• Introduction
Structure

1. LOOP
2. Syntax of LOOP
3. Extensions of LOOP
4. LOO-PC
5. LOOP-VM
6. LOOP-0 sources and presentations
• Language for Object-Oriented Programming
• Object methods and object attributes only
  – No class methods or class attributes
• Variables are references to objects
  – Base data types exist only internally, in the compiler
  – Autoboxing
• Program start
  – Class Main is instantiated, and its method main is called

```plaintext
{ this program just writes its input. }
CLASS Main IS
  METHOD main IS
    c : Integer;
    BEGIN
      READ c;
      WHILE c # -1 DO
        WRITE c;
        READ c;
      END WHILE
    END METHOD
END CLASS
```
Syntax: Classes, Methods

program ::= classdecl

classdecl ::= CLASS identifier IS
    { memberdecl }
    END CLASS

memberdecl ::= vardecl ;'
    | METHOD identifier IS methodbody

vardecl ::= identifier { ',', identifier } ':' identifier

methodbody ::= { vardecl ;' }
    BEGIN statements
    END METHOD
Syntax Statements

```
statements ::= { statement }

statement ::= READ memberaccess ';
             | WRITE expression ';
             | IF relation
             |     THEN statements
             |     END IF
             | IF relation
             |     THEN statements
             |     END IF
             | WHILE relation
             |     DO statements
             |     END WHILE
             | memberaccess [ '=>' expression ] ';'
```
Syntax: Expressions

relation ::= expression [ ( '=' | '#’ | '<' | '>' | '<=' | '>=' ) expression ]

expression ::= term { ( '+' | '-' ) term }

term ::= factor { ( '*' | '/' | MOD ) factor }

factor ::= '-‘ factor
| memberaccess

memberaccess ::= literal { ‘.’ varorcall }

literal ::= number
| character
| NULL
| SELF
| NEW identifier
| ‘(‘ expression ‘)’
| varorcall

varorcall ::= identifier

identifier ::= letter { letter | digit }

Number ::= digit { digit }

letter ::= ‘A’ .. ‘Z’ | ‘a’ .. ‘z’

digit ::= ‘0’ .. ‘9’

character ::= ‘ ’
( Sichtbares-US-ASCII-Zeichen-kein-backslash
| ‘\’ ‘n’
| ‘\’ ‘\’
)’’
Extensions of LOOP

1. TRUE, FALSE (5%)
2. ELSE, ELSEIF (5%)
3. Boolean arithmetic (AND, OR, NOT) (5%)
4. Predefined class Boolean (10%)
5. Several user classes (10%)
6. Methods with parameters (10%)
7. Methods with results (RETURN) (10%)
8. Exception handling (10%)
9. Inheritance (virtual methods, BASE) (15%)
10. Garbage collection, variable initialization, (20%)
    Bonus tasks (30%)

19.03.2012
Criteria for Student Work

• Work in groups of up to three students

• The source code and a documentation has to be provided (30%).

• Published (and some unpublished) test programs must execute, or yield the expected compilation errors, resp. (70%).

• Scores:
  – Quality of description and source code
  – <50% → 9; ≥50% → 10; ≥55% → 11 ... ≥100% → 20
LOOP Compiler

- **Lexical analysis**
  - Partitioning the source code into symbols (tokens)

- **Syntax analysis**
  - Parsing the symbols sequence according to the grammar
  - Constructing the syntax tree

- **Context analysis**
  - Identification
    - (association of identifier uses to their declarations)
  - Type checking

- **Synthesis**
  - Generation of machine code
Using the LOOP Compiler

  -c show result of context analysis
  -h show this help information
  -hs <n> Reserve <n> words on the heap (standard is 100)
  -i show associations of identifiers
  -l show the result of lexical analysis
  -s show the result of syntax analysis
  -ss <n> Reserve <n> Words for the stack (standard is 100)
LOOP-C Class Structure
5. LOOP Virtual Machine

