COBOL — An Introduction

```
identification division.
program-id. COBOL-Workshop.
author. Mike Harris.
procedure division.
display "Hello OxDUG!".
```

My Programming Background

- Started with ZX BASIC on ZX81 and ZX Spectrum
- Moved on to Mallard BASIC (Amstrad PCW) and then to (the excellent and still my favourite) GFA BASIC (Atari ST)
- Learnt Pascal, C, Ada, C++, and OCCAM at University
- Learnt Java professionally, then never used it much
- For my sins, programmed in Perl and PHP for (far too many) years.
- Also wrote bad JavaScript, tried to learn good JavaScript, and toyed with stuff like AngularJS, Node.js and React.js
- Done some Python (nice) and Ruby (hmm)
- Basically messed about with lots of languages over the years

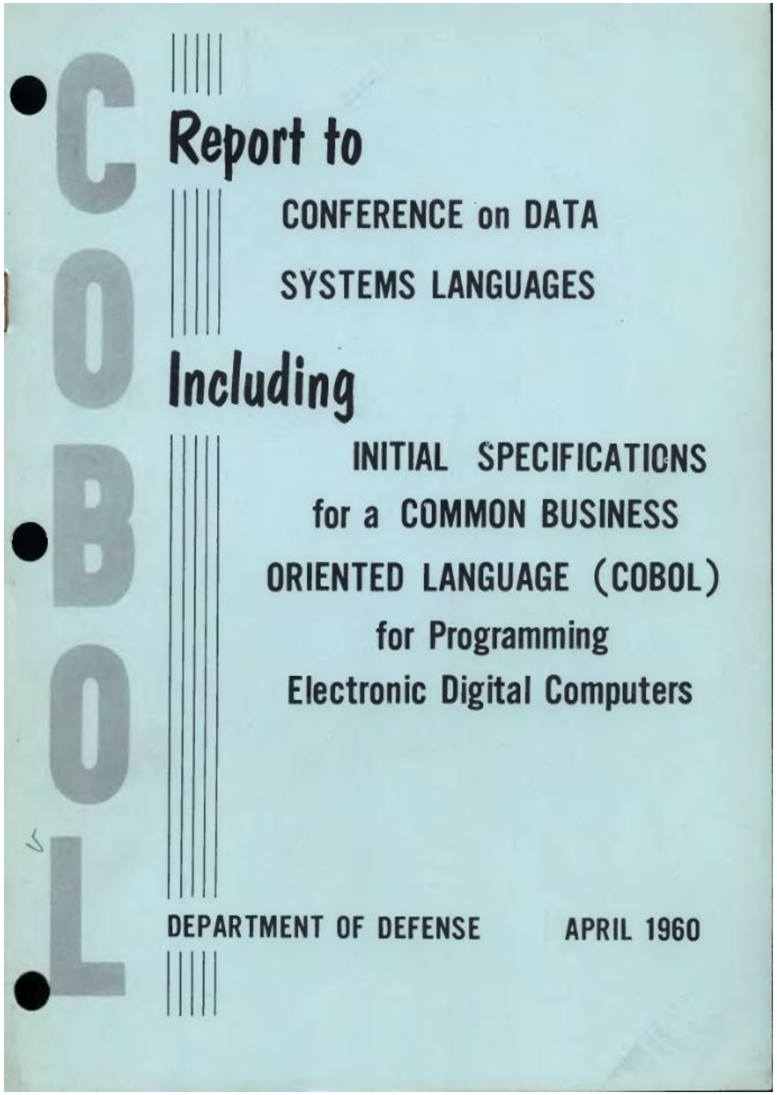
COBOL – History

- **CO**mmon **B**usiness **O**rientated **L**anguage

  (Completely Obsolete Business Orientated Language?)
- “Invented” by Grace Hopper, who was the inventor of FLOW-MATIC.
- Standardised between 1959 and 1960 by our friends at the Pentagon by the group CODASYL.
- Design goal was to be platform and proprietor independent.

