

COBOL: Your Secret To Long Term Job Security

IEEE FWCS Pace Group

October, 2025



Welcome To The World of COBOL

- COBOL: **C**ommon **B**usiness **O**rientated **L**anguage
- Who uses COBOL Today?
 - Accounting Systems
 - Banking
 - Financial Services
 - Inventory Control
 - Payroll
 - Etc.



The World of The 1950's

Scientific Programming FORTRAN	Business Programming COBOL
--	--------------------------------------

Where Did COBOL Come From?

- Grace Brewster Hopper was an American computer scientist, mathematician, and United States Navy rear admiral.
- She was part of the team that developed the UNIVAC I computer. She managed the development of one of the first COBOL compilers.
- She believed that programming should be simplified with an English-based computer programming language.
- Her compiler converted English terms into machine code understood by computers.
- In 1959, she participated in the CODASYL consortium, helping to create a machine-independent programming language called COBOL, which was based on English words.



What Was The Big Deal?

- FORTRAN was too scientific for business applications
- In 1959, programs developed in machine/assembly were not portable.
- It was very expensive to develop programs: \$800,000 on average for a program to be developed (in 1959 dollars!).
- Porting a program to run on another computer would cost \$600,000.
- Goals for creating a programming language for business:
 - Be flexible and work in Government, Business, Healthcare, etc.
 - Be able to handle huge amounts of data on a large scale
 - Syntax that resembles everyday English
- COBOL was designed to be a high capacity language: it can process enormous amounts of data.

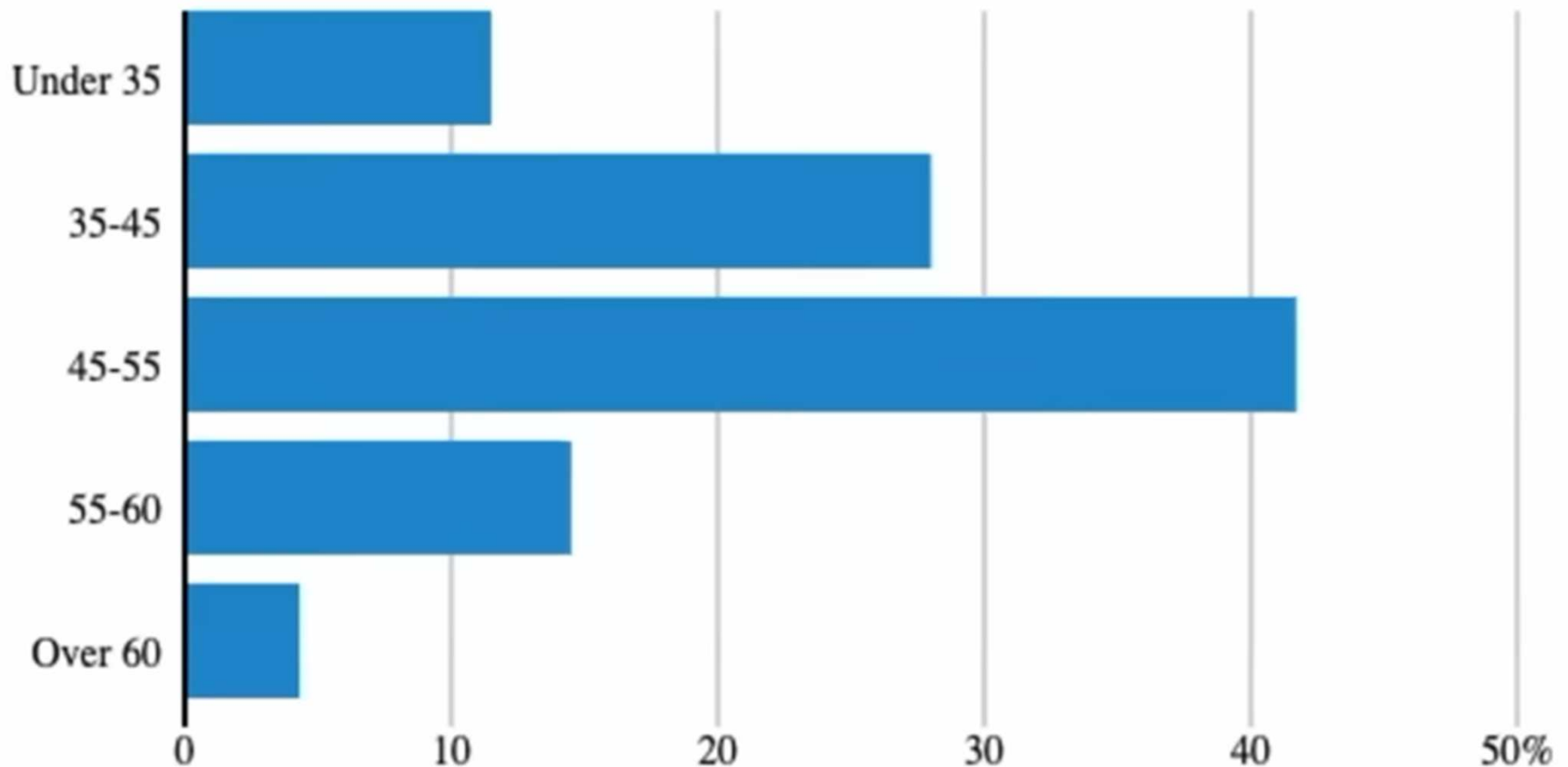
Why Bother To Learn COBOL?

- The reality is that COBOL is the major programming language for business applications even today.
- More than 220 billion lines of COBOL code are still in use which equals ~80% of the world's actively used code.
- 90% of critical business applications use COBOL.
- 95% of ATM transactions are handled by COBOL / 60% healthcare records
- U.S. Government: Social Security, Department of Defense Payment Systems, Internal Revenue Service
- Every day \$3 Trillion worth of transactions are handled by COBOL code.
- No – new applications are generally not developed in COBOL. But all of the existing ones that run the world were developed in COBOL and they have to be enhanced and maintained...

Average Age Of A COBOL Developer

AVERAGE AGE OF DEVELOPERS

On average, COBOL programmers are most likely to be between 45-55 years old.



Caution!

- Just learning COBOL may not be enough to secure a high paying job
- COBOL program often run on Mainframes
- To work in a Mainframe environment you also have to know...
- JCL
 - JCL, or Job Control Language, is a scripting language used primarily on IBM mainframe computers to define and control the execution of batch jobs.
- IMS
 - IMS, or Information Management System, is a premier transaction and hierarchical database management system developed by IBM for mainframes. It's designed for critical online applications and data requiring high availability, performance, and integrity.
- DB2
 - DB2 is a family of database server products developed by IBM. It's a relational database management system (RDBMS) known for its scalability, reliability, and performance, particularly in handling large volumes of data.

