Master Thesis

Understanding Legacy Software: The Current Relevance of COBOL

by

Ashish Upadhaya

2560152/AUA800

First supervisor: Dr. Sieuwert van Otterloo

Second reader: Dr. Joost Schalken-Pinkster

December 4, 2023

Submitted in partial fulfillment of the requirements for the VU degree of Master of Science in Information Sciences

Abstract

In the ever-evolving landscape of technology, the persistent shortage of COBOL experts has emerged as a critical challenge for industries relying on COBOL programming language. As organizations grapple with the need to maintain and modernize their COBOL-based systems, the scarcity of skilled professionals becomes a pivotal factor impacting efficiency and innovation. This research delves into the result of the ongoing shortage of COBOL experts, exploring its implications on businesses and potential strategies to address this concern.

To provide a comprehensive understanding, this study focuses on the relevance of COBOL in today's tech world, utilizing a Dutch Leading bank as a case study. Various research methods, including literature review, interviews, and testing of different programs in different compilers have been employed to examine its current uses and applications. This approach contributes to the existing discourse by offering a in depth understanding on COBOL, addressing the gaps in recent studies that have not adequately explored its significance.

A case study was undertaken to examine the utilization of COBOL in a leading bank in the Netherlands, revealing insights into the particular applications of COBOL within the banking system and the challenges faced by the bank in utilizing this programming language. The findings from the case study highlighted specific ways in which COBOL is employed within the bank's operations and the challenges bank is facing with COBOL.

Additionally, the job market research offered an overview of the current demand for COBOL expertise. The results of the job market research shed light on the industries and areas where COBOL skills are particularly sought after and that there is demand for COBOL expertise.

Simultaneously, an investigation into available educational resources for COBOL was conducted to evaluate the accessibility of learning materials. This examination aimed to determine whether there are sufficient resources to facilitate effective knowledge transfer and skill development in COBOL.

The findings contribute to a comprehensive understanding of the landscape surrounding COBOL, encompassing its practical applications, demand in the job market, and the educational tools available for those looking to acquire proficiency in the language. Additionally, the study lists educational resources available to learn COBOL, aiming to bridge the education gap and further contribute to the understanding of this essential programming language.

Keywords: COBOL, JCL, Mainframe, Continues Relevance, Financial sector, Data Processing Capabilities, Professional Skill Shortage, Infrastructure Modernization, Technological Challenges**.**

Table of contents

1. Introduction and Historical Overview of COBOL

1.1 Introduction

Programming languages serve as the foundational pillars for software development, with their evolution significantly changing our interaction paradigm with computers. Among the programming languages that have been developed over time, COBOL holds a unique position, having contributed to the evolution of programming languages. Originally designed to meet the needs of business applications, COBOL quickly gained popularity due to its user-friendly nature and efficiency (Ensmenger, 2011).

While COBOL's historical importance has been extensively examined, encompassing its pivotal role in resolving the Y2K issue, addressing migration challenges, and acknowledging its significance in the business sector, there is a noticeable gap in recent academic literature specifically focusing on the current relevance of COBOL. Nevertheless, it remains a highly popular language. COBOL's enduring popularity can be attributed to its consistent updates, tailored to meet evolving business needs. Scot Nielsen, Director of Product Management for COBOL at Micro Focus, underscores the resilience of systems developed decades ago, emphasizing their continued effective operation. This not only eliminates the necessity to discard and start a new but also encourages a seamless process of building upon the existing foundation (Wayner, 2022).

In today's landscape, COBOL continues to play a crucial role in industries that prioritize reliability, efficiency, scalability, and compatibility (Varie, 2023). Yet, companies are considering updating their systems and embracing alternative programming languages or platforms to keep up with the changing business needs and customer expectations. However, they encounter challenges migrating to newer platforms associated with COBOL due to skill shortage and high cost of migration (Varie, 2023; Sneed, 2009; Ciborowska et al., 2021). To tackle these challenges, certain organizations are exploring strategies to transition from COBOL to alternative languages like Java or C# (Dorninger et al., 2017; Suganuma et al., 2008; Sneed, 2013; Van Assen et al., 2023). This often involves leveraging automated conversion tools or adopting cloud-based solutions (Basetech, 2023; Astadia, 2023; The Cloud for Mainframe & COBOL: Migration & Modernization, 2023;). Alternatively, some organizations are focusing on strengthening the COBOL proficiency of their current or incoming workforce. This is achieved through initiatives such as training programs on work as 95 percent of all COBOL runs on mainframes, COBOL is best taught side-by-side with mainframe computing (No, COBOL Is Not a Dead Language, 2021).

In the academic context, particularly within Dutch universities, a notable gap exists in educational resources dedicated to COBOL, as no specific programs for this language are currently offered. This gap can be traced back to the emergence of new programming languages. As new programming languages and technologies emerged, there were predictions that COBOL would

become obsolete and decline in popularity. Despite the challenges of emerging programming languages, COBOL persists due to its remarkable stability, even though the contemporary preference for flexible applications has contributed to its declining popularity (Asay, 2018). However, the ongoing demand for COBOL expertise, especially in pivotal tasks such as maintaining or even migrating legacy systems, has reignited discussions about the importance of introduction of COBOL in education (Thibodeau, 2013; Reis, 2021).

This thesis is set out to fill the gaps in recent COBOL research addressed, with a specific focus on the relevance of COBOL. Its exploration will encompass the historical context of COBOL, its current applications, the challenges associated with transitioning away from it, and the available educational resources. The aim is to provide valuable insights to academics about the current relevance of COBOL and to find out if there are enough resources to learn Cobol as there is lack of institution that teach COBOL. It is also expected that this research will stimulate increased interest among academic in the programming domain of COBOL. Through the presentation of data and the emphasis on the continued relevance of COBOL in today's IT landscape, an effort is made to shine a light on the crucial role COBOL plays in the current IT landscape.

1.2 History of COBOL

Before delving into the current relevance of COBOL, it is important to have a thorough understanding of its historical development. This includes investigating its inception, identifying the individuals responsible for its development, assessing its initial areas of application, and examining other significant historical milestones that have collectively led to COBOL's current form and relevance.

COBOL, or Common Business Oriented Language, was created in the late 1950s as a collaborative effort among government, academic institutions, and the computing industry. Its development was primarily driven by the urgent requirement for a universal high-level language that could be used for diverse business applications on numerous computers (Kappelman, 2000). The lack of compatibility among different computer systems was a significant challenge for businesses that needed to move data and programs from one system to another. As a universal language, COBOL promised a solution to this crisis (Kappelman, 2000).

Grace Hopper, a distinguished computer scientist and naval officer, is often linked with the development of COBOL. Her work in developing the first compiler for an early programming language established the foundation for COBOL's capabilities as a cross-platform language. Hopper played a crucial role in defining COBOL's specifications and in its implementation. Hopper passionately advocated for the use of programming languages by a wider audience, encompassing business experts and government officials (Silverberg, 1996).

Before COBOL, businesses relied primarily on machine or assembly languages, which were difficult to write, understand, and maintain. COBOL's development was facilitated by the desire for a more understandable and maintainable language that closely resembled human English. Another key factor was the demand for standardization, ensuring that business applications could be portable by using the same programming language across various computer systems (Ensmenger, 2011). “A 1959 survey had found that in any data processing installation, the programming cost US$800,000 on average and that translating programs to run on new hardware would cost $600,000” (“Grace Hopper and the Invention of the Information Age,” 2010, p. 282).

FLOW-MATIC, an early English-like data processing language built for the UNIVAC I, was a forerunner to COBOL. Behind its development was Grace Hopper. When COBOL was on its way, the success of FLOW-MATIC demonstrated the potential of including English-like statements in programming languages. As a result, much of COBOL's inspiration and specific features came from FLOW-MATIC, with Grace Hopper playing a key role in COBOL's development due to her prior expertise (Sammet, 1978; Ensmenger, 2011). The Conference on Data Systems Languages, or CODASYL, was founded in 1959 to direct the development of COBOL and, later, to provide standards for other IT disciplines. CODASYL was born with the support of the Department of Defence, which was eager to have a standardized business language (Ensmenger, 2011). COBOL arose in response to the business world's desire for a standardized, high-level programming dialect. FLOW-MATIC's roots and CODASYL's stewardship were critical in influencing COBOL's trajectory (Ensmenger, 2011).

From the beginning, COBOL was designed to be accessible to business individuals with little to no programming experience. Its design prioritized business needs, with features such as decimal arithmetic support and the ability to handle large data files. The language quickly became popular and emerged as the top choice for business applications. During the 1960s and 1970s, COBOL was recognized as the most extensively used programming language globally (Glass, 1997).

COBOL's historical journey has been shaped by various influential developments, with one pivotal turning point being the ground-breaking "GOTO considered harmful" letter by Edgar W. Dijkstra, published in 1968 (Dijkstra, 1968). This letter had a profound impact on the software development community, including COBOL. Dijkstra's challenge to the unrestricted use of GOTO statements in programming languages inspired a change in thinking toward structured programming principles. While COBOL initially allowed GOTO statements for flow control, modern COBOL programming now emphasizes structured control flow constructs like PERFORM, IF-ELSE, and EVALUATE, promoting code readability and maintainability in line with Dijkstra's vision.

In COBOL, the use of ‘GO TO’ statements are typically avoided due to a variety of problems. These statements can create "spaghetti code," which is complex and difficult to follow, making it tough to track the flow of the program. This complexity also makes maintenance and debugging harder, as minor modifications can lead to unforeseen results. ‘Go TO’ statements also elevate the risk of errors by potentially causing execution jumps that may result in infinite loops or missed steps. Furthermore, these statements disrupt the clean structure of the code, resulting in a format that is hard to read and interpret. To emphasize this, refer to Appendix 2 which includes two sets of programs: 'GOTO1' and 'GOTO2', along with their structured solutions 'GOTO1 Solution' and 'GOTO2 solution'. Each set illustrates how code can be improved by avoiding ‘Go To’ statements, thereby enhancing readability and maintainability.

Although legacy COBOL code may still contain remnants of GOTO statements, COBOL codes prevented use by developing practices prioritizing organized and efficient code, aligning with broader software engineering trends (Marcotty & Ledgard, 1986). Nevertheless, COBOL's path to success faced challenges, including criticism for its verbosity and limited support for structured programming constructs. Yet, COBOL displayed adaptability, continuously growing, and improving with the release of newer versions (Murach, 2001). Another important development in the history of COBOL was the Y2k problem. COBOL emerged in an era where computer memory was both a coveted and costly commodity. A common programming practice, born out of necessity, was to represent years with only two digits in date fields, a decision that would later have far-reaching implications. This led to the infamous Year 2000 problem or Y2K problem, where COBOL systems, extensively used in businesses and governments worldwide, were particularly susceptible.

This memory-saving practice posed no issues until the arrival of the new millennium, which introduced a significant ambiguity: '00'. Without a clear century prefix, '00' could be misread as 1900 or 2000. This misinterpretation could lead to catastrophic results in applications where accurate data interpretation was crucial, such as banking systems calculating interest or operations where chronological order mattered.

To counter the impending Y2K crisis, organizations globally undertook significant efforts to rectify the issue in their software systems. This often meant manually reviewing and modifying countless lines of COBOL code, representing a significant investment of time, resources, and capital (Bennatan, 1997).

Despite worldwide efforts, concerns about system failures remained as the new millennium approached. Thankfully, due to the collective efforts of the software community to tackle the Y2K bug, the anticipated widespread disasters were avoided. This Y2K crisis and COBOL-related challenges highlight the importance of strategic planning, skill preservation, and continuing

maintenance for legacy systems such as COBOL. Businesses can use these insights to overcome the current COBOL programming shortage by investing in education, training, documentation, and a long-term view of COBOL's role in their technology ecosystem. Despite the Y2K and GOTO issue hurdles, COBOL has experienced substantial advancements throughout its lifespan. Figure 1 highlights the major developments in COBOL's history (Glen, 2022).

To conclude, COBOL was designed to standardize business applications across different computer systems. Despite its initial obstacles, it rose to dominate the realm of business applications, maintaining its prominence today. However, the growing age of COBOL systems and the scarcity of professional programmers are raising formidable challenges for organizations that depend on these systems. Regardless, COBOL's influential role in the computing world is certain and continues to rule the global economy (Jones, 1997).

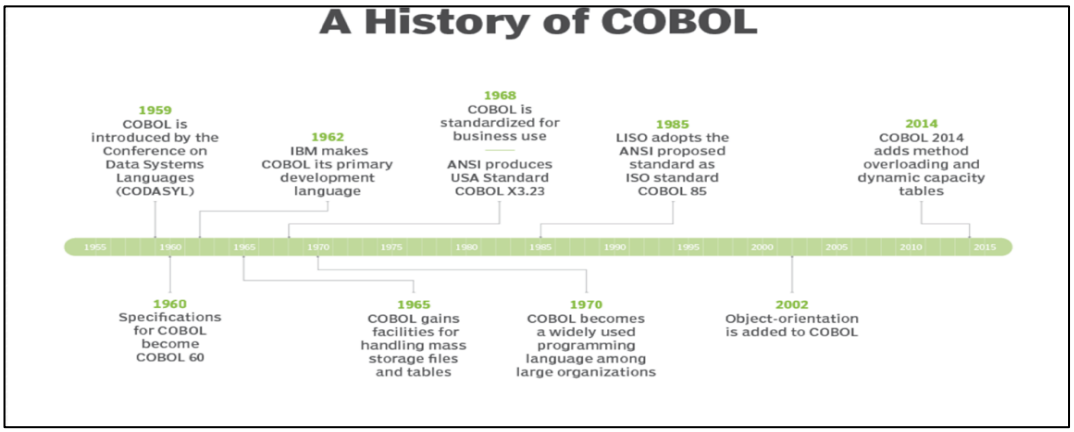


Figure 1. History of COBOL, Source: TechTarget(2022)

1.2.1 The Y2K Problem

As already shortly mentioned in the previous section, the Y2K problem, also known as the millennium bug, was a widespread issue linked to the way calendar data was formatted and stored in computer systems. Specifically, many older systems, such as those written in COBOL, used two-digit representations for years to conserve memory (for instance, '99 for 1999). This practice introduced ambiguity with the arrival of the year 2000, as '00 could be misconstrued as either 1900 or 2000.

Below is a simple COBOL program that illustrates the Y2K problem:

Original Code (Y2K Problem)

```
IDENTIFICATION DIVISION.
Author.  Ashish Upadhaya.
PROGRAM-ID. Y2K-Problem.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 YEAR PIC 9(2).

PROCEDURE DIVISION.
MAIN-PROCEDURE.
MOVE 99 TO YEAR.
DISPLAY YEAR.

STOP RUN.
```

Output

```
(base) elitePro:cobol_test ashishupadhaya$ cobc -x y2kproblem.cbl
(base) elitePro:cobol_test ashishupadhaya$ ./y2kproblem
99
```

The program prints all years in 2 digits, which poses a problem when distinguishing between the years 1900 and 2000. For example, the year 2000 would be displayed as "00," leading to potential dysfunctions in certain systems. The inability to differentiate between these two critical periods could have profound consequences for the program's accuracy and reliability.

To address the Y2K problem, systems must be upgraded to handle four-digit years. Below is a corrected program version that resolves the Y2K problem:

Improved Code (Y2K Solution)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. Y2K-Problem-Solved.
Author. Ashish.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 YEAR PIC 9(4).

