Polyverse Boost Source Analysis Details: ./factorial.cbl

Date Generated: Saturday, October 21, 2023 at 12:27:56 PM PDT

Boost Architectural Quick Summary Security Report

Last Updated: Saturday, October 21, 2023 at 12:27:12 PM PDT

Executive Report:

1. **Architectural Impact**: The analysis of this file has not revealed any severe issues.
2. **Risk Analysis**: The analysis of this file has not revealed any severe issues.
3. **Potential Customer Impact**: Based on the analysis, there are no severe issues that could potentially impact customers.
4. **Performance Issues**: Our analysis did not identify any explicit performance issues in the file.
5. **Risk Assessment**: Based on the current analysis of this file, no severe issues have been found. However, this doesn't guarantee that the file is risk-free.

Highlights:

- No severe issues were identified in the current analysis of this file.

Boost Architectural Quick Summary Performance Report

Last Updated: Saturday, October 21, 2023 at 12:27:46 PM PDT

Executive Report:

1. **Architectural Impact**: The analysis of this file has not revealed any severe issues.
2. **Risk Analysis**: The analysis of this file has not revealed any severe issues.
3. **Potential Customer Impact**: Based on the analysis, there are no severe issues that could potentially impact customers.
4. **Performance Issues**: Our analysis did not identify any explicit performance issues in the file.

5. **Risk Assessment**: Based on the current analysis of this file, no severe issues have been found. However, this doesn't guarantee that the file is risk-free.

Highlights:

- No severe issues were identified in the current analysis of this file.

Boost Architectural Quick Summary Compliance Report

Last Updated: Saturday, October 21, 2023 at 12:28:00 PM PDT

Executive Report:

1. **Architectural Impact**: The analysis of this file has not revealed any severe issues.
2. **Risk Analysis**: The analysis of this file has not revealed any severe issues.
3. **Potential Customer Impact**: Based on the analysis, there are no severe issues that could potentially impact customers.
4. **Performance Issues**: Our analysis did not identify any explicit performance issues in the file.
5. **Risk Assessment**: Based on the current analysis of this file, no severe issues have been found. However, this doesn't guarantee that the file is risk-free.

Highlights:

- No severe issues were identified in the current analysis of this file.

Detailed Analysis

./factorial.cbl line 0:

Programming Language: plaintext

```
 IDENTIFICATION DIVISION.
PROGRAM-ID. FACTORIAL.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 CMD-ARGS                 PIC X(38).
  01 DECINUM                  PIC S9999v99.
  01 NUM                      PIC S9(7).
  01 FACTORIAL                PIC 9(15) VALUE 1.
  01 LEFT-JUST-NUMBER         PIC X(16).
  01 WS-TALLY1                PIC 99 VALUE 0.
```

```
           01 CNT                      PIC 9(7) VALUE 1.

         PROCEDURE DIVISION.
            ACCEPT CMD-ARGS FROM COMMAND-LINE.

            IF CMD-ARGS IS ALPHABETIC THEN
               PERFORM ERROR-PARA.

       * Convert CMDARGS to it's numeric value
            COMPUTE DECINUM = FUNCTION NUMVAL(CMD-ARGS).

            IF DECINUM < 0 THEN
               PERFORM ERROR-PARA.

       * Move the Decimal number to Non decimal number
            MOVE DECINUM TO NUM

       * If both are equal, then it was an integer
            IF NUM IS EQUAL TO DECINUM THEN
               IF NUM IS EQUAL TO 0 OR NUM IS EQUAL TO 1 THEN
                  DISPLAY 1
                  STOP RUN
               ELSE
                  PERFORM CALC-FACT UNTIL CNT > NUM

       * Process to left justify the number
                  INSPECT FACTORIAL TALLYING WS-TALLY1 FOR LEADING ZEROS
                  Move FACTORIAL (WS-TALLY1 + 1 :) TO LEFT-JUST-NUMBER
       * Display the left justified result
                  DISPLAY LEFT-JUST-NUMBER
                  STOP RUN
            ELSE
               PERFORM ERROR-PARA.


           CALC-FACT.
             COMPUTE FACTORIAL = FACTORIAL * CNT
             COMPUTE CNT = CNT + 1.

           ERROR-PARA.
            DISPLAY "Usage: please input a non-negative integer".
            STOP RUN.
```

