CBL1

COBOL Gets it Done

Tight and tidy code with precision

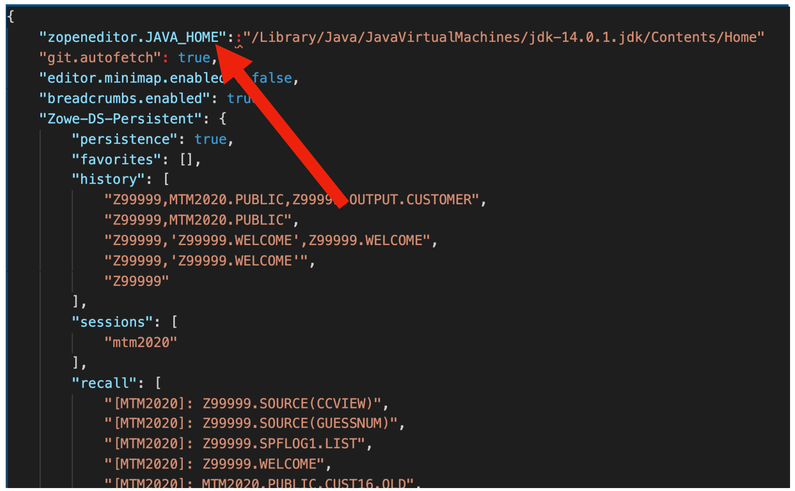
12 steps 6 hours+

THE CHALLENGE

COBOL is a language written specifically to handle high-precision business data. That means bank account balances, credit card transactions, interest calculations, paychecks, insurance statements... if it needs to be fast, accurate, and dependable, chances are it's been written in COBOL. We'll crack open a working COBOL program, mess around with some of the code, and then craft our own program. Being able to add COBOL to your own personal list of talents will definitely get you attention.

BEFORE YOU BEGIN

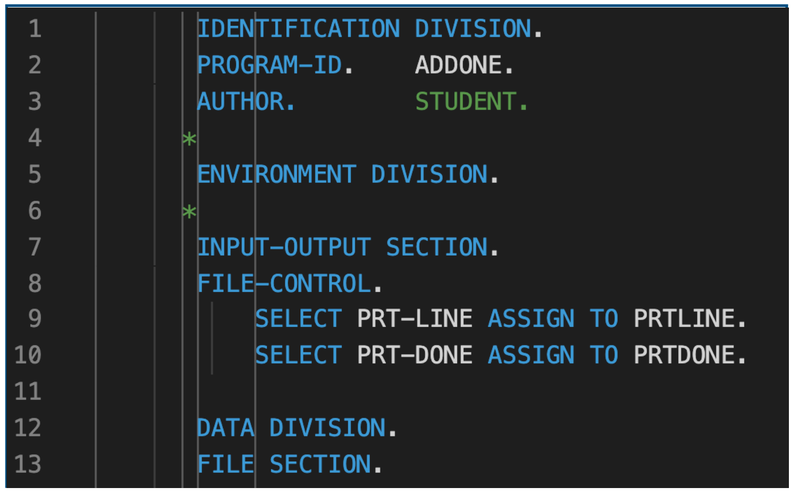
If you're in Level 3, you're ready for this challenge. Nothing besides the fundamentals (VS Code, data sets, JCL) is required.



1. INSTALL JAVA

To use the COBOL enhancements in the IBM Z Open Editor (and trust me, you want them), you need to have a Java JDK installed. Not just a JRE, but a full JDK. You have options, you can use Oracle Java JDK 8, or version 8 or 11 of the OpenJDK.