PROCEDURE DIVISION.
MAIN-PROCEDURE.
    MOVE 2000 TO YEAR.
    DISPLAY YEAR.
    STOP RUN.
```

Output

```
(base) elitePro:cobol_test ashishupadhaya$ cobc -x y2ksolved.cbl
(base) elitePro:cobol_test ashishupadhaya$ ./y2ksolved
2000
```

The updated version can handle years from 0000 to 9999 without any confusion, providing a basic solution for dealing with the Y2K problem. However, it often required significant efforts as it necessitated software code and data storage changes. In Appendix 2 one more Y2K problem has been tested and a solution to that problem is given. Many programs were able to generate 4-digit dates, however, programmers failed to add an exception for years after 1999. This oversight led to system malfunctions, outputting dates like 1900, 1901, and 1902 instead of 2000, 2001, and 2002, respectively.

The immediate solution to the Y2K problem involved comprehensive inspection and modification of the affected code. Organisations initiated thorough systems audits, pinpointing where dates were processed or stored. The necessary modifications frequently included extending two-digit year fields to four digits and adjusting related computations (Kappelman, 2000).

Returning to the COBOL example, the Y2K problem in the original program originated from representing years with only two digits and assuming the century to be '19'. This could result in misunderstandings, such as interpreting the year '30' as '1930' instead of '2030'. (Kappelman, 2000). While the solution seems straightforward, the practical challenges lay in the volume of legacy code that needed inspection and modification. These changes also necessitated corresponding modifications in data storage and databases to handle four-digit years. Moreover, meticulous testing was required to ensure these changes did not spawn new bugs (Kappelman, 2000). Automated tools were also developed to facilitate this process, which could identify and modify date-related code such as CA-Fix/2000 (Ricciuti, 1998). “Highly touted and automated Y2K tools also typically fell somewhat short of expectations, and caused more manual assessment, renovation, and testing work than had been planned” (Young, 2000).

To sum up, the Y2K problem was resolved through a systematic process of identifying affected systems, remediating code to expand date representations, rigorous testing, global coordination, automated tools, and public awareness campaigns. The key lessons from this experience include the need for proactive risk assessment, comprehensive testing of critical systems, effective collaboration among stakeholders, regulatory frameworks, legacy system modernization, public awareness, and continuous monitoring. These lessons underscore the importance of proactive risk management, collaborative efforts, and sustained diligence to prevent and mitigate potential crises in the ever-evolving technology landscape, ensuring the reliability and security of critical systems in the future.

1.3 The Current Programming Landscape

Government organisations, particularly in the United States, use COBOL systems for administrative duties and record-keeping tasks. For example, social security, taxation, and unemployment benefits systems still heavily rely on COBOL (CNN, 2020). These sectors highlight COBOL's persistent significance in modern software ecosystems. Even though the rise of new programming languages offers increased flexibility, COBOL's stronghold remains evident, particularly in industries that demand data-intensive operations and high reliability. The associated costs, complexity, and risks of migrating from COBOL to newer systems further substantiate its ongoing relevance (Sneed, 2013).

COBOL is not the first pick for creating new applications, but it is vital for many existing systems in industries that manage a lot of data or transactions. The ongoing use of COBOL comes from a few key reasons. First off, there is a massive amount of COBOL code out there – there are hundreds of billions of lines - that run systems vital for businesses worldwide (Today's Business Systems Run on COBOL, 2021).

Secondly, COBOL is not inherently more reliable or better than other programming languages, its reliability is enhanced when deployed within the mainframe environment. Mainframes are a preferred choice for businesses where system reliability is critical. A combination of hardware redundancy, fault tolerance, high availability features, and robust operating systems makes it a preferred choice worldwide (Today's Business Systems Run on COBOL, 2021).

Thirdly, according to the Popularity of Programming Language (PYPL), COBOL holds the 28th position among programming languages globally as shown in Figure 2. It is created by analysing how often language tutorials are searched on Google. Figure 2 indicates that in the last year, the demand for COBOL programmers has not decreased which again indicates the importance and relevance of this language (PYPL Popularity of Programming Language Index, 2023).

Worldwide, Jul 2023 :

Rank	Change	Language	Share	1-year trend
1		Python	27.43 %	-0.2 %
2		Java	16.19 %	-1.0 %
3		JavaScript	9.4 %	-0.1 %
4		C#	6.77 %	-0.3 %
5		C/C++	6.44 %	+0.2 %
6		PHP	5.03 %	-0.4 %
7		R	4.45 %	+0.1 %
8		TypeScript	3.02 %	+0.3 %
9	↑	Swift	2.42 %	+0.4 %
10	↑↑↑	Rust	2.15 %	+0.6 %
11	↓↓	Objective-C	2.13 %	+0.0 %
12	↓	Go	2.01 %	+0.0 %
13	↓	Kotlin	1.79 %	+0.0 %
14		Matlab	1.59 %	+0.0 %
15		Ruby	1.1 %	-0.0 %
16	↑↑↑↑	Ada	1.06 %	+0.3 %
17	↑	Powershell	1.06 %	+0.2 %
18	↓↓	VBA	0.91 %	-0.1 %
19	↓↓	Dart	0.86 %	-0.0 %
20	↑↑	Lua	0.64 %	+0.0 %
21		Visual Basic	0.58 %	-0.0 %
22	↑↑	Abap	0.57 %	+0.1 %
23	↓↓↓↓	Scala	0.57 %	-0.2 %
24	↓	Julia	0.42 %	-0.1 %
25		Groovy	0.42 %	-0.0 %
26	↑	Haskell	0.3 %	+0.0 %
27	↓	Perl	0.29 %	-0.0 %
28		Cobol	0.24 %	-0.0 %
29		Delphi/Pascal	0.16 %	+0.2 %

© Pierre Carbonnelle, 2023

Figure 2. Programming language market share (PYPL, 2023)

1.3.1 Current Applications of COBOL

COBOL remains crucial in various modern information systems. According to Enlyft there are 40626 companies worldwide that still use COBOL, of which 7466 are based in the United States of America (Enlyft, 2023). Its enduring relevance can be attributed to its robustness, efficiency in processing data-intensive operations, and the risks of transitioning from established legacy systems to newer technologies. If we are after speed, COBOL is clearly the fastest at handling data (Cave et al., 2017). Figure 3 illustrates the largest systems running on COBOL, based on data from 2013. According to Micro Focus, the number of lines of COBOL code has grown to a range of 775-

Figure 3. COBOL current use in the industries, Source: MicroFocus, 2023

850 billion by 2023. Nevertheless, this figure provides a clear depiction of the ongoing prevalence of COBOL applications, suggesting that its replacement is not forthcoming soon.

These sectors highlight COBOL's persistent significance in modern software ecosystems. Even though the rise of new programming languages offers increased flexibility, COBOL's stronghold remains evident, particularly in industries that demand data-intensive operations and high reliability. The associated costs, complexity, and risks of migrating from COBOL to newer systems further substantiate its ongoing relevance (Dubov, 2023).

1.3.2 IBM release cycle

IBM constantly releases updates to improve security fix bugs in the system, or to introduce additional features. Below a Gantt diagram is shown in Figure 4 for the release cycle of IBM COBOL compilers. Appendix 1 has the same diagram with better quality. The figure indicates that most of the compilers are outdated and are not supported anymore by IBM for security/bugs etc. Currently only versions above 6.3 and are supported by IBM and support end date is not published yet. Those versions get support from IBM and are being updated frequently to solve bugs and improve security.

Introduction to newer versions of the COBOL compiler underscores the language's continuous development and adaptability to evolving programming requirements, featuring additional functionalities introduced over the decades. This indicates that COBOL has been continuously developed to include advanced features over the decades. For a thorough exploration of the complete list of options within IBM COBOL compilers, including those introduced with each compiler version, it is advised to visit IBM's official website. Their comprehensive documentation details the full list of options, providing descriptions and availability across various compiler versions (IBM Documentation, 2022).

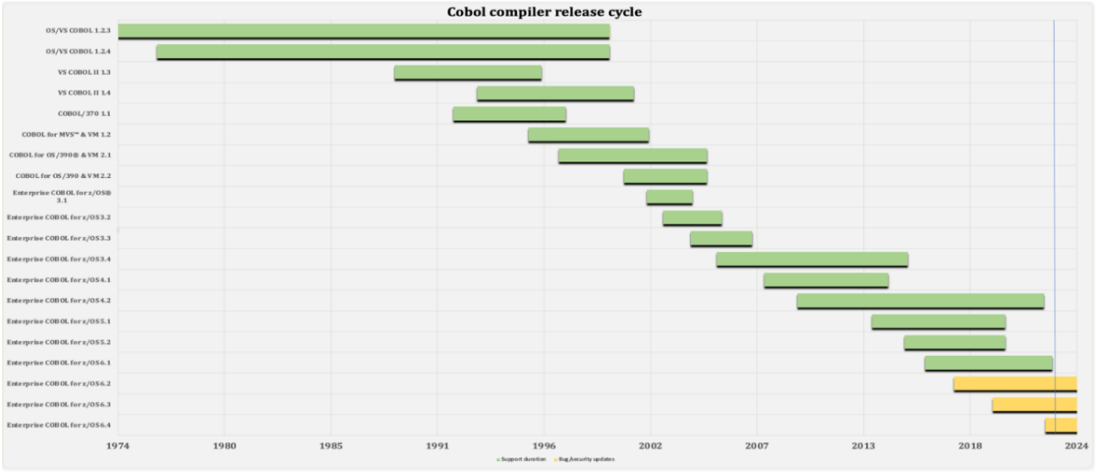


Figure 4. COBOL compiler release cycle, IBM

2. Research questions and methodology

As highlighted in the introduction, previous research has admirably outlined the historical context and foundational aspects of COBOL. However, there remains a notable gap in comprehensively addressing its current relevance in modern computing landscapes to contribute to the existing research gap, this thesis aims to delve into the ongoing significance of COBOL and its integration within contemporary technology and available educational resources. The following research questions were formulated to guide as a framework for this thesis.

2.1 Research Questions

RQ: To what extent does COBOL programming maintain its contemporary role and relevance in modern information systems, considering its current applications, migration challenges, and the availability of educational resources for learners?

SRQ1: What are the current uses and applications of COBOL?

SRQ2: What challenges do organizations face when migrating from COBOL to newer programming languages, and what strategies can be employed to mitigate those challenges?

SRQ3: What open-source tools and sample programs support COBOL learners and contribute to its current and future relevance?

2.2 Methodology

In chapter one, the research is initiated by delving into the historical evolution of COBOL through an examination of relevant literature. Key themes and trends were identified from academic papers, and the contemporary significance of COBOL was assessed by checking its popularity through different popularity indexes. The setting of this historical background was crucial as it laid the foundation for the subsequent discovery of COBOL's current uses and implications.

The focus then shifts to an analysis of popularity and the contemporary job market for COBOL programmers in section 3.1 covering SQ1. Information from different programming index websites help to understand COBOL's standing among other programming languages. Validation of these findings is then conducted using GitHub which involves searching for the total files associated with each programming language listed in the PYPL Programming Index. In GitHub, a search and filter operation was conducted for a specific programming language, excluding files that contained different language. This process resulted in the total count of files limited to the specified programming language. Furthermore, the fact that COBOL is used in different domain does not necessarily imply an ongoing need for COBOL experts; there might already be enough COBOL experts in the market. Therefore, job vacancies were examined. The obtained data provides insights into the current use and demand for COBOL expertise in the job market and shows the real time relevance of COBOL. This information was obtained by searching for job postings on LinkedIn and Indeed.com. The total number of job postings for various programming

languages, including Cobol, was searched by conducting a search for job listings associated with each programming language on LinkedIn e.g “COBOL”, “Python”. Additionally, an assessment of the ongoing relevance of Cobol in different countries was carried out by searching for job listings using the keyword "COBOL" and specifying the location for each specific country on LinkedIn.

A job list is also created in Appendix 8, showcasing which companies in the Netherlands continue to use COBOL and their corresponding salary information. This was obtained by searching for “COBOL” or “Mainframe” on Indeed.com and reading the requirement of the job. Indeed.com was chosen for this case because it featured jobs that genuinely required COBOL or mainframe expertise. A search on LinkedIn for COBOL or mainframe revealed numerous job listings, but a significant portion of them lacked true relevance to COBOL or Mainframe.

Following the popularity and the job market demand for COBOL, this research takes a hands-on with a case study of a renowned bank in the Netherlands in section 3.2 to examine current use and application more closely. The challenges this bank faces with COBOL will also be discussed, addressing SRQ2. This illustrates how COBOL is utilized within the banking sector In this stage of the research, interviews with experts were conducted and several questions were asked regarding the use of COBOL in a bank. The questions and answers can be found in Appendix 3. Their experiences over 10 years and insights provide a unique understanding of COBOL's applications in a real-world context. The information gathered from the interviews will be thoroughly reviewed, and a suggestion from the findings will be made in section 3.2.1. The selection of interview participants was purposeful. The interviewees were all COBOL developers at the bank, who have been actively involved in the maintenance and evolution of COBOL systems. The qualitative data collected through interviews was analysed using thematic analysis. This approach made it possible to identify recurring themes, challenges, and strategies that the bank uses in managing COBOL systems. The thematic analysis aimed to provide a rich and in-depth exploration of the bank's journey with COBOL.

Section 3.3 also covers SRQ2 and discusses the complexities associated with managing legacy COBOL code, the ongoing efforts to modernize COBOL applications and the role between COBOL and other programming languages in Modern Business Applications. It explores the interplay between COBOL and other programming languages within modern business applications, notably Java. The chapter features a literature review, including analysis of some recently available articles to reveal the intricate relationship between COBOL and various programming languages in this context. The analysis is conducted using google scholar for “migration from COBOL to x”, where x is representing a programming language. selected articles were screened based on publication dates post-2018. After the selection process, a thorough reading, and analysis were conducted to identify the relevance to this thesis. Additionally, a program will be executed in different programming language to determine if the length of the legacy Cobol code is a factor for

migrating to different programming language. A GitHub repository was used to identify this. The repository that was used for this investigation was: https://github.com/q60/rot13/tree/main.

When addressing SRQ3, the research moves to a more technical aspect; a comparison of different COBOL compilers is conducted. In this phase, a variety of COBOL programs covering the fundamental aspects of programming are collected and compiled using Gnu compilers and z/OS COBOL 6.3. The Gnu compiler serves as an alternative to mainframes due to the high cost associated with purchasing mainframe computers for public use. By executing the samples programs a conclusion can be drawn if Gnu compiler is a good compiler to run COBOL programs. Furthermore, tutorials from YouTube and Udemy are analysed on the completeness of basic fundamentals of programming languages concepts, and sample COBOL codes are gathered using a search across multiple platforms including Google and GitHub. YouTube tutorials are accessed over the last 5 years, using the search term "Cobol tutorial," while keyword "Cobol samples" is used to searched for COBOL samples on Google and GitHub. For Udemy, a search will be conducted using the keyword "COBOL" filtering the results on most relevance and selecting the top 5 as educational resources. By looking at this, it can be researched whether there are enough resources available for individuals interested in starting their programming journey in COBOL.

In conclusion, throughout the research a combination of approaches is used in order to come to the right conclusion.

3. Results and analysis

3.1 Current use and application of COBOL

COBOL remains relevant, particularly in legacy systems, and remains essential in the financial and government sectors. It is key to functions like core banking and transactions, with job opportunities in COBOL still present. This enduring importance is evident among government agencies and financial institutions, which rely on COBOL for system stability and security.

3.1.1 Popularity of Programming Languages

As per 2023, according to the PYPL index, COBOL holds the 28th position among programming languages globally as shown in Figure 2. During the research, a trend was observed by looking at job listings such as LinkedIn and Indeed: a substantial number of companies within the financial sector expressed interest in professionals with COBOL Expertise. Prominent organizations such as ABN-AMRO, Belastingdienst, Capgemini, and Sociale Verzekeringsbank were among these companies. Appendix 8 provides a list with companies that are searching for COBOL developers as per October 2023 from Indeed.com. The monthly salary for the functions is also displayed in this list.

COBOL's enduring popularity is a testament to its stability, readability, and the fact that billions of lines of COBOL code still power crucial infrastructure worldwide. In an era dominated by languages such as Python, Java, and JavaScript, COBOL's presence in the top 30 affirms its place in the realm of programming history and its continued importance in today's digital landscape. Figure 2 indicates that in the last year the COBOL popularity was in the top 30, which again indicates the importance and the relevance of this language (PYPL Popularity of Programming Language Index, 2023). TIOBE, another website that tracks the popularity of programming languages indexed COBOL on the 21st place, whereas in Stack Overflow's Developer Survey, it ranked 39th (Stack Overflow Developer Survey 2022, 2022; TIOBE Index - TIOBE, 2022). TIOBE's survey tracks popularity by analyzing online search queries for programming languages. Stack Overflow's Developer Survey collects data from thousands of developers worldwide through an annual survey, providing insights into language preferences and usage in the developer community.

To gain a clearer perspective of COBOL's popularity, a search was conducted to determine the number of files currently available on GitHub for each programming language from the PYPL list. These searches revealed that, among the languages in the list, COBOL had the smallest number of available files, as shown in Figure 5. Nevertheless, it is noteworthy that approximately 27,100 COBOL files are still accessible o GitHub.

Although COBOL may appear to rank lower in comparison to languages like Java in various surveys, its consistent presence in these rankings underscores its enduring relevance.

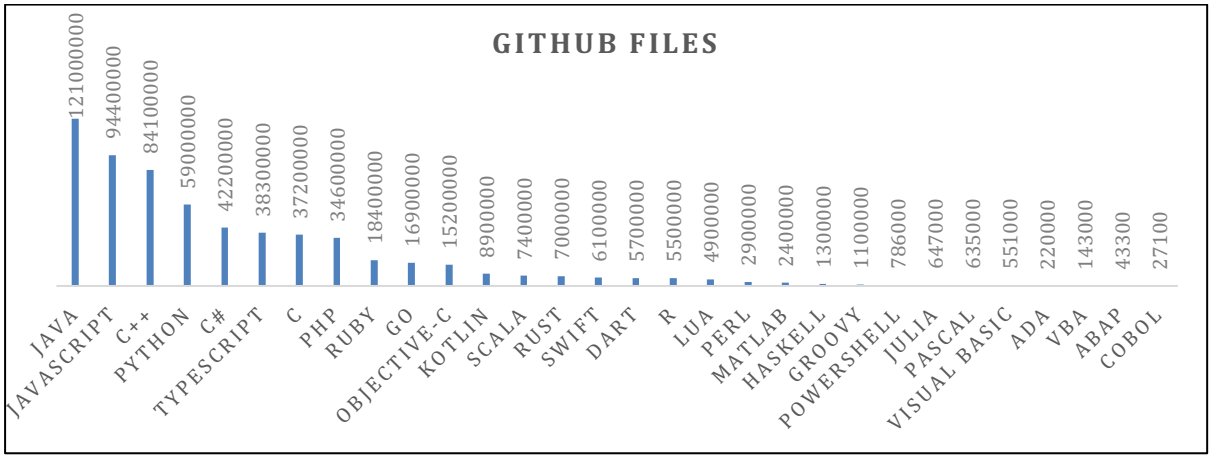


Figure 5. GitHub files per programming language, 2023

3.1.2 Current COBOL job market

One interesting way to understand COBOL's importance in today's tech scene is to look at job postings. According to LinkedIn data from June 2023 in the Netherlands, 2,019 job listings were asking for COBOL skills, whereas there were only 384 job listings for Fortran, which originated from the same era. This information was obtained by searching for job postings on LinkedIn, using the names of the programming languages. Comparatively, other programming languages showed higher numbers of job listings as shown in Figure 6.

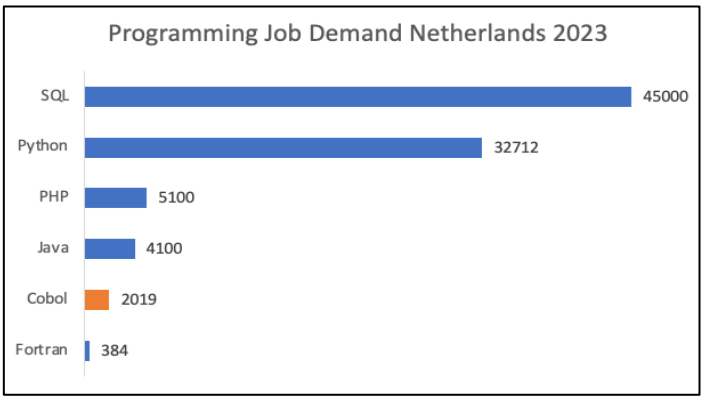


Figure 6. Programming Job Market Netherlands, 2023-06-25

While the number of COBOL job listings is lower compared to other languages, the fact that there were still over 2,000 job listings seeking COBOL skills indicates a considerable demand for COBOL professionals in the job market. However, it is worth noting that this number may not be precise because it contains all job listings containing the keyword "COBOL" in their vacancies. When the search was narrowed specifically to 'COBOL developer,' 78 results were found on LinkedIn, 46 on Indeed.nl, and 37 on randstand.nl, while a search for 'mainframe developer' on LinkedIn yielded 429 results. This suggests that there are still numerous open positions for COBOL-related roles. Furthermore, as more individuals retire, the demand for new COBOL professionals is likely to increase accordingly.

COBOL, despite originating in the 1950s, results from the job demand, several conclusions can be drawn. In the financial sector, institutions like ABN-AMRO Bank N.V. and the tax authority still use COBOL according to the job postings on LinkedIn and Indeed. The complicated network of banking operations, where both speed and reliability are important, COBOL in combination with mainframe infrastructure is a reliable solution to use for complex banking operation. On the other end of the spectrum, a multinational steel company also rely on COBOL for critical operations according to a job posting on Indeed.com. The complete list of jobs and companies using COBOL retrieved from indeed.com can be found in Appendix 8.

Given the real-time demands of retail, it comes as no surprise that a tried-and-tested system, even if older, continues to be a preferred choice, ensuring that day-to-day operations proceed seamlessly. According to job listings on LinkedIn and Indeed.com, IT consulting giants, including Capgemini, Ordina, and Huxley, recognize the importance of versatility. While they stand at the forefront of modern technological solutions, their expertise in COBOL ensures they can cater to a range of clients, some of whom operate on legacy systems. Public institutions like Sociale Verzekeringsbank and Belastingdienst, with their wide user base and essential services, often lean on long-standing infrastructures (Back to basic met COBOL - werken.belastingdienst.nl - Werken bij de Belastingdienst, 2023).

Within these large corporations' significant amounts of data must be processed, batch processing is often necessary for this. This process involves the use of Job Control Language (JCL) to delineate and manage batch jobs, including the execution of COBOL programs. Batch processing is the automated execution of tasks without user intervention. Job Control Language (JCL) is a scripting language used to define and manage batch jobs in mainframe systems like IBM z/OS. JCL ensures successful execution by managing job dependencies and resource allocation. It is essential for efficient and reliable batch processing. In large corporations, batch processing holds a vital position in efficiently managing and processing enormous amounts of data. Batch processing enables the automation and streamlining of several business processes, such as data processing, reporting, and system maintenance, on a scheduled or recurring basis. It ensures timely completion of tasks, optimizes system performance, maintains data integrity, supports complex workflows, enables scalability, improves efficiency, and reduces costs. This is where COBOL's history for unwavering reliability becomes indispensable. The continued reliance on COBOL across varied sectors underlines its unmatched robustness and the nuanced balance companies strike between historical reliability and modern innovation.

LinkedIn's job vacancy data also provides the global demand for COBOL expertise, revealing the extent of its entrenchment in different regions around the world. During a search for COBOL demand on LinkedIn using COBOL as a keyword to find jobs in a specific region the following data was obtained shown in figure 7. However, this data may not be accurate and needs more refinement as it was only based on the keyword "COBOL".

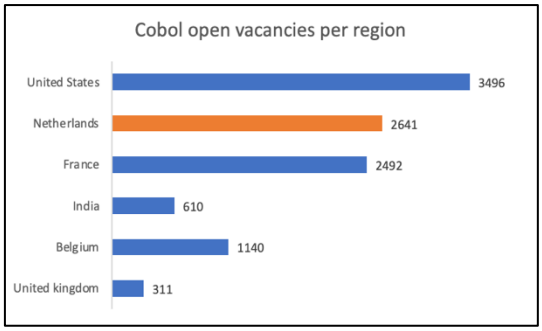


Figure 7. Open vacancies in specific region for COBOL, 2023

In summary, despite being an older programming language, COBOL maintains its global relevance, especially in the field of legacy systems. It is widely used in various sectors, including finance and government, where it plays a vital role in critical functions such as core banking and transactions. COBOL's strength lies in its ability to process massive datasets while ensuring data integrity. Career opportunities that require expertise in mainframe and COBOL remain, underscoring its ongoing importance. In the Netherlands institutions such as Social Verzekering, the tax authority, and the banking sector rely on COBOL to process extensive data, particularly in high-transaction environments. Research consistently shows COBOL's presence in relevant indexes, reaffirming its enduring relevance in the ever-evolving technology landscape. The GitHub search showed that COBOL is still in active use.

3.2 Case Study findings: COBOL Usage in a Leading Dutch Bank

In today's digital era, it is easy to assume that modern banks would have uniformly adopted the latest technological tools and languages. However, the truth is more complicated. Despite the rise of innovative technologies, many global financial institutions continue to rely on legacy systems, with COBOL being a prime example. One leading Dutch bank's enduring reliance on COBOL demonstrates this relationship between the old and the new. Serving as the backbone of their IT operations, COBOL oversees core processes like payments, transactions, and Financial Basic Systems (FBS). These processes necessitate handling vast data volumes efficiently, an arena where COBOL excels.

The primary goal of this case study is to evaluate a leading Dutch bank's long-standing relationship with COBOL, a legacy programming language, in the context of rapid technological changes. The main question guiding this case study is: " What are the current uses and applications of COBOL programming?".

COBOL's ability to adapt was tested during the Y2K bug crisis in the bank, a situation that could have disrupted key banking operations. As seen in the previous sections, the bank resolved this issue by adjusting all the COBOL systems to handle a four-digit year code manually. Furthermore,

the bank continued to upgrade to newer versions of COBOL, currently using version 6.3, which offers improved data processing and resource management. 6.4 is the most recent version supported by IBM however due to the reliability and security reason of 6.4 they are currently using 6.3 (Appendix 4).

Even though COBOL, as a legacy system, presents maintenance challenges, the bank believes its benefits - cost-effectiveness, security, and control - surpass these hurdles. Banks have reinforced these benefits with secure coding practices such as data encryption, multi-factor authentication, and proper input validation. Despite technological advances such as AI and cloud computing, COBOL remains indispensable due to its unique capability to efficiently process significant amounts of data. Several factors influence COBOL within the bank. These include the challenge of replacing COBOL with modern languages, the shortage of COBOL-experienced developers, and the pressure of adapting to evolving infrastructures.

However, the bank plans to keep COBOL because of its reliability, the excessive costs and risks associated with transitioning to another language, and COBOL's proficiency in processing vast data volumes. To address the problem of the shortage of experienced COBOL developers, de bank is currently hiring and giving training in India to keep the mainframe environment running. The bank has also developed an application that translates query-type language into COBOL, addressing the challenge of a declining availability of COBOL experts. With this application, individuals without programming knowledge can efficiently work and generate COBOL codes based on their specific requirements.

In the FBS department, the mainframe team consists of around 20 specialists deeply engaged with the mainframe system. Broadening the perspective, 150-200 individuals are actively involved with mainframe operations at the bank. They maintain over 27,000 active COBOL programs, a number that historically peaked at 57,000. As the bank gradually phases out these older programs, there is a marked shift towards modern languages like Java. This transition is further evidenced by the bank's integration tools such as OneSumX, Beam, and Topaz. Leveraging platforms like Informatica ETL, Hadoop, Databricks, and the data analytics capabilities of SAS Enterprise Guide, the bank has set its sights on a future migration to Azure.

SAS Enterprise Guide is a popular tool known for its point-and-click interface that enables users to access, manage, and analyse data from various sources. Within the bank, SAS Enterprise Guide provides the ability to query DB2 tables directly, making data analysis and reporting tasks more streamlined and efficient. DB2 is a database management system that is mostly used to run mission-critical workloads in mainframe.

While COBOL continues to serve as a foundational pillar, modern development practices are seamlessly integrated. Jenkins, a widely used automation server, streamlines tasks like building, testing, and deploying applications, ensuring smoother and more efficient workflows. SonarQube, on the other hand, is a tool dedicated to continuously inspecting the quality of source code and detecting bugs, vulnerabilities, and code smells. This ensures that the code adheres to set quality standards and best practices. The bank also uses Topaz, a suite of developer tools that aids in mainframe application development by providing a clearer view into data structures and relationships, making it easier to understand, debug, and maintain the code. When a component needs to be promoted it should happen from topaz. Once someone tries to promote their component from a test environment to a higher environment a continues integration pipeline gets triggered and SonarQube will start testing the code on several aspects as it can be seen in Appendix 5.

The bank's technological evolution extends beyond merely relying on IBM's COBOL compiler.; it is about evolving and expanding integration capabilities with systems like DB2. Specialized tools, such as Cobra which were particularly developed for this bank for database management, and the Topaz, are tailored to address specific operational demands.

But challenges persist. The retirement wave of mainframe professionals and the influx of a newer, younger generation brings forth the potential for knowledge gaps. It is a challenging task to transfer 30-35 years of experience into just 2 or 3 years of training. In the past, due to frequent management changes, some IBM colleagues were reluctant to share their expertise, fearing it might cost their job security as new management could replace them easily. However, these walls are gradually breaking down.

A new generation of professionals brings a fresh collaborative spirit, enhancing knowledge sharing in recent years. At this moment there is a balance between new and more experienced people. In a stable situation, there may not be an inherent knowledge gap; however, during periods of uncertainty, departments could face challenges as colleagues may be reluctant to share their knowledge to secure their positions.

In terms of security, the bank's mainframe, while not impervious, serves as a robust defence mechanism. This security includes various aspects, including confidentiality, integrity, and non-repudiation. Comprehensive logging ensures that every action within the system is traceable, enhancing both integrity and non-repudiation. Stringent access controls further bolster security, safeguarding against both accidental and malicious threat vectors. While vulnerabilities may exist, it is worth noting that the mainframe, powered by COBOL, has a strong security infrastructure that often surpasses that of many contemporary systems, providing a multi-faceted defence against potential threats.

In conclusion, the bank maintains a balance between depending on an enduring technology like COBOL and adapting to the fast-paced digital world. COBOL plays a vital role in the bank's business objectives, particularly in fast transaction processing - a key factor in customer satisfaction. This case study not only offers insights into the current state of FBS domain but also provides valuable lessons applicable beyond this context. It highlights COBOL's enduring importance in banking, strategies for addressing maintenance challenges, and the balance between legacy systems and modernization. While these insights may resonate with other domains within the bank and financial institutions like Rabobank or SVB, their applicability will depend on each organization's unique circumstances. The recorded responses to the interview questions used for this case study can be found in Appendix 3.

3.2.1 Suggestions from case study

This case study shows how COBOL, an older programming language, is still vital to a major Dutch bank's operations. Despite its age, COBOL's reliability and ability to handle large data sets make it irreplaceable in certain areas. However, reliance on COBOL also brings challenges such as finding skilled programmers and integrating modern technologies.

For these reasons, here are some suggestions:

- Skill Training: The bank should invest in training new IT staff in COBOL to ensure an expert team is always available for system maintenance and updates.
- Explore Modern Technologies: While COBOL is essential for some tasks, the bank can consider using modern programming languages and cloud computing for tasks that are less data intensive.
- Support COBOL Advancements: The bank could help develop new COBOL tools or contribute to open-source projects to ensure the language continues to evolve and remain relevant.
- Plan for the Future: The bank should have a contingency plan to deal with potential issues, like a lack of COBOL-skilled programmers or the language becoming obsolete.

3.3 Modernization of COBOL

The modernization of COBOL is a critical endeavour as organizations seek to update their legacy systems. Efforts involve integrating COBOL with newer technologies, enabling better scalability and performance. Additionally, advancements in COBOL dialects, such as Micro Focus COBOL, are accommodating modern programming paradigms, making it easier to adapt and integrate COBOL applications with contemporary platforms. These modernization initiatives ensure that COBOL remains an asset in today's ever-evolving digital landscape.

3.3.1 Dealing with COBOL Programs: Language migration vs Platform migration

Many programming languages have tried to replicate COBOL, but none have truly succeeded. Java, however, has emerged as a potential contender (Sneed et al., 2013). A simple Google Scholar search reveals that migrating from COBOL to Java holds the highest appeal when compared to other programming languages. An analysis conducted in August 2023 using google scholar for "migration from COBOL to x", where x is representing a programming language, revealed approximately 4,410 papers related to COBOL-to-Java migration, 1,030 related to Python, 1,240 for COBOL-to-C#, and 230 for COBOL-to-Scala migration. This search was done without implementing any filter. In a similar search, but excluding papers before 2018, the number of relevant papers significantly decreased, with 671 for Java migration, 337 for Python, 267 for C#, and 88 for Scala. A visual representation of these findings is presented in Figure 8, which shows the prevalence of different programming languages in COBOL migration papers. This search was essential to understand the current roles of other languages in COBOL's modernization or migration efforts. Subsequently, few papers from the first 2 pages in google scholar were selected for few programming languages, leading to the following conclusions regarding the focus of the most recent papers. The title of the analyzed papers and their relevance to this thesis can be found in Appendix 6.

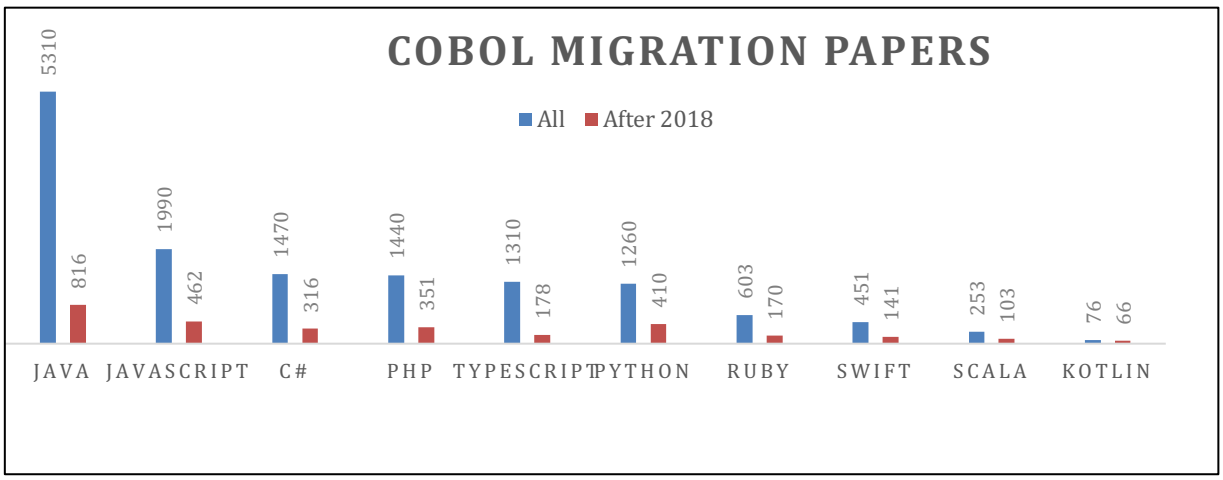


Figure 8. Scientific Papers on COBOL Migration, 2023

Migrating from COBOL to Java: Papers in this category explore practical aspects of migrating from COBOL to Java, including hybrid approaches. They also investigate microservices for modernizing legacy software, efficient platform migration, and open-source methods. Additionally, the papers cover automated code transformations, screen scraping for mainframe migration, and the assessment of antipatterns and complexity. These papers also address topology considerations for legacy system migration and experiences with cost-driven software migration, including re-implementing legacy systems.

Migrating from COBOL to Python: Most of the papers focus on unsupervised code translation and the use of automated unit tests. They delve into machine learning-based program translation and learning representations of COBOL code. The papers also discuss cloud migration, language porting, human-AI collaborations in code translation, security considerations and multilingual code snippet training relevant for modernizing COBOL. Python is known for its ease of scripting and automation. Various migration tasks, such as data extraction, transformation, and loading (ETL), can be automated with Python scripts, making the migration process more efficient. However, none of the papers provided a concrete real-world scenario showcasing the use of Python for migrating from COBOL. With the advent of OpenAI Codex and advancements in AI technology, there is optimism that soon it will become capable of seamlessly executing the migration process from COBOL to Python. An article on towardsdatascience.com reported successful tests in converting a few programs to and from COBOL, although it acknowledged the current limitations due to OpenAI Codex not being fully trained (Ryan, 2022). Despite the imperfect functionality at present, there is anticipation that ongoing development efforts will lead to breakthroughs, enabling Codex to translate between COBOL and Python, and potentially any programming language in the future.

Migrating from COBOL to C#: These papers emphasize CFlat, an intermediate representation language, and its role in software migration to Java and C#. The papers also assess antipatterns and complexities in COBOL-to-C# migration. They highlight the importance of qualification in large-scale modernization, software refactoring in system modernization, access to COBOL repositories, enhancements in programming language transformation, developer perspectives on defects, cost-driven software migration, and post-migration adjustments.

Migrating from COBOL to Scala: The focus is on automatic code conversion to Scala and the learning of migration models for incremental language transitions. The papers discuss automated unit tests for code translation, insights from 30 years of software refactoring research, and challenges faced by software language engineers. Additionally, the selection introduces a comprehensive benchmark for code translation, explores support for lambda expressions in Java as an intermediary step before Scala, and discusses building applications with modern technologies, including Spring, Java, and PostgreSQL. Upon reviewing the results, it becomes

evident that most of the literature suggests that Java can serve as a replacement for COBOL in many scenarios. Other programming languages only help in the modernisation process of COBOL. COBOL and Java, with their different origins, designs, and goals, represent two ends of the programming spectrum. COBOL was primarily developed for business applications, while Java was designed as a general-purpose language for various programming tasks. Nevertheless, despite their differences, these two languages have a complex relationship that spans over decades.

The use cases of these languages play a significant role in the relationship between COBOL and Java. Typically, COBOL supports legacy systems, while Java is often chosen for modern, web-focused applications. Therefore, when businesses and organizations look to update their legacy systems, Java frequently comes up as a suitable replacement for COBOL. However, this switch can be difficult given the necessary migration of massive amounts of data and applications collected over many years or even decades. One example of a successful COBOL to Java migration is the case of The New York Times Company. In 2018, the company migrated its business-critical COBOL application for home delivery of newspapers to Java. The project was completed on time and within budget, and the new application has been in production ever since (De Marco et al., 2018).

