LARP Users Guide
Third edition

Marco Lavoie

Copyright © 2004-2008 Marco Lavoie
LARP Users Guide

Table of content

1 Introduction .................................................................................................................. 17
  1.1 License agreement .................................................................................................. 18
  1.1.1 Freeware version .............................................................................................. 18
  1.1.2 Shareware version ............................................................................................ 19
  1.1.3 ........................................................................................................................... 19
  1.2 LARP's author ....................................................................................................... 21
  1.3 Installation ............................................................................................................. 21
    1.3.1 Minimal requirements in hardware and software ........................................... 21
    1.3.2 Installation from a CD .................................................................................... 22
    1.3.3 Installation from a downloaded file ................................................................. 22
    1.3.4 Uninstallation .................................................................................................. 22
  1.4 Registration ............................................................................................................ 23
    1.4.1 Registration procedure ................................................................................... 24
    1.4.2 Ordering super user keys ................................................................................ 25
  1.5 LARP updates ....................................................................................................... 26
  1.6 Technical support .................................................................................................. 27
    1.6.1 Online help ...................................................................................................... 27
    1.6.2 Bug reports ...................................................................................................... 28
    1.6.3 LARP's Web site ............................................................................................... 30
  2 Development environment ......................................................................................... 33
    2.1 Help available in LARP ....................................................................................... 33
    2.2 Interface elements ............................................................................................... 33
      2.2.1 Top menu ....................................................................................................... 35
      2.2.2 Control panel ................................................................................................ 38
      2.2.3 Document browser ....................................................................................... 40
      2.2.4 Template panel .............................................................................................. 40
      2.2.5 Editors ........................................................................................................... 41
      2.2.6 Message panel ............................................................................................... 43
      2.2.7 Status panel .................................................................................................. 43
      2.2.8 Execution console ......................................................................................... 44
      2.2.9 Step execution window ................................................................................ 45
    2.3 Textual editor functionalities ................................................................................. 46
      2.3.1 Editing a textual document ............................................................................ 46
      2.3.2 Search and replace ......................................................................................... 46
      2.3.3 Syntax highlight ............................................................................................... 47
      2.3.4 Textual editor configuration .......................................................................... 48
      2.3.5 Textual editor's edit commands ..................................................................... 48
        2.3.5.1 Textual editor commands accessible through menus ............................ 48
        2.3.5.2 Textual editor commands accessible through the keyboard ............... 49
        2.3.5.3 Mouse control in the textual editor ....................................................... 49
    2.4 Graphical editor functionalities ............................................................................. 50
      2.4.1 Flowchart instructions .................................................................................... 50
      2.4.2 Editing a flowchart ......................................................................................... 51
      2.4.3 Manipulating flowchart instructions ............................................................... 52
        2.4.3.1 Inserting, moving and deleting flowchart instructions ......................... 53
        2.4.3.2 Editing flowchart instructions ................................................................ 54
      2.4.4 Search and replace in a flowchart ................................................................. 55
      2.4.5 Zooming the display ....................................................................................... 55
      2.4.6 Instructions highlighting in step execution .................................................... 55
      2.4.7 Graphical editor configuration ...................................................................... 56
List of figures

Figure 1-1: Freeware identification................................................................. 19
Figure 1-2: Shareware identification............................................................... 19
Figure 1-3: Registration reminder................................................................. 23
Figure 1-4: Registering LARP...................................................................... 25
Figure 1-5: Super user key........................................................................... 25
Figure 1-6: Splash window........................................................................... 25
Figure 1-7: Downloading and installing updates ........................................ 26
Figure 1-8: Error messages.......................................................................... 28
Figure 1-9: Application error........................................................................ 28
Figure 1-10: Bug report form..................................................................... 29
Figure 1-11: Bug report files........................................................................ 30
Figure 1-12: Bug report transmission......................................................... 31
Figure 2-1: Application desktop ................................................................. 34
Figure 2-2: Control panel............................................................................ 38
Figure 2-3: Document browser ................................................................. 40
Figure 2-4: Pseudo code templates............................................................. 41
Figure 2-5: Flowchart templates................................................................. 41
Figure 2-6: LARP's textual editor.............................................................. 42
Figure 2-7: LARP's graphical editor......................................................... 42
Figure 2-8: Message panel.......................................................................... 43
Figure 2-9: Status panel............................................................................. 43
Figure 2-10: Execution console................................................................. 44
Figure 2-11: Step execution window.......................................................... 45
Figure 2-12: Textual editor panel............................................................... 46
Figure 2-13: Search window....................................................................... 47
Figure 2-14: Replace window.................................................................... 47
Figure 2-15: Graphical editor panel......................................................... 50
Figure 2-16: New main module................................................................. 52
Figure 2-17: New auxiliary module........................................................... 52
Figure 2-18: Selected insertion node......................................................... 52
Figure 2-19: Selected flowchart instruction............................................ 52
Figure 2-20: Inserting a flowchart instruction through an insertion node's contextual menu 53
Figure 2-21: Editing a flowchart instruction............................................ 54
Figure 2-22: Next instruction to be executed.......................................... 55
Figure 2-23: Highlighted break point....................................................... 55
Figure 2-24: Compilation state window................................................... 58
Figure 2-25: Next instruction to execute (step mode) in the textual editor... 61
Figure 2-26: Next instruction to execute (step mode) in the graphical editor... 62
Figure 2-27: Variables inspection panel.................................................. 62
Figure 2-28: Call stack inspection panel.................................................. 63
Figure 2-29: Break points inspection panel............................................. 64
Figure 2-30: Highlighted break point in the textual editor...................... 64
Figure 2-31: Highlighted break point in the graphical editor.................... 65
Figure 2-32: Step execute window's animation panel................................ 66
Figure 2-33: Animating the evaluation of a condition............................. 66
Figure 2-34: General configuration window............................................ 69
Figure 2-35: Configurating algorithm execution...................................... 70
Figure 2-36: Selecting a directory for temporary files............................. 71
Figure 12-2: Specifying a channel number in a flowchart READ instruction ....................................... 155
Figure 11-2: Outputs from auxiliary module Menu ...................................................................... 138
Figure 9-2: Order of appreciation of the previous condition ........................................................ 116
Figure 4-2: Welcome window ........................................................................................................ 86
Figure 3-2: Printing documents ..................................................................................................... 79
Figure C-13: Results of invoking MAXIMUM ............................................................................... 177
Figure C-12: Results of invoking LOWERCASE ......................................................................... 177
Figure C-10: Results of invoking LOG10 ..................................................................................... 175
Figure C-9: Results of invoking FORMAT ................................................................................... 175
Figure C-8: Results of invoking FLOOR ...................................................................................... 171
Figure C-7: Results of invoking EXP ........................................................................................... 171
Figure C-6: Results of invoking DATE ......................................................................................... 169
Figure C-5: Results of invoking COUNT ..................................................................................... 169
Figure C-3: Results of invoking CEILING .................................................................................... 167
Figure C-2: Results of invoking ARCTANGENT ......................................................................... 166
Figure C-1: Results of invoking ABSOLUTE ................................................................................ 166
Figure 12-3: Specifying a channel number in a flowchart WRITE instruction ............................. 156
Figure 12-1: Input/output buffers in a project .............................................................................. 150
Figure 11-3: Transfer of data through parameters ...................................................................... 140
Figure 9-3: Graphical representation of the order in which operators are evaluated .................. 116
Figure 7-3: Interpreting text entered on a READ instruction ....................................................... 100
Figure 7-4: Reading values in a flowchart ................................................................................... 101
Figure 7-5: Writing to the execution console .............................................................................. 101
Figure 7-6: Writing expression values in a flowchart ................................................................... 102
Figure 7-7: An inelegant query .................................................................................................... 102
Figure 7-8: The QUERY instruction........................................................................................... 103
Figure 7-9: Insert a query in a flowchart ...................................................................................... 103
Figure 7-10: Changing the separator ......................................................................................... 104
Figure 7-11: Using the separator to properly read data .............................................................. 104
Figure 9-1: Compound condition ................................................................................................. 116
Figure 9-2: Order of appreciation of the previous condition ........................................................ 116
Figure 9-3: Graphical representation of the order in which operators are evaluated ................. 116
Figure 11-1: Creating an auxiliary module .................................................................................. 136
Figure 11-2: Outputs from auxiliary module Menu .................................................................... 138
Figure 11-3: Transfer of data through parameters ...................................................................... 140
Figure 12-1: Input/output buffers in a project .............................................................................. 150
Figure 12-2: Specifying a channel number in a flowchart READ instruction ............................. 155
Figure 12-3: Specifying a channel number in a flowchart WRITE instruction ............................. 156

Figure C-1: Results of invoking ABSOLUTE ................................................................................ 166
Figure C-2: Results of invoking ARCTANGENT ......................................................................... 166
Figure C-3: Results of invoking CEILING .................................................................................... 167
Figure C-4: Results of invoking COSINUS .................................................................................. 168
Figure C-5: Results of invoking COUNT ..................................................................................... 169
Figure C-6: Results of invoking DATE ......................................................................................... 169
Figure C-7: Results of invoking EXP ........................................................................................... 171
Figure C-8: Results of invoking FLOOR ....................................................................................... 171
Figure C-9: Results of invoking FORMAT .................................................................................. 175
Figure C-10: Results of invoking LOG10 .................................................................................... 175
Figure C-11: Results of invoking LOGE ...................................................................................... 176
Figure C-12: Results of invoking LOWERCASE ...................................................................... 177
Figure C-13: Results of invoking MAXIMUM .............................................................................. 177
Figure C-14: Results of invoking MINIMUM ................................................................. 178
Figure C-15: Results of invoking PI ............................................................................... 179
Figure C-16: Results of invoking POSITION ................................................................. 180
Figure C-17: Results of invoking RANDOM ................................................................. 181
Figure C-18: Results of invoking ROUND .................................................................... 182
Figure C-19: Results of invoking SINUS ....................................................................... 182
Figure C-20: Results of invoking SIZE ......................................................................... 183
Figure C-21: Results of invoking SQUAREROOT ......................................................... 183
Figure C-22: Results of invoking SUBSET ..................................................................... 184
Figure C-23: Results of invoking TIME ........................................................................ 185
Figure C-24: Results of invoking TOCHARACTERS ................................................. 186
Figure C-25: Results of invoking TOSTRING ............................................................... 187
Figure C-26: Results of invoking UPPERCASE ............................................................ 187
LARP Users Guide

List of tables

Table 2-1: Commands accessible via the top menu ................................................................. 38
Table 2-2: Interface elements in control panel ....................................................................... 40
Table 2-3: Keyboard commands for the textual editor ............................................................. 49
Table 2-4: Flowchart instructions ............................................................................................. 51
Table 2-5: Graphical editor commands accessible through the keyboard ............................. 57
Table 5-1: Escape sequences .................................................................................................. 93
Table 6-1: Functions for manipulating containers .................................................................... 97
Table 7-1: Interpretation of text read ....................................................................................... 100
Table 7-2: Interpretation of space in READ and WRITE instructions................................. 103
Table 8-1: Arithmetic operators .............................................................................................. 105
Table 8-3: Operators for character strings .............................................................................. 106
Table 8-4: Operators for container .......................................................................................... 107
Table 8-5: Predefined mathematical functions ........................................................................ 109
Table 8-6: Predefined functions for handling character strings ............................................ 109
Table 8-7: Predefined functions for handling containers ....................................................... 110
Table 9-1: IF conditional structure ......................................................................................... 111
Table 9-2: Relational operators .............................................................................................. 112
Table 9-3: Logical operators .................................................................................................. 115
Table 9-4: Priority of operators .............................................................................................. 115
Table 9-5: Conditional structures IF and IF-ELSE ................................................................. 116
Table 9-6: IF-ELSE-IF conditional structure ......................................................................... 121
Table 9-7: Flowchart instructions required to build an IF-ELSE-IF structure ...................... 122
Table 9-8: SELECT structure ............................................................................................... 125
Table 10-1: WHILE iterative structure .................................................................................... 127
Table 10-2: REPEAT-UNTIL iterative structure .................................................................... 129
Table 10-3: FOR iterative structure ....................................................................................... 131
Table 12-1: ASCII table ......................................................................................................... 159
Table 12-2: Correspondences between hexadecimal, decimal and binary coding.............. 160
LARP Users Guide