COBOL – History

- Appeared in 1959.
- CODASYL COBOL-60
- ANSI COBOL-68
- ANSI COBOL-74 (at this point the most used language in world)
- ANSI COBOL-85 (structured programming additions)
- ISO COBOL-2002 (object orientated additions)
- ISO COBOL-2014 (dynamic tables and modular features)



COBOL

Report to

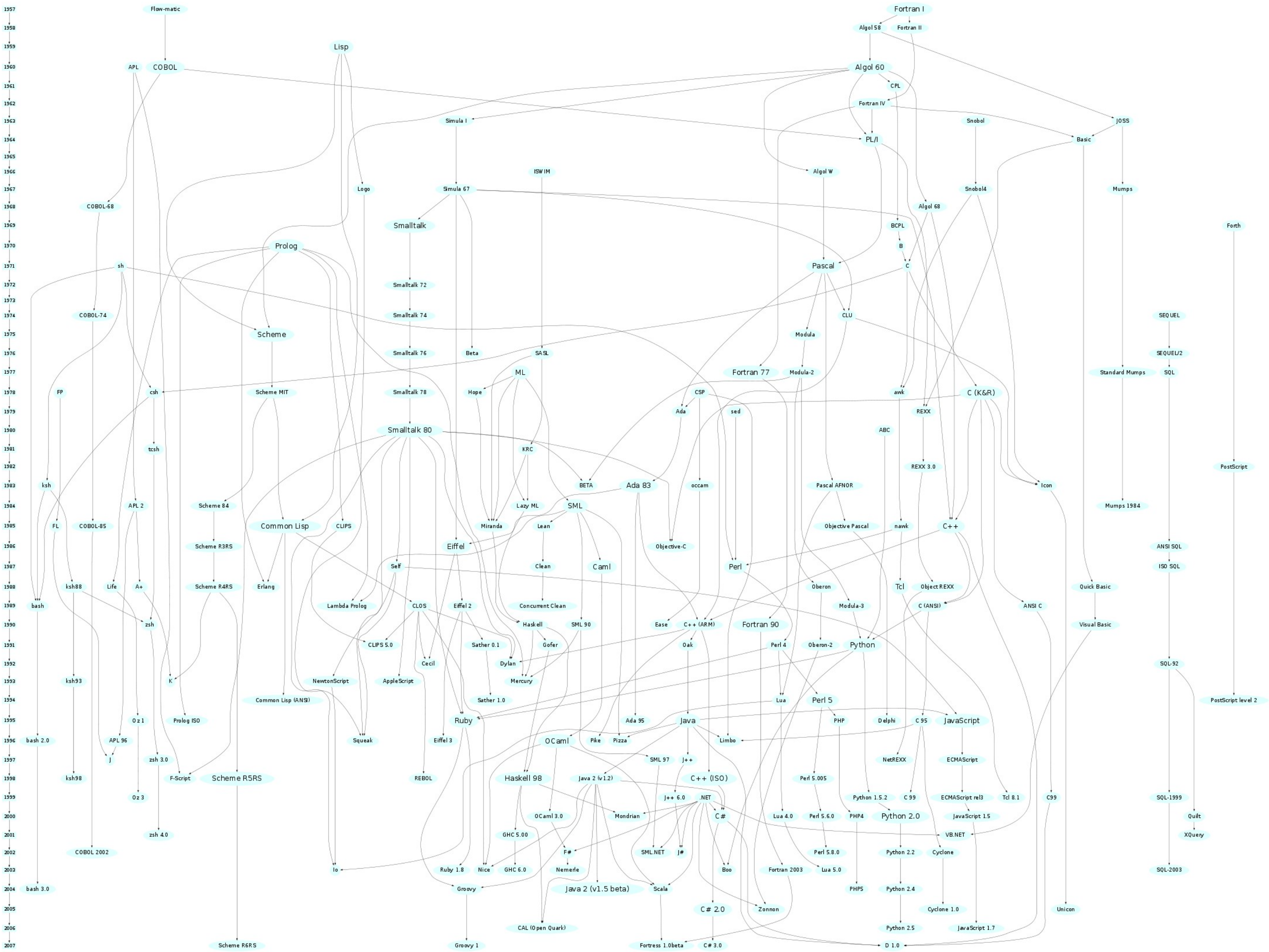
CONFERENCE on DATA
SYSTEMS LANGUAGES

Including

INITIAL SPECIFICATIONS
for a COMMON BUSINESS
ORIENTED LANGUAGE (COBOL)
for Programming
Electronic Digital Computers