The COBOL to Java migration at The New York Times Company is a success story that shows that it is possible to modernize legacy systems without disrupting critical business operations. However, it is important to note that not all COBOL to Java migrations is as successful. The success of a migration project depends on several factors, including the complexity of the legacy system, the availability of resources, and the level of technical expertise (De Marco et al., 2018).

Many organizations have developed tools enabling COBOL programs to compile and operate on Java virtual machines to facilitate this transition. This strategy, termed "Java interoperability" empowers organizations to leverage the benefits of Java without necessitating a complete overhaul of their legacy systems (Java and COBOL Interoperability Introduction, IBM.). This method has gained popularity among organizations aiming to update their systems while mitigating the risks and costs of a comprehensive migration.

To bridge the gap between these disparate worlds, various companies have developed tools and frameworks to enable smooth collaboration between COBOL and Java. For instance, IBM has launched a tool named "z/Transaction Processing Facility" enabling the integration of COBOL applications with Java-based web applications. As the number of COBOL developers are declining, IBM also expanded the abilities of its generative AI based Watson Code Assistant to include COBOL code translation into Java. The product is targeted at modernizing mainframe applications that run on IBM Z systems (Ghoshal, 2023).

This strategy permits organizations to capitalize on the strengths of both languages while mitigating the risks involved in full migration (IBM, 2023). OpenFrame from TmaxSoft is another solution for transitioning old mainframe applications to modern platforms like Linux, Unix, Docker Containers, or the public cloud. Additionally, TmaxSoft's JEUS application server allows COBOL and Java applications to operate together seamlessly. This means organizations can upgrade their existing COBOL systems with Java capabilities, simplifying the entire process of modernizing and integrating mainframe applications (Mainframe Modernization | OpenFrame | TMAXSoft, 2022 ; Tmaxsoft, 2021).

The experience report on "Bottom-up and Top-down COBOL System Migration to Web Services" highlights that adopting a service-oriented architecture (SOA) is a promising approach when migrating legacy systems to modern programming languages (Rodríguez et al., 2011). By converting COBOL programs to Web Services, organizations can leverage the benefits of SOA, enabling seamless communication and integration across heterogeneous systems.

The bottom-up migration approach involves encapsulating discrete functionalities within the COBOL code as independent Web Services, while the top-down migration approach focuses on designing the Web Services architecture first and aligning the COBOL legacy system accordingly. Both methods emphasize the utility of service-oriented principles, facilitating the creation of a coherent and extensible Web Services ecosystem. By implementing SOA, businesses can achieve smoother transitions to other programming languages, ensuring improved interoperability, scalability, and maintainability in the modern software landscape (Rodríguez et al., 2011).

To sum up, while many programming languages have attempted to replace COBOL, Java has emerged as a strong contender, supported by a lot of research that favours the switch from COBOL to Java. COBOL and Java are like opposites in the programming world: COBOL is designed for business, while Java is a versatile language for all kinds of tasks. They both have their places; COBOL is great for legacy systems, while Java shines in modern apps. When businesses want to update their old systems, Java often seems a desirable choice to replace COBOL. But switching is not always easy because of all the data and apps that have been collected over the years.
The New York Times Company did this successfully and showed that modernization was possible without causing much chaos. Companies are using strategies like Java interoperability and tools to make this transition smoother. They also use SOA to migrate old systems to modern languages like Java, making things work better in today's software world. This collaboration between COBOL and Java shows how adaptable programming languages are and provides flexible ways to upgrade and succeed in today's digital age.

3.3.2 The Challenge of Maintaining Legacy COBOL Code and Transitioning Away from COBOL

Migrating to more modern languages poses various challenges for organizations. These challenges arise due to COBOL's inherent characteristics and critical role in legacy systems.

One of the most significant challenges during the migration process is the potential loss of complex business logic embedded within legacy COBOL applications. COBOL systems, refined over decades, embody a wealth of domain-specific knowledge integral to an organization's operations. Many of these systems lack proper documentation, making it difficult to accurately capture and translate intricate business rules or functionality (Bailey, 2023). Translating this business logic into a new language becomes a challenging and error-prone task (Sneed, 2001). As COBOL applications have gained strategic significance, a comprehensive research study, encompassing respondents from 49 countries and focusing on companies with a workforce of at least 1000 employees, revealed a noteworthy trend. Among these respondents, a significant majority, specifically 72%, expressed a preference for modernizing COBOL applications as their chosen path forward, rather than rip and replace (MicroFocus, 2022).

In addition, the shrinking pool of COBOL-trained developers in the current IT market adds to system migration challenges. Accurately interpreting and translating legacy COBOL code into a new language requires a deep understanding of COBOL, which is becoming less common among developers. The scarcity of skilled COBOL developers complicates the migration process and increases the risk of errors (Sneed, 2013). In 2020, the average age of a COBOL developer was approximately 50 years and 34 weeks, and this age continues to increase with each passing year (MicroFocus, 2020). This aging trend among COBOL developers underscores the growing need for strategies to address the potential shortage of expertise in maintaining and modernizing critical COBOL-based systems. Moreover, COBOL is scarcely taught in academic settings, with only 37 universities worldwide offering dedicated mainframe courses (Botella, 2020). This limited exposure to COBOL in education, coupled with the aging workforce, raises concerns about the sustainability of the vast banking infrastructure heavily reliant on COBOL expertise in the future, creating a pressing need for initiatives to bridge this knowledge gap (MicroFocus, 2020).

Migrating from an existing COBOL system to a new one often requires system downtime. Even with planned outages, there are significant costs and potential disruptions to normal business operations. Unforeseen issues during the migration process can prolong downtime, leading to higher costs and operational impacts (Sneed, 2001).

Surprisingly, the length of COBOL code is not the issue or a challenge that many might assume, given its status as an older language, and concerns about its efficiency. The research, which compared the length of code required to achieve the same functionality across various

programming languages, found that there was not a substantial difference. As Figure 9 illustrates, COBOL's code length falls within the average range. A GitHub repository was used to come at this conclusion. The repository that was used for this investigation was: https://github.com/q60/rot13/tree/main.

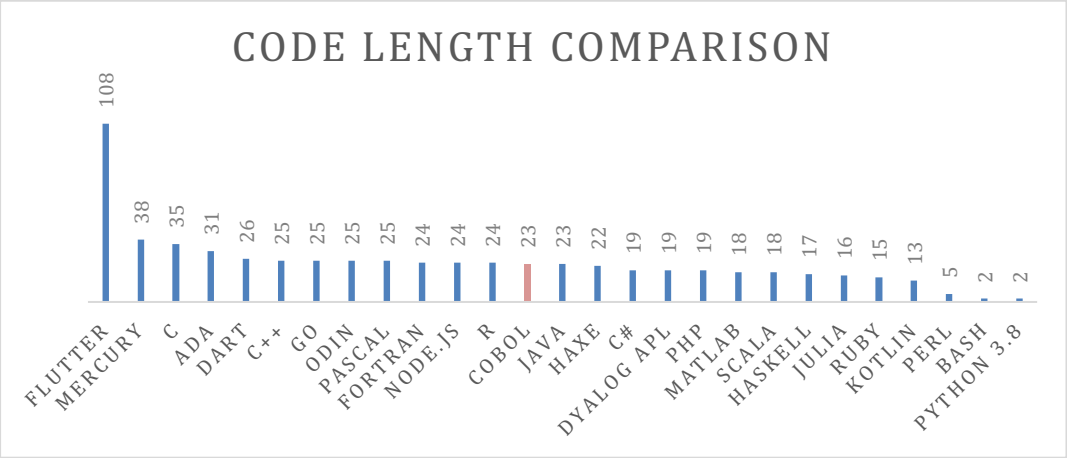


Figure 9. Code length comparison

3.3.3 Transforming Strategies and Modernizing COBOL Applications

To address the challenges of migrating from COBOL, organizations can employ strategies to smooth the transition process.

First, an incremental migration approach minimizes risks and disruptions to operations. This method involves gradually replacing parts of the COBOL system, while allowing the remaining sections to continue functioning. By adopting an incremental approach, organizations can systematically test and validate the new components without affecting the operation of the legacy system. This approach reduces the overall risk associated with migration (Sneed, 2013).

Secondly, automated code conversion tools can streamline the translation of COBOL code into other languages. Although these tools may not capture all the nuances of the original COBOL code, they significantly reduce the time and effort required for manual translation. The resulting code can serve as a baseline for the new system, which can then be refined and optimized as needed (Bodhuin et al., 2003).

In addition, investing in COBOL training for existing employees or hiring COBOL specialists can mitigate the challenges posed by the shrinking pool of COBOL expertise. With the necessary knowledge and expertise, the organization can accurately interpret and translate the legacy COBOL codebase, ensuring a smoother migration process. This approach also helps in the knowledge transfer between experienced COBOL developers and recruits (Erenkrantz et al., 2010).

To address the challenge of data compatibility, organizations can also leverage data migration tools. These tools automate the data mapping and transformation process, reducing the manual effort required and minimizing the risk of errors. Data migration tools streamline the conversion of data structures and types from COBOL formats to those used by modern languages and databases (Bodhuin et al., 2003).

Data migration offers a concrete solution to the challenges presented by legacy systems. It preserves historical data critical for compliance and continuity, enhances data accuracy through cleansing and transformation, and aligns data with modern technology standards.

This process minimizes downtime, supports scalability, reduces costs, and enhances data security and compliance. Modern systems resulting from data migration are easier to maintain and update, streamlining operations and ensuring organizations can embrace innovation and can adapt to changing business needs while safeguarding valuable data and ensuring business continuity.

Despite the challenges in migrating from COBOL to newer programming languages, organizations can effectively manage the transition process and ensure a successful migration with careful planning, strategic use of available tools, and the right resource investment.

3.4 COBOL Learning

This section delves into the world of COBOL compilers and available learning resources, exploring open-source and online resources that empower individuals to learn this enduring programming language. These valuable tools provide a gateway for learners, offering a range of materials and courses that enhance COBOL proficiency, thus contributing to its ongoing relevance in the ever-evolving technology landscape.

3.4.1 Exploration of COBOL resources

Like other languages, there are frameworks and libraries with common functions. IBM's Enterprise COBOL provides a suite of libraries that enhance the capabilities of the COBOL language. These libraries include robust data manipulation and file handling functionalities, essential for efficiently processing substantial amounts of data. Additionally, IBM offers libraries that facilitate integration with other systems commonly found in legacy ecosystems, such as DB2, IMS, and CICS (IBM, 2020). However, a license from IBM is necessary to obtain this tool, and it is not freely downloadable (IBM Enterprise COBOL for Z/OS, 2023.). As a result, its limited accessibility makes it less practical for learners.

Contrarily, GnuCOBOL, a free and open-source COBOL compiler, provides various libraries that enhance COBOL’s capabilities. These libraries cover string handling, mathematics, and system

interaction. While not as comprehensive as some commercial offerings, GnuCOBOL's libraries are valuable resources for the open-source COBOL community (GnuCOBOL - GNU Project, 2023.). Extensive research on the use of GnuCOBOL as an educational resource is presented in the next section.

3.4.2 Available Online Resources

By doing a simple search on google, it was determined that it is difficult to easily find sample programs on the internet. The website www.ibmmainframes.com and www.Github.com were one of the few websites to contain COBOL examples. A search using the keyword “COBOL” resulted in over 4200 repositories in GitHub. Many of them only contained the word “COBOL” but after filtering for repositories with actual COBOL code, it resulted in around 2700 repositories. For those specifically looking for COBOL samples, using “COBOL samples” as the keyword retrieved 53 repositories, providing sufficient resources to learn the basic fundamentals of COBOL programming. Few of the good repositories for educational purposes can be found in Appendix 9. Going through 50+ available sample repositories in GitHub, few were selected based on the completeness of fundamentals of COBOL programming.

Furthermore, a series of tests were performed to test the GnuCOBOL compiler's suitability for learning coding. These tests involved various COBOL programs which are present in Appendix 2, providing insights into the compiler's compatibility with different coding scenarios. The outcomes revealed that the GNUCOBOL compiler was successful in compiling and executing the tested COBOL programs, further emphasizing its appropriateness as a tool for learning COBOL programming for students. An only challenge currently is the limited availability of COBOL programs for educational purposes. One significant reason for this scarcity is the absence of COBOL courses in university curricula, especially in the Netherlands, where not a single university currently offers a course on COBOL.

The following programs given in table 1 have been tested using z/OS compiler 6.3 and GnuCOBOL compiler 3.1.2. The selected programs were chosen to encompass a wide range of COBOL programming constructs and functionalities, ensuring that the study's findings are representative of the language's diverse applications. Each program was selected based on its ability to highlight specific COBOL features, such as file handling, conditional statements, loops, arithmetic operations, and more.

In terms of compiler evaluation, the primary criterion was whether the code compiled successfully. The goal was to ensure that the compilers used in this study could handle the compilation of a variety of COBOL programs, encompassing both simple and complex code structures. While additional tests for execution results, memory usage, and performance were

not conducted, the successful compilation of the selected programs serves as a baseline evaluation of the compilers' ability to handle COBOL code effectively.

Table 1. Tested COBOL programs.

Category	**Program**	**z/OS COBOL 6.3**	**GnuCOBOL 3.1.2**
General	Current date	Yes	Yes
	Accept and display	Yes	Yes
	Dayfinder	Yes	Yes
	Mileage counter	Yes	Yes
Programming concept	If-else implementation	Yes	Yes
	Iteration	Yes	Yes
	Perform program	Yes	Yes
	Multiplier	Yes	Yes
Data structure and algorithm	Fibonacci sequence	Yes	Yes
	Prime number checker	Yes	Yes
	Random number generator	Yes	Yes
	Sort	Yes	Yes
Specific problem/solution	Y2k solution	Yes	Yes
	GOTO1	Yes	Yes
	GOTO1 Solution	Yes	Yes
	GOTO2	Yes	Yes
	GOTO2 Solution	Yes	Yes
String/Text-Based	Xml generator 1	Yes	No
	Xml-Generator GnuCOBOL	Yes	Yes
Playful	99 Bottles of Beer 1	Yes	Yes
	99 Bottles of Beer 2	Yes	Yes
Overall (GitHub) https://github.com/neopragma/COBOL-samples/tree/main/src/main/COBOL	Attract	Yes	Yes
	Brakes	Yes	Yes
	Cond88	Yes	Yes
	CPSEQFR	Yes	Yes
	CPSEQVR	Yes	Yes
	Date1	Yes	Yes
	Date2	Yes	Yes
	Hello	Yes	Yes
	Hex2text	Yes	Yes
	Ifeval	Yes	Yes
	Invcalc	Yes	Yes
	Moveme	Yes	Yes
	Notbool	Yes	Yes
	Notbool	Yes	Yes
	Reformer	Yes	Yes
	stringt	Yes	Yes
https://github.com/JohnDovey/GNUCOBOL-Samples	Day-from-date	No	No
	Colors	Yes	Yes
https://github.com/q60/rot13/tree/main	Encoding	Yes	Yes

The codes of some of these programs are given in Appendix 2. All these tested programs mentioned in table 1 work with both compilers except the Xml generator 1. The given COBOL xml generator 1 program is designed to process data and generate XML output. It has been tested by IBM on GnuCOBOL compiler and Enterprise z/OS compiler. As this program required JCL it was not supported by Gnu Compiler.

GnuCOBOL did not offer support for JCL execution, a crucial aspect when working with COBOL and mainframes. The Hercules emulator (Hercules, 2015) seemed to be a suitable option. But when the software was downloaded an installation failure occurred. This failure occurred due to the TN3270 terminal emulator being incompatible with the Mac OS Venture Version 13.5.2, the computer used during research (Appendix 7). TN3270 acts as the interface between the mainframe emulated by Hercules and the user. A setback emerged from the outdated documentation, with the most recent updates dating back to 2015 (Hercules, 2015). The absence of current documentation made it difficult to resolve issues and understand the emulator's functionality in the context of contemporary operating systems. This lack of up-to-date resources significantly hindered the progress of the research, and prevented fully exploring the emulator's capabilities, which would have made it possible to execute COBOL programs that rely on JCLs.

By integrating COBOL into academic syllabuses, there could be an uptick in interest, leading to more sample codes being shared online. This would also result in a larger pool of individuals proficient in COBOL, potentially addressing the existing shortage of COBOL programmers, as evidenced by the numerous unfilled job vacancies in the field. Even though universities in the Netherlands do not provide any courses regarding Cobol, on YouTube there can be found several good tutorials. Out of these, few tutorials were selected for inclusion in Appendix 9, primarily due to their comprehensive coverage of fundamental programming concepts. Among these resources, "COBOL Tutorial: Learn COBOL in One Video" by Derek Banas stands out. This tutorial comprehensively covers essential COBOL concepts, enhancing understanding with practical examples and sample programs.

While the other tutorials offer a strong theoretical foundation, Derek Banas' tutorial's inclusion of practical samples makes it exceptionally useful for those seeking a hands-on approach to learning COBOL programming. Moreover, four courses related to COBOL were selected from Udemy, filtered on their relevance.

In conclusion, the compiler can be easily downloaded by following official Gnu Compiler documentation. Currently GnuCOBOL compiler version 3.1.2 is available, and it is an accessible and user-friendly solution for compiling and running simple COBOL programs. Even though the compiler might exhibit limitations in dealing with complex programs and batch processing, it remains a precious resource for learning COBOL and practicing rudimentary program execution. Integration with JCL expertise becomes pivotal for companies with significant batch processing demands. The exhaustive testing carried out affirmed the GNUCOBOL compiler's efficiency for learning coding and its ability to compile and execute COBOL programs in simpler scenarios. Batch processing in large companies is indispensable for effective data management and processing, offering benefits such as data integrity, system performance, scalability, and cost-effectiveness. GnuCOBOL primarily serves as a COBOL compiler and runtime system and is not

equipped for the direct execution of JCL (Job Control Language). To perform JCL execution, a mainframe emulator is indispensable, and one such emulator is Hercules. According to Hercules documentation, it does enable the execution of JCLs. Therefore, it is essential to recognize that GnuCOBOL alone cannot fully unlock the potential of COBOL in a mainframe context, as the use of a mainframe emulator like Hercules is essential for comprehensive COBOL and JCL experience. GnuCOBOL is an excellent choice for learning COBOL programming in combination with online resources like GitHub sample codes and YouTube tutorials. It provides a straightforward and accessible environment for beginners to grasp the fundamentals of the COBOL language, understand its syntax, and practice coding. GnuCOBOL's simplicity and ease of use make it an ideal tool for educational purposes and for those who are just starting to explore COBOL.

3 Conclusion

COBOL continues to hold significant relevance in today's IT environment, even with the emergence and spread of newer programming languages. COBOL's strong capability, including its reliability, efficiency, scalability, and compatibility, make it a durable choice for industries that rely on dependable and efficient data processing. It is still one of the popular languages because it is reliable and consistent as it has been refined over time. Regular updates keep it secure and up to date with new business requirements. A bank case study showed that it works well with handling large amounts of data efficiently and it is there for a long run. However, this bank is also facing challenges due to lack of COBOL experts. Despite the presence of more modern programming languages, COBOL remains among the top 30 in popularity indexes, affirming its continued relevance. The job market reflects its importance, with job listings seeking COBOL skills. In summary, COBOL's applications persist, emphasizing its resilience and enduring relevance across various sectors.

Despite the myth of COBOL being irrelevant and prediction of phasing out decades ago, there is still a significant demand for experts within companies still utilizing this programming language. On one hand, companies address this demand by providing internal training. On the other hand, many companies are migrating from COBOL to newer programming languages or modernizing, however, this process comes with several challenges. One major obstacle is preserving the complex business logic within COBOL applications, which often lack proper documentation, making it difficult to translate these intricate rules. The scarcity of developers trained in COBOL further compounds the difficulty. Data compatibility is crucial, but it can be addressed with data migration tools. To cope with these challenges, organizations can use strategies such as incremental migration, gradually replacing parts of the COBOL system to minimize risks. Automated code conversion tools simplify the translation process, and investing in COBOL training or hiring specialists helps with expertise shortages. These strategies help organizations in transitioning to newer languages while safeguarding their crucial functionality and data integrity. The likelihood of a decrease in COBOL's relevance is growing, given that businesses are in the process of migration.

This shift is noticeable in how educational institutions approach COBOL, with a clear lack of courses including this programming language. There is a decreasing emphasis on creating new educational programs solely for COBOL, as it is recognized that the current demand for COBOL experts can be met effectively through available resources. Notably, resources like GNUCOBOL and numerous online tutorials are now comprehensive and accessible platforms meeting the educational needs associated with COBOL expertise. As the educational landscape adapts to changing programming language trends, these online resources play a crucial role in helping individuals learn COBOL proficiently.

4 Limitations and Further Research

5.1 Limitations

The first limitation of this study was the scarcity of data and statistics specific to COBOL's contemporary role. The dominance of newer programming languages has led to a lack of recent, comprehensive research and quantitative data on the language's utilization and relevance. Most of the existing literature primarily addresses the historical aspects of COBOL and the challenges associated with its maintenance or migration. The absence of up-to-date surveys or statistical analysis makes it challenging to quantitatively assess the extent of COBOL's use in modern information systems, as well as its role in different sectors. The limitation of search filters and the focus on web pages rather than academic papers further restricts access to scholarly data, leaving gaps in the understanding of COBOL's current standing in the ever-evolving landscape of IT systems.

Secondly, quantifying the relevance of COBOL has proven to be challenging, primarily because it is used in legacy systems within industries such as banking, insurance, and government. These sectors often refrain from disclosing specific details about their IT infrastructure due to security concerns. As a result, there is a scarcity of concrete data available regarding the current usage and dependence on COBOL in these sectors. Additionally, during an interview with a leading bank, it was stipulated that any information or insights obtained from the interviews could only be included in the study anonymously.

Thirdly, assessing the real-world demand for COBOL skills in the job market posed a limitation. While job postings are requiring COBOL, this does not capture the full picture. Many companies may require COBOL skills but might not list them explicitly in job postings, instead preferring to offer on-the-job training for the necessary COBOL skills.

Lastly, numerous vacancies were identified on LinkedIn and Indeed. However, a substantial portion of them proved to be irrelevant to the research focus. Additionally, the limited filtration options on the platform contributed to the inclusion of irrelevant data, thereby impacting the accuracy of the collected information.

5.2 Further research

This research provides invaluable insights into the ongoing relevance of COBOL in the modern IT landscape. However, there are several areas where further research could provide more comprehensive knowledge and deepen understanding regarding the relevance of COBOL.

1. In the current study, the Hercules Emulator could not be used, which made it impossible to research whether this emulator supports JCL execution and is a better alternative to GnuCOBOL. Therefore, it would be advisable to consider using the Hercules emulator for the machine instead of GnuCOBOL.
2. To improve the of future research, the data regarding the COBOL job market should be refined through additional filtration.
3. COBOL in Education: Researching the role and presence of COBOL in computer science education could provide important insights into the future supply of COBOL developers. Understanding whether and how COBOL is being taught to future developers can inform strategies for addressing the apparent skills gap in this area.
4. COBOL and Emerging Technologies: Further research could be undertaken to explore how COBOL can be integrated with modern technologies. Given the ongoing importance of COBOL in many systems, understanding how it can work alongside or within cloud computing, artificial intelligence, and other emerging technologies could be valuable.

The ongoing relevance of COBOL presents a rich area for further investigation. These research directions could provide a more holistic understanding of COBOL's role and importance in today's digital landscape, its challenges, and outlook.

5 References

- Asay, M. (2018, August 27). All The Rich Kids Are Into COBOL—But Why? ReadWrite. https://readwrite.com/cobol-programming-language-hot-or-not/
- Astadia. (2023, August 1). Migrating COBOL to Java with Automated Conversion. Astadia. https://www.astadia.com/blog/migrating-cobol-to-java-with-automated-conversion
- Back to basic met COBOL - werken.belastingdienst.nl - Werken bij de Belastingdienst. (2023). Werken bij de Belastingdienst. https://werken.belastingdienst.nl/nieuws-en-artikelen/back-to-basic-met-COBOL-6
- Bailey, C. (2023, September 18). The Challenges of Modifying COBOL Applications | Blog | FairCom. FairCom. https://www.faircom.com/insights/the-challenges-of-modifying-cobol-applications
- Bodhuin, T., Guardabascio, E., & Tortorella, M. (2003). Migration of non-decomposable software systems to the web using screen proxies. https://doi.org/10.1109/wcre.2003.1287247
- Botella, E. (2020, April 9). Why New Jersey's unemployment insurance system uses a 60-Year-Old programming language. Slate Magazine. https://slate.com/technology/2020/04/new-jersey-unemployment-cobol-coronavirus.html
- C. Cave, W., E. Wassmer, R., T. Irvine, K., & F. Ledgard, H. (2017). A Disruptive Solution to Parallel Processing. University of Toledo. http://www.visisoft.com/PDF_Files/DisruptiveSolutionToHPC.pdf
- Ciborowska, A., Chakarov, A., & Pandita, R. (2021, September). Contemporary COBOL: developers' perspectives on defects and defect location. In 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 227-238). IEEE.
- CNN. (2020). Wanted urgently: People who know a half-century-old computer language so states can process unemployment claims. https://edition.cnn.com. Retrieved October 15, 2023, from https://edition.cnn.com/2020/04/08/business/coronavirus-cobol-programmers-new-jersey-trnd/index.html
- De Marco, A., Iancu, V., & Asinofsky, I. (2018). COBOL to Java and Newspapers Still Get Delivered. IEEE Software. https://doi.org/10.1109/icsme.2018.00055
- Dijkstra, E. (1968, March). Edgar Dijkstra: Go To statement Considered Harmful. https://homepages.cwi.nl. Retrieved July 3, 2023, from https://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.pdf
- Dorninger, B., Moser, M., & Pichler, J. (2017). Multi-language re-documentation to support a COBOL to Java migration project. International Conference on Software Analysis, Evolution and Reengineering (SANER). https://doi.org/10.1109/saner.2017.7884669

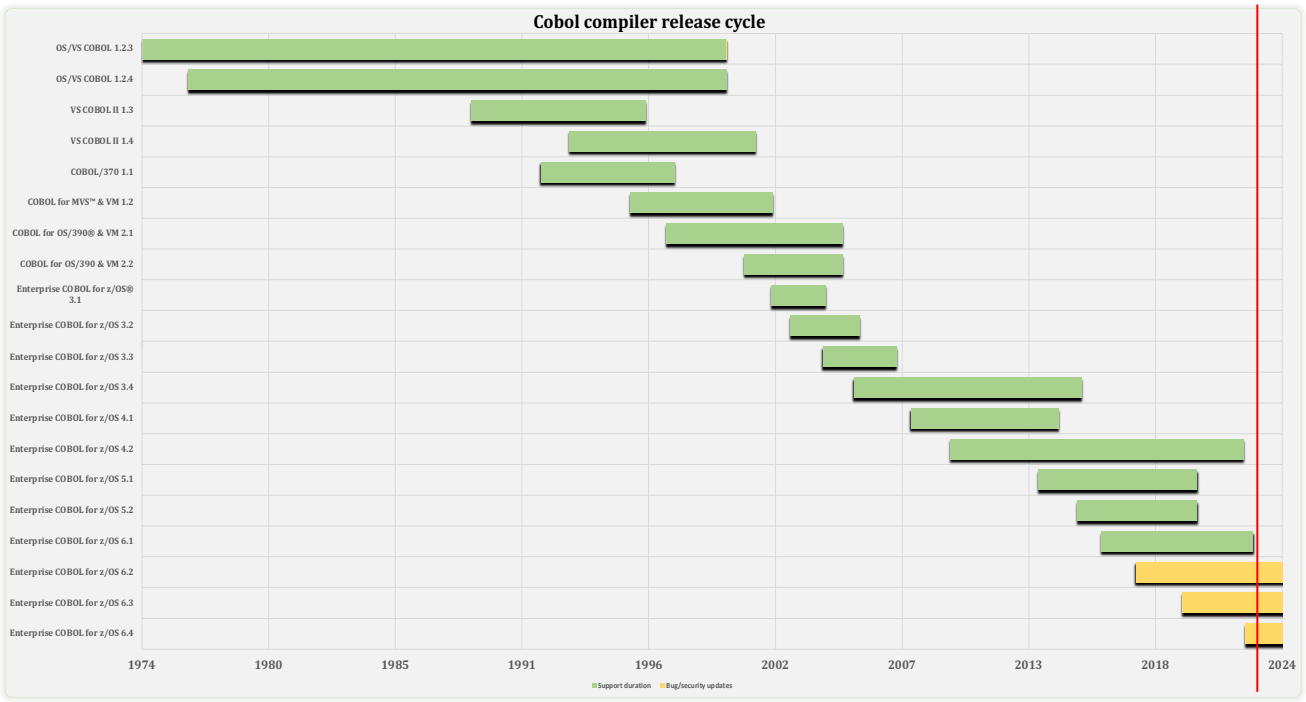
- Dubov, A. (2023, May 28). The secret failure in the banking system of migrating from COBOL. Medium. https://medium.com/@alxdubov/the-failure-of-the-banking-system-to-migrate-from-cobol-is-a-complex-issue-with-many-factors-7189279d7181
- Enlyft. (2023, March 8). COBOL commands 0.48% market share in Programming Languages. Retrieved May 10, 2023, from https://enlyft.com/tech/products/COBOL
- Ensmenger, N. (2011). The computer boys take over: computers, programmers, and the politics of technical expertise. Choice Reviews Online, 48(06), 100. https://doi.org/10.5860/choice.48-3324
- Ghoshal, A. (2023, 22 augustus). IBM WatsonX to use generative AI to translate COBOL code into Java. InfoWorld. https://www.infoworld.com/article/3705251/ibm-watsonx-to-use-generative-ai-to-translate-COBOL-code-into-java.html
- Glass, R. L. (1997, September). COBOL - A Contradiction and an Enigma. Communications of the ACM, 40(9). Retrieved from https://dl.acm.org/doi/pdf/10.1145/260750.260752
- GnuCOBOL - GNU Project. (2023). Retrieved from https://GnuCOBOL.sourceforge.io/
- Grace Hopper and the invention of the information age. (2010). Choice Reviews Online, 47(06), 47–3208. https://doi.org/10.5860/choice.47-3208
- Hercules. (2015, November 30). Hercules – Installation Guide Version 3.12. https://hercdoc.glanzmann.org. https://hercdoc.glanzmann.org/V312/HerculesInstallation.pdf
- IBM Documentation. (2022). Retrieved from https://www.ibm.com/docs/en/COBOL-zos/6.3?topic=appendixes-option-comparison
- IBM Enterprise COBOL for z/OS. (2023.). https://www.ibm.com/products/cobol-compiler-zos
- IBM. (2023). CICS Transaction Server for z/OS. Retrieved May 10, 2023, from https://www.ibm.com/docs/en/SSGMCP_5.5.0/pdf/java-applications_pdf.pdf
- Jones, C. (1997, January). The global economic impact of the year 2000 software problem. Retrieved from https://www.govinfo.gov/content/pkg/GPO-CPRT-106sprt10/pdf/GPO-CPRT-106sprt10-6.pdf
- Kappelman, L. A. (2000, March/April). Some Strategic Y2K Blessings. IEEE Software. Retrieved from https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=3fa057087d13a94bb745bc29c9f1cf1a17a626a1
- Marcotty, M., & Ledgard, H. (1987). The world of programming languages. Springer books on professional computing. https://doi.org/10.1007/978-1-4612-4692-3
- Micro Focus. (2022, januari). How much COBOL is out there. Brighttalk. Geraadpleegd op 12 september 2023, van https://www.brighttalk.com/resource/core/379731/how-much-COBOL-is-really-out-there_819568.pdf

- Murach, M. (2001, February). Is COBOL Dying ... or Thriving? The COBOL Newswire. Retrieved from https://dl.acm.org/doi/pdf/10.1145/342251.342256
- No, COBOL is not a dead language. (2021, February 8). Data Center Knowledge | News and Analysis for the Data Center Industry. https://www.datacenterknowledge.com/design/no-cobol-not-dead-language
- PYPL PopularitY of Programming Language index. (2023). Retrieved from https://pypl.github.io/PYPL.html
- Reis, R. (2021). Proposal of a learning design model developed for the creation of training courses: COBOL Programming Course Case Study. https://eric.ed.gov/?id=ED621914
- Ricciuti, M. (1998, January 6). CA ships another Y2K tool. CNET. Retrieved from https://www.cnet.com/tech/tech-industry/ca-ships-another-y2k-tool/
- Rodríguez, J. M., Crasso, M., Mateos, C., Zunino, A., & Campo, M. (2011). Bottom-up and top-down COBOL system migration to Web Services: An experience report. IEEE Internet Computing. https://doi.org/10.1109/mic.2011.162
- Ryan, M. (2022, January 5). Back to the Future with Codex and COBOL - Towards Data Science. Medium. https://towardsdatascience.com/back-to-the-future-with-codex-and-cobol-766782f5ae8f
- Sammet, J. E. (1978). The early history of COBOL. Sigplan Notices, 13(8), 121–161. https://doi.org/10.1145/960118.808378
- Sneed, H. M. (2001). Extracting business logic from existing COBOL programs as a basis for redevelopment. https://doi.org/10.1109/wpc.2001.921728
- Sneed, H. M. (2009). A pilot project for migrating COBOL code to web services. International Journal on Software Tools for Technology Transfer, 11(6), 441–451. https://doi.org/10.1007/s10009-009-0128-z
- Sneed, H. M., & Erdoes, K. (2013). Migrating AS400-COBOL to Java: A Report from the Field. https://doi.org/10.1109/csmr.2013.32
- Sneed, H. M., & Verhoef, C. (2020). From COBOL to Business Rules—Extracting Business Rules from Legacy Code. Integrating Research and Practice in Software Engineering, 187-208.
- Stack Overflow Developer Survey 2022. (2022.). Stack Overflow. https://survey.stackoverflow.co/2022/#most-popular-technologies-language
- Suganuma, T., Yasue, T., Onodera, T., & Nakatani, T. (2008). Performance pitfalls in large-scale java applications translated from COBOL. Companion to the 23rd ACM SIGPLAN Conference on Object-oriented Programming Systems Languages and Applications. https://doi.org/10.1145/1449814.1449822

- The Cloud for Mainframe & COBOL: Migration & Modernization. (2023). https://www.microfocus.com/en-us/destination-cloud
- Thibodeau, P. (2013, April 8). Should universities offer Cobol classes? Computerworld. https://www.computerworld.com/article/2828320/should-universities-offer-cobol-classes-.html
- TIOBE Index - TIOBE. (2022, June 3). TIOBE. https://www.tiobe.com/tiobe-index/
- Tmaxsoft. (2021). Tmaxsoft OpenFrame: A Modern Platform for COBOL Applications. Retrieved May 10, 2023, from https://www.tmaxsoft.com/wp-content/uploads/TmaSof_eBook_OpenFrame_2021.pdf
- TmaxSoft. (2022, November 10). Mainframe Modernization | OpenFrame| TMAXSoft. Retrieved from https://www.tmaxsoft.com/products/openframe/
- Van Assen, M., Ntagengerwa, M. A., Sayilir, Ö., & Zaytsev, V. (2023). Crossover: Towards Compiler-Enabled COBOL-C Interoperability. Association for Computing Machinery. https://doi.org/10.1145/3624007.3624055
- Wayner, P. (2022, June 27). Why programmers still love COBOL. TechBeacon. https://techbeacon.com/enterprise-it/why-programmers-still-love-cobol#:~:text=%22It%27s%20still%20here%20primarily%20because,build%20on%20top%20of%20it.%22
- Young, D. A. (2000). Reflection on Y2K. PM Network, 14(7), 37–41.

6 Appendices

Appendix 1: COBOL compiler release cycle



The Release Cycle Giant Chart provides a comprehensive overview of various compilers and their respective life cycles. This visually striking chart effectively distinguishes between support duration, denoted in green, and currently supported versions, represented in yellow and a vertical red line indicating the current timestamp.

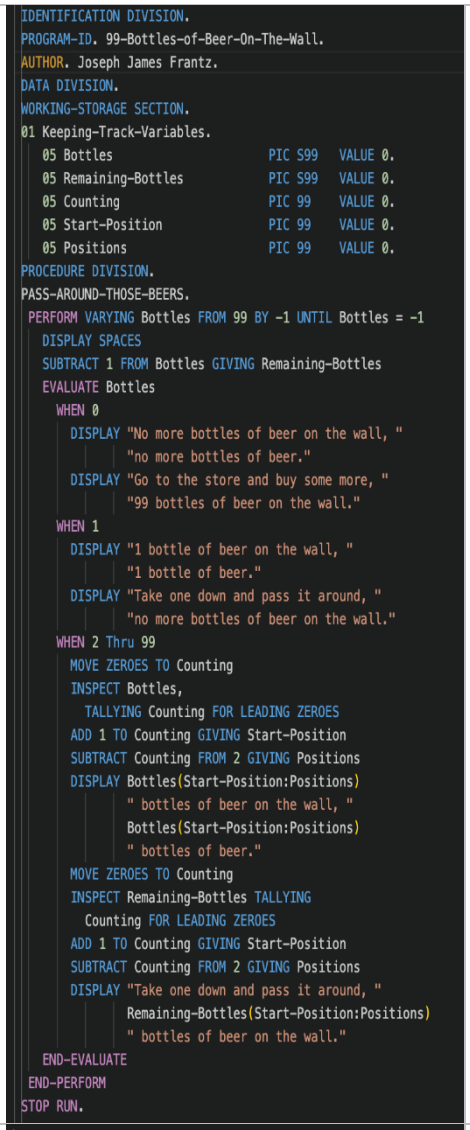
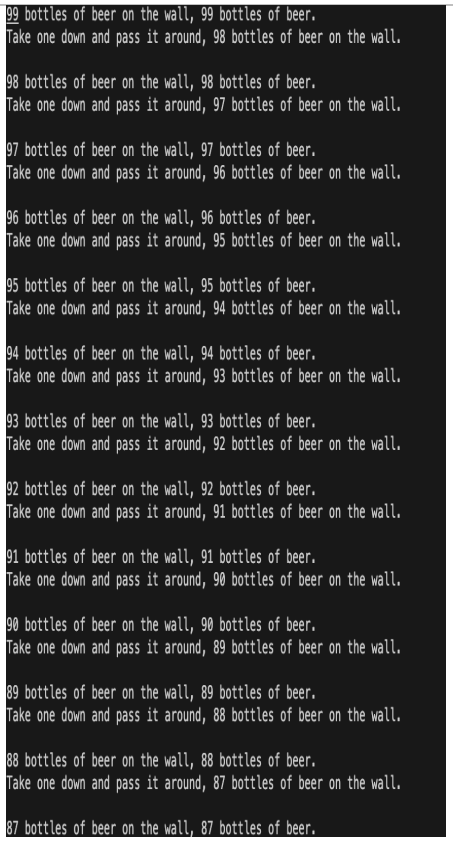
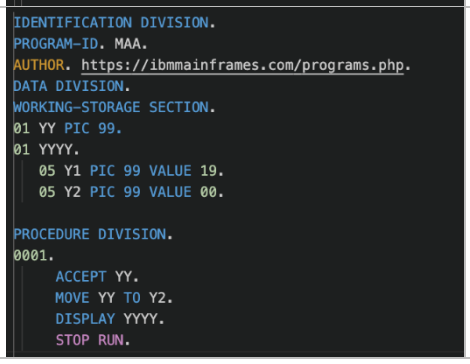
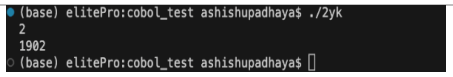
Appendix 2: COBOL test results

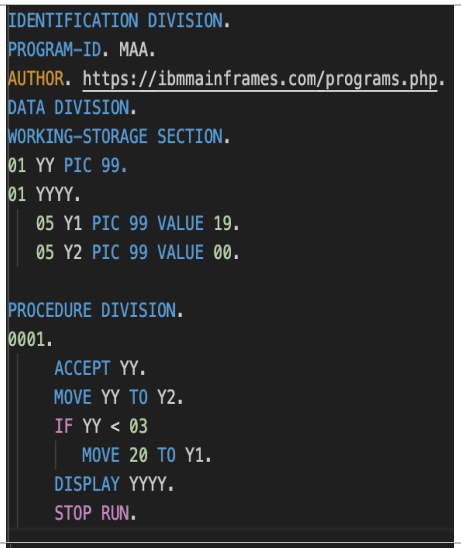
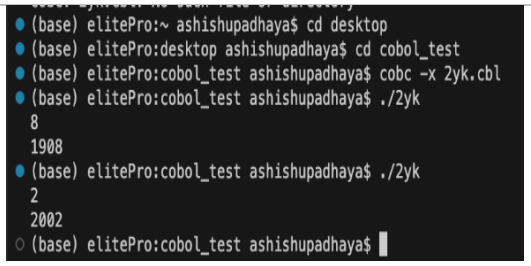
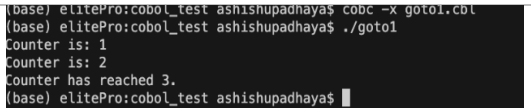
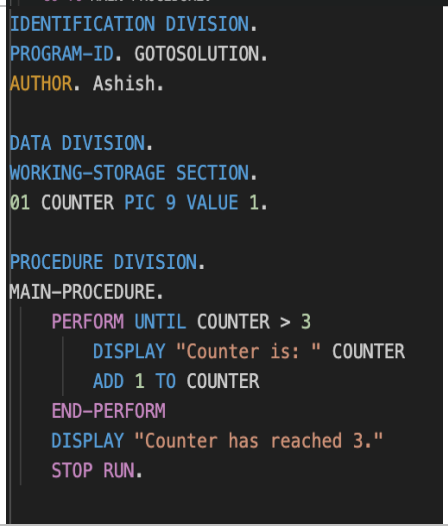
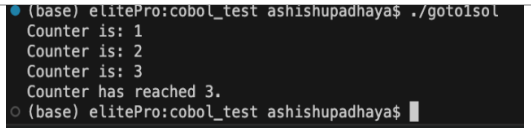
The below programs work with GnuCOBOL Compiler 3.1.2 and Enterprise z/OS 6.3.

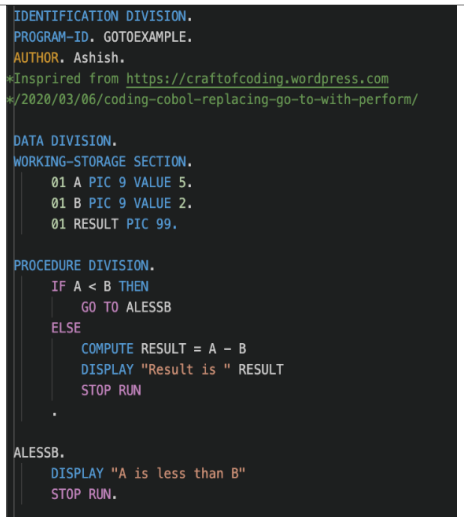
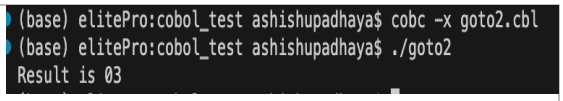
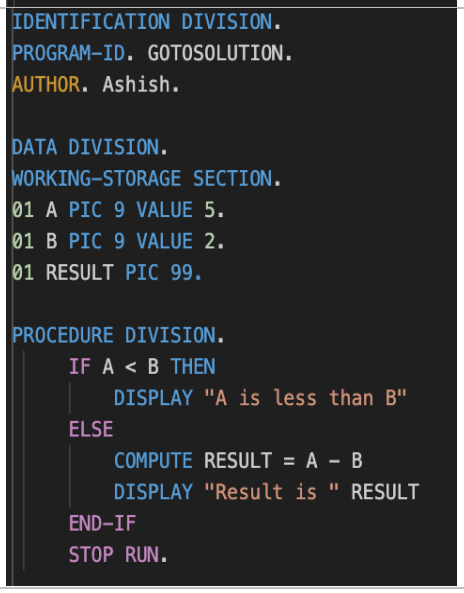
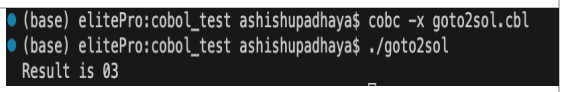
Program name	Code	Result

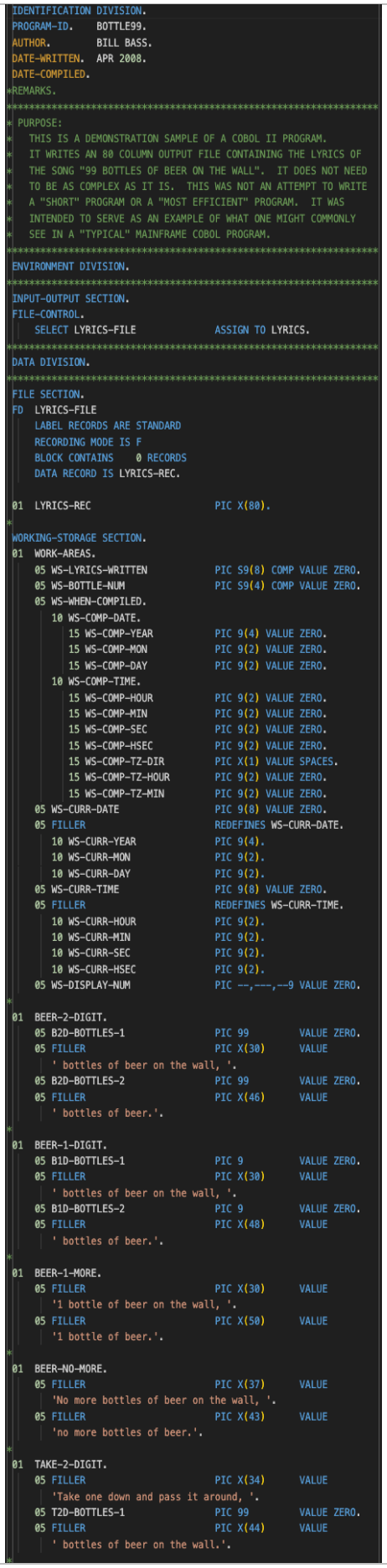
Xml- generator	IDENTIFICATION DIVISION.
	PROGRAM-ID. GJ758.
	Author.Ashish.
	ENVIRONMENT DIVISION.
	CONFIGURATION SECTION.
	SOURCE-COMPUTER. IBM-370.
	OBJECT-COMPUTER. IBM-370.
	SPECIAL-NAMES.
	DECIMAL-POINT IS COMMA.
	INPUT-OUTPUT SECTION.
	FILE-CONTROL.
	SELECT XML-FILE ASSIGN TO GJ758O01.
	SELECT SSF-FILE ASSIGN TO GJ758I01.
	SELECT XML-LOGS ASSIGN TO GJ758L01.
	DATA DIVISION.
	FILE SECTION.
	FD XML-FILE
	BLOCK CONTAINS 0.
	01 XML-OUTPUT PIC X(2600).
	FD XML-LOGS
	BLOCK CONTAINS 0.
	01 XML-LOG-OUTPUT PIC X(100).
	FD SSF-FILE
	BLOCK CONTAINS 0.
	
	01 GJ-XMLINPUT.
	03 IN-001 PIC X(50).
	03 IN-002 PIC X(12).
	03 IN-003 PIC X(15).
	03 IN-004 PIC X(12).
	
	01 SSF-XML-REC.
	03 RC.
	05 OUT-001 PIC X(50).
	05 OUT-002 PIC X(12).
	05 OUT-003 PIC X(15).
	05 OUT-004 PIC X(12).
	
	2110-MANUAL-COPY-INPUT.
	INITIALIZE SSF-XML-REC
	MOVE IN-001 TO INPUT-ALPHANUM
	PERFORM CHECK-ALPHANUM
	MOVE REPLY-ALPHANUM
	TO OUT-001
	MOVE IN-002 TO INPUT-ALPHANUM
	PERFORM CHECK-ALPHANUM
	MOVE REPLY-ALPHANUM
	TO OUT-002
	MOVE IN-003 TO INPUT-ALPHANUM
	PERFORM CHECK-ALPHANUM
	MOVE REPLY-ALPHANUM
	TO OUT-003
	MOVE IN-004 TO INPUT-ALPHANUM
	PERFORM CHECK-ALPHANUM
	MOVE REPLY-ALPHANUM
	TO OUT-004
	.
	2400-XML-PARA.
	XML GENERATE XML-OUTPUT FROM SSF-XML-REC
	COUNT IN WS-COUNT
	WITH ENCODING 1047
	NAME OF
	SSF-XML-REC IS 'ProductIsForCounterparty'
	OUT-001 IS 'Product'
	OUT-002 IS 'ProductSourceSystemIdentifier'
	OUT-003 IS 'Counterparty'
	OUT-004 IS 'CounterpartySourceSystemIdentifier'
	SUPPRESS
		<ProductIsForCounterparty><Product>122</Product><ProductSourceSystemIdentifier>aab.sys1</ProductSourceSystemIdentifier><Counterparty>b09484</Counterparty><CounterpartySourceSystemIdentifier>AAB</CounterpartySourceSystemIdentifier></ProductIsForCounterparty>
		
		Required JCl to execute the program.
		
		//GJ758 JOB GJ000000, MSGCLASS=G,CLASS=P,
		NOTIFY=ashish
		//STEP01 EXEC PGM=GJ758
		//STEPLIB DD DSN=LBPS$.PR.LOADLIB, DISP=SHR
		//GJ758I01 DD DSN=ZJ000.sortout.pmgj768,
		// GJ758O01 DD DSN=zjooo.sortout.pmgj768.outputfile
		// DISP=(NEW,CATLG,DELETE),SPACE=(CYL,(1,1),RLSE),
		// DCB=(RECFM=FB,LRECL=40,BLKSIZE=0)
		//SYSPRINT DD SYSOUT=*
		//SYSOUT DD SYSOUT=A
		//

	EVERY NONNUMERIC ELEMENT WHEN SPACES . EXIT. END PROGRAM GJ758.	
XMl-Generator GnuCOBOL	IDENTIFICATION DIVISION. PROGRAM-ID. XMLGENERATOR. Author. Ashish. DATA DIVISION. WORKING-STORAGE SECTION. 01 XML-OUTPUT PIC X(500). 01 XML-TEMPLATE. 05 XML-DECLARATION PIC X(40) VALUE '<?xml version="1.0" encoding="UTF-8"?>'. 05 XML-ROOT-ELEMENT PIC X(500) VALUE '<COBOLthesis>'. 05 XML-PERSON-ELEMENT PIC X(500) VALUE ' <person>'. 05 XML-NAME-ELEMENT PIC X(500) VALUE ' <name>ashish</name>'. 05 XML-AGE-ELEMENT PIC X(500) VALUE ' <age>24</age>'. 05 XML-END-TAG PIC X(500) VALUE ' </person>'. 05 XML-END-ROOT-ELEMENT PIC X(500) VALUE '</COBOLthesis>'. PROCEDURE DIVISION. MOVE XML-DECLARATION TO XML-OUTPUT XML GENERATE XML-OUTPUT FROM XML-TEMPLATE DISPLAY XML-OUTPUT STOP RUN.	<?xml version="1.0" encoding="UTF-8"?> <COBOLthesis> <person> <name>ashish</name> <age>24</age> </person> </COBOLthesis>

99 Bottles of Bears 1		
Y2k problem		

Y2k solution		
GOTO1		
GOTO1 Solution		

GOTO2		
GOTO2 Solution		

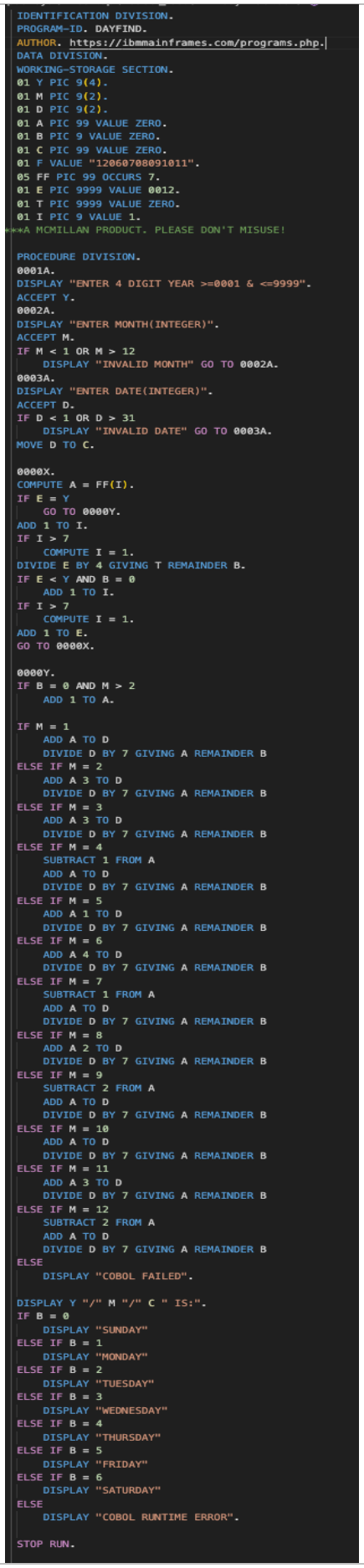
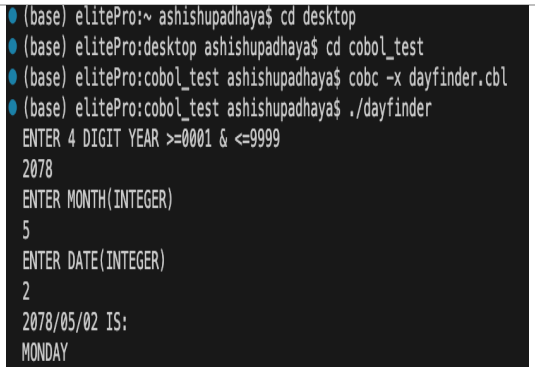
99 Bottle of bear 2		