List of pseudo codes

Pseudo code 1-1: A pseudo code ................................................................. 17
Pseudo code 4-1: Very simple pseudo code ................................................. 83
Pseudo code 4-2: Comments ........................................................................ 83
Pseudo code 4-3: Syntax of instructions .................................................... 84
Pseudo code 4-4: Separation of instructions .............................................. 85
Pseudo code 5-1: Variable names ................................................................. 92
Pseudo code 5-2: The operators ................................................................. 92
Pseudo code 5-3: Numerical values ....................................................... 93
Pseudo code 5-4: Character strings .......................................................... 93
Pseudo code 5-5: Single and double quotes in character strings .......... 93
Pseudo code 5-6: Assignments ................................................................. 94
Pseudo code 6-1: Containers ................................................................. 95
Pseudo code 6-2: Container in a container ................................................. 95
Pseudo code 6-3: Accessing elements in a container .......................... 95
Pseudo code 6-4: Accessing container elements ................................... 96
Pseudo code 6-5: Indeterminate elements in a container ..................... 96
Pseudo code 7-1: Reading a value ......................................................... 100
Pseudo code 7-2: Reading several values ................................................. 100
Pseudo code 7-3: Writing the value of expressions ............................... 101
Pseudo code 7-4: Prompting the user for an input ............................ 102
Pseudo code 7-5: The QUERY instruction ........................................ 102
Pseudo code 7-6: Querying for multiple inputs ........................................ 103
Pseudo code 7-7: The SEPARATOR instruction .................................. 104
Pseudo code 7-8: Changing separator .................................................... 104
Pseudo code 8-1: Arithmetic operators .................................................. 105
Pseudo code 8-2: Operator + applied to values of distinct types ........ 106
Pseudo code 8-3: Joining containers .......................................................... 107
Pseudo code 8-4: Subtraction of containers ........................................ 107
Pseudo code 9-1: Simple conditions .................................................... 113
Pseudo code 9-2: Condition testing if a value is indeterminate ........ 114
Pseudo code 9-3: Compound conditions .............................................. 115
Pseudo code 9-4: Conditional structures IF and IF-ELSE .......... 117
Pseudo code 9-5: Invalid IF structures ...................................................... 118
Pseudo code 9-6: Sequence of related IF structures .......................... 118
Pseudo code 9-7: Structures IF-ELSE embedded one in the other ...... 119
Pseudo code 9-8: Embedded conditional structures ............................. 120
Pseudo code 9-9: IF-ELSE-IF conditional structure ......................... 122
Pseudo code 9-10: IF-ELSE-IF structure testing a single variable .... 123
Pseudo code 9-11: SELECT structure ..................................................... 124
Pseudo code 10-1: WHILE iterative structure .............................................. 128
Pseudo code 10-2: REPEAT-UNTIL iterative structure .................... 130
Pseudo code 10-3: WHILE structure equivalent to the loop in Pseudo code 10-2 .............................................. 130
Pseudo code 10-4: Iterate from one value to the next with a WHILE structure .................................................. 132
Pseudo code 10-5: FOR iterative structure ............................................ 132
Pseudo code 10-6: FOR structure with step value other than 1 ......... 133
Pseudo code 10-7: Backward iterations in a FOR structure ............... 133
Pseudo code 10-8: Backward iterations in a FOR structure with negative step value ................... 134
# LARP Users Guide

## List of flowcharts

<table>
<thead>
<tr>
<th>Flowchart</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>A flowchart</td>
<td>18</td>
</tr>
<tr>
<td>4-1</td>
<td>A very simple flowchart</td>
<td>83</td>
</tr>
<tr>
<td>4-2</td>
<td>Syntax of instructions</td>
<td>85</td>
</tr>
<tr>
<td>4-3</td>
<td>Extending flowchart instructions</td>
<td>86</td>
</tr>
<tr>
<td>5-1</td>
<td>Sequential operations</td>
<td>92</td>
</tr>
<tr>
<td>5-2</td>
<td>Example of escape sequences</td>
<td>94</td>
</tr>
<tr>
<td>6-1</td>
<td>Modifying elements of a container</td>
<td>96</td>
</tr>
<tr>
<td>6-2</td>
<td>Packing a container</td>
<td>97</td>
</tr>
<tr>
<td>7-1</td>
<td>Changing separator</td>
<td>104</td>
</tr>
<tr>
<td>8-1</td>
<td>Concatenating character strings</td>
<td>106</td>
</tr>
<tr>
<td>8-2</td>
<td>CONTAINER type validation</td>
<td>107</td>
</tr>
<tr>
<td>9-1</td>
<td>Condition testing the type of value in a variable</td>
<td>114</td>
</tr>
<tr>
<td>9-2</td>
<td>IF structure</td>
<td>117</td>
</tr>
<tr>
<td>9-3</td>
<td>IF-ELSE structure</td>
<td>117</td>
</tr>
<tr>
<td>9-4</td>
<td>Embedded conditional structures</td>
<td>121</td>
</tr>
<tr>
<td>9-5</td>
<td>IF-ELSE-IF conditional structure</td>
<td>123</td>
</tr>
<tr>
<td>9-6</td>
<td>Example of SELECT structure</td>
<td>124</td>
</tr>
<tr>
<td>10-1</td>
<td>WHILE iterative structure</td>
<td>128</td>
</tr>
<tr>
<td>10-2</td>
<td>REPEAT-UNTIL iterative structure</td>
<td>130</td>
</tr>
<tr>
<td>10-3</td>
<td>FOR iterative structure</td>
<td>133</td>
</tr>
<tr>
<td>11-1</td>
<td>Main module</td>
<td>137</td>
</tr>
<tr>
<td>11-2</td>
<td>Simple auxiliary module as flowchart</td>
<td>138</td>
</tr>
<tr>
<td>11-3</td>
<td>Flowchart invoking an auxiliary module</td>
<td>139</td>
</tr>
<tr>
<td>11-4</td>
<td>Auxiliary module with parameters</td>
<td>141</td>
</tr>
<tr>
<td>11-5</td>
<td>Invoking an auxiliary module with arguments</td>
<td>141</td>
</tr>
<tr>
<td>11-6</td>
<td>Defining reference parameters</td>
<td>142</td>
</tr>
<tr>
<td>11-7</td>
<td>Call by value through value parameters</td>
<td>143</td>
</tr>
<tr>
<td>11-8</td>
<td>Declaring reference parameters</td>
<td>145</td>
</tr>
<tr>
<td>11-9</td>
<td>Auxiliary module with return value</td>
<td>146</td>
</tr>
<tr>
<td>11-10</td>
<td>Examples of alternate call syntax</td>
<td>148</td>
</tr>
<tr>
<td>12-1</td>
<td>Using reserved word BUFFER</td>
<td>152</td>
</tr>
<tr>
<td>12-2</td>
<td>Specifying access mode to a document</td>
<td>153</td>
</tr>
<tr>
<td>12-3</td>
<td>Closing multiple input/output channels at once</td>
<td>154</td>
</tr>
<tr>
<td>12-4</td>
<td>Writing in a document using an input/output channel</td>
<td>155</td>
</tr>
</tbody>
</table>
1 Introduction

*LARP* is in fact an acronym. It is a compression of the phrase «Logics of Algorithms and Resolution of Problems», conceived by Marco Lavoie. *LARP* is a programming language for the purpose of rapid prototyping of algorithms.

*LARP*’s main advantage over traditional programming languages is its flexible and semi natural syntax, allowing one to formulate algorithms without the impediments of cryptic languages such as *C++*, *Pascal* or *Java*.

Here is a sample *LARP* pseudo code indicating whether a value entered through the keyboard is positive or negative:

```
\ Simple pseudo code
BEGIN
  WRITE "Enter a number"
  READ N

  IF N < 0 THEN
    WRITE "Negative number"
  ELSE
    WRITE "Positive number"
  ENDIF
END
```

Pseudo code 1-1: A pseudo code

As you can see in the above algorithm, *LARP*’s syntax is straightforward and easy to understand, even for a non-programmer.

*LARP* also allows algorithms to be expressed as flowcharts. Pseudo code 1-1 can be represented equivalently in *LARP* as Flowchart 1-1.

*LARP* provides a simple and convivial development environment, allowing any user to rapidly learn the functionalities of the software. The user therefore focuses on conceiving algorithms rather than learning to use a complex interface or to program with an arid syntax.

The flexibility of *LARP*’s programming language as well as the user friendliness of its development environment make the software particularly practical for teaching programming. The instructor can use pseudo codes and/or flowcharts in *LARP* to introduce in a straightforward and concise manner programming notions such as conditions, loops and modularity. In practice, students can use *LARP* to implement and study algorithms introduced in class. In fact, a computer science instructor for teaching structured programming originally developed *LARP*.

To facilitate to use of *LARP* in a teaching environment, its development environment offers online help, presenting *LARP*’s programming syntax in pedagogic form. The online documentation allows one not only to learn to program algorithms, but also to exploit programming notions such as variables and containers, conditional and iterative structures, modularity and data storage. These programming concepts are thoroughly explained and put into context through concrete examples, making them easier to learn.
LARP is educational software essential to teaching algorithms and structured programming. Whether it is used in class or simply by its own, LARP makes programming a breeze to learn.

1.1 License agreement

LARP is distributed as freeware as well as shareware. Both versions are identical except for the following functionalities:

- The freeware version does not support plagiarism prevention functionalities while the registered shareware version does.
- Automatic updates are not available in the freeware version (updates must be performed manually) while they are automatically downloaded and installed in the registered shareware version.
- The super user mode is not supported in the freeware version since plagiarism prevention is not available. The shareware version offers super user mode when plagiarism prevention functionalities are activated during installation.

Both versions of LARP are distributed within the same installation file. The version to be installed is selected during installation.

1.1.1 Freeware version

LARP is available as freeware, a fully functional version of software released by its author to the public so they can use it without cost.

Freely downloadable, a freeware is by definition free! Users installing and using the freeware have no financial obligation towards its author. The distribution of LARP in freeware version is however limited according to restrictions stipulated in the license agreement (see section 0 for details).
Once installed, the freeware version of LARP is easily identifiable by the title bar of the development environment’s main window:

![LARP - Freeware [...\Mes documents\Develop](image)](image)

**Figure 1-1: Freeware identification**

LARP’s splash window also indicates which version is currently running.

### 1.1.2 Shareware version

LARP is also available as shareware, a fully functional version of software released by its author to allow the public to test it prior to getting involved financially. This form of software release is similar to a period of « free try » during which the author gives users a chance to assess a software in a most obvious and efficient manner: by using it as though they had bought it.

Freely downloadable, a shareware is not however free. A moral contract links the author to whoever installs the software. It stipulates that if the user wishes to continue using the product at the end of the evaluation period, and for that matter keep it installed on his computer, he is obligated to buy a license. In other words, once having used the shareware for a predetermined number of days, the user must drop by the cash register to pay for the product. If for whatever reason he does not wish to acquire it, the product must return to the shelves, which in this context means the shareware must be uninstalled from the user's computer.

Once installed, the shareware version of LARP is easily identifiable by the title bar of the development environment’s main window:

![LARP - Registered shareware [...\Mes documents\Develop](image)](image)

**Figure 1-2: Shareware identification**

LARP’s splash window also indicates which version is currently running.

**License**

Here is the license agreement permitting to evaluate LARP. Please read carefully what follows before using the software. The use of the software implies that you agree to all terms of this license agreement.

This software (LARP) is distributed as freeware (i.e. it's free) and as shareware (i.e not free). The user selects which version to install during the installation process. Both versions are identical except the freeware version does not offer plagiarism prevention functionalities. Read LARP’s user’s guide (or its online help) for details.

All terms of this license is applicable to both versions except the first section which is specific to each version.

**Freeware version**

This software is distributed as freeware, which means it can be installed and used by anyone without costs nor registration requirements. Distributing a complete or limited version (in terms of functionalities) of a software as freeware means the author does not expect users to pay for using the software.

Here is the license agreement permitting to use the freeware. Please read carefully what follows before using the freeware. The use of the freeware implies that you agree to all terms of this license agreement.
Shareware version

This software is distributed as shareware, a fully functional version of software released by its author to allow the public to test it prior to getting involved financially. In accordance with the terms specified hereafter, the shareware can be used free of charge, for evaluation purposes, for a period extending up to 120 days from installation date.

Here is the license agreement permitting to evaluate the shareware. Please read carefully what follows before using the shareware. The use of the shareware implies that you agree to all terms of this license agreement.

Distribution

You are authorized to duplicate the unregistered (i.e. evaluation) version of this software and to distribute it without alterations by electronic means (Internet, BBS’s, CD, etc).

Other than a reasonable participation relating to your expenses (ex: packaging), you are not allowed to charge for duplication or distribution of this software. You must not state under any circumstances that you sell the software itself.

You cannot, under any circumstances, ask for any compensation relating to the use of this software itself. The distribution of the shareware version cannot imply any kind of services nor support on behalf of the author.

Registered version

A registered version of this software can only be used by a single person on one or several computers. You can run this software across a network as long as you acquired as many individual licenses as there are computers likely to run the software across the network. For instance, if six different hosts can run the software across the network, you need a registered license for six users, whether they use the software at various times or simultaneously.

Restrictions of use

You are not allowed to alter this software in any way, or to add, to remove or to modify any message or dialogue box. You may not decompile, reverse engineer, disassemble or transform this software into any form of source code. You must not change, lend, rent or sell this software.

You may not publish or distribute algorithms or utilities for the purpose of generating registration codes for this software. You may not also publish information related to registration codes, nor distribute or publish such registration codes, without written approval of the author.

Restrictions of warrantees

This software is provided « as is » and without warrantees related to its execution, its performances, its street value or any other matter, whether expressed or implied. Since the author has no regards on the various hardware and software configurations of computers on which this software may be installed, no guarantee of compatibility or conformity is given. Good practice dictates that any new software must be thoroughly tested by the user with non-critical data before using it with critical data. The user bares all risks related to usage of this software. The responsibility of the seller is exclusively limited to the replacement of the product or the reimbursement of its purchase price.

Copyright

This product is Copyright © 2004-2008 Marco Lavoie.