Components OF The COBOL Language

Components OF The COBOL Language

- Characters
- Reserved Keywords
- User Defined Words
- Variables, Literals, Structures
- Optional Words
- Constants
- Intrinsic Functions

Characters

- The COBOL language is made up of characters.
- The complete set of characters recognized by COBOL is shown in the table:

Character(s)	Meaning
A – Z	Alphabets (Upper case)
a – z	Alphabets (Lower case)
0 – 9	Digits
+	Plus (For addition)
-	Minus (For subtraction) or Hyphen
*	Asterisk (For multiplication)
/	Slant, Stroke or Slash (For division)
=	Equal to sign
\$	Dollar (For currency sign)
,	Comma
;	Semicolon
.	Decimal point or Period
"	Quotation mark
(	Left parenthesis
)	Right Parenthesis
>	Greater than
<	Less than
:	Colon
'	Apostrophe

Note: Single quote ➡

Let's Talk About Periods

- In COBOL, periods (.) serve as statement terminators and logical delimiters—but their usage has some quirks and historical baggage.
- In traditional COBOL, each paragraph, sentence, or section usually ends with a period:
`DISPLAY "HELLO, WORLD".`
`MOVE ZERO TO TOTAL.`
- A period tells the compiler: "This is the end of a complete statement or block."
- When you define a **paragraph** or **section**, COBOL expects a period after the last statement in that block:
`MAIN-PARAGRAPH.`
`DISPLAY "This is the main paragraph".`
`STOP RUN.`
- Periods end all open IFs, EVALUATEs, or PERFORMs, which can lead to logic issues if placed improperly.
- At the end of a **division**, **section**, or program, periods can help denote the end of a unit

Reserved Words In COBOL

1. Keywords
2. Optional Words
3. Figurative Constants
4. Special Character Words
5. Special Object Identifiers
6. Special Registers

Keywords

- **Def**: Reserved words that are required within a given clause, entry, or statement. Keywords appear in UPPERCASE.
- Example:
 - **ADD** – add two or more variables
 - **DELETE** – delete a record from a file
 - **SEARCH** – search for a value in an area or a table
 - **READ** – read data from a file
 - **WRITE** – write a record to a file
 - **CALL** – transfer control to a subprogram from the main program

```
PROCEDURE DIVISION.  
  
001507*  
001508* Read data from the Employee Month File.  
001509*  
001510 000030-READ-TREMP.  
001520      READ EMPRC AT END  
001530      MOVE 'Y'      TO EOF-OF-FILE  
001540      END-READ  
001550 ..... * *  
001570      ADD 1          TO TR-INREC  
..... * *
```

COBOL Program Structure

Hierarchical Structure Of A COBOL Program

4 divisions in a COBOL program:
Identification, Environment, Data, and Procedure.

Divisions

Sections

Sections contain one or more paragraphs.
They give a program structure

Paragraphs

Paragraphs are made up of one or more sentences. Can be used as modules called by other modules.

Sentences contain one or more statements and end with a period.

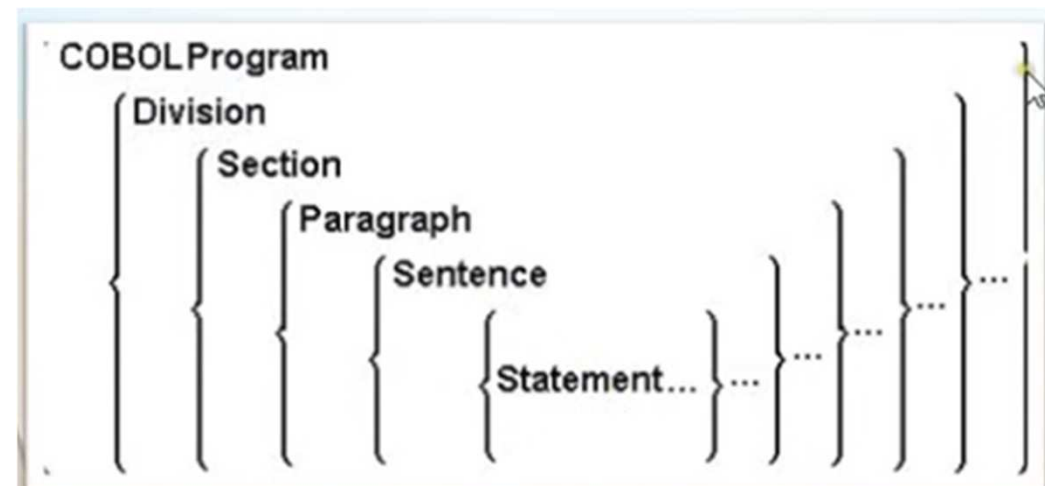
Sentences

Statements start with a verb and a verb shows an action: INITIALIZE, SET, SEARCH, MOVE, ADD. Use verbs with a variable to make a statement.

