NewElements

IT-Schulungen.com

COBOL - Grundkurs

Dr.sc.nat. Michael J.M. Wagner, New Elements*

Revision 1.109

*michael@wagnertech.de

Inhaltsverzeichnis

IT-Schulungen.com
Portfolio

IT-Schulungen.com ist eines der führenden, herstellerunabhängigen Seminarportale von Schulungen rund um die Informationstechnologie (IT) und das IT-Management. Seit über 15 Jahren ist IT-Schulungen.com eine anerkannte Anlaufstelle für viele Unternehmen und Behörden, wenn es um die Durchführung von DACH-weiten Schulungen geht.

- Applikationsserver / Middleware
- Business Intelligence
- Business-Skills und Führung
- Cloud
- CRM
- Datenbanken
- eBusiness
- ERP-Systeme
- IT Management
- IT-Recht / Lizenzierung
- ITIL
- Mobile
- Multimedia
- Office
- Open Source
- Portale
- SAP®
- Security
- Serversysteme
- Softwareentwicklung
- Systemmanagement

Zertifizierungen & Partnerschaften

Abbildung 1: Grace Hopper

1 Einstieg

Geschichte

COBOL steht für *Common Business Oriented Language*. COBOL wurde Ende der 50'ger Jahre entwickelt.

Situation in den 50-ger Jahren:

- Jeder Rechner spezielle Hardware
- Darauf spezielles Betriebssystem
- Assember-Programmierung
- Dringender Wunsch, eine hardware-unabhängige standardisierten problemorientierten Sprachen
- Im technisch-wissenschaftlichen Bereich gab es bereits FORTRAN

COBOL wurde nun für die Erstellung von Programmen für den betriebswirtschaftlichen Bereich entwickelt. Eine vom amerikanischen Verteidigungsministerium eingesetzte Arbeitsgruppe entwickelte einen Standard. Namentlich beteiligt hieran war Grace M. Hopper (Abb. 1).

Besondere Anforderungen im betriebswirtschaftlichen Bereich:

- große Datenmengen
- Ein- und Ausgabe

Weitere Entwicklung

- 1960 als COBOL-60 von CODASYL verabschiedet
- Heute am weitesten (noch?) verbreitete Programmiersprache
- Klassische Host-Sprache
- Weiterentwicklungen (xCOBOL, XML-Parser)
- Weitere Standardisierung 1985

Programmaufbau

Ursprünglich wurde COBOL nur in Großbuchstaben geschrieben (man hatte nur Lochkarten und Zeilendrucker ohne Kleinbuchstaben). Heute wird nicht zwischen Groß- und Kleinschreibung unterschieden (COBOL ist somit nicht case sensitive).

Ein COBOL-Programm ist in Teile (`DIVISION`), Kapitel (`SECTION`) und Abschnitte (`PARAGRAPH`) gegliedert. Die vier `DIVISION`s sind in ihrer festgelegten Reihenfolge:

- `Identification Division` mit dem Programmnamen und einigen weitgehend obsoleten Paragraphen für Kommentare;
- `Environment Division`, wo Schnittstellen zum Betriebs- und dessen Dateisystem definiert werden. Hier lässt sich bei Bedarf der Dezimaltrenner auf das im Deutschsprachigen übliche Komma umstellen:
  `DECIMAL-POINT IS COMMA.`
- `Data Division` mit der Definition der Programmvariablen und Datenstrukturen und
- `Procedure Division` mit den prozeduralen Anweisungen

Strikte Trennung von Datendeklaration und prozeduralen Anweisungen.

Im Prozedurteil kann man nur Variablen benutzen, die vorher im Datenteil deklariert worden sind (völlig anders bei FORTRAN). Das Kodierschema bei Cobol entspricht der Lochkarte mit ihren 80 Spalten:

- Spalte 1 bis 6: Zeilennummen
- Spalte 7: Kennzeichnung einer Kommentarzeile
- Spalte 8 bis 11 (Area A): Namen von Divisions, Sections und Paragraphen
- Spalte 12 bis 72 (Area B): Anweisungen (statements)
- Spalte 73 bis 80: Markierungen wie z. B. den Namen des Programms oder von Quelltext-Elementen

Heutige COBOL-Compiler nehmen dieses Schema, je nach Einstellung, nicht so streng.