It is protected by the Canadian law relating to copyrights and by international conventions. You acknowledge that no rights regarding intellectual ownership of this software are transferred to you. You acknowledge that full rights of this software remain the sole ownership of the author, and that you acquire no rights others than those specifically stated in this license agreement.
1.2 LARP’s author

The author of LARP is:

Name: Marco Lavoie  
M. Sc. in Computer science and mathematics

Mail address: 1, Avenue du Parc  
Gatineau, Qc (Canada)  
J8Y 1G5

Electronic mail: info@marcolavoie.ca

Web site: www.marcolavoie.ca

1.3 Installation

The installation program for LARP installs on your computer the software with all related files and utilities. Once the installation is completed and in accordance with the directives provided during installation, you can start LARP in any of the following ways:

- Through the LARP directory;
- Through the Start button, under Programs » LARP;
- Through an icon located on the computer’s Desktop;
- Through a button located in the Quick Launch area of the desktop.

Initiating LARP’s installation program varies depending on whether the program is on CD or is downloaded via Internet.

The installation CD or installation file allows to install both versions of LARP (freeware and shareware) in multiple languages, among which English and French.

1.3.1 Minimal requirements in hardware and software

Here are minimal system configurations to install and run LARP:

**Hardware requirements:**

- PC type computer
- 64 Mb of RAM (random access memory)
- 5 Mb of free space on hard drive
- CD drive (if LARP is to be installed from a CD)

**Software requirements:**

- Microsoft® Windows® 95, 98, ME, NT, on 2000 or XP.
- Netscape Browser 3.01, Netscape Communicator 4.x or Microsoft® Internet Explorer v4.0, or more recent versions of these utilities.
1.3.2 Installation from a CD

Here is the installation process when LARP is to be installed from a CD:

1. Start your computer and wait for Windows® to finish loading. **IMPORTANT:** If Windows® is already running, close all applications before proceeding with the installation of LARP.

2. If an anti-virus utility is running, temporarily deactivate it prior to installing LARP.

3. Insert LARP’s installation CD (label upwards) in the computer’s CD drive.

4. If LARP’s installation program does not start automatically, it must be started manually:
   4.1. Open My Computer (from the desktop) or start Windows Explorer.
   4.2. Click twice on the letter corresponding to the CD drive (in general D:, E: or F:).
   4.3. Click twice on the Setup file.

Follow instructions displayed on screen to install the software.

1.3.3 Installation from a downloaded file

If LARP’s installation program was downloaded via Internet, the downloaded file's name should be LarpSetup.exe.

To install LARP from the downloaded file, follow these instructions:

1. Start your computer and wait for Windows® to finish loading. **IMPORTANT:** If Windows® is already running, close all applications before proceeding with the installation of LARP.

2. If an anti-virus utility is running, temporarily deactivate it prior to installing LARP. It is recommended to verify that the downloaded file is virus free.

3. Open My Computer (from the desktop) or start Windows Explorer.

4. Find the downloaded installation file LarpXxSetup.exe (where Xx varies according to the interface language). If you experience difficulties in locating the file on your computer, use the Search tool in Windows Explorer.

5. Click twice on file LarpSetup.exe.

Follow instructions displayed on screen to install the software.

1.3.4 Uninstallation

LARP’s installation program also installs an automated uninstallation script on the computer. When run, the script removes:

- All files related to LARP (the executables, documentation files and sample files distributed with the software), and
- The launch links located on the Desktop, in the Quick Launch area and in the Start menu.
While the uninstallation script removes all files related to the LARP software, all project files created by the user prior to uninstallation remain on the computer.

To uninstall LARP, follow these instructions:

1. Click on the Start button and choose Settings, then Control Panel.
2. Click twice on the Add/Remove programs icon.
3. Browse through the list of programs and select LARP version # (where # is the version number of LARP installed on the computer).
4. Click on the Change/Remove button.

Once the software is uninstalled, click the OK button to conclude the uninstallation process.

1.4 Registration

This section of the guide is relevant exclusively to the shareware version of LARP.

As stated in the license agreement, you must register LARP in shareware version once the evaluation period has expired. Otherwise, you are required to stop using the software and uninstall it from your computers.

During the evaluation period, a registration reminder (see Figure 1-3) is automatically displayed on every start-up (and eventually on shutdown) of the software. This window enumerates the benefits of registration and indicates the remaining number of days to the evaluation period (the notice blinks when the evaluation period has expired).

![Registration of LARP](image)

Your LARP installation in shareware version is not registered. The author of LARP encourages you to buy a registered license in order to promote the development of LARP.

Among the benefits of registration, you will get:

- Free upgrades of the software, for life!
- Technical support through Internet.
- A printable User’s guide.

Most of all, the greatest benefit of registration is encouraging the author to further improve LARP in order to offer you a more robust and complete software.

Please proceed immediately to registration by clicking the Proceed with registration button.

Your evaluation license expires in 120 day(s).

[ ] Proceed with registration  [X] Register later

Figure 1-3: Registration reminder
As stated in the license agreement to which you agreed upon installation of the shareware version of *LARP*, you must cease to evaluate *LARP* once the evaluation period ends. You then have three alternatives:

1. **Register LARP** and keep on using it.
2. **Uninstall LARP** from your computers and cease using it.
3. Uninstall LARP and install its freeware version.

Any attempt to circumvent the evaluation period by uninstalling then re-installing the shareware version of *LARP* is strictly prohibited by the license agreement. Such attempts are automatically detected and blocked by *LARP*.

### 1.4.1 Registration procedure

*This section of the guide is relevant exclusively to the shareware version of LARP.*

To register *LARP*, you must acquire a **registration key** (alphanumeric sequence required to transform an evaluation version of the software into a registered version):

- To buy a registration key **by credit card** through a secured Web site, see *LARP*’s web site. The registration process through Internet is rapid, registration keys being sent by electronic mail (i.e. email) within minutes following clearance of the financial transaction.

- For those who are reluctant to buy by credit card over Internet, it is possible to transmit credit card information **by fax**. Processing orders received by fax takes more time than orders received over Internet, so a few days may be necessary for registration keys to be sent via email.

- Orders may also be paid **by check or money order**. See *LARP*’s web site for more information.

When in possession of your registration key, you may access the **Registering LARP window** (Figure 1-4) via the **Proceed with registration** button in the registration reminder window (Figure 1-3), or through the top menu. Fill out fields with information received upon registration (i.e. your registration name and key). Your evaluation version *LARP* will then be converted into a registered version, and registration reminders will no longer be displayed.

To install *LARP* on another computer once it has been registered, you will have to repeat the above procedure for entering registration data. Note however that the **license agreement** imposes restrictions on installing *LARP* on more than one computer.
1.4.2 Ordering super user keys

This section of the guide is relevant exclusively to the shareware version of LARP.

A super user key is required to activate LARP’s super user mode (available only in the shareware version of the software). Every key is pre-programmed with a unique and permanent username.

To order super user keys, you must own a registered license of the LARP shareware. When ordering keys you will be to provide your registration name and key, as well as your license number as they appear in LARP’s Splash window (see Figure 1-6). This window can be displayed anytime through LARP’s top menu.

LARP super user keys may be ordered via Internet, by fax or by mail:

- You may order a super user key by credit card through LARP’s secured web site.
1. Introduction

- For those who are reluctant to provide credit card information over Internet, the information may be transmitted by fax (see LARP's web site).

- Orders may also be paid by check or money order. See LARP's web site for more information.

Super user keys are shipped by courier upon clearance of the financial transaction.

1.5 LARP updates

This section of the guide is relevant exclusively to the shareware version of LARP.

The shareware version of LARP includes an integrated updating system activated explicitly through the top menu or implicitly at each startup of the shareware. Periodically updating a LARP installation ensures the shareware operates adequately, most recently discovered bugs are eradicated and new functionalities are available. The integrated updating system is exclusively available when LARP in shareware version is registered.

When the updating system is activated, LARP connects to an updates distribution Web server to determine whether new updates are available. If so, the latest updates are automatically downloaded and installed on the computer (Figure 1-7); then LARP shuts down and restarts for the updates to be activated.

**Figure 1-7: Downloading and installing updates**

Downloading LARP updates may occasionally fail for various reasons, in which cases their installation is interrupted and an appropriate error message is displayed. The possible error messages are:

- **Download and installation of updates cancelled by user**: the user intentionally interrupted the download of updates by pressing the **Cancel** button.

- **Internet connexion lost**: the Internet link has been severed, voluntarily or not.

- **Web server holding updates is unreachable**: the Web server distributing LARP updates is currently unavailable. In these circumstances updates should be downloaded at a later time. LARP's technical support should also be informed.

- **Updates directory currently unavailable**: the Web server distributing LARP updates is not functioning properly. Please inform LARP's technical support.
• **Access to updates currently suspended:** the configuration of the Web server distributing LARP updates may be corrupted. Please inform LARP's technical support.

• **Some updates files are missing on Web server:** the configuration of the Web server distributing LARP updates may be corrupted. Please inform LARP's technical support.

It is highly recommended to activate automatic search for new updates upon starting LARP. This option must be enabled through LARP's general configuration, which is exclusively available in the shareware version of the application. When automatic updates are enabled and the computer is connected to Internet, LARP silently checks for new updates on starting up and informs the user when such updates are available for download.

### 1.6 Technical support

In order to alleviate the use of LARP, various sources of help are available to the user:

- LARP's online help gives a detailed picture the software's development environment as well as its pseudo code syntax and flowchart instructions.

- Hint help corresponding to interface elements at the mouse cursor location is displayed in the status panel, at the bottom of the development environment.

- LARP's automated bug report system allows the user to report all bugs encountered while using the software.

- LARP's Web site is an excellent source of information on LARP. The latest version of the software is distributed through this site.

If you do not find answers to your questions in these sources of information, you can contact LARP's technical support via electronic mail at larp@marcolavoie.ca. While LARP's technical support makes all attempts to answer requests related to the use of the software, no promises are made to answer questions or requests related to the logic or the debugging of algorithms formulated with LARP.

#### 1.6.1 Online help

LARP’s online help is accessible at all times by pressing the keyboard’s F1 key or through the top menu, under item Help. Online documentation includes:

- A detailed description of LARP's development environment,

- A description of LARP’s pseudo code syntax and flowchart instructions, and

- More information on each of LARP’s warning and error messages.

Numerous pseudo code and flowchart examples illustrating the characteristics of LARP’s pseudo code syntax are thoroughly described. Various notions of structured programming are also introduced in pedagogic form.

Online documentation is formatted in standard format Microsoft® HTML Help. Most windows in LARP’s development environment provide a direct access to appropriate help texts via a Help button or by pressing the F1 key on the keyboard.
1.6.2 Bug reports

In spite of all the attention of LARP’s author to provide you with robust and bug free software, programming errors or omissions are sometimes overlooked. It is probably the case with LARP, unfortunately.

When a bug is encountered in LARP while editing algorithms or during their execution, in most cases an error message is displayed on screen to inform the user (see Figure 1-8).

![Error Message](image)

Figure 1-8: Error messages

In some circumstances LARP may also show more detailed error messages, such as:

![Application Error](image)

Figure 1-9: Application error

The Application Error window (Figure 1-9) displays diagnosis information on the encountered bug.

When such error occurs, technical support should be informed of the problem so that it gets corrected in the next release of the software.

LARP has an automated report system allowing the user to send bug diagnosis information to technical support by electronic mail. This bug report system is accessible via the top menu, under the Help item. Its characteristics are:

- A logging system records in a file all error messages displayed by LARP.
- If a project file is loaded in LARP when a bug is encountered, a copy of the project file is automatically secured for dispatch to LARP’s technical support.
If the computer is equipped with an *electronic mail agent* (for instance, *Microsoft® Outlook®*), an email can automatically be sent to LARP’s technical support team.

The bug reporting system consists of a sequence of windows gathering detailed information on the events and data having led to the encountered bug:

1. The first window allows the user to identify himself and to describe circumstances having led to the bug (see Figure 1-10).

2. The next window (Figure 1-11) enumerates files to be transmitted to LARP’s technical support along with the bug report. These files will help isolate the location and causes of the reported bug. User authorization is required for each file to be dispatched.

3. The third window (Figure 1-12) requests authorization to transmit the bug report with its attached files. Upon pressing the **Transmit** button, LARP transmits the report through the computer’s default electronic mail agent.

3.1. If the electronic mail agent fails to send the report, the user may transmit it manually using another electronic mail agent.

Transmitting bug reports to LARP’s technical support team allows to correct software errors so the next version of the software will be more robust.
1.6.3  LARP’s Web site

Please consult LARP’s Web site (larp.marcolavoie.ca) to get additional information on the software. There you will find:

- The most recent version LARP.
- A list of reported bugs and their status.
- A list of frequently asked questions received by LARP’s technical support, and the corresponding answers.
- Project file samples highlighting the various functionalities of LARP pseudo code syntax and flowchart instructions.

If you do not find answers to your questions in LARP’s Web site, contact technical support by electronic mail at larp@marcolavoie.ca. Sorry, no phone support.
LARP is ready to transmit the bug report and attached files by electronic mail.

Upon clicking on the "Transmit" button, the bug report with attached files will be automatically transmitted to destination through the system's default electronic mail manager.

Figure 1-12: Bug report transmission
2 Development environment

LARP’s development environment provides a graphical interface adhering to Microsoft® Windows® standards. Users familiar with traditional development environments such as Microsoft® Development Studio® and Borland® Delphi will feel at ease using LARP’s interface elements. Reciprocally, new programmers using LARP will acquire the basic skills to program with more sophisticated development tools.