```
IDENTIFICATION DIVISION.
PROGRAM-ID.    BOTTLE99.
AUTHOR.        BILL BASS.
DATE-WRITTEN.  APR 2008.
DATE-COMPILED.
*REMARKS.
*****************************************************************
* PURPOSE:
*   THIS IS A DEMONSTRATION SAMPLE OF A COBOL II PROGRAM.
*   IT WRITES AN 80 COLUMN OUTPUT FILE CONTAINING THE LYRICS OF
*   THE SONG "99 BOTTLES OF BEER ON THE WALL".  IT DOES NOT NEED
*   TO BE AS COMPLEX AS IT IS.  THIS WAS NOT AN ATTEMPT TO WRITE
*   A "SHORT" PROGRAM OR A "MOST EFFICIENT" PROGRAM.  IT WAS
*   INTENDED TO SERVE AS AN EXAMPLE OF WHAT ONE MIGHT COMMONLY
*   SEE IN A "TYPICAL" MAINFRAME COBOL PROGRAM.
*****************************************************************
ENVIRONMENT DIVISION.
*****************************************************************
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT LYRICS-FILE              ASSIGN TO LYRICS.
*****************************************************************
DATA DIVISION.
*****************************************************************
FILE SECTION.
FD  LYRICS-FILE
    LABEL RECORDS ARE STANDARD
    RECORDING MODE IS F
    BLOCK CONTAINS    0 RECORDS
    DATA RECORD IS LYRICS-REC.