Boost Code Explanation

Last Updated: Saturday, October 21, 2023 at 12:26:43 PM PDT

The code you provided is written in COBOL (Common Business-Oriented Language), a programming language that is primarily used in business, finance, and administrative systems.

The program calculates the factorial of a given number. The factorial of a non-negative integer n is the product of all positive integers less than or equal to n.

Here is a detailed explanation of the code:

1. **IDENTIFICATION DIVISION and PROGRAM-ID**: This is the program identification section where the program is named.

2. **DATA DIVISION and WORKING-STORAGE SECTION**: In this section, all the variables used in the program are declared.

3. **PROCEDURE DIVISION**: This division contains the actual logic of the program.

4. **ACCEPT CMD-ARGS FROM COMMAND-LINE**: The program accepts a command-line argument and stores it in the variable `CMD-ARGS`.

5. It then checks if `CMD-ARGS` is alphabetic. If it is, it performs `ERROR-PARA` which displays an error message and stops the program.

6. If `CMD-ARGS` is not alphabetic, it converts `CMD-ARGS` to its numeric value and stores it in `DECINUM`. If `DECINUM` is negative, it performs `ERROR-PARA`.

7. The decimal number is then moved to a non-decimal number (`NUM`). If `NUM` and `DECINUM` are equal, it means the input was an integer.

8. If `NUM` is 0 or 1, it displays 1 and stops the program because the factorial of 0 and 1 is 1.

9. If `NUM` is greater than 1, it calculates the factorial of `NUM` by repeatedly multiplying `FACTORIAL` by `CNT` (which starts at 1 and is incremented by 1 each time) until `CNT` is greater than `NUM`.

10. It then left justifies the result by tallying the number of leading zeros in `FACTORIAL` and moving the characters in `FACTORIAL` from the position after the leading zeros to `LEFT-JUST-NUMBER`.

11. Finally, it displays the left justified result and stops the program.

12. If `NUM` and `DECINUM` are not equal, it means the input was not an integer and it performs `ERROR-PARA`.

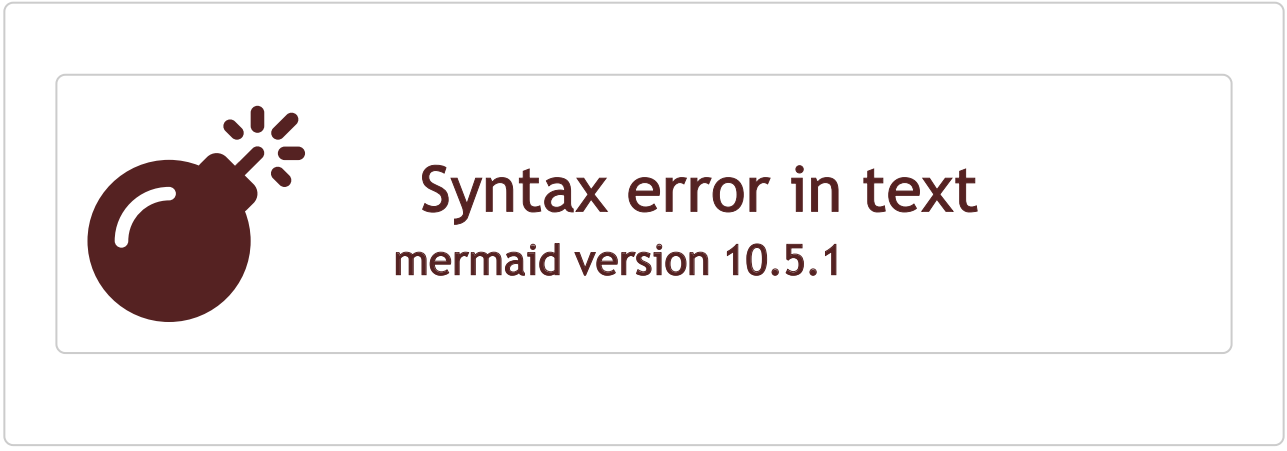
The algorithm used in this code is a simple loop that calculates the factorial of a number. It does not use any special algorithms or data structures.

