Universidad Zaragoza

Escuela Universitaria Politécnica - Teruel
Universidad Zaragoza

Práctica 1

Mantenimiento de código legado

Radu Constantin Robu
Alberto Pérez Blasco

Sistemas Legados

Escuela Universitaria Politécnica de Teruel
Departamento de Informática e Ingeniería de Sistemas

7 de noviembre de 2021

Resumen ejecutivo

Adaptación de un programa de cajero automático implementado en lenguaje de programación COBOL 85 a la versión más reciente de GnuCOBOL. Se han realizado los cambios pertinentes para hacer funcionar el programa de forma idéntica al funcionamiento del programa en COBOL 85.

Para comenzar con la ejecución del programa dado, se corrigieron errores para conseguir la compilación del código proporcionado en COBOL 85. Después de este proceso, se crearon programas auxiliares para escribir en los ficheros necesarios para la ejecución correcta del programa. Estos ficheros fueron ESPEC.DAT, USERS.DAT y LOGIN.DAT ya que el fichero MOVS.DAT, se escribía automáticamente cuando se realizaban distintos movimientos en el programa (ingresar dinero, comprar entradas de espectáculos...)

Finalmente, se han realizado los cambios necesarios para cambiar la representación del año de las fechas del programa al estándar de 4 dígitos, cambiar la representación de las cuentas de los usuarios al estándar IBAN actual, y permitir que un usuario disponga de hasta tres cuentas para una misma tarjeta. Para las fechas y el IBAN, se utilizó la herramienta refactor. Mientras que para las tres cuentas, se realizó un proceso más extenso implementando una pantalla específica y utilizando una variable, que especifica qué cuenta y saldo se están utilizando durante todo el proceso.

Índice

1. Introducción

La primera práctica de la asignatura de Sistemas Legados tiene como objetivo principal el mantenimiento de un programa de cajero automático implementado en el lenguaje de programación legado COBOL 85.

La tarea principal de la práctica es la portación del programa a la última versión de GnuCOBOL, realizando aquellos cambios necesarios para que el programa tenga un funcionamiento idéntico al funcionamiento del programa original.

Para llevar a cabo esta tarea, se ha tenido que instalar primero el entorno necesario para la programación en COBOL. Esto incluye la instalación de GnuCOBOL en las máquinas empleadas, instalación de un entorno de programación con soporte para COBOL y el estudio del manual del programador de COBOL.

Posteriormente, se deben implementar ciertos cambios en el programa para permitir que el programa funcione con estándares más recientes. Estos cambios incluyen el cambio del formato de la fecha para que el año tenga cuatro dígitos en vez de dos dígitos, cambiar el formato del número de las cuentas al formato internacional IBAN, y modificar el programa para permitir que un usuario tenga hasta 3 tarjetas de usuario.

Todos los cambios y modificaciones quedan reflejadas en detalle en este documento, incluyendo los aspectos más relevantes de cada cambio realizado.

2. Instalación del entorno y compilación del código

El entorno de trabajo para la realización de la práctica han sido dos máquinas con una distribución de Linux. En cada máquina se ha instalado la última versión de GnuCOBOL y se ha empleado el entorno integrado de desarrollo (*Integrated Development Environment*) VisualStudio Code.
También se probó el IDE para Windows de GNUCobol, pero al final al práctica se realizó en Linux debido a la mayor facilidad de trabajo en las máquinas de Linux con el uso de VisualStudio Code al estar ambos alumnos acostumbrados a su uso.

La primera tarea necesaria para la realización de la práctica ha sido el estudio del manual de GnuCOBOL para conseguir la el éxito durante la compilación del código original del programa.

A partir de la primera compilación del código original se obtendrán aquellos problemas principales que están impidiendo el funcionamiento del programa.

La compilación en el entorno de Linux se realiza a través de una terminal abierta en el directorio que contiene el código fuente del programa. Al principio de la práctica, se intentó realizar la compilación a través del comando `cobc -x -free -std=cobol2014 fichero.cbl` para conseguir compilar el código original empleando el estándar 2014 del lenguaje COBOL. Sin embargo, este comando de compilación producía resultados contradictorios con los resultados esperados del manual del programador de COBOL.

Estos resultados incluyen:

- Lecturas de teclado sin soporte aún cuando la sintaxis del código correspondía con la sintaxis del manual.
- Tratamiento de excepciones de teclado erróneas.
- Teclas del teclado que no era reconocidas por el programa.

Posteriormente, la compilación se realizó empleando el comando `cobc -x -free fichero.cbl`. Se observa que ambos comando emplean dos opciones: `-x` y `-free`.

- `-x`: indica al compilador que se desea producir un fichero ejecutable a partir del código.
- `-free`: indica al compilador que el formato del código va a ser libre, es decir, se ignoran las columnas del código y una línea de código empieza en la columna 1 y sigue hasta el final de línea.

Una vez conseguido que el código sea compilado por el compilador, se ha procedido a la solución de los errores que el compilador ha encontrado en el código original.

2.1. Errores de compilación

- **Comentarios incorrectos**

  Uno de los errores más sencillos de solucionar fue el formato incorrecto de las líneas de comentario. Un comentario en COBOL debe indicarse como:

  ```
  *> Comentarios correctos en COBOL
  ```

  Sin embargo, todos los comentarios del código original tenían el formato:

  ```
  * Comentarios incorrectos en COBOL
  ```

  Por tanto, todos los comentarios del código original tuvieron que ser modificados para tener el formato correcto.

  Además, en el código original se encontraron varias líneas de código cuyo único contenido era una serie de asteriscos seguidos. Estas líneas, aunque no provocaban errores de compilación, se eliminaron para favorecer la visibilidad del código.

- **Cambios en funciones**

  Se cambió el nombre de la función MENU debido a que la palabra MENU es una palabra reservada del lenguaje de programación COBOL, y por tanto no puede emplearse como nombre de función.

Código del menú original

```
MENU.
  PERFORM RESTAURAR-CAMPOS-ACCESO.
  DISPLAY PANTALLA-MENU-PRINCIPAL.
  PERFORM LEER-TECLA.

  IF CODIGO-TECLA = 27
    MOVE 0 TO NUM-ERRORES-CNUEVA
    MOVE 0 TO NUM-ERRORES-CACTUAL
    GO TO INICIO
  ELSE
    IF TECLA = 1
      GO TO CONSULTAR-SALDO
    ELSE
      IF TECLA = 2
        GO TO CONSULTAR-MOVS
      ELSE
        IF TECLA = 3
          GO TO RETIRAR-EFECTIVO
        ELSE
          IF TECLA = 4
            GO TO INGRESAR-EFECTIVO
          ELSE
            IF TECLA = 5
              GO TO HACER-TRANSFERENCIA
            ELSE
              IF TECLA = 6
                GO TO COMPRAR-ENTRADAS
              ELSE
                IF TECLA = 7
                  GO TO CAMBIAR-CLAVE
                ELSE
                  GO TO MENU.
```

