Modificación de COBOL Legado

SL. Práctica 1

Diego Sanz Fuertes | 825015

20 de Octubre de 2023

Índice

Resumen ejecutivo

Esta práctica consistía en realizar tres de las acciones más comunes en los sistemas legados, migrar código, migrar interfaces y añadir nuevas funcionalidades.

A partir del archivo fuente COBOL proporcionado, se pedía migrar el código de un dialecto de COBOL para que compilase con GnuCOBOL. Mediante los errores del compilador y consultas en la documentación oficial se ha creado un script que, mediante sustituciones, es capaz de convertir el fichero proporcionado en código compilable por GnuCOBOL. Estas sustituciones han consistido principalmente en el formato de las directivas `DISPLAY` y `ACCEPT` y en la inclusion de `END PROGRAM`. Una vez creados los archivos sobre los que trabaja la aplicación, el programa se ejecuta correctamente.

Para utilizar una interfaz basada en los formularios `SCREEN SECTION` también se ha hecho uso de un script que generaba formularios s a partir de las directivas `DISPLAY` y `ACCEPT`. Una vez generados, los formularios se han sustituido a mano en el código fuente. El resultado es un programa con una interfaz en la que se pueden rellenar los datos navegando con el tabulador.

Por último se ha añadido un nuevo `PROGRAM` a partir de los anteriores el cuál, en vez de leer o escribir un registro en el fichero, lo elimina.

Introducción

En esta práctica se pretende modificar una aplicación para la gestión de recursos humanos desarrollada en el lenguaje Cobol. Las modificaciones requeridas son las siguientes:

- Migrar el código de dialecto desconocido a GnuCOBOL.
- Sustituir la interfaz de usuario por una que utilize formularios mediante `SCREEN SECTION`.
- Añadir la funcionalidad de dar de baja a los empleados.

Parte 1: Migración del código

El desarrollo de esta práctica se ha llevado a cabo en Arch Linux con la versión 3.2.0 del compilador GnuCOBOL. Las herramientas utilizadas han sido:

- Visual Studio Code: como editor de texto con la extensión COBOL.
- Python: para la realización de los scripts.
- Obsidian y Pandoc: para la realización de este documento.

El archivo fuente proporcionado para la práctica no compilaba sin realizar ninguna modificación, por lo que se ha utilizado el compilador para guiar la migración.

El primer error indica un fallo en la sintaxis de la directiva `DISPLAY`. Mediante el uso de la documentación oficial [1] y documentación extra [2] se ha desarrollado un script en python para sustituir las sentencias incorrectas mediante expresiones regulares a sentencias siguiendo la documentación.

Para corregir todos los errores dados por las directivas `DISPLAY` y `ACCEPT` las cuales son muy similares se han utilizado las siguientes sustituciones:

- Para limpiar toda la pantalla:

```
line = re.sub(r"DISPLAY ?\(.*\) ?ERASE", "DISPLAY \" \" AT 0101 ERASE
    EOS", line)
```

- Para escribir `DISPLAY` en una sola linea:

```
if re.search(r"DISPLAY ?\(.*\)\n", line):
    line = line + file.readline().lstrip()
```

- Para colocar la fila y columna en la posición correcta:

```
line = re.sub(r"DISPLAY ?\((.*)\) ?\n?(.*)\.", r"DISPLAY \2 AT \1.",
    line)
```

```
line = re.sub(r"DISPLAY ?\((.*)\) ?\n?(.*)", r"DISPLAY \2 AT \1.", line
    )

line = re.sub(r"ACCEPT ?\((.*)\) ?(.*)\.", r"ACCEPT \2 AT \1.", line)
```

- Para separar varios valores en un `DISPLAY` a varios `DISPLAY`s porque no acepta más de 1 valor por directiva:

```
if match := re.search(r"DISPLAY ? \"(.*)\" (.*) AT (.*) (.*)\.", line):
    a = (f"           DISPLAY \"{match[1]}\" AT {match[3]} {match[4]}.\
       n")
    a = re.sub(r"AT (\d\d) (\d\d)", r"AT \1\2", a)
    a = re.sub(r"AT (\d\d) (\d)", r"AT \g<1>0\2", a)
    a = re.sub(r"AT (\d) (\d\d)", r"AT 0\1\2", a)
    a = re.sub(r"AT (\d) (\d)", r"AT 0\g<1>0\2", a)
    outfile.write(a)
    line = (
      f"DISPLAY {match[2]} AT {match[3]} {int(match[4])+ 1 +len(match
         [1])}.\n")
```