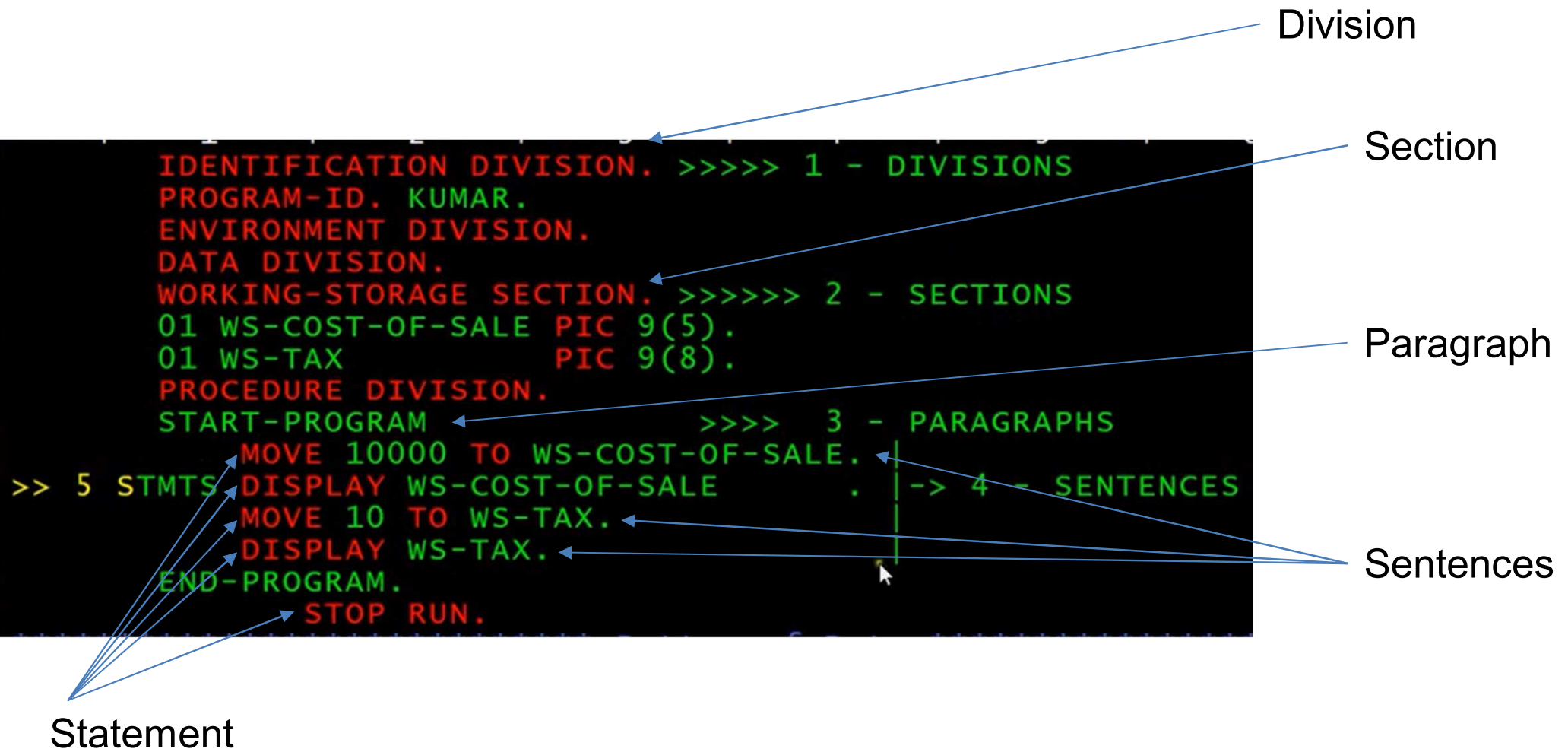
Statements

Components Of A COBOL Program

Character	Digits (0-9), Alphabets (A-Z), Space (b), Special Characters (+ - * / () = \$; " > < . ,)
Word	One or more characters- User defined or Reserved
Clause	One or more words. It specifies an attribute for an entry
Statement	One or more valid words and clauses
Sentence	One or more statements terminated by a period
Paragraph	One or more sentences.
Section	One or more paragraphs.
Division	One or more sections or paragraphs
Program	Made up of four divisions

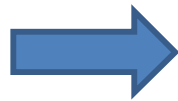


Example Components



COBOL Coding Rules

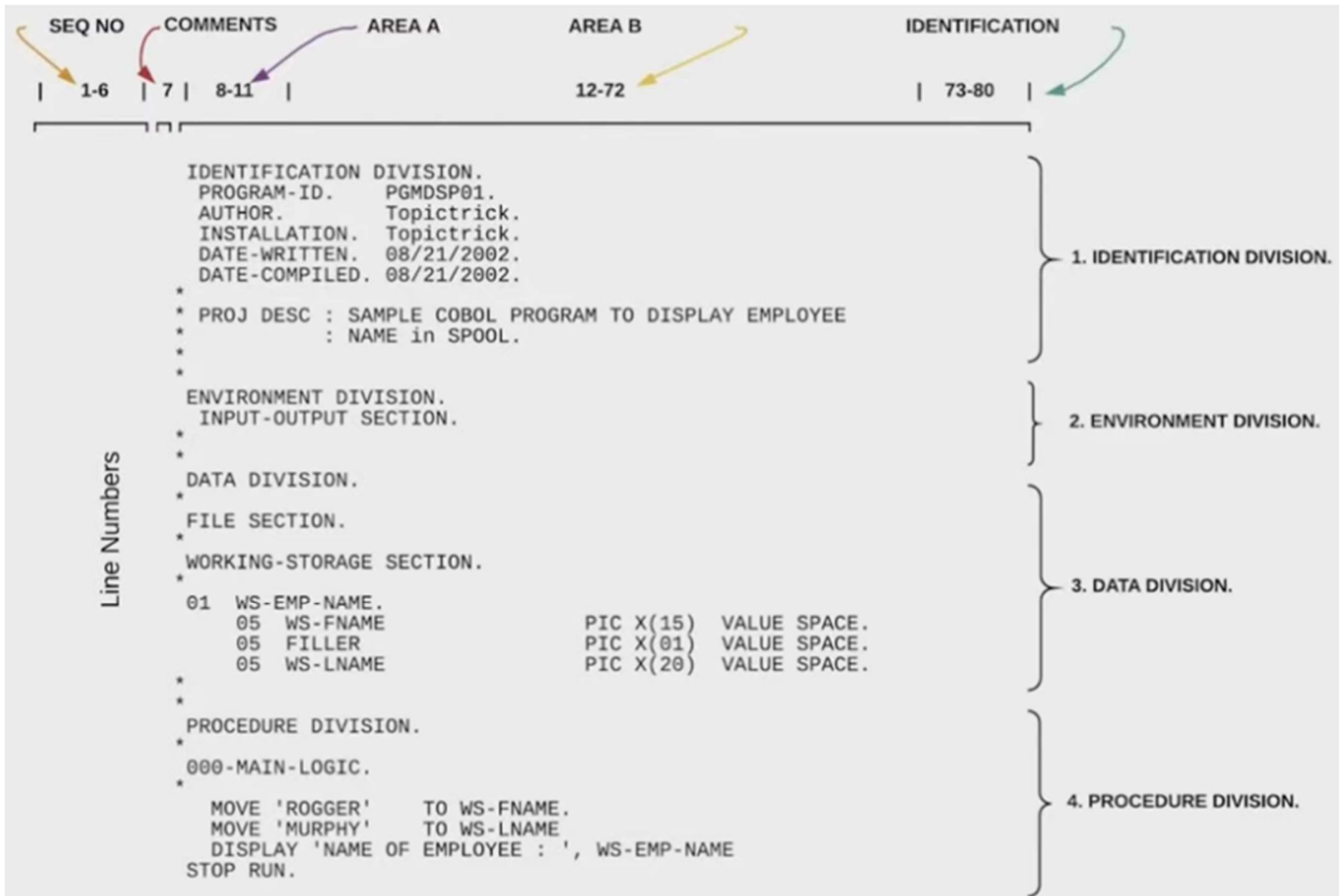
- In the 1960's, computer programs were created on punch cards which were then fed into a computer where the cards were read, the program loaded, and then run.



COBOL Program Formatting

----- 01 - 80 -----				
Sequence No		Section A	Section B	Identification Area
----- 1-6 -----	- 7 -	----- 8-11 -----	----- 12 - 72 -----	---- 73-80 -----
Sequence number is added to each coding line when the program is compiled.	Continuation, Comments, or form feeds.	Used for special entries such as DIVISION, SECTION, Paragraph name and some items of the DATA Division.	Used for most COBOL entries, including PROCEDURE DIVISION sentence.	Used in olden days for identification purpose.

Example COBOL Program



Free Format COBOL

- By default, GnuCOBOL assumes **fixed-format** unless explicitly told otherwise.
- Modern COBOL (especially with COBOL 2002 and later compilers like GnuCOBOL or Micro Focus) supports:
 - No fixed columns
 - No need for sequence numbers or special column positioning
 - Indentation and spacing become stylistic, not required
 - Permits lowercase and modern-style syntax
 - Source files can use **.cbl**, **.cob**, or even **.cobol**



Divisions Of A COBOL Program

Overall Structure Of A COBOL Program

- A COBOL program is divided into 4 logical divisions.
- Each division is then divided into sections, paragraphs, sentences, and statements.
- **Identification Division (MANDATORY)**
 - Used to identify the program to the operating system
 - Used for documentation purpose
- **Environment Division (Optional)**
 - Specify file name and specific computer equipment that will be used by program
- **Data Division (Optional)**
 - Describes input/output formats to be used by program
 - Define constants and work areas
- **Procedure Division (Optional)**
 - Contains business logic to process input and create output