Zum Übersetzen von COBOL-Programmen auf dem PC stehen u.A. folgende Umgebungen zur Verfügung:

- GNUCobol-Compiler für die Kommandozeile
- OpenCobolIDE (wird nicht mehr gepflegt)
- Verschiedene COBOL-Plugins für VSCode
- MicroFocus-IDE for Eclipse

Hier ein Beispielprogramm:[1]

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HALLOWELT.
PROCEDURE DIVISION.
DISPLAY "Hallo Welt".
STOP RUN.
END PROGRAM HALLOWELT.
```

Aufgabe:

Übersetzen Sie das Beispielprogramm.

2 Definition von Daten

In der DATA DIVISION müssen alle im Modul verwendeten Daten statisch definiert werden. COBOL arbeitet ausschließlich mit statischen Daten.

- Vorteil: COBOL-Programme laufen sehr stabil. Der Ressourcenbedarf kann vorab genau ermittelt werden. Es gibt keine bösen Überraschungen wie *stack overflow*, etc.
- Nachteil: Datenstrukturen werden oft „sicherheitshalber" überdimensioniert.

Die DATA DIVISION enthält:

- `File Section`: Dateien, auf die im Modul zugegriffen werden
- `Working-Storage Section`: Daten, die innerhalb des Moduls verwendet werden
- `Linkage Section`: Daten, die mit anderen Modulen ausgetauscht werden

Die DATA DIVISION [Ste Kap. 7.7.3, `src/picture.cbl`]

Weitere Möglichkeiten in der DATA DIVISION:

- REDEFINES: Speicher ist mit unterschiedlichen Namen ansprechbar.

```
01 WS-RECORD.
  03 WS-NAME    PIC X(30).
  03 WS-DOB     PIC 9(8).
  03 WS-DOB-XX  REDEFINES WS-DOB PIC X(8).
```

- FILLER:

```
01 EMPLOYEE-RECORD.
  03 FILLER          PIC X(37).
  03 EMPLOYEE-GENDER PIC X.
  03                 PIC X(135).
```

[1] Ralph Steyer, COBOL Grundlagenkurs, Herdt-Verlag 2017

- Formatierte Felder (*numeric edited*): Diese dienen zur Ausgabe (Bildschirm/Datei). Hier können Zahlenformate mit zusätzlichen Zeichen angereichert werden. [src/numeric_edited.cbl]

Initialisierung von Datenfeldern:

```
INITIALZE EMPLOYEE-RECORD
```

INITIALIZE füllt immer mit dem Defaultwert (ZERO/SPACES) des Datentyps. Will man folgendes erreichen: „Fülle mit den angegebenen Werten (falls vorhanden), ansonsten mit dem Defaultwert" lautet die Anweisung:
`INITIALIZE struktur ALL TO VALUE THEN TO DEFAULT`

Damit Datenstrukturen, die in mehreren Modulen Verwendung finden nicht stets neu definiert werden müssen (was fehleranfällig ist), kann die Definition in eine Datei („Copystrecke") ausgelagert werden und mit COPY in die COBOL-Datei geladen werden.

Sollen innerhalb einer Datenstruktur die Felder gleich heißen und sich nur der Name der gesamten Struktur (01-Ebene) ändern, empfiehlt sich folgendes Vorgehen:

- Die Definitionen der Stufen > 01 kommen in eine Datei `employee.cpy`.
- In der `Working Section` des Moduls kommt folgende Definition:

```
01 EMPLOYEE-1.
COPY "employee.cpy".
01 EMPLOYEE-2.
COPY "employee.cpy".
```

Zahlen und Zeichen lassen sich auch über hexadezimale Zahlen vorbelegen. Hinter VALUE steht dann beispielsweise

- `X"CA84"` für zwei Zeichen mit dem entsprechenden Hex-Code. Vorsicht: Auf IBM-Rechnern gilt nicht unbedingt eine ASCII-Codierung!
- `H"CA84"` für die Zahl 51844.[2]

> Aufgabe:
>
> Lassen Sie das Programm [src/numeric_edited.cbl] einmal mit und einmal ohne DECIMAL-POINT IS COMMA laufen und achten Sie auf die Unterschiede in der Ausgabe.

> Aufgabe:
>
> Mit den folgenden Übungen soll eine „Büchereianwendung" gebaut werden.

[2] https://www.ibm.com/support/pages/how-display-hexadecimal-using-cobol (29.10.2024)

- Legen Sie ein Projekt *Bucherei* mit dem Modul `Bucherei.cbl` und der Copystrecke `medium.cpy` an.
- Die in `medium.cpy` definierte Datenstruktur soll folgende Elemente enthalten:
  - `SIGNATUR`: Alphanumerisch, 4 Stellen
  - `AUTOR`: Alphabetisch, 30 Stellen
  - `TITEL`: Alphanumerisch, 100 Stellen
  - `TYP`: Alphabetisch, 1 Stelle
  - `SEITENZAHL`: Numerisch, 5 Stellen
  - `SPIELDAUER`: Numerisch, 5 Stellen als `REDEFINE` auf `SEITENZAHL`

  Der Defaultwert für die Signatur sei `A000`, für den Typ `B` (Buch).
- In der `Working Storage Section` des Moduls soll eine Instanz der Struktur angelegt werden.
- In der `Procedure Division` soll die Struktur initialisiert und ausgegeben werden.

3 Programm-Anweisungen

3.1 Verben

Gliederung: [Ste Kap. 6 Einl., `src/gliederung.cbl`]

Das Ende einer Anweisung wird nicht mehr mit einem Punkt gekennzeichnet. Der Punkt steht nur noch am Ende eines Paragraphen.

COBOL-Anweisungen beginnen stets mit einem *Verb*. COBOL-Verben sind:

- `DISPLAY`: Anzeige auf dem Bildschirm
- `ACCEPT`: Lesen von der Tastatur
- `STOP`: Beenden des Programms
- `MOVE`: Wertzuweisung an Datenfelder
- `PERFORM`: Steuerung der Programmlogik
- `IF`: Verzweigunen
- `EVALUATE`: Verzweigunen
- `COMPUTE`: Berechnungen
- `GO TO`: Wird nicht mehr verwendet

Einfache Verben: [Ste 8.1, `verben.cbl`]

`PERFORM`: [Ste 9.2.3, nur erstes Bsp., `kap9/perform1.cbl`]

Aufgabe:

Ergänzen Sie die Büchereianwendung. Diese soll nun aus folgenden Paragraphen bestehen:

- `PROG.` ruft die weitern Paragraphen nacheinander auf.
- `READ-RECORD.` belegt mit 6 `ACCEPT`-Aufrufen die Datenstruktur.
- `PRINT-OUT.` gibt die Datenstruktur aus.

3.2 Schleifen

Schleifen mit `PERFORM`: [Ste 9.3.1, `src/perform2.cbl`, `kap9/countdown2.cbl`]

Aufgabe:

Um mit den verschiedenen Schleifentypen vertraut zu werden, lösen Sie jede Aufgabe jeweils auf beide Arten:

```
PERFORM Para UNTIL ...

PERORM UNTIL ...
... code
END-PERFORM
```

- Sie geben eine Zahl ein und das Programm gibt alle Zahlen von 1 beginnend bis zur eingegebenen Zahl aus (`VARYING ...`).
- Sie geben so lange Zahlen ein, bis die Summe der eingegeben Zahlen größer 100 ist. Falls die erste Zahl bereits größer 100 ist, darf keine weitere Zahl mehr eingegeben werden.

3.3 Verzweigungen

Für Verzweigungen im Programmcode stehen das Verb `EVALUATE`, die `IF`-Anweisung, sowie die Verwendung von *level 88*-Strukturen zur Verfügung.

`IF`-Anweisung: [Ste 9.1.1, `kap9/if1-6.cbl`]

`IF-ELSE IF`-Kaskaden sind in COBOL weniger üblich. Diese werden mit `EVALUATE TRUE` erzeugt, s.u.

Level 88 ist ein Konstrukt, mit dem zweierlei erreicht werden kann:

- Sprechende Namen für ausgezeichnete Werte (z.B. „Buch" für „B")

- Prüfung, ob die Variable diesen ausgezeichneten Wert hat

```
01 MEDIUM-TYP PIC A.
  88 BUCH VALUE 'B'.
  88 CD  VALUE 'C'.
```

Ein `INITIALIZE MEDIUM-TYP` setzt die Variable auf „Leerzeichen", damit ist weder `BUCH` noch `CD` wahr.

Bei so einer Konstruktion sollte ein
`MOVE 'B' TO MEDIUM-TYP` ersetzt werden durch
`SET BUCH OF MEDIUM-TYP TO TRUE.`

Entsprechend kann die Abfrage
`IF MEDIUM-TYP = 'B'` ersetzt werden durch
`IF BUCH OF MEDIUM-TYP.`

Weitere Möglichkeiten von *level 88*: [Ste 9.1.3, `kap9/bereichstest.cbl`]

Mit *level 88* lässt sich auch eine einfache Datumsprüfung bauen:

```
01 MY-DATE.
  05 DATE-YEAR  PIC 9999.
  05 DATE-MONTH PIC 99.
    88 VALID-MONTH VALUE 1 THRU 12.
  05 DATE-DAY   PIC 99.
    88 VALID-DAY  VALUE 1 THRU 31.
```

`EVALUATE`: [Ste 9.1.2.2, `kap9/eval.cbl, kap9/bewertung.cbl`]

Weitere Beispiele:

```
EVALUATE a-var
WHEN 2
   ...
WHEN 3 THRU 7
   ...
WHEN OTHER
   ...
END-EVALUATE

EVALUATE a-var ALSO b-var
WHEN 2 ALSO "nicht"
...
```

Aufgabe:

Ergänzen Sie die Büchereianwendung um eine Prüfung der Eingabe:

- Legen Sie dazu in der `WORKING-STORAGE SECTION` eine Variable `is-record-valid` mit den Ausprägungen `IS-VALID` und `NOT-VALID` an und initialisieren Sie ihn mit `IS-VALID`.
- Schreiben Sie einen Paragraphen `CHECK-RECORD`, der die Daten des gelesenen Records prüft. Falls ein Fehler gefunden wird, wird `IS-RECORD-VALID` auf `NOT-VALID` gesetzt.

- Die Gültigkeit des Medientyps prüfen Sie über ein *level 88*-Kennzeichen in der Copy-Strecke.
- Fügen Sie einen Paragraphen `READ-IN` hinzu, der `READ-RECORD` so lange aufruft, bis der Record gültige Daten enthält.

4 Modulaufruf

In größeren Projekten wird der Code auf mehrere Module aufgeteilt. Dazu gibt es verschiedene Designansätze:

- Strukturiertes Design: Der Code wird horizontal geschichtet. Aufgerufen werden dürfen nur Module der direkt darunter liegenden Schicht.
- Objektorientiertes Design: Alle dieselbe Datenstruktur (=Objekt) betreffenden Operationen werden in einem Modul zusammengefasst. Aufrufe sind auch innerhalb einer Ebene erlaubt.
- Komponentenorientiertes Design: Hier wird auf oberster Ebene versucht, die Aufgaben auf Komponenten zu verteilen, um die „Monolithenbildung" zu vermeiden.
- Serviceorientiertes Design: Ähnlich dem komponentenorientierten Design, nur dass hier das Augenmerk auf den zur Verfügung gestellten Schnittstellen und deren Verhalten gelegt wird.
- Ein grundlegendes Prinzip ist die Trennung von Aspekten, beispielsweise fachlicher und technischer Gesichtspunkte.

Cobol kennt als Compileeinheit nur das Modul. Die kompilerten Module stehen am Host in einer Objektbibliothek zur Verfügung und werden bei Bedarf geladen und aufgerufen. Der Versuch ein nichtexistenes Modul zu rufen führt zu einem Laufzeitfehler.

Der Linuxcompiler unterscheidet im Gegensatz zum Host ausführbare Module (Hauptprogramm) und rufbare Module, die als *shared libraries* abgelegt werden.

Die Datenübergabe zwischen zwei Modulen erfolgt über einen gemeinsamen Speicherbereich, der in der `LINKAGE SECTION` definiert wird.

`CALL`: [Ste 10.1, `src/main.cbl, unterprogramm1.cbl`]

Anmerkungen:

- Die `LINKAGE SECTION` definiert einen gemeinsamen Speicherbereich, in den sowohl Aufrufer wie auch Gerufener schreiben können. Wer also was schreibt/liest ist „Verhandlungssache".
- Standardmäßig entspricht das Verhalten einem *call by reference*. Alternativ kennt COBOL:
  - `USING BY CONTENT`: Hier legt der Compiler einen weiteren Speicherbereich an, in den beim Aufruf die Daten kopiert werden. Ändert das gerufene Programm die Daten, hat dies keine Auswirkungen auf den Aufrufer.

  - USING BY VALUE: Hier wird der Wert direkt in das Unterprogramm kopiert. Dies ist nur für Ganzzahl-Variablen möglich.

- Es wird stets das Modul als ganzes gerufen. Daher ist es häufig üblich einen action-Parameter vorzusehen und im Modul dann auf entsprechende Paragraphen zu verzweigen. [CobolClassEclipse/Programm12.cbl/12a.cbl]

Aufgabe:

Stellen Sie das Beispiel src/main.cbl, unterprogramm1.cbl auf *call by content* um.

Aufgabe:

In der Büchereianwendung sollen alle Operationen, die die Medien-Datenhaltung betreffen, in ein eigenes Modul ausgelagert werden.

- Erstellen Sie das Module MedienModul.
- Das Modul sollen gundsätzlich folgende generische Datenoperationen unterstützen. Nicht implementierte Operationen solln eine DISPLAY-Ausgabe machen und einen status-code setzen.
  - Create-Record: Anlegen eines neuen Datensatzes
  - Read-Record: Lesen eines einzelnen Datensatzes mit Hilfe seines Schlüssels
  - Update-Record: Aktualisierung eines Datensatzes
  - Delete-Record: Löschen eines Datensatzes
  - Read-All: Prüft an einem entsprechenden Kennzeichen, ob bereits eine Datenoperation läuft. In diesem Fall wird der nächste Datensatz gelesen und zurückgegeben. Falls nicht wird ein neuer Lesevorgang gestartet und der erste Datensatz zurückgegenen. Optional kann ein Filterausdruck mitübergeben werden.
- Verwenden Sie für die Schnittstelle Copystrecken.

Um COBOL an das Verhalten von modernen Programmiersprachen etwas heranzuführen, wurde das interne Unterprogramm (*nested program*) eingeführt. Dazu erhält ein Modul innerhalb der PROCEDURE DIVISION weitere komplette Porgramme mit alles Divisionen. Damit lässt sich ein Verhalten wie ein Prozeduraufruf einer strukturierten Sprache nachahmen.

Nested Programs.[3]

[3] https://www.ibm.com/docs/en/cobol-linux-x86/1.2?topic=programs-nested

5 Dateioperationen

COBOL unterscheidet Dateiorganisation und Zugriffsmethode. Dateiorganisation:

- `SEQUENTIAL`: Datensätze liegen hintereinander ohne Zeilenumbruch
- `LINE SEQUENTIAL`: Datensätze liegen hintereinander mit Zeilenumbruch
- `RELATIVE`: Durchnummerierte Datensätze mit Zugriff über Satznummer
- `INDEXED`: Mit Schlüsselindex, auch mehrere Schlüssel möglich

Zugriffmethode:

- `SEQUENTIAL`: Zugriff lesend **oder** schreibend, Satz für Satz
- `RANDOM`: Auch lesend **und** schreibend möglich. Wahlfreier Zugriff über Satznummer oder Index
- `DYNAMIC`: Sequenziell und wahlfrei möglich

5.1 Sequenzielle Dateien

Die einfachste Art von Dateien sind einfache Textdateien. Diese Dateien kommen auch häufig vor. Um in COBOL damit arbeiten zu können, muss für Dateien folgendes definiert sein:

- Definition des Dateihandles [`kap11/datei1.cbl`, `ENVIRONMENT DIVISION`]
- Definition des Dateiformats [`kap11/datei1.cbl`, `DATA DIVISION`]

Zur Bearbeitung von Dateien in der `PROGRAM DEVISION` stehen dann folgende Verben zur Verfügung:

- `OPEN`: Öffnen der Datei
- `READ`: Datensatz in den `FD` lesen
- `WRITE`: Datensatz aus dem `FD` in die Datei schreiben
- `CLOSE`: Datei schließen

Beispiel: [Ste 11.4, 11.5, `kap11/datei1-2.cbl`]

Das Kopieren der Daten aus/in den FD-Record kann in einem Schritt mit dem `READ/WRITE` erfolgen:

```
MOVE ws-data TO fd-record
WRITE fd-record
* ist das gleiche wie
WRITE fd-record FROM ws-data

READ file-desc
MOVE fd-record TO ws-data
* ist das gleiche wie
READ file-desc INTO ws-data
```

Soll geprüft werden, ob ein Dateizugriff erfolgreich war, muss bei der Definition des Dateihandles ein FILE STATUS mit angegeben werden und dieser dann im Programm abgefragt werden: [Ste 11.6]

```
FILE-CONTROL.
  SELECT MEDIA-FILE ASSIGN "Medien.csv"
  FILE STATUS IS file-status
  ORGANIZATION LINE SEQUENTIAL.
...
WORKING-STORAGE SECTION.
01  file-status  PIC XX.
...
WORK SECTION.
OPEN EXTEND MEDIA-FILE
  EVALUATE file-status
  WHEN '00'
    CONTINUE
  WHEN '35'
    * File is not existing
...
```

Anmerkungen:

- Der MicroFocus-Compiler ist hier nicht so streng: Wenn bei EXTEND die Datei nicht vorhanden ist, wir kein Fehler geworfen, sondern die Datei angelegt.
- Am Host bezeichnet der Dateiname im SELECT einen Dateideskriptor, der in der JCL einer konkreten Datei zugewiesen werden muss.
- Die MicroFocus-Umgebung ermöglicht ein entsprechendes Vorgehen in der Laufzeiteinstellung.
- Beim GnuCOBOL-Compiler kann mit der Compileroption `-ffilename-mapping` eingestellt werden, dass der Name im SELECT als Umgebungsvariable interpretiert wird, aus der dann zur Laufzeit der Dateiname gelesen wird.

Aufgabe:

Analysieren Sie die Beispielprogramme:

- `SequentialRead.cbl`
- `DecisionMakingProgram1.cbl`
- `DecisionMakingProgram2.cbl`
  In `DecisionMakingProgram1.cbl` finden sich alphanumerische Entscheidungen über Stufe-88-Ausdrücke, in `DecisionMakingProgram2.cbl` numerische.

Aufgabe:

Ergänzen Sie die Büchereianwendung um Dateizugriffe:

- Ergänzen Sie das Hauptprogramm um eine Auswahl, welche Operation durchgeführt werden soll (`Lesen, Anlegen, Andern, Loschen, Alle Ausgeben`).
- Im `MedienModul` soll der gelesene und geprüfte Datensatz der Datei `medien.txt` hinzugefügt werden.
- Bei *Alle Ausgeben* rufen Sie zuerst `Read-First`, danach in einer Schleife nach der Ausgabe `Read-Next` auf (*read ahead*-Steuerung).
- Ergänzen Sie das Programm um die Option einen einzelnen Datensatz aus der Datei zu lesen. Dazu fragen Sie die Signatur ab. Im `MedienModul` wird in einer Leseschleife nach der entsprechenden Signatur gesucht. Dazu wird:
  - Ein `gefunden-knz` mit `NICHT-GEFUNDEN` initialisiert,
  - die Datei zum Lesen geöffnet und Zeile für Zeile gelesen,
  - Für jeden Datensatz geprüft, ob der gelesene Datensatz der abgefragten Sigatur gleicht. Falls ja, wird die Datei geschlossen und das `gefunden-knz` auf `DS-GEFUNDEN` gesetzt,

  Das `gefunden-knz` wird dann auf einen entsprechenden Rückgabewert abgebildet.

  Im Hauptprogramm wird nun der Datensatz, falls er gefunden wurde, ausgegeben. Falls nicht wird eine entsprechende Fehlermeldung ausgegeben.
- Das Anlegen eines neuen Datensatzes wird nun auch um den Aufruf von `Read-Record` erweitert: Falls der Datensatz bereits vorhanden ist, wird das Anlegen abgebrochen, da die Signatur schon im System vorhanden ist.

Für das Sortieren von Dateien stellt COBOL eigene Funktionalität zur Verfügung. Da beim Sortieren stets Datensätze zwischengespeichert werden müssen, muss dazu in der `FILE SECTION` mit dem Kennzeichen `SD` ein *sort descriptor* zur Verfügung gestellt werden.