Either way, once installed, check out the instructions on the IBM Z Open Editor for configuration instructions. You'll need to set the **"zopeneditor.JAVA_HOME"** variable in your settings.json file. To get there, go to your list of extensions, and click the gear next to IBM Z Open Editor, and go down to Extension Settings. Follow any link in there to edit the settings.json file, and just make sure you're putting the JAVA_HOME entry in a valid location (indicated in the instructions)

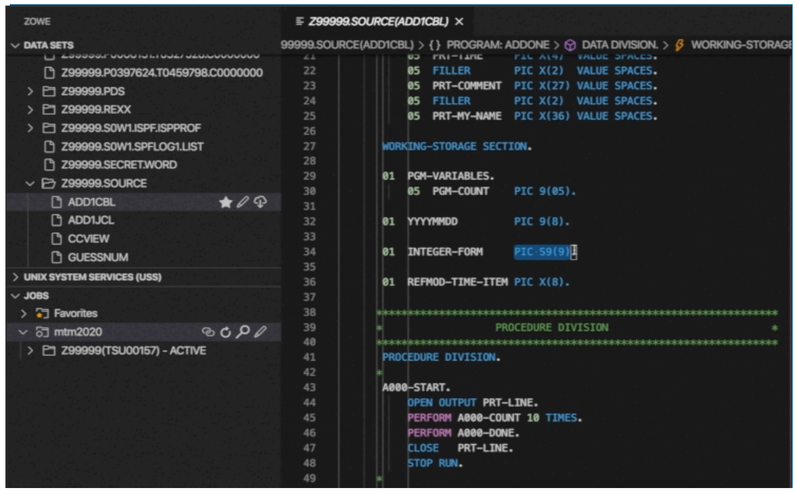


2. VERIFY IT WORKED

Restart VS Code after saving your settings.json file, just to make sure changes take effect. When you come back up, take a peek at **ADD1CBL** in the **MTM2020.PUBLIC.SOURCE** data set.

If it didn't automatically detect that it's a COBOL file, go to the bottom-right corner of your editor and change it from Plain Text to COBOL. If that works, and you see a nice bunch of code in various colors, then you've done it successfully.

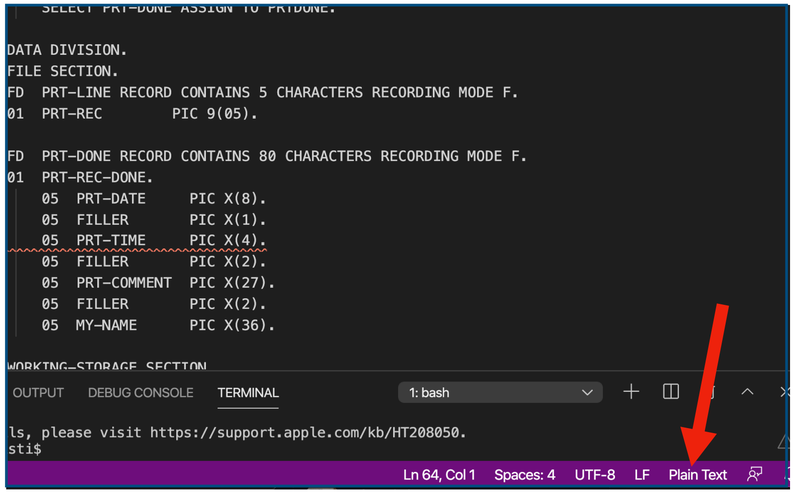
If not, check your Java installation and make sure there aren't any other extensions overpowering the IBM Z Open Editor.



3. GET FAMILIAR WITH IT ALL

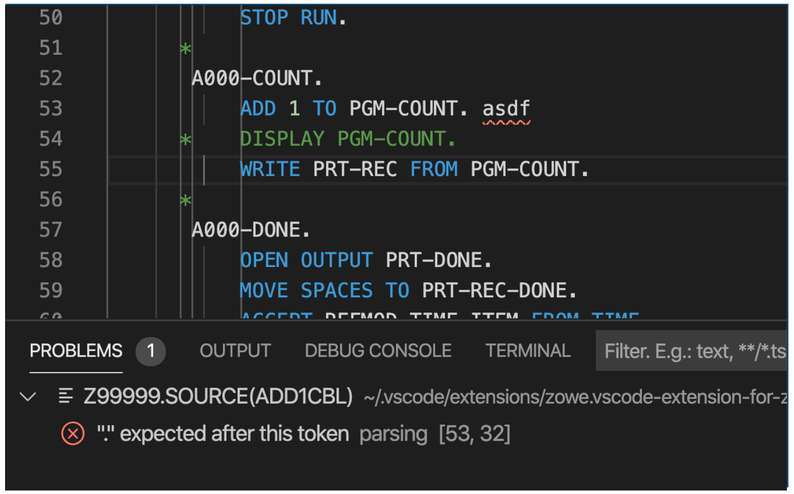
It takes more than a few paragraphs of text to get familiar with COBOL. As straightforward as the code is, there is a lot of tightly-packed information going on in here. For a brief readthrough of the code, watch this video.