Código del menú modificado

```
MENU-OPCIONES.
  PERFORM RESTAURAR-CAMPOS-ACCESO.
  DISPLAY PANTALLA-MENU-PRINCIPAL.
  PERFORM LEER-TECLA.

  IF COB-CRT-STATUS = 2005
    MOVE 0 TO NUM-ERRORES-CNUEVA
    MOVE 0 TO NUM-ERRORES-CACTUAL
    GO TO INICIO
  ELSE
    IF TECLA = 1
      GO TO CONSULTAR-SALDO
    ELSE
      IF TECLA = 2
        GO TO CONSULTAR-MOVS
      ELSE
        IF TECLA = 3
          GO TO RETIRAR-EFECTIVO
        ELSE
          IF TECLA = 4
            GO TO INGRESAR-EFECTIVO
          ELSE
            IF TECLA = 5
              GO TO HACER-TRANSFERENCIA
            ELSE
              IF TECLA = 6
                GO TO COMPRAR-ENTRADAS
              ELSE
                IF TECLA = 7
                  GO TO CAMBIAR-CLAVE
                ELSE
                  GO TO MENU-OPCIONES.
```

Otro cambio realizado tiene que ver con las llamadas a la función DISPLAY. En el código original, se empleaba la función DISPLAY junto con la palabra reservada LOW, que indica que el brillo de la pantalla deberá ser bajo durante la ejecución del programa. Sin embargo, esta palabra reservada causaba errores de compilación. Además, según el manual de COBOL, si en una llamada DISPLAY no se indica ninguna palabra reservada relacionada con el brillo, se usa HIGH (alto) por defecto. Por tanto se decidió eliminar la palabra LOW de las llamdas a la función DISPLAY.

Por último, se realizó un cambio en la pantalla PANTALLA-BIENVENIDA. En el código original, se empleaba de forma errónea las palabras reservadas para especificar los colores de la pantalla.

Código pantalla bienvenida original

```
01 PANTALLA-BIENVENIDA FOREGROUND IS WHILTE BACKGROUND IS BLUE.
   02 BLANK SCREEN.
```

Código pantalla bienvenida modificado

```
01 PANTALLA-BIENVENIDA FOREGROUND-COLOR IS 7
   BACKGROUND-COLOR IS 1.
```

Las palabras reservadas FOREGROUND-COLOR y BACKGROUND-COLOR fueron reemplazadas por FOREGROUND y BACKGROUND, y las palabras WHITE y BLUE fueron reemplazadas por el código del color indicado en el manual de COBOL.

3. Generación ficheros de datos

El programa original implementado en COBOL necesita 4 ficheros .DAT para su funcionamiento. Estos ficheros corresponden a los usuarios de la aplicación (USERS.DAT), las claves de acceso de los usuarios (LOGIN.DAT), los movimientos realizados entre cuentas de la aplicación (MOVS.DAT) y los espectáculos para los cuales se pueden comprar entradas (ESPEC.DAT).

Para el correcto funcionamiento de la aplicación, estos ficheros han tenido que ser creados, debido a que su ausencia no permite el funcionamiento de la aplicación. Algunos de estos ficheros, como USERS y LOGIN deben contener la información correspondiente a los datos del usuario y su clave de acceso. Por tanto, se han diseñado aplicaciones para la escritura de información en los dos ficheros.

Además, para cada uno de los ficheros de datos se han implementado pequeños programas auxiliares, que realizan la lectura de cada uno de los ficheros y muestran en pantalla los registros almacenados. De esta forma se podía comprobar que la escritura en los ficheros se realizaba de la forma esperada.

3.1. Creación ficheros USERS y LOGIN

Para el correcto funcionamiento del programa cajero es necesaria la existencia de un fichero que contenga a los usuarios del cajero, y un fichero que contenga información acerca del acceso al cajero de los usuarios.

Estos dos ficheros, USERS y LOGIN, necesitan ser creados antes de la ejecución del programa cajero, y deben contener los datos necesarios. Para ello, se ha implementado un programa auxiliar en COBOL que se encarga de la creación de los ficheros y de la administración de sus datos.

La primera tarea que ha sido necesaria realizar para el diseño e implementación del programa auxiliar ha sido entender la información que los ficheros deben contener, así como entender cómo se realiza la escritura de ficheros en COBOL.

Variables de los datos de los usuarios

```
FD USERFILE.
01 REG-USUARIO.
02 USER-TARJ                PIC 9(10).
02 USER-PIN                 PIC 9(4).
02 USER-DNI                 PIC X(9).
02 USER-NOM-APE             PIC X(30).
02 USER-TFNO                PIC X(9).
02 USER-DIRECCION           PIC X(25).
02 USER-BLOQUEADA           PIC X.
02 CUENTA-USUARIO OCCURS 3 TIMES.
    03 USER-NUM-CUENTA      PIC X(24).
    03 USER-SALDO           PIC 9(9)V99.
```

Variables de las claves de acceso

```
FD LOGINFILE.
01 REG-LOGIN.
02 LOGIN-TARJ               PIC 9(10).
02 LOGIN-NUM-INTENTOS       PIC 9.
```

Observando el código original de la aplicación, se ha determinado que el fichero USERS debe contener el número de tarjeta del usuario, el código PIN del usuario, el saldo del usuario, el DNI del usuario, el nombre y los apellidos del usuario, el teléfono, la dirección, si la tarjeta está bloqueada o no, y el número de cuenta del usuario.

En el fichero LOGIN se observa que la información necesaria para el usuario es el número de tarjeta y el número de intentos que se han realizado para acceder a la tarjeta.

Una vez conocidos los datos que los usuarios deben tener almacenados en el fichero de usuarios, se ha procedido a implementar el programa que permita añadir, modificar o eliminar usuarios del fichero de usuarios, y que permita añadir, modificar o eliminar las claves de acceso del fichero de acceso.

En el programa, se han declarado los registros que se van a necesitar, así como las variables necesarias, y se han implementado las funciones de escritura del fichero de usuarios y el fichero de acceso.

Declaración de los ficheros

```
FILE-CONTROL.
  SELECT USERFILE ASSIGN TO "USERS.DAT"
      ORGANIZATION IS INDEXED
      ACCESS MODE IS DYNAMIC
      RECORD KEY IS USER-TARJ
      FILE STATUS IS FSU.

  SELECT LoginFile ASSIGN TO  "LOGIN.DAT"
      ORGANIZATION IS INDEXED
      ACCESS MODE IS DYNAMIC
      RECORD KEY IS LOGIN-TARJ
      FILE STATUS IS FSL.
```

Se observa que el acceso a ambos ficheros se realiza de forma indexada, utilizando como clave de acceso el número de la tarjeta. Además, se han declarado dos variables `FSU` y `FSL` que se emplean para recoger el estado del fichero.

Cuerpo del programa

```
INICIO.
   PERFORM FIND-USERFILE.
   PERFORM FIND-LOGINFILE.
   DISPLAY CLEAR-SCREEN.
   DISPLAY PANTALLA-USUARIO-REGISTRADO.
   STOP RUN.
```

El programa primero intenta encontrar si existen los ficheros, si éstos existen, los abre para poder insertar nuevos usuarios o actualizar alguno ya existente.

Procedimiento de búsqueda del fichero USERS

```
FIND-USERFILE.
  OPEN INPUT USERFILE.
  IF FSU = "35"
      MOVE "NO" TO FICHERO-USERS-EXTEND
  ELSE
      PERFORM READ-USERFILE.
  CLOSE USERFILE.
  PERFORM WRITE-USER-DATA.
```

Procedimiento de lectura del fichero USERS

```
READ-USERFILE.
 READ USERFILE
 AT END MOVE "NO" TO FICHERO-USERS-EXTEND
 NOT AT END MOVE "SI" TO FICHERO-USERS-EXTEND.
```

El procedimiento de búsqueda de los ficheros, tanto del fichero USERS como del fichero LOGIN, emplea una característica propia de los ficheros, su estado. En COBOL, si se intenta abrir para lectura o escritura un fichero que no existe, el programa pone el valor de la variable declarada como FILE STATUS a 35. Por tanto, se ha empleado este valor para saber si el fichero existe o no.

