Language concepts in old and recent programming languages

Programming Languages Guest Lecture

Ivaylo Bonev
Aachen, 10th December 2018

Agenda

itestra
be excellent

01 Motivation
- Why “old” programming languages are still relevant

02 Selected programming languages
- RPG, COBOL, PL/I, C, Java Script

03 Selected language concepts
- Parameterization und Reuse
- Program generators and 4th Level Languages

04 Conclusion

Who we are

About itestra

itestra GmbH = **i**nformation **te**chnology and **stra**tegy

Foundet in 2003 gegründet, bis heute technologisch und finanziell unabhängig

Currently approx. 75 employees and 25 students

8 locations across Europe - Munich, Cologne, Stuttgart, Nuremberg, Hamburg, Hannover, Madrid, Tallinn

Service Portfolio

We ensure and reestablish IT efficiency.

Solution Engineering

- Planning, design and implementation of cost effective, high performance, mission critical IT solutions

Governance & Renovation

- Measurement of software quality and costs
- Systems renovation and optimization
- Reduced costs and increased value

Our Clients

itestra
be excellent

Automotive / Industry	Banks	Insurance	IT/Software	Others
BMW		GENERALI Informatik Services	O2	SIXT rent a car
DAIMLER	ebase	Allianz	e-plus	KUEHNE+NAGEL
KMW	HypoVereinsbank	AdvoCARD	BITMARCK	Lufthansa
thyssenkrupp	.comdirect	ZURICH	ITERGO	Flughafen München
e.on	finanz informatik	VERSICHERUNGSKAMMER BAYERN	FIS KORDOBA	Deutsche Post
EVN	IT SOLUTIONS	AXA	RTT	CHIRON
TRW	3Banken EDV	ADAC		InBev
	GAD IT für Banken	SwissLife RHV		

In total ~ 100 core applications with ~ 500 Mio LoC analyzed.

Reality in most large scale enterprises

100s or even **1.000s of “applications”**
(from standard products to individual SW)

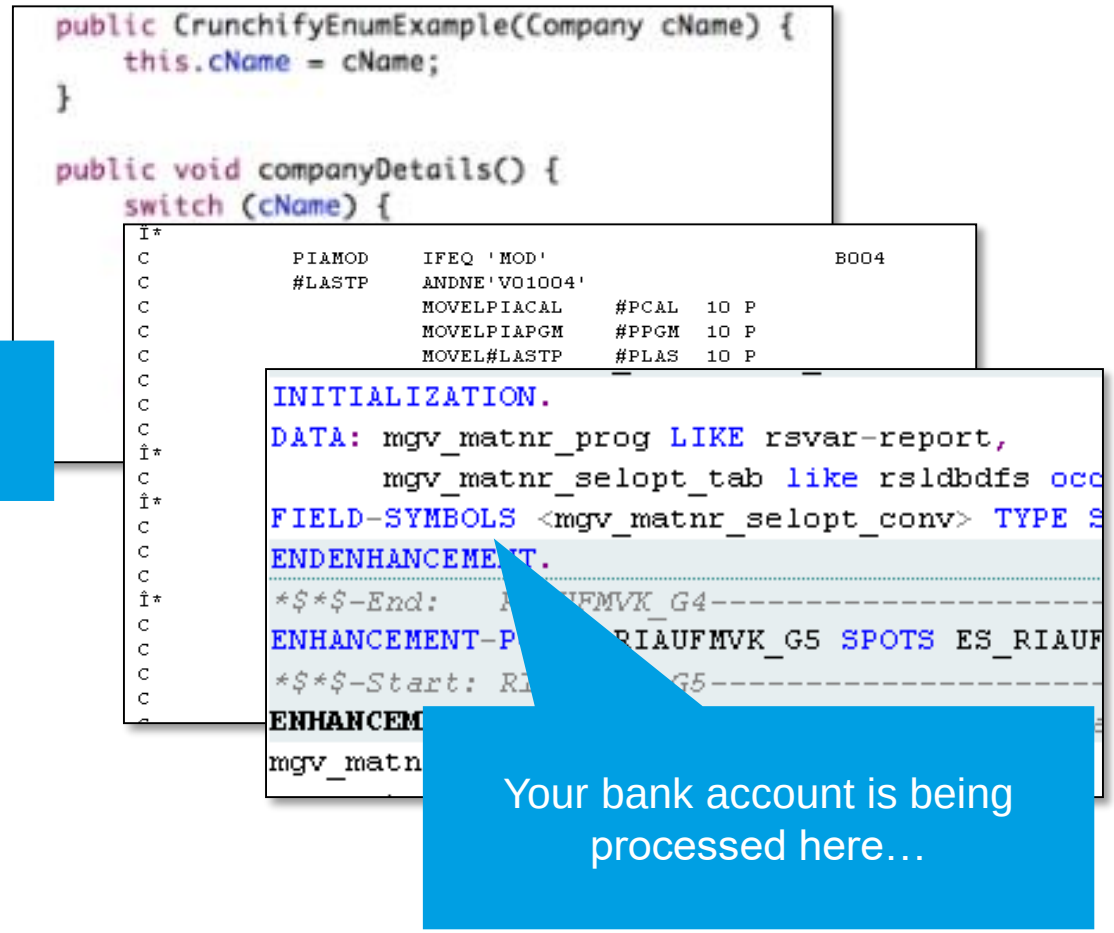
10 - 100 Mio. LoC, value of 100 – 1.000 Mio. €

often several **decades old**

Languages

- COBOL, Java, ABAP
- Assembler
- PL/I, RPG, NATURAL, C/C++
- VaGen, DeltaGen,
- SAS, Easytrieve, PowerBuilder, Gupta, Synon, …

age & size mean **success**!



Your bank account is being processed here…

Heterogenity is unavoidable in practice!

Long lifecycle, too big to reimplement/migrate everything every 10 years

Mergers & acquisitions

(besides, heterogenity also due to hypes/personal preferences, lack of strategy, …)

We get to see interesting systems …

itestra
be excellent

1968

- Salary calculation system in COBOL
- Extended and modified for now 50 years!