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.      PGMDSP01.  
AUTHOR.          Topictrick.  
INSTALLATION.    Topictrick.  
DATE-WRITTEN.    08/21/2002.  
DATE-COMPILED.   08/21/2002.  
*  
* PROJ DESC : SAMPLE COBOL PROGRAM TO DISPLAY EMPLOYEE  
*           : NAME In SPOOL.  
*  
*  
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
*  
*  
DATA DIVISION.  
*  
FILE SECTION.  
*  
WORKING-STORAGE SECTION.  
01 WS-EMP-NAME.      PIC X(15) VALUE SPACE.  
   05 WS-FNAME.      PIC X(01) VALUE SPACE.  
   05 FILLER.         PIC X(01) VALUE SPACE.  
   05 WS-LNAME.      PIC X(20) VALUE SPACE.  
*  
*  
PROCEDURE DIVISION.  
000-MAIN-LOGIC.  
    PERFORM 005-PRT-EMP-DATA
```

Identification Division

- This is the first division in a COBOL program.
- It's purpose is to identify the program.
- **The IDENTIFICATION Division is MANDATORY for a COBOL program.**
- The paragraph "**PROGRAMID.**" followed by a program name is **MANDATORY.**
 - The PROGRAMID parameter is used to specify the name of the COBOL program. It can consist of 1 to 30 characters.
- All other paragraphs are optional and are used for documentation:
 - AUTHOR
 - DATE-WRITTEN
 - DATE-COMPILED
 - SECURITY

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.      PGMDSP01.  
AUTHOR.          Topictrick.  
INSTALLATION.    Topictrick.  
DATE-WRITTEN.    08/21/2002.  
DATE-COMPILED.   08/21/2002.
```

Environment Division

- This section is related to the program's environment and includes such items as computer, hardware, and files that will be used.
- Division includes:
 - Configuration Section – where the program will be compiled [NOT REQUIRED]
 - SOURCE-COMPUTER
 - OBJECT-COMPUTER
 - Input-Output Section – Files used in the program [REQUIRED]
 - FILE-CONTROL
 - I-O CONTROL

Data Division

- This division lists every data item that will be processed by the program
- The variables that will be used in the PROCEDURE Division need to be declared here.
- Data is divided into two separate types: temporary and permanent.
- Temporary Data
 - Available only during the execution of the program
- Permanent
 - File section
 - Working-storage section
 - Linkage section

WORKING STORAGE Section

- Used for declaring user variables or data names.
- Variable naming standards:
 - Must start with an alphabetic character
 - Name should contain between 1-30 alphanumeric characters
 - No space permitted between characters
 - No reserved words: ex. TIME, ADD, COMPUTE
 - Hyphens can be used but cannot be first character and cannot be consecutive ("--").
 - Not case sensitive: TOTAL and ToTaL are the same variable.
 - No special characters: \$, #, etc.

```
DATA DIVISION.  
WORKING-STORAGE SECTION.  
    01 COUNTER          PIC 9(3) VALUE 0.  
    01 CUSTOMER-NAME PIC X(30).  
    01 TOTAL-AMOUNT  PIC 9(5)V99 VALUE 0.
```

LOCAL STORAGE Section

- The Local-Storage Section is like the Working-Storage Section's "short-term memory."
- Local-Storage is allocated each time the program is called and will be de-allocated when the program stops via an EXIT PROGRAM, GOBACK, or STOP RUN.
- It is defined in the DATA DIVISION after WORKING-STORAGE SECTION
- It's where you declare variables that:
 - Are freshly allocated every time the program, subprogram, or paragraph is entered.
 - Are reinitialized to their defined VALUE (or spaces/zeros if no VALUE) on each entry.
 - Do not retain values between calls.

```
DATA DIVISION.  
LOCAL-STORAGE SECTION.  
    01 TEMP-VAR    PIC 9(3) VALUE 0.  
  
PROCEDURE DIVISION.  
    ADD 1 TO TEMP-VAR  
    DISPLAY "TEMP-VAR is " TEMP-VAR.  
    STOP RUN.
```

Procedure Division

- The **Procedure Division** is the part of the program where the **actual processing logic** is written — the instructions that tell the program what to do with the data.
- It's where you:
 - Control the flow of execution (using PERFORM, IF, EVALUATE, GO TO, etc.).
 - Perform calculations and data manipulation.
 - Handle input/output operations (read from files, write to files, display output).
 - Implement business rules and error handling (including declaratives).

```
PROCEDURE DIVISION.  
MAIN-PARA.  
    OPEN INPUT CUSTOMER-FILE  
    READ CUSTOMER-FILE AT END  
        DISPLAY "No more records"  
    STOP RUN  
END-READ  
PERFORM PROCESS-RECORD  
CLOSE CUSTOMER-FILE  
STOP RUN.  
  
PROCESS-RECORD.  
    DISPLAY "Processing record..."
```

Paragraphs In COBOL Programs

- A "paragraph" is a named section of code that performs a specific task within the a division or a section of the program.
- It is typically used within the PROCEDURE DIVISION.
- It has a name
- It is followed by one or more COBOL statements
- Ends when the next paragraph starts or a STOP-RUN or EXIT is encountered.
- You can use a PERFORM statement to call/execute a paragraph.
- COBOL uses "gravity driven programming"
- Execution of a program "falls through" the program until a condition or a GOTO redirects the flow to a different part of our file.
- An "open paragraph" is an example of how we "fall through code"
- A "closed paragraph" is executed by name

Paragraphs In COBOL Programs

- In a COBOL program, the PROCEDURE Division will consist mainly of a collection of paragraphs.
- These paragraphs are reusable.
- When you run a COBOL program, the program will be executed paragraph by paragraph in sequential order.
- To EXECUTE a specific paragraph, the PERFORM clause is used.