Se emplea una variable denominada `FICHERO-USERS-EXTEND` para tener constancia de la existencia del fichero.

Si el fichero existe, lo abriremos para poder leer o escribir de él, principalmente para poder permitir actualizar alguno de los usuarios del programa.

Si el fichero no existe, el programa lo crea únicamente en el modo de escritura. Como el fichero estará vacío después de la creación, no hace falta leerlo para saber si contiene información o no

La tarea del procedimiento de lectura del fichero es simplemente comprobar si el fichero contiene información. Si no la contiene se puede escribir directamente. En cambio, si la contiene, esta no debe ser modificada a menos que el usuario administrador quiera realizar específicamente dicha tarea.

Después del condicional, el fichero se cierra debido a que no nos interesa leerlo, sino escribirlo o actualizarlo, acciones no disponibles en el modo `INPUT` del fichero. Posteriormente, se llama al procedimiento que se encarga de la escritura de datos del usuario en el fichero.

Procedimiento de actualización del fichero USERS

```
WRITE-USER-DATA.
  IF FICHERO-USERS-EXTEND = "SI"
      OPEN I-O USERFILE
  ELSE
      OPEN OUTPUT USERFILE.
  PERFORM ASK-USER-DATA.
  PERFORM WRITE-USERFILE.
  CLOSE USERFILE.
```

Este procedimiento es el que se encarga de la escritura o actualización del fichero USERS. Gracias a la constancia de presencia de información, se puede saber si el fichero debe ser abierto para actualización o para escritura.

Después del condicional, se pregunta por los datos del usuario a introducir o actualizar, se escribe la información en el fichero y se cierra el fichero.

Procedimiento de introducción de datos de usuario

```
ASK-USER-DATA.
  DISPLAY PANTALLA-DATOS-USUARIO.
  ACCEPT PANTALLA-DATOS-USUARIO
      IF COB-CRT-STATUS = 1009
          STOP RUN.
```

Este procedimiento se encarga de mostrar la pantalla de introducción de datos y de recoger los datos del usuario que son introducidos. En esta pantalla si se pulsa la tecla F9 se cierra el programa.

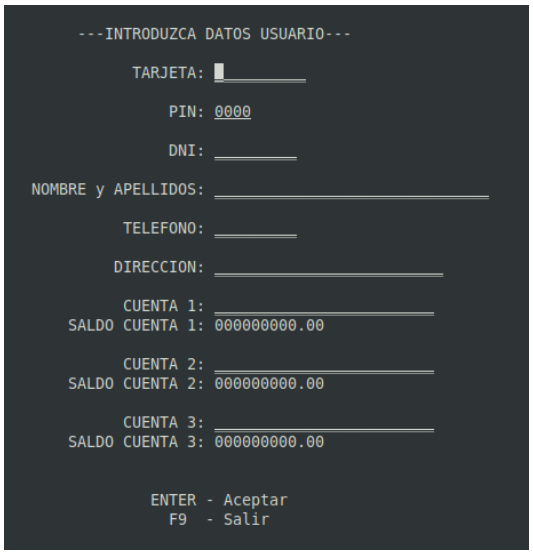

Figura 1: Pantalla de introducción de datos del programa.

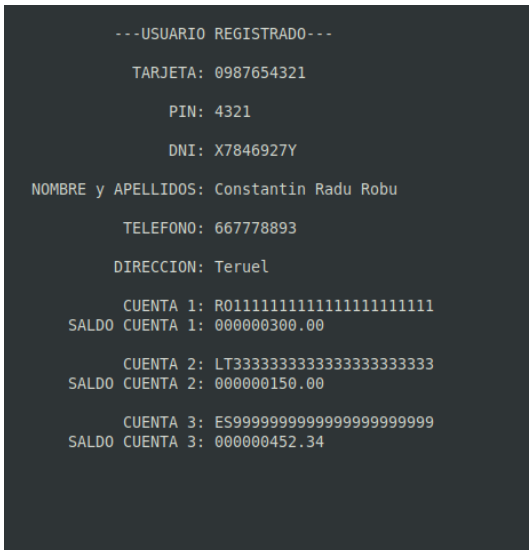

Figura 2: Un nuevo usuario introducido en el fichero.

Procedimiento de escritura de datos de usuario

```
WRITE-USERFILE.
  MOVE TARJ TO USER-TARJ.
  MOVE PIN TO USER-PIN.
  MOVE DNI TO USER-DNI.
  MOVE NOM-APE TO USER-NOM-APE.
  MOVE TFNO TO USER-TFNO.
  MOVE DIRECCION TO USER-DIRECCION.
  MOVE "0" TO USER-BLOQUEADA.
  MOVE CUENTA1 TO USER-NUM-CUENTA(1).
  COMPUTE SALDO1 = (CENTS1 / 100) + EUROS1.
  MOVE SALDO1 TO USER-SALDO(1).
  MOVE CUENTA2 TO USER-NUM-CUENTA(2).
  COMPUTE SALDO2 = (CENTS2 / 100) + EUROS2.
  MOVE SALDO2 TO USER-SALDO(2).
  MOVE CUENTA3 TO USER-NUM-CUENTA(3).
  COMPUTE SALDO3 = (CENTS3 / 100) + EUROS3.
  MOVE SALDO3 TO USER-SALDO(3).
  WRITE REG-USUARIO.
  REWRITE REG-USUARIO.
```

El procedimiento de escritura de datos se encarga de actualizar las variables que serán escritas en el fichero. Se observa que la final del procedimiento se emplea tanto la sentencia `WRITE REG-USUARIO` como `REWRITE REG-USUARIO`. Esto es intencional, debido a que `WRITE REG-USUARIO` no puede sobrescribir un usuario si este existe en el fichero.

Esta explicación cubre únicamente la parte de introducción de usuarios en el fichero USERS.DAT. Como se ha mencionado, el programa actualiza además el fichero LOGIN.DAT. Sin embargo, esta funcionalidad no necesita una explicación detallada, puesto que la forma en la que se busca el fichero, se lee y se escribe es igual a la del fichero USERS.

Para realizar la actualización del fichero LOGIN, se emplean procedimientos derivados de los del fichero USERS, cambiado únicamente el nombre del fichero y los datos a escribir en el fichero.

El usuario no tiene que introducir ningún dato por la entrada estándar ya que el fichero LOGIN se actualiza de forma automática en función del número de tarjeta que el usuario introduce en la pantalla de datos del usuario.

Procedimiento de escritura de datos de acceso

```
WRITE-LOGINFILE.
  MOVE TARJ TO LOGIN-TARJ.
  MOVE 0 TO LOGIN-NUM-INTENTOS.
  WRITE REG-LOGIN.
  REWRITE REG-LOGIN.
```

Se observa que el procedimiento coge el número de tarjeta introducido posteriormente por el usuario, y escribe esa variable y la variable `LOGIN-NUM-INTENTOS` en el fichero, de forma automática.

Una consideración a tener en cuenta a la hora de usar este programa es que una ejecución del programa permite añadir un usuario al fichero USERS. Por tanto, el

programa debe ser ejecutado cada vez que se desee insertar un usuario al fichero USERS.