1970

- Life Insurance
- 5 Mio LoC Assembler

1980+

- Diverse systems in proprietary „4 GLs“: Coolgen, VaGen, Gupta, PowerBuilder,…

2012

- General insurance
- 1 Mio LoC Java
  → very messy and extremely high costs ;-)

Motivation (1)

A significant amount of all **business critical software systems** is implemented in programming languages, which

- are not being taught anymore
  → difficult to find staff, knowledge loss,
- do not offer concepts of modern languages,
- have limited tool-support,
- yet still need to be mastered in order to be able to maintain, run and develop the extremely important systems built with them

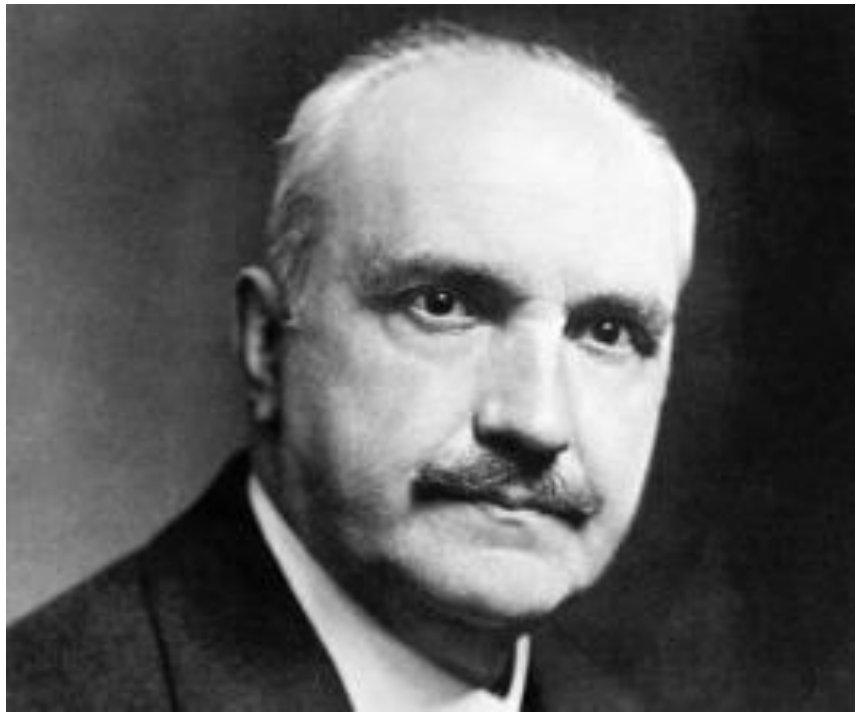
Motivation (2)

itestra
be excellent

Supposedly “new” products and approaches often lack important basic concepts. The consequences can be studied using existing systems, implemented in „old“ languages.

“Those who cannot remember the past are condemned to repeat it.“

George Santayana (US Philosoph, 1863-1952)



How are languages chosen for projects?

itestra
be excellent

In the absence of alternatives
(in the past there were (even) less proven-in-use languages available)

Dependency on hardware and infrastructure software

Successful marketing and sales by the providers

Compatibility to already existing components
(if a system is already implemented in COBOL, new modifications cannot be done in an arbitrary language)

Inherited system, e.g. from a merger

Available **qualifications**

Concepts / Performance? … secondary!

History of development

Major developments 1960 – 2010s (1/2)

itestra
be excellent

Hardware

Punch cards → magnetic tape → disks (i.e. from bytes to terabytes)

Closed systems → Internet → mobile Computing & Cloud

Performance of the CPUs, Multi-Core, Memory kBytes → GBytes

Infrastructural software

OS: Single-User & Task → Scheduling, parallelism

DB: Files → hierarchical databases → relational databases → distributed DB

Compiler: text replacement → attributed parse trees

Major developments 1960 – 2010s (2/2)

Software Engineering

Separation of Concerns, etc.