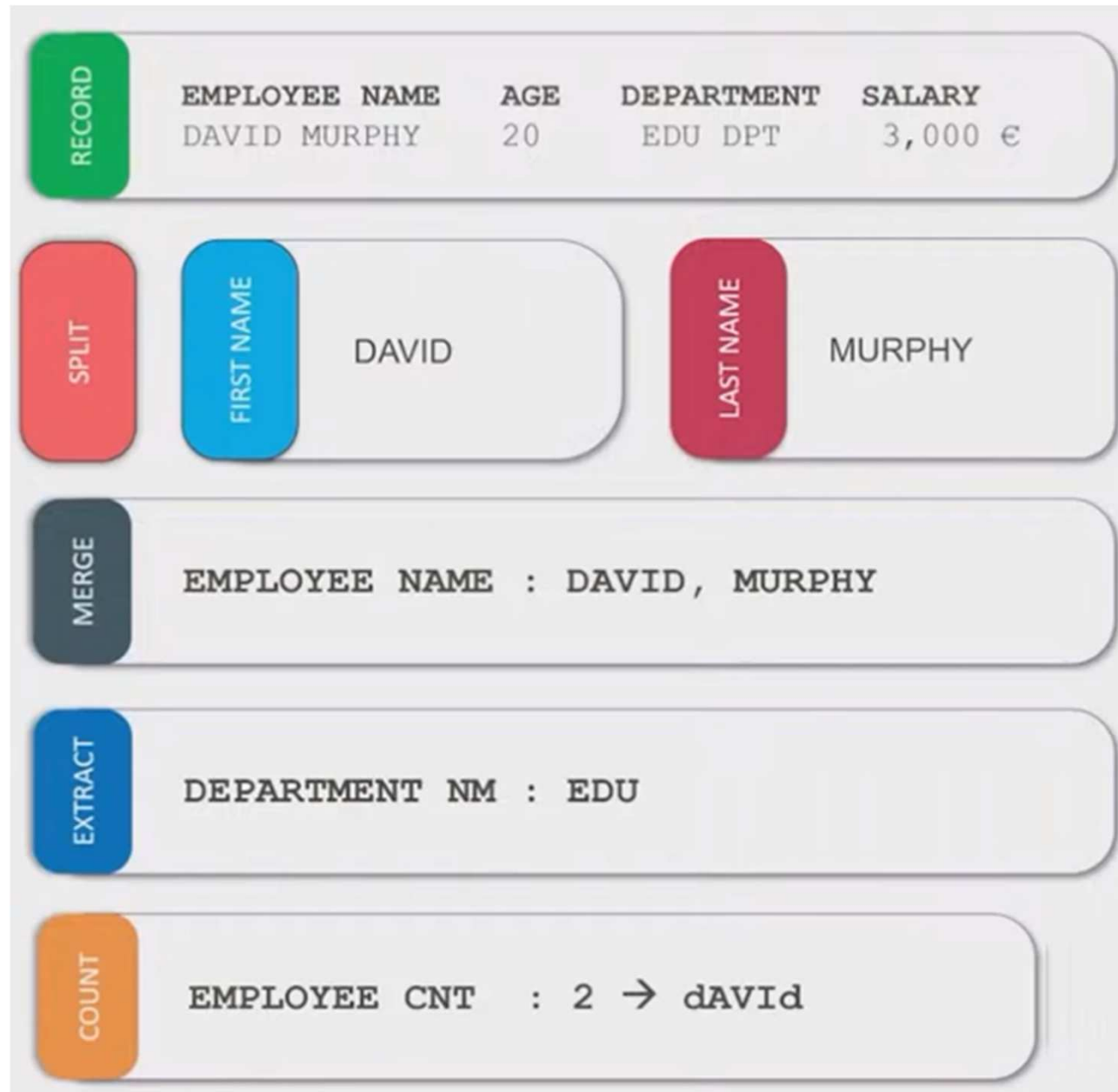
Paragraph Example

```
11 SubOne.  
12     DISPLAY "In Paragraph 1"  
13     PERFORM SubTwo  
14     DISPLAY "Returned to Paragraph 1"  
15     PERFORM SubFour 2 TIMES.  
16     STOP RUN.  
17     END-PERFORM  
18     STOP RUN.  
19  
20 SubThree.  
21     DISPLAY "In Paragraph 3".  
22  
23 SubTwo.  
24     DISPLAY "In Paragraph 2"  
25     PERFORM SubThree  
26     DISPLAY "Returned to Paragraph 2".  
27  
28 SubFour.  
29     DISPLAY  
30  
31 STOP RUN.
```

```
In Paragraph 1  
In Paragraph 2  
In Paragraph 3  
Returned to Paragraph  
2  
Returned to Paragraph  
1  
Repeat
```

Strings

COBOL String Manipulation



COBOL String Handling

- **STRING Clause:** String statements concatenate two or more sending fields into one receiving field.
 - Key words used with the STRING clause:
 - STRING
 - DELIMITED BY
 - WITH POINTER
 - END STRING
- **UNSTRING Clause:** the UNSTRING statement unstrings a field into the fields that are listed in the INTO clause.
- **INSPECT Clause:** The INSPECT statement allows you to count characters in a field or replace characters in a field.
- **REFERENCE Modification:** The REFERENCE modification feature lets you refer to a specific location within a field.

String Example

Means copy the entire variable
- all 20 characters, even blanks

```
WORKING-STORAGE SECTION.  
  01 WS-CUST-NAME      PIC X(40).  
  01 WS-CUST-FIRST-NM  PIC X(20) VALUE IS "VIJAY".  
  01 WS-CUST-LAST-NM   PIC X(20) VALUE IS "KUMAR".  
  01 WS-CUST-POINT     PIC 9(2) VALUE 05.  
PROCEDURE DIVISION.  
  STRING  
  WS-CUST-FIRST-NM , WS-CUST-LAST-NM  
  DELIMITED BY SIZE  
  INTO  
  WS-CUST-NAME  
  WITH POINTER WS-CUST-POINT  
  END-STRING.  
  DISPLAY WS-CUST-NAME.
```

Start placing "Vijay Kumar" into WS-CUST-NAME
at location 5.

Editing Strings

```
8  WORKING-STORAGE SECTION.  
9  01 StartNum PIC 9(8)V99 VALUE 00001123.55.  
10 01 NoZero PIC ZZZZZZZ9.99.  
11 01 NoZPlusC PIC ZZ,ZZZ,ZZ9.99.  
12 01 Dollar PIC $$,$$$,$$9.99.  
13 01 BDay PIC 9(8) VALUE 12211974.  
14 01 ADate PIC 99/99/9999.  
15 PROCEDURE DIVISION.  
16 MOVE StartNum TO NoZero  
17 DISPLAY NoZero  
18 MOVE StartNum TO NoZPlusC  
19 DISPLAY NoZPlusC  
20 MOVE StartNum TO Dollar  
21 DISPLAY Dollar  
22 MOVE BDay TO ADate  
23 DISPLAY ADate  
24 STOP RUN.
```

```
1123.55  
1,123.55  
$1,123.55  
12/21/1974
```

Editing Strings: Starting Variables

```
8  WORKING-STORAGE SECTION.  
9  01 SampStr      PIC X(18) VALUE 'eerie beef sneezed'.  
10 01 NumChars     PIC 99 VALUE 0.  
11 01 NumEs        PIC 99 VALUE 0.  
12 01 FName        PIC X(6) VALUE 'Martin'.  
13 01 MName        PIC X(11) VALUE 'Luther King'.  
14 01 LName        PIC X(4) VALUE 'King'.  
15 01 FLName       PIC X(11).  
16 01 FMLName      PIC X(18).  
17 01 SStr1        PIC X(7) VALUE "The egg".  
18 01 SStr2        PIC X(9) VALUE "is #1 and".  
19 01 Dest         PIC X(33) VALUE "is the big chicken".  
20 01 Ptr          PIC 9 VALUE 1.  
21 01 SStr3        PIC X(3).  
22 01 SStr4        PIC X(3).
```

String Processing

```
9 01 SampStr    PIC X(18) VALUE 'eerie beef sneezed'.
10 01 NumChars  PIC 99 VALUE 0.
11 01 NumEs     PIC 99 VALUE 0.
12 01 FName     PIC X(6) VALUE 'Martin'.
13 01 MName     PIC X(11) VALUE 'Luther King'.
14 01 LName     PIC X(4) VALUE 'King'.
```

```
26 INSPECT SampStr TALLYING NumEs FOR ALL 'e'.
27 DISPLAY "Number of Es : " NumEs.
28 DISPLAY FUNCTION UPPER-CASE(SampStr)
29 DISPLAY FUNCTION LOWER-CASE(SampStr)
30
31 STRING FName DELIMITED BY SIZE
32 SPACE
33 LName DELIMITED BY SIZE
34 INTO FLName.
35 DISPLAY FLName.
36 STOP RUN.
```

```
Number of Characters
: 18
Number of Es : 08
EERIE BEEF SNEEZED
eerie beef sneezed
Martin King
```

Means "get the whole thing"