3.2. Creación del fichero ESPEC

Para la creación del fichero de ESPEC, se ha seguido el mismo procedimiento que para la escritura de los ficheros de LOGIN y USERS, empleando los mismos métodos, pero en este caso solicitando los datos pertinentes.

```
01 REG-ESPECTACULO.
   02 ESPEC-NUMERO           PIC 99.
   02 ESPEC-NOMBRE           PIC X(20).
   02 ESPEC-PRECIO-ENTRADA   PIC 999V99.
   02 ESPEC-DESCRIPCION      PIC X(30).
   02 ESPEC-ENT-DISPONIBLES  PIC 9(3).
   02 ESPEC-FECHA.
      03 DDE                 PIC 99.
      03 FILLER              PIC X.
      03 MME                 PIC 99.
      03 FILLER              PIC X.
      03 AAE                 PIC 9999.
```

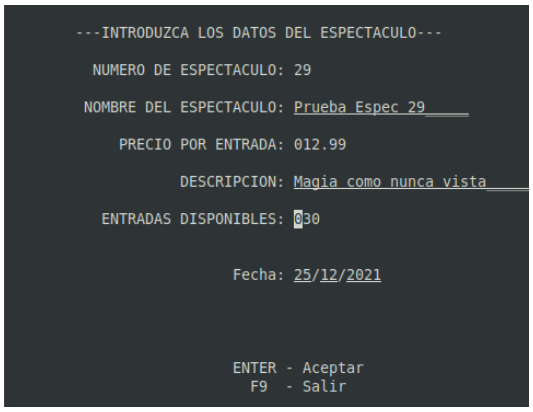

Figura 3: Datos del nuevo espectáculo.

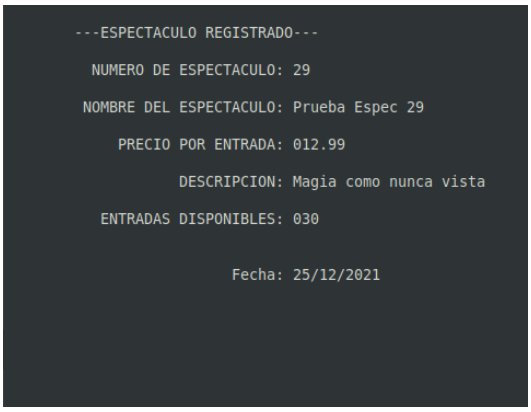

Figura 4: Espectáculo registrado.

3.3. Creación del Fichero MOVS

El último fichero necesario para el correcto funcionamiento del programa cajero es el fichero de almacenamiento de los movimientos realizados.

Este fichero no contiene una gran cantidad de código puesto que todos los procedimientos que usan este fichero únicamente requieren de su existencia. Por tanto, el fichero de creación del fichero MOVS.DAT contiene los procedimientos para crear el fichero y cerrarlo para que pueda ser usado por el programa cajero.

```
INICIO.
  PERFORM WRITE-MOVS-FILE.
  STOP RUN.

WRITE-MOVS-FILE.
  OPEN OUTPUT MOVSFILE.
  CLOSE MOVSFILE.
```

El fichero MOVS es un fichero de datos secuencial, que contiene la información relevante correspondiente a los movimientos del programa cajero. Aunque de momento el programa auxiliar implementado únicamente crea el fichero MOVS, su implementación a través de procedimientos se ha realizado por si en el futuro se añaden nuevas funcionalidades no contempladas en este momento.

En principio este programa no haría falta. Su principal función es la de crear el fichero si no existe, o borrar el fichero y escribirlo de nuevo. Sin embargo, este fichero es creado automáticamente por el programa cajero cuando se realiza un movimiento en el cajero. Por tanto, el programa es más bien una salvaguarda para los casos de error en los que no se crea el fichero MOVS.DAT.

4. Cambios de funcionamiento del cajero

4.1. Lectura de teclas especiales

4.1.1. Reemplazo de ESC por F9 y de las flechas arriba-abajo por F2-F3

En el código original del programa cajero se realizaba la lectura de las teclas del teclado mediante una variable llamada `CODIGO-TECLA`. Sin embargo, en la portación del programa a GnuCOBOL se detectó un problema con este método: no se detectaban las teclas especiales (ENTER, ESC, etc.), afectando el funcionamiento del programa.

Estudiando el manual de GnuCOBOL se halló una forma de poder capturar y tratar las teclas del teclado. Cada vez que el usuario pulsa una tecla especial del teclado, esta tecla queda reflejada en un registro especial propio del lenguaje de programación llamado `COB-CRT-STATUS`.

Por tanto, se decidió reemplazar todas las líneas de código en las que aparecen llamadas a la variable `CODIGO-TECLA` y emplear `COB-CRT-STATUS` para el tratamiento de teclas especiales.

Un ejemplo del cambio de `CODIGO-TECLA` por `COB-CRT-STATUS` se puede observar en el procedimiento que permite al usuario visualizar los movimientos realizados en su cuenta.

Sección de código original que empleaba CODIGO-TECLA

```
PERFORM LEER-TECLA.
  IF CODIGO-TECLA = 27
    PERFORM RESTAURAR-CAMPOS-MOVIMIENTOS
    GO TO MENU
  ELSE
    IF CODIGO-TECLA = 52
      IF NUM-PANTALLA-MOV = 1
      MOVE 12 TO LINEA-MOV
      GO TO MOSTRAR-PANTALLA-MOV
    ELSE
      SUBTRACT 1 FROM NUM-PANTALLA-MOV
      MOVE 12 TO LINEA-MOV
      GO TO MOSTRAR-PANTALLA-MOV
    ELSE
      IF CODIGO-TECLA = 53
        IF NUM-PANTALLA-MOV =
          TOTAL-PANTALLAS-MOV
          MOVE 12 TO LINEA-MOV
          GO TO MOSTRAR-PANTALLA-MOV
        ELSE
          ADD 1 TO NUM-PANTALLA-MOV
          MOVE 12 TO LINEA-MOV
          GO TO MOSTRAR-PANTALLA-MOV
      ELSE
        MOVE 12 TO LINEA-MOV
        GO TO MOSTRAR-PANTALLA-MOV.
```

Sección de código modificada para emplear COB-CRT-STATUS

```
PERFORM LEER-TECLA.
  IF COB-CRT-STATUS = 2005
    PERFORM RESTAURAR-CAMPOS-MOVIMIENTOS
    GO TO MENU-OPCIONES
  ELSE
    IF COB-CRT-STATUS = 2003
      IF NUM-PANTALLA-MOV = 1
        MOVE 12 TO LINEA-MOV
        GO TO MOSTRAR-PANTALLA-MOV
      ELSE
        SUBTRACT 1 FROM NUM-PANTALLA-MOV
        MOVE 12 TO LINEA-MOV
        GO TO MOSTRAR-PANTALLA-MOV
    ELSE
      IF COB-CRT-STATUS = 2004
        IF NUM-PANTALLA-MOV =
          TOTAL-PANTALLAS-MOV
          MOVE 12 TO LINEA-MOV
          GO TO MOSTRAR-PANTALLA-MOV
        ELSE
          ADD 1 TO NUM-PANTALLA-MOV
          MOVE 12 TO LINEA-MOV
          GO TO MOSTRAR-PANTALLA-MOV
      ELSE
        MOVE 12 TO LINEA-MOV
        GO TO MOSTRAR-PANTALLA-MOV.
```

Para realizar el cambio del uso de unas teclas por otras, se pasaron a utilizar los valores que tiene el registro para cada una de las teclas.
Al ejecutar el código, las teclas ESC y las flechas de arriba y abajo seguían sin funcionar, por lo que se optó por utilizar otras teclas, como son F9 para el ESC, F2 para arriba y F3 para abajo. Finalmente, las equivalencias de los códigos originales con los del registro 'COB-CRT-STATUS' son:

Tecla	CODIGO-TECLA	COB-CRT-STATUS
Esc	27	2005
Enter	13	0
Flecha arriba	52	2003
Flecha abajo	53	2004
F2		1002
F3		1003
F9		1009

Cuadro 1: Equivalencia de las distintas flechas

4.1.2. Vuelta al empleo de teclas especiales

Después de la implementación de toda la funcionalidad del programa, se decidió intentar modificar el programa cajero para volver a utilizar las teclas especiales (ESC, flecha arriba, flecha abajo) para el manejo del programa. Empleando como base programas implementados por la comunidad, se tomó nota de las condiciones necesarias para la captura y tratamiento de las teclas especiales por los programas

en COBOL.

