

COBOL – An Introduction

```
program-id. Greetings.  
  procedure division.  
display "Hello HacktionLab!".
```

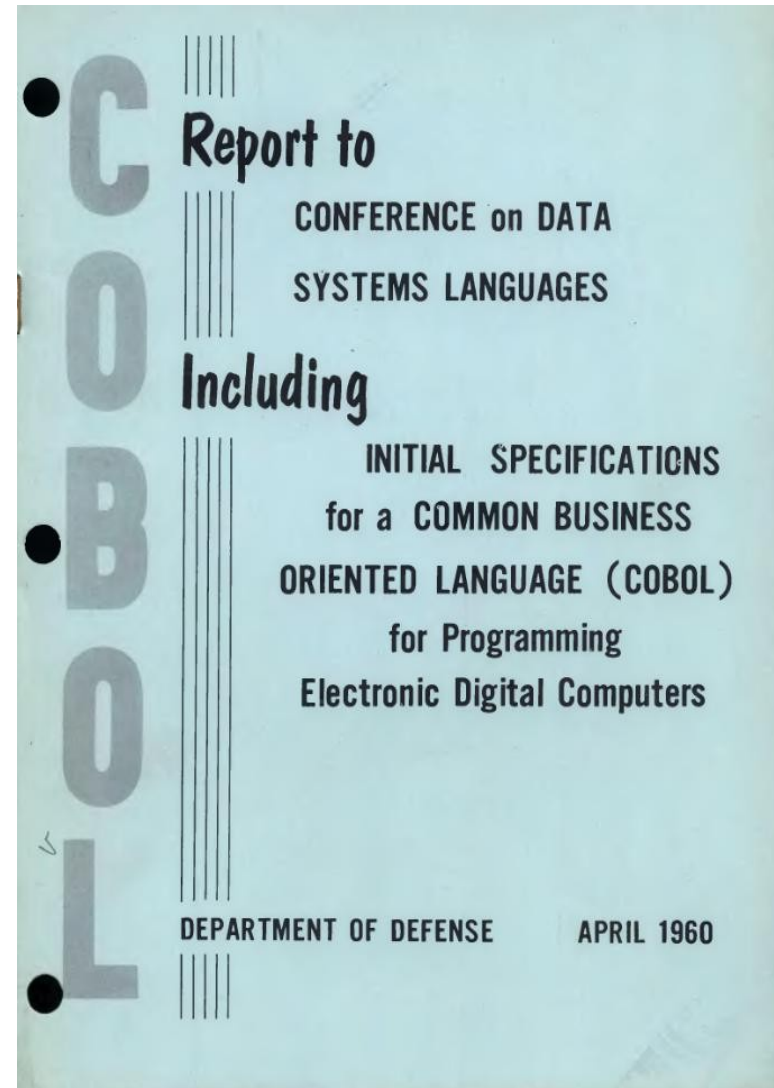
COBOL - History

- **CO**mmun **B**usiness **O**rientated **L**anguage
- “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.