LARP’s development environment consists of several interface elements including the application desktop, the execution console, the step execution window and online help.

2.1 Help available in LARP

LARP’s online help consists of document files installed on the computer along with the software. It can be invoked in several ways:

- Pressing the keyboard’s F1 key at all times displays help information related to the current context.
- Invoking online help through the top menu or contextual menus.
- Pressing the Help button on one of the software’s windows brings up supplemental information related to that window.
- When an error message with a reference number is selected in the message panel, pressing the Ctrl+F1 key combination brings up help information related to the selected error.
- Pressing the control panel’s help button ( ) brings up contextual help (it is equivalent to pressing Ctrl+F1 if an error message is selected, F1 otherwise).
- Hint help corresponding to interface elements at the mouse location is displayed continuously in the status panel.
- During the installation process, a link to LARP’s online help is created in the Programs » LARP folder, accessible through Windows®’s Start menu.

The software’s Users Guide is also available, published in Acrobat® PDF format. This guide is the printable equivalent of LARP’s online help.

2.2 Interface elements

LARP’s application desktop (Figure 2-1) is the core of its development environment. It consists of several interface elements:

- The top menu gives access to all of the development environment functionalities.
• The control panel regroups interface elements (mostly buttons) providing direct access to various commands from the top menu. These buttons invoke functionalities most often used when programming algorithms.

• The editors constitute the predominant panel in the application desktop. It allows the programmer to formulate and edit algorithms. LARP offers two editors: the textual editor allows to edit pseudo code modules and input/output buffers, and the graphical editor allows to edit flowchart modules.

• The message panel, located below the editors, displays various messages generated by LARP while compiling and executing algorithms (information, warnings and error messages). The message panel can be optionally deactivated through a top menu command.

• The document browser, on left, lists all modules and input/output buffers defined with the LARP project. Users can edit a project document by clicking on its name with the mouse. The document browser can be optionally deactivated through a top menu command.

• The template panel allows the user to insert instructions into project modules by drag and drop. The instructions available in the template panel depends on the type of project module being edited: pseudo code instructions are listed when pseudo code modules are edited, while flowchart instructions are listed when flowchart modules are edited.

• The status panel, located at the bottom of the development environment, shows information on LARP’s current state of operations: the position of the cursor within the
active editor, the current insert mode, the current username (only for the shareware version of LARP with plagiarism prevention functionalities activated) and help hints.

2.2.1 Top menu

The top menu in LARP’s development environment is located at the top of the application desktop (see Figure 2-1). It lists all the commands available in LARP (not to be confused with pseudo code instructions).

Table 2-1 briefly describes all commands accessible through the top menu. Some of the command can alternatively be invoked through an accelerator key (combination of keyboard keys) and/or by pressing a button (I.E. for Interface Element) in the control panel. Commands marked with symbol § in the table have restricted access according to the LARP version and/or whether or not super user mode is activated.

<table>
<thead>
<tr>
<th>Command</th>
<th>Accel.</th>
<th>I.E.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td></td>
<td></td>
<td>Commands relating to project files.</td>
</tr>
<tr>
<td>New...</td>
<td>Ctrl+N</td>
<td></td>
<td>Create a new project or document.</td>
</tr>
<tr>
<td>Open...</td>
<td>Ctrl+O</td>
<td></td>
<td>Open a project file.</td>
</tr>
<tr>
<td>Reopen</td>
<td>Alt+O</td>
<td></td>
<td>Reopen a project file opened previously.</td>
</tr>
<tr>
<td>Close project</td>
<td></td>
<td></td>
<td>Close the current project file.</td>
</tr>
<tr>
<td>Save</td>
<td>Ctrl+S</td>
<td></td>
<td>Save the current project in its file.</td>
</tr>
<tr>
<td>Save as...</td>
<td></td>
<td></td>
<td>Save the current project in a selected file.</td>
</tr>
<tr>
<td>Print... (§)</td>
<td>Ctrl+P</td>
<td></td>
<td>Print the current module or project.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This command is not available in the shareware version of LARP when plagiarism prevention functionalities are activated but super user mode is not.</td>
</tr>
<tr>
<td>Exit</td>
<td>Alt+F4</td>
<td></td>
<td>Close the application.</td>
</tr>
<tr>
<td>Edit</td>
<td></td>
<td></td>
<td>Commands relating to document editing.</td>
</tr>
<tr>
<td>Undo</td>
<td>Ctrl+Z</td>
<td></td>
<td>Cancel the last action.</td>
</tr>
<tr>
<td>Cut</td>
<td>Ctrl+X</td>
<td></td>
<td>Cut the selection and put it on the Clipboard.</td>
</tr>
<tr>
<td>Copy</td>
<td>Ctrl+C</td>
<td></td>
<td>Copy the selection on the Clipboard.</td>
</tr>
<tr>
<td>Paste</td>
<td>Ctrl+V</td>
<td></td>
<td>Insert the Clipboard contents at cursor position.</td>
</tr>
<tr>
<td>Clear</td>
<td>Del</td>
<td></td>
<td>Erase the selection.</td>
</tr>
<tr>
<td>Clear all</td>
<td>Ctrl+Del</td>
<td></td>
<td>Erase whole I/O buffer content.</td>
</tr>
<tr>
<td>Select all</td>
<td>Ctrl+A</td>
<td></td>
<td>Select whole document content.</td>
</tr>
<tr>
<td>Content...</td>
<td></td>
<td></td>
<td>Edit content of selected flowchart instruction.</td>
</tr>
<tr>
<td>Command</td>
<td>Accel.</td>
<td>I.E.</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
<td>------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Augment indent</td>
<td></td>
<td></td>
<td>Augment indentation of the selection or the line at cursor in the <strong>textual editor</strong>.</td>
</tr>
<tr>
<td>Reduce indent</td>
<td></td>
<td></td>
<td>Reduce indentation of the selection or the line at cursor in the <strong>textual editor</strong>.</td>
</tr>
<tr>
<td>Search...</td>
<td>Ctrl+F</td>
<td></td>
<td>Find the specified text.</td>
</tr>
<tr>
<td>Search next</td>
<td>F3</td>
<td></td>
<td>Repeat the last find.</td>
</tr>
<tr>
<td>Replace...</td>
<td>Ctrl+H</td>
<td></td>
<td>Replace specified text with different text.</td>
</tr>
<tr>
<td>View</td>
<td></td>
<td></td>
<td>Commands relating to the display of development interface panels.</td>
</tr>
<tr>
<td>Document</td>
<td></td>
<td></td>
<td>Select a project document to edit.</td>
</tr>
<tr>
<td>Lateral panel</td>
<td></td>
<td></td>
<td>Activate/deactivate display of <strong>document browser</strong> and <strong>template panel</strong> (at left of editor panel).</td>
</tr>
<tr>
<td>Messages</td>
<td></td>
<td></td>
<td>Activate/deactivate display of <strong>message panel</strong> (below the editor).</td>
</tr>
<tr>
<td>Console</td>
<td>F5</td>
<td></td>
<td>Bring to front <strong>execution console</strong> when active.</td>
</tr>
<tr>
<td>Zoom in</td>
<td></td>
<td></td>
<td>Augment the <strong>graphical editor</strong>'s scaling factor for displaying flowcharts.</td>
</tr>
<tr>
<td>Zoom out</td>
<td></td>
<td></td>
<td>Reduce the <strong>graphical editor</strong>'s scaling factor for displaying flowcharts.</td>
</tr>
<tr>
<td>No zoom</td>
<td></td>
<td></td>
<td>Show flowcharts at their normal size in the <strong>graphical editor</strong>.</td>
</tr>
<tr>
<td>Pseudo code… (§)</td>
<td></td>
<td></td>
<td>Display as pseudo code the flowchart module in the <strong>graphical editor</strong>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This command is not available in the <strong>shareware version</strong> of <strong>LARP</strong> when plagiarism prevention functionalities are activated but <strong>super user mode</strong> is not.</td>
</tr>
<tr>
<td>Execute</td>
<td></td>
<td></td>
<td>Commands relating to running projects.</td>
</tr>
<tr>
<td>Compile...</td>
<td>Ctrl+F7</td>
<td></td>
<td><strong>Compile</strong> current project without executing it.</td>
</tr>
<tr>
<td>Execute...</td>
<td>F7</td>
<td></td>
<td><strong>Compile</strong> and <strong>run</strong> current project.</td>
</tr>
<tr>
<td>Execute step-by-step...</td>
<td>Shift+F7</td>
<td></td>
<td><strong>Compile</strong> and <strong>run</strong> current project in <strong>step mode</strong>.</td>
</tr>
<tr>
<td>Terminate execution...</td>
<td></td>
<td></td>
<td>Terminate <strong>step execution</strong> of the project.</td>
</tr>
<tr>
<td>Execute one step</td>
<td>F6</td>
<td></td>
<td>Execute next instruction in <strong>step mode</strong>.</td>
</tr>
<tr>
<td>Animate step</td>
<td>Ctrl+F6</td>
<td></td>
<td><strong>Animate</strong> the execution of the next instruction in <strong>step mode</strong>.</td>
</tr>
<tr>
<td>Walk execution</td>
<td></td>
<td></td>
<td>Activate step execution in <strong>walk mode</strong>.</td>
</tr>
<tr>
<td>Command</td>
<td>Accel.</td>
<td>I.E.</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>--------</td>
<td>------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Continue execution</td>
<td></td>
<td></td>
<td>Activate step execution in continuous mode.</td>
</tr>
<tr>
<td>Pause step execution</td>
<td>Shift+F6</td>
<td></td>
<td>Temporarily pause step execution of the project.</td>
</tr>
<tr>
<td>Activate/Deactivate break point</td>
<td>F8</td>
<td></td>
<td>Activate a new break point or deactivate the one at cursor position.</td>
</tr>
<tr>
<td>Project</td>
<td></td>
<td></td>
<td>Commands relating to project management.</td>
</tr>
<tr>
<td>New</td>
<td></td>
<td></td>
<td>Commands to create a new project.</td>
</tr>
<tr>
<td>Pseudo code...</td>
<td></td>
<td></td>
<td>Create a new project using pseudo code in the main module.</td>
</tr>
<tr>
<td>Flowchart...</td>
<td></td>
<td></td>
<td>Create a new project using flowchart in the main module.</td>
</tr>
<tr>
<td>Close</td>
<td></td>
<td></td>
<td>Close current project.</td>
</tr>
<tr>
<td>Statistics... (§)</td>
<td>F9</td>
<td></td>
<td>Display statistics on current project.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This command is not available in the shareware version of LARP when plagiarism prevention functionalities are activated but super user mode is not.</td>
</tr>
<tr>
<td>Modules</td>
<td></td>
<td></td>
<td>Commands relating to module management.</td>
</tr>
<tr>
<td>New</td>
<td></td>
<td></td>
<td>Commands relating to creating new auxiliary modules in current project.</td>
</tr>
<tr>
<td>Pseudo code...</td>
<td></td>
<td></td>
<td>Create a new auxiliary module using pseudo code.</td>
</tr>
<tr>
<td>Flowchart...</td>
<td></td>
<td></td>
<td>Create a new auxiliary module using flowchart.</td>
</tr>
<tr>
<td>Rename...</td>
<td></td>
<td></td>
<td>Change the name of module currently edited.</td>
</tr>
<tr>
<td>Delete...</td>
<td></td>
<td></td>
<td>Delete the currently edited auxiliary module from project.</td>
</tr>
<tr>
<td>Import... (§)</td>
<td></td>
<td></td>
<td>Import text file content into the currently edited pseudo code module.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This command is not available in the shareware version of LARP when plagiarism prevention functionalities are activated but super user mode is not.</td>
</tr>
<tr>
<td>Export... (§)</td>
<td></td>
<td></td>
<td>Export content of currently edited pseudo code module into a text file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This command is not available in the shareware version of LARP when plagiarism prevention functionalities are activated but super user mode is not.</td>
</tr>
<tr>
<td>I/O buffers</td>
<td></td>
<td></td>
<td>Commands relating to I/O buffer management.</td>
</tr>
<tr>
<td>New...</td>
<td></td>
<td></td>
<td>Create a new I/O buffer in current project.</td>
</tr>
<tr>
<td>Rename...</td>
<td></td>
<td></td>
<td>Change the name of I/O buffer currently edited.</td>
</tr>
<tr>
<td>Delete...</td>
<td></td>
<td></td>
<td>Delete the currently edited I/O buffer from project.</td>
</tr>
</tbody>
</table>
### Command Reference

