**Department of Information Systems and Operations Management**

# Procedural and Object-oriented Programming 1

## Statement, Routine (Procedure, Function), "Stem"-Variable

**Business Programming 1** | **Business Programming 2**


| | | | | |
|---|---|---|---|---|
| Basics, Parsing | Commands, APIs | Window-Automatisation, Web-Scripting | Security, Debugging | Graphical User Interfaces (GUI), Sockets, ... |



Prof. Rony G. Flatscher

# Labels

- Identifier, followed by a colon (**:**)
- Serves as a target for an internal routine
  - **CALL**-statements (invoking procedures)
  - Function invocations
  - **SIGNAL**-statements (like a "**GOTO**" instruction in other languages)
  - Exception handling (**SIGNAL ON** resp. **CALL ON**)

```
DO i = 1 TO 3
    SAY "Oho!" i
    IF i = 1 THEN SIGNAL fin
END
fin : SAY "C'est la fin!"
```

**Output:**

```
Oho! 1
C'est la fin!
```



*Cf. rexxref.pdf (1.12.3. Labels)*

# Internal Routines, 1

- Grouping of statements which repeatedly get executed by different parts in a program
- Start with a label
- Invocation
  - **CALL** label
  - Statements after the label get executed
  - The **RETURN**-statement returns control (to the statement immediately following the **CALL**-statement)
- A „routine“ may also be called „procedure“


Prof. Rony G. Flatscher

# Internal Routines, 2

```
/* A Rexx-Programm with an internal routine named 'TIMESTAMP' */
CALL TimeStamp       /* call a subroutine   */
CALL SysSleep 5      /* sleep 5 seconds     */
CALL TimeStamp       /* call a subroutine   */
EXIT                 /* exit program        */

TimeStamp :          /* label               */
   SAY "It is rather late ..."
   RETURN
```

**Output:**

```
It is rather late ...
It is rather late ...
```



*Cf. rexxref.pdf (7.2. Functions and Subroutines)*

# Functions, 1

- Routines that return a value ("function value") to the caller via the ***RETURN***-statement
- Invocation
  - Variant 1
    - Invocation: the name (label), immediately followed by a left and right parenthesis
    - The return value ("function value") replaces the invocation

```
today = DATE()
```

  - Variant 2
    - Invocation: CALL followed by the name (label) of the routine
    - Interpreter stores the return value in the special variable RESULT

```
CALL DATE
today = result
```


Prof. Rony G. Flatscher

# Functions, 2

```
/* A Rexx-Programm with an internal function named 'TIMESTAMP' */
SAY  TimeStamp()     /* function call          */
CALL SysSleep 5      /* sleep 5 seconds        */
CALL TimeStamp       /* procedure call         */
SAY result           /* show function value    */
EXIT                 /* exit program           */

TimeStamp :          /* label                  */
    RETURN "It is rather late ..."
```

**Output:**

```
It is rather late ...
It is rather late ...
```



*Cf. rexxref.pdf (2.3. CALL)*

# Rexx Variables Used By Rexx

- The Rexx interpreter may set the value of the following variables
  - **RESULT**
    - If a routine gets invoked with **CALL** and returns a value with its **RETURN** statement then the Rexx interpreter stores that returned value in a variable named **RESULT**
  - **RC**
    - Stores the "**r**eturn **c**ode" that the external command returned: a value of **0** (zero) indicates that the external command completed without any errors or failures
  - **SIGL**
    - Stores the line number which contains the **CALL** or **SIGNAL** statement that caused the jump to the routine ("**sig**nal **l**ine")



*Cf. rexxref.pdf (11.3.3. RC, 11.3.4. SIGL)*

# All Functions of REXX

- Rexx provides the following built-in functions ("BIF"), some are specific to **strings** or **filenames**:

| | | | | |
|---|---|---|---|---|
| ABBREV() | CHARS() | FORM() | RANDOM() | TRUNC() |
| ABS() | COMPARE() | FORMAT() | REVERSE() | VALUE() |
| ADDRESS() | COPIES() | FUZZ() | RIGHT() | VAR() |
| ARG() | COUNTSR() | INSERT() | SETLOCAL() | VERIFY() |
| B2X() | D2C() | LASTPOS() | SIGN() | WORD() |
| BEEP() | D2X() | LEFT() | SOURCELINE() | WORDINDEX() |
| BITAND() | DATATYPE() | LENGTH() | SPACE() | WORDLENGTH() |
| BITOR() | DATE() | LINEIN() | STREAM() | WORDPOS() |
| BITXOR() | DELSTR() | LINEOUT() | STRIP() | WORDS() |
| C2D() | DELWORD() | LINES() | SUBSTR() | X2B() |
| C2X() | DIGITS() | MAX() | SUBWORD() | X2C() |
| CENTER() | DIRECTORY() | MIN() | SYMBOL() | XRANGE() |
| CHANGESTR() | ENDLOCAL() | OVERLAY() | TIME() | |
| CHARIN() | ERRORTEXT() | POS() | TRACE() | |
| CHAROUT() | FILESPEC() | QUEUED() | TRANSLATE() | |



*Cf. rexxref.pdf (7.4. Built-in Functions (BIF))*

# External Rexx Function Packages

- Standardised Interfaces to and from Rexx
  - Loading an external Rexx function package from Rexx, an example

```
IF RxFuncQuery("SysLoadFuncs") THEN DO
    CALL RxFuncAdd "SysLoadFuncs","RexxUtil","SysLoadFuncs"
    CALL SysLoadFuncs          /* no quotes! */
END
```

- Function packages, which supply new functions to Rexx that are not part of the language, an example
  - Mark Hessling's "RexxSQL" – Direct access to the most important relational database management systems (DB2, Oracle, SQL-Server, MySQL, etc.)
    - https://sourceforge.net/projects/rexxsql/

# Function Package "RexxUtil" (Excerpt)

- Contains more than 60 "useful" functions, some of which are only available for **Windows** or **Unix** operating systems.

```
RxMessageBox
RxWinExec
SysAddRexxMacro
SysBootDrive
SysClearRexxMacroSpace
SysCls
SysCreatePipe
SysCurPos
SysCurState
SysDriveInfo
SysDriveMap
SysDropRexxMacro
SysDumpVariables
SysFileCopy
SysFileDelete
SysFileExists
SysFileMove
SysFileSearch
SysFileSystemType
SysFileTree
SysFork
SysFormatMessage
SysFromUnicode
SysGetErrorText
SysGetFileDateTime
SysGetKey
SysGetMessage
SysGetMessageX
SysGetShortPathName
SysIni
SysIsFile
SysIsFileCompressed
SysIsFileDirectory
SysIsFileEncrypted
SysIsFileLink
SysIsFileNotContentIndexed
SysIsFileOffline
SysIsFileSparse
...
```



*Cf. rexxref.pdf (8. Rexx Utilities (RexxUtil))*

# Searching for Routines, 1

- Searching order for routines

  1a) Internal routines (in the program itself)

  1b) Routines defined as directives (in the program itself)

  2) The built-in Rexx routines ("built-in functions (BIF)")

  3) External routines (e.g. Rexx programs)

- It is possible to use the names of the language built-in functions
  - Overlay the respective routines
  - The overlayed built-in function can always be invoked by **enclosing its uppercased name in quotes**!

# Searching for Routines, 2

```
/* */
SAY date()          /* invoke self programmed function below  */
SAY "DATE"()        /* invoke the Rexx built-in function      */
EXIT
DATE :              /* "DATE" is in effect a Rexx function !  */
    RETURN "Date(), self programmed!"
```

Output (if run on 2059-05-20):

```
Date(), self programmed!
20 May 2059
```

# Scopes, 1

- Define which variables and labels are seen in which part of a Rexx program
  - By default all variables in a program are globally visible/accessible, they belong to the **same scope**
  - Labels in a program are always global
- If the keyword instruction **PROCEDURE** follows a label, then a new ("local") scope will be created for it
  - Should there be a need to access variables outside a local scope, then one must use the **EXPOSE** subkeyword of the **PROCEDURE**-Statement denoting those variable names.