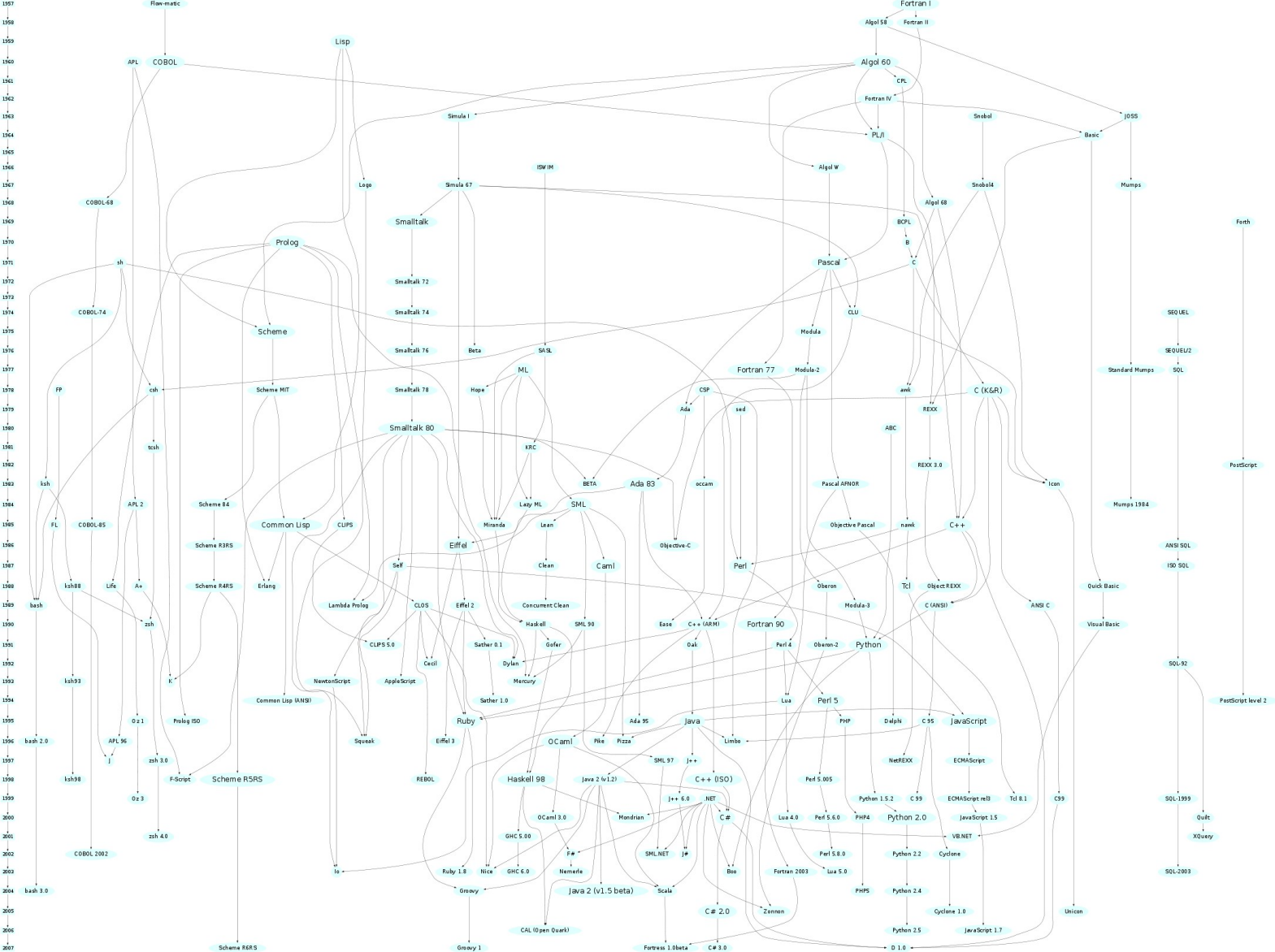


COBOL - History

- 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)



Genealogy



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-INTEGERS, DATE-TO-YYYYMMDD, DAY-OF-INTEGERS, 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 EVALUATE 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;

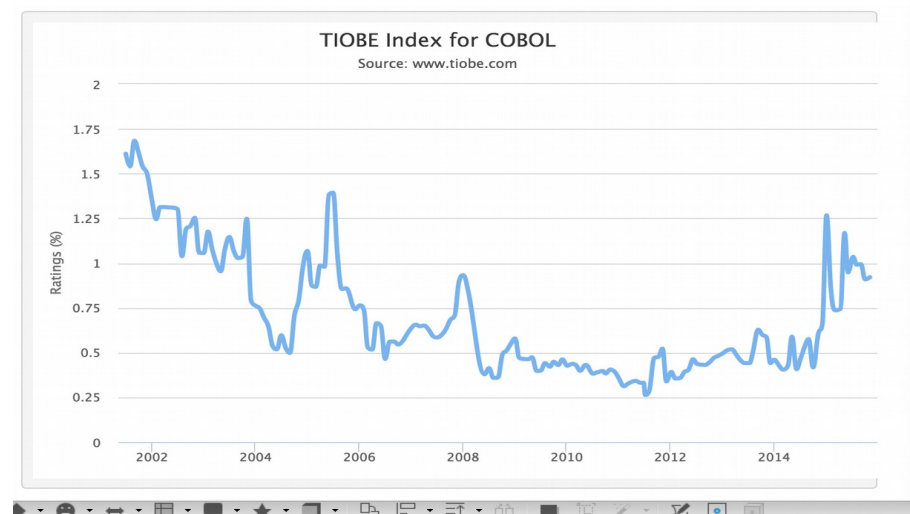
public class sumofintegers {

    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.



Let 's program some COBOL

Installing 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>

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

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.
- Program must 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(4) value "Mike".  
procedure division.  
display "Hello " Friend.
```

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

Let ' s program some COBOL

- Getting it and an IDE
- Type in Hello World example
- Introduce PIC and ACCEPT.
- Write Hello “Name” program.
- Introduce reading from command line and improve our program.
- Introduce IF and EVALUATE.
- Improve our example again.

Hello World example

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

Extending our program

- PIC (PICTURE) clause used to define storage for a variable.
- 01 YourName PIC X(20).
- Defines a variable called 'YourName' that can hold 20 alphanumeric characters.
- Level 88 allows for conditions on variables!
- ACCEPT reads input from the console and writes to a variable.
- ACCEPT SomeVar
- Takes input from the console and writes the result into SomeVar.

Personanlised Hello World

identification division.

program-id. HelloWorld.

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

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.

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
```

Rock, paper, scissors

```
EVALUATE TRUE ALSO TRUE
    WHEN RockA ALSO RockB SET NobodyWins TO TRUE
    WHEN PaperA ALSO PaperB SET NobodyWins TO TRUE
    WHEN ScissorsA ALSO ScissorsB SET NobodyWins TO TRUE
    WHEN RockA ALSO ScissorsB SET PlayerAWins TO TRUE
    WHEN RockB ALSO ScissorsA SET PlayerBWins TO TRUE
    WHEN PaperA ALSO RockB SET PlayerAWins TO TRUE
    WHEN PaperB ALSO RockA SET PlayerBWins TO TRUE
    WHEN ScissorsA ALSO PaperB SET PlayerAWins TO TRUE
    WHEN ScissorsB ALSO PaperA SET PlayerBWins TO TRUE
    WHEN OTHER SET NobodyWins TO TRUE
END-EVALUATE.
```

Calculator

```
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
```

Final exercise

Write a new version of Rock, Paper, Scissors that has the following new features:

1. When called with a single command line parameter, it will run the game that number of times before stopping. If no parameters are entered then it runs just once, as it does currently.
2. When called with three command line parameters, the first is used as the number of rounds to play and the next two are the names of the players, for example:

```
$ ./rps 5 Alice Bob
```

Plays 5 rounds between Alice and Bob. If Alice were to win a round it should output her name, rather than PlayerA, like it does at present.

What else can it do?

- File handling – sequential and direct access.
- 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.

```
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.
```

```
DISPLAY "Thank You".  
STOP RUN.
```