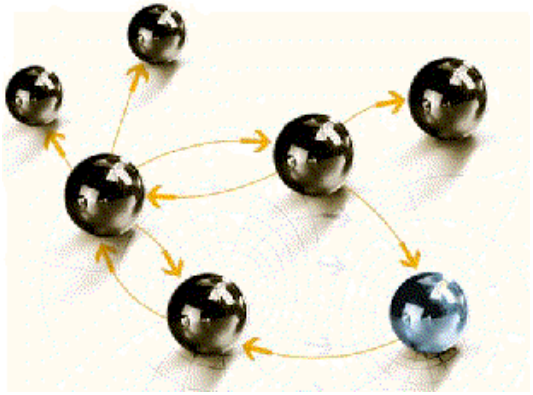


COBOL

COBOL ≡ BCOOL

Teil 2: Strukturen, Unterprogramme und Compiler

cps4it
consulting, projektmanagement und seminare für die informationstechnologie
Ralf Seidler, Stromberger Straße 36A, 55411 Bingen
Fon: +49-6721-992611, Fax: +49-6721-992613, Mail: ralf.seidler@cps4it.de
Internet: http://www.cps4it.de

Inhaltsverzeichnis

Ziele des Seminars

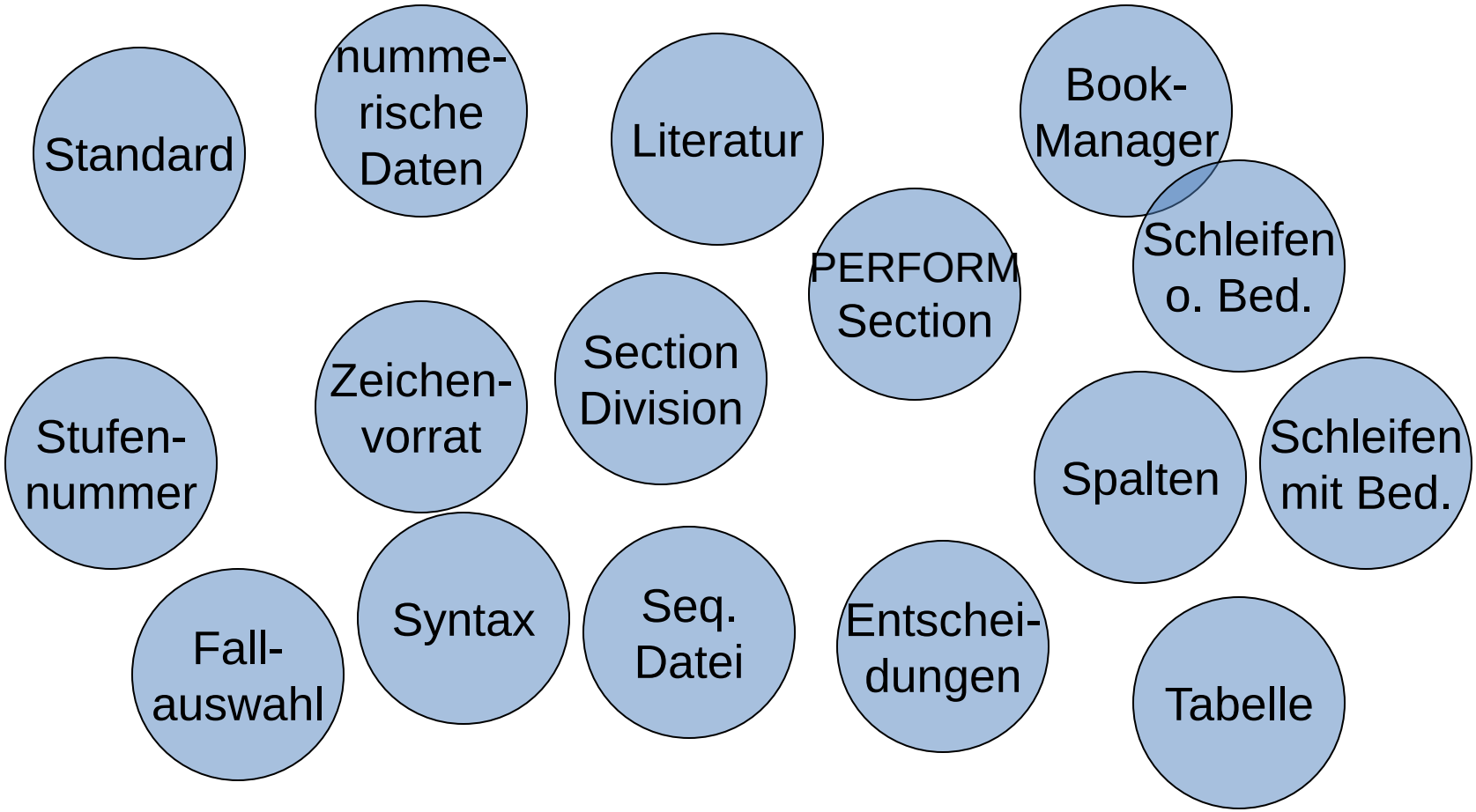
- Sprache COBOL kennen lernen
- Syntax von COBOL beherrschen – fast alles ;-)
- Praxisbeispiele kennen lernen
- üben … üben … üben
- Besonderheiten

Inhalt

- Einführung: Rückblick auf Kurs Grundlagen Teil 1
- Datendefinition
- Strukturen
- Zeichenketten
- Compiler
- Unterprogramme
- Testhilfen und Performance
- Neuerungen aus den letzten (20) Jahren
- Fragen und Antworten

Einführung

bekannte Begriffe



Einführung

Section – ohne Punkte / ohne Paragraphen kodieren

- PERFORM VORLAUF
- . . .
- VORLAUF SECTION.
- [VORLAUF-01.]
  - anweisung-1
  - anweisung-2
  - CONTINUE.
- [VORLAUF-EX.
  - EXIT.]

- PERFORM VORLAUF
- . . .
- VORLAUF SECTION.
  - anweisung-1
  - anweisung-2
  - CONTINUE.

sinnvoll (nur) wegen Xpediter: „GO PARA“

Dateiverarbeitung – 1

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT  AMT-EINGABE ASSIGN TO DD01.
    SELECT  AMT-AUSGABE ASSIGN TO DD02.
```

Einführung

Dateiverarbeitung – 2

```
 DATA DIVISION.
 FILE SECTION.
***
 FD   AMT-EINGABE
      BLOCK  CONTAINS  0  RECORDS
      RECORDING  MODE  IS  F.
 01   AMT-EIN-SATZ          PIC  X(80).
```

Dateiverarbeitung – 3

```
***
 FD   AMT-AUSGABE
      BLOCK  CONTAINS  0  RECORDS
      RECORDING  MODE  IS  F.
 01   AMT-AUS-SATZ          PIC  X(80).
/*************************
 WORKING-STORAGE SECTION.
 01   EINGABE-SATZ.
      02  E-ZEILE  PIC  X(200) OCCURS 200.
```

Einführung

Dateiverarbeitung – 4

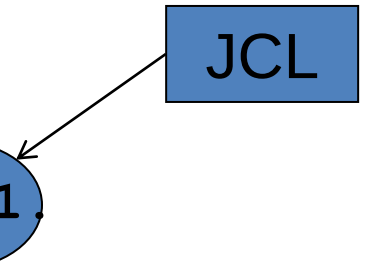
```
 PROCEDURE DIVISION.
***
      OPEN INPUT   AMT-EINGABE
      OPEN OUTPUT  AMT-AUSGABE
      READ AMT-EINGABE
           INTO EINGABE-ZEILE(1)
      MOVE EINGABE-ZEILE(1)
           TO   AMT-AUS-SATZ
      WRITE AMT-AUS-SATZ
      GOBACK.
```

Einführung

Dateiverarbeitung – Zusammenfassung

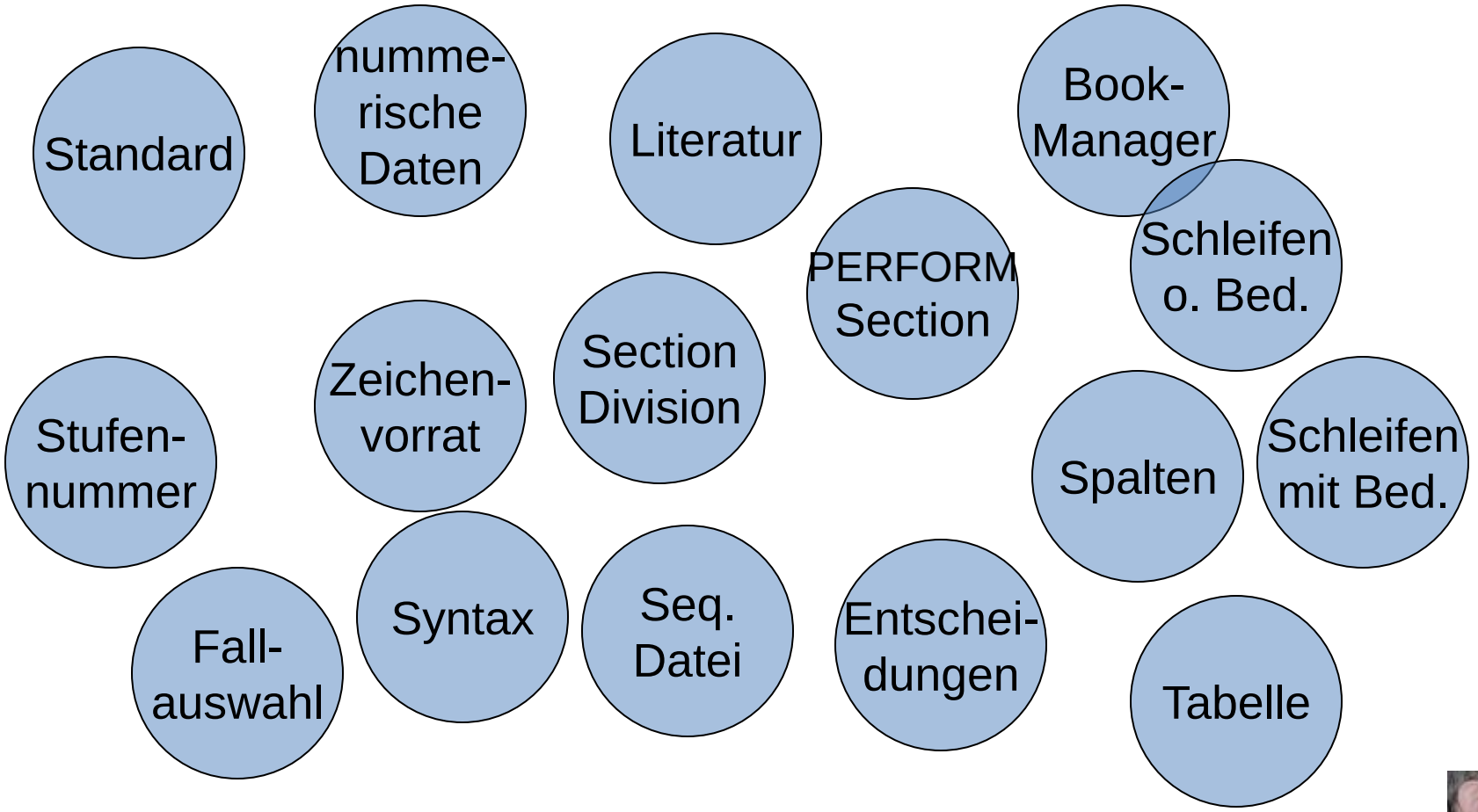
```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT AMT-EINGABE ASSIGN TO DD01.
DATA DIVISION.
FILE SECTION.
FD  AMT-EINGABE.
01  AMT-EIN-SATZ          PIC  X(80).
PROCEDURE DIVISION.
     OPEN INPUT AMT-EINGABE
     READ AMT-EINGABE

