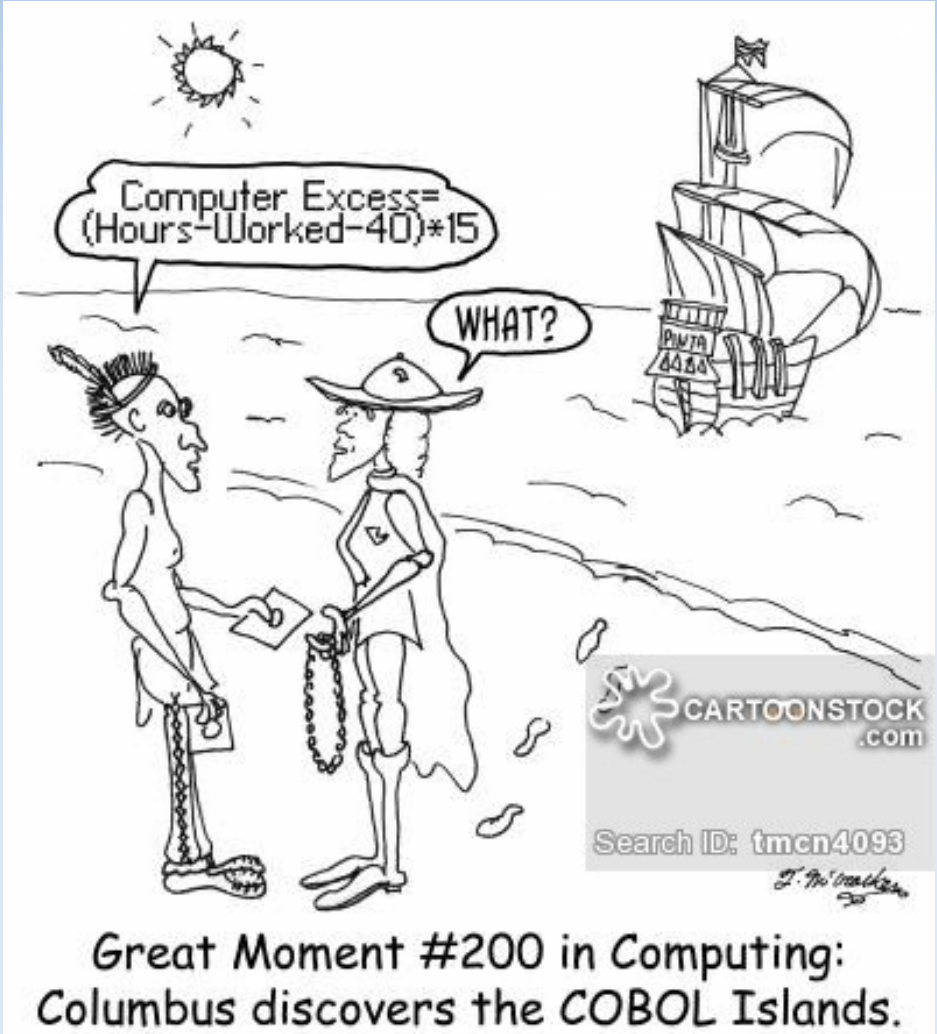
Data names must be defined in the Data Division before using them in the Procedure Division, and must have a user-defined name. Data names can be elementary or group type. Data names give reference to physical memory locations where actual data is stored.

The Picture Clause

The Picture clause (PIC) is used to define the type of data and its presentation. The data type (and assocated symbol) can be numeric (9), alphabetic (A), or alphanumeric (X). The Alphabetic type consists of letters A to Z and spaces. The Alphanumeric type consists of digits, letters, and special characters. There are also associations for implicit decimals (V), assumed decimal (P), and sign vales (S) in numerics. Each PIC clause has a length which defines the the number of bytes used by the data item.

The following illustrates the use of the data division in with various PIC clauses:

```
`wget
http://levlafayette.com/files/posmo
v1.cob`, `cobc -Wall -free -x
posmov1.cob`, `./posmov1`
```



A Whole Bunch of Verbs

COBOL uses 'verbs' are used for input and output of data. The 'Accept' verb is used to get data from the user or even operating system, where as the already illustrated 'Display' verb shows the result on standard output. The 'Initialize' verb is used to initialize a group or elementary item. Alphanumeric or alphabetic data types are set to spaces according to the length and numeric data items are replaces with 0s.

The 'Move' verb copies data from source to a destination and can be used on both elementary and group data items. The sort of move is limited by the data type; e.g., alphabetic content can be moved to alphabetic and alphanumeric items but not numeric.

The 'Add', 'Subtract', 'Multiply' and 'Divide' verbs perform the relevant mathematical actions and store the result in a specified destination. Do not use unsigned numeric items in arithmetic operations. The 'Compute' verb initiates equations using +, -, *, and / instead.