DEPARTMENT OF DEFENSE APRIL 1960

Genealogy


1957
Flow-matic
Fortran I
1958
Algol 58
Fortran II
1959
Lisp
1960
APL
COBOL
Algol 60
1961
CPL
1962
Fortran IV
1963
Simula I
Snobol
JOSS
1964
PL/I
Basic
1965
1966
ISWIM
Algol W
1967
Logo
Simula 67
Snobol4
Mumps
1968
COBOL-68
Algol 68
1969
Smalltalk
BCPL
Forth
1970
Prolog
B
1971
sh
Pascal
C
1972
Smalltalk 72
1973
1974
COBOL-74
Smalltalk 74
CLU
SEQUEL
1975
Scheme
Modula
1976
Smalltalk 76
Beta
SASL
SEQUEL/2
1977
ML
Fortran 77
Modula-2
Standard Mumps
SQL
1978
FP
csh
Scheme MIT
Smalltalk 78
Hope
CSP
awk
C (K&R)
1979
Ada
sed
REXX
1980
Smalltalk 80
ABC
1981
tcsh
KRC
1982
REXX 3.0
PostScript
1983
ksh
BETA
Ada 83
occam
Pascal AFNOR
Icon
1984
APL 2
Scheme 84
Lazy ML
SML
Mumps 1984
1985
FL
COBOL-85
Common Lisp
CLIPS
Miranda
Lean
Objective Pascal
nawk
C++
1986
Scheme R3RS
Eiffel
Objective-C
ANSI SQL
1987
Self
Clean
Caml
Perl
ISO SQL
1988
ksh88
Life
A+
Scheme R4RS
Erlang
Oberon
Tcl
Object REXX
Quick Basic
1989
bash
Lambda Prolog
CLOS
Eiffel 2
Concurrent Clean
Modula-3
C (ANSI)
ANSI C
1990
zsh
Haskell
SML 90
Ease
C++ (ARM)
Fortran 90
Visual Basic
1991
CLIPS 5.0
Sather 0.1
Gofer
Oak
Perl 4
Oberon-2
Python
1992
Cecil
Dylan
SQL-92
1993
ksh93
K
NewtonScript
AppleScript
Mercury
1994
Common Lisp (ANSI)
Sather 1.0
Lua
Perl 5
PostScript level 2
1995
Oz 1
Prolog ISO
Ruby
Ada 95
Java
PHP
Delphi
C 95
JavaScript
1996
bash 2.0
APL 96
Squeak
Eiffel 3
OCaml
Pike
Pizza
Limbo
1997
J
zsh 3.0
SML 97
J++
NetREXX
ECMAScript
1998
ksh98
F-Script
Scheme R5RS
REBOL
Haskell 98
Java 2 (v1.2)
C++ (ISO)
Perl 5.005
1999
Oz 3
J++ 6.0
.NET
Python 1.5.2
C 99
ECMAScript rel3
Tcl 8.1
C99
SQL-1999
2000
OCaml 3.0
Mondrian
C#
Lua 4.0
Perl 5.6.0
PHP4
Python 2.0
JavaScript 1.5
Quilt
2001
zsh 4.0
GHC 5.00
VB.NET
XQuery
2002
COBOL 2002
F#
SML.NET
J#
Perl 5.8.0
Python 2.2
Cyclone
2003
Io
Ruby 1.8
Nice
GHC 6.0
Nemerle
Boo
Fortran 2003
Lua 5.0
SQL-2003
2004
bash 3.0
Groovy
Java 2 (v1.5 beta)
Scala
PHP5
Python 2.4
2005
C# 2.0
Zonnon
Cyclone 1.0
Unicon
2006
CAL (Open Quark)
Python 2.5
JavaScript 1.7
2007
Scheme R6RS
Groovy 1
Fortress 1.0beta
C# 3.0
D 1.0