aber:WRITE AMT-AUS-SATZ !!
```



Einführung

bekannte Begriffe



Einführung

Dateien für Übungen

- uid.KURSCOB.COBOL
- uid.KURSCOB.JCL

- SCLM – eigenes Projekt

- 1 Programmname in SCLM wählen
- Ergebnis nach uid.KURSCOB.COBOL kopieren

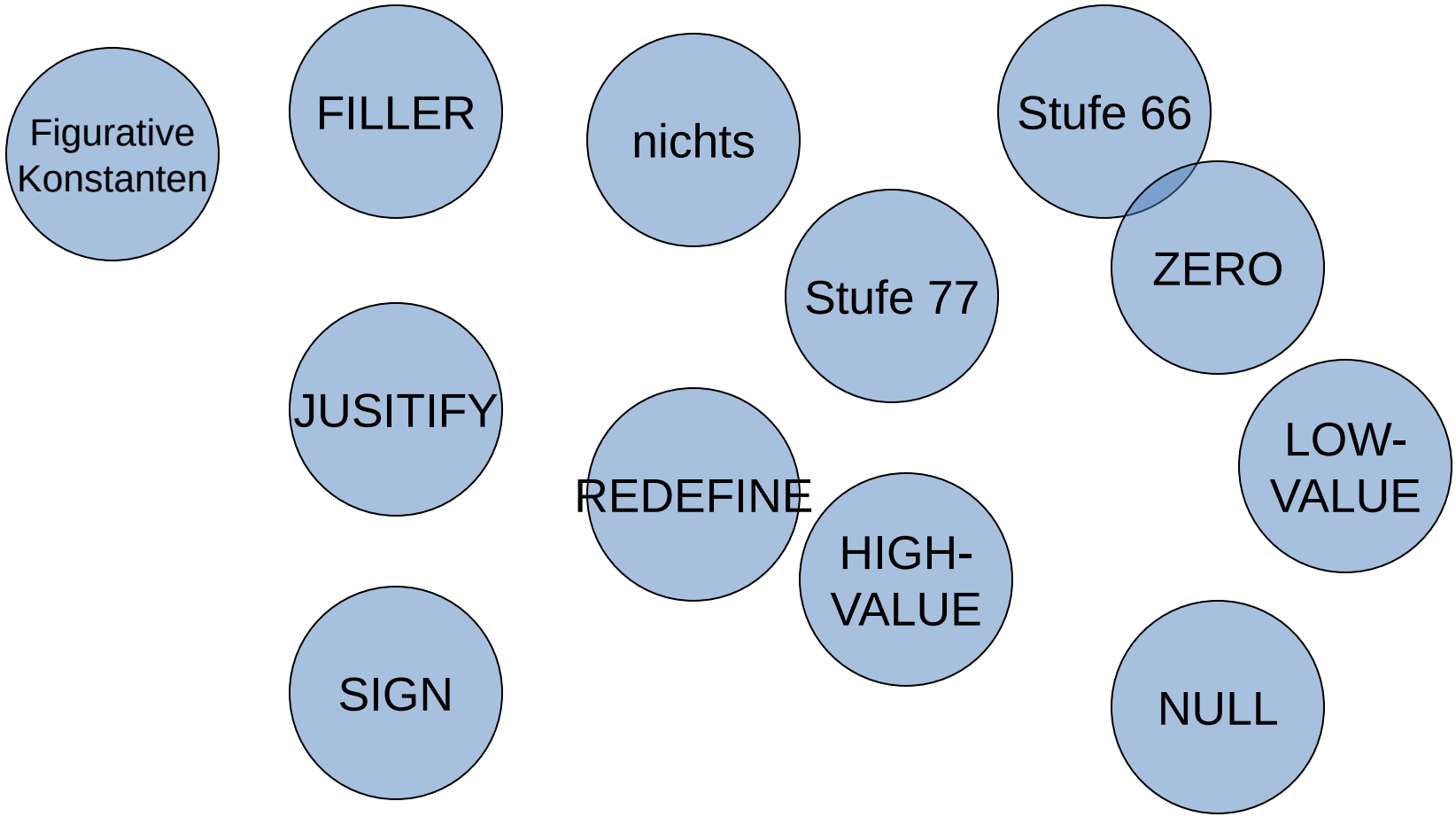
- uid-ref.KURSCOB.TEXT

Inhalt

- Einführung: Rückblick auf Kurs Grundlagen Teil 1
- Datendefinition
- Strukturen
- Zeichenketten
- Compiler
- Unterprogramme
- Testhilfen und Performance
- Neuerungen aus den letzten (20) Jahren
- Fragen und Antworten

Datendefinition

Begriffe



Datendefinition

Init in WS – Figurative Konstanten – 1

- Syntaxbeispiele
  - 01 FELD-1-n PIC 999 VALUE ZERO.
  - 01 FELD-2-n PIC 999 VALUE ALL ZEROES.
  - 01 FELD-3-n PIC 999 VALUE HIGH-VALUE.
  - 01 FELD-4-n PIC 999 VALUE LOW-VALUES.
  - 01 FELD-5-n PIC 999 VALUE NULL.
  - 01 FELD-6-n PIC 999 VALUE ALL NULLS.

Datendefinition

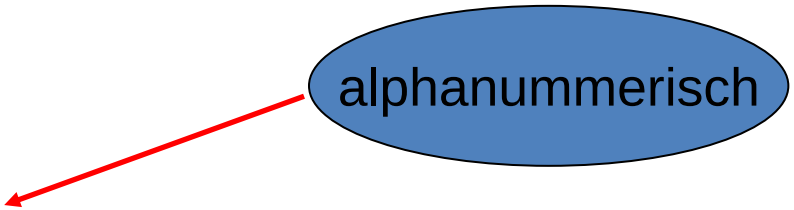
Init in WS – Figurative Konstanten – 2

- Syntaxbeispiele
  - 01 FELD-1-x PIC XXX VALUE ZERO.
  - 01 FELD-2-x PIC XXX VALUE ALL SPACE.
  - 01 FELD-3-x PIC XXX VALUE HIGH-VALUE.
  - 01 FELD-4-x PIC XXX VALUE LOW-VALUE.
  - 01 FELD-5-x PIC XXX VALUE QUOTE.
  - 01 FELD-6-x PIC XXX VALUE NULLS.

Datendefinition

Init in WS – FILLER

- Der FILLER füllt einen Teil einer Struktur mit “nichts”.
- Syntaxbeispiel
  - 01 STRUKTUR.
    - 05 FELD-1 PIC X(25).
    - 05 FELD-2 PIC S9(09) PACKED-DECIMAL.
    - 05 FILLER PIC X(10).
    - 05 FELD-3 PIC S9(04) BINARY.
    - 05 PIC X(10).
    - 05 FELD-4 PIC S9(08) BINARY.



Datendefinition

„obsolete“ ím Standard

- Stufennummer 77
  - gehört nicht mehr zum Standard und wird in einem der nächsten COBOL Releases wegfallen
- RENAMES
  - Die RENAMES-Klausel war/ist dazu da, um eine Struktur mit anderen Namen als eine Kopie anzusprechen. In dem Standard ist diese Klausel nicht mehr berücksichtigt und wird in einem der nächsten Compiler nicht mehr unterstützt.
  - RENAMES -> Stufennummer 66
  - Andere Lösung: REDEFINES

Datendefinition

REDEFINES

- Definition
  - Die REDEFINES-Klausel beschreibt ein vorher definiertes Feld auf dessen Speicher mit einem anderen Namen.
- Format
  - st name-1 REDEFINES name-2 PIC was.

Datendefinition

REDEFINES – Beispiele

```
*
 01  N6721-CHKP-FREQUENZ    PIC  X(06).
 01  N6721-CHKP-FREQ-N REDEFINES
     N6721-CHKP-FREQUENZ    PIC  9(06).
*****************************************************
*
 05 N6721-INT-BST-TABELLE      PIC X(18).
 05 N6721-INT-BST-TB REDEFINES N6721-INT-BST-TABELLE.
    10 N6721-BST-PGMNAME       PIC X(06) OCCURS 3.
          88 N6721-BST-PGMNAME-KK     VALUE 'N6721K'.
          88 N6721-BST-PGMNAME-RW     VALUE 'N6721R'.
          88 N6721-BST-PGMNAME-LEER   VALUE  SPACE.
```

Datendefinition

weitere Angaben bei Datendefinition – 1

- SIGN
  - Angabe der Position eines Vorzeichens, sowie die Bestimmung, ob dieses einen eigenen Speicherplatz belegen soll.
- SYNC | SYNCRONIZED
  - Gibt an, dass binär definierte Datenfelder im Speicher auf Doppelwortgrenze ausgerichtet werden.

Datendefinition

weitere Angaben bei Datendefinition – 2

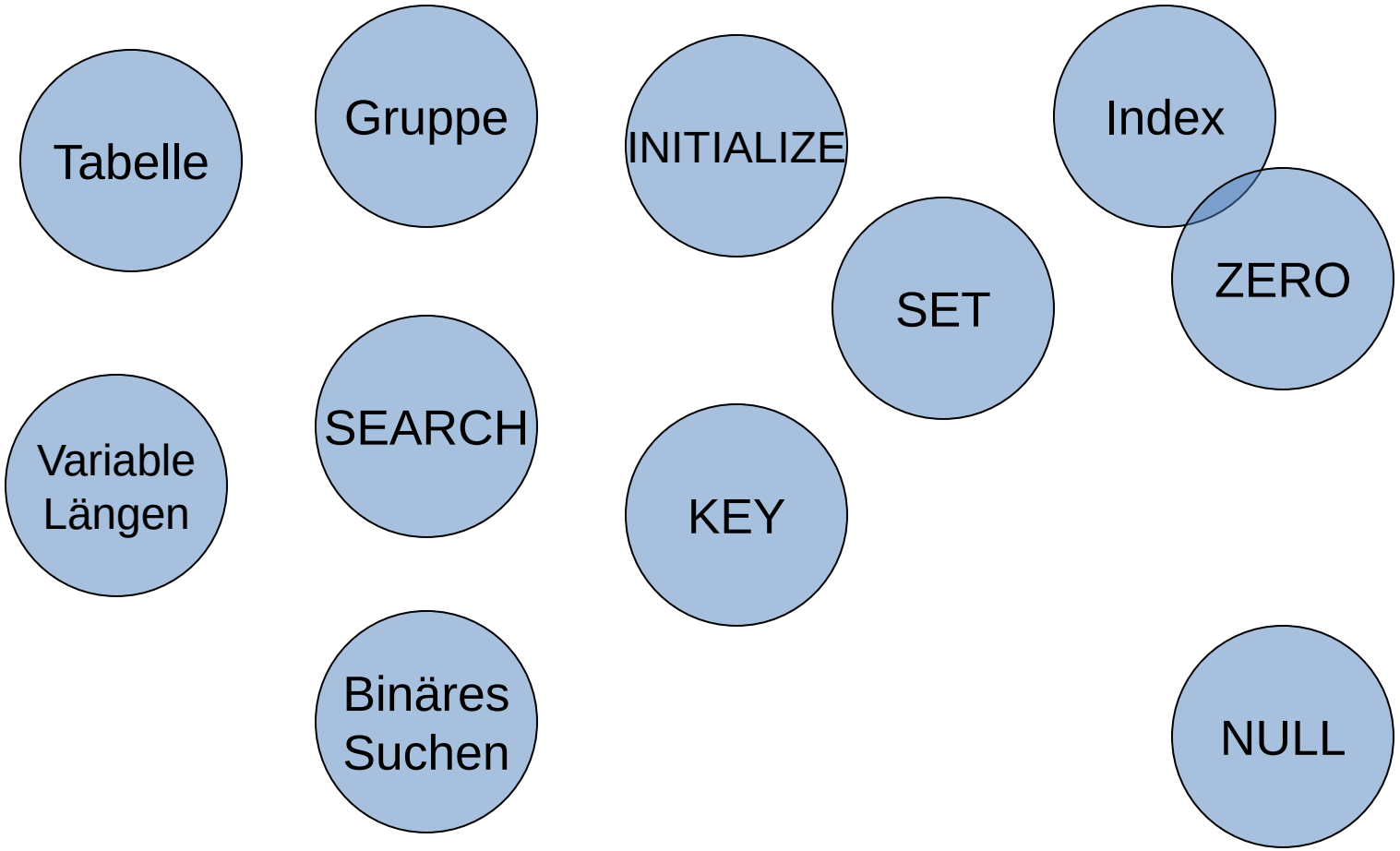
- BLANK WHEN ZERO
  - Ersetzen des Inhaltes durch Leerzeichen
  - nur bei nummerischen Feldern
- JUSTIFIED
  - für rechtsbündige Speicherung
  - nur bei alphanummerischen Feldern
- alphabetische Felder
  - PIC A

Inhalt

- Einführung: Rückblick auf Kurs Grundlagen Teil 1
- Datendefinition
- Strukturen
- Zeichenketten
- Compiler
- Unterprogramme
- Testhilfen und Performance
- Neuerungen aus den letzten (20) Jahren
- Fragen und Antworten

Strukturen

Begriffe



Strukturen

allgemeines

- Tabelle
  - Diese Sonderform einer Struktur wurde schon behandelt.
  - Siehe Kurs Grundlagen
- COPY
  - Es ist sinnvoll, allgemein gültige Strukturen separat zur Verfügung zu stellen. Der COBOL-Befehl COPY zieht aus einer (im Normalfall separaten) Bibliothek die entsprechende Struktur an.
  - Beispiele siehe SCLM

Strukturen

Beispiel

```
*
 01   GRUPPE.
      05 DATUM.
         10 JAHR              PIC  9(04).
         10 MONAT             PIC  9(02).
         10 TAG               PIC  9(02).
      05 BESTELL-MENGE        PIC  9(05) COMP-3.
```

Strukturen

Übung(en) – Ü 2.1 – COP1201

- Für jeden Kunden einer Bank sei ein Kundenstammsatz gespeichert, der die folgenden Felder zum Inhalt haben soll:

Nachname	alphanumerisch	20 Zeichen
Vorname	alphanumerisch	20 Zeichen
Kundennummer-1	nummerisch	4 Ziffern
Kundennummer-2	nummerisch	2 Ziffern
Kundennummer-3	alphanumerisch	1 Zeichen
Kundennummer-4	nummerisch	3 Ziffern
Kunde seit	nummerisch	8 Ziffern
Postleitzahl	alphanumerisch	10 Zeichen
Ort	alphanumerisch	20 Zeichen
Straße	alphanumerisch	20 Zeichen
Hausnummer	alphanumerisch	5 Zeichen

Strukturen

INITIALIZE

- Für die Initialisierung einer Struktur bietet sich der Befehl INITIALIZE an.
- Syntax
  - INITIALIZE struktur
- Ergebnis
  - jedes einzelne Feld wird auf Initialwert gesetzt.
    - alphanumerische Felder auf Space
    - nummerische Felder auf Null
  - die Felder werden “sauber” initialisiert
  - *alle* Felder werden “sauber” initialisiert (OCCURS!)
  - FILLER bleibt unberücksichtigt !!!

Strukturen

INITIALIZE mit REPLACING

- Syntax

```
INITIALIZE struktur
     REPLACING etwas BY {literal | variable}
```

  - „etwas“ kann sein
    - ALPHABETIC
    - ALPHANUMERIC
    - NUMERIC
    - und Sonderformen
- detaillierte Syntax und Beispiele siehe COBOL-Bücher

Strukturen

INDEX

- Tabellen können (schneller) mit einem (Maschinen-) Index angesprochen werden.
- Syntaxbeispiel:

```
*
 01    STAMMSATZ.
       05 UMSATZ-TAG OCCURS 366 INDEXED BY UMS-IND
                     PIC 9(4)V99.
```

Strukturen

INDEX mit Key

- Tabellen können, wenn sie auf- oder absteigend sortiert sind, einen Key enthalten.
- Syntaxbeispiel