For an example of input, output, and mathematical verbs, download, review, compile, and run the following:

```
`wget http://levlafayette.com/files/posmov2.cob`, `cobc -Wall -free
-x posmov2.cob`, `./posmov2`
```

Modifying Code and Including Datasets

The 'Redefines' clause is used to define a storage with different data description, thus modifying the data layout. An alphanumeric value could be converted, for example, to a numberical or an alphabetical type. Level numbers of redefined item and redefining item must be the same. As previously noted, they cannot be 66 or 88 level number. The 'Renames' clause gives different names to existing data items, whether groups or elementary items. The Level number 66 is reserved for rename clauses.

```
`wget http://levlafayette.com/files/redefines.cob`, `wget
http://levlafayette.com/files/renames.cob`, `cobc -Wall -free -x
redefines.cob`, `cobc -Wall -free -x renames.cob`, `./redefines`,
`./renames`
```

A 'copybook' is a selection of code that defines data structures. If data structure is used in multiple programs then instead of re-writing it it can be stored in a file and inclued with with the the COPY statement in the Working-Storage Section. For example, the data structures in posmov2.cob could be removed to a FINANCIALS copybook which could be included with the 'COPY FINANCIALS' statement. See for example:

```
`wget http://levlafayette.com/files/posmov3.cob`, `wget
http://levlafayette.com/files/FINANCIALS` `cobc -Wall -free -x
posmov3.cob`, `./posmov3`
```

Conditionals and Loops

COBOL has the capacity for conditional clauses. An IF statement checks for condition and if true, the following statements are executed, and if the condition is false, the ELSE block is executed. IF statements can be nested. IF statements are typically applied with relational values ('<' '>' '=','IS', 'NOT', 'AND', 'OR'. Sign conditions can be used to check class conditions (e.g., Alphabetic, Alphabetic-Lower, Alphabetic-Upper). A ladder of IF statements uses the 'Evaluate.. When' verb combination - the equivalent of the C "case" statement.

COBOL also has looping mechanisms with the 'PERFORM' clause, which can be combined with the 'THRU', 'UNTIL', 'TIMES', and 'VARYING' conditions to repeat tasks until the conditional statement is met.

```
`wget http://levlafayette.com/files/posmov4.cob`, `wget
http://levlafayette.com/files/class.cob`, `wget
http://levlafayette.com/files/evaluate.cob` `cobc -Wall -free -x
posmov4.cob`, `cobc -Wall -free -x class.cob`, `./posmov4`,
`./class` ``./evaluate`
```

Multiple Source Files

Serious projects usually consist of multiple source files. GnuCOBOL can use static or dynamic linking, shared libraries can be built and external libraries accessed. As mentioned previously, GnuCOBOL can be combined with GNU C.

Static linking can be as simple as compiling all the source files into a single executable (e.g., `cobc -Wall -x -free main.cob subroutine1.cob subroutine2.cob -o finalprog`). C routines can be linked as well (e.g., `cobc -Wall -free -o main.cob subroutine1.c -o finalprog`), and any linked programs will be converted into the equivalent C code.

Dynamic linking can be achieved by compiling with the `-m` option, and installing the module files in a library directory (e.g., `cobc -Wall -m -free subr.cob`, creating subr.so, then `cp subr.so /your/cobol/lib`, `export COB_LIBRARY_PATH=/your/cobol/lib`, compile the main progam `cobc -Wall -free -x main.cob -o main`, `./main`.

A shared library by combining multiple COBOL programs and C routines with the -c option: (e.g., `cobc -c subr1.cob`, `cobc -c subr2.cob`, `cc -c subr3.c`, `cc -shared -o libsubrs.so subr1.o subr2.o subr3.o`). A shared library can be used by linking it with the main program (e.g., `cp libsubrs.so /usr/lib`, `cobc -x main.cob -lsubrs`).

The Places You Will Go

This brief lecture and workshop does not include COBOL's string handling features, table processing, file handling, sequential and non-sequential access, database connectivity, debugging, and report writing, and much more!

HINT: With GnuCOBOL, you can also make use of parallelisation.

Other places you can go:

GnuCOBOL FAQ
http://opencobol.add1tocobol.com/gnucobol/

OpenCOBOL 1.1 Programmer's Guide, Gary Cutler
http://open-cobol.sourceforge.net/guides/OpenCOBOL%20Programmers%20Guide.pdf

Tutorials Point (although not GnuCOBOL, some examples used in this lecture)
http://www.tutorialspoint.com/cobol/index.htm

Jay Mosley's GnuCOBOL examples (watch out for ^s in the scripts however - remove with sed)
http://www.jaymoseley.com/gnucobol/index.html

A Quick Summary of COBOL syntax
https://en.wikibooks.org/wiki/Software_Engineers_Handbook/Language_Dictionary/COBOL
.


THANKS FOR WATCHING
& LISTENING PATIENTLY
makeameme.org