<table>
<thead>
<tr>
<th>Command</th>
<th>Accelerator</th>
<th>I.E.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import…</td>
<td></td>
<td></td>
<td>Import text file content into the currently edited I/O buffer.</td>
</tr>
<tr>
<td>Export…</td>
<td></td>
<td></td>
<td>Export content of currently edited I/O buffer into a text file.</td>
</tr>
<tr>
<td>Options</td>
<td></td>
<td></td>
<td>Commands relating to the configuration of LARP.</td>
</tr>
<tr>
<td>General…</td>
<td></td>
<td></td>
<td>Display and modify the general configurations.</td>
</tr>
<tr>
<td>Colors…</td>
<td></td>
<td></td>
<td>Display and modify the color configurations.</td>
</tr>
<tr>
<td>Registration...</td>
<td>($)</td>
<td></td>
<td>Registering your LARP distribution. This command is accessible only in the shareware version of LARP.</td>
</tr>
<tr>
<td>Get updates…</td>
<td>($)</td>
<td></td>
<td>Download and install latest software updates. This command is accessible only in the shareware version of LARP.</td>
</tr>
<tr>
<td>Identification...</td>
<td>($)</td>
<td></td>
<td>Change active username. This command is accessible only in the shareware version of LARP when plagiarism prevention functionalities are activated.</td>
</tr>
<tr>
<td>Help</td>
<td></td>
<td></td>
<td>Commands relating to online help.</td>
</tr>
<tr>
<td>Contextual help...</td>
<td>$F1$</td>
<td></td>
<td>Display LARP's online help according to context.</td>
</tr>
<tr>
<td>Contents...</td>
<td>Shift+$F1$</td>
<td></td>
<td>Display index of LARP's online help.</td>
</tr>
<tr>
<td>Error…</td>
<td>$Ctrl+F1$</td>
<td></td>
<td>Display help information on error message currently selected in message panel.</td>
</tr>
<tr>
<td>Report a bug...</td>
<td></td>
<td></td>
<td>Report a bug by email.</td>
</tr>
<tr>
<td>About…</td>
<td></td>
<td></td>
<td>Display information on the installed version of LARP.</td>
</tr>
</tbody>
</table>

Table 2-1: Commands accessible via the top menu

### 2.2.2 Control panel

The control panel (Table 2-2) in LARP groups interface elements (predominantly buttons) giving a quick access to some of the top menu's most often invoked commands:

![Control panel](image)

Figure 2-2: Control panel

Accessibility of control panel elements is established according to the current context. For instance, the Cut button is enabled only when a block of text is selected in the textual editor or a flowchart instruction selected in the graphical editor.
Table 2-2 describes the interface elements (I.E.) available in the control panel:

<table>
<thead>
<tr>
<th>I.E.</th>
<th>Commande</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![File icon]</td>
<td>File » New...</td>
<td>Create a new project or document.</td>
</tr>
<tr>
<td>![File icon]</td>
<td>File » Open...</td>
<td>Open a project file.</td>
</tr>
<tr>
<td>![File icon]</td>
<td>File » Close project</td>
<td>Close the current project file.</td>
</tr>
<tr>
<td>![File icon]</td>
<td>File » Save</td>
<td>Save the current project in its file.</td>
</tr>
</tbody>
</table>
| ![File icon] | File » Print... | Print the current module or project.  
This command is not available in the shareware version of LARP when plagiarism prevention functionalities are activated but super user mode is not. |
| ![Edit icon] | Edit » Undo | Cancel the last action. |
| ![Edit icon] | Edit » Cut | Cut the selection and put it on the Clipboard. |
| ![Edit icon] | Edit » Copy | Copy the selection on the Clipboard. |
| ![Edit icon] | Edit » Paste | Insert the Clipboard contents at cursor position. |
| ![Edit icon] | Edit » Augment indent | Augment indentation of the selection or the line at cursor in the textual editor. |
| ![Edit icon] | Edit » Reduce indent | Reduce indentation of the selection or the line at cursor in the textual editor. |
| ![View icon] | View » Zoom in | Augment the graphical editor’s scaling factor for displaying flowcharts. |
| ![View icon] | View » Zoom out | Reduce the graphical editor’s scaling factor for displaying flowcharts. |
| ![Edit icon] | Edit » Search... | Find the specified text. |
| ![Edit icon] | Edit » Search next | Repeat the last find. |
| ![View icon] | View » Console | Bring to front execution console when active. |
| ![Execute icon] | Execute » Compile... | Compile current project without executing it. |
| ![Execute icon] | Execute » Execute... | Compile and run current project. |
| ![Execute icon] | Execute » Execute step-by-step... | Compile and run current project in step mode. |
2. Development environment

I.E. | Commande | Description
--- | --- | ---
Help » Contextual help... | Display LARP's online documentation according to context:
- If a message with reference number is selected in the message panel, online help related to the message is displayed;
- If a module instruction is selected in the editor, online help related to that instruction is displayed;
- Otherwise, online help's table of contents is displayed.
MAIN | Select the document to edit.

Table 2-2: Interface elements in control panel

2.2.3 Document browser

The document browser (Figure 2-3), located on the upper left side of the application desktop, enumerates the documents (modules and input/output buffers) contained in the project:

![Figure 2-3: Document browser](image)

The name of the document currently being edited is highlighted. An icon along each document name indicates document type: a blue page (for example MAIN in Figure 2-3) indicates a pseudo code module, a yellow diagram (PADDING in Figure 2-3) indicates a flowchart module and a white page (REGISTER in Figure 2-3) indicates an input/output buffer. A document may be edited by mouse clicking on the document's name in the browser.

The browser provides a contextual menu (accessible with a right mouse button click on the browser) with commands to add new documents to the project, and to rename or remove existing documents.

The document browser can be optionally hidden through the menus.

2.2.4 Template panel

The template panel, located in the lower left side of the application desktop, offers various algorithm instruction templates. The proposed templates varies according to the type of module being edited: pseudo code instruction templates (Figure 2-4) are displayed in the panel when a pseudo-code module is being edited, while flowchart instruction templates (Figure 2-5) are displayed in the panel when a flowchart module is edited. The template panel is blank when an input/output buffer is displayed in the editor.
Algorithm instructions may be inserted into modules by *drag and drop* operations from the template panel to the editors. These operations essentially consist in selecting an instruction from the template panel and dragging it to an appropriate location into the module displayed in the editor. For more information on these operations, see sections Textual editor functionalities and Graphical editor functionalities.

### 2.2.5 Editors

*LARP*’s development environment includes two editors for editing project documents: the **textual editor** edits pseudo code modules and **input/output buffers**, while the **graphical editor** edits flowchart modules. These editors can open, edit and save documents listed in the document browser.

*LARP*’s textual editor (Figure 2-6) provides functionalities usually found in the most conventional text editors. Among these functionalities are cutting and copying text to the clipboard, and pasting text back from it, highlighting pseudo code keywords, importing and exporting text and displaying line numbers in the margin with caret coordinates in the status panel.
2. Development environment

LARP Users Guide

Figure 2-6: LARP's textual editor

LARP's graphical editor (Figure 2-7) uses drag and drop along with the template panel to build and modify flowcharts. Among these functionalities are cutting, copying and pasting flowchart instructions to and from the clipboard, inserting, moving, flipping and deleting flowchart instructions, and editing these instructions.

Figure 2-7: LARP's graphical editor

Both editors offer an intuitive interface and common functionalities, among which search and replace, undo operations, printing documents, automated security backups of project files and highlighting instructions in step execution modes.

The accessibility to some functionalities of both editors is restricted according to the version of LARP used and/or whether or not the super user mode is activated. For more information see the following sections: textual editor functionalities and graphical editor functionalities.
2.2.6 Message panel

The **message panel** (Figure 2-8), located below the editors, displays various messages (information, warnings and mistakes) generated by **LARP** when compiling and executing algorithms:

```
Compiling project...
Compiling module MAIN...
Compiling module PADDING...
Compiling module HEADINGS...
Linking project...
ERROR (Module PADDING, line 9) - Number of arguments provided in module call does not care
INTERRUPTED - errors were detected.
```

**Figure 2-8: Message panel**

Information and warning messages are usually displayed in black (depending on current **Windows®** configuration), while error messages are displayed in red. Clicking on a warning or an error message within the message panel resets editors to the corresponding erroneous algorithm instruction:

- when the error is located in a pseudo code module, the **textual editor**'s caret is relocated to the corresponding pseudo code instruction in error;
- when the error is located in a flowchart module, the flowchart instruction in error is selected within the **graphical editor**.

The message panel provides a contextual menu (accessible with a right button mouse click on the panel) with commands to clear the panel's content or to hide the whole panel. The hidden message panel can displayed through the **top menu**. The contextual menu also offers a command to access online help in order to get more information on selected messages.

See section **Compilation and execution** for more information on warning and error messages.

2.2.7 Status panel

The **status panel** (Figure 2-9) at the bottom of the **development environment** displays various status information on **LARP**:

```
11   Ins  5765e6b  Augment indentation of the selection or the line at cursor.
```

**Figure 2-9: Status panel**

The following status fields are displayed (from left to right):

1. The content of the first field depends on the active editor: it displays the **position of the caret** within the **textual editor** (in **row:column** format), or the ID of the **selected instruction** in the graphical editor.

2. The **active insert mode** or the **zoom factor**: indicates whether the text editor's insertion mode of activated (**Ins**) or not, or indicates the graphical editor's zoom factor (in %).

3. **Active username**: displays the **username** specified by the user, or **SU** (in red) if **super user mode** is enabled. This field is only displayed in the **shareware version** of **LARP** when plagiarism prevention functionalities are activated.

4. **Hint help**: displays short **contextual help** information on the interface element at mouse location.
2.2.8 Execution console

When LARP runs an algorithm, read and write operations are mostly performed via the execution console (Figure 2-10).

![Console](image)

Figure 2-10: Execution console

The execution console is a window automatically displayed upon starting the execution of an algorithm. The console allows the user to interact with the running algorithm, providing inputs to **READ** instructions and displaying information according to **WRITE** instructions (**REQUEST** instructions also interacts through the console).

Once the execution of the algorithm terminates, the user may close the console by pressing any keyboard key. The console can also be closed at all times by clicking the **X** button on the window's caption, or by interrupting **step-by-step execution** of the algorithm. If an algorithm is running while an attempt is made to close the console, the user is asked for confirmation prior to interrupting execution and closing the console. A status panel located on top of the console indicates the state of the algorithm's execution (**In execution**, **Paused**… or **Finished**).

The development environment cannot manage more than one execution console at a time. An opened console must therefore be closed before the algorithm can be run again.

The scrollbars to the right and bottom of the console allow panning through the console's content. By default, the console displays 25 lines of 80 characters, but it retains up to the last 200 lines of input/output operations. As any common window, the console window can be resized with the mouse.

By default, the console's background is black and outputs are displayed in white. Output colors are configurable (see section **Colors in the execution console**).
2.2.9 Step execution window

When step execution of the algorithm is initiated, the step execution window (Figure 2-11) is automatically displayed along with the execution console:

![Step Execution Window](image)

Figure 2-11: Step execution window

The step execution window provides various functionalities for controlling the execution of the algorithm, among which:

- **Single step execution**: the algorithm’s instructions may be executed one at a time with pauses in between in order to inspect variable values and their evolution during execution.

- **Variables inspection**: in between the execution of the algorithm’s instructions, variable values may be watched. Container elements may also be inspected individually.

- **Call stack status inspection**: the execution stack contains the calls of all modules originating from the main module that have led to the instruction currently being executed. The step execution window continuously displays module names in the call stack.

- **Break points management**: break points flag algorithm instructions where execution must be paused when executing in step mode. The step execution window allows managing all active break points in the algorithm’s modules.

- **Instructions animation**: the step execution window may animate the execution of individual instructions within the algorithm. Animation allows viewing how and in which order instruction elements are executed by LARP.

Step execution functionalities are described in more details in section 2.5.2.
2.3 Textual editor functionalities

*LARP*’s development environment includes a text editor for creating and editing textual project documents (i.e. pseudo code modules and input/output buffers). The editor can open, edit and save the documents listed in document browser. It offers many functions commonly found in conventional text editors, including cut and paste, syntax highlighting and automatic indentation.

2.3.1 Editing a textual document

To edit a pseudo code module or an input/output buffer within a project currently loaded in *LARP*, the user may choose the targeted document from the document browser or via the control panel. The content of the selected document is then loaded into the editor. In fact *LARP’s* textual editor is automatically activated when the document selected for editing is a pseudo code module or an input/output buffer.

The editor panel consists of two sections:

- a lateral margin, on left, display line numbers, bookmarks and step execution markers;
- the main section, on right, displays the contents of a selected document for editing.

![Figure 2-12: Textual editor panel](image)

The background color for the editing section (on right) depends on the type of document edited: a green background (by default, modifiable via Color selection) indicates a module containing pseudo code, while a white background (by default, also modifiable via color selection) indicates an input/output buffer containing data. When the edited document is a module, syntax highlighting of *LARP* pseudo code keywords is automatically activated.

Editor commands are accessible through the top menu, through the editor’s contextual menu (accessible by clicking on the edit panel with the right mouse button), through buttons on control panel, through the keyboard, and/or through the mouse.

Access to some commands is restricted to super user mode in the shareware version of *LARP*, such as clipboard management and document printing. These commands are disabled for non super users.

2.3.2 Search and replace

*LARP*’s textual editor offers text search and replace functionalities. The Search window (Figure 2-13) searches for a sequence of characters in the current document or throughout all documents.
in the project. The *Replace window* (Figure 2-14) provides the additional functionality of replacing the located text with an alternate sequence of characters.

![Search window](image)

**Figure 2-13: Search window**

Both windows provide dropped down lists to recall previous searches.

![Replace window](image)

**Figure 2-14: Replace window**

Search and replace functionalities in *LARP* are also applicable to *flowcharts*.

### 2.3.3 Syntax highlight

When a module is displayed in the textual editor, elements of its pseudo code are displayed in distinct colors according to their significance. By default, these colors are:

- Reserved keywords of pseudo code are displayed in **black bold** characters.
- Predefined function names are displayed in **navy bold** characters.
- Comments are displayed in *magenta italics* characters.
- Character strings are displayed in "*red*" characters.