La primera necesidad para la captura de teclas especiales en COBOL es la declaración del tratamiento de caracteres especiales en la sección `PROCEDURE DIVISION`.

```
PROCEDURE DIVISION.
SPECIAL-CHARACTERS.
    SET ENVIRONMENT 'COB_SCREEN_EXCEPTIONS' TO 'Y'.
    SET ENVIRONMENT 'COB_SCREEN_ESC' TO 'Y'.
```

Una vez declarado el medio para el uso de las teclas especiales, fue necesario recorrer el código buscando aquellas líneas de código en el que se empleaban las teclas F2 y F3, ya que ahora no se emplearían los códigos 1002 y 1003 del `COB-CRT-STATUS`, sino 2003 y 2004, equivalentes a las teclas flecha arriba y abajo.

Sin embargo, aunque se haya conseguido realizar el uso de las teclas especiales, la tecla ESC tiene una demora considerable entre el momento en el que se ha pulsado y la realización de la acción en la pantalla. Se especula que el motivo de la demora tiene que ver con la configuración de la terminal de comandos de LINUX.

Una posible solución a la demora según la comunidad [4] sería la modificación de una línea de un fichero de configuración, incluyendo `set -s escape-time 0` en el fichero.

No obstante, esta solución se ha encontrado para servidores basados en LINUX, por lo que la solución no ha sido probada en las máquinas empleadas para la realización de las prácticas.

4.2. Cambio de fechas

Para introducir el año completo a la hora de mostrar las fechas, se realizaron varios cambios:

- Modificación del tamaño de las variables de fecha (de 8 a 10 caracteres)
- Modificación del espacio designado en pantalla para las variables y contenedores en las tablas
- Cambios en los espaciadores para que mantener las mismas distancias con respecto al inicio

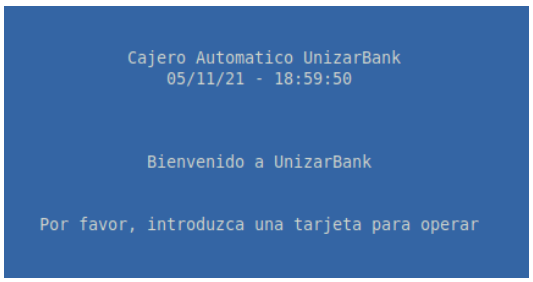

Figura 5: Fecha original con años con dos dígitos.

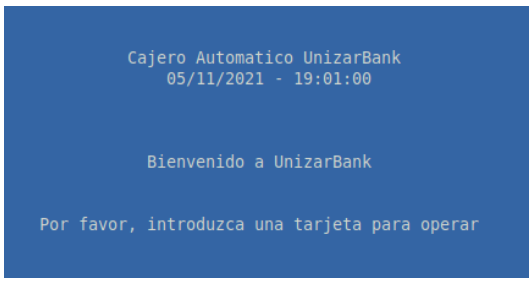

Figura 6: Fecha modificada con años con cuatro dígitos.

4.3. Cambiar cuentas a IBAN

Para el cambio de las cuentas a IBAN, se buscó el tamaño que conforma una cuenta de IBAN, comprobando así que su tamaño era de 24 caracteres alfanuméricos. En el código original, este número de cuenta aparecía con 20 por lo que se tuvo que cambiar en los datos del programa cajero y de los programas auxiliares para escribir los ficheros como se ha comentado en el apartado 3. Además, se tuvieron que modificar las pantallas del programa en las que aparecían los números de cuenta para tener en cuenta el nuevo tamaño.

Código original

```
02      MOV-CUENTA-DESTINO      PIC X(20).
```

Código nuevo

```
02      MOV-CUENTA-DESTINO      PIC X(24).
```

Finalmente, para reemplazar estos datos y para reemplazar las fechas y las secuencias de control de una forma semi-automática, se utilizó la herramienta de refactor que tiene Visual Studio Code para así poder realizar los cambios pero revisándolos uno a uno para comprobar si se realizaba sobre el campo correcto o sobre uno similar.

4.4. Permitir 3 cuentas para usuarios

A la hora de permitir que los usuarios puedan acceder a 3 cuentas con la misma tarjeta, las distintas fases de código por las que se ha pasado son:

Código original

```
01 REG-USUARIO.
   02 USER-TARJ            PIC 9(10).
   02 USER-PIN             PIC 9(4).
   02 USER-SALDO           PIC 9(9)V99.
   02 USER-DNI             PIC X(9).
   02 USER-NOM-APE         PIC X(30).
   02 USER-TFNO            PIC X(9).
   02 USER-DIRECCION       PIC X(25).
   02 USER-BLOQUEADA       PIC X.
   02 USER-NUM-CUENTA      PIC X(24).
```

Primera versión

```
01 REG-USUARIO.
   02 USER-TARJ            PIC 9(10).
   02 USER-PIN             PIC 9(4).
   02 USER-SALDO           PIC 9(9)V99 OCCURS 3 TIMES.
   02 USER-DNI             PIC X(9).
   02 USER-NOM-APE         PIC X(30).
   02 USER-TFNO            PIC X(9).
   02 USER-DIRECCION       PIC X(25).
   02 USER-BLOQUEADA       PIC X.
   02 USER-NUM-CUENTA      PIC X(24) OCCURS 3 TIMES.
```

Versión final

```
01 REG-USUARIO.
   02 USER-TARJ               PIC 9(10).
   02 USER-PIN                PIC 9(4).
   02 USER-DNI                PIC X(9).
   02 USER-NOM-APE            PIC X(30).
   02 USER-TFNO               PIC X(9).
   02 USER-DIRECCION          PIC X(25).
   02 USER-BLOQUEADA          PIC X.
   02 CUENTA-USUARIO          OCCURS 3 TIMES.
      03 USER-NUM-CUENTA      PIC X(24).
      03 USER-SALDO           PIC 9(9)V99.
```

Para la implementación de las 3 cuentas, la primera idea que se ha intentado aplicar ha sido permitir que cada usuario tenga un vector de 3 posiciones mediante `OCCURS`. Esto permitiría escribir las 3 cuentas en el fichero de una forma más sencilla.

Sin embargo, esta solución no se implementó debido a que la implementación mediante un struct `CUENTA-USUARIO` permitiría implementar un procedimiento más simple para el ingreso de dinero en una cuenta de destino durante la realización de una transferencia.