String Processing

```
12 01 FName      PIC X(6) VALUE 'Martin'.
13 01 MName      PIC X(11) VALUE 'Luther King'.
14 01 LName      PIC X(4) VALUE 'King'.
15 01 FLName     PIC X(11).
16 01 FMLName    PIC X(18).
```

```
37 STRING FLName DELIMITED BY SPACES
38 SPACE
39 MName DELIMITED BY SIZE
40 SPACE
41 LName DELIMITED BY SIZE
42 INTO FMLName
43 ON OVERFLOW DISPLAY 'Overflowed'.
44 DISPLAY FMLName.
```

```
Overflowed
Martin Luther King
```

String Processing

```
17 01 SStr1      PIC X(7) VALUE "The egg".  
18 01 SStr2      PIC X(9) VALUE "is #1 and".  
19 01 Dest       PIC X(33) VALUE "is the big chicken".
```

```
46 STRING SStr1 DELIMITED BY SIZE  
47 SPACE  
48 SStr2 DELIMITED BY "#"  
49 INTO Dest  
50 WITH POINTER Ptr  
51 ON OVERFLOW DISPLAY 'overflowed'.  
52 DISPLAY Dest.
```

The egg is chicken

String Processing

```
17 01 SStr1      PIC X(7) VALUE "The egg".
18 01 SStr2      PIC X(9) VALUE "is #1 and".
19 01 Dest       PIC X(33) VALUE "is the big chicken".
20 01 Ptr        PIC 9 VALUE 1.
21 01 SStr3      PIC X(3).
22 01 SStr4      PIC X(3).
```

```
54 UNSTRING SStr1 DELIMITED BY SPACE
55 INTO SStr3, SStr4
56 END-UNSTRING.
57 DISPLAY SStr4.
```

egg

Numbers In COBOL

Number Accuracy In COBOL

- Most computer languages used floating point calculations.
- Problem: this can introduce errors due to how computers represent real numbers. Example: $01.+0.2 = 0.300000000000000004$
- COBOL uses fixed point decimal arithmetic.
- Fixed point decimal arithmetic is a method of representing numbers that have a fixed number of digits after the decimal point.
- Fixed point keeps the decimal point in a constant position.
- Fixed point is precise and quantifiable especially for money and quantities.
- COBOL allows you to specify how you want to round values

Financial Calculations

```
8 WORKING-STORAGE SECTION.  
9 01 Price PIC 9(4)V99.  
10 01 TaxRate PIC V999 VALUE .075.  
11 01 FullPrice PIC 9(4)V99.  
12 PROCEDURE DIVISION.  
13 DISPLAY "Enter the Price : " WITH NO ADVANCING  
14 ACCEPT Price  
15 COMPUTE FullPrice ROUNDED = Price + (Price *  
    TaxRate)  
16 DISPLAY "Price + Tax : " FullPrice.  
17 STOP RUN.
```

```
Enter the Price : 4567  
.98  
Price + Tax : 4910.49
```

COBOL Variable Type Checking

- COBOL does not perform strict type checking in the way modern strongly-typed languages like Java or C++ do.
- Instead, COBOL uses a concept called category matching and level number compatibility to determine whether variables are compatible during operations like MOVE, arithmetic operations, or procedure calls.
- Category Matching COBOL groups data items into categories such as: Alphabetic (PIC A), Alphanumeric (PIC X), Numeric (PIC 9), Decimal or Floating-point.
 - When using MOVE, COMPUTE, or arithmetic verbs, COBOL checks if the source and target categories are compatible, not whether their types match exactly.
- Level Numbers: COBOL uses level numbers (01, 05, 77, etc.) to define structure.
 - Operations are valid if level structure allows (e.g., moving a group item to another of compatible structure).

What COBOL Doesn't Strictly Enforce

- You can often move an alphanumeric variable to a numeric one, and the compiler will convert or truncate the data at runtime.
- You can pass variables with different PIC clauses to a subprogram and the compiler won't complain unless CALL is explicitly ... USING BY VALUE or BY CONTENT.
- Example:

```
01 WS-NUMBER  PIC 9(4).  
01 WS-TEXT   PIC X(4).  
MOVE WS-TEXT TO WS-NUMBER.
```
- This will compile, but may cause runtime issues if WS-TEXT doesn't contain digits.

Arithmetic Operations

Arithmetic Operators			
Binary	Meaning	Unary	Meaning
+	Addition	+	Multiplication by +1
-	Subtract	-	Multiplication by -1
*	Multiplication		
/	Division		
**	Exponentiation		

← Safer: COMPUTE RESULT = FUNCTION POW(A,B)

```
15 COMPUTE FullPrice ROUNDED = Price + (Price *  
    TaxRate)
```

```
ADD 1 TO Ind.
```

Arithmetic Program

```
22 01 Num1 PIC 9 VALUE 5.  
23 01 Num2 PIC 9 VALUE 4.
```

```
50 ADD Num1 TO Num2 GIVING Ans  
51 DISPLAY Ans  
52 SUBTRACT Num1 FROM Num2 GIVING Ans  
53 DISPLAY Ans  
54 MULTIPLY Num1 BY Num2 GIVING Ans  
55 DISPLAY Ans  
56 DIVIDE Num1 INTO Num2 GIVING Ans  
57 DISPLAY Ans  
58 DIVIDE Num1 INTO Num2 GIVING Ans REMAINDER  
   Rem  
59 DISPLAY "Remainder " Rem
```

```
+09.00  
-01.00  
+20.00  
+00.80  
Remainder 0.00
```

Handle errors:

```
ADD WS-A, WS-B, WS-ITEM GIVING WS-TOT  
ON SIZE ERROR  
DISPLAY "NO MEMORY".
```

Another Arithmetic Program

```
50 ADD Num1, Num2 TO Num3 GIVING Ans
51 ADD Num1, Num2, Num3 GIVING Ans
52 DISPLAY Ans
53 COMPUTE Ans = Num1 + Num2
54 COMPUTE Ans = Num1 - Num2
55 COMPUTE Ans = Num1 * Num2
56 COMPUTE Ans = Num1 / Num2
57 DISPLAY Ans
58 COMPUTE Ans = Num1 ** 2
59 DISPLAY Ans
60 COMPUTE Ans = (3 + 5) * 5
61 DISPLAY Ans
62 COMPUTE Ans = 3 + 5 * 5
63 DISPLAY Ans
64 COMPUTE Ans ROUNDED = 3.0 + 2.005
65 DISPLAY Ans
```

```
+12.00
+01.25
+25.00
+40.00
+28.00
+05.01
```

Variables

Constants

- **Def**: a constant is a data item that has only one value.
- Example:

```
19 01 PIValue CONSTANT AS 3.14.
```

Figurative Constants

- **Def:** Figurative constants are reserved words that refer to specific constant values that have been defined by the compiler.

- List of figurative constants:

- ZERO

- ZEROS

- SPACE

- SPACES

- HIGH-VALUE

- LOW-VALUE

- QUOTE

- NULL

- ALL

Functionally identical.