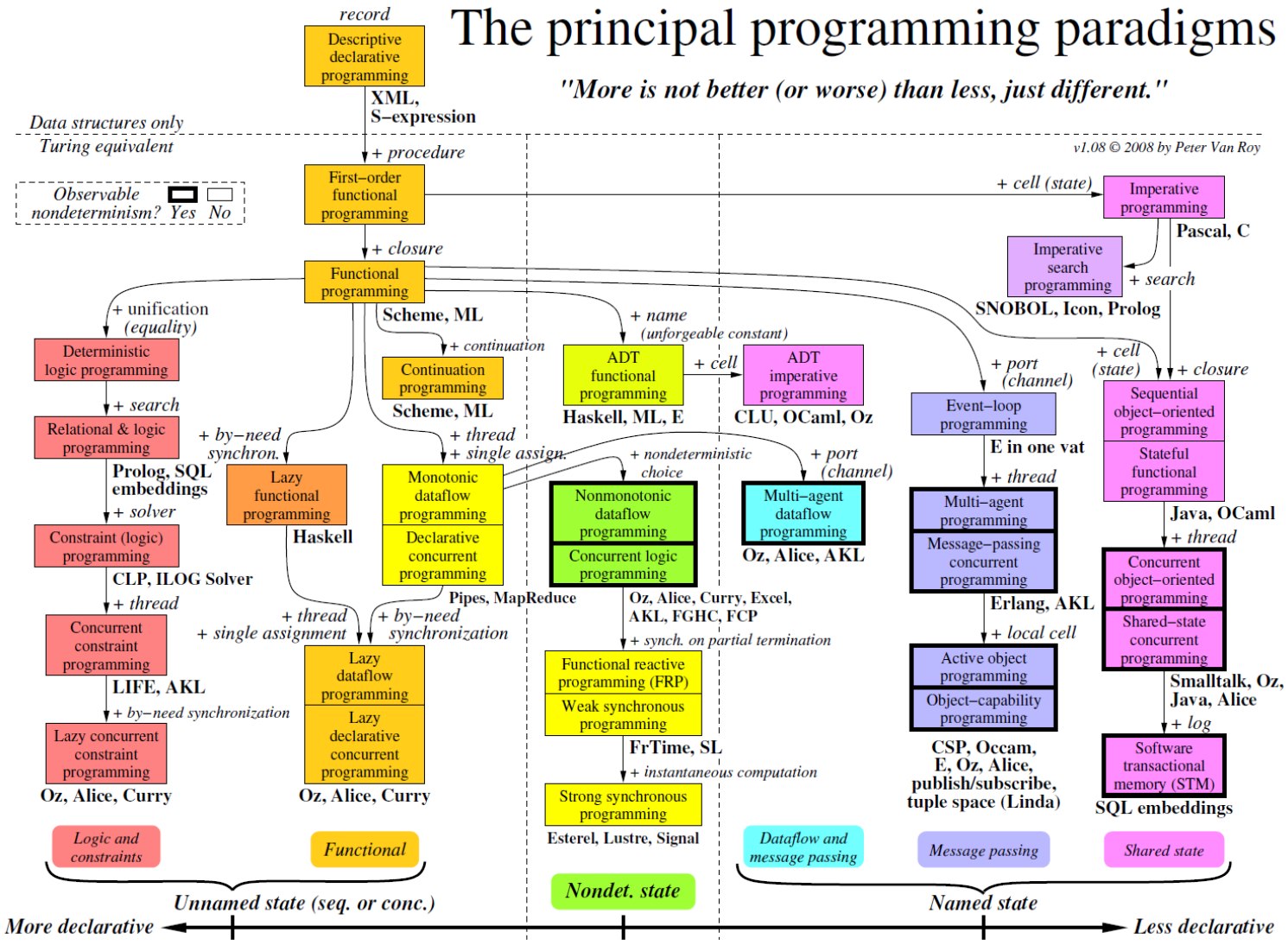
User requirements and market

Personal talk at the bank / insurance / … and written communication
→ 7x24 online, complete transparency, social networks

Established sales network → fast monopolization through technology and venture capital, willingness to switch

amazon.com

Programming Paradigms



Some important conceptual features

itestra
be excellent

Syntactic structure

- Tokens vs. columns, Lines vs. statements (in some languages only 1 per line)
- Fixed vs. variable number of operands

Scope

- Local variables in blocks (methods, …)
- Differentiated control (private, public, friend, static, …)

Modularization, Parameterization, Reuse

- Parameterization (Units: procedures, functions, classes, packages, …)
- Encapsulation of data AND operations
- Inheritance, Generic data structures and polymorphism
- Library concept (none vs. technical via text-inclusion and linker vs. language)

Avoidance of programming errors

- Compile-time checking of types, signatures, …
- Memory management: Garbage Collection

The existence/ absence of these concepts has enormous consequences!

itestra
be excellent

Selected languages: RPG

RPG

Report Program Generator

Developed by IBM in 1959

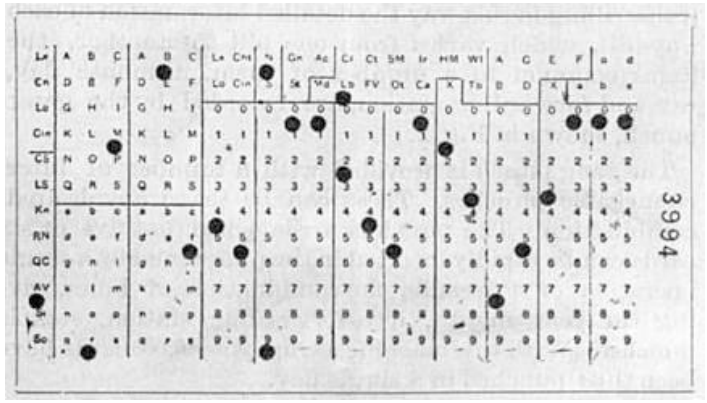
For the creation of commercial reports

Data processing using tabulating machines and punch cards → column-based Syntax

Global variables per RPG-program

Data access only via **“file” cursors**

Still new systems developed in RPG in the 90s!




```
C                   #LASTP    IFEQ 'VO1004'                          B003
C                             MOVEL#PLAS      #LASTP
C                             MOVEL#PPGM      PIAPGM
Î*
C                   PINDAB    IFEQ *ZEROS                            B004
C                   PIAMO7    ANDEQ'1'
C                             Z-ADD32         PINRTC                  DAB EING.
C                             MOVEL'VO1001'   PIACAL
C                             ELSE                                   X004
C                             MOVEL#PCAL      PIACAL
C                             ENDIF                                  E004
Î*
C                             ENDIF                                  E003
```

itestra
be excellent

Selected languages:
COBOL

COBOL

COBOL stands for „**C**ommon **B**usiness **O**riented **L**anguage“

First version: 1960, created as part of US DoD program

Objectives:

- **Human-readable, no need for computer scientists**
  ```
  ADD STEUER TO NETTO GIVING BRUTTO.
  ```
- Hardware-independent, standardized, problem-oriented language
- Easy-to-implement solutions for commercial problems

Instructions

Instructions: conditions, loops

```
      *A.1.B..+....2....+....3....+....4....+....5....+....6....+....7..I.+....8
       ------------------------------------------------------------------------
       STEUERUNG SECTION.
      *------------------------------------------------------------------------
           PERFORM VORLAUF.
      *    NOCH KEINE UEBERNAHME ERFOLGT: ALLES MERKEN
           IF AGSBF-M-KEINE-KEST
              PERFORM KEST-SOLZ-EINGABE
           END-IF.
      *    KIST KANN JEDESMAL KOMMEN
           PERFORM KIST-EINGABE.
      *    NUR BEI ZU VERSTEUERNDEM BETRAG U. STEUERSATZ, SONST "TU NIX"
           IF    NOT AGSBF-E-KEIN-BRUTTO
             IF AGSBF-E-STLIQ-FUER-KEST-SOLZ-J AND
               NOT AGSBF-M-BRUTTO-GES-KEIN
               PERFORM STEUERN-BERECHNEN-VON-STLIQ
             ELSE IF AGSBF-E-STLIQ-FUER-KEST-SOLZ-N AND
                     NOT AGSBF-M-KEINE-KEST
      *        STEUERSAETZE IN FAKTOREN UMRECHNEN
               PERFORM UMRECHNEN-SATZ-IN-FAKTOR
               PERFORM STEUER-BERECHNEN
             END-IF
           END-IF.
```

No user-defined functions (SECTION does not return any value)

Program-global variables, no parameters or local vars to SECTIONs

Functionality is often integrated in the language (thus >300 keywords),
e.g. INSPECT, UNSTRING, REWIND, COMPUTE, …