01  LYRICS-REC                     PIC X(80).
*
WORKING-STORAGE SECTION.
01  WORK-AREAS.
    05 WS-LYRICS-WRITTEN           PIC S9(8) COMP VALUE ZERO.
    05 WS-BOTTLE-NUM               PIC S9(4) COMP VALUE ZERO.
    05 WS-WHEN-COMPILED.
       10 WS-COMP-DATE.
          15 WS-COMP-YEAR          PIC 9(4) VALUE ZERO.
          15 WS-COMP-MON           PIC 9(2) VALUE ZERO.
          15 WS-COMP-DAY           PIC 9(2) VALUE ZERO.
       10 WS-COMP-TIME.
          15 WS-COMP-HOUR          PIC 9(2) VALUE ZERO.
          15 WS-COMP-MIN           PIC 9(2) VALUE ZERO.
          15 WS-COMP-SEC           PIC 9(2) VALUE ZERO.
          15 WS-COMP-HSEC          PIC 9(2) VALUE ZERO.
          15 WS-COMP-TZ-DIR        PIC X(1) VALUE SPACES.
          15 WS-COMP-TZ-HOUR       PIC 9(2) VALUE ZERO.
          15 WS-COMP-TZ-MIN        PIC 9(2) VALUE ZERO.
    05 WS-CURR-DATE                PIC 9(8) VALUE ZERO.
    05 FILLER                      REDEFINES WS-CURR-DATE.
       10 WS-CURR-YEAR             PIC 9(4).
       10 WS-CURR-MON              PIC 9(2).
       10 WS-CURR-DAY              PIC 9(2).
    05 WS-CURR-TIME                PIC 9(8) VALUE ZERO.
    05 FILLER                      REDEFINES WS-CURR-TIME.
       10 WS-CURR-HOUR             PIC 9(2).
       10 WS-CURR-MIN              PIC 9(2).
       10 WS-CURR-SEC              PIC 9(2).
       10 WS-CURR-HSEC             PIC 9(2).
    05 WS-DISPLAY-NUM              PIC --,---,--9 VALUE ZERO.
*
01  BEER-2-DIGIT.
    05 B2D-BOTTLES-1               PIC 99         VALUE ZERO.
    05 FILLER                      PIC X(30)      VALUE
       ' bottles of beer on the wall, '.
    05 B2D-BOTTLES-2               PIC 99         VALUE ZERO.
    05 FILLER                      PIC X(46)      VALUE
       ' bottles of beer.'.
*
01  BEER-1-DIGIT.
    05 B1D-BOTTLES-1               PIC 9          VALUE ZERO.
    05 FILLER                      PIC X(30)      VALUE
       ' bottles of beer on the wall, '.
    05 B1D-BOTTLES-2               PIC 9          VALUE ZERO.
    05 FILLER                      PIC X(48)      VALUE
       ' bottles of beer.'.
*
01  BEER-1-MORE.
    05 FILLER                      PIC X(30)      VALUE
       '1 bottle of beer on the wall, '.
    05 FILLER                      PIC X(50)      VALUE
       '1 bottle of beer.'.
*
01  BEER-NO-MORE.
    05 FILLER                      PIC X(37)      VALUE
       'No more bottles of beer on the wall, '.
    05 FILLER                      PIC X(43)      VALUE
       'no more bottles of beer.'.
*
01  TAKE-2-DIGIT.
    05 FILLER                      PIC X(34)      VALUE
       'Take one down and pass it around, '.
    05 T2D-BOTTLES-1               PIC 99         VALUE ZERO.
    05 FILLER                      PIC X(44)      VALUE
       ' bottles of beer on the wall.'.
*
```

```
(base) elitePro:desktop ashishupadhaya$ cd cobol_test
(base) elitePro:cobol_test ashishupadhaya$ cobc -x 99BOttlesofBeer2.cbl
(base) elitePro:cobol_test ashishupadhaya$ ./99BottlesofBeer2
**************************************************************
**** BEGIN PROGRAM BOTTLE99
**** COMPILED: 2023/07/30 16:22:16.36
**** START AT: 2023/07/30 16:22:28.35
**************************************************************
*
**************************************************************
**** RUN STATISTICS FOR PROGRAM BOTTLE99
**************************************************************
*
* LYRICS RECORDS WRITTEN =       299
*
**************************************************************
**** END PROGRAM BOTTLE99
**** ENDED AT: 2023/07/30 16:22:28.36
**************************************************************
(base) elitePro:cobol_test ashishupadhaya$
```