*Cf. rexxref.pdf (2.19. PROCEDURE)*

# Scopes, 2

```
/* */
a = 1
b = 2
SAY "a=" a "b=" b
CALL calc
SAY "a=" a "b=" b
EXIT
calc :
    a = a * 2
    b = b * 3 / 4
    RETURN
```

**Output:**

```
a= 1 b= 2
a= 2 b= 1.5
```

# Scopes, 3

```
/* */
a = 1
b = 2
SAY "a=" a "b=" b
CALL calc
SAY "a=" a "b=" b
EXIT
calc: PROCEDURE /* no access to global "a" und "b" !  */
    a = 5       /* hence, "a" must be defined locally */
    b = 6       /* hence, "b" must be defined locally */
    a = a * 2
    b = b * 3 / 4
    RETURN
```

**Output:**

```
a= 1 b= 2
a= 1 b= 2
```



*Cf. rexxref.pdf (2.19. PROCEDURE)*

# Scopes, 4

```
/* */
a = 1
b = 2
SAY "a=" a "b=" b
CALL calc
SAY "a=" a "b=" b
EXIT
calc: PROCEDURE EXPOSE b /* no access to "a", but to "b" !     */
    a = 5                /* hence, "a" must be defined locally */
    a = a * 2
    b = b * 3 / 4
    RETURN
```

**Output:**

```
a= 1 b= 2
a= 1 b= 1.5
```



*Cf. rexxref.pdf (2.19. PROCEDURE)*

# "Stem" Variable (Associative Arrays), 1

- Identifier contains one or more **dots**
- The sequence of characters from the beginning up to and including the first dot is called **stem**

```
a.n                       = "aha"
a.OnE                     = 1
a.1                       = "Richard"
Austria.Tyrol             = 750000
Austria.Tyrol.Innsbruck = 135000
SAY a.1 a.n a.OnE
SAY Austria.Tyrol
```

**Output:**

```
Richard aha 1
750000
```



*Cf. rexxref.pdf (1.13.4. Stems)*

# "Stem" Variable (Associative Arrays), 2

- Some RexxUtil functions (e.g. *SysFileTree()*) use a convention, which mandates that after the dot only integer numbers be used
  - **stem.0** – stores the total number of "elements"; this allows iterating over all stem entries starting with "1" going up to the number stored in stem.0

```
file.1 = "max.doc"
file.2 = "moritz.doc"
file.0 = 2              /* maximum number of "elements"  */
DO i=1 TO file.0
   SAY file.i           /* "i" is also called "index"    */
END
```

**Output:**

```
max.doc
moritz.doc
```



*Cf. rexxref.pdf (1.13.4. Stems)*

# PARSE by Blank, 1

- **PARSE** statements allow parsing a string and assigning (parts of it) to Rexx variables in one step

```
text = "  Stiegler   Seppl   Stumm     Zillertal/Tirol"
PARSE VAR text famName firstName rest
SAY famName
SAY firstName
SAY rest
```

**Output:**

```
Stiegler
Seppl
    Stumm   Zillertal/Tirol
```



*Cf. rexxref.pdf (9. Parsing)*

# PARSE by Blank, 2

```
ruler = COPIES("1234+6789|", 5)
text = "  Stiegler   Seppl   Stumm     Zillertal/Tirol"
PARSE VAR text famName firstName rest
SAY ruler; SAY text ; SAY
SAY pp(famName); SAY pp(firstName)
SAY pp(ruler);   SAY pp(rest)
EXIT
PP : RETURN "[" || ARG(1) || "]" -- "pretty print" ;)
```

**Output:**

```
1234+6789|1234+6789|1234+6789|1234+6789|1234+6789|
  Stiegler   Seppl   Stumm     Zillertal/Tirol

[Stiegler]
[Seppl]
[1234+6789|1234+6789|1234+6789|1234+6789|1234+6789|]
[  Stumm     Zillertal/Tirol]
```



*Cf. rexxref.pdf (9. Parsing)*

# PARSE by Pattern, 3

```
      /*        10        20        30        40
        1234+6789|1234+6789|1234+6789|1234+6789| */
text = "  Ruaniger Annelle       Stumm   Zillertal / Tirol "
PARSE VAR text before "/" after
SAY pp(before)
SAY pp(after)
EXIT
PP : RETURN "[" || ARG(1) || "]" -- "pretty print" ;)
```