COBOL: pros & cons

- It's arguably very well adapted to its domain of finance and mass data processing.
- It's verbose and this helps readability of code and thus is said to be self-documenting.
- It's very stable. With the exception of OO additions, the last major change was in 1985. This makes it also very maintainable.
- It runs across many, many platforms and OSes.
- It's relative simple as a language.
- It's nonproprietary.
- It has powerful file, string and numerical handling functions built in.
- Legacy code base is very stable with almost no new bugs being introduced.

- There's a LOT of legacy code, which is spaghetti-like (but then there's a lot of JavaScript like that!)
- OO is still not fully supported in all versions.
- It's not suitable for a lot of applications, such as embedded programming.
- It has a lot of reserved words, which could be a good thing depending on your viewpoint.
- Structured programming is possible, but it may feel 'clunky' compared to other languages.
- Best IDE is MicroFocus, but this is commercial software – then again the best IDEs normally are (excepting vi & emacs).

Reserved words

ABS, ACOS, ANNUITY, ASIN, ATAN, BYTE-LENGTH, CHAR, COMBINED-DATETIME, CONCATENATE, COS, CURRENCY-SYMBOL, CURRENT-DATE, DATE-OF-INTEGER, DATE-TO-YYYYMMDD, DAY-OF-INTEGER, DAY-TO-YYYYDDD, E, EXCEPTION-FILE, EXCEPTION-LOCATION, EXCEPTION-STATEMENT, EXCEPTION-STATUS, EXP, EXP10, FACTORIAL, FORMATTED-CURRENT-DATE, FORMATTED-DATE, FORMATTED-DATETIME, FORMATTED-TIME, FRACTION-PART, HIGHEST-ALGEBRAIC, INTEGER, INTEGER-OF-DATE, INTEGER-OF-DAY, INTEGER-OF-FORMATTED-DATE, INTEGER-PART, LENGTH, LENGTH-AN, LOCALE-COMPARE, LOCALE-DATE, LOCALE-TIME, LOCALE-TIME-FROM-SECONDS, LOG, LOG10, LOWER-CASE, LOWEST-ALGEBRAIC, MAX, MEAN, MEDIAN, MIDRANGE, MIN, MOD, MODULE-CALLER-ID, MODULE-DATE, MODULE-FORMATTED-DATE, MODULE-ID, MODULE-PATH, MODULE-SOURCE, MODULE-TIME, MONETARY-DECIMAL-POINT, MONETARY-THOUSANDS-SEPARATOR, NUMERIC-DECIMAL-POINT, NUMERIC-THOUSANDS-SEPARATOR, 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, SQRT, STANDARD-DEVIATION, STORED-CHAR-LENGTH, SUBSTITUTE, SUBSTITUTE-CASE, SUM, TAN, TEST-DATE-YYYYMMDD, TEST-DAY-YYYYDDD, TEST-FORMATTED-DATETIME,
TEST-NUMVAL, TEST-NUMVAL-C, TEST-NUMVAL-F, TRIM, UPPER-CASE, VARIANCE, WHEN-COMPILED, YEAR-TO-YYYY IDENTIFICATION DATA DIVISION SECTION
GREATER LESS SET STRING UNSTRING EVA
WHEN IS THEN IF END PROGRAM FUNCTION
PICTURE

COBOL has some 500+ reserved words.

C in contrast has just 50.

Prolog has none!