- Para evitar sobrescribir con `ACCEPT` la primera linea:

```
line = re.sub(r"ACCEPT (\w+)", r"ACCEPT \1 AT 0201", line)
```

- Para sustituir la fila y columna por el formato en 4 cifras FFCC:

```
line = re.sub(r"AT (\d\d) (\d\d)", r"AT \1\2", line)
line = re.sub(r"AT (\d\d) (\d)", r"AT \g<1>0\2", line)
line = re.sub(r"AT (\d) (\d\d)", r"AT 0\1\2", line)
line = re.sub(r"AT (\d) (\d)", r"AT 0\g<1>0\2", line)
```

Una vez arregladas las directivas `DISPLAY` y `ACCEPT` el compilador indica que los bloques `PROGRAM` no terminan. Al inspeccionar el código se ha detectado la necesidad de terminar el bloque mediante `END PROGRAM`. El cambio se ha llevado a cabo en el script de python para poder hacer el código original compilable de una manera automatizada que permite iterar en el proceso, en este caso, tres versiones del script.

```
current_program: str | None = None
while line := old.readline():
    ...
    if line.lstrip().startswith("IDENTIFICATION DIVISION."):
        if current_program is not None:
            new.write(f"       END PROGRAM {current_program}")
            current_program = None
    elif line.lstrip().startswith("PROGRAM-ID."):
        current_program = line.lstrip().split(' ')[1]

    new.write(line)
```

```
12
13 if current_program is not None:
14     new.write(f"       END PROGRAM {current_program}")
```

Una vez realizados todos estos cambios, el código ha podido ser compilado y ejecutado.

Antes:

```
1            DISPLAY(10 25) "1. HRMS WRITE".
2            DISPLAY(12 25) "2. HRMS READ".
3            DISPLAY(14 25) "3. EXIT".
4            DISPLAY(16 25) "ENTER YOUR CHOICE :".
5            ACCEPT(16 46) CHOICE.
```

Después:

```
1            DISPLAY "1. HRMS WRITE" AT 1025.
2            DISPLAY "2. HRMS READ" AT 1225.
3            DISPLAY "3. EXIT" AT 1425.
4            DISPLAY "ENTER YOUR CHOICE :" AT 1625.
5            ACCEPT CHOICE AT 1646.
```

Al ejecutar el código, la interfaz parecía funcionar pero los registros añadidos no se guardaban, para solucionar este problema, se han creado archivos con los nombres encontrados en el código fuente y al crear registros, los archivos tomaron un formato “Berkeley DB (Btree, version 10, native byte-order)” el cual ya permitió guardar y consultar los registros.

Parte 2: Interfaz de usuario

La segunda modificación a realizar consistía en sustituir las directivas `DISPLAY` con formularios mediante `SCREEN SECTION`s. Estos formularios permiten navegar entre los campos con el tabulador. `SCREEN SECTION` acepta cadenas, variables de lectura, y variables de lectura/escritura por lo que se pueden sustituir las directivas `DISPLAY` por el equivalente a cadenas y variables de lectura; y las directivas `ACCEPT` por las de lectura/escritura.

Para esta modificación también se ha desarrollado un script en python para asistir con la generación de `SCREEN SECTION`s. Se ha creado una clase en la que se puede almacenar el contenido de la directiva `ACCEPT` o `DISPLAY`, el tipo de `SCREEN SECTION`, la fila y la columna. El script genera un formulario para cada grupo de `ACCEPT` o `DISPLAY` separados por limpiezas de pantalla y los escribe a la salida estándar.