Data

Data: Structures, Fixed Size Arrays (**only**!)

```
....•.*A.1.B..•..•.2....•....3....•....4....•....5....•....6....•....7..I.•....8
      05 HF-3-AZSPF-FELDER.
         10 HF-3-AZSPF-STATUS          PIC 9          VALUE ZERO.
            88 HF-OHNE-3-AZSPF-VERARB                 VALUE ZERO.
            88 HF-MIT-3-AZSPF-VERARB                  VALUE 1.
         10 HF-3-AZSPF-VV-STATUS       PIC 9          VALUE ZERO.
            88 HF-OHNE-3-AZSPF-VV-VERARB              VALUE ZERO.
            88 HF-MIT-3-AZSPF-VV-VERARB               VALUE 1.
         10 HF-3-AZSPF-DAUER           PIC S9(02)           COMP-3
                                                     VALUE ZERO.
....•.*A.1.B..•....2..•.•....3....•....4....•....5....•....6....•....7..I.•....8
         10 HF-VG-DDAT-ZF                PIC X(08)      VALUE SPACES.
         10 HF-VG-DDAT-ZF-R      REDEFINES HF-VG-DDAT-ZF.
            15 HF-VG-DDAT-ZF-JHJJ        PIC 9(04).
            15 HF-VG-DDAT-ZF-MMTT        PIC 9(04).
            15 FILLER            REDEFINES HF-VG-DDAT-ZF-MMTT.
               20 HF-VG-DDAT-ZF-MM       PIC 9(02).
               20 HF-VG-DDAT-ZF-TT       PIC 9(02).
         10 HF-VG-DDAT-ZF-9      REDEFINES HF-VG-DDAT-ZF
                                         PIC 9(08).
```

No reusable definition of custom types
(except with COPY)

Native fix-comma types and arithmetics – still one important argument why COBOL is used for financial purposes – compare Java BigDecimal:
`brutto = steuer.Add(netto)`

Generally intensive use of nested structures and REDEFINEs (storage overlay) – amongst others also because type definition is not possible

COPY-Concept

First approach for reuse of interface definitions

```
      *----------------------------------------------------------------
       LINKAGE SECTION.
      *----------------------------------------------------------------
           COPY XUFMELDW.
           COPY XHINMELW.
           COPY XSQLRETW.
           COPY XDEVKURW.
      *    UEBERGABE-BEREICH IN DIESES AGSBCEMU UND AGSBREMU
           COPY AGSBREMW.
      *    UEBERGABE-BEREICH IN AUFGERUFENES AGSBEEMU
           COPY AGSBEEMW.
      *    UEBERGABE-BEREICH IN AUFGERUFENES AGSBFEMU
      *    UEBERGABEBEREICH IN SUPER.AAKBUEMU
           COPY AGSBFEMW.
           COPY AAKBUEMW.
```

Sometimes misused, e.g. for inclusion of code

```
      ******************************************************************
      *** PROCEDURE-COPY-STRECKEN OHNE SQL-ZUGRIFF                   ***
      ******************************************************************
      *** SECTIONS ZUM AUFRUF DES UNTERPROGRAMMS XDBCOSYU ZUM AN- BZW.
      *** ABMELDEN VON DER DATENBANK
            COPY XDBCOSYP.
      *** DATEI-FILESTATUS IN SYSTEMUNABHAENGIGEN RETURNCODE UMSETZEN
            COPY XDFSRCFP.
      *** SYSTEMABH. SQL-RETURNCODE IN SYSTEMUNABH. RETURNCODE UMSETZEN
            COPY XSQLRCFP.
      *** SECTION ZUR AUFBEREITUNG VON FELDERN
            COPY XFLDAIUP.
```

SQL and transactions

Embedded SQL

Embedded CICS (e.g. EXEC CICS LINK, oder GETMAIN, …)

```
      *A.1.B..+....2..+.+....3....+....4....+....5....+....6....+....7..I.+....8
      *---------------------------------------------------------------
       STEUER-KI-LESEN-START.
           INITIALIZE                              HV-STEUER-DGDEV-ISOCODE.
      *
           EXEC SQL
                SELECT
                   DG_DEV_ISOCODE
                INTO
                   :HV-STEUER-DGDEV-ISOCODE
                    INDICATOR :HV-I-STEUER-DGDEV-ISOCODE
                FROM
                   AZKIK
                WHERE
                       AZ_KIK_STEUER_KNZ    = 1
                   AND (   AZ_KIK_KAUSZ_KNZ = 0
                        OR AZ_KIK_KAUSZ_KNZ IS NULL)
                   AND DG_DEV_ISOCODE IS NOT NULL
           END-EXEC.
      *
           PERFORM XSQLRCFP-RETCODE-FUELLEN.
```

itestra
be excellent

Selected languages: PL/I

Remarks

- Developed in the 1960s
- Available on: z/OS, AS/400, Unix, Windows, ...

Objectives

- Domain-independent programming language
- So far: Focus on specific application scope
- Replace COBOL and Fortran on the Mainframe
- Very powerful preprocessor (subset of PL/I)
- **Powerful features regarding it's age**, but difficult to compile and not widespread
- **No reserved keywords**

```
IF IF = THEN THEN THEN = ELSE; ELSE ELSE = IF;
```

- Memory allocation,
  Pointer arithmetic
- Fixed-point und floating-point arithmetic
- Many data types, definition of data structures
- Exceptions (Conditions)
- Local variables, parameters (!)
- Built-in functions

itestra
be excellent

Selected languages:
C

Remarks

itestra
be excellent

- All executable code structured as **functions** → positive
- No **generic data structures** (except by using pointer arithmetic),
  Only **weak type safety**
- **Library concept** is based on macros and external tools (preprocessor, linker) – **not a language feature**!
  Thus (amongst others):
  - Limited ability to compile separately
  - Explicit inclusion order spec. and multiple inclusion prevention needed
  - Misuse possible (e.g. code in includes)
- No **objects/inheritance, limited encapsulation**
  → no powerful libraries besides basic funcs (e.g. no generic collection)
- **Side effects** → high risks for correctness & security

Consequences: see for example Obfuscated C Contest
(counter measure e.g. MISRA)