.[Ste 11.8, datei4.cml]

Aufgabe:

Die Mediendatei soll stets sortiert sein.

- Nach dem Hinzufügen eines neuen Datensatzes in `Create-Record` soll nun im Paragraph `Medien-Sortieren` der Inhalt von `medien.txt` nach `medien.neu` sortiert werden.
- Danach wird im Paragraph `Medien-Kopieren` der Inhalt von `medien.neu` Zeile für Zeile nach `medien.txt` rückübertragen.

5.2 Indizierte Dateien

In den 60ger Jahren waren relationale Datenbanksysteme, wie sie seit den 80gern üblich sind, noch nicht vorhanden. Daher gibt es für COBOL die Möglichkeit Datenbankfunktionalität rudimentär

auf Dateien abzubilden. Die Möglichkeit hierfür bieten „indizierte Dateien". Wie in Datenbanken üblich werden die einzelnen Datensätze über „Schlüssel" eindeutig angesprochen.

```
SELECT file-name4 ASSIGN TO "dat4.dat"
ORGANIZATION IS INDEXED
ACCESS MODE IS RANDOM
RECORD KEY EMPLOYEE-NO

OPEN I-O file-name4

* Lesen
MOVE 123 TO EMPLOYEE-NO
READ EMPLOYEE-FILE
  INVALID KEY
    DISPLAY 'RECORD NOT FOUND'
    STOP RUN
END-READ

* Anlegen
WRITE EMPREC
  INVALID KEY
    DISPLAY 'CANNOT INSERT RECORD '
    STOP RUN
END-WRITE

* Uberschreiben
REWRITE EMPREC
  INVALID KEY
    DISPLAY 'FAILURE TO REWRITE '
    STOP RUN
END-REWRITE

* Loschen
DELETE EMPLOYEE-FILE
  INVALID KEY
    DISPLAY 'FAILURE TO DELETE'
    STOP RUN
END-DELETE

CLOSE file-name4
```

Beispiel: [`COBOLCLass/IndexedFileProgram.cbl`]
Anmerkung: Die binären Formate von MicroFocus und GnuCobol unterscheiden sich!

Aufgabe:

- Kleine Variante:
  Schreiben Sie ein Programm, das eine neue *employee*-Datei mit einem einzelnen Datensatz anlegt.
- Große Variante:
  Schreiben Sie ein Programm, das den Anwender nach *employee*-Daten fragt und diese in die neue Datei schreibt. Die Datei soll `I-O` geöffnet werden. Beim Schreiben soll geprüft werden, ob es sich um einen gültigen Schlüssel handelt.

- In beiden Fällen soll dann `IndexedFileProgram.cbl` mit dieser Datei lauffähig sein.

Aufgabe:

Ergänzen Sie Büchereianwendung um ein `NutzerModul`.

- Ein Benutzer-Datensatz habe die Felder `Benutzernummer, Vorname, Nachname, Ort.`
- Die Daten werden in einer indizierten Datei abgelegt.
- Implementieren Sie die oben genannten `CRUD`-Operationen.
- Ergänzen Sie das Hauptprogramm
  - um eine zusätzliche Auswahl, ob Benutzer oder Medien verwaltet werden sollen,
  - um die Eingabe der Benutzernummer und weiterer Datenfelder, falls nötig.
  - Benutzen Sie eine zweidimensionale Auswertung von Datenart/Operation mit `EVALUATE ... ALSO ....`

6 Datenoperationen

Numerische Operationen

`ADD, SUBSTRACT, MULTIPLY, DIVIDE`: [Ste 8.2.1, `kap8/calc1.cob`]

Diese doch etwas seltsam anmutende Syntax wird in altem Code aus Performancegründen bevorzugt. Bei neueren Compilern empfiehlt sich `COMPUTE`, da diese Formulierung der in anderen Programmiersprachen ähnelt: [Ste 8.2.2, `mathverbs.cbl`]

Da aufgrund der fixen Größendefinition von numerischen Feldern oft ein Überlauf droht, können numerische Operationen mit `ON SIZE ERROR` abgesichert werden:

```
COMPUTE WS-INVOICE-AMT = WS-ITEM-PRICE * WS-QUANTITY
  ON SIZE ERROR
    DISPLAY 'ERROR - TOTAL IS TOO LARGE'
END-COMPUTE
```

Operationen für Zeichenketten

`INSPECT`: [Ste 10.3.1, `kap10/ZEICHENKETTEN.cbl, string2.cbl`]

`STRING`: [Ste 10.3.2, `string3.cbl`]

`UNSTRING`: [Ste 10.3.3, `STRINGSPLITTEN.cbl`]

Teilzeichenketten:

```
01 WS-ALPHABET     PIC X(26) VALUE 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'.
01 WS-SMALLER-STRING PIC X(10) VALUE SPACES.

MOVE WS-ALPHABET(12:6) TO WS-SMALLER-STRING.
* This will result in WS-SMALLER-STRING containing "LMNOPQ"

MOVE WS-ALPHABET(12:) TO WS-SMALLER-STRING.
* This will result in WS-SMALLER-STRING containing "LMNOPQRSTU"
```

Aufgabe:

Die Verwendung von Dateien mit fester Feldlänge ist heute eher unüblich. Ein übliches Format sind *comma seperated values* (CSV). Ergänzen Sie Büchereianwendung um eine Ausleiheverwaltung.

- Legen Sie ein Modul `AusleiheModul` an, das dieselben generischen Datenoperationen vorsieht. Implementiert werden soll erst mal nur die Ausleihe (Hinzufügen eines Datensatzes), sowie `READ-FIRST/READ-NEXT`.
- Ein Ausleihe-Datensatz habe die Felder `Benutzernummer, Signatur, Ausleihedatum (PIC 9(8))`.
- Das Datumsformat soll wie in 3.3 beschrieben geprüft werden.
- Vor dem Schreiben muss die Zeichenkette mit `STRING` aufgebaut werden.
- Nach dem Lesen muss die Zeichenkette mit `UNSTRING` zerlegt werden.
- Ergänzen Sie das Hauptprogramm
  - um die Auswahl, ob ein Medium ausgeliehen werden soll,
  - um die Eingabe der Datenfelder,
  - um die Prüfung der Gültigkeit von Benutzernummer und Signatur,
  - um die Ausgabe aller Ausleihen.

7 Tabellen

Eindimensionale Tabellen: [Ste 10.4.1, `kap10/ARRAY1.cbl`]

Mehrdimensionale Tabellen: [Ste 10.4.2, `kap10/ARRAY2.cbl`]

Ein Index, die performante Alternative zum Subscript, enthält den Offset des Datensatzes im Speicher: [Ste 10.4.3, 10.4.4.1, `kap10/array3.cbl`]

Indizierte Tabellen können durchsucht werden: [Ste 10.4.5, `kap10/array4.cbl`]

Hat eine Tabelle zusätzlich einen Schlüssel und ist die Tabelle nach diesem Schüssel sortiert, so kann nach diesem Schlüssel mit `SEARCH ALL` sehr effektiv zugegriffen werden (Binärsuche): [`CobolClassEclipse/Petfoodtablesearchall.cbl`]

Tabellen mit variabler Länge:

```
01 CUSTOMER-RECORD.
  03 CUSTOMER-NO        PIC 9(6).
  03 CUSTOMER-BALANCE   PIC S9(8)V99.
  03 CUSTOMER-INVOICE-CNT PIC 999.
  03 CUSTOMER-INVOICE
    OCCURS 1 TO 500
    DEPENDING ON CUSTOMER-INVOICE-CNT.
    05 INVOICE-NO       PIC 9(6).
    05 INVOICE-DATE     PIC 9(8).
    05 INVOICE-AMT      PIC 9(6)V99.

