PL/I – REXX Comparison Introduction

REXX (née REX) was modeled after PL/I, but has significantly deviated from it. This remains true with ANSI Rexx and ooRexx. In this presentation, bolding indicates literal text of the language, e.g. keywords, but not generic modifiers, e.g., “**DO** statement”; italics indicates a variable, e.g., “*expression*”, and the PL/I features are those of the OS PL/I “Optimizing” and Checkout compilers, which were current at the time. Also, despite a few references to NetRexx and ooRexx syntax, they are mostly beyond the scope of this presentation.

BIF is Built In Function.

PL/I – REXX Comparison Guesses

I am not aware of any paper explaining what lead to the differences from PL/I. However, there are several factors that would naïvely seem to require accommodation in the syntax.

PL/I has no reserved words and does not allow a statement to be a bare expression. Rexx allows a statement to be a bare expression.

PL/I operators are explicit; Rexx has implied abutment operators.

PL/I requires every statement to end with a semicolon; Rexx allows many semicolons to be omitted

PL/I – REXX Comparison Topics

The differences from PL/I include

- Abbreviations
- Block structure
- Bounds checking
- Condition handling
- Continuation
- Control structures
- Data types
- Name scoping
- Storage allocation

PL/I – REXX Comparison
Abbreviations

In PL/I, many keywords may be abbreviated, e.g., **PROC** for **PROCEDURE**. In REXX, keywords are spelled out.

PL/I – REXX Comparison Block structure

In PL/I, code is structured into blocks beginning with **BEGIN** or **PROCEDURE** (**PROC**), and groups beginning with **DO**, and ending with **END**. It is not possible to explicitly transfer to code inside a block or group from outside of it, although a subroutine can signal a condition that is handled within the block that called it. The term unit may refer to a single statement, a block or a group.

Although REXX does have DO loops, it has no true block structure.

PL/I – REXX Comparison

Block structure Example

```
FOO: DO I=1 TO BAR;
     X(I)=0;
     ...
     BAZ: ITERATE FOO;
     ...
     END;
GO TO BAZ /* Invalid */;
```

PL/I – REXX Comparison
Bounds Checking

When it is enabled, PL/I raises **STRINGRANGE** for attempted access beyond the size of a string.

When it is enabled, PL/I raises **SUBSCRIPTRANGE** for attempts to access an array elements with subscripts outside of the bounds declarded for the array.

PL/I – REXX Comparison Bounds Checking Example

```
(STRINGRANGE,SUBSCRIPTRANGE):

MYLABEL: BEGIN;

   DCL   NAME  CHAR(50),

         S(10) CHAR(20);

   ...

   S(I) = SUBSTR(NAME,J,K);

   /* Raise SUBSCRIPTRANGE

      if I<1 or I>10      */

   /* Raise STRINGRANGE

      if J<1 or J+K>21    */

   ...

   END;
```

PL/I – REXX Comparison Condition Handling

- PL/I has dynamic establishment of condition handling blocks within a lexical context; A **GOTO** out of the condition handler pops the control stack appropriately.
- REXX has dynamic establishment of condition handling at arbitrary labels, including labels inside a **DO** loop or unrelated procedure. A SIGNAL flushes the stack rather than popping it appropriately.

PL/I – REXX Comparison Condition Handling Example

PL/I	REXX
ON *condition* **GO TO** FOO;	**SIGNAL** , **ON** *condition* , **NAME** FOO
ON *condition* **CALL** FOO;	**SIGNAL ON** *condition* **CALL** FOO

PL/I – REXX Comparison Continuation

- In PL/I, every statement must end in a semicolon, so there is no need to indicate continuation lines.
- In REXX, a trailing comma serves as a continuation indicator, continuation is often implicit, and a trailing semicolon is often implicit. This allows for a slightly terser coding style, but it is a common error to forget to add a comma for continuation after a trailing comma.

PL/I – REXX Comparison Continuation Example

PL/I	Rexx

`A = B*C`
`+ C*D;`
	`A=B*C ,`
	`+D*E`

PL/I – Rexx Comparison Control Structures

- **BEGIN**
- **CALL**
- Condition prefix
- **DO**
- Function reference
- **GOTO**
- **IF**
- **ITERATE**
- Label**:**
- **LEAVE**
- **ON**
- **OTHERWISE**
- **PROCEDURE**
- **RETURN**
- **SELECT**
- **SIGNAL**
- **WHEN**

PL/I – REXX Comparison Control structures **BEGIN**

- In PL/I, a BEGIN statement establishes a lexical block that can declare local variables and procedures invisible from outside the block. PL/I automatically pops the stack on block exit, freeing any automatic variables.
- REXX has no equivalent.