Hashmap example

itestra
be excellent

```
static inline khint_t kh_put_##name(kh_##name##_t *h, khkey_t key, int *ret) \
{                                                                             \
    khint_t x;                                                                \
    if (h->n_occupied >= h->upper_bound) {                                    \
        if (h->n_buckets > (h->size<<1)) kh_resize_##name(h, h->n_buckets - 1); \
        else kh_resize_##name(h, h->n_buckets + 1);                           \
    }                                                                         \
    {                                                                         \
        khint_t inc, k, i, site, last;                                        \
        x = site = h->n_buckets; k = __hash_func(key); i = k % h->n_buckets; \
        if (__ac_isempty(h->flags, i)) x = i;                                 \
        else {                                                                \
            inc = 1 + k % (h->n_buckets - 1); last = i;                       \
            while (!__ac_isempty(h->flags, i) && (__ac_isdel(h->flags, i) || !__ha
                if (__ac_isdel(h->flags, i)) site = i;                        \
                if (i + inc >= h->n_buckets) i = i + inc - h->n_buckets; \
                else i += inc;                                                \
                if (i == last) { x = site; break; }                           \
            }                                                                 \
            if (x == h->n_buckets) {                                          \
                if (__ac_isempty(h->flags, i) && site != h->n_buckets) x = site; \
                else x = i;                                                   \
            }                                                                 \
        }                                                                     \
    }                                                                         \
```

itestra
be excellent

Selected languages:
Java Script

Remarks

itestra
be excellent

- Weak type safety (including object orientation)
  → many errors at runtime
  → Creation of **TypeScript**
- Dynamic execution (eval) → **static analysis limited**, possible runtime errors
- Library concept based on includes

```
// Position area object. Represents a location and precision area.
var geoPositionArea = {
    area: L.circle([0.0, 0.0], 0.0),
    position: L.circleMarker([0.0, 0.0]),

    addToLayer: function(map) {
        map.addLayer(this.area);
        map.addLayer(this.position);
    },

    removeFromLayer: function(map) {
        map.removeLayer(this.area);
        map.removeLayer(this.position);
    },
```

Parameterization and Reuse

Parameterization

itestra
be excellent

Idea: **Extraction** of common parts, **parameterized** reuse

Missing e.g. in COBOL

Consequences

High duplication

Large SECTIONs and files (80.000 LoC max.)

Lots of technical code, no separation between technical and business code

Enormous overhead for the development process

Key feature of Java and other newer languages

User created functionality can be encapsulated and reused

Separation of concerns

Cloning: Example

Modul A

```
 EINMALBEITRAG.
*    BERECHNUNGEN FUER VERSICHERUNGEN GEGEN EINMALBEITRAG
*    NACH BEGINN DER ALTERSRENTE
 EB-RECHNUNGSGRUNDLAGEN.
      MOVE 0.040      TO IP IV.
      MOVE 0.96153846 TO VAUP VAUV.
      MOVE 0.33       TO ALPHAP.
*****COMPUTE ALTERSOK = X + N - 65.
      ADD  X  N  GIVING  ALTERSOK.
      SUBTRACT  65  FROM  ALTERSOK.
      IF ALTERSOK GREATER 0
*******COMPUTE ALPHAP = ALPHAP - ALTERSOK * 0.01.
        MULTIPLY  ALTERSOK  BY  0.01  GIVING  ZW8-1
        SUBTRACT  ZW8-1  FROM  ALPHAP.
        IF ALPHAP < 0 MOVE 0 TO ALPHAP.
      MOVE 0.015      TO BETA.
      MOVE 0          TO GAMMA.
      MOVE 0.005      TO GAMMA1.
      MOVE 0.015      TO GAMMA2.
      MOVE GAMMA2     TO GAMMAF.
      MOVE 1          TO ZILLKO.
      MOVE 1          TO RIKOAR.
      MOVE ALPHAP     TO AMZU.
      MOVE 0          TO ALPHAV STKZ STORNO PXN PXNZ PIXN PSTR AQ
                         TV TVZ TVK TVB TR.
      MOVE 70400      TO FKZ.

*****************************************************************
```

Modul B

```
 EINMALBEITRAG.
*    BERECHNUNGEN FUER VERSICHERUNGEN GEGEN EINMALBEITRAG
*    NACH BEGINN DER ALTERSRENTE
 EB-RECHNUNGSGRUNDLAGEN.
      MOVE 0.035      TO IP.
      MOVE 0.96618357 TO VAUP.
      MOVE 0.35       TO ALPHAP.
*****COMPUTE ALTERSOK = X + N - 65.
      ADD  X  N  GIVING  ALTERSOK.
      SUBTRACT  65  FROM  ALTERSOK.
      IF ALTERSOK GREATER 0
*******COMPUTE ALPHAP = ALPHAP - ALTERSOK * 0.01.
        MULTIPLY  ALTERSOK  BY  0.01  GIVING  ZW8-1
        SUBTRACT  ZW8-1  FROM  ALPHAP.

      MOVE 0.015      TO BETA.
      MOVE 0          TO GAMMA.
      MOVE 0.005      TO GAMMA1.
      MOVE 0.015      TO GAMMA2.
      MOVE GAMMA2     TO GAMMAF.
      MOVE 1          TO ZILLKO.
      MOVE 1          TO RIKOAR.
      MOVE ALPHAP     TO AMZU.
      MOVE 0          TO ALPHAV STKZ STORNO PXN PXNZ PIXN PSTR AQ
                         TV TVZ TVK TVB TR.


*****************************************************************
```

Possibilities for clone avoidance

itestra
be excellent

Inheritance, polymorphism

Extraction of common logic

```
abstract class LetterBase {

public void printLetter() {
   …
    printBankAccount();
   …
 }

 abstract void printBankAccount ();

}

…

class LetterWorld
   extends LetterBase {

  void printBankAccount() {
    String nation = searchNationDescription(nationCode);
    …
  }
```

The base class implements the common parts

The procedure printBankAccount can be **called**, although not **implemented** in the base class

Subclass **inherits** the common parts from the base class and implements only the **specific functionality**