```
*
 01    PERSONAL-TABELLE.
       05 PERS-EINTRAG OCCURS 1000
                       ASCENDING KEY IS PERS-NR
                       INDEXED BY PERS-IND.
          10 PERS-NR                 PIC  9(06).
          10 PERS-NAME               PIC  X(20).
```

Strukturen

arbeiten mit dem INDEX

- Der Index hat besondere Befehle für die Modifikation.

- SET index-1 TO index-2
- SET index-1 TO variable-2
- SET variable-1 TO index-2
- SET index-1 TO literal
- SET index-1 UP BY {variable | literal}
- SET index-1 DOWN BY {variable | literal}

Inhalt des Index im Dump – Beispiel 1

- 01 TAB OCCURS 20 INDEXED BY IND
  PIC X(88).

- Anzeige in DUMP: B0

- B0 = 176 (dezimal)
- Berechnung:
  (176 / 88) + 1 = 3
  Der Index hat also den Wert 3!

Strukturen

Inhalt des Index im Dump – Beispiel 2

- 01 TABX OCCURS 20 INDEXED BY INDX
  PIC X(27).

- Anzeige in DUMP: 6C

- 6C = 108 (dezimal)
- Berechnung:
  (108 / 27) + 1 = 5
  Der Index hat also den Wert 5!

Strukturen

Übung(en) – Ü 2.2 – PGM1201

- Initialisieren Sie den vorher definierten Kundenstammsatz
  - einmal mit MOVE ...
  - einmal mit INITIALIZE
- Testen Sie die Laufzeit des Programms, indem Sie beide Fälle bis zu 10.000.000 Mal laufen lassen. Welcher Code ist schneller?

Strukturen

Übung(en) – Ü 2.3 – COPY1202

- Der Kundenstammsatz soll um Kontonummern erweitert werden. Die Kontonummer ist 9 Stellen nummerisch. Der Kunde kann bis zu 10 Kontonummern haben, wobei er für jede Kontonummer eine separate Anschrift haben kann.
- Definieren Sie die neue Struktur, initialisieren Sie diese mit MOVEs und INITIALIZE und vergleichen Sie die Laufzeiten.
- Nutzen Sie den Maschinen-Index.

Strukturen

Übung(en) – Ü 2.4 – PGM1202

- Definieren Sie eine Tabelle mit 52 Wochen und 7 Tagen. Die Felder auf der untersten Ebene sollen als Werte den Wochentag, den laufenden Tag der Woche und den laufenden Tag des Jahres aufnehmen.
- Initialisieren Sie die Tabelle geschickt für das Jahr 2019.
- Nutzen Sie den Maschinen-Index.

Strukturen

Übung(en) – Ü 2.5 – PGM1203

- Für die vorher genannten Tabellen haben Sie den (Maschinen)Index benutzt. Arbeiten Sie jetzt mit Subscripten. Wählen Sie unterschiedliche Definitionen dafür. Lassen Sie die Verarbeitungen bis zu 10.000.000 Mal laufen und vergleichen Sie die Laufzeiten.

Strukturen / Tabellen

Suchen in einer Tabelle – SEARCH

- Funktion
- Suche nach einem bestimmten Element in einer Tabelle.

- Syntax

```
SEARCH struktur [VARYING index]
       [AT END     anweisung]
        WHEN   bedingung-1 anweisung-1
END-SEARCH
```

Strukturen / Tabellen

Suchen in einer Tabelle – SEARCH – Regeln

- struktur muss mit INDEX BY definiert sein.
- Beginn des Suchens ab Wert von Index
- Nach anweisung-1 wird mit Statement nach SEARCH weiter gearbeitet.
- Ist der Index für struktur definiert, wird dieser genommen, sonst der nächste in der Hierachie.
- Beliebig viele WHEN-Angaben sind erlaubt.
- Wird der Begriff nicht gefunden, zieht AT END.
- Es kann nur eine Dimension durchsucht werden.

Strukturen / Tabellen

binäres Suchen – SEARCH

- Wenn die Tabelle auf- oder absteigend sortiert ist, kann diese binär durchsucht werden.

- Syntax

```
SEARCH ALL struktur [VARYING index]
       [AT END    anweisung]
        WHEN   bedingung-1
       [AND    bedingung-2] []...
        anweisung-2]
END-SEARCH
```

Strukturen / Tabellen

Übung(en) – Ü 2.6 – PGM1204

- Suchen Sie in der Wochen/Tage-Tabelle den Tag des Jahres, den Sie über SYSIN einlesen.
  - Nutzen Sie den SEARCH-Befehl.
  - Nutzen Sie binäres Suchen.
  - Kodieren Sie den Such-Algorithmus per Hand.
  - Was ist der schnellste Weg?

Strukturen / Tabellen

Tabellen mit variabler Länge

- Tabellen können variabel definiert werden, wenn die genaue Anzahl der Plätze nicht bekannt ist.
- Syntaxbeispiel

```
*
 01   BEWEGUNGS-TABELLE.
      05 BEWEGUNGS-SATZ    PIC  X(80)
         OCCURS 1 TO 500 DEPENDING ON SATZ-ANZAHL
         INDEXED BY BEW-IND.
 01 SATZ-ANZAHL            PIC S9(04) COMP VALUE ZERO.
    88 ANZAHL-MAX                        VALUE 500.
 01 DATEI-ENDE-SCHALTER    PIC  X(01) VALUE 'N'.
    88 DATEI-ENDE                     VALUE 'Y'.
```

Strukturen / Tabellen

Tabellen mit variabler Länge – Code

```
READ EINGABE
        AT END SET DATEI-ENDE       TO TRUE
END-READ
PERFORM WITH TEST BEFORE
        VARYING BEW-IND FROM 1 BY 1
        UNTIL  (DATEI-ENDE OR ANZAHL-MAX)
   MOVE EINGABE-SATZ  TO BEWEGUNGS-SATZ(BEW-IND)
   ADD  1             TO SATZ-ANZAHL
   READ EINGABE
        AT END SET DATEI-ENDE       TO TRUE
   END-READ
END-PERFORM
```

Strukturen / Tabellen

Übung(en) – Ü 2.7 – PGM1205

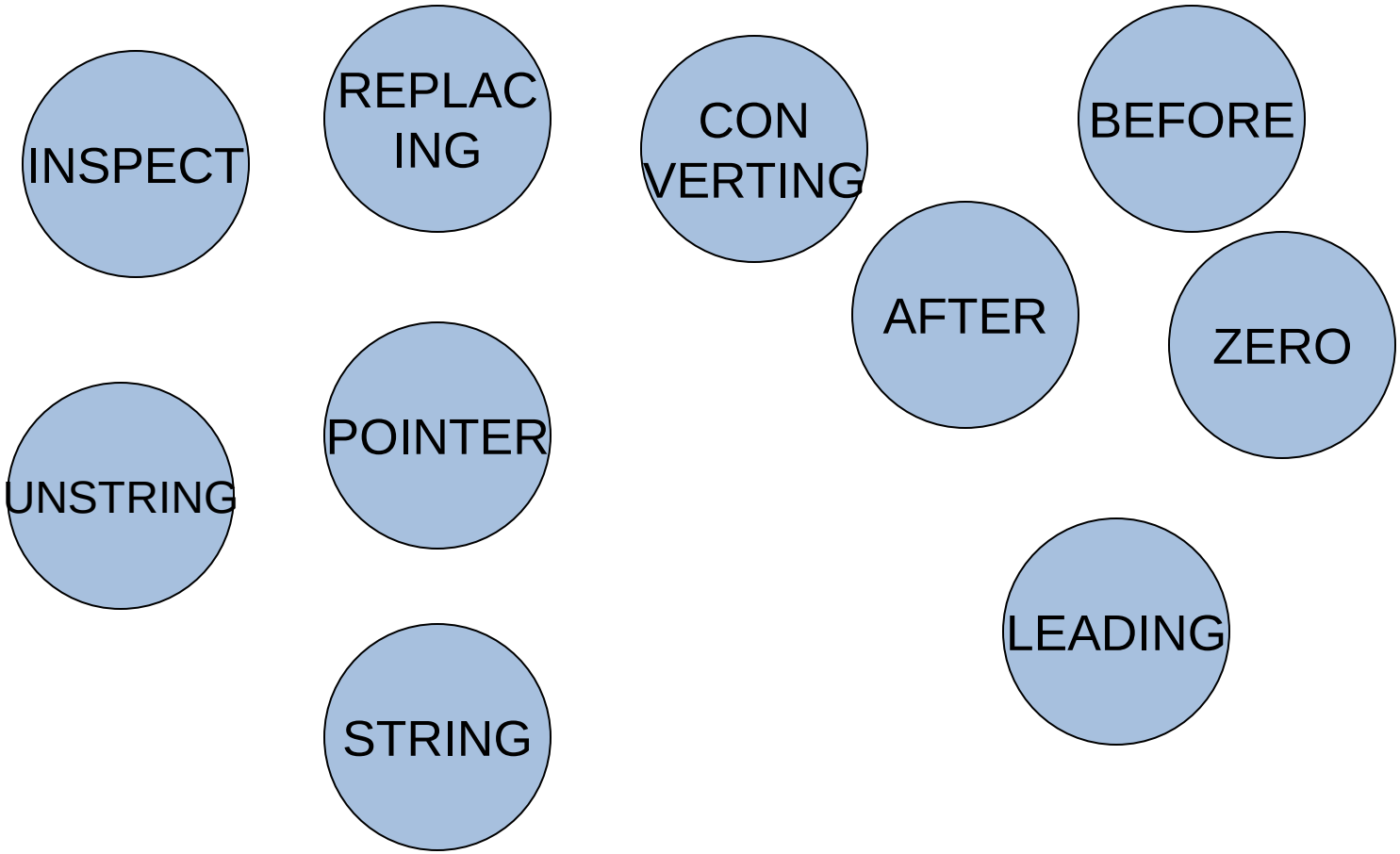
- Legen Sie eine Section an, in der Sie die Verarbeitung mit variabler Satz-Anzahl kodieren.
  - Nutzen Sie die Datei der Stadtverwaltung.
  - Achten Sie auf die richtige Definition der Zähler.
  - Kann die Logik anders als im Beispiel kodiert werden?
  - Welche Werte stehen nach der Schleife in den Zählern? Gibt es Unterschiede je nach Art der Kodierung?

Inhalt

- Einführung: Rückblick auf Kurs Grundlagen Teil 1
- Datendefinition
- Strukturen
- Zeichenketten
- Compiler
- Unterprogramme
- Testhilfen und Performance
- Neuerungen aus den letzten (20) Jahren
- Fragen und Antworten

Zeichenketten

Begriffe



Zeichenketten

INSPECT – 1

- Funktion
  - Zählen von Zeichen
  - Ersetzen von Zeichen
- Syntax – Zählen

```
INSPECT feld-1 TALLYING feld-2
        FOR {ALL | LEADING | CHARACTERS}
            {feld-3 | lit-3}
       {BEFORE | AFTER} INITIAL {feld-4 | lit-4}
```

Zeichenketten

INSPECT – 2

- Syntax – Ersetzen – 1

```
INSPECT feld-1 REPLACING CHARACTERS
        BY {feld-2 | lit-2}
       {BEFORE | AFTER} INITIAL {feld-3 | lit-3}

INSPECT feld-1 REPLACING
       {ALL | LEADING | FIRST} {feld-2 | lit-2}
        BY {feld-3 | lit-3}
       {BEFORE | AFTER} INITIAL {feld-4 | lit-4}
```

Zeichenketten

INSPECT – 3

- Beispiel – Ersetzen – 1

```
INSPECT EING-MENGE REPLACING ALL SPACE BY ZERO
        AFTER INITIAL '%'
        FIRST ',' BY '.'
Ergebnis:
Vorher    >%   235,34 <
Nachher   >%000235.340<
```

Zeichenketten

INSPECT – 4

- Ersetzen - 2
  - Funktion:
    Jedes Zeichen einer Zeichenkette wird einem bestimmten anderen Zeichen zugeordnet.
- Syntax – Ersetzen – 2

```
INSPECT feld-1 CONVERTING {feld-2 | lit-2}
                        TO {feld-3 | lit-3}
       {BEFORE | AFTER} INITIAL {feld-4 | lit-4}
```

Zeichenketten

INSPECT – 5

- Beispiel – Ersetzen – 2

```
INSPECT feld-A CONVERTING '0123456789'
                       TO 'SAFETY-NOW'
```

- Ergebnis:

```
Vorher:  >35742<
Nachher: >EYNTF<
```

Zeichenketten

Übung(en) – Ü 3.1 – PGM1301 (Basis: PGM1204)

- Legen Sie eine Section an, in der Sie verschiedene Formen des INSPECT ausprobieren. Beispiele:
  - Wochentage (MO, DI) in (01, 02) verändern.
  - Monatsnamen in 01, 02 verändern. Geht das?
  - Datei einlesen. Alle blanks werden in low-value verändert. Gleichzeitig sollen alle low-values in blanks umgesetzt werden. Funktion 2 Mal laufen lassen. Ist der alte Inhalt wieder da?
  - Suchen Stelle, in der der Monat Mai in der Wochen/Monatstabelle auftritt.

Zeichenketten

Verketten – 1

- Syntax

```
STRING {feld-1|lit-1} . . .
       [DELIMITED BY {feld-2|lit-2|SIZE} . . .]
       INTO feld-3 [WITH POINTER feld-4]
      [    ON OVERFLOW anweisung-5]
      [NOT ON OVERFLOW anweisung-6]
[END-STRING]
```

Zeichenketten

Verketten – 2

- Regeln – 1
  - Alle Literale sind nicht nummerisch.
  - feld-1, lit-1 . . . sind Sendefelder.
  - feld-3 ist Empfangsfeld. Es darf nicht druckaufbereitet und nicht mit JUSTIFY definiert sein.
  - Bei DELIMITED BY SIZE wird Sendefeld vollständig übertragen.
  - Feld-4 ist die Anfangsposition für die Daten im Empfangsfeld. Es muss ganzzahlig und groß genug sein, um die Länge des Empfangsfeldes plus 1 aufzunehmen.

Zeichenketten

Verketten – 3

- Regeln – 2
  - Bei der Angabe des Pointer-Zusatzes muss der Anfangswert von feld-4 explizit gesetzt werden. 1<= feld-4 <= len(feld-3)
  - Ohne Pointer-Angabe ist der implizite Zeiger auf 1 gesetzt.
  - Der Wert beträgt des Zeigers beträgt am Ende len(übertragene Zeichen) + 1
  - STRING ist beendet, wenn Zielfeld gefüllt oder alle Sendefelder bearbeitet sind.

Zeichenketten

Verketten – 4

- Regeln – 3
  - Nur der Teil des Empfangsfeldes ist verändert, in den Zeichen übertragen worden sind.
  - Ist OVERFLOW angegeben und werden die Grenzen durch feld-3 oder Zeiger überschritten, wird die Bearbeitung beendet und die entsprechende Anweisung ausgeführt.

Zeichenketten

Verketten – Beispiel