Other elements of pseudo code are displayed in black characters.

Furthermore, the following pseudo code instructions are highlighted during *step execution*:

- The next line to be executed in *single step mode*.
- Lines to which are attached *break points*.

The choice of highlight colors is configurable via *Color selection*. 
2.3.4 Textual editor configuration

The following characteristics of the textual editor are configurable:

- **Editing functions**, and
- **Colors** for displaying pseudo code.

For more information on configuration of the editor, consult the corresponding sections (Configuration of editors and Colors in the editors).

2.3.5 Textual editor’s edit commands

Commands for editing documents are accessible via:

- the **top menu**,  
- the editor’s **contextual menu** (accessible by clicking on the edit panel with the right mouse button),
- buttons in the **control panel**,  
- the **keyboard**, and/or  
- the **mouse**.

2.3.5.1 Textual editor commands accessible through menus

There are two types of menus in **LARP**:

1. the **top menu**, displayed at the top of the development environment, and
2. **contextual menus**, linked to various interface elements of the development environment and accessible with a click of the mouse’s right button on the targeted element.

The textual editor responds to commands accessible through the top menu (see Top menu section for a list of these commands) as well as through its contextual menu. Accessibility of these commands depends on the type document being edited:

- When editing a pseudo code **module**, the editor’s contextual menu lists commands related to **clipboard management** and **text search**.

- When editing an **input/output buffer** (i.e. data), the editor’s contextual menu lists commands related to clipboard management and text search, along with commands for importing and exporting data.

For further description of menu commands, consult the related sections (Top menu and Control panel).
2.3.5.2 Textual editor commands accessible through the keyboard

Along with commands accessible through menus, LARP’s textual editor responds to commands exclusively invoked through the keyboard:

<table>
<thead>
<tr>
<th>Keys</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;End&gt;</td>
<td>Move the caret at the end of the current line.</td>
</tr>
<tr>
<td>&lt;Home&gt;</td>
<td>Move the caret at the beginning of the current line.</td>
</tr>
<tr>
<td>&lt;Enter&gt;</td>
<td>Insert a new line at current caret position.</td>
</tr>
<tr>
<td>&lt;Ins&gt;</td>
<td>Activate and deactivate insertion mode.</td>
</tr>
<tr>
<td>&lt;Del&gt;</td>
<td>Delete a character to the right of the caret.</td>
</tr>
<tr>
<td>&lt;Backspace&gt;</td>
<td>Delete a character to the left of the caret.</td>
</tr>
<tr>
<td>&lt;Tab&gt;</td>
<td>Insert a tabulation (contextual or conventional) to the right of the caret.</td>
</tr>
<tr>
<td>&lt;Left&gt;</td>
<td>Move the caret to the previous character.</td>
</tr>
<tr>
<td>&lt;Right&gt;</td>
<td>Move the caret to the next character.</td>
</tr>
<tr>
<td>&lt;Up&gt;</td>
<td>Move the caret to the previous line.</td>
</tr>
<tr>
<td>&lt;Down&gt;</td>
<td>Move the caret to the next line.</td>
</tr>
<tr>
<td>&lt;Page Up&gt;</td>
<td>Move the caret upwards a number of lines corresponding to the height of the edit panel.</td>
</tr>
<tr>
<td>&lt;Page Down&gt;</td>
<td>Move the caret downwards a number of lines corresponding to the height of the edit panel.</td>
</tr>
<tr>
<td>&lt;Ctrl&gt;+&lt;Up&gt;</td>
<td>Move the caret to the beginning of the document.</td>
</tr>
<tr>
<td>&lt;Ctrl&gt;+&lt;End&gt;</td>
<td>Move the caret to the end of the document.</td>
</tr>
<tr>
<td>&lt;Shift&gt;+&lt;Left&gt;</td>
<td>&lt;Right&gt;</td>
</tr>
<tr>
<td>&lt;Alt&gt;+&lt;Left&gt;</td>
<td>&lt;Right&gt;</td>
</tr>
<tr>
<td>&lt;Ctrl&gt;++&lt;Shift&gt;+0...9</td>
<td>Activate and deactivate a bookmark (bookmarks are numbered from 0 to 9) .</td>
</tr>
<tr>
<td>&lt;Ctrl&gt;+0...9</td>
<td>Move the caret to the corresponding bookmark.</td>
</tr>
</tbody>
</table>

Table 2-3: Keyboard commands for the textual editor

Along with the commands listed above, the editor also responds to top menu commands invoked through their accelerator key.

2.3.5.3 Mouse control in the textual editor

LARP’s textual editor responds in standard ways to various mouse actions:

- A click of the left mouse button moves the caret to the mouse location in the edited document.
- A click of the right mouse button displays the editor's contextual menu at the mouse location.
- Pressing the left mouse button and dragging the cursor over text selects a block of text.
Furthermore, the mouse may be used along with the template panel to insert pseudo code instructions into the edited module by drag and drop operations. To do so, a template must be selected with the mouse in the template panel, dragged over the textual editor and dropped at the appropriate line in the pseudo-code module. The inserted instruction may afterward be customized to the needs of the algorithm.

2.4 Graphical editor functionalities

To edit a flowchart module within a project currently loaded in LARP, the user may choose the targeted document from the document browser or via the control panel. The content of the selected document is then loaded into the graphical editor. LARP’s graphical editor is automatically activated when the document selected for editing is a flowchart module.

![Figure 2-15: Graphical editor panel](image)

The graphical editor allows building and editing flowcharts by drag and drop using the mouse along with the template panel. Among functionalities available in the graphical editor are copy, cut and paste operations with the clipboard, insert, move, flip orientation, transform and delete flowchart instructions, and edit flowchart instructions.

2.4.1 Flowchart instructions

The template panel lists all flowchart instructions available in LARP:

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Sequential instruction" /></td>
<td>Sequential instruction: allows formulating sequential instructions such as assignments as well as opening and closing input/output channels, etc.</td>
</tr>
<tr>
<td>Instructions</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td><img src="image" alt="Input/output instruction" /></td>
<td>Input/output instruction: allows formulating input/output instructions to the console, input/output buffer or files.</td>
</tr>
<tr>
<td><img src="image" alt="Auxiliary module call" /></td>
<td>Auxiliary module call: invoke an auxiliary module (pseudo code or flowchart) during execution.</td>
</tr>
<tr>
<td><img src="image" alt="Comment" /></td>
<td>Comment: insert non executable informations in the flowchart.</td>
</tr>
<tr>
<td><img src="image" alt="IF conditional structure" /></td>
<td>IF conditional structure: a sequence of instructions to be executed or not according to the value of a given condition.</td>
</tr>
<tr>
<td><img src="image" alt="IF-ELSE conditional structure" /></td>
<td>IF-ELSE conditional structure: two sequences of instructions, one of which is to be executed according to the value of a given condition.</td>
</tr>
<tr>
<td><img src="image" alt="WHILE repetitive structure" /></td>
<td>WHILE repetitive structure: a sequence of instructions to be executed repeatedly according to the value of a given condition.</td>
</tr>
<tr>
<td><img src="image" alt="REPEAT-UNTIL repetitive structure" /></td>
<td>REPEAT-UNTIL repetitive structure: a sequence of instructions to be executed repeatedly according to the value of a given condition.</td>
</tr>
<tr>
<td><img src="image" alt="FOR repetitive structure" /></td>
<td>FOR repetitive structure: a sequence of instructions to be executed repeatedly a given number of times.</td>
</tr>
<tr>
<td><img src="image" alt="SELECT structure" /></td>
<td>SELECT structure: conditional structure consisting of one or more sequences of instructions, one of which is to be executed according to the value of a given mathematical expression.</td>
</tr>
<tr>
<td><img src="image" alt="IF-ELSE-IF conditional structure" /></td>
<td>IF-ELSE-IF conditional structure: conditional structure consisting of one or more sequences of instructions, one of which is to be executed according to the value of a given conditions.</td>
</tr>
<tr>
<td><img src="image" alt="Branching for conditional structures" /></td>
<td>Branching for conditional structures: allows adding alternate sequences of instructions in SELECT structure and IF-ELSE-IF conditional structure.</td>
</tr>
</tbody>
</table>

Table 2-4: Flowchart instructions

Many instructions offer a choice of orientation (such as conditional structures). The orientation of such instructions is purely esthetical and of no significance in the execution of the algorithm.

2.4.2 Editing a flowchart

To edit a flowchart module within a project currently loaded in LARP, the user may choose the targeted document from the document browser or through the control panel. The selected
document’s content is then loaded into the **graphical editor. LARP**'s graphical editor is automatically activated when the document selected for editing is a flowchart module. When the selected document contains text (i.e. a pseudo code module or an input/output buffer), the **textual editor** is activated.

When a new flowchart module is created (through the **top menu** or the **document browser**), a minimal flowchart is automatically generated by **LARP**:

- If the new flowchart is the project's **main module**, **START** and **END** instructions are inserted in the flowchart.

- If the new flowchart is an **auxiliary module**, **ENTER** and **RETURN** instructions are inserted in the flowchart.

![Figure 2-16: New main module](image)
![Figure 2-17: New auxiliary module](image)

The graphical editor displays an **insertion node** on each line linking flowchart instructions. These nodes allow inserting new instructions in the flowchart through **contextual menus** or by **drag and drop** using the mouse and instruction templates in the **template panel**.

### 2.4.3 Manipulating flowchart instructions

**LARP**'s graphical editor panel is a drawing canvas where flowchart instructions may be manipulated (see Figure 2-15). Each line linking adjacent instructions displays an insertion node (a small gray circle) for inserting new flowchart instructions by **drag and drop** or through **contextual menus**.

![Figure 2-18: Selected insertion node](image)
![Figure 2-19: Selected flowchart instruction](image)
Flowchart instructions and insertion nodes may be selected by clicking on the corresponding graphical component with the mouse or by browsing through them using keyboard arrow keys. The current selection is highlighted with a dashed rectangle surrounding the instruction or the node (see Figure 2-18 and Figure 2-19).

A selected instruction may be edited, deleted or moved to an insertion node. On the other hand, a selected insertion node may only receive flowchart instructions by drag and drop or through its contextual menu. It may not be moved nor deleted.

### 2.4.3.1 Inserting, moving and deleting flowchart instructions

New instructions may be inserted into the edited flowchart through *insertion nodes*:

- through the contextual menu displayed when clicking on the selected node with the right mouse button or with the proper keyboard key (see Figure 2-20);
- by drag and drop, using the mouse to drag instruction templates from the template panel and dropping them on targeted nodes.

When an instruction template is dragged from the template panel onto an insertion node, the receiving node's color indicates whether it can accept the instruction or not: a red node indicates it's logically unsound to insert the dragged instruction at this location in the flowchart, while a green node authorizes the insertion.

![Figure 2-20: Inserting a flowchart instruction through an insertion node's contextual menu](image)

An instruction from the edited flowchart may also be moved to another location within the flowchart by drag and drop:

1. Select the flowchart instruction to be moved by clicking on it with the left mouse button.
2. Drag the selected instruction onto the targeted insertion node.
3. Drop the dragged instruction onto the node by releasing the mouse button.

The target node's color indicates whether it can accept the instruction or not: a red node indicates it's logically unsound to move the instruction there, while a green node authorizes the insertion. If the dragged instruction is not dropped on a node, the move is cancelled.

Instructions may also be moved within the flowchart using cut and paste. These commands are accessible through the keyboard or the contextual menus. Note however that access to cut, copy and paste commands is restricted when super user mode is not activated in the shareware version of LARP.

Selected flowchart instructions may be deleted through the top menu, through the selection's contextual menu or by pressing the Del key on the keyboard.

2.4.3.2 Editing flowchart instructions

The selected flowchart instruction's attributes may be modified by double clicking on the selected instruction in the graphical editor or through its contextual menu (accessed by clicking on the selection with the right mouse button). The flowchart instruction editing window (Figure 2-21) displays flowchart instruction attributes and allows modifying each attribute.

![Figure 2-21: Editing a flowchart instruction](image)

Attributes associated with a flowchart instruction depend on the type of instruction. While some instructions are relatively simple (for example sequential operations and comments), others (conditional and iterative structures) include more complex attributes such as orientation of branches, as for the IF-ELSE conditional structure of Figure 2-21.

LARP's menus also provide access to major attribute modification commands, such as:

- orientation flipping, which flips the direction of instruction branches in conditional and iterative structures, and/or
• *transformation*, which allows to transform an instruction to a sibling instruction type (for example, a **WHILE** loop may be transformed into a **REPEAT-UNTIL** loop).

Flowchart instruction editing in *LARP*’s graphical editor is relatively straightforward and intuitive.

### 2.4.4 Search and replace in a flowchart

Search and replace functionalities in *LARP* are available in its graphical editor as well as in its textual editor.

All occurrences of the searched text are highlighted in a flowchart instruction. Likewise, a replace operation updates at once all occurrences of the searched text in a flowchart instruction with the replaced text.

For more information on *LARP*’s search and replace functionalities, see the corresponding section for the textual editor.

### 2.4.5 Zooming the display

The textual editor allows rescaling the display of flowcharts. The **zoom factor** resizes the displayed flowchart from 25% to 200% of its normal size. Such rescaling permits viewing a large flowchart entirely or zooming in on a small section of it.