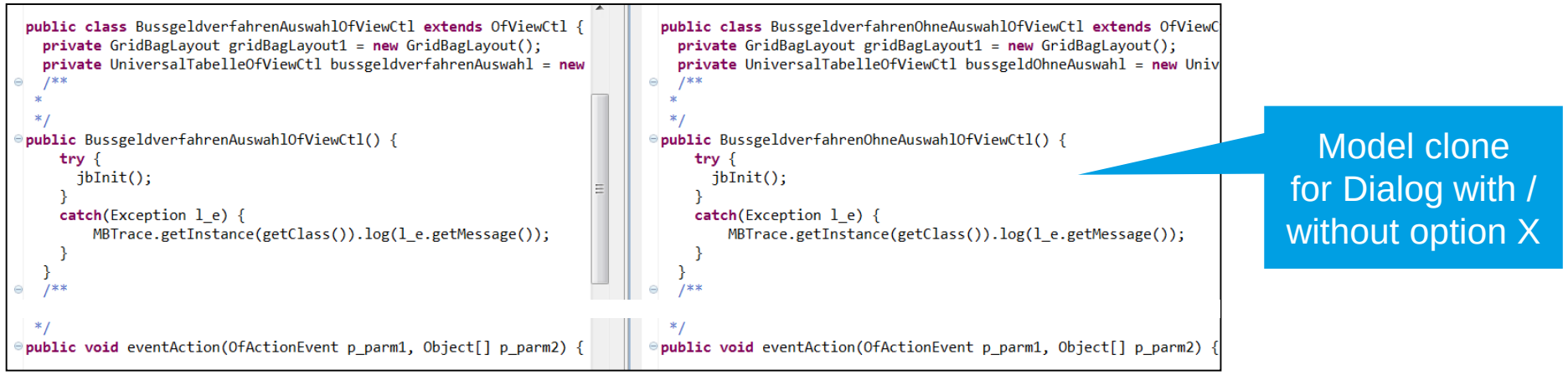
The Model-driven disaster (1)

Two projects with intensive MDA/MDD usage:

Parameterization in the model barely possible
→ **Cloning in the model** and the generated classes

Enormous amount of trivial, highly redundant code
(increases compile, deploy, debug costs etc.)

The worst observed productivity in Java yet!



```
public class BussgeldverfahrenAuswahlOfViewCtl extends OfViewCtl {
  private GridBagLayout gridBagLayout1 = new GridBagLayout();
  private UniversalTabelleOfViewCtl bussgeldverfahrenAuswahl = new
  /**
  *
  */
public BussgeldverfahrenAuswahlOfViewCtl() {
    try {
      jbInit();
    }
    catch(Exception l_e) {
        MBTrace.getInstance(getClass()).log(l_e.getMessage());
    }
  }
  /**

  */
public void eventAction(OfActionEvent p_parm1, Object[] p_parm2) {
```

```
public class BussgeldverfahrenOhneAuswahlOfViewCtl extends OfViewC
  private GridBagLayout gridBagLayout1 = new GridBagLayout();
  private UniversalTabelleOfViewCtl bussgeldOhneAuswahl = new Univ
  /**
  *
  */
public BussgeldverfahrenOhneAuswahlOfViewCtl() {
    try {
      jbInit();
    }
    catch(Exception l_e) {
        MBTrace.getInstance(getClass()).log(l_e.getMessage());
    }
  }
  /**

  */
public void eventAction(OfActionEvent p_parm1, Object[] p_parm2) {
```

Model clone
for Dialog with /
without option X

Inheritance/Interfaces

Allows the development of Frameworks, i.e.

- Generic solution
- Reuse of algorithms
- E.g.: HashMap, Persistence Layer

Dictionary<TKey, TValue>(Int32, IEqualityComparer<TKey>)	Initializes a new instance of the Dictionary<TKey, TValue> class that is empty, has the specified initial capacity, and uses the specified IEqualityComparer<T>.

Modeling of Parent-Child relations

- E.g. Domain model, UI elements

Libraries, COBOL example

itestra
be excellent

Nearly all „necessary“ base funcionality is integrated in laguage (keywords)

Hardly any COBOL libraries available

High effort for simple functionality (if not language integrated): e.g. String.trim() in COBOL:

- String being „inspected“ for spaces from left and right

```
MOVE FUNCTION REVERSE(STRING-TO-TRIM) TO INPUT-R.
INSPECT INPUT-R TALLYING COUNTER FOR LEADING SPACES.
COMPUTE INP-LEN = LENGTH OF INPUT-R - COUNTER.
MOVE INPUT(1:INP-LEN) TO R-TRIMED-STRING

INSPECT R-TRIMED-STRING TALLYING COUNTER FOR LEADING SPACES.
COMPUTE INP-LEN = LENGTH OF R-TRIMED-STRING - COUNTER.
MOVE R-TRIMED-STRING(1:INP-LEN) TO TRIMMED-STRING
```

Special chars like tab are not matched!

Libraries, Java example

itestra
be excellent

In Java

Easy packaging of modules in jar files,
easy integration and execution by Java-VM (no recompile).

OO allows for powerful, generic frameworks

Consequences

Large collection of useful libraries, frameworks, …

E.g. Apache commons, Guava, JSF,…

```
} finally {
    IOUtils.closeQuietly(bw);
}
```

Any Stream instance!

Visibilities / Encapsulation

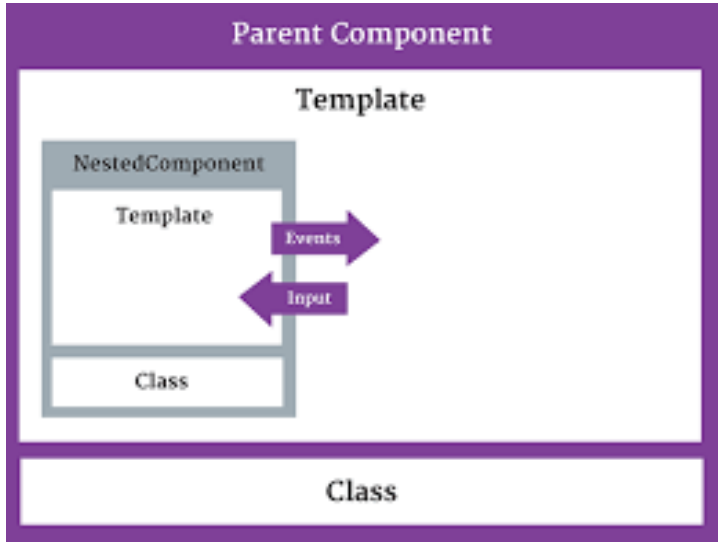
itestra
be excellent

Java

public / protected / private for methods, variables and classes

How about larger components?
→ no established, language-integrated solutions besides “package” level, further possibilities include