Verbosity

```
// Calculation in C:

 if (hours_worked <= standard_hours)
         amount = 40 * payrate;
 else
         amount = hours * payrate;
```

```
*> Calculation in COBOL:

IF NumberOfHoursWorked IS LESS THAN OR EQUAL TO StandardHours THEN
        MULTIPLY Payrate BY 40 GIVING Amount
ELSE
        MULTIPLY Payrate BY Hours GIVING Amount
END-IF.
```

```
*> Shorter-form calculation in COBOL:

IF NumberOfHoursWorked <= StandardHours
        COMPUTE Amount = Payrate * 40
ELSE
        COMPUTE Amount = hours * payrate
END-IF.
```

Legibility

```
identification division.
program-id. SalesTax.
working-storage section.
01 beforeTax      picture 999V999 value 123.45.
01 salesTaxRate   picture V9999     value .065.
01 afterTax       picture 999.99.
procedure division.
Main.
      compute afterTax rounded = beforeTax + (beforeTax
* salesTaxRate)
      display “After tax amount is “ afterTax.
```

```
import java.math.BigDecimal;
public class SalesTaxWithBigDecimal
{
  public static void main(java.lang.String[] args)
  {
    BigDecimal beforeTax    = BigDecimal.valueOf(12345, 2);
    BigDecimal salesTaxRate = BigDecimal.valueOf(65, 3);
    BigDecimal ratePlusOne  =
salesTaxRate.add(BigDecimal.valueOf(1));
    BigDecimal afterTax   = beforeTax.multiply(ratePlusOne);
    afterTax = afterTax.setScale(2, BigDecimal.ROUND_HALF_UP);
    System.out.println( "After tax amount is " + afterTax);
} }
```

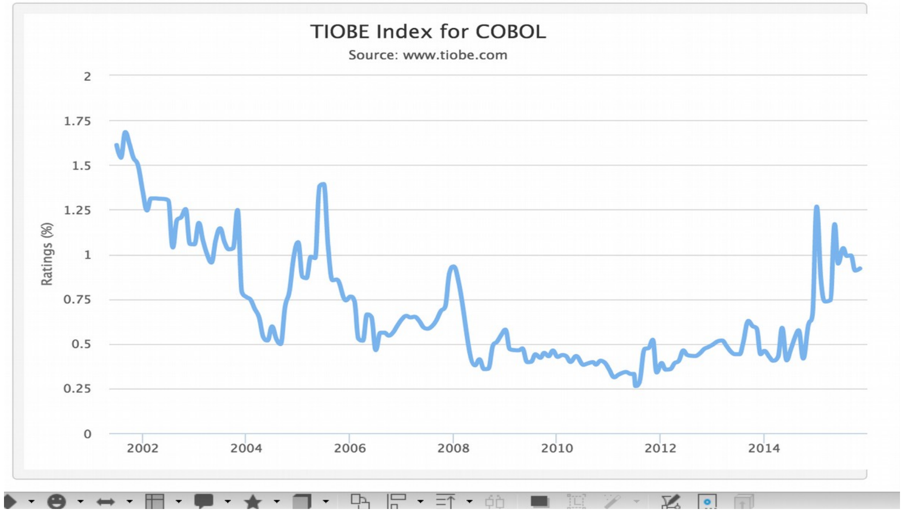
```
identification division.
program-id. sumofintegers.
data division.
working-storage section.

01 n binary-int.
01 i binary-int.
01 sum binary-int.

procedure division.

display “Enter a positive integer:”
accept n
perform varying i from 1 by 1 until i is greater than n
      add i to sum
end-perform
display “The sum is:” sum.
```

```
import java.util.Scanner;

public class sumofintegers {

      public static void main(String[] arg) {
            System.out.println(“Enter a positive integer:”);
            Scanner scan=new Scanner(System.in);
            int n=scan.nextInt();
            int sum=0;
            for (int i=1;i<=n;i++) {
                  sum=sum+i;
            }

            System.out.println(“The sum is “+sum);
      }
}
```

COBOL Usage

- 2012 Computer World survey found that over 60% of organisations used COBOL with 54% saying that more than half of their internal business code was written in it (compared to 39% for Java).
- Over 27% said that COBOL was used for more than half of new development.
- In May 2013 IBM noted that 15% of all new enterprise functionality is written in COBOL and that there are 200,000,000,000 lines of code in use, growing between 3% and 5% per year.
- 2005 report cited that COBOL handles 75% of all computer transactions and 90% of all financial transactions.
- But I work in the world of the web and nobody is talking about it there.