A la hora de facilitar la selección de una cuenta, se ha introducido una pantalla entre la de introducir el número de cuenta y el menú de acciones a realizar para una cuenta. En esta pantalla, le aparecen al usuario las distintas cuentas para que seleccione la cuenta con la que va a trabajar en el menú.

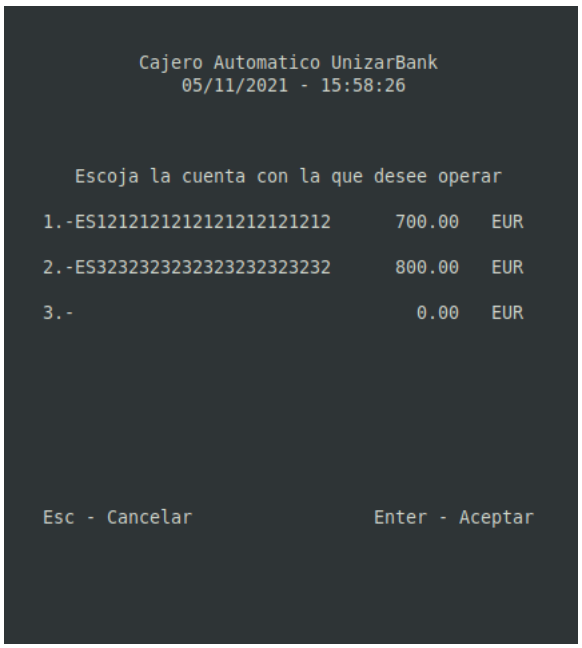

Figura 7: Captura de pantalla de la pantalla de selección de cuenta

Los cambios que se realizaron en el programa cajero (cajero.cbl) para la implementación de las 3 cuentas en el flujo del programa, fueron los siguientes:

4.4.1. Secuencia de ejecución

El método MOSTRAR-PANTALLA-SELECCION-CUENTA, se ejecuta después de comprobar si el acceso a la cuenta es correcto, los pasos que realizará este método serán:

```
MOSTRAR-PANTALLA-SELECCION-CUENTA.
    PERFORM LEER-SALDOS-CUENTAS.
    DISPLAY PANTALLA-SELECCION-CUENTA.
    ACCEPT PANTALLA-SELECCION-CUENTA.
    IF COB-CRT-STATUS = 2005
        PERFORM RESTAURAR-CAMPOS-ACCESO
        GO TO LOGIN.
    PERFORM OBTENER-CUENTA-SALDO-A-USAR.
```

Comprobando COB-CRT-STATUS en caso de que el usuario quiera salir pulsando ESC, para así restaurar los campos de acceso que introdujo el usuario (nº tarjeta y pin) y volviendo a la pantalla de login.

4.4.2. Lectura de las cuentas de un usuario en el fichero

```
LEER-SALDOS-CUENTAS.
    OPEN INPUT USERFILE.

    MOVE USER-NUM-CUENTA(1) TO WS-USER-NUM-CUENTA(1).
    MOVE USER-SALDO(1) TO WS-USER-SALDO(1).
```

```
MOVE USER-NUM-CUENTA(2) TO WS-USER-NUM-CUENTA(2).
MOVE USER-SALDO(2) TO WS-USER-SALDO(2).

MOVE USER-NUM-CUENTA(3) TO WS-USER-NUM-CUENTA(3).
MOVE USER-SALDO(3) TO WS-USER-SALDO(3).

CLOSE USERFILE.
```

En este método, se obtendrán del fichero las tres cuentas del usuario con sus respectivos saldos, para así poder mostrarlos en el display.

4.4.3. Creación del DISPLAY

```
01 PANTALLA-SELECCION-CUENTA.
   02 BLANK SCREEN.
   02 LINE 3 COL 26 VALUE "Cajero Automatico UnizarBank".
   02 LINE 4 COL 30 PIC X(10) FROM FECHAF.
   02 LINE 4 COL 41 VALUE "-".
   02 LINE 4 COL 43 PIC X(8) FROM HORAF.
   02 LINE 8 COL 20 VALUE
      "Escoja la cuenta con la que desee operar".
   02 LINE 10 COL 17 VALUE "1.-".
   02 LINE 10 COL 20 PIC X(24) FROM WS-USER-NUM-CUENTA(1).
   02 LINE 10 COL 45 VALUE "          .   EUR".
   02 LINE 10 COL 45 PIC ZZZZZZZ9.99 FROM WS-USER-SALDO(1).
   02 LINE 12 COL 17 VALUE "2.-".
   02 LINE 12 COL 20 PIC X(24) FROM WS-USER-NUM-CUENTA(2).
   02 LINE 12 COL 45 VALUE "          .   EUR".
   02 LINE 12 COL 45 PIC ZZZZZZZ9.99 FROM WS-USER-SALDO(2).
   02 LINE 14 COL 17 VALUE "3.-".
   02 LINE 14 COL 20 PIC X(24) FROM WS-USER-NUM-CUENTA(3).
   02 LINE 14 COL 45 VALUE "          .   EUR".
   02 LINE 14 COL 45 PIC ZZZZZZZ9.99 FROM WS-USER-SALDO(3).

   02 LINE 16 COL 44 PIC 9 USING SELECCION-CUENTA
       BLANK WHEN ZERO.
   02 LINE 23 COL 17 VALUE "Esc - Cancelar".
   02 LINE 23 COL 48 VALUE "Enter - Aceptar".
```

Para la creación del display, se tomaron de ejemplo algunos ya utilizados en el programa, para así poder seguir con la metodología utilizada en el programa, mostrando la fecha y hora y continuando con el enter o cancelar con ESC. Un ejemplo de visualización en ejecución, sería la figura 7. Finalmente, con el accept del display, se obtendrá el número de cuenta sobre el que quiere trabajar le usuario.

4.4.4. Uso de dos variables para almacenar la cuenta y saldo seleccionados

```
OBTENER-CUENTA-SALDO-A-USAR.
   IF SELECCION-CUENTA > 3
       PERFORM MOSTRAR-ERROR-SELECCION-CUENTA
   ELSE
       IF WS-USER-NUM-CUENTA(SELECCION-CUENTA) = CUENTA-VACIA
           PERFORM MOSTRAR-ERROR-SELECCION-CUENTA
       ELSE
```

```
MOVE WS-USER-NUM-CUENTA(SELECCION-CUENTA) TO CUENTA-SELECCIONADA
MOVE WS-USER-SALDO(SELECCION-CUENTA) TO SALDO-SELECCIONADO.
```

Para finalizar la ejecución de la selección de la cuenta a utilizar, se implementó este método para comprobar que el número introducido era correcto. Primero, se comprueba que el número sea mayor que 3 (para en ese caso mostrar una pantalla de error). Después, se comprueba si el número de cuenta no es la cuenta vacía (24 caracteres en blanco) y finalmente, se guarda en las variables CUENTA-SELECCIONADA Y SALDO-SELECCIONADO, la cuenta y saldo con el que se va a trabajar, teniendo que realizar este cambio en todos los lugares que se utiliza la cuenta o el saldo, ya que anteriormente, se utilizaban las variables USER-NUM-CUENTA Y USER-SALDO para este cometido.