```
NAME: cps4it, Ralf Seidler
PLZ : 55411
ORT : Bingen
TEL : +49-06721-992611
MOVE            1 TO  POSITION
STRING      NAME DELIMITED BY ‘,’
       ‘*’, PLZ  DELIMITED BY SIZE
       ‘*’, ORT  DELIMITED BY ‘ ‘
       ‘*Tel.’, TEL DELIMITED BY ‘ ‘ ‘*’
     INTO Z-FELD WITH POINTER POSITION
END-STRING
```

```
cps4it*55411*Bingen*Tel.+49-06721-992611*
```

Zeichenketten

Trennen (Entketten) – 1

- Syntax

```
UNSTRING feld-1
        [DELIMITED BY [ALL] {feld-2|lit-2}
                   [OR [ALL] {feld-3|lit-3}] ...]
        INTO feld-4 [DELIMITER IN feld-5]
                    [COUNT     IN feld-6]
       [WITH POINTER feld-7] [TALLYING IN  feld-8]
       [    ON OVERFLOW anweisung-A]
       [NOT ON OVERFLOW anweisung-B]
[END-UNSTRING]
```

Zeichenketten

Trennen (Entketten) – 2

- Regeln – 1
  - Alle Literale sind nicht nummerisch.
  - feld-1 ist das Sendefeld.
  - feld-4 ist Empfangsfeld.
  - feld-2 bzw. lit-2 sind die Begrenzer.
  - feld-5 ist das Begrenzerempfangsfeld.
  - feld-6 ist das Zählerfeld für die Datenübertragung; es enthält die Anzahl der Zeichen, die innerhalb von feld-1 bis zum Begrenzer gefunden und übertragen wurden.

Zeichenketten

Trennen (Entketten) – 3

- Regeln – 2
  - feld-7 ist die Anfangsposition für die Datenübertragung im Sendefeld.
  - feld-8 ist das Zählerfeld für die Anzahl der Daten empfangenden Felder.
  - Die OVERFLOW Bedingung zieht,
    - wenn alle Daten empfangenden Felder verarbeitet sind und feld-1 enthält noch ungeprüfte Zeichen oder
    - wenn feld-7 die Grenzen von feld-4 überschreitet.

Zeichenketten

Trennen (Entketten) – Beispiel

```
Z-FELD: cps4it*55411*Bingen*Tel.+49-06721-992611*

MOVE              ZERO TO  Z-ANZ-F, Z-LEN-TEL
UNSTRING Z-FELD DELIMITED BY '*' OR '*TEL.'
     INTO      NAME, PLZ, ORT, TEL
     COUNT     IN Z-LEN-TEL
     TALLYING IN Z-ANZ-F
END-UNSTRING
```

```
NAME: cps4it                          Z-LEN-TEL: 16
PLZ : 55411                           Z-ANZ-F  :  4
ORT : Bingen
TEL : +49-06721-992611
```

Zeichenketten

Übung(en) – Ü 3.2 – PGM1302

- Legen Sie eine Section an, in der Sie Ihre Adresse, getrennt durch einen Stern zusammensetzen.
- Legen Sie eine Section an, in der Sie die zusammen gesetzte Adresse wieder auseinander nehmen.

Zeichenketten

Übung(en) – Ü 3.3 – PGM1303

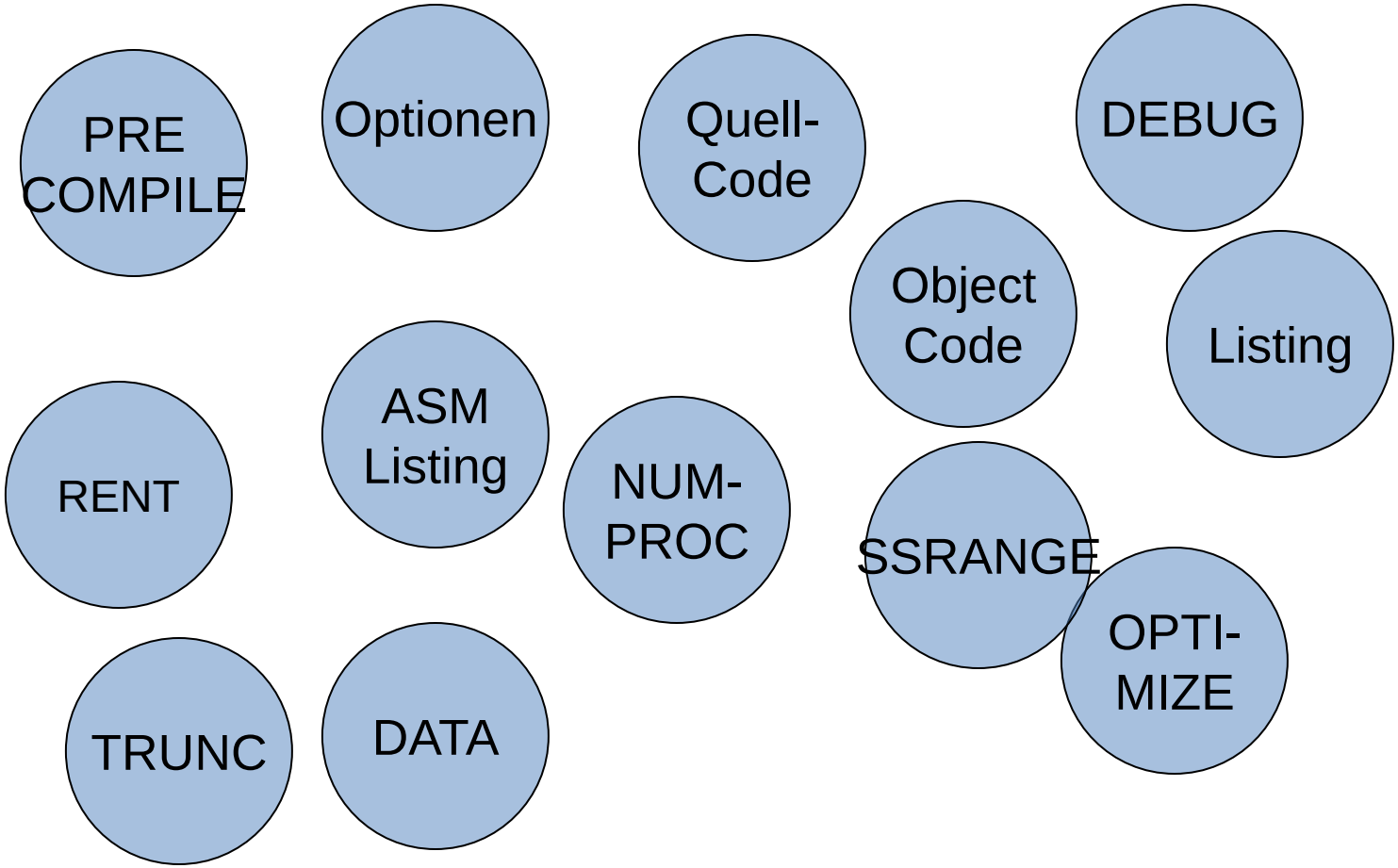
- Für eine bestimmte Anwendung müssen beginnend ab 1 bis zu einer Zahl x, die durch 4 teilbar sein muss, zwei Zeilen ausgegeben werden:
  - 1;2;5;6;9;10;13;14; etc.
  - 3;4;7;8;11;12;15;16 etc.
  - Wie groß darf x maximal sein, wenn die Zeilen kleiner als 80 sein müssen?
  - Wenn die Zeilen größer als 80 werden, geben Sie die Ausgabe mehrzeilig aus.

Inhalt

- Einführung: Rückblick auf Kurs Grundlagen Teil 1
- Datendefinition
- Strukturen
- Zeichenketten
- Compiler
- Unterprogramme
- Testhilfen und Performance
- Neuerungen aus den letzten (20) Jahren
- Fragen und Antworten

Compiler

Begriffe



Compiler

Precompiler / Proprocessor (auch Coprocessor)

- Innerhalb des Compilers wird vorab ein Step ausgeführt, der u.a. die Funktionen hat:
  - Auflösung COPY
  - Auflösung EXEC SQL
  - Auflösung EXEC DLI
  - Auflösung EXEC CICS
- Ziel ist es, dem Compiler reinen COBOL-Code zur Verfügung zu stellen.

Compile Options – Allgemeines

- Dem Compiler können verschiedene Optionen angegeben werden zur
  - Interpretation des Quellcodes
  - Behandlung des Datums zur Laufzeit
  - Ausgabe des Compile Listings
  - Art des Object Moduls
  - Kontrolle des Object Moduls
  - Behandlung virtueller Speicher
  - Debugging und Diagnose
  - sonstiges

Compiler

Compile Options – 1

- Interpretation Quellcode
  - ARITH(COMPAT|EXTENT) Firma:
  - QUOTE|APOST Firma:
  - CICS|NOCICS Firma:
  - SQL|NOSQL Firma:
  - etc.
- Interpretation Datum
  - YEARWINDOW(yyyy) Firma:

OPTIONS DER R+V SELBST EINFÜGEN ;-)

Compiler

Compile Options – 2

- Compiler Listing
  - LINECOUNT(nn) Firma:
  - NOLIST|LIST Firma:
  - NOMAP|MAP Firma:
  - NOOFFSET|OFFSET Firma:
  - NOSOURCE|SOURCE Firma:
  - XREF(SHORT|FULL) Firma:
- Generieren Object Modul
  - COMPILE|NOCOMPILE(W|E|S)

Compiler

Compile Options – 3

- Object Code Control
  - NOAWO|AWO Firma:
  - NODLL|DLL Firma:
  - NOFASTSRT|FASTSRT Firma:
  - NUMPROC(NOPFD|PFD) Firma:
  - ZONEDATA(NOPFD|PFD) Firma:
  - NOOPT|OPT(STD|FULL) Firma:
  - OPT(0|1|2|3) Firma:
  - TRUNC(STD|OPT|BIN) Firma:

Compiler

Compile Options – 4

- Virtueller Speicher
  - DATA(24|31) Firma:
  - NODYNAM|DYNAM) Firma:
  - NORENT|RENT Firma:
  - RMODE(AUTO|24|ANY) Firma:
- Diagnose und Testhilfen
  - NOFLAG|FLAG(I,I) Firma:
  - NOTEST|TEST(a,b,c) Firma:
  - NOSSRANGE|SSRANGE Firma:

Compiler

Übung(en) – Ü 4.1 – PGM0703

- Sehen Sie sich das Compile-Menü / die SCLM-Methoden näher an.
- Nehmen Sie ein eigenes Programm und compilieren (und testen) Sie es mit unterschiedlichen Optionen.
- Nehmen Sie ein Programm, das mit einem Subscript fehlerhaft arbeitet und testen Sie dieses mit SSRANGE.
  - Fehlermeldung beachten.

Compiler

Übung(en) – Ü 4.2 – PGM1401

- Testen Sie die maximalen Werte eines binären Feldes mit den Einstellungen TRUNC(STD) und TRUNC(BIN).
- Wandeln Sie Ihr Programm mit der Option LIST um. Schauen Sie sich das Ergebnis an.
- Wandeln Sie Ihr Programm mit der Option LIST und achten Sie auf die Unterschiede
  - NUMPROC(PFD)
  - NUMPROC(NOPFD)

Compiler

Compileliste

- Inhalte der Compileliste sind „lebensnotwendig“
  - Dump-Analyse
  - Messtools
- Speicherung der Compileliste im CCM-Tool* des Unternehmens wie
  - SCLM
  - Endevor
  - Panvalet
  - eigenes CCM-Tool

* Change- und Configurationmanagement

Compiler

Inhalte der Compileliste

- Compilerrelease
- Datum und Uhrzeit der Umwandlung
- Optionen
- Source
- Crossreferenzen Felder
- Crossreferenzen Sections / Paragraphen
- relative Speicheradressen der Felder
- relative Speicheradressen der Befehle (COBOL oder ASM)
- Informationen / Warnungen / Fehlermeldungen

Compiler

DUMP – was ist das?

- to dump:
  abladen; schütten; auskippen; fallen lassen;
  abziehen; lagern; stapeln; verklappen;
- the dump
  Abzug; Dump; Auflistung; Depot; Kaff; Dreckloch;
  Sauladen; Schutthaufen, Abfallhaufen;
- to dump s.b.
  jdm. abschieben, jdm. loswerden
- to dump memory
  Speicherinhalt anzeigen

Compiler

DUMP – Haltung bei einem Dump

Beispiel

Compiler

LE – Meldungen – Verursacher

- CEE CEL (aber könnte woanders hin zeigen)
- IGZ COBOL
- IBM PL1
- AFH FORTRAN
- EDC C/C++

- Details zu COBOL siehe zum Beispiel:
  z/OS V1R9.0 Language Environment Run-Time Messages,
  Kapitel 7.0 COBOL Run-Time Messages

Compiler

Meldungen – Aufbau und Typen von Meldungen (COBOL)

- IGZnnnnx mit x=
  - I Informational message
  - W Warning message
  - E Error message
  - S Severe error message
  - C Critical error message

cps4it

Meldungen – Beispiel 1 (COBOL)

```
IGZ0006S The reference to table ??? by verb number ???
         on line ??? addressed an area outside
         the region of the table.
```

Explanation: When the SSRANGE option is in effect, this message is issued to indicate that a fixed-length table has been subscripted in a way that exceeds the defined size of the table, or, for variable-length tables, the maximum size of the table.

The range check was performed on the composite of the subscripts and resulted in an address outside the region of the table. For variable-length tables, the address is outside the region of the table defined when all OCCURS DEPENDING ON objects are at their maximum values; the ODO object's current value is not considered. The check was not performed on individual subscripts.

Programmer Response: Ensure that the value of literal subscripts and/or the value of variable subscripts as evaluated at run-time do not exceed the subscripted dimensions for subscripted data in the failing statement.

System Action: The application was terminated.

Symbolic Feedback Code: IGZ006

Meldungen – Beispiel 2 (COBOL)

```
IGZ0011C ??? was not a proper module for this system
         environment.
```

Explanation: A library subroutine that is system sensitive is inappropriate for the current system environment. For example, an OS environment specific module has been loaded under CICS. The likely causes are:

- Improper concatenation sequence of partitioned data sets that contain the subroutine library, either during run-time or during link-edit of the COBPAC.
- An attempt to use a function unsupported on the current system (for example, ACCEPT on CICS).

Programmer Response: Check for the conditions stated above, and modify the environment or the application as needed.

System Action: The application was terminated.

Symbolic Feedback Code: IGZ00B

Meldungen – Beispiel 3 (COBOL)