- 2.6 million lines of code (LOC) in 100 programs.
- Estimates are that some 4 million new lines of code written every year.
- It's currently at position 20 TIOBE's index of top programming languages (up from 28th last year)
- It's been 8th and 47th in the last 14 years.

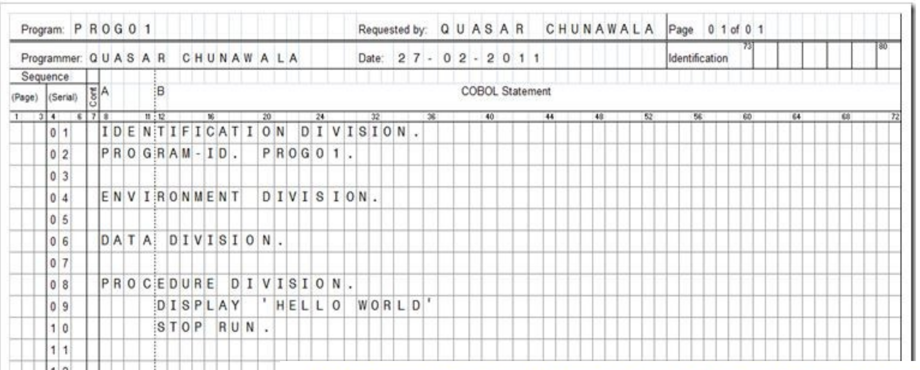

TIOBE Index for COBOL

Source: www.tiobe.com

Let's look at some COBOL

Hello World

- Classic COBOL

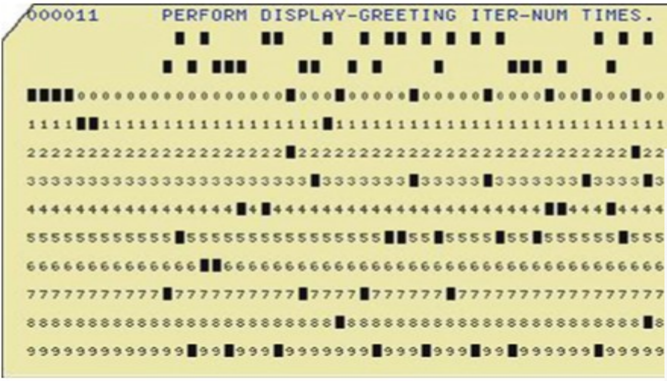


Program: PROG01 Requested by: QUASAR CHUNAWALA Page 01 of 01

Programmer: QUASAR CHUNAWALA Date: 27-02-2011 Identification

```
01 IDENTIFICATION DIVISION.
02 PROGRAM-ID.  PROG01.
03
04 ENVIRONMENT  DIVISION.
05
06 DATA DIVISION.
07
08 PROCEDURE DIVISION.
09        DISPLAY 'HELLO WORLD'
10        STOP RUN.
```

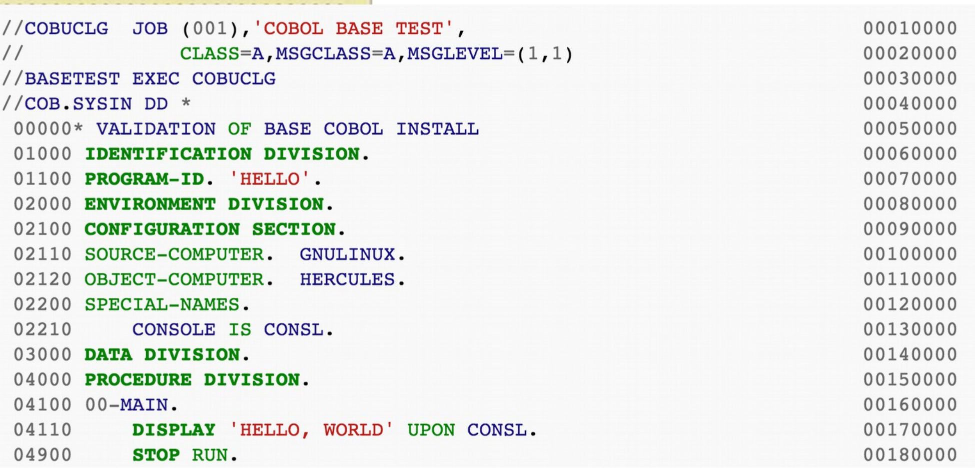
Sequence
(Page)
(Serial)
Cont
A
B
COBOL Statement



```
000011     PERFORM DISPLAY-GREETING ITER-NUM TIMES.                GREETING
```