The graphical editor’s zoom factor is set through the **top menu** or through the graphical editor’s commands. The current zoom factor is continuously displayed in the **status panel**.

### 2.4.6 Instructions highlighting in step execution

When executing a project **step by step**, *LARP*’s graphical editor highlights the next instruction to be executed (left figure below) as well as active **break points** (right figure below):

![Figure 2-22: Next instruction to be executed](image)

![Figure 2-23: Highlighted break point](image)

Highlight colors in the graphical editor are **configurable**.
2.4.7 Graphical editor configuration

Both display colors and text fonts are configurable in LARP’s graphical editor. For more information, consult the section titled Configuration of LARP.

2.4.8 Graphical editor’s edit commands

Edit commands as well as project management commands are accessible through one or more of the followings:

- the top menu,
- contextual menus,
- the control panel,
- the keyboard, and
- the mouse.

Furthermore, LARP’s graphical editor may be used along with the template panel to build flowchart modules by drag and drop.

2.4.8.1 Graphical editor commands accessible through menus

There are two types of menus in LARP:

1. the top menu, displayed at the top of the development environment, and
2. contextual menus, linked to various interface elements in the development environment and accessible with a click of the mouse’s right button on the targeted element.

The graphical editor responds to commands accessible through the top menu (see the Top menu section for a list of these commands) as well as through contextual menus associated with instructions and insertion nodes. These commands include:

- **flowchart editing** commands such as instruction insertion at nodes (see Figure 2-20),
- **instruction manipulation** commands such as content editing, orientation flipping and transformation, and
- **clipboard management** and text search commands.

For a comprehensive description of menu commands, consult the related sections (Top menu, Control panel and Editing a flowchart).

2.4.8.2 Graphical editor commands accessible through the keyboard

Along with commands accessible through menus, LARP’s graphical editor responds to commands which are exclusively invoked through the keyboard:

<table>
<thead>
<tr>
<th>Keys</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;End&gt;</td>
<td>Select the last instruction in the flowchart.</td>
</tr>
<tr>
<td>&lt;Home&gt;</td>
<td>Select the first instruction in the flowchart.</td>
</tr>
<tr>
<td>Keys</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&lt;Enter&gt;</td>
<td>Display the contextual menu associated to the selected instruction or the selected insertion node.</td>
</tr>
<tr>
<td>&lt;Del&gt;</td>
<td>Delete the selected instruction.</td>
</tr>
<tr>
<td>+</td>
<td>Increase the zoom factor.</td>
</tr>
<tr>
<td>-</td>
<td>Decrease the zoom factor.</td>
</tr>
<tr>
<td>&lt;Left&gt;</td>
<td>&lt;Up&gt;</td>
</tr>
<tr>
<td>&lt;Right&gt;</td>
<td>&lt;Down&gt;</td>
</tr>
<tr>
<td>&lt;Shift&gt;+&lt;Left&gt;</td>
<td>&lt;Right&gt;</td>
</tr>
<tr>
<td>&lt;Ctrl&gt;+&lt;Left&gt;</td>
<td>&lt;Right&gt;</td>
</tr>
</tbody>
</table>

Table 2-5: Graphical editor commands accessible through the keyboard

Using the <Shift> or <Ctrl> key along with arrow keys (<Left>, <Right>, <Up> or <Down>) moves the selection to an alternate neighbor instruction or insertion node. Since selecting a neighbor instruction or node may not result in the selection anticipated by the user, using the <Shift> or <Ctrl> key provides an alternate change of selection in some circumstances. In short, if the arrow keys alone do not move the selection to the targeted flowchart component, using the <Shift> or <Ctrl> key along with arrow keys may yield the desired results.

Along with the commands listed above, the graphical editor also responds to top menu commands invoked through their accelerator key.

### 2.4.8.3 Mouse control in the graphical editor

*LARP*'s graphical editor responds in standard ways to mouse actions:

- A left click on an instruction or an insertion node selects the corresponding flowchart component.
- A double click on an instruction with the left mouse button edits the instruction (see Figure 2-21).
- A right click on an instruction or an insertion node displays the contextual menu associated with the selected flowchart component.

The mouse may also be used along with the template panel to insert new instructions in the edited flowchart by drag and drop. The insertion is accomplished by selecting an instruction template from the template panel, dragging it on the graphical editor and dropping it on the insertion node where the new instruction must be inserted in the flowchart. The content of the new instruction may then be modified according to needs.

The mouse is also used to move instructions by drag and drop within the flowchart. The instruction to be moved must first be selected with the mouse, dragged to the insertion node where it must be moved, and finally dropped on this node. The graphical editor completes the operation by moving the instruction to its new position within the flowchart.
When an instruction (either a new instruction from the template panel or an instruction moved within the flowchart) is dragged on an insertion node, the node’s color indicates whether it can accept the instruction or not: a red node indicates that it’s logically impossible to drop the instruction at that position within the flowchart, while a green node indicates that the instruction may be inserted there.

## 2.5 Compilation and execution

Compilation of a *LARP* project consists of verifying the conformity of its pseudo code and flowcharts to *LARP*’s syntax rules and, if so, transforming the modules into executable code. If syntax errors are detected in a module during compilation, appropriate error notifications are displayed in the message panel and no executable code is produced.

When compiling a project, *LARP* presents a window indicating the current state of the compilation process. The **compilation state window** (Figure 2-24) indicates the name of the project being compiled as well as compilation results (i.e. if it successfully generated executable code or not). Indicators point out the total number of warning and error messages produced during compilation and displayed in the **message panel**.

![Compilation state window](image)

When a project has no errors and is successfully compiled, it can be executed via the **execution console**.

Compilation and execution commands are accessible through the **top menu** and the **control panel**.

### 2.5.1 Running a project

When a project is successfully compiled with no syntax error detected, it can be executed through the **execution console**. If the user attempts to execute a project which has not been compiled beforehand, *LARP* automatically compiles it prior to its execution.

When executing algorithms, input and output instructions are handled through the execution console. If a logical error is encountered during execution, an appropriate warning or error message is immediately displayed in the **message panel** and if the error is fatal execution is interrupted.

The user can interrupt execution of a *LARP* project any time by closing the execution console.
2.5.2 Step execution

Step execution allows running an algorithm in a controlled environment. In such context, the user may momentarily suspend the execution of the algorithm, run it one instruction at a time, follow the evolution of each variable’s value and animate the execution of some instructions.

Step execution is started through the top menu (Execute » Execute step-by-step). The control panel also provides a button to activate step execution.

2.5.2.1 Step execution interface

Figure 2-25 presents the step execution window and its various interface elements. This window is automatically displayed along with the execution console when step execution of the algorithm is activated (through the top menu’s command Execute » Execute step-by-step or through the control panel’s corresponding button in the main window of the development environment).

![Step execution window](image)

Figure 2-25: Interface elements of the step execution window

The step execution window’s interface elements are:

1. The control panel: not to be confused with the control panel in the development environment’s main window, the step execution window’s control panel controls the execution of an algorithm. Every button in the control panel provides a specific functionality related to step execution:
   - Runs the next algorithm instruction (in step mode). The execution is automatically paused after running the next instruction. See step execution modes for more information.
   - Animates the execution of the next algorithm instruction. Animation speed is controlled using the arrow buttons under the Walk button. For more information see the section on animation.
   - Activates walk execution mode. The arrow buttons under the Walk button controls walk speed. Walk execution may be temporarily paused by pressing the Pause button. See step execution modes for more information.
Activates continuous execution mode. Continuous execution may be temporarily paused by pressing the Pause button. See step execution modes for more information.

Temporarily suspends walk or continuous execution. See step execution modes for more information.

Ends execution of the algorithm. This button closes the step execution window along with the execution console.

Displays online help on step execution.

Step execution may be stopped at any time by closing the step execution window or by pressing the control panel’s Terminate button.

2. The status panel: the step execution window’s status panel, not to be confused with the status panel in the development environment’s main window, displays various information relating to the execution of the algorithm:

```
| Module: #0004 | 0.25 sec | i is of type integer. |
```

![Figure 2-26 : Status panel of the step execution window](image)

The status panel is divided into three sections:

- The first section identifies the next algorithm instruction to be executed (module name and instruction line or number).
- The second section indicates step execution and animation speed, in seconds. Execution and animation speed can be modified through the arrow buttons under the Walk button in the control panel.
- The third section displays information on the currently selected variable, break point or call stack entry.

3. The variables inspection panel: lists all variables and parameters defined in the algorithm module currently being executed, along with their current value. When a variable is selected in the inspection panel, its value’s type is displayed in the status panel.

For more information on variables inspection, see section 2.5.2.3.

4. The call stack inspection panel: displays the current call stack content. The call stack lists all modules currently in execution leading to the last algorithm instruction executed. The call stack inspection panel lists all modules in the call stack (starting with the main module), along with their respective parameter values. A module’s complete call is displayed in the status panel when selected in the inspection panel.

For more information on call stack inspection, see section 2.5.2.4.

5. The animation panel: animates instructions in step execution. When the next instruction is executed with animation, this panel presents a full motion animation of the evaluation process involved in executing the instruction.

For more information on instruction animation, see section 2.5.2.6.
6. The **break points management panel**: lists all break points currently defined in the algorithm under execution. Break points may be individually deactivated through this panel.

For more information on break points management, see section 2.5.2.5.

The following sections described in details the various interface elements of the step execution window.

### 2.5.2.2 Step execution modes

A project may be executed step by step in three modes:

1. **Single step mode**: a single algorithm instruction is executed. The user may thereafter use the *step execution window* to inspect variables content or the call stack entries. **Animation** may be activated during single step execution. The step execution window’s control panel provides a **Single step** button which executes the next algorithm instruction, then pauses. The **Animate** button also executes the next instruction while animating that execution.

2. **Walking mode**: the project is executed in slow motion, a momentary pause inserted in between each instruction executed. The step execution window’s control panel provides a **Walk** button which executes the project in walk mode. The arrow buttons below the **Walk** button may be used to accelerate or reduce walk speed. Walk speed is displayed (in seconds of delay inserted between instruction execution) on the step execution window’s status panel.

   Execution in walk mode is stopped when the end of the project’s **main module** is reached or when a **break point** is encountered. The **Pause** button in the control panel allows to suspend walk mode execution at any time.

3. **Continuous mode**: continuous mode is similar to walking mode, except no delay is inserting in between instructions during execution. Execution in continuous mode is stopped when the end of the project’s main module is reached or when a break point is reached. The **Pause** button in the control panel allows to suspend continuous mode execution at any time.

   Note that continuous mode execution through the step execution window is considerably slower than **conventional execution** (i.e. without the step execution window), which ignores encountered break points. Managing break points during step execution and refreshing the step execution window’s panel requires considerable processing power, slowing down execution.

When executing a project in single step mode or in walking mode, both the *textual editor* and the *graphical editor* highlight the instruction waiting to execute:

![Figure 2-27: Next instruction to execute (step mode) in the textual editor](image-url)
2.5.2.3 Variables inspection

The variables inspection panel (Figure 2-29) displays the value contained in each variable encountered during step executing the current project module.

The panel lists all variables encountered in the module up to the last executed instruction. If the last executed instruction modified a variable’s content, its entry in the panel is displayed in bold characters.

When a variable is selected within the list, its content’s type is displayed in status panel at the bottom of the step execution window. When the selected variable is a container, the call stack inspection panel is temporarily replaced by another panel, titled Container, which allows to browse through the container’s elements.
Tracking variable values during step execution facilitates identifying and eliminating bugs in a LARP project.

### 2.5.2.4 Call stack inspection

The call stack of a LARP project in execution lists all modules currently in execution, from the main module to the last instruction executed. The call stack inspection panel lists all modules in the call stack along with their respective parameter values:

![Call stack inspection panel](image)

In Figure 2-30, instruction 0003 in module `ComputeExpenses` is next to be executed. That module was invoked by module `MeanExpenses` which was invoked in turn by the main module (PRINCIPAL). The call stack inspection panel displays all parameter values for each call.

A module call is fully displayed in the status panel when selected from the call stack inspection panel.

### 2.5.2.5 Break points

The break points inspection panel (Figure 2-31) lists all active break points during step execution of the current LARP project. A break point is attached to a module instruction and flags to step execution window to pause walk mode or continuous mode execution at the instruction, allowing the user to inspect variables and the call stack, change execution mode or activate animation.

The break points inspection panel lists all break points defined in the project and highlights (in red in Figure 2-31) the break point corresponding the the instruction at which step execution is currently paused:
2. Development environment

Figure 2-31: Break points inspection panel

The break points inspection panel’s contextual menu, accessible through a right click of the mouse on the panel itself, allows to locate a break point in the project and to deactivate break points.

The development environment’s textual editor and the graphical editor both highlight break points attached to module instructions:

Figure 2-32: Highlighted break point in the textual editor
Both editors’ contextual menus as well as the development environment’s top menu provide access to break point activation and suppression commands.

### 2.5.2.6 Animation

The step execution window’s animation panel (Figure 2-34) allows to visualize the evaluation process of algorithmic instructions. Animation aids in understanding how the various components of an instruction get evaluated, and in which order they get evaluated. The value of animation is mainly pedagogic: it provides viewers with insight on the consequence on evaluation of operator priorities when executing instructions involving arithmetic operators, relational operators and logical operators.