The difference is purely stylistic or contextual—not semantic.

```
WORKING-STORAGE SECTION.
```

```
. . .
```

```
01 WS-DPT-NME          PIC X(30)          VALUE SPACES.
```

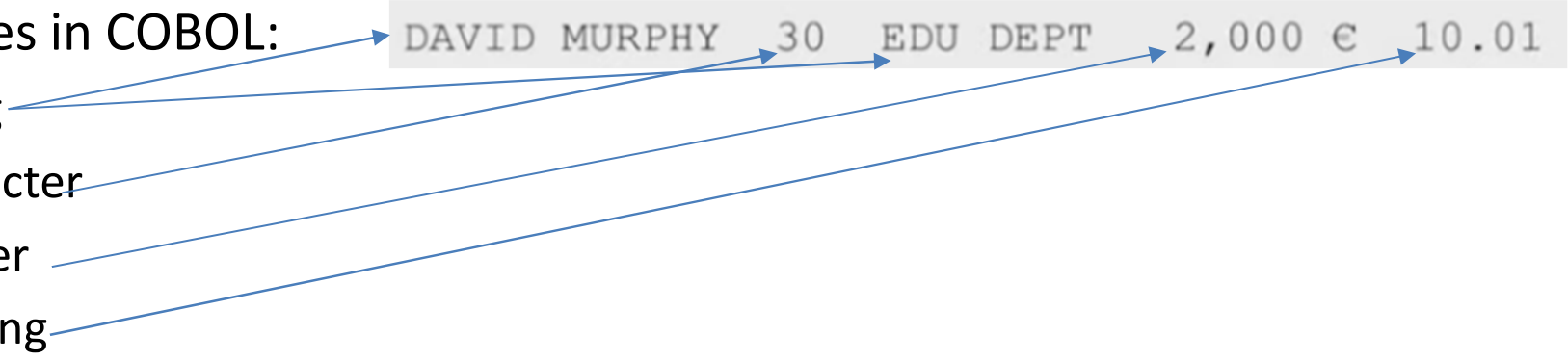
```
. . .
```

```
01 WS-MORT-INT-RTE     PIC 9(01)V9(04)     VALUE ZEROES.
```

```
. . .
```

```
01 WS-EMPNO           PIC 9(06)           VALUE ZERO.
```

COBOL Data Types

- Data types in COBOL:
 - String
 - Character
 - Integer
 - Floating
 - Boolean
- As in Java, C, etc. variables have to have their data type declared prior to being used.

COBOL Data Types

- **Alphabetic** – made up of uppercase/lowercase letters [DAVE MURRY]
Maximum size is 35,535 characters.
- **Numeric** – made up of digits [30].
Maximum number of digits is 18.
- **Alphanumeric** – made up of letters and digits [\$2000].
Maximum size is 35,535 characters.

Variables

- Variables are "Data-Name" or "Data Item"
- Variable names can contain letters, digits, and "-"
- Maximum variable name size: 30 characters
- Four pieces of information required to define a variable:
 - Level Number
 - Variable name
 - PICTURE Clause
 - Data type

Variable Level Number

- In COBOL, level numbers specify the hierarchy of data within a record and identify special-purpose data entries.
- A level number begins a data description entry and has value taken from the set of integers between 1 and 49, or from one of the special-level numbers: 66, 77, or 88.
- The following table shows the significances of the various level numbers that can be used with data items:

LEVEL NUMBERS	
Level Number	Meaning
01	For record description
02-49	For fields with records
01 / 77	For independent items
66	For RENAMEs clause
88	For condition names

77 Level Numbers

- 77 is only used for independent data names.
- An independent data name refers to a standalone data item.
 - A variable that is not part of any group structure.
 - It is not subordinate to another field.
 - Does not contain subordinate fields.
 - Has a PIC clause.
 - Holds an actual value.
- Level 77 is valid but rarely used in modern COBOL.
- Most developers just define all data items under 01 level and use sublevels as needed for grouping.
- Note that ANY level is the same as another level assuming that they don't have any subordinate items.

```
01 EMP-NAME          PIC X(35) .  
77 EMP-ADD           PIC X(40) .
```

COBOL Level 88 Conditions

- A condition name is a name that refers to a condition.
- To define a condition name, you use an 88 level in the Data Division.
- Once defined, the condition name can be used as the condition in an IF, PERFORM UNTIL, or EVALUATE statement.
- Condition names are frequently used with switches and flags.
- A condition name is always coded on the 88 level and only has a VALUE clause associated with it.
- Since a condition name is NOT a name of a field, it will not contain a PICTURE clause.
- The condition name must be unique and its VALUE must be a literal constant with the data type of the field proceeding it.

COBOL Level 88 Conditions Example

Syntax – **Condition Names Syntax.**

```
88 condition-name VALUE IS {literal-1 [THRU] literal-2}
```

```
*  
WORKING-STORAGE SECTION.  
05 MARITAL-STATUS PIC X(01)  
  88 SINGLE VALUE 'S'.  
  88 MARRIED VALUE 'M'.  
  88 DIVORCED VALUE 'D'.  
...  
*  
PROCEDURE DIVISION.  
*  
A001-MAIN-LOGIC.  
  ...  
  ...  
  SET DIVORCED TO TRUE  
  IF DIVORCED THEN  
    DISPLAY 'DIVORCED'  
  END-IF  
  
  IF MARTIAL-STATUS = 'S' THEN  
    DISPLAY 'SINGLE'  
  END-IF
```

Variable can take on one of three different condition names.

Variable can have a value of a single character: S/M/D.

88 means that these are condition names – not elements

COBOL Level 88 Conditions Example

```
WORKING-STORAGE SECTION.  
77 WS-MARITAL-STATUS PIC 9.  
88 WS-SINGLE VALUE 0.  
88 WS-MARRIED VALUE 1.  
88 WS-DIVORCE VALUE 2.  
88 WS-WIDOWED VALUE 3.  
77 WS-AMOUNT PIC 9(3) VALUE 100.  
PROCEDURE DIVISION.  
EMPLOYEE-TAX-CAL.  
    ACCEPT WS-MARITAL-STATUS.  
    IF WS-SINGLE SUBTRACT 10 FROM WS-AMOUNT.  
    IF WS-MARRIED SUBTRACT 100 FROM WS-AMOUNT.  
    IF WS-DIVORCED SUBTRACT 20 FROM WS-AMOUNT.  
    DISPLAY WS-AMOUNT.  
    STOP RUN.
```

Group & Elementary Variables

- Group Data Names
 - A group data name refers to a logical grouping of related fields (elementary items) under a single name.
 - Similar to a STRUCT or RECORD in other languages.
 - These variables are declared without a PIC class.
 - The first group level is always 01.
 - Can contain other group data names
 - Can be moved as a unit
- Elementary Data names
 - They should always use data levels that are greater than 01
 - Example:
 - 01 WS-DATE
 - 02 WS-YYYY PIC 9(4)
 - 02 WS-MM PIC 9(2)
 - 02 WS-DD PIC 9(2)

COBOL Group Items

- In COBOL, a Group Items consists of one or more elementary items (data names). A group item is described by:
 - Level number
 - Data name
 - Value clause
- The level numbers must be between 01-49. Typically you will start with 01, and then use multiples of 5: 5, 10, 15, etc.
- Level 01 items must be in the A margin. Other level numbers can begin in the A or B margins.
- You can't code a Picture clause for a group item. Instead, you should to code a Picture clause for an elementary item.
- A group item will always be treated as an alphanumeric item, no matter how the elementary items beneath it are defined.

COBOL Group Items Examples

WORKING-STORAGE SECTION.