```
*
01  TAKE-1-DIGIT.
    05 FILLER                    PIC X(34)      VALUE
      'Take one down and pass it around, '.
    05 T1D-BOTTLES-1             PIC 9          VALUE ZERO.
    05 FILLER                    PIC X(45)      VALUE
      ' bottles of beer on the wall.'.
*
01  TAKE-1-MORE.
    05 FILLER                    PIC X(34)      VALUE
      'Take one down and pass it around, '.
    05 FILLER                    PIC X(46)      VALUE
      '1 bottle of beer on the wall.'.
*
01  TAKE-NO-MORE.
    05 FILLER                    PIC X(34)      VALUE
      'Take one down and pass it around, '.
    05 FILLER                    PIC X(46)      VALUE
      'no more bottles of beer on the wall.'.
*
01  BUY-SOME-MORE.
    05 FILLER                    PIC X(35)      VALUE
      'Go to the store and buy some more, '.
    05 FILLER                    PIC X(45)      VALUE
      '99 bottles of beer on the wall.'.
*
01  BLANK-LINE                   PIC X(80)      VALUE SPACES.
****************************************************************
PROCEDURE DIVISION.
****************************************************************
    ACCEPT WS-CURR-DATE          FROM DATE YYYYMMDD
    ACCEPT WS-CURR-TIME          FROM TIME
    MOVE FUNCTION WHEN-COMPILED    TO WS-WHEN-COMPILED
*
    DISPLAY '************************************'
            '************************************'
    DISPLAY '**** BEGIN PROGRAM BOTTLE99'
    DISPLAY '**** COMPILED: '
           WS-COMP-YEAR '/' WS-COMP-MON '/' WS-COMP-DAY ' '
           WS-COMP-HOUR ':' WS-COMP-MIN ':'
           WS-COMP-SEC  '.' WS-COMP-HSEC
    DISPLAY '**** START AT: '
           WS-CURR-YEAR '/' WS-CURR-MON '/' WS-CURR-DAY ' '
           WS-CURR-HOUR ':' WS-CURR-MIN ':'
           WS-CURR-SEC  '.' WS-CURR-HSEC
    DISPLAY '************************************'
            '************************************'
    DISPLAY '*'
*
    OPEN OUTPUT LYRICS-FILE
*
    MOVE 99                      TO B2D-BOTTLES-1
    MOVE 99                      TO B2D-BOTTLES-2
    WRITE LYRICS-REC             FROM BEER-2-DIGIT
    ADD +1                       TO WS-LYRICS-WRITTEN
*
    PERFORM 1000-MATCHING-VERSES    THRU 1000-EXIT
        VARYING WS-BOTTLE-NUM FROM 98 BY -1
        UNTIL WS-BOTTLE-NUM < 2
*
    WRITE LYRICS-REC             FROM TAKE-1-MORE
    WRITE LYRICS-REC             FROM BLANK-LINE
    ADD +2                       TO WS-LYRICS-WRITTEN
*
    WRITE LYRICS-REC             FROM BEER-1-MORE
    WRITE LYRICS-REC             FROM TAKE-NO-MORE
    WRITE LYRICS-REC             FROM BLANK-LINE
    ADD +3                       TO WS-LYRICS-WRITTEN
*
    WRITE LYRICS-REC             FROM BEER-NO-MORE
    WRITE LYRICS-REC             FROM BUY-SOME-MORE
    ADD +2                       TO WS-LYRICS-WRITTEN
*
    CLOSE LYRICS-FILE
*
    DISPLAY '************************************'
            '************************************'
    DISPLAY '**** RUN STATISTICS FOR PROGRAM BOTTLE99'
    DISPLAY '************************************'
            '************************************'
    DISPLAY '*'
    MOVE WS-LYRICS-WRITTEN       TO WS-DISPLAY-NUM
    DISPLAY '* LYRICS RECORDS WRITTEN = ' WS-DISPLAY-NUM
    DISPLAY '*'
*
```

```
*
    CLOSE LYRICS-FILE
*
    DISPLAY '*************************************'
            '*************************************'
    DISPLAY '**** RUN STATISTICS FOR PROGRAM BOTTLE99'
    DISPLAY '*************************************'
            '*************************************'
    DISPLAY '*'
    MOVE WS-LYRICS-WRITTEN          TO WS-DISPLAY-NUM
    DISPLAY '* LYRICS RECORDS WRITTEN = ' WS-DISPLAY-NUM
    DISPLAY '*'
*
    DISPLAY '*************************************'
            '*************************************'
    DISPLAY '**** END PROGRAM BOTTLE99'
    ACCEPT WS-CURR-DATE           FROM DATE YYYYMMDD
    ACCEPT WS-CURR-TIME           FROM TIME
    DISPLAY '**** ENDED AT: '
           WS-CURR-YEAR '/' WS-CURR-MON '/' WS-CURR-DAY
           WS-CURR-HOUR ':' WS-CURR-MIN ':'
           WS-CURR-SEC  '.' WS-CURR-HSEC
    DISPLAY '*************************************'
            '*************************************'
*
    GOBACK.
*****************************************************
*    THIS PARAGRAPH WRITES THE FIRST 98 MATCHING VERSES
*****************************************************
1000-MATCHING-VERSES.
*****************************************************
    IF WS-BOTTLE-NUM > 9
       MOVE WS-BOTTLE-NUM          TO T2D-BOTTLES-1
       MOVE WS-BOTTLE-NUM          TO B2D-BOTTLES-1
       MOVE WS-BOTTLE-NUM          TO B2D-BOTTLES-2

       WRITE LYRICS-REC          FROM TAKE-2-DIGIT
       WRITE LYRICS-REC          FROM BLANK-LINE
       WRITE LYRICS-REC          FROM BEER-2-DIGIT
       ADD +3                      TO WS-LYRICS-WRITTEN
    ELSE
       MOVE WS-BOTTLE-NUM          TO T1D-BOTTLES-1
       MOVE WS-BOTTLE-NUM          TO B1D-BOTTLES-1
       MOVE WS-BOTTLE-NUM          TO B1D-BOTTLES-2

       WRITE LYRICS-REC          FROM TAKE-1-DIGIT
       WRITE LYRICS-REC          FROM BLANK-LINE
       WRITE LYRICS-REC          FROM BEER-1-DIGIT
       ADD +3                      TO WS-LYRICS-WRITTEN
    END-IF
    .
1000-EXIT. EXIT.
```

Dayfinder		

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DAYFIND.
AUTHOR. https://ibmmainframes.com/programs.php.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 Y PIC 9(4).
01 M PIC 9(2).
01 D PIC 9(2).
01 A PIC 99 VALUE ZERO.
01 B PIC 9 VALUE ZERO.
01 C PIC 99 VALUE ZERO.
01 F VALUE "120607080910 11".
05 FF PIC 99 OCCURS 7.
01 E PIC 9999 VALUE 0012.
01 T PIC 9999 VALUE ZERO.
01 I PIC 9 VALUE 1.
***A MCMILLAN PRODUCT. PLEASE DON'T MISUSE!

PROCEDURE DIVISION.
0001A.
DISPLAY "ENTER 4 DIGIT YEAR >=0001 & <=9999".
ACCEPT Y.
0002A.
DISPLAY "ENTER MONTH(INTEGER)".
ACCEPT M.
IF M < 1 OR M > 12
    DISPLAY "INVALID MONTH" GO TO 0002A.
0003A.
DISPLAY "ENTER DATE(INTEGER)".
ACCEPT D.
IF D < 1 OR D > 31
    DISPLAY "INVALID DATE" GO TO 0003A.
MOVE D TO C.

0000X.
COMPUTE A = FF(I).
IF E = Y
    GO TO 0000Y.
ADD 1 TO I.
IF I > 7
    COMPUTE I = 1.
DIVIDE E BY 4 GIVING T REMAINDER B.
IF E < Y AND B = 0
    ADD 1 TO I.
IF I > 7
    COMPUTE I = 1.
ADD 1 TO E.
GO TO 0000X.

0000Y.
IF B = 0 AND M > 2
    ADD 1 TO A.

IF M = 1
    ADD A TO D
    DIVIDE D BY 7 GIVING A REMAINDER B
ELSE IF M = 2
    ADD A 3 TO D
    DIVIDE D BY 7 GIVING A REMAINDER B
ELSE IF M = 3
    ADD A 3 TO D
    DIVIDE D BY 7 GIVING A REMAINDER B
ELSE IF M = 4
    SUBTRACT 1 FROM A
    ADD A TO D
    DIVIDE D BY 7 GIVING A REMAINDER B
ELSE IF M = 5
    ADD A 1 TO D
    DIVIDE D BY 7 GIVING A REMAINDER B
ELSE IF M = 6
    ADD A 4 TO D
    DIVIDE D BY 7 GIVING A REMAINDER B
ELSE IF M = 7
    SUBTRACT 1 FROM A
    ADD A TO D
    DIVIDE D BY 7 GIVING A REMAINDER B
ELSE IF M = 8
    ADD A 2 TO D
    DIVIDE D BY 7 GIVING A REMAINDER B
ELSE IF M = 9
    SUBTRACT 2 FROM A
    ADD A TO D
    DIVIDE D BY 7 GIVING A REMAINDER B
ELSE IF M = 10
    ADD A TO D
    DIVIDE D BY 7 GIVING A REMAINDER B
ELSE IF M = 11
    ADD A 3 TO D
    DIVIDE D BY 7 GIVING A REMAINDER B
ELSE IF M = 12
    SUBTRACT 2 FROM A
    ADD A TO D
    DIVIDE D BY 7 GIVING A REMAINDER B
ELSE
    DISPLAY "COBOL FAILED".

DISPLAY Y "/" M "/" C " IS:".
IF B = 0
    DISPLAY "SUNDAY"
ELSE IF B = 1
    DISPLAY "MONDAY"
ELSE IF B = 2
    DISPLAY "TUESDAY"
ELSE IF B = 3
    DISPLAY "WEDNESDAY"
ELSE IF B = 4
    DISPLAY "THURSDAY"
ELSE IF B = 5
    DISPLAY "FRIDAY"
ELSE IF B = 6
    DISPLAY "SATURDAY"
ELSE
    DISPLAY "COBOL RUNTIME ERROR".

STOP RUN.
```

```
(base) elitePro:~ ashishupadhaya$ cd desktop
(base) elitePro:desktop ashishupadhaya$ cd cobol_test
(base) elitePro:cobol_test ashishupadhaya$ cobc -x dayfinder.cbl
(base) elitePro:cobol_test ashishupadhaya$ ./dayfinder
ENTER 4 DIGIT YEAR >=0001 & <=9999
2078
ENTER MONTH(INTEGER)
5
ENTER DATE(INTEGER)
2
2078/05/02 IS:
MONDAY
```

If else implementation

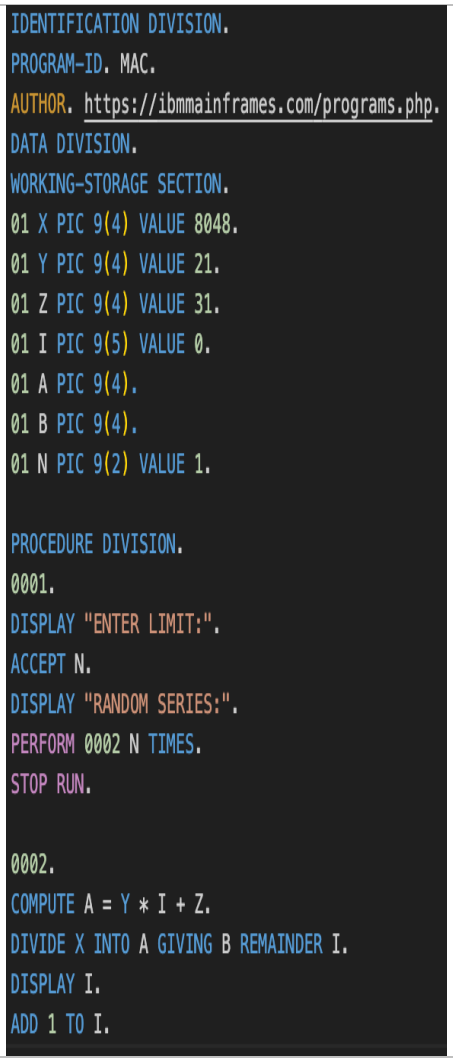
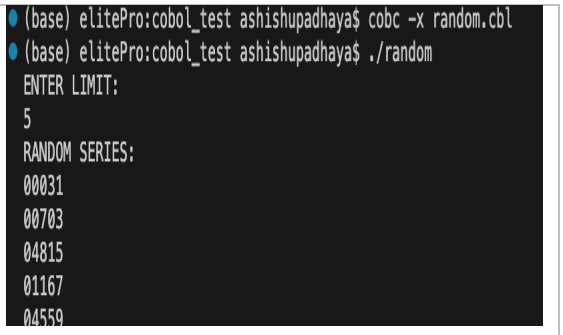
```
*> create variables for testing classes
01 CLASS1 PIC X(9) VALUE 'ABCD '.
*> create statements that can be fed
*> into a COBOL conditional
01 CHECK-VAL PIC 9(3).
88 PASS VALUES ARE 041 THRU 100.
88 FAIL VALUES ARE 000 THRU 040.

PROCEDURE DIVISION.
*> set 25 into num1 and num3
*> set 15 into num2 and num4
MOVE 25 TO NUM1 NUM3.
MOVE 15 TO NUM2 NUM4.

*> comparing two numbers and checking for equali
IF NUM1 > NUM2 THEN
DISPLAY 'IN LOOP 1 - IF BLOCK'
IF NUM3 = NUM4 THEN
    DISPLAY 'IN LOOP 2 - IF BLOCK'
ELSE
    DISPLAY 'IN LOOP 2 - ELSE BLOCK'
END-IF
ELSE
DISPLAY 'IN LOOP 1 - ELSE BLOCK'
END-IF.

*> use a custom pre-defined condition
*> which checks CHECK-VAL
MOVE 65 TO CHECK-VAL.
IF PASS
DISPLAY 'PASSED WITH ' CHECK-VAL ' MARKS.'.
IF FAIL
DISPLAY 'FAILED WITH ' CHECK-VAL ' MARKS.'.

*> a switch statement
EVALUATE TRUE
WHEN NUM1 < 2
    DISPLAY 'NUM1 LESS THAN 2'
WHEN NUM1 < 19
    DISPLAY 'NUM1 LESS THAN 19'
WHEN NUM1 < 1000
    DISPLAY 'NUM1 LESS THAN 1000'
END-EVALUATE.

STOP RUN.
```

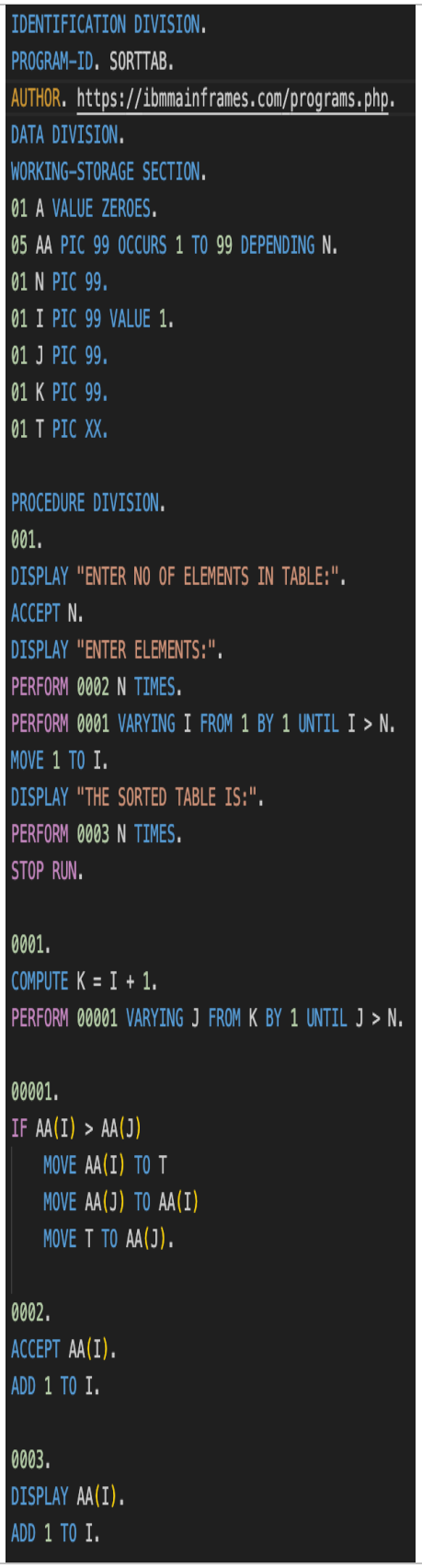
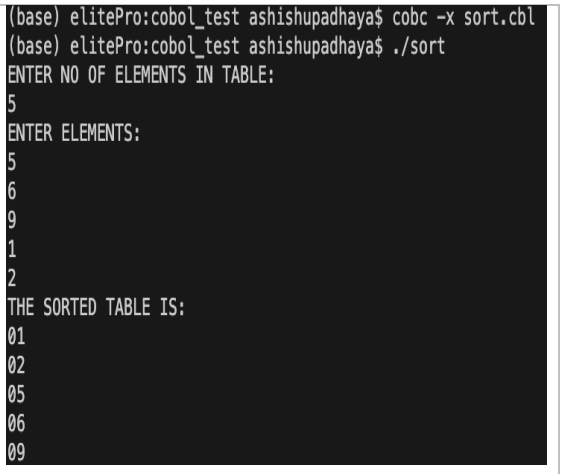
```
(base) elitePro:cobol_test ashishupadhaya$ cobc -x ifelse.cbl
(base) elitePro:cobol_test ashishupadhaya$ ./ifelse
IN LOOP 1 - IF BLOCK
IN LOOP 2 - ELSE BLOCK
PASSED WITH 065 MARKS.
NUM1 LESS THAN 1000
```

Prime number

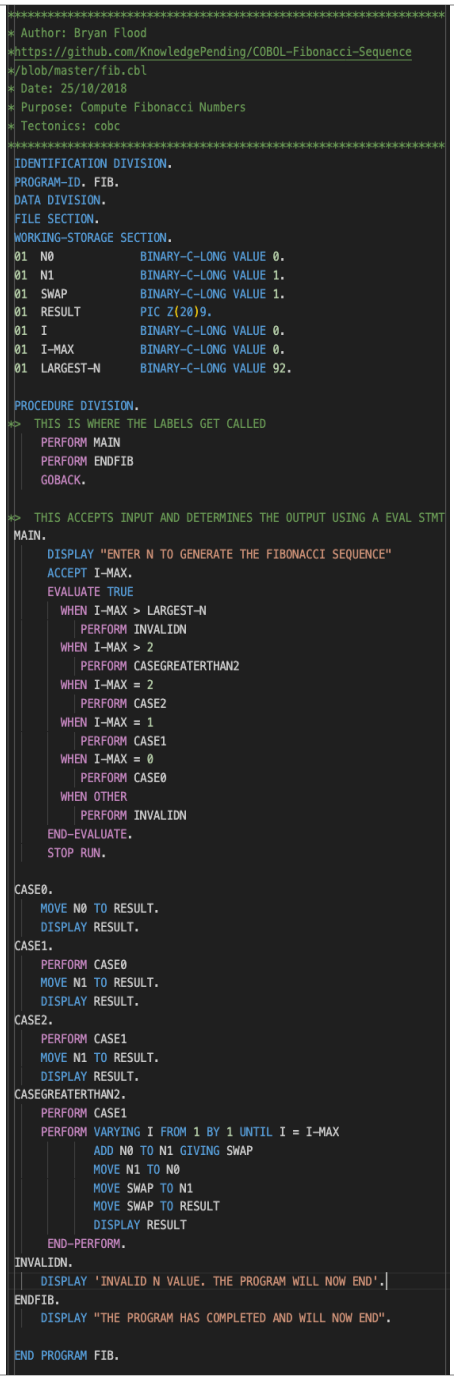
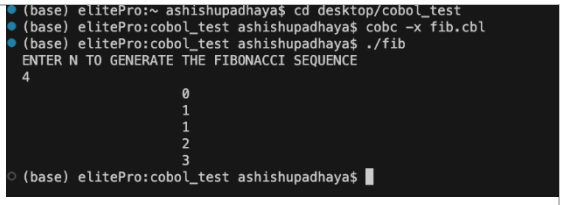
```
IDENTIFICATION DIVISION.
PROGRAM-ID. PRIME.
AUTHOR. https://ibmmainframes.com/programs.php.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 A.
05 AA PIC 99 OCCURS 10 TIMES.
01 I PIC 99 VALUE 1.
01 N PIC 9(10) VALUE 1.
01 D PIC 9(10) VALUE 2.
01 V PIC 9(10) VALUE 1.
01 T1 PIC 99.
01 T2 PIC 99.

PROCEDURE DIVISION.
0001.
DISPLAY "ENTER NO".
ACCEPT N.
DISPLAY "PRIME FACTORS ARE: ".
COMPUTE V = N / 2.
PERFORM 0002.
COMPUTE I = 1.
PERFORM 0003 UNTIL I > 10.
STOP RUN.

0002.
DIVIDE N BY D GIVING T1 REMAINDER T2.
IF T2 = 0
    COMPUTE AA(I) = D
    ADD 1 TO I
    COMPUTE N = N / D
ELSE
    ADD 1 TO D.
IF D < N OR = N
    GO TO 0002.

0003.
DISPLAY AA(I).
ADD 1 TO I.
```

```
(base) elitePro:cobol_test ashishupadhaya$ cobc -x prime.cbl
(base) elitePro:cobol_test ashishupadhaya$ ./prime
ENTER NO
12
PRIME FACTORS ARE:
02
02
03
00
```