4.4.5. Cambios en funciones que reescriben para que se realice correctamente

Una vez implementadas las 3 cuentas, se producía un error debido a que no se llegaba a escribir correctamente el saldo actualizado en el fichero, por lo que en la siguiente ejecución del programa, el usuario contaba con el mismo dinero que había comenzado en la ejecución anterior. A continuación, se muestra un ejemplo del código original y el código modificado en uno de estos casos:

Código original

```
COMPUTE USER-SALDO = USER-SALDO - DINERO-A-SACAR.
REWRITE REG-USUARIO.
```

Código modificado

```
COMPUTE SALDO-SELECCIONADO = SALDO-SELECCIONADO - DINERO-A-SACAR.
MOVE SALDO-SELECCIONADO TO USER-SALDO(SELECCION-CUENTA).
REWRITE REG-USUARIO.
```

4.5. Mostrar movimientos

A la hora de realizar la práctica también se han cometido errores durante el proceso de adaptación del código original a las nuevas especificaciones del programa.

Un ejemplo de estos errores se ha dado a la hora de realizar el cambio de formato del número de cuenta de 20 dígitos a 24 (según el estándar IBAN). Durante el proceso de cambio del número de cuenta, se pasó por alto el cambio del tamaño de una variable necesaria para el mostrado de los movimientos de una cuenta.
par

Para la búsqueda de los movimientos de una cuenta, el programa hace uso de dos variables: el número de cuenta y `MOV-ID`. `MOV-ID` representa el número de cuenta de un movimiento, empleado como clave que identifica al movimiento.

En el código original, las dos variables estaban declaradas como:

```
02      USER-NUM-CUENTA         PIC X(20).
02      MOV-ID                  PIC X(20).
```

Sin embargo, durante las modificación de las especificaciones del programa, se dejó de emplear `USER-NUM-CUENTA` para la búsqueda de movimientos puesto que ahora los usuarios pueden tener hasta tres cuentas. En el nuevo código de búsqueda, se emplea `CUENTA-SELECCIONADA`, declarada como:

```
77      CUENTA-SELECCIONADA     PIC X(24).
```

Sin embargo, el error cometido fue no estudiar detenidamente el código del procedimiento que se encargaba de la búsqueda de los movimientos. Esto llevó a que el programa no realizaba la búsqueda correcta de los movimientos.

Después de varios intentos se observó el porqué de este error. Aunque `CUENTA-SELECCIONADA` se había declarado como `PIC X(24)`, `MOV-ID` seguía estando declarado como `PIC X(20)` y por tanto el programa no podía encontrar los movimientos de la cuenta debido a la discordancia entre `MOV-ID` y `CUENTA-SELECCIONADA`.

4.6. Reflejar transferencia positiva en movimientos en la cuenta a favor

Otro cambio que se ha realizado en la realización de la práctica ha sido la programación de la funcionalidad correspondiente al ingreso del dinero de la transferencia en la cuenta de destino.

En el código original, al realizar una transferencia, el dinero era restado de la cuenta de origen, pero nunca se realizaba ese ingreso en la cuenta de destino. Esto claramente es un error bastante grave en un programa de banca y por tanto ha sido necesario corregir este error.

La idea detrás de la implementación ha sido fijarse en métodos existentes en el código y realizar los cambios necesarios para ajustar el programa a la nueva funcionalidad.

En principio, se han implementado tres nuevos procedimientos: un procedimiento que se encarga de realizar la transferencia del dinero, un procedimiento auxiliar que busca la cuenta correcta para añadir el dinero, y un procedimiento que guarda la transferencia en los movimientos de la cuenta destino como una "transferencia a favor"del dueño de la cuenta destino.

Los dos nuevos procedimientos principales se llaman dentro del procedimiento `HACER-TRANSFERENCIA` puesto que es aquí donde se realiza la transferencia entre cuentas.

Procedimientos llamados dentro de `HACER-TRANSFERENCIA`

```
PERFORM TRANSFERIR-DINERO-CUENTA-DESTINO
    THRU FIN-TRANSFERIR-DINERO.

PERFORM GUARDAR-MOV-TRANSFERENCIA-DESTINO.
```

4.6.1. Procedimientos para la transferencia de dinero

Como se ha mencionado anteriormente, el primer procedimiento implementado se encarga de realizar el ingreso de la suma de dinero transferida en la cuenta de destino.

Para ello, se tienen que leer el fichero de cuentas para ir recorriendo cada registro de usuario del programa. Si un número de cuenta del usuario corresponde con el número de cuenta introducido por el usuario, se pasa realiza el ingreso de dinero.

Procedimiento de búsqueda de cuenta

```
TRANSFERIR-DINERO-CUENTA-DESTINO.
  OPEN I-O USERFILE.

  INICIO-OBTENER-CUENTAS.
    READ USERFILE NEXT RECORD INTO WS-REG-USUARIO
      AT END GO TO FIN-OBTENER-CUENTAS.
    MOVE 1 TO M.

    PERFORM TRANSFERIR-DINERO UNTIL M = 4.

    GO TO INICIO-OBTENER-CUENTAS.

  FIN-OBTENER-CUENTAS.
  CLOSE USERFILE.
FIN-TRANSFERIR-DINERO.
```

Procedimiento auxiliar para la transferencia

```
TRANSFERIR-DINERO.
  IF WS-USER-NUM-CUENTA(M) = CUENTA-DESTINO
      COMPUTE WS-USER-SALDO(M) = WS-USER-SALDO(M) + DINERO-A-TRANSFERIR
      MOVE WS-USER-SALDO(M) TO USER-SALDO(M)
      REWRITE REG-USUARIO.
  ADD 1 TO M.
```

La tarea del segundo procedimiento es simplemente la de buscar el número de cuenta entre las 3 cuentas del usuario y realizar el ingreso del dinero en la cuenta.

4.6.2. Procedimiento de guardado de la transferencia

El último procedimiento implementado tiene como tarea guardar la transferencia realizada en el fichero de movimientos, con el fin de que la transferencia quede reflejada como un movimiento tanto en la cuenta de origen como la de destino.

Procedimiento implementado

```
GUARDAR-MOV-TRANSFERENCIA-DESTINO.
  OPEN EXTEND MOVFILE.

  MOVE CUENTA-DESTINO TO MOV-ID.
  MOVE "Transferencia a su favor" TO MOV-CONCEPTO.
  MOVE DINERO-A-TRANSFERIR TO MOV-CANTIDAD.
  MOVE " " TO MOV-CUENTA-DESTINO.
  PERFORM OBTENER-SALDO-CUENTA-DESTINO THRU FIN-OBTENER-SALDO.
```

```
MOVE SALDO-DESTINO TO MOV-SALDO.
PERFORM OBTENER-FECHA.
MOVE FECHAF TO MOV-FECHA.
MOVE HORAF TO MOV-HORA.
WRITE REG-MOVIMIENTOS.
CLOSE MOVFILE.
```

Este procedimiento se ha implementado empleado como base el procedimiento ya existente `GUARDAR-MOV-INGRESAR-EFECTIVO`. Al fin y al cabo, la transferencia en una cuenta destino no es más que un ingreso realizado desde otra cuenta. Por tanto, se ha podido emplear el código que guarda un ingreso para el procedimiento que guarda una transferencia entrante. Para que esto funcione se han tenido que modificar unas variables para que tenga sentido el movimiento, visto desde la cuenta de destino.

El método guarda en el fichero MOVS.DAT el movimiento como un ingreso denominado "Transferencia a su favor.

4.6.3. Demostración de funcionamiento

Para que se aprecie el funcionamiento en un caso cualquiera del programa, se va a realizar una transferencia desde una cuenta a otra.

Se ha preparado el entorno del programa eliminando todas las entradas del fichero MOVS.DAT para que se vean solamente los movimientos implicados en la transferencia.