Una vez generados los formularios se han sustituido en el código fuente a mano, añadiendo las `SCREEN SECTION`s en las declaraciones del programa y llamándolas desde su anterior posición:

Antes:

```
       LEAVE-PARA.
           DISPLAY " " AT 0101 ERASE EOS.
           OPEN INPUT LEAVEFILE.
           DISPLAY "ENTER CODE :".
           ACCEPT LEMPID AT 0201.
           DISPLAY " " AT 0101 ERASE EOS.
           READ LEAVEFILE INVALID KEY GO TO ERROR-LEAVE-PARA.
           DISPLAY " CODE           :" AT 0101.
           DISPLAY LEMPID AT 0119.
           DISPLAY " DATE           :" AT 0201.
           DISPLAY LFMDATE AT 0219.
           DISPLAY " DATE           :" AT 0301.
           DISPLAY LTODATE AT 0319.
           DISPLAY " LEAVE CATEGORY :" AT 0401.
           DISPLAY LLEVCAT AT 0419.
           CLOSE LEAVEFILE.
           DISPLAY "PRESS ENTER TO RETURN TO HRMS READ MENU" AT 2010.
           STOP ' '.
           GO TO MAIN-PARA.
```

Después:

```
       ...
      01 leave-para-select.
           03 line 1 column 1 value "ENTER CODE :".
           03 line 2 column 1 using LEMPID.

       01 leave-para-menu.
           03 line 1 column 1 value " CODE           :".
           03 line 1 column 19 from LEMPID.
           03 line 2 column 1 value " DATE           :".
           03 line 2 column 19 from LFMDATE.
           03 line 3 column 1 value " DATE           :".
           03 line 3 column 19 from LTODATE.
           03 line 4 column 1 value " LEAVE CATEGORY :".
           03 line 4 column 19 from LLEVCAT.
           03 line 20 column 10 value
               "PRESS ENTER TO RETURN TO HRMS READ MENU".
       ...
       LEAVE-PARA.
           DISPLAY " " AT 0101 ERASE EOS.
           OPEN INPUT LEAVEFILE.
           accept leave-para-select.
           DISPLAY " " AT 0101 ERASE EOS.
           READ LEAVEFILE INVALID KEY GO TO ERROR-LEAVE-PARA.
           accept leave-para-menu.
           CLOSE LEAVEFILE.
           STOP ' '.
           GO TO MAIN-PARA.
```

Parte 3: Eliminar empleados

El objetivo de la tercera parte consistía en añadir funcionalidad al sistema legado en la forma de posibilitar la eliminación de empleados, ya que solamente se podían crear y consultar.

Para implementar esta funcionalidad se han añadido los menús necesarios, y basándose en los procedimientos de creación y consulta de empleados, un nuevo `PROGRAM` con el archivo de empleados, el formato del registro, las variables, los `SCREEN SECTION`s necesarios y el código para eliminar un registro de un fichero, el cual se tiene que abrir como I/O.

```
        EMP-PARA.
            DISPLAY " " AT 0101 ERASE EOS.
            OPEN I-O EMPFILE.
            accept emp-para-select.
            DISPLAY " " AT 0101 ERASE EOS.
            DELETE EMPFILE INVALID KEY GO TO ERROR-EMP-PARA.
            CLOSE EMPFILE.
            GO TO MAIN-PARA.
```

Conclusiones

La modificación de un programa escrito en COBOL puede resultar difícil y tediosa, por lo que estimar cuando hay que realizar los cambios a mano o cuando crear una herramienta que los haga por ti, es clave para reducir costes de mantenimiento. Por ejemplo, si en esta práctica se tuviera que modificar decenas o cientos de miles de lineas de código para migrar de `DISPLAY` a `SCREEN SECTION` no resultaría eficiente realizar la migración formulario por formulario como se ha realizado en esta práctica. Se debería buscar una herramienta ya existente o crear una nueva utilizando, por ejemplo, un parser de COBOL y generación de código fuente.

Para añadir funcionalidades extra, también se podría contemplar el uso de otros lenguajes compatibles con GnuCOBOL como puede ser C y C++, los cuales pueden ser llamados desde COBOL.

Referencias

- [1] Documentación oficial: (https://gnucobol.sourceforge.io/faq/master.html)
- [2] Documentación extra: (https://devdocs.io/gnu_cobol/index)