**Output:**

```
[   Ruaniger Annelle     Stumm   Zillertal ]
[  Tirol ]
```



*Cf. rexxref.pdf (9. Parsing)*

# PARSE by Pattern, 4

```
pattern = "/"
      /*        10        20        30        40
        1234+6789|1234+6789|1234+6789|1234+6789| */
text = "  Ruaniger Annelle        Stumm    Zillertal / Tirol "
PARSE VAR text before (pattern) after
SAY pp(before)
SAY pp(after)
EXIT
PP : RETURN "[" || ARG(1) || "]" -- "pretty print" ;)
```

**Output:**

```
[   Ruaniger Annelle     Stumm    Zillertal ]
[  Tirol ]
```



*Cf. rexxref.pdf (9. Parsing)*

# PARSE by Position, Length, Blank, 5

```
      /*        10        20        30        40
        1234+6789|1234+6789|1234+6789|1234+6789| */
text = "  Ruaniger Annelle        Stumm    Zillertal / Tirol "
PARSE VAR text 3 famName +8 12 firstName city .
SAY pp(famName)
SAY pp(firstName)
SAY pp(city)
EXIT
PP : RETURN "[" || ARG(1) || "]" -- "pretty print" ;)
```

**Output:**

```
[Ruaniger]
[Annelle]
[Stumm]
```



*Cf. rexxref.pdf (9. Parsing)*

# PARSE, 6

```
text = "Sattler;Cilli;Stumm;Zillertal/Tirol"
PARSE VAR text famName ";" firstName ";" city
SAY pp(famName)
SAY pp(firstName)
SAY pp(city)
EXIT
PP : RETURN "[" || ARG(1) || "]" -- "pretty print" ;)
```

**Output:**

```
[Sattler]
[Cilli]
[Stumm;Zillertal/Tirol]
```



*Cf. rexxref.pdf (9. Parsing)*

# PARSE, 7

```
text = ";Sattler;Cilli;Stumm;Zillertal/Tirol"
PARSE VAR text 1 a +1 famName (a) firstName (a) city (a) .
SAY pp(famName)
SAY pp(firstName)
SAY pp(city)
EXIT
PP : RETURN "[" || ARG(1) || "]" -- "pretty print" ;)
```

**Output:**

```
[Sattler]
[Cilli]
[Stumm]
```



*Cf. rexxref.pdf (9. Parsing)*

# PARSE VALUE ... WITH ...

- **PARSE VALUE expression WITH pattern**
  - Makes it possible to immediately parse the result of an **expression**

    Example:

```
PARSE VALUE date() WITH day month year
say "year:" year "month:" month "day:" day
```

**Output** (assuming date() returned the string `"20 May 2059"`)**:**

```
year: 2059 month: May day: 20
```



*Cf. rexxref.pdf (9. Parsing)*

# Input from Keyboard (“STDIN”)

- **PARSE PULL** statements allow parsing a string read from the keyboard and assigning (parts of it) to Rexx variables in one step

```
SAY "1. What is your name?"   /* Keyboard input: "Max"      */
PARSE PULL name
SAY "Your name is:" pp(name)
SAY "2. What is your name?"   /* Keyboard input: "moritz"   */
PULL name
SAY "Your name is:" pp(name)
EXIT
PP : RETURN "[" || ARG(1) || "]" -- "pretty print" ;)
```

**Output:**

```
1. What is your name?
Max
Your name is: [Max]
2. What is your name?
moritz
Your name is: [MORITZ]
```



*Cf. rexxref.pdf (2.18. PARSE)*

# Retrieving Arguments PARSE ARG

- **PARSE ARG** allows to fetch argument values

```
a = 1; b = 2
SAY "a=" a "b=" b
CALL calc a , b
SAY "a=" a "b=" b
EXIT
calc: PROCEDURE      /* caller's variables "a" and "b" not visible !       */
   PARSE ARG a , b   /* fetch arguments and assign them to local variables */
   SAY "calc: a=" a "b=" b
   a = a * 2
   b = b * 3 / 4
   SAY "calc: a=" a "b=" b
   RETURN
```

**Output:**

```
a= 1 b= 2
calc: a= 1 b= 2
calc: a= 2 b= 1.5
a= 1 b= 2
```



*Cf. rexxref.pdf (2.18. PARSE)*