For more information on COBOL, you can visit this link: COBOL tutorial

As per your architectural guidelines, this code doesn't conflict with any special constraints or guidelines since none were provided. It's a standalone COBOL program that performs a specific task.

Boost Flow Diagram

Last Updated: Saturday, October 21, 2023 at 12:26:59 PM PDT



The control flow graph above represents the control flow of the given source code. The primary path is shown in green, while the error path is shown in red.

The control flow starts at the `CMD-ARGS` block, where the command-line arguments are accepted. If the `CMD-ARGS` is alphabetic, the control flow goes to the `ERROR-PARA` block, which displays an error message and stops the program.

If the `CMD-ARGS` is numeric, the control flow goes to the `DECINUM` block, where the command-line argument is converted to its numeric value. If the `DECINUM` is less than 0, the control flow goes to the `ERROR-PARA` block, which displays an error message and stops the program.

If the `DECINUM` is greater than or equal to 0, the control flow goes to the `NUM` block, where the decimal number is moved to the non-decimal number `NUM` . If the `NUM` is equal to the `DECINUM` , the control flow goes to the `EQUAL TO` block.

If the `NUM` is equal to 0 or 1, the control flow goes to the `DISPLAY` block, which displays 1 and stops the program. If the `NUM` is greater than 1, the control flow goes to the `CALC-FACT` block, where the factorial is calculated until `CNT` is greater than `NUM` .

After the factorial calculation, the control flow goes to the `DISPLAY` block, which displays the left-justified result of the factorial. Finally, the control flow stops the program.

If any of the conditions are not met, the control flow goes to the `ERROR-PARA` block, which displays an error message and stops the program.

Boost Source-Level Security Analysis

Last Updated: Saturday, October 21, 2023 at 12:27:08 PM PDT

No bugs found

Boost Source-Level Performance Analysis

Last Updated: Saturday, October 21, 2023 at 12:27:42 PM PDT

1. **Severity**: 4/10

   Line Number: 14

   Bug Type: CPU

   Description: The use of FUNCTION NUMVAL for converting string to number is CPU-intensive. This function performs a lot of unnecessary checks for a simple integer conversion.

   Solution: Use a simpler conversion function or method if available in the COBOL version you are using. If not, consider writing a custom conversion routine that does the bare minimum required for your specific use case.

2. **Severity**: 6/10

   Line Number: 33

   Bug Type: CPU

   Description: The factorial calculation is done using a loop which is not efficient for large numbers. This can lead to high CPU usage and long execution times.

   Solution: Consider using a more efficient algorithm for calculating factorial. For example, you could use a recursive function or utilize a memoization technique to store previously calculated results. However, be aware that recursion in COBOL can be tricky and may not be supported in all versions.

3. **Severity**: 3/10

   Line Number: 40

   Bug Type: Memory

   Description: The INSPECT TALLYING FOR LEADING ZEROS operation is not efficient as it scans the entire string. This can be memory-intensive for large strings.

   Solution: Consider using a more efficient method to remove leading zeros. For example, you could write a loop that iterates over the string from the beginning and stops when it encounters a non-zero character.

Boost Source-Level Data and Privacy Compliance Analysis

Last Updated: Saturday, October 21, 2023 at 12:27:56 PM PDT

No bugs found