- different Eclipse projects
- OSGI
- Java 9 Jigsaw



COBOL

All variables are program-global

Example for large information system (~800kSLoC):
more than 5.000 variable accessible in nearly all programs

Complete static analysis of access impossible due to memory overlay, …

itestra
be excellent

Program generators, proprietary languages and click-tools

Idea

itestra
be excellent

Assumption: Requirements are already specified in a formal way (diagrams, tables)

Thus: Generate code directly from the requirements

```
input employee

output  total;

when Salary calculator
then
   total = employee.Salary;
   /* bonus */
   total += (employee.Salary * 0.10);
   /* commission */
   total += (employee.Salary * 0.30);
end
```

Code generation exists since the 70s

~2000: Hype „Model Driven Architecture“ (MDA), „Domain Specific Languages“ (DSLs)

- Modeling of business objects and processes with UML, subsequent multi-stage generation, promoted by OMG
- Graphical or textual DSLs, Oriented towards a business domain, e.g. insurance
- Amongst others: support in Visual Studio (Domain-Specific Language Tools) and Eclipse (EMF)

Program generators, MDA, DSLs

itestra
be excellent

Code generation, MDA, DSLs are no „Silver Bullets“

- No technology is able to completely reproduce **complicated business logic** within a **simplified model**
- A model language, powerful enough to describe all the specific cases (and these do occur!), is usually **too complicated itself**

Generation is almost obsolete in powerful, modern languages (Java) (instead: skillful usage of OO-concepts, AOP etc.)

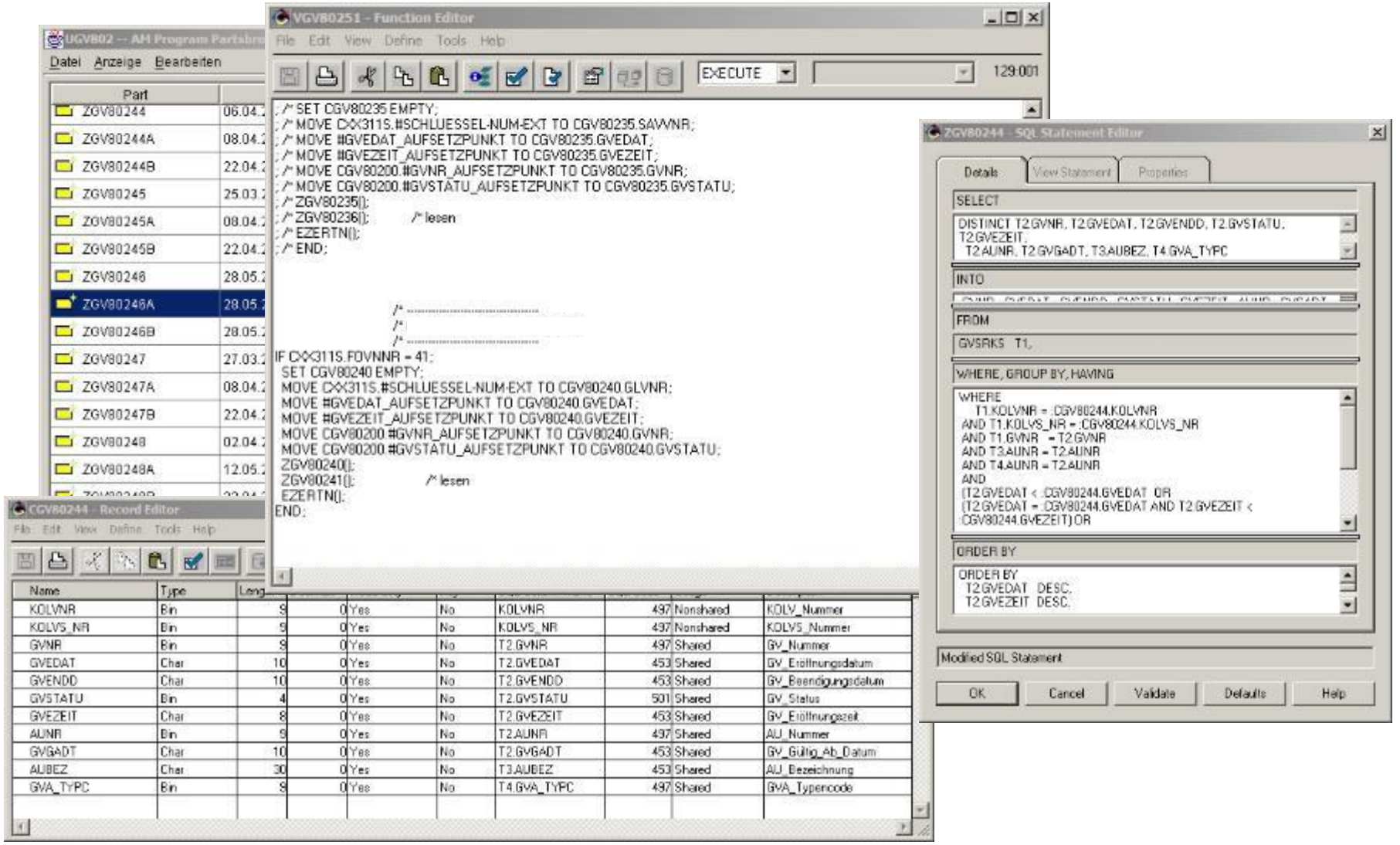
- Generation is primarily a relict from COBOL-times!
- Exceptions: Adapter for fixed interfaces, very high performance (e.g. parser)

„Clicking-up“ of program logic is often more time-consuming than typing, full feature search, refactorings, etc. are often not even possible!

Besides that:

- Generation should always be automated and reproducible
  Otherwise: Instant Legacy (efficient change with one-shot is impossible)
- The lifecycle of the generator is essential (open source?)

VisualAge generator



The Model-driven Disaster (2)

itestra
be excellent

Two projects with intensive MDA/MDD usage:

Basic functionality not available in the tool:

- merge
- Diff
- concurrent editing

→ Editing conflicts between the developers

→ Weekly (!) update because of slow code generation

→ In the meantime: manual change of the generated code

Conclusion

Conclusion

itestra
be excellent

Older programming languages are in practice still very important **(and exciting!)** for computer scientists

These languages often miss essential concepts from today's point of view

Particular relevance is attributed to the concepts of:

- **Composition** and **(parameterized)** reuse of components
- Definition of **visibilities**/scopes

Type safety and other approaches for **error avoidance** (e.g. Garbage Collection)

„Modern" products and technologies (rule engines, MDA, scripting languages, …) should be carefully examined to this effect!