COBOL in Ten Minutes



4. SET YOUR STYLE

Notice the file type section of your editor. Sometimes the Open Editor extension will assume a file is COBOL source code because it has the letters 'cbl' in it, or is in a COBOL data set. When that's the case, you can override it and say "No, this is just a regular plain text file" by changing it here.



5. MIND THE LINES

As you type, you'll get alerted to any errors in basic syntax down in the Problems section of your screen. Errors down here means your code probably won't compile, so pay attention to them.

You can also hover your mouse cursor over certain variables and paragraphs to get detailed information about how they were defined and referenced. Make use of this whenever possible, it will save you lots of scrolling back and forth.

6. COBOL, YES. BUT JCL, TOO.

As you saw in the video, there's a JCL component of compiling and running COBOL. You'll find the JCL to run this program in the same **MTM2020.PUBLIC.SOURCE** data set with the name **ADD1JCL**.

Before you try to make any changes and find that you don't have access to edit those MTM2020.PUBLIC files, copy ADD1CBL and ADD1JCL from the the MTM2020.PUBLIC.SOURCE data sets into your own SOURCE data set.

The JCL should require no changing, since it uses symbolics. After copying, fire off the JCL (Not the COBOL code directly. Easy mistake, we've all been there) and make sure it runs and produces the output you expect.

WHY ARE WE LEARNING COBOL TODAY? AS IN, THIS DECADE?

Programming languages are typically built around what your average programmer is doing at a specific time. In order for a language to remain relevant and useful, it needs to either be very flexible and extensible (like Python) or be so artfully created that its power scales with the hardware it runs on (like C).

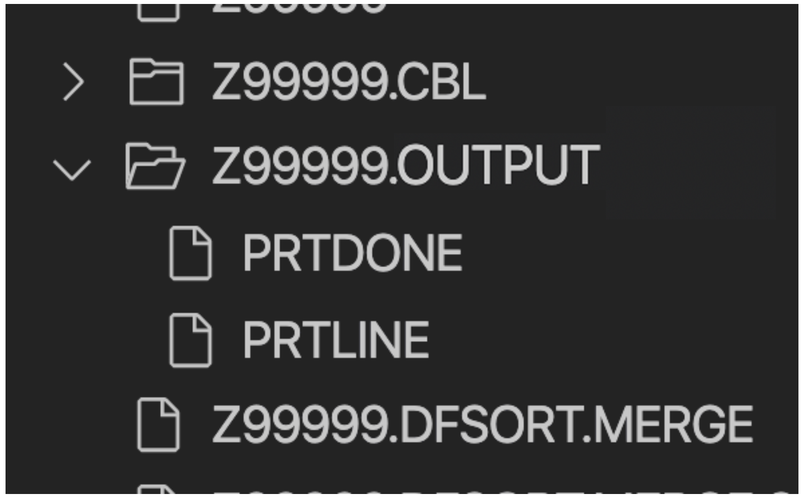
COBOL is not a general purpose programming language. You wouldn't write an operating system, AI algorithm, or racing sim in it. Where it excels is turning out high-precision business transactions like the ones you find in banks, credit card processors, and other financial institutions. Its rigid structure and adherence to a very specific specification mean that it's also easy to read and trace through, something that helps with audits and compliance. Today, COBOL runs much of the world's highest-earning businesses, and once you start to see how it was architected, you'll begin to see just why.

```
//*************************************************
// IF RC = 0 THEN
//*************************************************
//RUN      EXEC PGM=ADD1CBL
//STEPLIB    DD DSN=&SYSUID..LOAD,DISP=SHR
//PRTDONE    DD SYSOUT=*,OUTLIM=15000
//PRTLINE    DD SYSOUT=*,OUTLIM=15000
//SYSOUT     DD SYSOUT=*,OUTLIM=15000
//CEEDUMP    DD DUMMY
//SYSUDUMP   DD DUMMY
//*************************************************
```