```
IGZ0100S Argument-1 for function ??? in program ??? at
         displacement ??? was less than or equal to -1.
```

Explanation: An illegal value was used for Argument-1.

Programmer Response: Ensure that argument-1 is greater than -1.

System Action: The application was terminated.

Symbolic Feedback Code: IGZ034

Meldungen – Beispiel 4 (COBOL)

```
IGZ0017S The open of DISPLAY or ACCEPT file with
         environment name ??? was unsuccessful.
```

Explanation: An error occurred while opening the DISPLAY/ACCEPT file.

Programmer Response: Check to make sure a ddname has been defined for the file.

System Action: The application was terminated.

Symbolic Feedback Code: IGZ00H

Compiler

Steuerblöcke – COBOL – Baselocator

- ~~BLW Working-Storage~~
- BLL Linkage Section
- BLF Files
- BLS Sort Items
- BLX external Data
- BLT Base Locator Table
- BLV Variable Located Data
- IX Indizes

- **Symbols used in LIST and MAP output Programming Guide**

Compiler

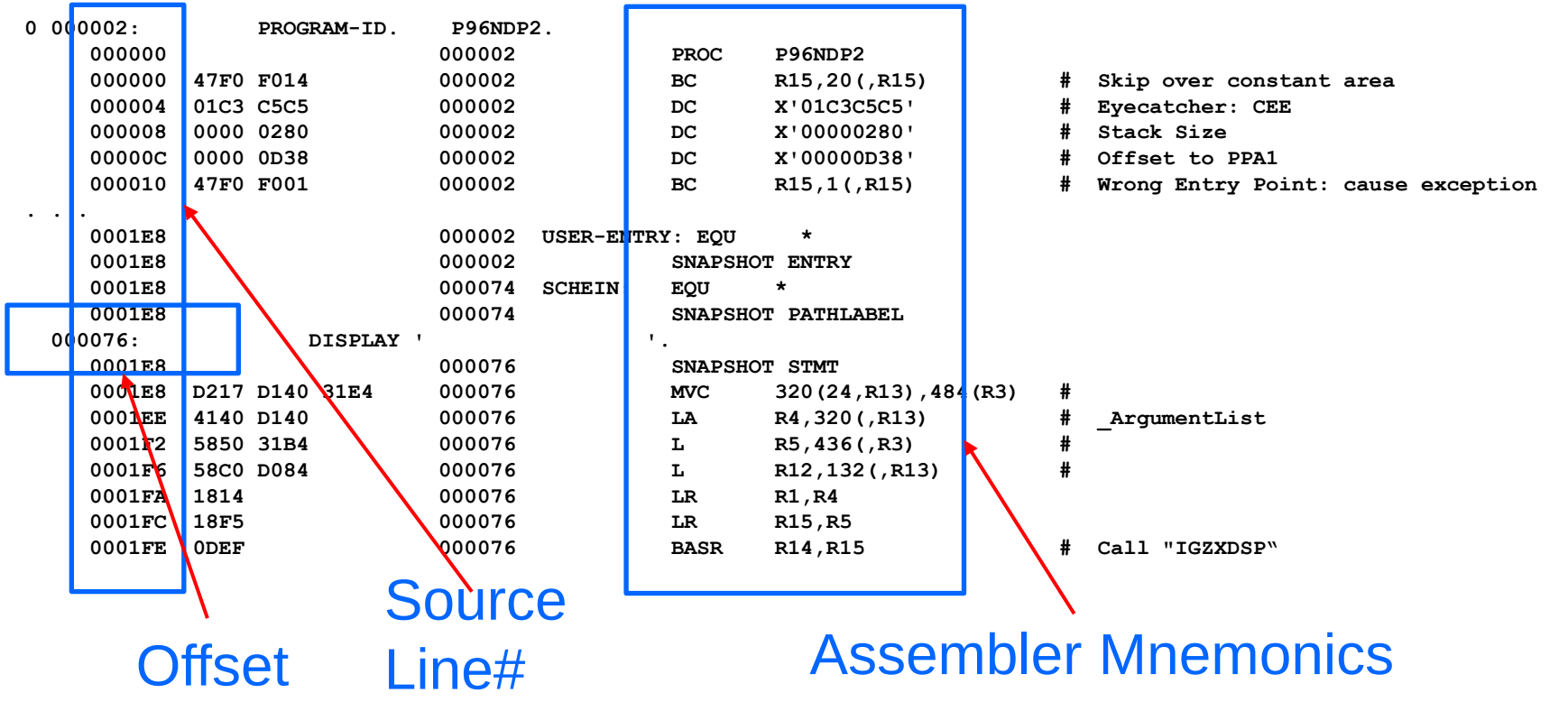
COBOL – Systembereiche in Auswahl

- PPA (programm prolog area)
  - reservierter Bereich für Standardinformationen
  - PPA 1, 2, 3, 4
- Constant Area (CGT)
  - Literale und Konstanten
- COBDSACB (COB V5+, TGT Ersatz)
  - Aber ohne Working Storage Adressen
- CLLE (cobol load list entry address)
  - für dynamisch aufgerufene programe
- FCB (file control block)
  - für alle Files

Compiler

COBOL – Compiler Listing – Assembly

- wichtige Optionen: NOTEST(DWARF),LIST,MAP
- Assembly: Anfang ist Offset 000000 (Init-Code)



Compiler

Beziehungen der Informationen in Dump zu Programm

- Abbruchadresse
- Feldadresse

Befehl

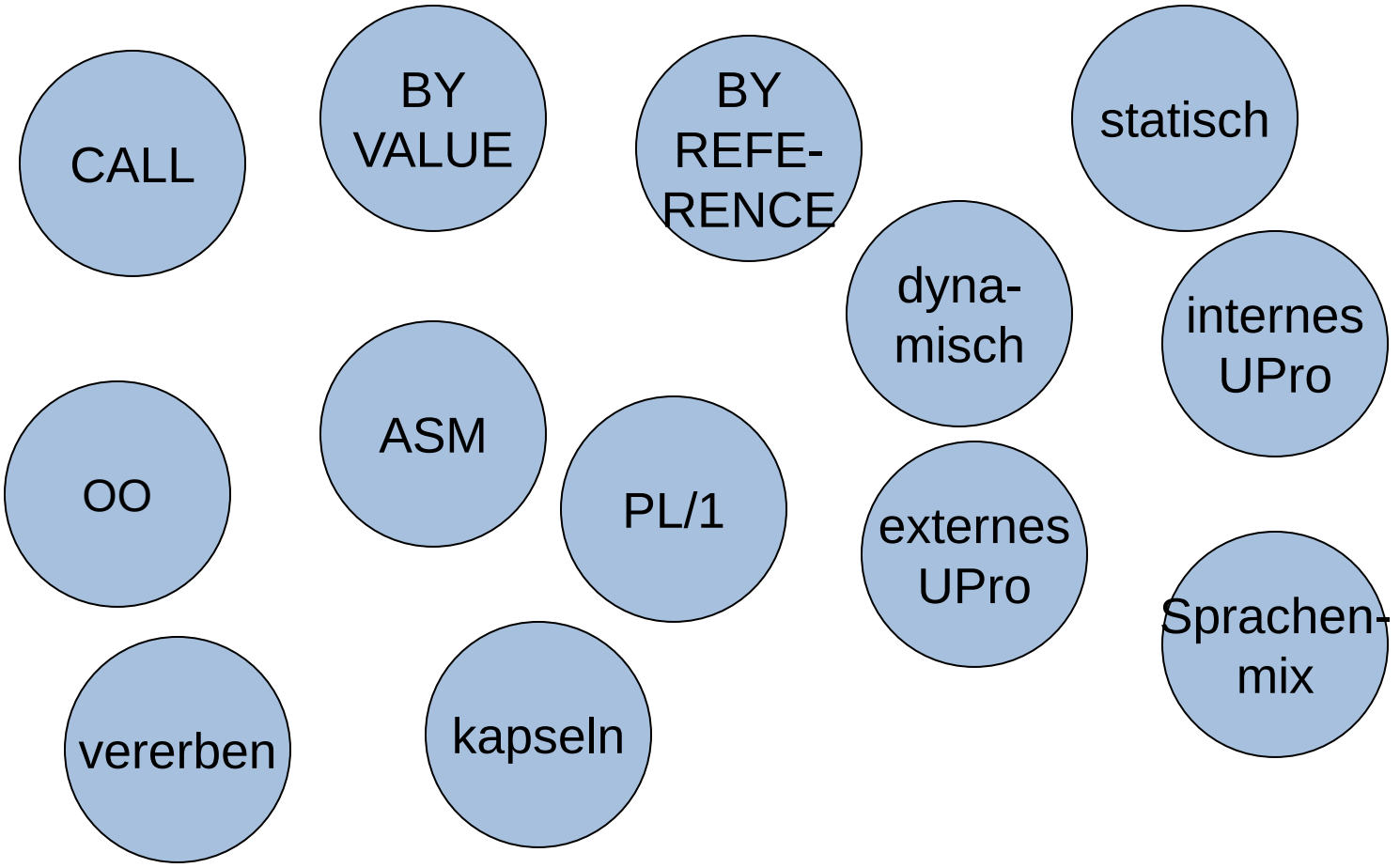
Feldname

Inhalt

- Einführung: Rückblick auf Kurs Grundlagen Teil 1
- Datendefinition
- Strukturen
- Zeichenketten
- Compiler
- Unterprogramme
- Testhilfen und Performance
- Neuerungen aus den letzten (20) Jahren
- Fragen und Antworten

Unterprogramme

Begriffe



Unterprogramme

warum?

- Modularisierung als Werkzeug der strukturierten Programmierung
- Kapselung von technischen und fachlichen Funktionen oder Bausteinen
- Umsetzung der üblichen Architekturregeln wie 3-tier-architecture
- Übersichtlichkeit
- mehrere Programmierer

Unterprogramme

Herausforderungen

- Modularisierung vernünftig aufbauen
- viele Programmierer benötigen qualifizierte Projektleitung
- Definition der Schnittstellen
- Definition der Eigentümer von Daten

Unterprogramme

Typen

- Externe Programme in
  - COBOL
  - ASM
  - FORTRAN
  - PL/1
  - Java
  - C/C++
- interne Unterprogramme
- Funktionen der Laufzeitumgebung Language Environment

Unterprogramme

Logik bei Aufruf

- Definition des Unterprogramms
- Aufruf des Unterprogramms
- Übergabe der Daten von HP an das UP
- Zugriff auf die Daten durch UP
- Verarbeitung innerhalb des UP
- Beenden des UP
- Rücksprung zum Aufrufer
- evtl. Weiterverarbeitung der Daten durch HP

Unterprogramme

Syntaxbeispiel

- CALL pgm-name USING var-1 ... var-n
  [END-CALL] dynamischer Aufruf

- CALL ‘pgm-name’ USING var-1 ... var-n
  [END-CALL] statischer Aufruf *

- möglich:
  - BY REFERENCE UP erhält Originaldaten
  - BY CONTENT UP erhält Kopie der Daten

* Hinweis: Hier hat auch die Compile-Option “(NO)DYNAM” noch einen Einfluss.

Unterprogramme

dynamischer oder statischer Aufruf?

- statischer Aufruf heißt
  - gerufenes Modul wird zum Aufrufer fest gelinkt
  - bei einer Änderung des Moduls müssen alle Aufrufer erneut gelinkt werden
  - Aufruf ist minimal schneller
- dynamischer Aufruf heißt
  - flexibler Umgang mit Änderungen in Moduln

Unterprogramme

Aufruf an einem Beispiel

- Hauptprogramm

```
PROGRAM-ID. TES46.
WORKING-STORAGE SECTION.
01  UPRO      PIC  X(08).
01  LISTE.
 copy S03UAAAK.
01 HUGO.
  copy jsdgf.
PROCEDURE DIVISION.
. . .
   MOVE 'AAAA' TO UPRO
   CALL UPRO USING LISTE
                   HUGO
. . .
```

- Unterprogramm

```
PROGRAM-ID. AAAA.
WORKING-STORAGE SECTION.
01  irgendwas
LINKAGE SECTION.
01  UEBERGABE.
    copy S03UAAAK.
01 OTTO.
  copy jsdgf
PROCEDURE DIVISION
        USING UEBERGABE
              OTTO.
. . .