Figura 8: Cuentas del usuario que va a realizar la transferencia.

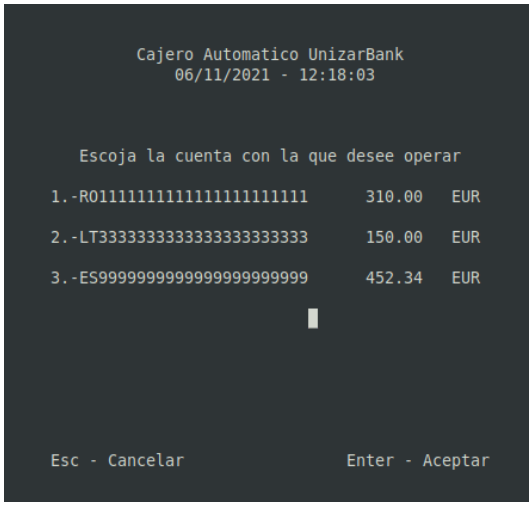

Figura 9: Cuentas del usuario que va a recibir la transferencia.

Para este ejemplo, se va a realizar una transferencia por un valor de 80 euros desde la segunda cuenta del usuario de origen (ES3232323232323232323232) a la segunda cuenta del usuario de destino (LT33333333333333333333333).

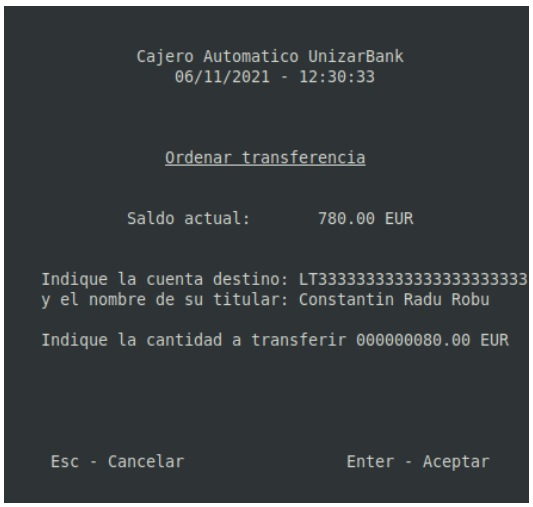

Figura 10: Datos introducidos por el usuario de origen.

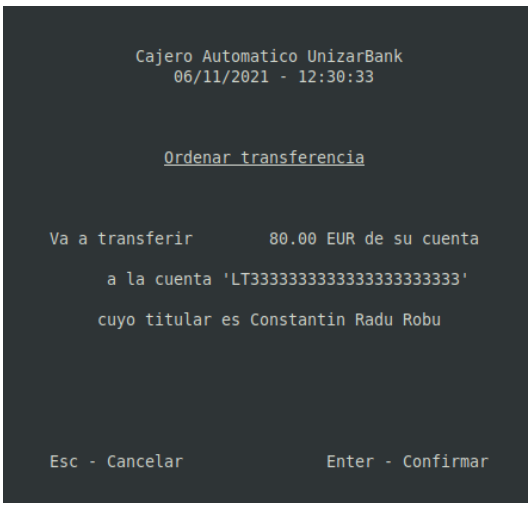

Figura 11: Confirmación de los datos de la transferencia.

Una vez realizada la transferencia se puede en los saldos de las cuentas y en el registro de movimientos.

Figura 12: Cuentas del usuario origen tras la transferencia.

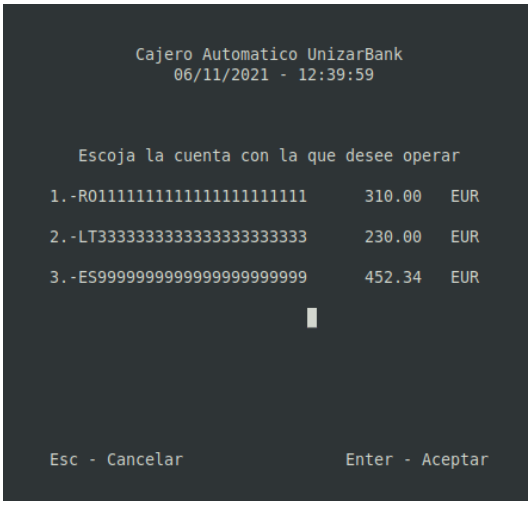

Figura 13: Cuentas del usuario destino tras la transferencia.

Se observa que la suma de 80 euros se ha restado al saldo de la cuenta de origen y se ha sumado al saldo de la cuenta destino.

Figura 14: Transferencia reflejada en la cuenta origen.

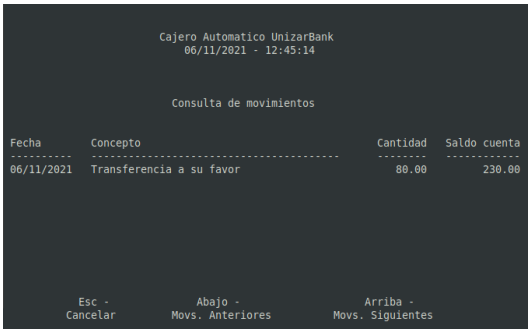

Figura 15: Transferencia reflejada en la cuenta destino.

5. Conclusiones

Para concluir, esta práctica ha sido un primer contacto con el lenguaje de programación COBOL, junto con sus características y peculiaridades. Gracias a esta práctica se ha visto la importancia del mantenimiento de código legado, especialmente en programas implementados en lenguajes que disponen de una comunidad más reducida.

Como trabajo futuro sobre este programa, se podrían implementar ciertas barreras de seguridad y cambios en el flujo del programa. Esto se debe a que en la actualidad, ciertas funcionalidades del programa no corresponden con el funcionamiento propio de un programa de banca. Un ejemplo, sería que el programa no contempla, ni en el código original ni en el código nuevo, el nombre del titular para la realización de una transferencia. El programa ingresa el dinero en todas las cuentas cuyo número de cuenta es igual al número de cuenta de la transferencia (dos personas distintas pueden tener acceso a la misma cuenta).

Referencias

[1] NISHIDA, Keisuke, WHILE, Roger, TIFFIN, Brian and SOBISCH, Simon, 2020, GnuCOBOL Manual. *GnuCOBOL - GNU Project* [online]. 10 November 2020. [Accessed 13 October 2021]. Available from: https://gnucobol.sourceforge.io/doc/gnucobol.pdf

[2] MONTES, Andres, [no date], Cobol en Español. *COBOL EN ESPAÑOL* [online]. [Accessed 2 November 2021]. Available from: http://www.escobol.com/index.php

[3] The foreground-color clause, [no date]. *Digital Transformation and Enterprise Software Modernization* [online], [Accessed 2 November 2021]. Available from: https://www.microfocus.com/documentation/enterprise-developer/ed50pu3/ED-Eclipse/HRLHLHPDF50B.html

[4] SHRIKANT SHARATSHRIKANT SHARAT 3, VICENT MARTIVICENT MARTI 1, KTFKTF 2 and DTYLERDTYLER 28611 SILVER BADGE44 BRONZE BADGES, 1960, ESC key causes a small delay in terminal due to its alt+ behavior. *Unix & Linux Stack Exchange* [online]. 1 January 1960. [Accessed 5 November 2021]. Available from: https://unix.stackexchange.com/questions/23138/esc-key-causes-a-small-delay-in-terminal-due-to-its-alt-behavior