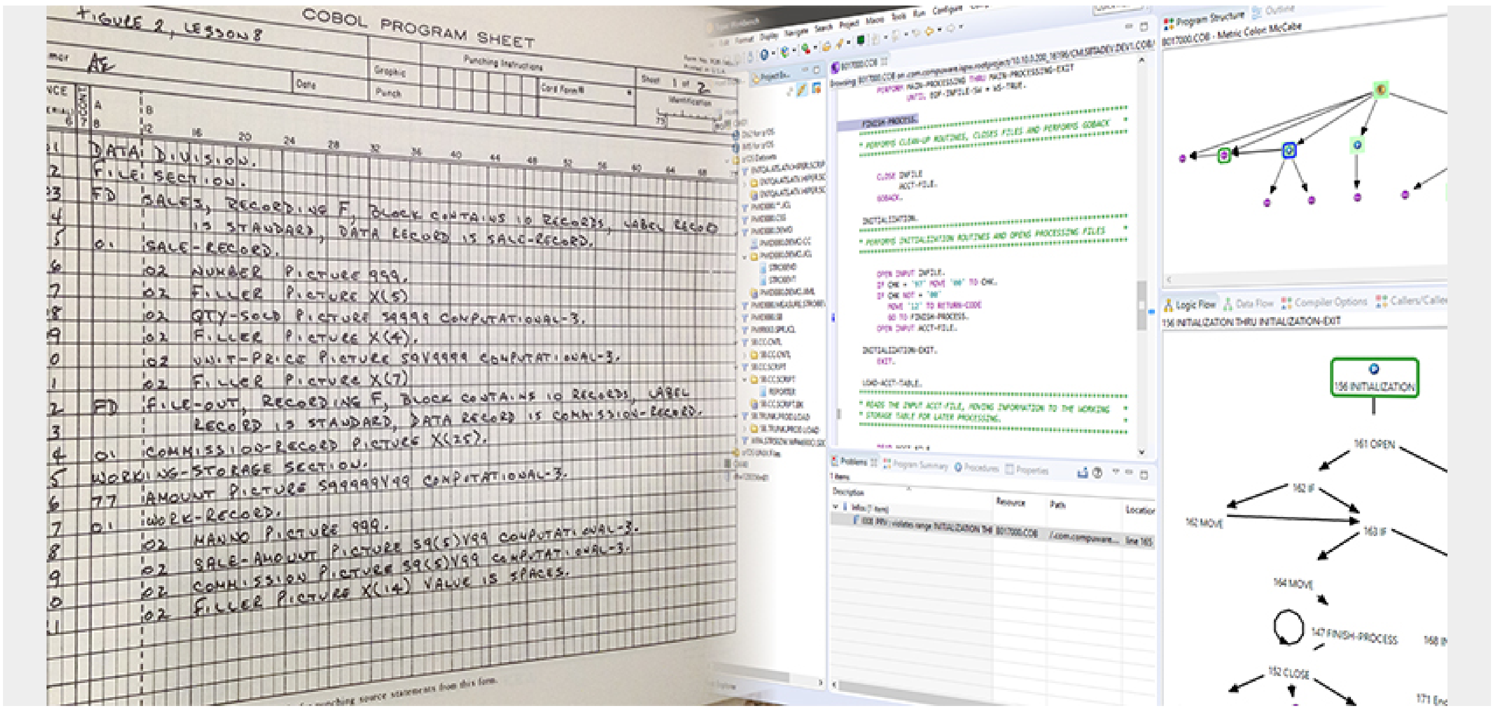
TWO GENERATIONS OF COBOL: MY GRANDMA AND ME



```
DATA DIVISION.
FILE SECTION.
FD  SALES, RECORDING F, BLOCK CONTAINS 10 RECORDS, LABEL RECORD
        IS STANDARD, DATA RECORD IS SALE-RECORD.
01  SALE-RECORD.
    02  NUMBER  PICTURE 999.
    02  FILLER  PICTURE X(5)
    02  QTY-SOLD PICTURE S9999 COMPUTATIONAL-3.
    02  FILLER  PICTURE X(4).
    02  UNIT-PRICE PICTURE S9V9999 COMPUTATIONAL-3.
    02  FILLER  PICTURE X(7)
FD  FILE-OUT, RECORDING F, BLOCK CONTAINS 10 RECORDS, LABEL
        RECORD IS STANDARD, DATA RECORD IS COMMISSION-RECORD.
01  COMMISSION-RECORD PICTURE X(25).
WORKING-STORAGE SECTION.
77  AMOUNT PICTURE S99999V99 COMPUTATIONAL-3.
01  WORK-RECORD.
    02  MANNO PICTURE 999.
    02  SALE-AMOUNT PICTURE S9(5)V99 COMPUTATIONAL-3.
    02  COMMISSION PICTURE S9(5)V99 COMPUTATIONAL-3.
    02  FILLER PICTURE X(14) VALUE IS SPACES.
```

```
           PERFORM MAIN-PROCESSING THRU MAIN-PROCESSING-EXIT
               UNTIL EOF-INFILE-SW = 88-TRUE.

       FINISH-PROCESS.
      *****************************************************************
      * PERFORMS CLEAN-UP ROUTINES, CLOSES FILES AND PERFORMS GOBACK  *
      *****************************************************************

           CLOSE INFILE
                 ACCT-FILE.
           GOBACK.

       INITIALIZATION.
      *****************************************************************
      * PERFORMS INITIALIZATION ROUTINES AND OPENS PROCESSING FILES   *
      *****************************************************************

           OPEN INPUT INFILE.
           IF CHK = '97' MOVE '00' TO CHK.
           IF CHK NOT = '00'
              MOVE '12' TO RETURN-CODE
              GO TO FINISH-PROCESS.
           OPEN INPUT ACCT-FILE.

       INITIALIZATION-EXIT.
           EXIT.

       LOAD-ACCT-TABLE.
      *****************************************************************
      * READS THE INPUT ACCT-FILE, MOVING INFORMATION TO THE WORKING  *
      * STORAGE TABLE FOR LATER PROCESSING.                           *
      *****************************************************************
```