   GOBACK.
```

Unterprogramme

Übung(en) – Ü 5.1 – PGM1501 / PGM15011

- Schreiben Sie ein Programm, das ein Unterprogramm aufruft.
  - Im Hauptprogramm lesen Sie die Daten der Stadtverwaltung mit ACCEPT ein.
  - Das Unterprogramm wird für Satzart-1 = 1 aufgerufen. Dieses soll die übergebenen Daten in einer “lesbaren” Form ausgeben.

Unterprogramme

interne Unterprogramme

- Aufbau, Logik auf Aufruf wie externe Programme
- sind Bestandteil des Gesamtprogramms
- ist wichtig für Nutzung der OO-Features
- Regeln:
  - Jedes Programm hat eine ID DIVISION
  - Programmnamen müssen eindeutig sein.
  - CONFIGURATION SECTION nur im äußersten Pgm.
  - Jedes Pgm hat ein “END PROGRAM pgmname.”
  - Ein inneres Programm kann nur von direkten “Vater” aufgerufen werden.
  - Es gibt globale und lokale Variablen.

Unterprogramme

Schnittstellen zu anderen Anwendungen / Sprachen

- Benutzung der folgenden Compile-Optionen wird dringend empfohlen
  - NUMPROC(NOPFD)
    - bei allen Schnittstellen
  - ~~TRUNC(BIN)~~ COMP-5
    - bei Schnittstellen zu ~~C und PL/1~~ anderen Sprachen
- defensiver Umgang mit Feldern
  - bei allen Schnittstellen
- normaler Umgang mit Feldern
  - sonst

Unterprogramme

Übung(en) – Ü 5.2 – PGM1501/ PGM15011 / PGM15012 / PGM15013

- Erweitern Sie das Hauptprogramm, indem es eine Verarbeitungsart über ACCEPT einliest:
  - 1 = Standesamt
  - 2 = Einwohnermeldeamt
  - 3 = Polizeiamt
  - Für jede Verarbeitungsart soll es ein separates Unterprogramm geben, das die Daten ausgibt.
  - Nutzen Sie ein internes oder externes Unterprogramm.

Unterprogramme

Übung(en) – Ü 5.3 – PGM1502 / PGM15021 / PGM15022 / PGM15023

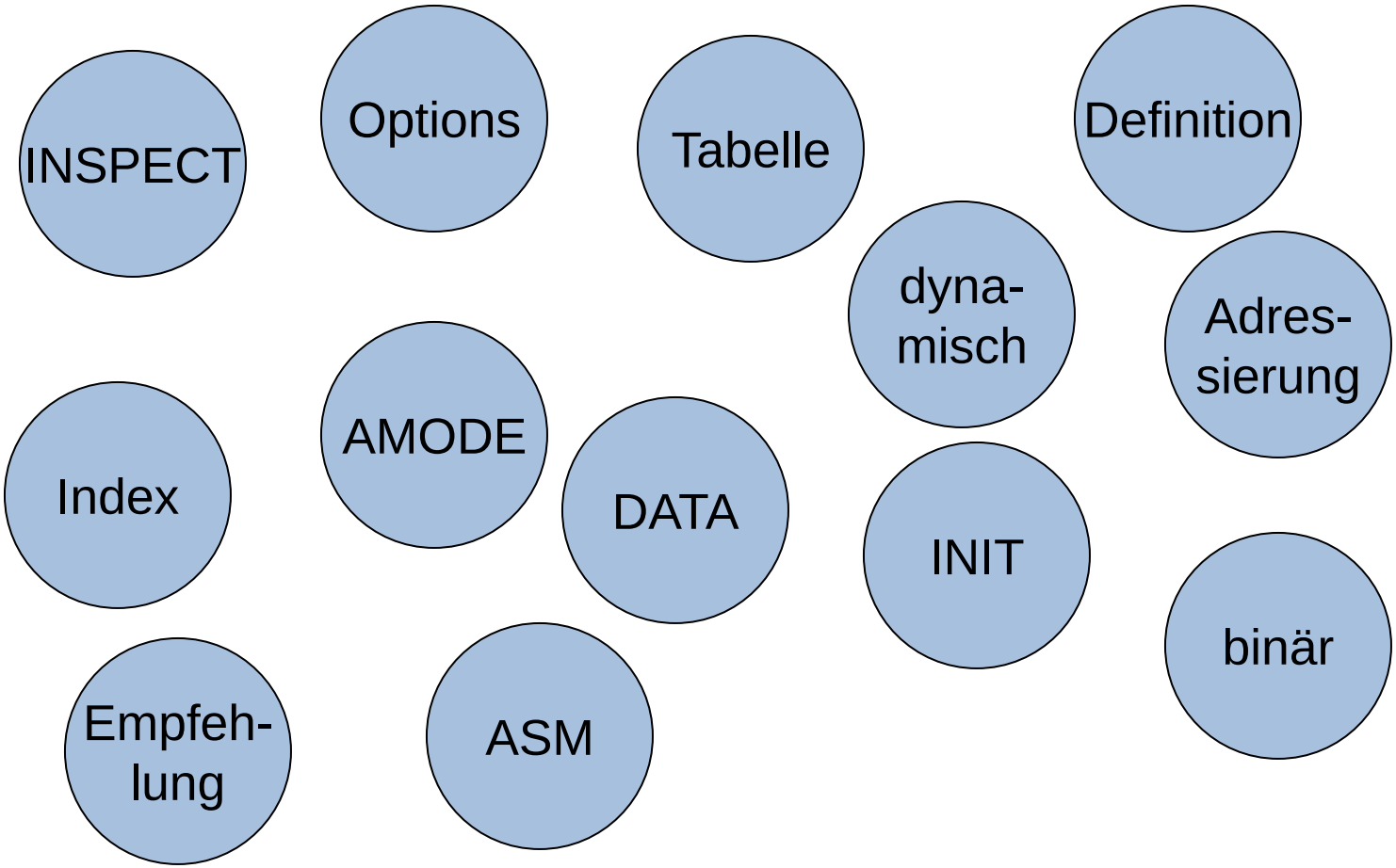
- Testen Sie ein Programm mit einem Unterprogramm, das ein Feld “by content” und eines “by reference” übergibt. Im Hauptprogramm und im Unterprogramm werden die Felder jeweils unterschiedlich gesetzt. Drucken Sie sich die Feldinhalte jeweils vor und nach der Modifikation und vor und nach dem UP-Aufruf an.

Inhalt

- Einführung: Rückblick auf Kurs Grundlagen Teil 1
- Datendefinition
- Strukturen
- Zeichenketten
- Compiler
- Unterprogramme
- Testhilfen und Performance
- Neuerungen aus den letzten (20) Jahren
- Fragen und Antworten

Testhilfen und Performance

Begriffe



Testhilfen und Performance

Allgemeines

- Testhilfen zu benutzen und performanten Code zu haben, widersprechen sich sehr oft und oftmals sehr deutlich.

- Damit ist die Zielrichtung klar:

Testhilfen sind für die Testumgebung

performanter Code ist für die Produktion

Testhilfen und Performance

Allgemeines

- DISPLAY
- Compile-Optionen
  - SSRANGE
  - NUMPROC(NOPFD)
  - OPT(0)
- absichtlicher Abbruch
- DEBUG Mode
- in Notfällen: Debugger wie
  - DEBUG-Tool oder Xpediter

Testhilfe – DISPLAY

- Der Befehl DISPLAY sollte in der Testumgebung intensiv genutzt werden, jedoch nicht in der Produktion.
  - Veränderung von Feldinhalten
  - Andrucken von Schaltern
  - Andrucken von Zwischenergebnissen
  - etc.

Testhilfen und Performance

Testhilfe – Compile-Optionen

- SSRANGE
  - gibt bei Index-Überlauf eine detaillierte Fehlermeldung
  - produziert inperformanten Code
- NUMPROC(NOPFD)
  - stellt durch raffinierten Code sicher, dass die veränderten nummerischen Felder korrekte Inhalte erhalten
- OPT(0)
  - erzeugt nur “normalen” Code

Testhilfen und Performance

Testhilfe – absichtlicher Abbruch

- Um im Testfall einen Abbruch zu erzeugen, kann ein S0C7 erzeugt werden z.B. durch

```
01    FELD-ALPHA   PIC X(03) VALUE “XXX”.
01    FELD-NUM  REDEFINES FELD-ALPHA
                        PIC  S9(05) PACKED-DECIMAL.
IF ABBRUCH-ERWUENSCHT
   ADD 1 TO FELD-NUM
END-IF
```

Testhilfen und Performance

Testhilfe – Debugging Mode