PL/I – REXX Comparison Control structures **CALL**

- in PL/I a **CALL** statement invokes an external procedure or an internal block headed by a **PROCEDURE** statement. The keyword **CALL** is followed by a procedure name and an optional parenthesized list of arguments.
- In REXX the procedure name can be any label, and the arguments are not enclosed in parentheses.

PL/I – REXX Comparison Control structures Condition prefix

- In PL/I, A prefix of the form (condition): or (**NO**condition): enables or disables condition in a single unit, which may be a block.
- REXX has no equivalent.

PL/I – REXX Comparison Control structures **DO**

- The DO statement in PL/I uses the label of the DO as a loop designator on END, ITERATE and LEAVE
- The DO statement in REXX uses the control variable of the DO as a loop designator on END, ITERATE and LEAVE
- ooRexx 5.0 adds an optional LABEL parameter to the DO statement.

PL/I – REXX Comparison
Control structures
Function References

- In PL/I, an internal function invocation must refer to a **PROCEDURE** statement and may optionally be followed by a parenthesized list of arguments.
- In REXX, the syntax is identical but the name may be an arbitrary label.

PL/I – REXX Comparison Control structures **GOTO**

- In PL/I a label is only visible within its enclosing block and has an associated stack frame and a GOTO pops the stack enough to leave the stack frame associated with the label as the top of stack.

- In REXX a label is visible from the entire source file and simply marks a location in the source code; there is no GOTO statement. It is a common misconception that SIGNAL is a GOTO, but the semantics are very different.

PL/I – REXX Comparison
Control structures
ITERATE

- In PL/I, the statement has an optional loop label.
- In REXX, the statement has an optional control variable
- In ooRexx 5.0, the statement has either an optional control variable or an optional label specified on the **LABEL** operand of the corresponding **DO**.

PL/I – REXX Comparison Control structures **LEAVE**

- In PL/I, the statement has an optional loop label.
- In REXX, the statement has an optional control variable
- In ooRexx 5.0, the statement has either an optional control variable or an optional label specified on the **LABEL** operand of the corresponding **DO**.

PL/I – REXX Comparison Control structures **ON**

- In PL/I, the **ON** statement activates an ON unit (exception handler). The statement following **ON** *condition* is normally either a block or a GOTO
- The closest equivalent in REXX is **SIGNAL ON** *condition* **NAME** *label*.
- OoRexx adds SIGNAL ON condition CALL label.

PL/I – Rexx Comparison Control structures **OTHERWISE**

- In PL/I, **OTHERWISE** is an optional statement withing a select group. A single unit, which may be a block or group, follows the keyword.
- In Rexx, **OTHERWISE** may be followed by multiple statements; the **END** closing the **SELECT** also closes the **OTHERWISE**.

PL/I – REXX Comparison Control structures **PROCEDURE**

- In PL/I, a procedure always begin with an optional condition prefix, followed by a label and a **PROCEDURE** keyword (either spelled out or abbreviated), followed by an optional parenthesized list of parameters.
- In REXX a procedure does not require a PROCEDURE statement unless variables need to be hidden. There is no declaration of parameters; the ARG BIF and the PARSE ARG statement can be used to access arguments,.
- In ooRexx arguments are also available via the USE ARG statement.

PL/I – REXX Comparison
Control structures
RETURN

The **RETURN** statement is basically the same in PL/I and Rexx.

PL/I – REXX Comparison
Control structures
SELECT

- Rexx does not allow an expression after **SELECT**, although ooRexx has an equivalent.
- PL/I has a parenthesized list of expressions after WHEN, while Rexx has does not require parentheses.
- PL/I does not allow THEN after WHEN, while Rexx requires it.
- PL/I only accepts a single block, group or statement after OTHERWISE while Rexx allows multiple statements.
- In PL/I there is no terminal **END** for **SELECT**, while in Rexx it is required.