New ways of programming?

itestra
be excellent

```
@Constraint(validatedBy = ValidAmountOfDoorsValidator.class)
@Target({ElementType.METHOD, ElementType.FIELD, ElementType.ANNOTATION_TYPE,
ElementType.CONSTRUCTOR,  ElementType.PARAMETER, ElementType.TYPE_USE})
@Retention(RetentionPolicy.RUNTIME)
@interface ValidAmountOfDoors {
    String message() default
"{ibiss.common.example.model.constraints.ValidAmountOfDoors.message}";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}

@ValidAmountOfDoors
class CarBE { /*...*/ }

class ValidAmountOfDoorsValidator implements ConstraintValidator<ValidAmountOfDoors,
CarBE> {
@Override
    public boolean isValid(final CarBE car, final ConstraintValidatorContext context) {
        context.buildConstraintViolationWithTemplate(
context.getDefaultConstraintMessageTemplate() )
                .addPropertyNode( CarBE.PROPERTY_AMOUNT_DOORS )
                .addConstraintViolation()
                .disableDefaultConstraintViolation();
        final CarTypeEnum carType = car.getCarType();
        final int amntDoors = car.getAmountOfDoors();
        return CAR_TYPE_TO_DOORS.get(carType).contains(amntDoors);
    }
```

New ways of programming?

itestra
be excellent

orElseThrow

We can indicate the `Optional` to throw an exception in case its value is null:

```
1 Optional optionalCarNull = Optional.ofNullable( carNull );
2 optionalCarNull.orElseThrow( IllegalStateException::new );
```

ifPresent

And also the option to execute actions directly when the value is present, in combination with Lambdas:

```
1 Optional stringToUse = Optional.of( "optional is there" );
2 stringToUse.ifPresent( System.out::println );
```

Optimization Examples

Hashmap in COBOL (1)

Column-wise storage of stock rates in a DB table

Datum	Fonds 1				Fonds 2				...
	FOKZN	BWFRUECK	ERT	BWFERT	FOKZN	BWFRUECK	ERT	BWFERT	...
1.10.2010	AA	4,38	0,00	0,00	AH	12,88	0,00	0,00	...
2.10.2010	AA	4,12	0,00	0,00	AH	...	...	...	...
3.10.2010	AA	...	...	...	AH	...	...	...	...

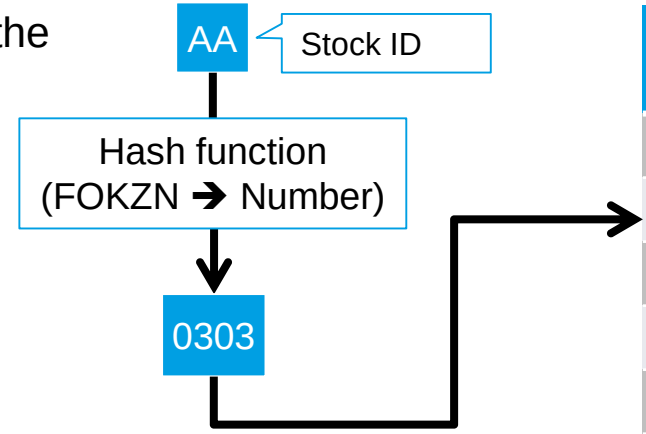
Search over

- Date (Index ✓)
- FOKZN (Stock ID)

Current approach:

- Read the line for the searched date
- Linear search for FOKZN in the (not sorted) data structure

Alternative:



1x calculate: Hashtable

Index/ Hash value	Index in Stock-rate-table.
0303	01
...	...
7377	126
...	...
8321	18

Stock-rate-table

Index	Fonds
1	AA
2	AH
3	AD
4	AC
...	...

Hashmap in COBOL (2)

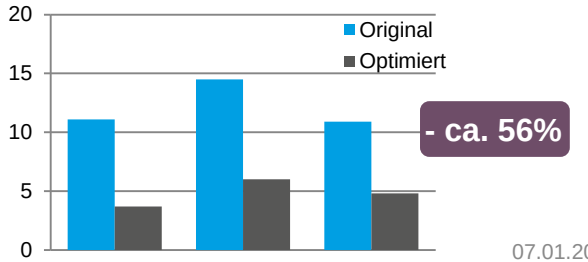
```
 01 HASH-TABLE-FIELDS.
*
    02 HASH-TABLE-DATA.
       COPY DGLMB680.
    02 HASH-TABLE REDEFINES HASH-TABLE-DATA.
       03 TARGET-INDEX          PIC 9(3)    OCCURS 10000.

 ...

MOVE ZW-SEARCH-FOKZN TO ZW-HASH-INPUT
PERFORM HASHWERT
MOVE TARGET-INDEX IN HASH-TABLE(IND-HASH-INDEX)
  TO ZW-RESULT-INDEX

 ...

 HASHWERT SECTION.
*
     COMPUTE PART1 IN IND-HASH-INDEX-PARTS =
         FUNCTION ord(CHAR1 IN ZW-HASH-INPUT) - HF-OFFSET-EBCDIC
*
     COMPUTE PART2 IN IND-HASH-INDEX-PARTS =
         FUNCTION ord(CHAR2 IN ZW-HASH-INPUT) - HF-OFFSET-EBCDIC

     CONTINUE.
*
```

Generated hashtable: once, but repeatable

Hash value / Index	Index in Fonds-S.
0303	01
...	...
7377	126
...	...
8321	18

Calculate hash value and determine index for the searched input

Simple, but effective hash function



Informatiker/innen gefragt!

- Festanstellung
- Werkstudententätigkeit
- Praktikum
- Abschlussarbeiten (Bachelor & Master)

Bewerben Sie sich!
jobs@itestra.de

Kontakt

itestra GmbH
Destouchesstraße 68
80796 München

E-Mail: info@itestra.com
Tel.: +49 89 381570-110
Fax: +49 89 381570-119

itestra
be excellent