```
 ENVIRONMENT DIVISION.
*==
 CONFIGURATION SECTION.
*SOURCE-COMPUTER. IBM-3090.
 SOURCE-COMPUTER. IBM-3090  WITH DEBUGGING MODE.
*==
PROCEDURE DIVISION.
DDECLARATIVES.
DDEBUG-DECLAR SECTION.
D    USE FOR DEBUGGING ON ALL PROCEDURES.
DDEBUG-DECLAR-PARAGRAPH.
D    DISPLAY DEBUG-NAME.
DDEBUG-DECLAR-END.
D    EXIT.
DEND DECLARATIVES.
*==
D    DISPLAY "Dies ist eine Testzeile."
```

FUNKTIONIERT BEI DER R+V NICHT MEHR ☹

Testhilfen und Performance

Compile-Options: OPT (0|1|*2*)

Notation: Standard *Empfehlung*

- Vorteile:
  - Unnötige interne Programmverzweigungen eliminiert
  - Out-of-Line PERFORM Statements werden, wenn möglich, In-Line dargestellt.
  - Nicht erreichbarer Programmcode wird eliminiert und damit die Größe des Lademoduls reduziert.
  - Optimierte Subscript Verarbeitung
  - Redundante Rechenoperationen werden eliminiert.
  - Rechenoperationen für Konstanten werden eliminiert.
  - Einzelne, fortlaufende MOVE Statements werden als Single MOVE aggregiert.

Anmerkung: Gleichzeitig muss die Option LIST gesetzt werden. Diese wird benötigt, damit der Abend-Aid Postprozessor in Verbindung mit OPTIMIZE ohne Fehler durchläuft. Ohne LIST kann Abend-Aid bei einem Abbruch zwar die Offset-Adresse ermitteln, nicht aber das zugehörige COBOL-Statement.

Testhilfen und Performance

Compile-Options: OPT (0|1|*2*)

Notation: Standard *Empfehlung*

- Nachteile:
  - Einzelne, fortlaufende MOVE Statements werden als Single MOVE aggregiert. Dazu mögliche fachliche Auswirkungen berücksichtigen.
  - COMPILE-Zeit länger
  - DEBUGGING evtl. erschwert
- Beispiel -> LINK
  - d.h. nummerische Felder werden als CHAR übertragen!

Testhilfen und Performance

cps4it

Compile-Options: *AWO* | NOAWO

Notation: Standard *Empfehlung*

- Auswirkungen:
  - Der Parameter bezieht sich auf geblockte sequentielle Dateien mit variabler Satzlänge im Output Modus.
  - COBOL prüft bei AWO, ob der zu schreibende Satz noch in den zur Verfügung gestellten Buffer passt.
  - Bei NOAWO (Compilerdefault) geschieht diese Prüfung nicht sondern der Buffer wird weggeschrieben, wenn der längste, im Programm definierte Satz nicht mehr in den Buffer paßt.
  - Mit AWO kann hier CPU und Laufzeit eingespart werden. Abhängig von den Satzdefinitionen können die Einsparungen über 50% erreichen.

Testhilfen und Performance

Compile-Options: *TRUNC(OPT)* | TRUNC(BIN) | TRUNC(STD)

Notation: Standard *Empfehlung*

- Auswirkungen – 1:
  - TRUNC ist bei allen Rechen- und Vergleichsoperationen mit binär definierten Feldern aktiv. Die empfohlene Einstellung ist bei der R+V *nicht* der Compilerdefault.
  - Der maximale Wertebereich von Binärfeldern (COMP) ist bei TRUNC(STD) durch die Anzahl der definierten Digits vorgegeben. Da der COBOL-Compiler intern dezimal „arbeitet“, sind keine Prüfungen auf Überläufe notwendig, sie sind automatisch erfüllt, was den CPU-Overhead reduziert.
  - TRUNC(BIN) heißt, dass der volle Wertebereich der Binärfelder benötigt wird. Dafür werden die Felder in ein internes dezimales Format konvertiert (der COBOL-Compiler arbeitet intern „dezimal“). Um die richtige Anzahl von Ziffern zu erhalten, müssen inperformante Formate und Operationen benutzt werden.

Testhilfen und Performance

Compile-Options: *TRUNC(OPT)* | TRUNC(BIN) | TRUNC(STD)

Notation: Standard *Empfehlung*

- Auswirkungen – 2:
  - TRUNC(OPT) versucht, keine dezimal-Arithmetik wie bei TRUNC(BIN) zu nutzen; manchmal ist es aber notwendig . . .
    d.h.: bei reinen binären Feldern -> voller Wertebereich
    bei Mischung von Formaten -> wie TRUNC(STD) !!
  - TRUNC(STD) hat einen minimalen Performanceverlust gegenüber TRUNC(OPT).
  - Definition COMP-5 heißt, dass dieses Feld wie TRUNC(BIN) behandelt wird. Alle anderen Felder halten sich an die Compiler-Option.

DB2: INTEGER / SMALLINT
CICS: EIBCALEN

- Also: voller binärer Wertebereich benötigt . . .
  Nutze COMP-5 mit TRUNC(OPT)
  Nutze nicht TRUNC(BIN) !!!

Testhilfen und Performance

Compile-Options: *NUMPROC(PFD)* | ***NUMPROC(NOPFD)***

Notation: Standard *Empfehlung*

- Auswirkungen:
  - NUMPROC(NOPFD) führt implizit Vorzeichen-prüfungen für packed decimal und usage display Felder durch. Bei Einsatz von NUMPROC(PFD), preferred sign, geht der Compiler davon aus, dass die nummerischen Felder das richtige Vorzeichen haben. Prüfungen auf das Vorzeichen finden nicht statt.
  - Rechen- und Vergleichsoperationen benötigen weniger CPU während der Ausführung.
- theoretischer Nachteil bei PFD:
  - bei unsicheren Datenquellen könnten erst später zur Laufzeit Fehler auftreten.

Testhilfen und Performance

Compile-Options: ***DATA(31) (mit RENT)*** | DATA(24)

Notation: Standard *Empfehlung*

- Auswirkungen:
  - Die QSAM-Buffer und die Working Storage werden above-the-line angelegt.
  - Das Programm wird bei RENT in die LPA/ELPA geladen.
- Vorteil:
  - schnellere I/O-Behandlung; bessere Speicherausnutzung
- Nachteil:
  - bei RENT wird zum Programmanfang minimal mehr Code generiert, der RENT prüft.

Testhilfen und Performance

Compile-Options: ***RMODE(ANY)*** | RMODE(24)

Notation: Standard *Empfehlung*

- Auswirkungen:
  - Programm wird dort hin geladen, wo Platz ist.
- Vorteil:
  - Das System sucht optimalen Platz für das Programm.
- Nachteil:
  - keiner bekannt

Testhilfen und Performance

COBOL Compile Options – Zusammenfassung

1. Überlegen, welche Option welche Auswirkungen hat.
   - Umgebung, Typ des Programms beachten
2. Hin und wieder auf Basis Assembler Listing entscheiden, was Sinn macht.
3. Nicht optimieren, weil es Spaß macht, sondern optimieren, weil/wo es Sinn macht.
4. Die fachlichen Hintergründe sind ein wesentlicher Maßstab zu entscheiden, wann welche Option eingesetzt wird.
5. COBOL schüttelt man nicht aus dem Ärmel.

Testhilfen und Performance

Felddefinitionen – 1

- Binärfelder
  - Halbwort S9(04) oder Vollwort S9(08) mit Vorzeichen
  - Compile Option TRUNC beachten
  - Doppelwort (z.Z.) sehr inperformant
- gepackte Felder
  - auf Bytegrenzen achten (S9(n) mit n ungerade <= 15)
- USAGE DISPLAY
  - nicht für Rechenoperationen verwenden

Testhilfen und Performance

Felddefinitionen – 2

- Tabellen
  - nur mit Indizes (INDEXED BY)
  - Ausnahme Binärfelder
  - niemals andere nummerische Felder benutzen