Before I worked in mainframe, my only exposure to COBOL was when it was briefly mentioned in the history section of my first computer engineering class. My grandmother wrote COBOL so it must be ancient, right?

I never expected that people were still out there using COBOL, much less that I would use it in my career. Working with the same technology that my grandma did 45 years ago has given me a newfound respect for her work and the impact of COBOL as a programming language.

In almost the same way I stumbled into mainframe, my grandma stumbled into programming without knowing much about it. She was an English major, but she wanted something different from her job as a newspaper editor. She took some programming classes because it was new, exciting and in demand. As she put it, "I thought of programming as just learning another 'language,' which I was good at."

After getting her degree, she did a little freelance programming and then got hired at a mail order catalog company (those were still a thing back then) in the early 1980s. They had data on hundreds of thousands of customers stored on 14 reels of tape. Her responsibility was to write COBOL programs that would sort and filter that customer data so it could be used in targeted marketing.

It seems trivial now, but this process was actually quite innovative for the time. Before computers, it just wasn't feasible to sort through that many customer records efficiently enough to do anything useful. Their shiny new IBM® machine allowed them to do amazing things, but we've come a long way since then.

Today, the data would probably be stored on super-efficient hard disk drives instead of reels of tape. If the data were organized into a Db2 database, Grandma's complex sorting and filtering programs could probably be simplified with a few well-crafted SQL queries. It could also be set up with encryption and redundant backups for security and data integrity, all automatically managed by z/OS®.

Of course, our programs run much faster on modern machines, but we can also write those programs faster as well. Grandma had to LITERALLY WRITE her programs on paper. When they were finished, she would send it off to another person that would punch it in for her (yes, they used punch cards then). She received a stack of cards back days later. Because the keypunchers sometimes made mistakes, she had to proofread that stack of cards before even attempting to compile it.

When I'm programming, I compile countless times a day to check my work and verify my program. I just click one button and get my results in seconds. In fact, with static code analysis features, I am sometimes notified of errors as I'm programming, before I hit that button to compile. When my program is finished, I have automated tests I can run that will verify it before sending it out. Grandma just had to cross her fingers and wait for the "IT WORKED!" fax. According to her, "No matter how much you checked everything, you were always a little nervous when it went out."

With over 50 years of updates for ease of use, efficiency, and scalability, COBOL isn't the same today as it was when my grandma was programming. Back then, it didn't even support basic features like pointers, floating-point and binary arithmetic, or recursion. Now, COBOL supports all of that plus more modern features like containers, function overloading, automatic compile-time optimization, and much more.

COBOL has features that compete against some of the trendiest and most advanced languages out there. However, one feature in particular is responsible for COBOL's staying power. COBOL was always designed with backward compatibility in mind to ensure your programs always work. COBOL programs initially written decades ago are still doing their job today. For example, the modern version of COBOL could still compile Grandma's old programs with minimal refactoring. She'd probably want to take advantage of those new features (and she'd definitely want to copy them off of those punch cards), but it would work even if she didn't.

COBOL on the mainframe works, has worked for the past 65 years, and will continue to work into the future. While there is still COBOL code out there, there will be tools to help streamline its development, testing, and maintenance. I bet COBOL developers of the future will have fancy new tools that multiply their productivity like the modern IDE has done for me. I'll sound like Grandma with her punch cards when I reminisce about when we had to type our programs out, rather than having AI read your mind or something. Maybe someone will learn about the work I did, and it will inspire them like my grandma has inspired me.