7. SWITCHING OUT THE DD

Make changes so that instead of producing output in the job log, it produces members in your **ZXXXXX.OUTPUT** data set. Make sure you use THAT specific data set, as it's been created with the formatting required for COBOL output.

Use the same format you've used for DD statements in the past. Remember, because of the way COBOL interacts with the system it's running on, it doesn't directly handle the file details (at least in this case). You can make this change without changing anything in your COBOL source code.



8. CHECK YOUR FILES

You should wind up with two data set members in OUTPUT, which look similar to what you saw in the output. When in doubt, check the disposition in those DD statements, and make sure it's pointing where you expect.

Moral of the story, you can make changes in JCL that determine the behavior of a program, while the COBOL source code determines the program flow and logic. This is just how it works in a production mainframe shop.

```
FD  PRT-DONE RECORD CONTAINS 80 CHARACTERS REC
01  PRT-REC-DONE.
    05  PRT-DATE     PIC X(8)  VALUE SPACES.
    05  FILLER       PIC X(1)  VALUE SPACES.
    05  PRT-TIME     PIC X(4)  VALUE SPACES.
    05  FILLER       PIC X(2)  VALUE SPACES.
    05  PRT-COMMENT  PIC X(27) VALUE SPACES.
    05  FILLER       PIC X(2)  VALUE SPACES.
    05  PRT-MY-NAME  PIC X(36) VALUE SPACES.
```

9. ADD YOUR NAME

If you watched the COBOL video to the end, or really investigated that source code, you have noticed that we never used the PRT-MY-NAME variable. Alter the code so that your name (or any name, really) goes in there, and it shows up in the output.

It's a small modification, but will familiarize you with some of the structure and syntax of COBOL in an environment you're fairly familiar with by now.

Note: You can check your results by removing the output files and running things again, or by editing the JCL to print out to the JCL job log.

THE DETAILS ARE IN THE DD STATEMENTS

We touched on DD statements back in the JCL challenges, but it's possible to miss exactly what's going on here. DD statements are a load-bearing piece of architecture in the world of z/OS. Master them, and not only will you be able to fix problems quickly, you'll be able to get the exact output you want, every time.

If you need a refresher, The Knowledge Center has you covered. You can also look at JCL you submitted for any earlier challenges that created output. Truth be told, most JCL doesn't get written from scratch. It's either copy/pasted from a well-known script, generated by REXX, or just modified from an established template, so don't feel guilty about "borrowing" working examples from all over, that's exactly what the professionals do, so you're on the right track.

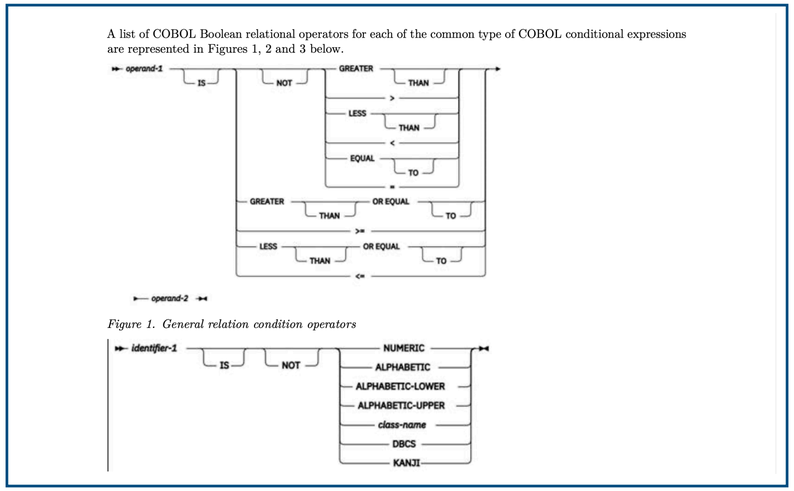
```
REPORT OF TOP ACCOUNT BALANCE HOLDERS
PREPARED FOR PAT STANARD ON 08.22.2020
# OF RECORDS: 19
====================================
Allison Oxshoot          8,652,205
Charles Fletch           8,662,995
Carmello Blonch          8,729,005
Allistair Sloder         8,730,553
Kirsten Niles            8,783,823
Robert Homeblip          8,800,800
Annette Van Hault        9,092,320
```