PL/I-Rexx Comparison
Control Structures
SELECT
Example 1

PL/I

```
SELECT (NAME);
   WHEN('TOM')
      N=1;
   WHEN('DICK')
      N=2;
   WHEN('HARRY')
      DO;
         N=3;
         END;
   OTHERWISE;
      N=4;
```

Rexx

```
/* ooRexx only */
SELECT CASE NAME
   WHEN('TOM')
      THEN N=1
   WHEN('DICK')
      THEN N=2
   WHEN('HARRY')
      THEN DO
         N=3
         END
   OTHERWISE
      N=4
   END
```

PL/I-Rexx Comparison
Control Structures
SELECT
Example 2

PL/I

```
SELECT;
  WHEN (FOO=1)
    NAME='TOM';
  WHEN (FOO=2)
    NAME='DICK';
  WHEN (FOO=3) DO;
    NAME='HARRY';
    END;
  OTHERWISE DO;
    NAME='PLONI';
    FLAG='!';
    END;
```

Rexx

```
SELECT
  WHEN FOO=1 THEN
    NAME='TOM'
  WHEN FOO=2 THEN
    NAME='DICK'
  WHEN FOO=3
  THEN DO
    NAME='HARRY'
    END
  OTHERWISE
    NAME='PLONI'
    FLAG='!'
  END
```

PL/I-Rexx Comparison Control Structures **SIGNAL**

- In PL/I, **SIGNAL** raises a condition.
- In Rexx, there are several forms of **SIGNAL**, used to raise a condition, establish a condition handler and deactivate a condition handler.

PL.I – Rexx Comparison Control structures **WHEN**

- In PL/I a **WHEN** statement has a list of expressions in parentheses that are matched against the expression on the corresponding **SELECT**; if the **SELECT** has no expression then PL/I converts the **WHEN** expressions to BIT. The associated code unit follows directly after the expressions with no separating semicolon.
- In Rexx a **WHEN** statement has a list of expressions that evaluate to 0 or 1. The expression list is followed by **THEN**.

PL/I – REXX Comparison
Data types

- In PL/I, variables have a type either explicitly declared or inferred from their names.
- In Rexx all variables are strings.
- In ooRexx, all variables are objects.

PL/I – REXX Comparison Data types Arrays

- In PL/I, arrays have extents and types either explicitly declared or inferred from their names. Each extent has a lower and upper bound, with a default lower bound of 1.

- In Rexx all variables are strings. Compound variables partially fill the role of arrays.

- In ooRexx, there is an **Array** class.

PL/I – REXX Comparison

Data types
Scalars

- In PL/I, scalar variables may be any of
- File
- Label
- Numeric
- Pointer
- String

PL/I – REXX Comparison
Data types
Scalars
File variables

- In PL/I, a file variable contains the attributes and status of a file.
- While Rexx has no equivalent, in ooRexx Stream objects play a similar role.

PL/I – REXX Comparison Data types Scalars File variables

- In PL/I, label variables refer to the combination of a location and a stack frame.
- Rexx has no equivalent.

PL/I – REXX Comparison
Data types
Scalars
Numeric variables

- In PL/I, numeric variables may be either **FIXED** or **FLOAT**, and may be declared with various attributes, e.g., base, size.

- In Rexx there is no equivalent, although the **NUMERIC** statemnt controls the results of numeric expressions.

PL/I – REXX Comparison
Data types
Scalars
Pointer variables

- In PL/I, pointer values either contain a storage address or contain the special value **NULL**. They are used to access **BASED** variables.
- Rexx has no equivalent, although the BIF **STORAGE** allow character variaables to play a similar role.

PL/I – REXX Comparison Data types Scalars String variables

- In PL/I, string variables may be either **BIT** or **CHARACTER**, and may either be fixed length or be **VARYING** with a maximum size. Comparisons yield a **BIT(1)** result.
- In Rexx, all variables are varying character strings.

PL/I – REXX Comparison
Data types
Structures

- In PL/I, structures contain named elements, which may themselves be arrays, scalars or structures.
- Rexx has no equivalent. However, compound variables and the **STORAGE** BIF fill some of the same roles.

PL/I – REXX Comparison
Name scoping

- In PL/I, names declared within a **BEGIN** or **PROCEDURE** block are local to that block and hide any equal names in an outer block.
- In REXX, The **PROCEDURE** statement dynamically hides all variable other than those listed in **EXPOSE**.

PL/I – REXX Comparison
Name scoping Example

```
MYCMD: PROC OPTIONS(MAIN);

   DECLARE FOO FLOAT,

           BAR FLOAT;

   CALL MYSUB;



MYSUB: PROCEDURE;

   DCL FOO CHAR;

   /* The outer declaration of FOO

      is not visible;

      the outer declaration of BAR

      is     visible */

   END MYSUB;

END MYCMD;
```

PL/I – REXX Comparison Storage Allocation

- PL/I allows the implicit allocation of **AUTOMATIC** variables, the explicit allocation of **BASED** variables and the explicit allocations of new generations of **CONTROLLED** variables.
- REXX has no equivalent.
- OoRexx has the **new** method for classes.