• Virtual Machine

• Register set
  – R0 ... R7
  – R0 is instruction counter

• storage
  – Machine words have 32 bits

• 17 instructions only

<table>
<thead>
<tr>
<th>Address</th>
<th>contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11272</td>
</tr>
<tr>
<td>1</td>
<td>197</td>
</tr>
<tr>
<td>2</td>
<td>3342</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>66</td>
</tr>
<tr>
<td>6</td>
<td>2341</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
## Machine Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRI Rx, n</td>
<td>Rx = n</td>
<td>Rx = 5</td>
</tr>
<tr>
<td>MRR Rx, Ry</td>
<td>Rx = Ry</td>
<td>Rx = 2</td>
</tr>
<tr>
<td>MRM Rx, (Ry)</td>
<td>Rx = s[Ry]</td>
<td>s[Rx] = 3</td>
</tr>
<tr>
<td>MMR (Rx), Ry</td>
<td>s[Rx] = Ry</td>
<td></td>
</tr>
<tr>
<td>ADD Rx, Ry</td>
<td>Rx += Ry</td>
<td>Rx += 1</td>
</tr>
<tr>
<td>SUB Rx, Ry</td>
<td>Rx -= Ry</td>
<td>Rx -= 1</td>
</tr>
<tr>
<td>MUL Rx, Ry</td>
<td>Rx *= Ry</td>
<td>Rx *= 2</td>
</tr>
<tr>
<td>DIV Rx, Ry</td>
<td>Rx /= Ry</td>
<td>Rx /= 1</td>
</tr>
<tr>
<td>MOD Rx, Ry</td>
<td>Rx %= Ry</td>
<td>Rx %= 2</td>
</tr>
<tr>
<td>AND Rx, Ry</td>
<td>Rx &amp;= Ry</td>
<td>Rx &amp;= 1</td>
</tr>
<tr>
<td>OR Rx, Ry</td>
<td>Rx</td>
<td>= Ry</td>
</tr>
<tr>
<td>XOR Rx, Ry</td>
<td>Rx ^= Ry</td>
<td>Rx ^= 1</td>
</tr>
<tr>
<td>ISZ Rx, Ry</td>
<td>Rx = (Ry == 0 ? 1 : 0)</td>
<td>Rx = (0 == 0 ? 1 : 0)</td>
</tr>
<tr>
<td>ISP Rx, Ry</td>
<td>Rx = (Ry &gt; 0 ? 1 : 0)</td>
<td>Rx = (1 &gt; 0 ? 1 : 0)</td>
</tr>
<tr>
<td>ISN Rx, Ry</td>
<td>Rx = (Ry &lt; 0 ? 1 : 0)</td>
<td>Rx = (-1 &lt; 0 ? 1 : 0)</td>
</tr>
<tr>
<td>JPC Rx, n</td>
<td>R0 = (Rx != 0 ? n : R0+3)</td>
<td>R0 = (1 != 0 ? 5 : R0+3)</td>
</tr>
<tr>
<td>SYS f, n</td>
<td>f == 0: Rn = read, f == 1: write Rn</td>
<td>f == 0: R0 = read, f == 1: write R0</td>
</tr>
<tr>
<td>DAT n, v</td>
<td>writes n v's into storage</td>
<td>writes 10 5's into storage</td>
</tr>
</tbody>
</table>
Comments

– Text behind ‘;’ up to end-of-line
MMR (R2), R5 ; like this comment

Labels

– Definition: label ':'
– Use: wherever numbers are allowed, but not as a first parameter of DAT

Assembly is done in two passes
1. Determine addresses of labels
2. Substitute labels with addresses

```
label ::= letter { letter | digit }
number ::= ['-'] digit { digit }
letter ::= 'A' .. 'Z' | 'a' .. 'z' | '_'
digit ::= '0' .. '9'
```
  – 1 output of first assembly pass
  – 2 output of second assembly pass
  – c Program is compiled, but not executed
  – h show this help information
  – i show instructions during execution
  – m show storage during execution
  – r show register set during execution
  – f2 show stack part for register R2
  – b2 show stack part for register R2 backwards
  – f4 show stack part for register R4
  – b4 show stack part for register R4 backwards
• LOOP-0 Initialization with VM options: –i –r –b2 –b4

00000000  MRI R1, 1
R0=3 R1=1 R2=0 R3=0 R4=0 R5=0 R6=0 R7=0
R2 backward: 1 1 *0 - - - - - - - - - - - - - - - - - - - -
R4 backward: 1 1 *0 - - - - - - - - - - - - - - - - - - - -

00000003  MRI R2, 165
R0=6 R1=1 R2=165 R3=0 R4=0 R5=0 R6=0 R7=0
R2 backward: 0 0 *0 5 0 1 3 3 2 1 3 4 3 5 2 1 3 5 5 2 5 4
R4 backward: 1 1 *0 - - - - - - - - - - - - - - - - - - - -

00000006  MRI R4, 265
R0=9 R1=1 R2=165 R3=0 R4=265 R5=0 R6=0 R7=0
R2 backward: 0 0 *0 5 0 1 3 3 2 1 3 4 3 5 2 1 3 5 5 2 5 4
R4 backward: 0 0 *0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0