01 WS-TEMP-VAL.

05 WS-REC-LN PIC 99.

05 WS-EMP-REC-IN PIC 9(06).

05 WS-ERR-MSG PIC X(12).

05 WS-TOT-CR-AMT PIC S9(08)V99.

05 WS-TOT-DR-AMT PIC S9(08)V99.

05 WS-EMP-DPT PIC A(30).

01 WS-EMPL-SALARY PIC Z,ZZZ,ZZZ.99.

Group variable

Elementary
Item

WORKING-STORAGE SECTION.

01 WS-EMP-REC.

05 WS-EMP-NO PIC 9(06).

05 WS-EMP-NAME.

10 WS-INITL PIC XX VALUE SPACES.

10 WS-EMP-FNAME PIC X(12) VALUE SPACES.

10 WS-EMP-LNAME PIC X(14) VALUE SPACES.

05 WS-EMP-DEPT PIC A(13).

05 WS-EMP-SAL PIC S9(10)V9(02).

77 TOT-REJ-REC-CNT PIC 9(05).

01 TOT-REC-CNT PIC 9(05) VALUE ZEROES.

Group variable

Note: To make the structure of the group variables easy to read, you should align the levels as shown in these examples.

Variable Level Number

"Elements"

Group variable
record description

Subrecord to
WS-TEMP-VAL

Condition names

```
WORKING-STORAGE SECTION.
01 WS-TEMP-VAL.
   05 WS-REC-LN          PIC 99.
   05 WS-EMP-REC-IN      PIC 9(06).
   05 WS-ERR-MSG         PIC X(12).
01 WS-END-OF-FILE       PIC X      VALUE 'N'.
88 END-OF-FILE           VALUE 'Y'.
88 NOT-END-OF-FILE      VALUE 'N'.
```

Group variable
record description

```
WORKING-STORAGE SECTION.
01 WS-EMP-REC.
   05 WS-EMP-NO          PIC 9(06).
   05 WS-EMP-NAME.
      10 WS-EMP-FNAME    PIC X(12)  VALUE SPACE.
      10 WS-INITL        PIC XX.
      10 WS-EMP-LNAME    PIC X(14).
   05 WS-EMP-DEPT        PIC A(13).
   05 WS-EMP-SAL         PIC S9(10)X9(02).
77 TOT-REJ-REC-CNT      PIC 9(05).
```

Independent Item
(not a group variable)

Note:

- Level 01 & 77 must begin in area A
- Levels 02 – 49 can be in Areas A or B
- Level 66 & 88 can be in Areas A or B

Picture Clause

- The Picture clause specifies the data type and the amount of storage that is required for a data item. It is denoted by PICTURE, often abbreviated as PIC.
- A PICTURE clause is specified only for elementary data items and consists of picture characters. Each picture character denotes storage to be reserved for a character of that type.
- The following are the general picture characters and their meaning:

PICTURE Clause		
Item Type	Characters	Meaning
Alphabetic	A	Alphabetic
Alphanumeric	X	Any character
Numeric	9	Digits
	S	Sign
	V	Assumed decimal point
Numeric Edited	9	Digit
	Z	Zero suppress digit
	,	Inserted comma
	.	Inserted decimal
	-	Minus sign if negative

Picture Clause Example

```
WORKING-STORAGE SECTION.  
01 WS-TEMP-VAL.  
   05 WS-REC-LN          PIC 99.  
   05 WS-EMP-REC-IN      PIC 9(06).  
   05 WS-ERR-MSG         PIC X(12).  
   05 WS-TOT-CR-AMT      PIC S9(08)V99.  
   05 WS-TOT-DR-AMT      PIC S9(08)V99.  
   05 WS-EMP-DPT         PIC A(30).  
01 WS-EMPL-SALARY       PIC Z,ZZZ,ZZZ.99.
```

Group variable

Store 0 - 99

Store 0 – 999,999

Store +/-000000000.00

Store 1.35

Sign determines if sign will be displayed when printed: +/-

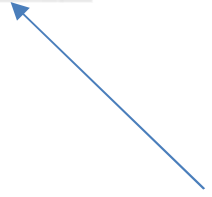
Picture Clause Example

```
WORKING-STORAGE SECTION.  
01 WS-EMP-REC.  
    05 WS-EMP-NO          PIC 9(06).  
    05 WS-EMP-FNAME       PIC X(12)  VALUE SPACE.  
    05 WS-INITL           PIC XX.  
    05 WS-EMP-LNAME       PIC X(14).  
    05 WS-EMP-DEPT        PIC A(13).  
    05 WS-EMP-SAL         PIC S9(10)V9(02).  
77 TOT-REJ-REC-CNT       PIC 9(05).  
01 TOT-REC-CNT           PIC 9(05)  VALUE ZEROES.
```

Initialized to space



Initialized to zero



Note:

- Alphanumeric items: unused spaces to the right are set to spaces
- Numeric items: unused spaces to the left are set to "0"

Picture Clause Examples

```
01 W-EDITING-CHAR.
* ZERO SUPRESING ZERO REPLACED WITH SPACE-> IN(0123) OUT( 123)
  02 WS-Z999          PIC Z999.
  02 WS-ZZ999         PIC ZZ999.
  02 WS-ZZZ999        PIC ZZZ999.
* USING ** SYMBOLS -> IN (00123) OUT (**123)
  02 WS-ASTERIC       PIC **999.
* USING $ SYMBOLS -> IN (123.25 ) OUT ( $123.25 )
  02 WS-DOLLAR        PIC $999.99.
* USING MINUS SYMOBOL -> IN(1234) OUT (-1234/1234-)
  02 WS-MINUS-L       PIC -9999.
  02 WS-MINUS-R       PIC 9999-.
* USING PLUS SYMOBOL -> IN(1234) OUT(+1234/1234+)
  02 WS-PLUS-L        PIC +9999.
  02 WS-PLUS-R        PIC 9999+.
```

Note: unlike using S, the +/- will always be printed

```
USING CREDIT & DEBIT SYMBOL -> IN(-1234) OUT(1234CR/1234DR)
  02 WS-CR            PIC 999CR.
  02 WS-DR            PIC 999DB.
USING DOT IN(123.44) OUT(123.44)
  02 WS-DOT           PIC 9(3).9(2).
USING , IN(12345) OUT(12,345)
  02 WS-CAMA          PIC 999,99.
USING BLANK IN(12345) OUT(12 345)
  02 WS-BLANK         PIC 99B999.
USING ZERO IN(123) OUT(12300)
  02 WS-ZERO          PIC 99900.
  02 WS-ZERO-F        PIC 00999.
USING / SLASH IN(07072020) OUT (07/07/2020)
  02 WS-SLASH         PIC 99/99/9999.
```

Note: not supported by all COBOL compilers. V is a better choice

YouTube COBOL References

- COBOL 101
<https://www.youtube.com/watch?v=cnz9y9k2jvs>
- COBOL Tutorial : Learn COBOL in One Video
<https://www.youtube.com/watch?v=TBs7HXI76yU>
- Complete COBOL Refresher in 1 Hour #COBOL
<https://www.youtube.com/watch?v=38pNOuGiSmw>
- COBOL Lesson 1 – Introduction
<https://www.youtube.com/watch?v=u9M52sAnrOs>