10. MAKING THE LIST

Pretend you work for a financial company, and your boss likes to know who their customers are, so let's generate a report showing a list of customers with exceptionally large account balances.

Use what you've learned to write a COBOL program that outputs the first and last names of everybody with an account balance over $8,500,000, including their account balance. The source data set can be found in **MTM2020.PUBLIC.INPUT(CUSTRECS)**. Name your program **TOPACCTS** and keep the source in your SOURCE data set.

This report must also include a count of how many customers have been identified. Refer to the screenshot above for an example (names and details have been altered)



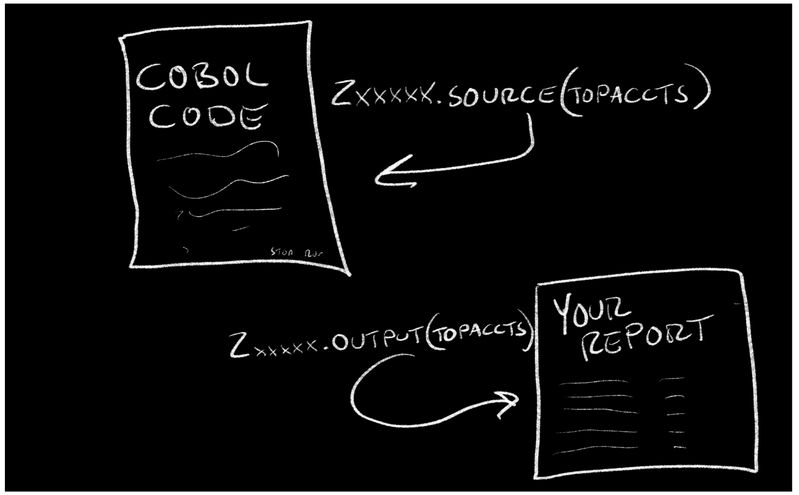
A list of COBOL Boolean relational operators for each of the common type of COBOL conditional expressions are represented in Figures 1, 2 and 3 below.

Figure 1. General relation condition operators

11. SOME GUIDANCE

Head over to the Open Mainframe Project's "COBOL Programming with VS Code" course, grab the latest version of the "Getting Started" pdf, and pay special attention to sections **10.3: COBOL programming techniques to read and write records sequentially**, and **13.1 Boolean logic, operators, operands, and identifiers**.

Big hint, look at the NUMVAL-C function.



12. REQUIRED FILES

You worked hard, make sure you have what we're looking for.

We need:
your COBOL program - **ZXXXXX.SOURCE(TOPACCTS)**
and its output – **ZXXXXX.OUTPUT(TOPACCTS)**

Make sure you include that count of the records in the report! When you're done, fire off **CHK3** (since we're in Level 3) and make sure your work gets counted!

NICE JOB! LET'S RECAP

When you see something written in COBOL, it's for good reason. There's an inherent precision you get from working in Binary Coded Decimal that you can't get from most other languages, specifically when it comes to fractional division.

You've demonstrated a solid understanding of some of the most important aspects of this language, and have some good examples of how you applied them to put to use with prospective employers, college entrance boards, or just other fellow fans of COBOL.

NEXT UP...

You've got COBOL. If you found that fun, rewarding, or it felt like something you'd like to dive into deeper, you know where to go for more education. Now's the time to finish up those Level 3 challenges, but make sure you officially put COBOL on your resume, you've earned that distinction.