  - möglichst 1-dimensional
  - ODO möglichst nicht nutzen; und wenn, dann: ODO-Feld muss binär sein

Testhilfen und Performance

Procedure Code – 1

- INITIALIZE
  - jedes einzelne Feld wird auf Anfangswert gesetzt
  - **jedes** einzelne Feld wird auf Anfangswert gesetzt
  - innerhalb Schleifen möglichst unterlassen
  - Hilfsfelder nutzen
  - jedes schwierige Beispiel muss separat beurteilt werden, daher kein “Kochrezept” möglich
- STRING/UNSTRING/INSPECT
  - zieht hohen CPU-Verbrauch nach sich
  - inzwischen werden durch Compiler einfache Befehle inline bearbeitet, statt eine Funktion aufzurufen

Testhilfen und Performance

Procedure Code – 2

- PERFORM VARYING
  - Schleifenzähler binär definieren
  - Begrenzer binär definieren
  - bei Tabellenverarbeitung nur mit INDEX arbeiten
  - jederzeit auf Formatgleichheit achten
- EVALUATE
  - (leider wieder:) häufigsten Fall zu Beginn codieren
  - einen Evaluate als einen geschachtelten if zu nutzen, entspricht nicht den Regeln der “Strukturierten Programmierung“ und erschwert die Wartung enorm

Testhilfen und Performance

Procedure Code – 3

- Rechenoperationen
  - beteiligte Felder mit gleichen Längen
  - beteiligte Felder mit gleichem Format
- Vergleichsoperationen
  - beteiligte Felder mit gleichen Längen
  - beteiligte Felder mit gleichem Format
- Substr-Move
  - besser: MOVE FELD-A(2:5) TO FELD-B (-> MVC)
  - nicht: MOVE FELD-A(2:N) TO FELD-B (-> MVCL)
    erste Zahl darf Variable sein

Testhilfen und Performance

Übung(en) – Ü 6.1 – PGM1601

- Schreiben Sie ein kleines Programm, in dem 2 mal 3 Felder mit den Definitionen PIC 9(09), PIC 9(09) COMP-3, PIC 9(09) COMP enthalten sind. Übertragen Sie die Inhalte des ersten Blocks in jedes Feld des zweiten Blocks.
  - Lassen Sie das jeweils 10.000.000 Mal laufen und vergleichen Sie.
  - Prüfen Sie den ASM Code.

Testhilfen und Performance

Übung(en) – Ü 6.2 – PGM1602

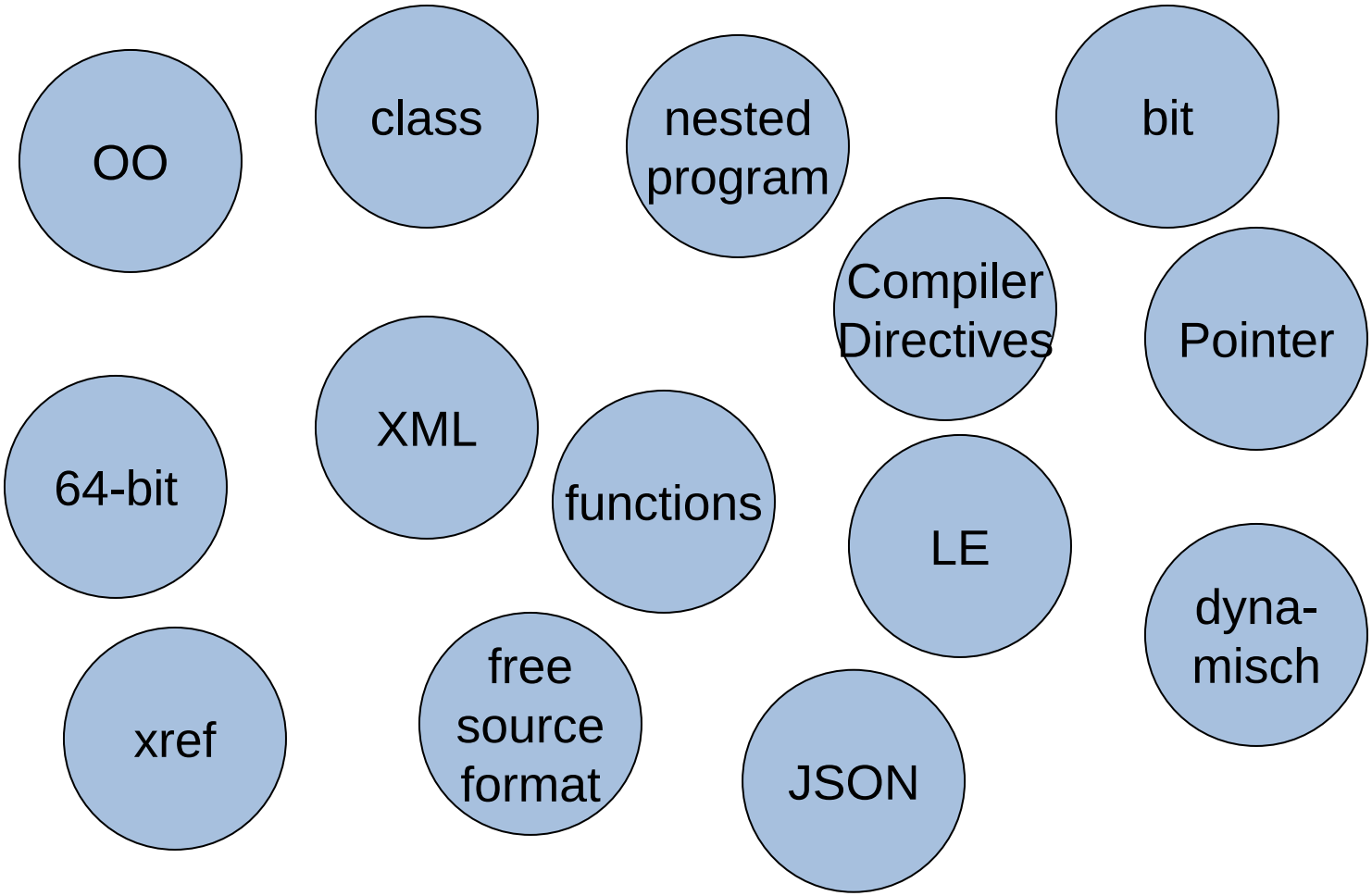
- Definieren Sie eine Tabelle, die Sie initialisieren. Tun Sie dieses unterschiedlich und schauen Sie sich bei 10.000.000 Aufrufen die Unterschiede in der CPU-Zeit an.

Inhalt

- Einführung: Rückblick auf Kurs Grundlagen Teil 1
- Datendefinition
- Strukturen
- Zeichenketten
- Compiler
- Unterprogramme
- Testhilfen und Performance
- Neuerungen aus den letzten (20) Jahren
- Fragen und Antworten

Neuerungen aus den letzten (20) Jahren

Begriffe



Neuerungen aus den letzten (20) Jahren

cps4it

Objektorientierung

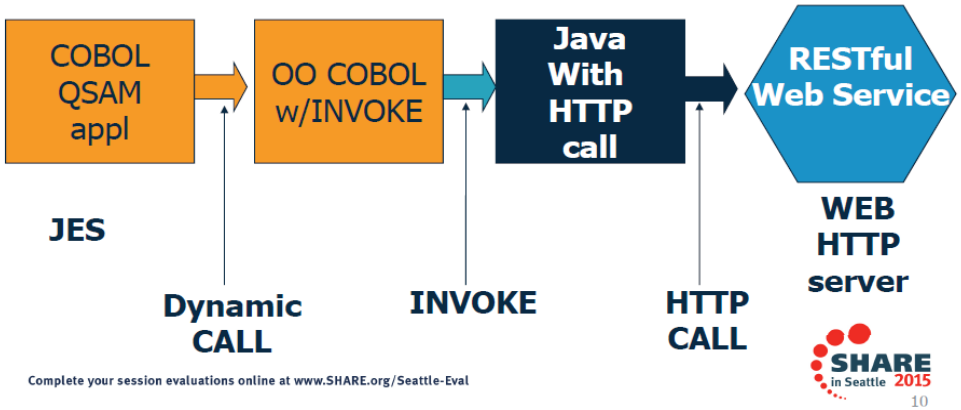
- C☺B☺L kann ☺☺
  - Kapselung von Funktionen und Daten
  - verschiedene Einstiegspunkte in das Programm
  - Bekanntgeben von Klassen
  - entsprechende Datentypen
  - etc.
- Kommunikation mit Java, C++, . . .
- Die Kommunikation mit OO-Sprachen geht, weil COBOL OO kann, nicht weil OO-Sprachen prozedural können! COBOL ist m☺dern!

Neuerungen aus den letzten (20) Jahren

cps4it

Objektorientierung – zu beachten

- interne Unterprogramme für classes und inheritance
- invoke als Aufruf
- DLL als Compile-Option
- DLL/NODLL vertragen sich nicht
  - XPLINK-convention vs. HLASM-convention



Neuerungen aus den letzten (20) Jahren

cps4it

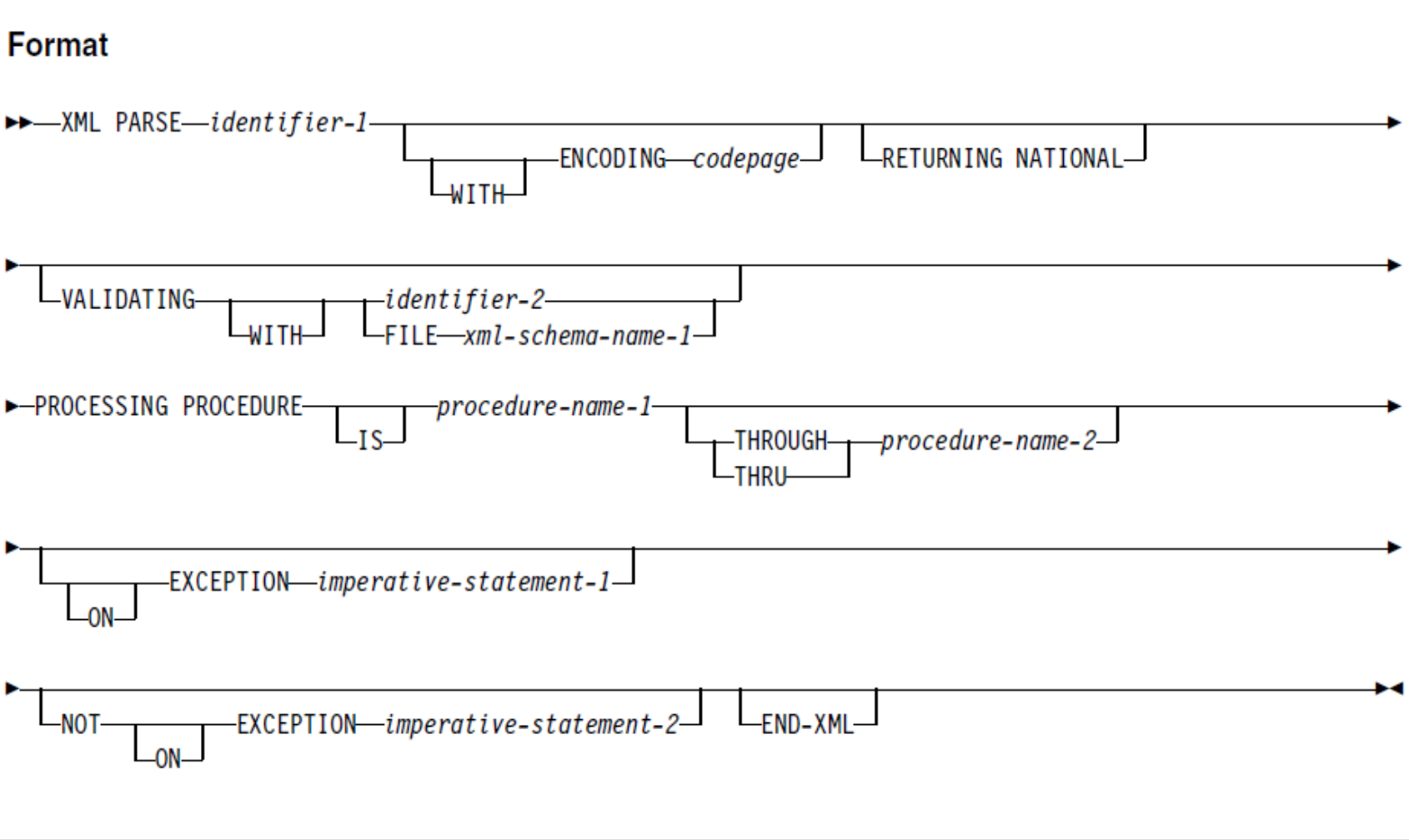
XML

- XML PARSE
  - eigener Parser, aber besser: Parser des z/OS
- Inhalte der Tags -> Felder in COBOL
- Conditionhandling
- XML GENERATE
  - XML generieren aus Struktur heraus
- XSD kann benutzt werden (Parameter)

Neuerungen aus den letzten (20) Jahren

XML Parse – Syntax

Format

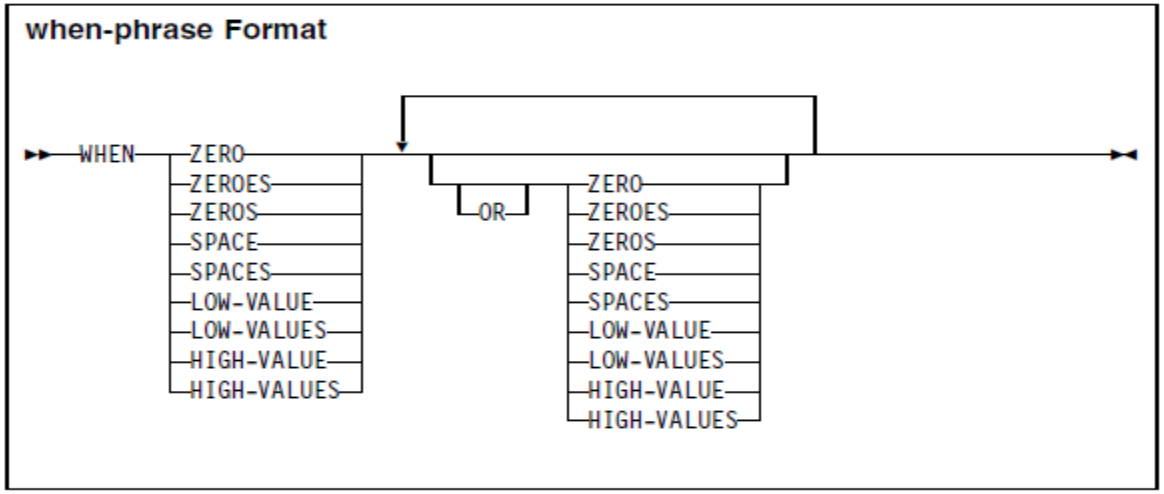


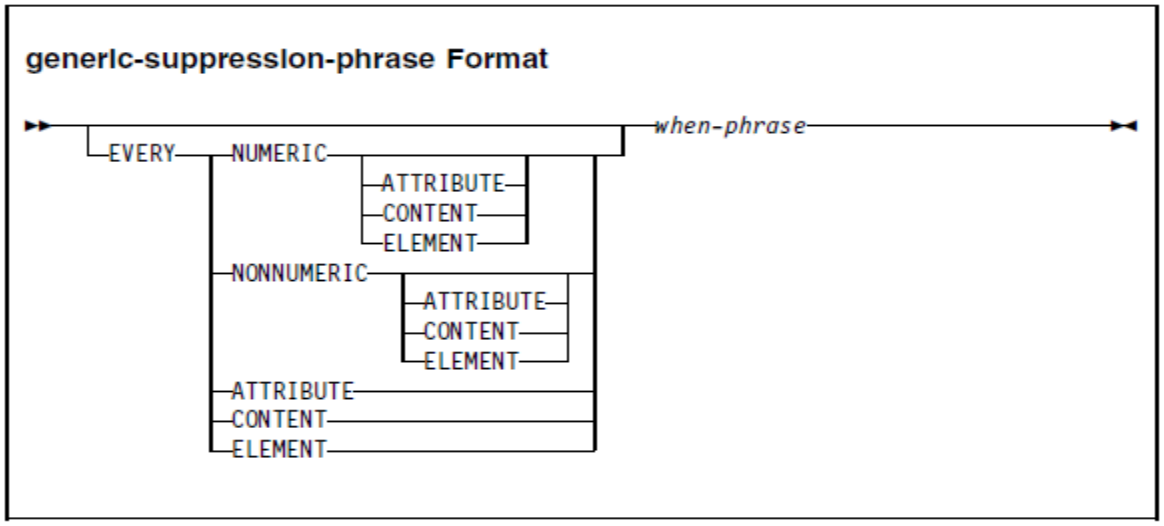
Neuerungen aus den letzten (20) Jahren

XML Generate – Syntax – 1

Neuerungen aus den letzten (20) Jahren

XML Generate – Syntax – 2





Neuerungen aus den letzten (20) Jahren

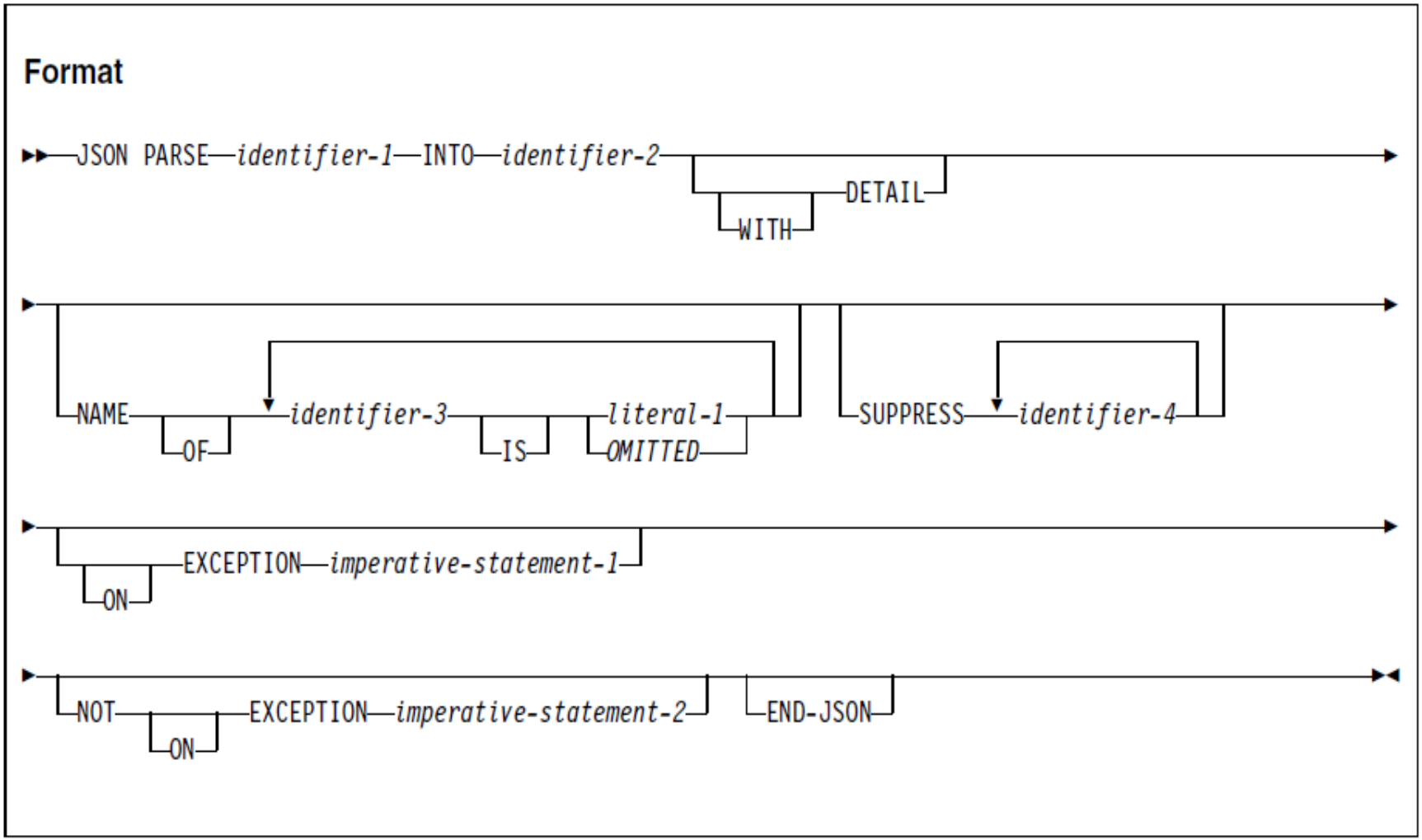
JSON

- JSON PARSE
- Inhalte der Tags -> Felder in COBOL
- Conditionhandling

- JSON GENERATE

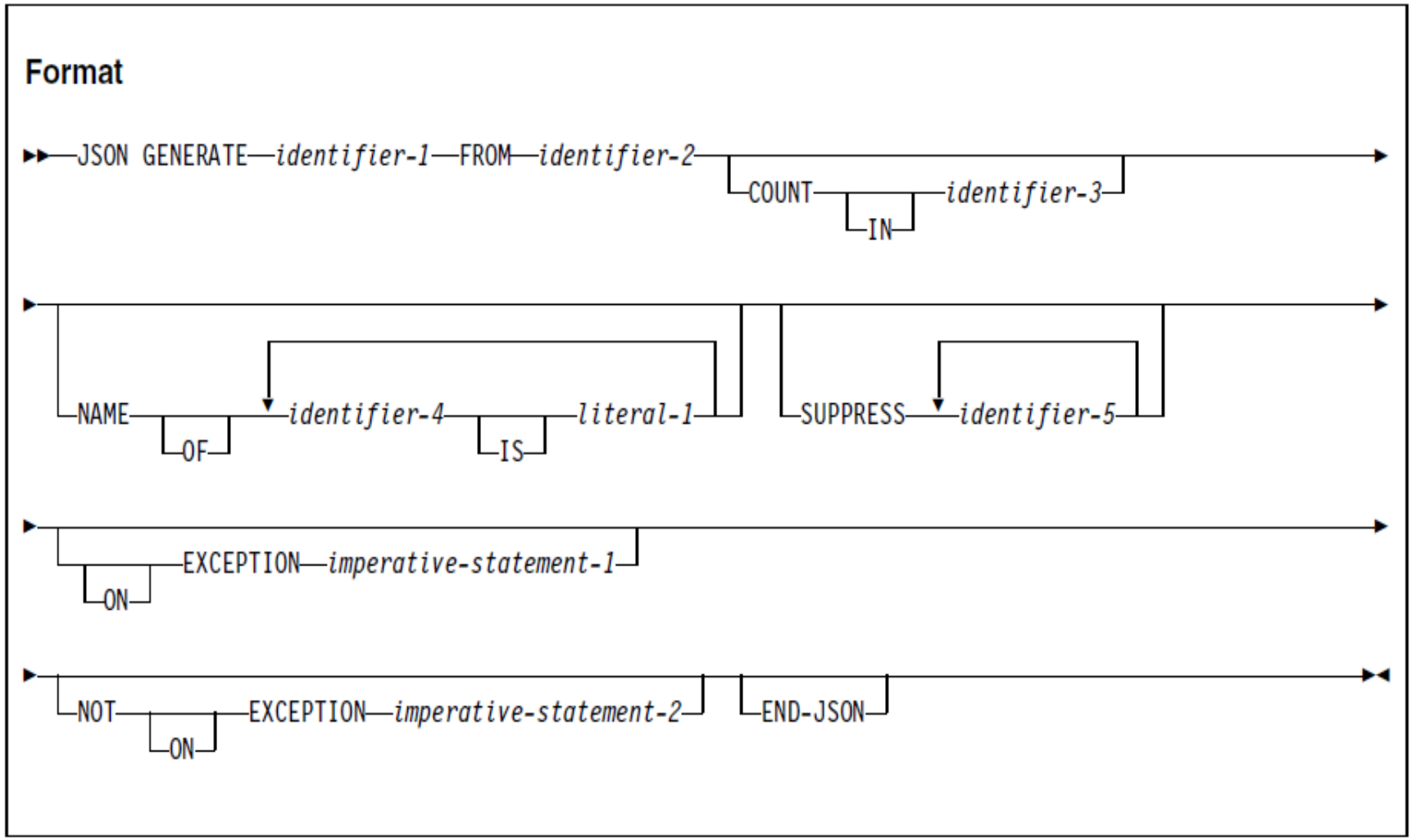
Neuerungen aus den letzten (20) Jahren

JSON Parse – Syntax



Neuerungen aus den letzten (20) Jahren

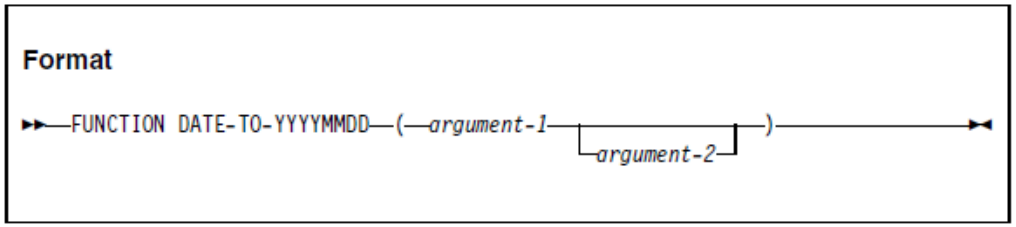
JSON Generate – Syntax



Neuerungen aus den letzten (20) Jahren

intrinsic Functions

- mathematische Funktionen
- statistische Funktionen
- Datum-/Zeit Konvertierungen
- Formatumwandlungen (char-num)
- Unicode / DBCS / National
- Definition von eigenen Functions (bald)



- Language Reference
  - http://publibfp.boulder.ibm.com/epubs/pdf/igy6lr20.pdf

Neuerungen aus den letzten (20) Jahren

Compiler-directing Statements – Compilerbefehle

- COPY – seit Urzeiten bekannt
- EJECT – seit Urzeiten bekannt
- CBL – seit Urzeiten bekannt
- DELETE
- INSERT

- hier wird intensiv „gearbeitet“, um den Standard nach und nach zu erfüllen

Neuerungen aus den letzten (20) Jahren

cps4it

free source format

- keine Begrenzung auf Spalten 7-72 (Lochkarte!!)
- daran wird gearbeitet
- Herausforderungen
  - Garantie für „altes“ Format
  - Mischung von Source-Formaten
  - eventuell eingeschränkt
- Steuerung voraussichtlich über Compileroptionen und Compilerdirectives

Neuerungen aus den letzten (20) Jahren

Was man so alles noch machen kann . . . (1)

- Pointer
- Speicher allokieren / freigeben
- interne Unterprogramme
- Unterstützung DB2 fetch rowset (bald)
- Referenzmodifikation hin und her
- LE functions nutzen
- 64-bit (ab/seit Release 5)
  - Achtung: Da ist intern vieles geändert!
- Inline-Kommentar *>

Neuerungen aus den letzten (20) Jahren

Was man so alles noch machen kann . . . (2)

- dynamic File allocation
- querying environment variables (z.B. DD-Name)
- invoke a C function written in COBOL
- cross reference of COPY libraries
- retrieve system info (z.B. Abbruchinfos)
- test bits / set bits

- zum Beispiel siehe (Tom Ross auf Share 2009!):
  http://www-01.ibm.com/support/docview.wss?uid=swg27009580&aid=1

Neuerungen aus den letzten (20) Jahren

Übung(en)

- Schauen Sie in den COBOL-Broschüren nach, welche Punkte in den Kursen nicht erwähnt worden sind.
- Schauen Sie die Dokumente an, die den COBOL 2002 Standard beschreiben.

Inhalt

- Einführung: Rückblick auf Kurs Grundlagen Teil 1
- Datendefinition
- Strukturen
- Zeichenketten
- Compiler
- Unterprogramme
- Testhilfen und Performance
- Neuerungen aus den letzten (20) Jahren
- Fragen und Antworten