ADD 1 TO CUSTOMER-INVOICE-CNT
MOVE WS-INVOICE-NO TO INVOICE-NO(CUSTOMER-INVOICE-CNT)
MOVE WS-TODAYS-DATE TO INVOICE-DATE(CUSTOMER-INVOICE-CNT)
MOVE WS-INVOICE-AMT TO INVOICE-AMT(CUSTOMER-INVOICE-CNT)
```

Aufgabe:

In der Büchereianwendung sollen nun die Ausleihen in einer Tabelle verwaltet werden:

- Ergänzen Sie `MedienModul` um eine indizierte Tabelle.
- Vor der ersten Datenoperation soll der gesamte Dateiinhalt in eine interne Tabelle variabler Länge geladen werden. Das bedeutet, dass vor jeder Datenoperation geprüft werden muss, ob die Tabelle geladen wurde. Dies kann daran erkannt werden, dass die Länge der Tabelle größer null ist.
- Implementieren Sie `Read-Medium` mit der Signatur als Schlüssel.
- Implementieren Sie `Schreibe-Tabelle`, das die Datei mit dem aktuellen Inhalt der Tabelle überschreibt.
- Implementieren Sie nun alle Datenoperationen:
  - Beim Hinzufügen wird nach der Prüfung der Datensatz wie gehabt an die Datei angehängt, danach die Tabelle neu gelesen.
  - Bei den anderen datenändernden Operationen wird zuerst die Tabelle geändert, dann die Datei neu geschrieben.

8 Embedded SQL

Als in den 80'gern relationale Datenbanken mit der Abfragesprache SQL (*structured query language*) aufkamen, wurde SQL eng in COBOL integriert, SQL wurde in COBOL eingebettet. Eine typische eingebettete Codesequenz sieht dabei so aus:

```
* CURSOR definieren
EXEC SQL
   DECLARE C1 CURSOR FOR
   SELECT EMP_NO, EMP_NAME, EMP_SALARY
```

```
         FROM EMP
         ORDER BY EMP_NO
END-EXEC.

* CURSOR öffnen
EXEC SQL
   OPEN C1
END-EXEC.

* read ahead
EXEC SQL
   FETCH C1 INTO :EMP-NO, :EMP-NAME, :EMP-SALARY
END-EXEC.

PERFORM UNTIL SQLCODE NOT = ZERO
   MOVE  EMP-NO       TO    D-EMP-NO
   MOVE  EMP-NAME     TO    D-EMP-NAME
   MOVE  EMP-SALARY   TO    D-EMP-SALARY
   DISPLAY D-EMP-REC

   * Nachlesen
   EXEC SQL
      FETCH C1 INTO :EMP-NO, :EMP-NAME, :EMP-SALARY
   END-EXEC
END-PERFORM.
```

Auf IBM-Großrechnern ist typischerweise ein DB2 Datenbank-Managementsystem (DBMS) in Verwendung. Bei der Verwendung auf dem PC dient hier das *open source* DBMS PostgreSQL. Dieses gilt als eines der zuverlässigsten in diesem Bereich. Für die Verbindung COBOL-PostgreSQL existiert OCESQL[4]. Eine ausführliche Installationsanleitung findet sich hier[5].

Wie auch am Großrechner wird hier dem eigentlichen Compilevorgang ein Präcompiler vorgeschaltet, der die *embedded SQL*-Ausdrücke in COBOL-CALLs umwandelt.

Auf der Entwicklungsumgebung sind folgende Werkzeuge eingerichtet:

- Ein PostgreSQL-Datenbanksystem mit einer Testdatenbank (`testdb`) und einem zugehörigen Nutzer mit Passwort (`cobc/cobc`)
- `ocecobc`: Compiler für Programme mit *embedded SQL*
- `ocerun`: Skript zur Ausführung dieser Programme

Aufgabe:

Bringen Sie das Beispiel `INSERTTBL`, `FETCHTBL` zum Laufen. Dazu müssen in dem Programm die Zugangsdaten zur Datenbank richtig eingetragen werden.

[4] `https://github.com/opensourcecobol/Open-COBOL-ESQL` (5.11.2024)
[5] `https://bigdanzblog.wordpress.com/2020/10/28/embedded-sql-for-gnucobol-using-ocesql/` (13.6.2025)

Aufgabe:

Implementieren Sie für eine Entität der Büchereianwendung die Datenhaltung in der Datenbank. Dazu erstellen Sie ein neues Verwaltungsmodul, das dieselbe Schnittstelle anbietet.

Im Hauptprogramm können nun dynamische Aufrufe verwendet werden, die wahlweise das Datei- oder das Datenbankbackend verwenden.

Die verwendeten Tabellen legen Sie zuvor am Kommandozeilen-Client an.

9 Quellen

Micro Focus, *Visual COBOL - Modern COBOL for the next generation*

Ralph Steyer, COBOL Grundlagenkurs, Herdt-Verlag 2017