- Modern COBOL

```
program-id. HelloWorld.
procedure division.
display "Hello World!".
```

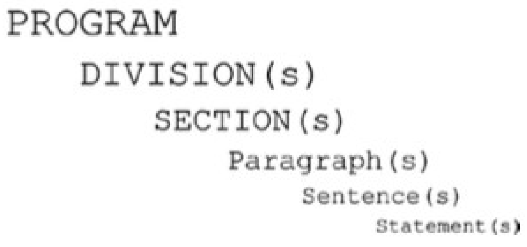


```
//COBUCLG  JOB (001),'COBOL BASE TEST',                                 00010000
//             CLASS=A,MSGCLASS=A,MSGLEVEL=(1,1)                          00020000
//BASETEST EXEC COBUCLG                                                   00030000
//COB.SYSIN DD *                                                          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
```

Basic Structure

- Programs are organised into Programs, Divisions, Sections, Paragraphs, Sections, Sentences and Statements.
- Not case sensitive, but traditional way is to use UPPER CASE; I prefer lower case.
- Program <u>must</u> have a *program-id*
- *It's very verbose and has a lot of noise words.*

```
identification division.
program-id. HelloWorld.
data division.
working-storage section.
01 Friend pic x(5) value “Bob”.
procedure division.
display "Hello " Friend
move “Alice” to Friend
display “Hello “ Friend.
```



```
PROGRAM
    DIVISION(s)
        SECTION(s)
            Paragraph(s)
                Sentence(s)
                    Statement(s)
```

Developing in COBOL

Linux (Debian)

```
sudo apt-get install open-cobol python3-pip python-qt5
sudo pip3 install OpenCobolIDE --upgrade
```

GnuCOBOL 1.1 is stable and 2.0 is in development.

Linux (CentOS/RedHat)

```
sudo yum install
```

OS X

Install home brew (from http://brew.sh/)

```
brew install gnu-cobol
```

Get the IDE from **http://ttfa.net/cobolide**

Run the IDE

Windows

Binary build from http://ttfa.net/gnucobol1

Get the IDE from http://ttfa.net/cobolide

HackEdit is a cool project – an editor that supports Python & COBOL

Find it on GitHub

Or try http://www.tutorialspoint.com/compile_cobol_online.php (doesn't work in Chrome for me; but does in Firefox)

Hello World example

```
program-id. HelloWorld.
procedure division.
display "Hello World!".
```

Personanlised Hello World

```
identification division.
program-id. HelloWorld.

data division.
working-storage section.
01 MyName   pic x(20).
   88 UserIsMike  value “Mike” spaces.
procedure division.
display "Enter your name: " with no advancing
accept MyName
display "Hello " MyName "!"
if UserIsMike then
   display “You're so great at COBOL!”
end-if.
```

```
$ cobc -x -free myprogram.cbl
$ ./myprogram
```

Command Line Hello World

```
identification division.
program-id. HelloWorld.

data division.
working-storage section.
01 MyName   pic x(20).

procedure division.
display "Enter your name: " with no advancing
accept MyName from argument-value
display "Hello " MyName "!".
```

```
$ cobc -x -free myprogram.cbl
$ ./myprogram “Mike Harris”
```