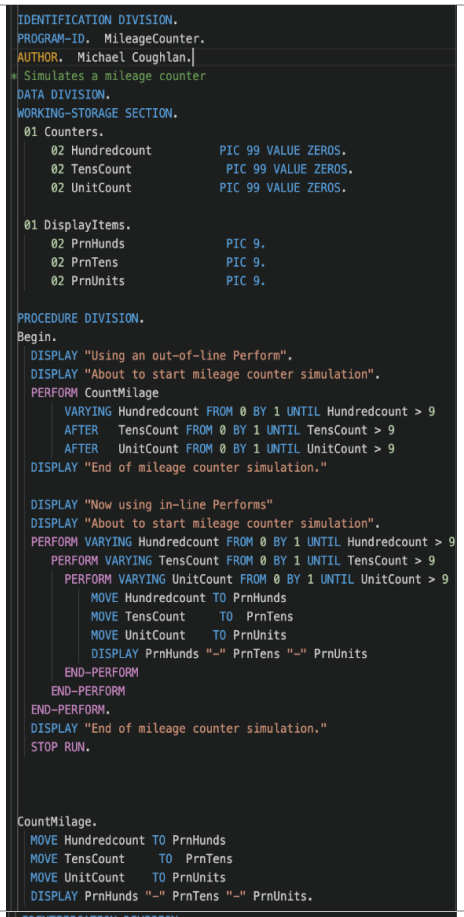
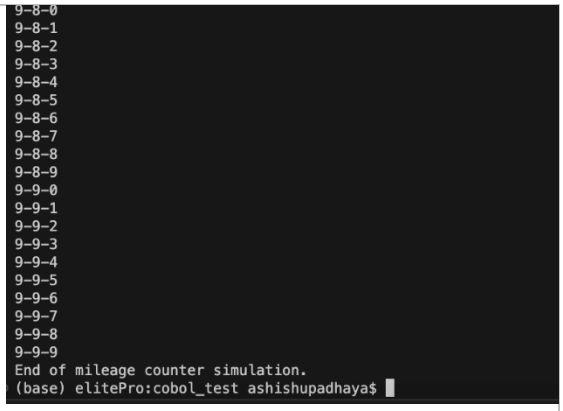
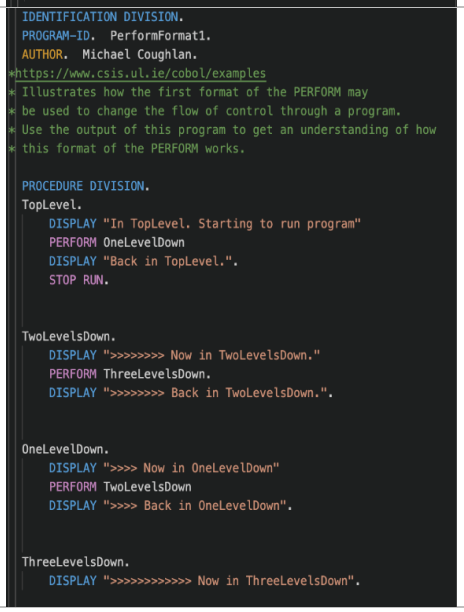
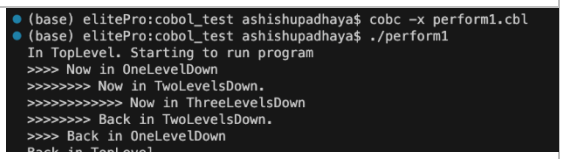
Random number generator		

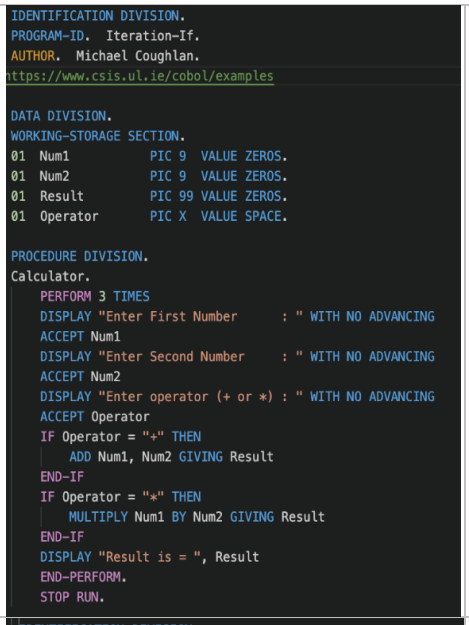
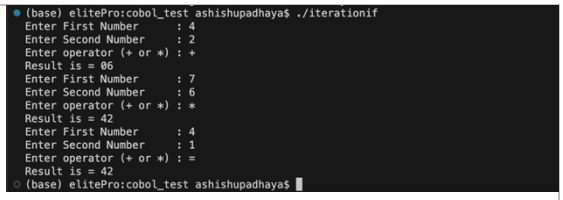
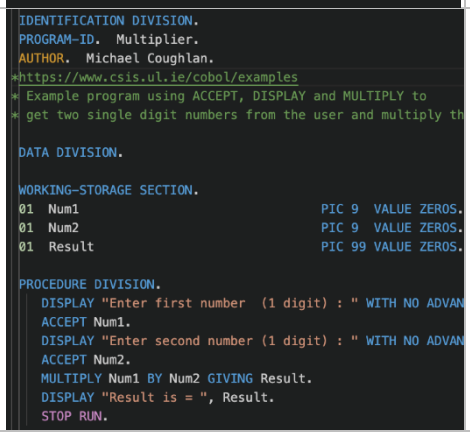
Sort	 IDENTIFICATION DIVISION. PROGRAM-ID. SORTTAB. AUTHOR. https://ibmmainframes.com/programs.php. DATA DIVISION. WORKING-STORAGE SECTION. 01 A VALUE ZEROES. 05 AA PIC 99 OCCURS 1 TO 99 DEPENDING N. 01 N PIC 99. 01 I PIC 99 VALUE 1. 01 J PIC 99. 01 K PIC 99. 01 T PIC XX. PROCEDURE DIVISION. 001. DISPLAY "ENTER NO OF ELEMENTS IN TABLE:". ACCEPT N. DISPLAY "ENTER ELEMENTS:". PERFORM 0002 N TIMES. PERFORM 0001 VARYING I FROM 1 BY 1 UNTIL I > N. MOVE 1 TO I. DISPLAY "THE SORTED TABLE IS:". PERFORM 0003 N TIMES. STOP RUN. 0001. COMPUTE K = I + 1. PERFORM 00001 VARYING J FROM K BY 1 UNTIL J > N. 00001. IF AA(I) > AA(J) MOVE AA(I) TO T MOVE AA(J) TO AA(I) MOVE T TO AA(J). 0002. ACCEPT AA(I). ADD 1 TO I. 0003. DISPLAY AA(I). ADD 1 TO I.	 (base) elitePro:cobol_test ashishupadhaya$ cobc -x sort.cbl (base) elitePro:cobol_test ashishupadhaya$ ./sort ENTER NO OF ELEMENTS IN TABLE: 5 ENTER ELEMENTS: 5 6 9 1 2 THE SORTED TABLE IS: 01 02 05 06 09

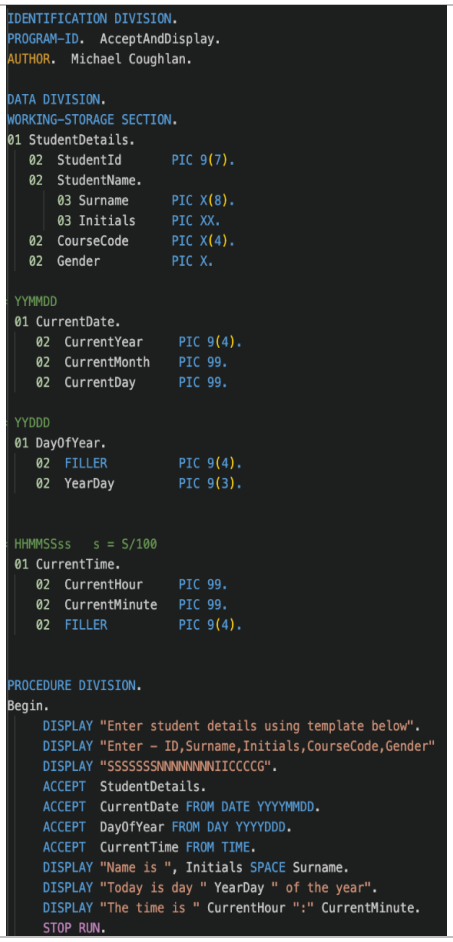
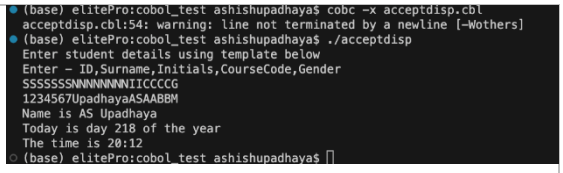
Current date	```
	IDENTIFICATION DIVISION.
	PROGRAM-ID. MAC.
	AUTHOR. https://ibmmainframes.com/programs.php.
	DATA DIVISION.
	WORKING-STORAGE SECTION.
	01 A.
	05 A1 PIC 99.
	05 A2 PIC 99.
	05 A3 PIC 99.
	01 B.
	05 A1 PIC 9(4).
	05 FILLER PIC X VALUE '/'.
	05 A2 PIC 9(2).
	05 FILLER PIC X VALUE '/'.
	05 A3 PIC 9(2).
	
	PROCEDURE DIVISION.
	MAAC.
	ACCEPT A FROM DATE.
	MOVE CORR A TO B.
	IF NOT (A1 OF A < 10)
	    INSPECT A1 OF B REPLACING FIRST "00" BY "19"
	ELSE
	    INSPECT A1 OF B REPLACING FIRST "00" BY "20".
	DISPLAY "CURRENT DATE IS (ISO) :" B.
	STOP RUN.
	``
		``
		(base) elitePro:cobol_test ashishupadhaya$ cobc -x year.cbl
		(base) elitePro:cobol_test ashishupadhaya$ ./year
		CURRENT DATE IS (ISO) :1923/07/30
		``

Fibonacci sequence		

```
(base) elitePro:~ ashishupadhaya$ cd desktop/cobol_test
(base) elitePro:cobol_test ashishupadhaya$ cobc -x fib.cbl
(base) elitePro:cobol_test ashishupadhaya$ ./fib
ENTER N TO GENERATE THE FIBONACCI SEQUENCE
4
                    0
                    1
                    1
                    2
                    3
(base) elitePro:cobol_test ashishupadhaya$
```

Mileage counter		
Perform program 1		

Iteration		
Multiplier		

Accept and display		

Github Repository: https://github.com/ashish27081998/COBOL_test_Programs.git

Appendix 3: Interviews

1. *How has COBOL been utilized in the IT systems of a leading bank in the Netherlands, and which critical applications have been constructed using this language?*

- **Answer 1:** This bank has critical applications like Payments, Transactions, FBS which uses batch processing. COBOL as a programming language helps in bulk data processing on mainframe without user interaction. Large chunks of data can be processed in no time. The processing is optimized and efficient.

- **Answer 2:** COBOL has been extensively used in this banks IT systems, particularly in critical applications. Some specific applications built using COBOL include payments processing, core banking, transaction settlement, risk management, and customer information systems. These applications rely on COBOL's reliability, efficiency, and secure data processing for their essential functions.

2. *What impact did the Y2K bug, also referred to as the Year 2000 bug or Millennium Bug, bring upon computerized systems, and how did it influence the bank's operations? How did the bank tackle this issue?*

- **Answer 1:** When complicated computer programs were being written in early days, a two-digit code was used for the year. The "19" was left out. Instead of a date reading 1987, it read 87. With 2000, it would have become 00. It posed a serious challenge to the banking industry, as failure to address this may disrupt banking operations, such as settlement and interest calculation. Banks had to adjust their systems to process four-digit code for year instead of two. Majority of the Banks fixed the issue on time to be compliant of the Y2K problem.

- **Answer 2:** The Y2K bug posed a significant risk to our computerized systems. It had the potential to cause disruptions and incorrect calculations. To address this issue, we conducted comprehensive remediation programs, updating our systems and applications to handle four-digit year representations correctly manually. Extensive testing was done to ensure a smooth transition, and as a result, we avoided major disruptions during the year 2000 rollover.

3. *Which versions of COBOL has the bank used over the years, and which version is presently in operation?*

- **Answer 1:** COBOL as a programming language was developed in 1959. The first version was released in 1960 which was called COBOL-60. There have been multiple versions after 1960. The current version available is 6.4. In this bank version 6.3 is used for all the COBOL programs.

- **Answer 2:** Throughout its lifecycle till now, the bank has used various versions of COBOL. The specific versions used may vary across different systems and applications. However, over the years, we have upgraded and migrated to newer versions of COBOL to stay current with technology advancements. As of the present, the bank is currently using COBOL version 6.3.

While newer versions may have been released since then, the decision to adopt the latest version depends on factors such as compatibility, reliability, and security. Our IT teams continuously evaluate the benefits and potential risks associated with upgrading to newer versions to ensure the stability and efficiency of our systems.

4. *What were the primary reasons for the decision to upgrade to a newer version of COBOL, and why was the current version selected over others? Does the latest version offer enhancements, superior libraries or other features absent in the prior iteration? Can you provide examples?*

- **Answer 1:** New COBOL version helps to maximize hardware utilization, reduce CPU usage, and improve performance of critical applications. Advanced data conversion methods are introduced which enhances data processing.

- **Answer 2:** upgrading to the current version of COBOL brings numerous advantages, making it a valuable decision for the bank. It not only enhances the performance and security of critical applications but also ensures that the bank remains competitive and up to date with the latest developments in the industry.

5. *What are the main reasons the bank persists in employing systems coded in COBOL? Is COBOL deemed irreplaceable in the bank's operations?*

- **Answer 1:** COBOL is the best when it comes to batch processing. It is robust, and easy to test, debug and analyse. Millions and millions of data can be processed in no time. Optimized programs result in cost savings. COBOL can be replaced with languages like JAVA, but they cannot handle such huge data processing. Batch processing on mainframe is more secured.

- **Answer 2:** The bank continues to use COBOL for its stability, cost-effectiveness, and expertise. While not considered irreplaceable, the complexity of legacy systems and cost considerations make a complete transition to newer technologies challenging. A balanced approach is taken to leverage COBOL's strengths while exploring modernization opportunities.

6. *How has the bank addressed the challenge of upkeeping and refreshing aging COBOL systems over the years?*

- **Answer 1:** Maintaining and updating COBOL systems are highly challenging. For example, a version upgrade involves activities like recompiling programs, testing etc. The benefit of using COBOL is more than the efforts involved in maintaining it.

- **Answer 2:** The bank has addressed legacy COBOL systems through continuous maintenance, selective modernization, skill development for IT teams, integration with new technologies, risk management, collaboration with vendors, business prioritization, and long-term planning. This strategic approach allows the bank to maintain stability while gradually adapting to newer technologies.

7. *What steps is the bank implementing to guarantee the security and dependability of its COBOL-integrated systems?*

- **Answer 1:** Safe coding practices which includes data encryption, multi-factor authentication, proper input validation is introduced which keeps the systems up and running. They are more reliable.

- **Answer 2:** Through regular security updates, data encryption, secure coding practices, security audits, disaster recovery plans, compliance with regulations, employee training, and collaboration with security experts. These measures protect customer data and maintain the stability of critical services.

8. *How has the integration of emerging technologies, such as cloud computing and artificial intelligence, affected the bank's deployment of COBOL within its IT infrastructure?*

- **Answer 1:** Not to a larger extent. The new technologies offer digital transformation, but COBOL works too well.

- **Answer 2:** Not a lot.

9. *In what manner has COBOL supported the bank in realizing its business goals, and are there any perceived disadvantages linked with its usage?*

- **Answer 1:** As a bank, processing of payments and transactions at a faster rate is importance which will make customers happy. COBOL facilitates batch processing. The changing infrastructure, requirements may raise challenges for the organization. It is becoming difficult to get COBOL experienced resources.

- **Answer 2:** COBOL has helped the bank to achieve its business objectives through its stability, efficient data processing, and cost-effectiveness. However, potential drawbacks include talent shortage, integration challenges with modern technologies, limited advancements, and concerns about long-term viability.

10. *How critical is COBOL in the bank's IT framework, and what is the institution's vision regarding its ongoing use? Are there discussions about transitioning to a distinct coding language?*

- **Answer 1:** COBOL will remain in place until the Bank does payments and transactions. It will be hard to replace it with more digitized solutions using cloud computing, AI. Replacing or rewriting what is done in COBOL will pose a challenge and risk to the way Bank operates. It would require lot of resources, time, and money. Weighing at the advantages of processing vast amounts of data, COBOL will stay for many more years to come.

- **Answer 2:** COBOL will continue to be the backbone of our operations as long as payments and transactions are core functions of the bank. While there is a push towards more digitized solutions using cloud computing and AI, the idea of replacing or rewriting what we have in COBOL is daunting. It would pose significant challenges and risks to the way our bank operates. Replacing COBOL-based systems would demand substantial resources, time, and financial investments.
  It would be a complex process, and any disruption to critical banking functions during the transition could have severe consequences. Given the advantages COBOL offers in efficiently processing vast amounts of data, it remains a reliable choice for handling critical operations.

11. *How do you envision the role of this language in the forthcoming IT strategy of the bank?*

- **Answer 1:** COBOL will still be an integral part of the Bank as it facilitates batch processing in a more controlled, secured, efficient, cost-effective way.

12. *Are there any other noteworthy details concerning the bank's adoption of COBOL that merit attention?*

- **Answer 1:** I have worked on batch systems which run on COBOL processing Payments, transactions, customer data. These systems are critical and play a key role in the functioning of the Bank.

13. *How many COBOL developers are there at the bank, and in what age categories?*

- **Answer 1:** Among our team, 20 people know the mainframe. In the accounting department, FBS is the sole mainframe system. The reporting Basel department also uses the mainframe, but only a few people support it. In comparison, FBS has about 4 or 5. Systems like core banking and transaction also use mainframes. On average, this specific bank has between 150-200 employees working directly with mainframe systems.

14. *How many COBOL programs are there?*

- **Answer 1:** the bank currently operates around 27,000 active COBOL programs, with a total of 57,000 when including those that have been deleted. 30,000 programs were filtered out when removed. Although the number of COBOL programs is decreasing, the bank is in the process of replacing some of these systems with Java and other languages. Systems such as OneSumX and Beam are using Informatica ETL, Hadoop, and Databricks to replace FBS. The bank's goal is to migrate to Azure, resulting in the decline of COBOL lines every year.

15. *Does the bank utilize modern development processes or practices within the COBOL teams, such as automated testing, version control with feature branches, tools for code quality measurements, scrum or kanban, automated deployment pipelines? Which practices specifically? Has the bank considered using Micro Focus, GnuCOBOL, or other alternatives to be less reliant on IBM COBOL?*

- **Answer 1:** Automated tests are conducted at the bank. FBS utilizes Jenkins for testing, and SonarQube for quality control. This setup forms the continuous integration pipeline. When promoting a component from unit testing to a higher environment, the pipeline gets triggered, executing test cases. The bank also uses Topaz for COBOL compilation. In addition, ISPW version 6.3 is in use, and Topaz serves as a tool for importing and compiling the code. These systems are integrated into the bank's infrastructure. While the bank does not solely depend on the COBOL compiler, it integrates other systems into its operations.

16. *Does the bank have anything specific for DB2?*

- **Answer 1:** The bank does not have any specific tools solely for DB2, but SAS can be integrated with it. Using SAS Enterprise Guide, one can connect with DB2 tables and query them. Bridge Cobra is specific to this bank and simplifies the management of complex table relationships.

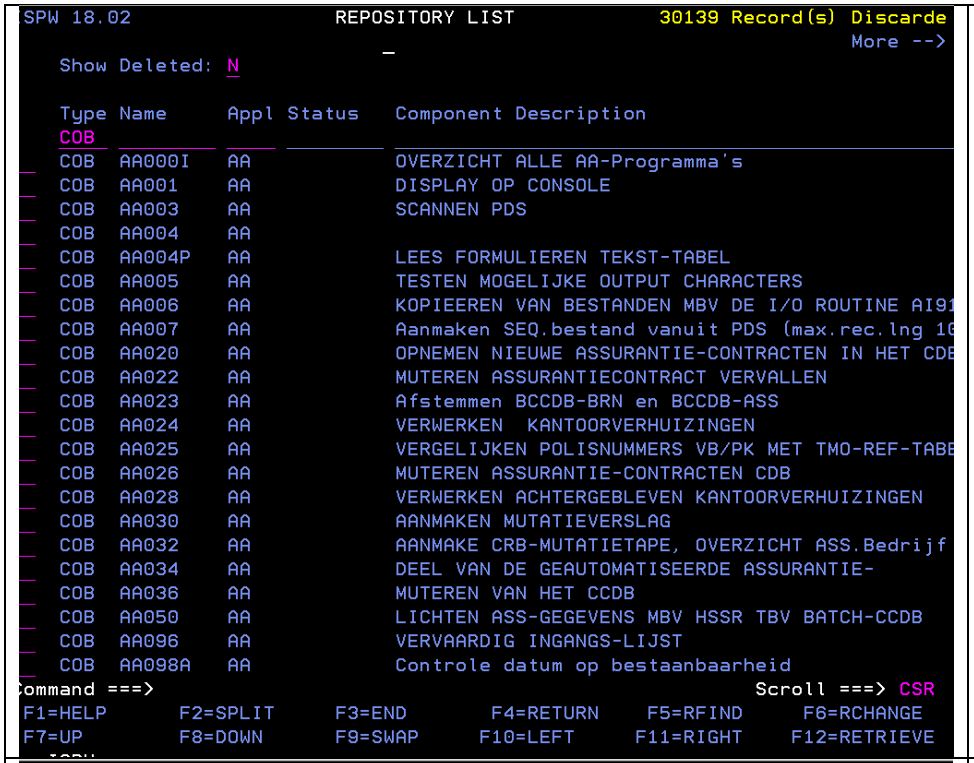
17. *Is there a risk related to knowledge transfer?*

- **Answer 1:** Within our team, the balance of knowledge is maintained. However, that is not the case across all teams. In the past, there were challenges when experienced employees were nearing retirement, leaving only younger team members. Achieving the same level of expertise as a 30-year veteran in just a few years is tough. Additionally, some employees were hesitant to share their expertise, fearing job security, especially those from IBM. The current trend is a younger workforce with a more open mindset. The bank is looking at a transition to Azure in the next 5-7 years, but it is unlikely that mainframes will be completely phased out due to their strengths, especially in handling large batches of data safely.

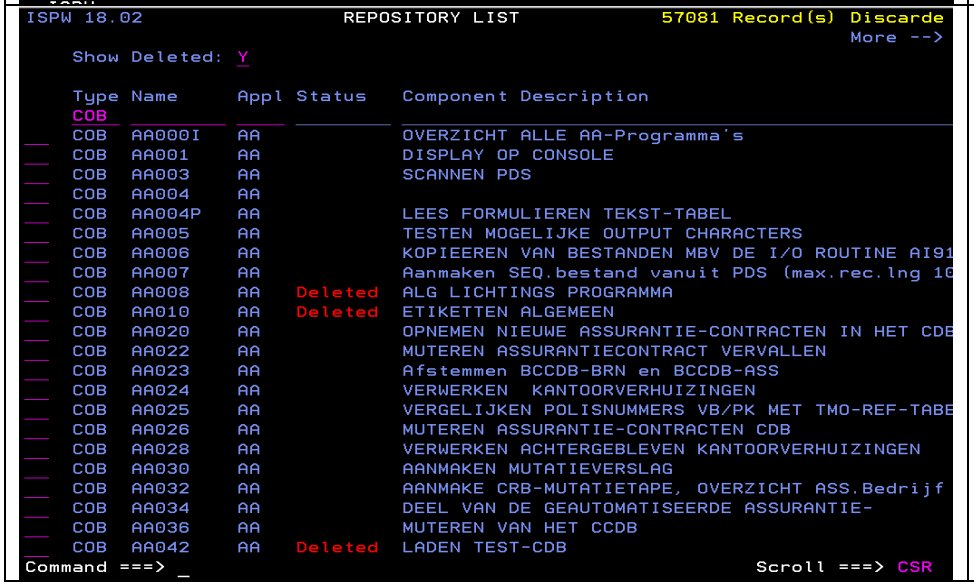
18. *Does this mean mainframes are unhackable?*

- **Answer 1:** No system is truly unhackable. While I have production access and could exploit it, everything is logged and traceable. Authorization is crucial, and merely having access does not provide full control. Mainframes might not be fool proof, but they offer better security than alternatives.

Appendix 4: COBOL repository



In the repository, there are currently 1,000 listed COBOL programs. An additional 30,139 records have been discarded, bringing the total number of available COBOL programs historically to 31,139.



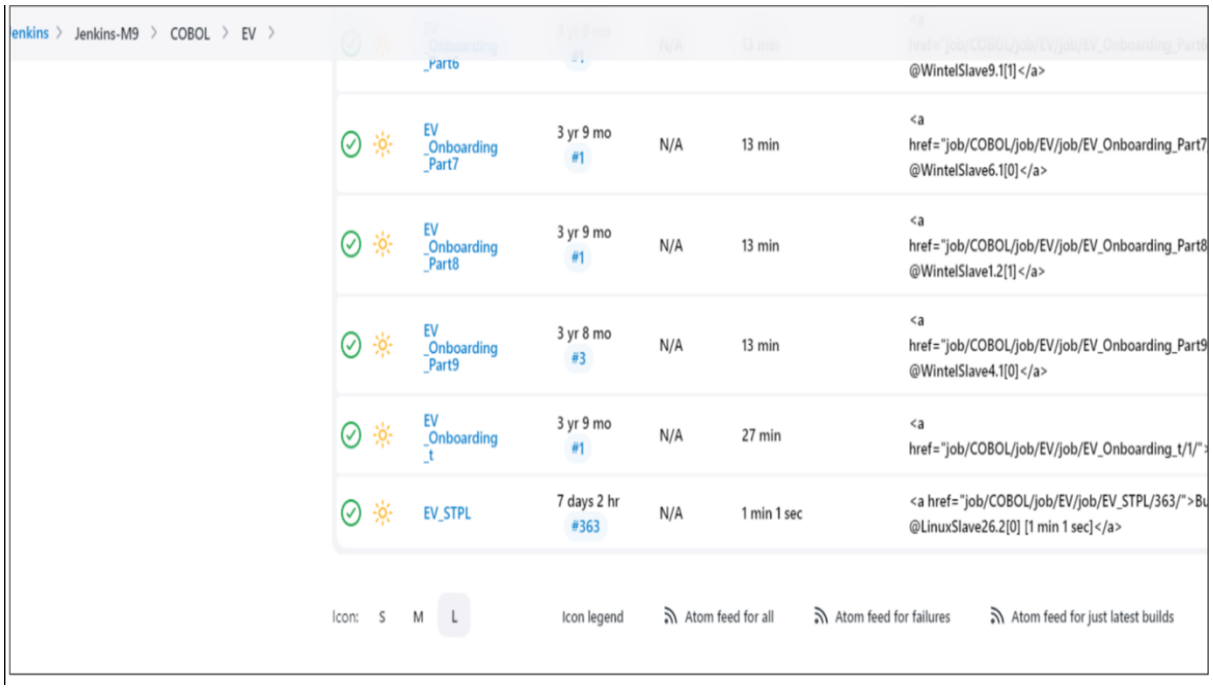
In the repository including deleted ones, 1,000 COBOL programs are actively listed. Additionally, 57,081 records have been discarded. This means that historically, there have been a total of 58,081 COBOL programs.

2020/07/13 Monday ========== News topic(s): Cobol compiler 6.3 activation. ========== * Cobol compiler 6.3 activation. ========== On Monday 13 July 2020 ISPW will be upgraded to make use of Cobol compiler version 6.3 Programs compiled with 6.3 will optimize our run-time environment, saving operation costs (MSU's) and decrease elapse times of batch-runs. Compiler version 6.3 will be set as the default in the generate panel for Cobol sources. Only for Telon sources the default is 4.2 After this upgrade, you can choose between 6.3 and 4.2 but version 6.3 is the preferred Cobol compiler from then on. So only in case of severe issues that seem to be related to the Compiler, developers still can use Cobol 4.2 In Topaz you can't choose, only Compiler 6.3 can be used there ! Just as the Data parm in your Cobol Compile, the use of the Compiler 4.2 Just as the Data parm in your Cobol Compile, the use of the Compiler 4.2 will be actively monitored. When you use DATA(24) or Compiler 4.2, an email will be send to the MAMO Team, when promoting the source to level cons. For additional information see the Blog on Connections from the MAMO Team.	Update that this bank will be using the compiler 6.3 after the upgrade.

Appendix 5 : Jenkins and SonarQube

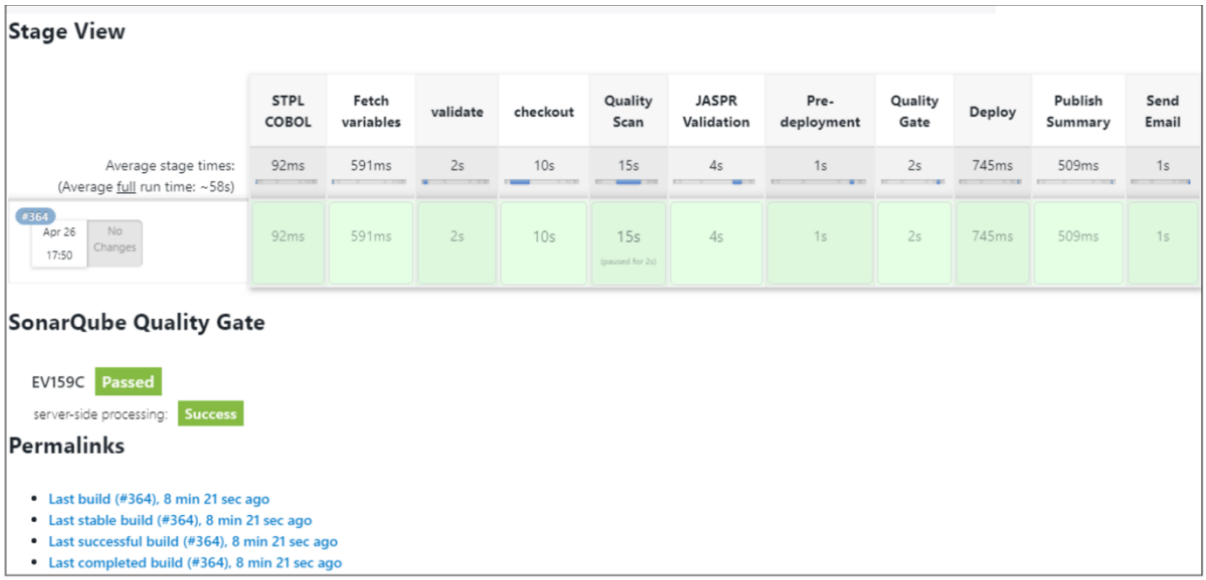
Jenkins

Jenkins evaluates the COBOL components to ensure it complies with all the standards set by the bank for a COBOL component.



SonarQube

SonarQube evaluates the COBOL code to ensure its code quality and if it meets all the requirement for a code prescribed by the bank. Once all checks are validated an email will be sent to the user that the program can be successfully deployed to a higher environment.



Appendix 6: Academic papers related to migration of COBOL.

1. COBOL to Java

Paper Title	Key Findings and Relevance to Thesis
De Marco, A., Iancu, V., & Asinofsky, I. (2018). COBOL to Java and Newspapers Still Get Delivered. ICSME. https://doi.org/10.1109/icsme.2018.00055	Discusses practical COBOL-to-Java migration and its real-world implications.
Brüne, P. (2018). A Hybrid Approach to Re-Host and Mix Transactional COBOL and Java Code in Java EE Web Applications using Open Source Software. International Conference on Web Information Systems and Technologies. https://doi.org/10.5220/0006943402390246	Explores a hybrid approach for migrating COBOL to Java and the role of open-source software.
Knoche, H., & Hasselbring, W. (2018). Using microservices for legacy software modernization. IEEE Software, 35(3), 44–49. https://doi.org/10.1109/ms.2018.2141035	Investigates the role of microservices in modernizing legacy software, which is relevant to COBOL migration.
Markus, S., & Streit, J. (2021). Efficient Platform Migration of a Mainframe Legacy System Using Custom Transpilation. IEEE International Conference on Software Maintenance and Evolution (ICSME),. https://doi.org/10.26226/morressier.613b5419842293c031b5b64a	Discusses efficient platform migration from mainframe systems, applicable to COBOL migrations.
Brüne, P. (2019). An open source approach for modernizing Message-Processing and transactional COBOL applications by integration in Java EE application servers. In Lecture notes in business information processing. https://doi.org/10.1007/978-3-030-35330-8_12	Provides insights into open-source methods for modernizing COBOL applications in Java EE servers.
Strobl, S., Zoffi, C., Haselmann, C., Bernhart, M., & Grechenig, T. (2020). Automated Code Transformations: Dealing with the Aftermath. 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER). https://doi.org/10.1109/saner48275.2020.9054813	Addresses automated code transformations, which may apply to post-migration code adjustments.
Flores-Ruiz, S., Pérez-Castillo, R., Domann, C., & Puica, S. (2018). Mainframe Migration Based on Screen Scraping. IEEE International Conference on Software Maintenance and Evolution (ICSME). https://doi.org/10.1109/icsme.2018.00077	Discusses mainframe migration using screen scraping techniques, which could be relevant to COBOL migration.
Mateos, C., Zunino, A., Flores, A., & Misra, S. (2019). COBOL Systems Migration to SOA: Assessing antipatterns and complexity. Information Technology and Control, 48(1). https://doi.org/10.5755/j01.itc.48.1.21566	Focuses on assessing migration challenges and complexity, which is relevant to COBOL-to-Java migration.
Strobl, S., Bernhart, M., & Grechenig, T. (2020). Towards a Topology for Legacy System Migration. Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops. https://doi.org/10.1145/3387940.3391476	Proposes a topology for migrating legacy systems, which may offer insights into structured migration approaches.
Sneed, H. M., & Verhoef, C. (2020). Cost-driven software migration: an experience report. Journal of Software: Evolution and Process, 32(7). https://doi.org/10.1002/smr.2236	Shares experiences related to cost-driven software migration, which can provide practical insights for COBOL migration.
Sneed, H. M., & Verhoef, C. (2019). Re-implementing a legacy system. Journal of Systems and Software, 155, 162–184. https://doi.org/10.1016/j.jss.2019.05.012	Discusses the re-implementation of legacy systems, which could be relevant to COBOL migration.

2. COBOL to Python

Paper Title	Key Findings and Relevance to Thesis
O'Hara, S. (2018). Improving programming language transformation. In Proceedings of the International Conference on Software Engineering Research and Practice (SERP). The Steering Committee of The World Congress in Computer Science, Computer (pp. 129-135).	Discusses language transformation and its relevance to COBOL-to-Python migration.
Lachaux, M., Rozière, B., Chanussot, L., & Lample, G. (z.d.). Unsupervised Translation of Programming Languages. arXiv (Cornell University). http://export.arxiv.org/pdf/2006.03511	Addresses unsupervised code translation, which is important for migration to Python.
Roziere, B., Zhang, J. M., Charton, F., Harman, M., Synnaeve, G., & Lample, G. (2021). Leveraging automated unit tests for unsupervised code translation. arXiv preprint arXiv:2110.06773.	Investigates the use of automated unit tests in code translation, which can streamline migration.
Malyala, A., Zhou, K., Ray, B., & Chakraborty, S. (2023). On ML-Based Program Translation: Perils and Promises. arXiv preprint arXiv:2302.10812.	Discusses machine learning-based program translation and its potential challenges and benefits for migration.
Kulshrestha, A., & Lele, V. (2022). Cobol2Vec: Learning Representations of Cobol code. arXiv preprint arXiv:2201.09448.	Focuses on learning representations of COBOL code, which is relevant for understanding code translation.
Surianarayanan, C., Chelliah, P. R., Surianarayanan, C., & Chelliah, P. R. (2019). Cloud Migration. Essentials of Cloud Computing: A Holistic Perspective, 221-240.	Discusses cloud migration, which can be part of the modernization process in COBOL-to-Python migration.
LEONIDA, J. (2021). LANGUAGE PORTING IN COMPUTING.	Discusses language porting, which is an essential aspect of code migration.
Weisz, J. D., Muller, M., Houde, S., Richards, J., Ross, S. I., Martinez, F., ... & Talamadupula, K. (2021, April). Perfection not required? Human-AI partnerships in code translation. In 26th International Conference on Intelligent User Interfaces (pp. 402-412).	Addresses the role of human-AI partnerships in code translation, which may be relevant to code migration projects.

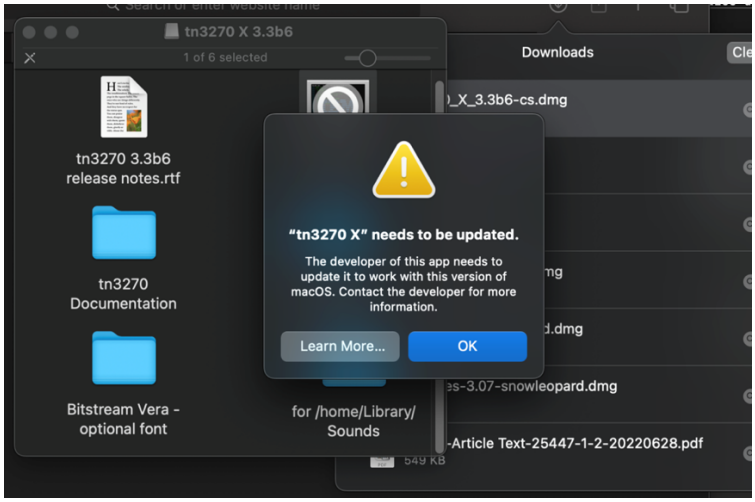
3. COBOL to C#

Paper Title	Key Findings and Relevance to Thesis
Yousif, H. S. (2018). CFlat: An Intermediate Representation Language for the Purpose of Software Migration to Java and C# (master's thesis, The University of Bergen).	Explores an intermediate representation language for migrating to Java and C#.
Mateos, C., Zunino, A., Flores, A., & Misra, S. (2019). Cobol systems migration to SOA: assessing antipatterns and complexity. Information Technology and Control, 48(1), 71-89.	Discusses the assessment of antipatterns and complexity in migration, relevant to COBOL-to-C# migration. Addresses the assessment of antipatterns and complexity in migrating from COBOL to SOA, which may have implications for C# migration.
Włodarski, L., Pereira, B., Povazan, I., Fabry, J., & Zaytsev, V. (2019, February). Qualify first! a large scale modernisation report. In 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 569-573). IEEE.	Discusses the importance of qualification and assessment in large-scale modernization projects, which is pertinent to COBOL-to-C# migration.
Ali, M. S., Manjunath, N., & Chimalakonda, S. (2023). X-COBOL: A Dataset of COBOL Repositories. arXiv preprint arXiv:2306.04892.	Discusses the availability of COBOL repositories, which can be valuable in understanding and accessing COBOL code for migration.
Ciborowska, A., Chakarov, A., & Pandita, R. (2021, September). Contemporary COBOL: developers' perspectives on defects and defect location. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (pp. 227-238). IEEE.	Investigates developers' perspectives on defects in contemporary COBOL, which can be relevant to quality assurance in migration.
Strobl, S., Zoffi, C., Haselmann, C., Bernhart, M., & Grechenig, T. (2020, February). Automated Code Transformations: Dealing with the Aftermath. In 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 627-631). IEEE.	Addresses automated code transformations, including handling the aftermath of code changes, which is relevant to post-migration adjustments.

4. COBOL to Scala

Paper Title	Key Findings and Relevance to Thesis
Espada, G. J. N. M. (2020). Automatic conversion of ADA source code to scala (Doctoral dissertation).	Discusses automatic code conversion, which may apply to COBOL-to-Scala migration.
Mateus, B. G., Martinez, M., & Kolski, C. (2023). Learning migration models for supporting incremental language migrations of software applications. Information and Software Technology, 153, 107082.	Addresses learning migration models for incremental language migrations, which is pertinent to COBOL-to-Scala migration.
Roziere, B., Zhang, J. M., Charton, F., Harman, M., Synnaeve, G., & Lample, G. (2021). Leveraging automated unit tests for unsupervised code translation. arXiv preprint arXiv:2110.06773.	Investigates the use of automated unit tests in unsupervised code translation, which can streamline migration to Scala.
Abid, C., Alizadeh, V., Kessentini, M., Ferreira, T. D. N., & Dig, D. (2020). 30 years of software refactoring research: A systematic literature review. *arXiv preprint arXiv:2007.02194*.	Discusses 30 years of software refactoring research, providing insights into the importance of refactoring during migration.
Zaytsev, V. (2020, November). Software language engineers' worst nightmare. In *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering* (pp. 72-85).	Discusses challenges faced by software language engineers, which may offer insights into potential pitfalls during language migration.
Yan, W., Tian, Y., Li, Y., Chen, Q., & Wang, W. (2023). CodeTransOcean: A Comprehensive Multilingual Benchmark for Code Translation. arXiv preprint arXiv:2310.04951.	Presents a comprehensive benchmark for code translation, which can be valuable for evaluating the effectiveness of translation in COBOL-to-Scala migration.
Petrulio, F., Sawant, A. A., & Bacchelli, A. (2021). The indolent lambdification of Java: Understanding the support for lambda expressions in the Java ecosystem. Empirical Software Engineering, 26, 1-36.	Explores the support for lambda expressions in the Java ecosystem, which is relevant when migrating to Java as an intermediate step before Scala.
Royal, P. (2022). Building with Modern Spring, Java, and PostgreSQL. In Building Modern Business Applications: Reactive Cloud Architecture for Java, Spring, and PostgreSQL (pp. 147-162). Berkeley, CA: Apress.	Discusses building applications with modern Spring, Java, and PostgreSQL, which can be pertinent to a migration path involving these technologies.
Smith, T. C., & Jones, L. (2021). First Course Programming Languages within US Business College MIS Curricula. Journal of Information Systems Education, 32(4), 283-293.	Investigates the use of programming languages in business college MIS curricula, which may provide insights into the educational aspects of language migration.

Appendix 7: Installation failure TN3270 terminal emulator



Failure of TN3270 terminal emulator due to no update for the most recent version of macOS. TN3270.

Appendix 8: COBOL jobs in the Netherlands

Job title	Company	Salary per month in euros
Cobol Ontwikkelaar (cics DB2)	De Belastingdienst	3608-5503
Backend test automation engineer	De Belastingdienst	3045-4848
COBOL Developer	Personeel Specialisten	3000-4500
COBOL applicatie ontwikkelaar	SVB	5737
Cobol Developer	Ordina	N/A
Test specialist Cobol Mainframe	SVB	5073
Cobol Developer	Experis	3500-5500
Cobol Developer	ABN AMRO	4400-5500
Junior IT Engineer Cross Border	ABN AMRO	3741-5345
IT Engineer Cross Border	ABN AMRO	4213-6019
Applicatie consultant	Capgemini	N/A
Junior Mainframe developer	Experis	3000-3500
DBA DB2 mainframe	De Belastingdienst	3608-6227
DevOps engineer Mainframe	De Belastingdienst	3608-5503
COBOL ontwikkelaar	Red Carpet IT Services	N/A
Mainframe Specialist	Teelor	N/A
SAS Developer	Gazelle Global Consulting	N/A
Transportplanner	Tata Steel Netherlands	5442
Senior COBOL ontwerper/ontwikkelaar	Inter-Sprint Banden	N/A
COBOL Developer	Personeel Specialisten	3000-4500
Functioneel Ontwerper bij Mainmen	Sterksen	N/A
Mainframe project manager	Axiom Software Solutions	N/A
Senior tester (cobol/mainframe omgeving)	Seven starts	N/A

Cobol related jobs retrieved from Indeed.com as per November 20. Please be aware that some positions may no longer be accessible, as they can be removed once they are occupied or no longer open for applications.

Appendix 9: Educational resources

YouTube	COBOL Tutorial: Learn COBOL in One Video https://www.youtube.com/watch?v=TBs7HXI76yU&t=40s channel: Derek Banas Date: 2020
	COBOL Programming Tutorial: From Basics to Advanced |Best COBOL Course | Learn COBOL Programming. https://www.youtube.com/watch?v=cnz9y9k2jvs&t=2814s Channel: Topic Trick Date published: 2023
	COBOL Course - Programming with VSCode https://www.youtube.com/watch?v=RdMAEdGvtLA Channel: FreeCodeCamp.org Date published: 2020
Udemy	Mainframe: The Complete COBOL Course From Beginner To Expert https://www.udemy.com/share/101ZCS3@nAcPFkFR5AO1eOiQt_W5dVzP0cH42vWw99qwN0J2yMBytZQkJdoYK4KpIsMNH3yG/ Author: Sandeep Kumar Last update: 05/2022
	COBOL Complete Reference Course! https://www.udemy.com/share/108BJm3@WSz5dXExIqk8oku2NSjb9yWsXoLilyn94a3u8izdW9aLqJIii-jltUQXAymO5fvK/ Author: Toptrick Education Last update:04/2023
	Mainframe COBOL Developer Training By Anil Polsani https://www.udemy.com/share/107FWK3@QPujpJuuf4eaLD9GvKHXr8k0nqAyoYa_Z1mBabTRsx5CG5tfz-ivOeRfySnb0DgG/ Author: Anil Polsani Last update: 11/2022
	The complete Mainframe Professional Course -4 Clurses in 1 https://www.udemy.com/share/101Ym83@VN5UXKMzSpmJnp9QCFLDiI4o32P95gg5WjEEWNxDdhX5N2AfQYOTCiif-yi7Y46Y/ Author Rathi Last update:10/2023
GitHub	https://github.com/neopragma/COBOL-samples/tree/main/src/main/COBOL https://github.com/ashish27081998/COBOL_test_Programs.git https://github.com/moderneinc/cobol-samples.git
Overall	https://ibmmainframes.com/programs.php https://www.infogoal.com/cbd/cobol_example_code_programs.htm