Saying hello to lots of people

```
program-id. HelloWorld.

environment division.

data division.

working-storage section.

01 MyName               pic x(255).

01 NumberOfArguments    pic 9.

01 CurrentArgumentIndex pic 9.

procedure division.

accept NumberOfArguments from argument-number

    perform varying CurrentArgumentIndex from 1 by 1
       until CurrentArgumentIndex > NumberOfArguments

        accept MyName from argument-value

        display "Hello " function trim(MyName trailing)
          " welcome to HacktionLab"

    end-perform

.
```

```
$ cobc -x -free myprogram.cbl
$ ./myprogram Mike Bob Alice
```

Calculator (Evaluate)

```
PERFORM WITH TEST BEFORE UNTIL OperatorIsStopRun
PERFORM EnterNumbers
EVALUATE TRUE
    WHEN OperatorIsAdd COMPUTE Result = Num1 + Num2
    WHEN OperatorIsSubtract COMPUTE Result = Num1 - Num2
    WHEN OperatorIsMultiply COMPUTE Result = Num1 * Num2
    WHEN OperatorIsDivide DIVIDE Num1 BY Num2 GIVING Result
    WHEN OTHER SET Result TO 0
END-EVALUATE

DISPLAY "Result is ", Result
PERFORM ValidateOperator WITH TEST AFTER UNTIL ValidOperator
END-PERFORM
```

Monty Hall

Suppose you're on a game show and you're given the choice of three doors. Behind one door is a car; behind the others, goats. The car and the goats were placed randomly behind the doors before the show.

The rules of the game show are as follows:

After you have chosen a door, the door remains closed for the time being. The game show host, Monty Hall, who knows what is behind the doors, now has to open one of the two remaining doors, and the door he opens must have a goat behind it. If both remaining doors have goats behind them, he chooses one randomly. After Monty Hall opens a door with a goat, he will ask you to decide whether you want to stay with your first choice or to switch to the last remaining door.

For example:
Imagine that you chose Door 1 and the host opens Door 3, which has a goat. He then asks you "Do you want to switch to Door Number 2?" Is it to your advantage to change your choice?

Note that the player may initially choose any of the three doors (not just Door 1), that the host opens a different door revealing a goat (not necessarily Door 3), and that he gives the player a second choice between the two remaining unopened doors.

Simulate at least a thousand games using three doors for each strategy and show the results in such a way as to make it easy to compare the effects of each strategy.

What else can it do?

- File handling – sequential and direct acess.
- External sub-programs (libraries)
- Copybooks (include files)
- Powerful string handling and other intrinsic functions.
- GnuCOBOL Hooks into databases, but not MySQL at the moment.
- Powerful report writing.
- User defined functions.
- Object-orientated COBOL with classes, objects, factories, inheritance, interfaces, etc.

```
CLASS-ID. Tester-cls AS "tester"
          INHERITS FROM Base.

REPOSITORY.
    CLASS BASE AS "base"
    CLASS Tester-cls AS "tester".

FACTORY.
WORKING-STORAGE SECTION.
01 InstCounter-fws   PIC 9 VALUE ZEROS.
01 FactoryData-fws   PIC 9 VALUE ZEROS.

METHOD-ID. New.
LOCAL-STORAGE SECTION.

01 LocalData-mls   PIC 9 VALUE ZEROS.
LINKAGE SECTION.
01 TestObject-lnk  OBJECT REFERENCE.
PROCEDURE DIVISION RETURNING TestObject-lnk.
. . . .
END METHOD New.
END FACTORY.
END CLASS TesterCls.
```

Conclusions

- COBOL is not dead
- COBOL is not really all that bad either
- Good, clean COBOL code can be written
- Old, nasty COBOL code can and should be refactored

we can learn some lessons from this….

- Good, clean code can be written in any programming language
- Old, nasty code in any language can and should be refactored
- Refactor your code early and often to avoid technical debt (even if it is written in the latest, snazziest thing using the trendiest programming paradigm).
- Re-writing a project in a new code base is often the highest risk approach to take; the result is likely not to be an improvement
- Any code you write in whatever programming language could end up being around for a very, very long time; as can the language (to wit ALGOL, FORTRAN, BASIC, COBOL, PL/1 and even Perl are all still out there!)

```
display “Thank You”
stop run.
end program COBOL-Workshop.
```