Figure 2-34 presents an example of animating an assignment instruction. The animation decomposes the evaluation of the mathematical equation on right of the assignment symbol (=) according to the relative priorities of its arithmetic operations. Here, \( 2^K \) (yielding 4.59479341889914) gets evaluated prior to the multiplication, which evaluates to 16.8628918513565. The add operator then gets evaluated, yielding 37.8628918513565. This last value finally gets assigned to variable \( Z \). At each stage of the evaluation process, the animation panel displays intermediate results obtained from evaluating the various arithmetic operators involved (for example, evaluating variable \( X \) returns 45, which square root gives 6.70820393249937).

Figure 2-35 presents an example in which the condition of a conditional structure (IF statement) gets evaluated. Decomposing the evaluation process of this condition allows to understand why its result is affirmative (i.e. the result is TRUE).
To animate the execution of the next instruction in step by step execution, the **Animate** button must be pressed instead of the **One step** button. The animation panel is then automatically activated and the execution of the next instruction gets animated with pauses inserted in between the evaluation of each arithmetic, relational and/or logical operator. The animation goes on until the instruction’s execution is completed. The duration of pauses inserted in between operator evaluations is the same as pauses inserted in between instructions when step executing a project in **walk mode**. The arrow buttons below the **Walk** button in the step execution window’s control panel (see section 2.5.2.1) may be used to adjust animation speed (as well as walk execution speed). The step execution window’s **status panel** displays the animation speed (duration of each pause in seconds).
Here are important notes regarding animation:

- Once the animation of an instruction is initiated, it cannot be interrupted until the instruction gets completely executed.

- Animation speed may be modified at any time (even during animation) through the arrow buttons under the Walk button.

- Animation is limited to arithmetic, relational and logical operators. The evaluation of other elements of an instruction usually cannot be animated.

- The animation panel always displays the animated instruction in pseudo code form, even if the module containing the instruction is in flowchart form.

### 2.5.3 Security backups

Since technology sometimes fails (more so the programming skills of LARP’s author), there may come a situation in which LARP fatally halts or shuts down without allowing the user to save the last modifications to the project.

To reduce losses and frustrations due to such errors, LARP periodically (by default, every 10 minutes) makes a backup of the edited project in a temporary file. Furthermore, since the likelihood of such malfunction increases during the execution of a project, LARP also performs a backup prior to each execution.

If, as it is generally the case, everything runs smoothly and the user is able to save the project through normal means (via the top menu or the control panel), security backups of the project are automatically discarded. If on the other hand LARP crashes before the user successfully saved his work, security backups are preserved.

At every start up, LARP seeks for the presence of security backups. If one such backup is located, LARP offers the user to reload it in order to recover the last modifications to the project. If the user refuses the offer, the backup file is not reloaded in LARP. In any case, security backups are destroyed afterwards.

### 2.5.4 Warnings and errors

When compiling or executing of a project, LARP displays various information in the message panel:

- **During compilation:** syntax errors encountered in pseudo code and flowchart modules are identified.

- **During execution:** logical errors encountered in pseudo code and flowchart modules are identified.

LARP also displays warning messages on occasions. These messages usually point out potentially non-fatal errors. Such errors do not usually prevent the project from being executed, but may yield unpredictable behaviour during execution.

Every warning and error message displayed in the message panel includes the followings:

1. the type of message (warning or error);

2. the position of the anomaly in the project (the module name along with the position of the erroneous instruction);
3. a short description of the anomaly; and

4. in most circumstances, a reference number relating to further information on the error in LARP's online help.

To locate an erroneous instruction in the project modules, the user can double click on the corresponding message in the message panel. The appropriate editor (either the textual editor or the graphical editor, depending on the type of module) displays the module in error and highlights the faulty instruction. When the message includes a reference number, additional information in online help is available through the message panel's contextual menu, accessible when clicking on the message with the right mouse button.

For more information on warning and error messages, see Appendix E. You can also consult sections describing the message panel and online help to get information on handling these messages in LARP.

2.6 Configuration of LARP

The following elements of LARP's development environment are configurable:

- editing functionalities,
- algorithm execution,
- managing the application, and
- display colors.

Current configuration settings are automatically saved in the system's registry upon shutting down LARP. This configuration is automatically restored when LARP is started up.

Note: It is strongly recommended to avoid altering the configuration of LARP directly through the registry (with a utility such as regedit). Corruption of the system registry can irremediably damage the operating system (i.e. Windows®). It is therefore recommended to use LARP's configuration interface to modify the software's configuration settings.

2.6.1 General configuration

LARP's General configuration window (Figure 2-36) allows configuring various elements of its development environment as well as the execution of algorithms.

This window is accessible through the top menu, under item Options » General…:
Configuration options are organized according to three topics selectable through tabs along the top:

1. Editors: to configure LARP’s editors.
2. Execution: to configure the execution of algorithms.
3. Management: to configure the super user mode and LARP’s integrated updating system.

2.6.1.1 Configuration of editors

Configuration options under the Editors tab allow modifying editing attributes.

The following attributes apply to both the textual editor and the graphical editor:

- **Font**: these attributes indicate the character font and size to use in the editors. It is recommended to use a fixed width font (such as Courier New) in order to facilitate vertical character alignment in output instructions.

- **Size of undo operations buffer**: this value corresponds to the size of the buffer where most recent editing operations are stored. This buffer allows cancelling the most recent editing operations performed by the user. The specified value indicates the maximum number of stored operations.

The following attributes apply exclusively to the textual editor:

- **Contextual tabulation**: when contextual tabulation is activated, the keyboard tabulation key inserts spaces at the caret position so that the character immediately at right of the caret is aligned with the next characters on the previous line. Contextual tabulation eases the alignment of pseudo code instructions on successive lines.

When contextual tabulation is deactivated, the tabulation key inserts a conventional tabulation at the caret position.

- **Syntax highlighting**: when syntax highlighting is activated, reserved words and other syntax elements of pseudo code are displayed with distinct colors in the editor. The highlighting allows the user to easily identify different elements of a pseudo code.
• **Left gutter**: when the left gutter is activated, a narrow gutter appears to the left of the textual edit panel. This gutter shows line numbers and bookmarks associated with the edited document.

• **Width of tabulation columns**: this value indicates the number of spaces corresponding to a conventional tabulation (i.e. when contextual tabulation is not active). For instance, if a value of 4 is specified for this option, pressing the tabulation key inserts blank characters at the caret in order to move it to the next column multiple of 4 (i.e. 4, 8, 12, 16, 20).

### 2.6.1.2 Configuration of the execution console

Configuration options under the **Execution** tab (Figure 2-37) allow configuring parameters driving the execution of algorithms.

![Figure 2-37: Configurating algorithm execution](image)

Configurable parameters include:

• **Maximum size of containers**: this parameter sets the maximum index a container element may have (the minimum index is fixed to 1). This limit prevents the creation of exceedingly large containers, which could cause the system to run out of random access memory (RAM) and make **LARP** terminate abnormally. The creation of huge containers is generally an indication of logical errors in an algorithm.

• **Maximum size of call stack**: this parameter sets the maximum number of embedded module calls allowed during the execution of algorithms (module calls are embedded when one module invokes a second module, which invokes a third module, which invokes a fourth ...). This limit is primarily imposed to prevent infinite recursive calls. See the section on **recursion** for more information.

• **Temporary files directory**: sets the path and name of the directory where are stored various temporary files created by **LARP** during the execution of algorithms. These include the files for managing **input/output buffers** as well as those for **security backups** (to recover work on a project in case of a fatal crash of **LARP**).
It is recommended to let LARP select an appropriate directory for temporary files. This directory must always be accessible and have enough available disk space for LARP to function properly.

To specify the directory where temporary files are to be stored, uncheck the option Let LARP select the directory and select the target directory (either by typing its name in the edit box or by pressing the browser button at right of the edit box):

![Figure 2-38: Selecting a directory for temporary files](image)

### 2.6.1.3 Configuration of the super user mode and the updating system

This section of the guide is relevant exclusively to the shareware version of LARP.

Configuration options under the Management tab (Figure 2-39) allow configuring and validating the detection of super user keys activating the super user mode, and configuring LARP's integrated updating system.

Super user mode is only accessible in the shareware version of LARP when plagiarism prevention functionalities are activated during the installation process. When plagiarism prevention functionalities are not available, the Super user section is not displayed under the Management tab.

![Figure 2-39: Super user mode and updating system configurations](image)
Super user keys deactivate functions related to prevention of plagiarism by enabling super user mode. The drop down list enumerates the key management libraries available on the computer. When a library is selected, the Test button attempts to detect and validate a super user key according to the specified username:

![Unlock test](image)

**Figure 2-40: Unlock test**

LARP's integrated updating system, also available exclusively in the shareware version, is configurable in two ways:

1. Where LARP updates are obtained from.
2. When LARP updates are retrieved.

LARP updates are usually retrieved from a Web server maintained by LARP's author. The default URL address of that server is `http://larp.marcolavoie.ca/en/Updates/updates.inf`. If no address is specified in the box titled Source (Figure 2-39), updates are retrieved from this default server. An alternate updates source may be used by entering the URL address to the file listing the updates. In most circumstances the default source should be used.

When the option Check for updates upon start up of LARP is activated, LARP automatically and silently queries the source for new updates upon each start up. When updates are available, the user is informed and prompted for authorization to download and install them. If the option is deactivated, users must periodically download and install the updates using the corresponding top menu command. It is highly recommended to activate this option in order to have most up-to-date LARP installations.

For more information on super user keys and plagiarism prevention in LARP, consult the section presenting plagiarism prevention functionalities. For more information on updating LARP, see LARP updates.

### 2.6.2 Color selection

LARP allows selecting colors for its editors, its execution console and for step execution. Color configurations are accessible through the top menu (Options » Colors).

The color configuration window (Figure 2-41) contains tabbed panels to select interface elements to configure the Console, the Editors and Step execution elements displayed in the editors. Color configurable elements are listed as radio buttons controlling the attribution of colors. The assignment of colors to interface elements is accomplished via the selection controls located in the right section of the window.

The Default button re-initializes interface elements to their default color, i.e. those attributed upon installation of LARP.
2.6.2.1 Colors in the execution console

The colors used in the execution console to display outputs produced during the execution of algorithms may be configured according to the type of value displayed. The display colors for the following three types of output are configurable:

1. integer numbers,
2. floating numbers, and
3. character strings.

Displaying these outputs with distinct colors allows identifying visually the type of a value displayed in the execution console with WRITE and QUERY instructions.

By default, white is used to display all types of values in the execution console.

2.6.2.2 Colors in the editors

The following color attributes are configurable in LARP’s editors:

- The background color of the edit panel when modules (pseudo code and flowcharts) are edited.
- The background color of the edit panel when input/output buffers (i.e. data) are edited.
- The background color of the gutter (numbering margin at left of the textual editor’s panel).

Furthermore, the textual editor’s syntax highlighting capabilities are configurable. The colors used to display following pseudo code elements are selectable:

- Reserved words of the LARP language
- The name of predefined functions
2. Development environment

LARP Users Guide

- Comments
- Character strings

The color of non-highlighted text in modules and input/output buffers may also be configured (via the radio button titled Others). Note that syntax highlighting does not apply to the content of flowchart instructions.

2.6.2.3 Colors for step execution

The following color attributes relating to step execution are configurable in LARP's editors. Both the textual editor and the graphical editor use this color configuration to display:

- The next instruction to be executed in step by step mode or walk mode.
- Instructions to which are attached break points.
3 Super user mode

This section of the guide is relevant exclusively to the shareware version of LARP. Furthermore, plagiarism prevention functionalities must be activated during software installation in order to have access to super user mode.

To all projects created with the shareware version of LARP is associated the « signature » of its author: the username is permanently inserted into the project file. Furthermore when plagiarism prevention functionalities are activated, no user can alter the username attached to a project file unless in super user mode.

Under normal mode of operations, LARP’s development environment prevents users from plagiarizing project files by restricting access to commands. By disabling specific commands in its menus, LARP prevents the use of functionalities which are likely to be exploited by a user to access and copy the documents in another user’s project files.

Appending the author’s username to a project file, along with other functionalities presented in the following sections, constitute plagiarism prevention measures in LARP. These measures aim at preventing any user from copying other people's project files content, in other words to cheat, in a context of a group of users (i.e. a class of students) being evaluated using LARP. In such context, the super user mode allows the instructor of deactivate plagiarism prevention measures for pedagogic purposes or for class management such as distributing projects or marking homework.

To restrict access to super user mode, super user keys are required to activate it.

3.1 Plagiarism prevention

This section of the guide is relevant exclusively to the shareware version of LARP. Furthermore, plagiarism prevention functionalities must be activated during software installation in order to have access to super user mode.

To prevent plagiarism in the context of a class of students using the shareware version of LARP to build algorithms, some of the development environment commands are deactivated or their functionalities are restricted. The reduced functionalities in LARP aim at preventing users to exchange or to share algorithm modules.

Thanks to command restrictions, the documents of a project created by a student cannot be ported to another student project. Furthermore, any attempt to modify a project file outside the development environment (for instance using a third party text editor) is detected by LARP in most circumstances. LARP automatically refuses to load any illicitly altered project file back into its development environment. Finally, encryption based on username is integrated into clipboard management (used for cut and paste commands within LARP), therefore preventing the use of the clipboard to transfer module content from one student's project to that of another student.

Plagiarism prevention in LARP is based on the uniqueness of usernames. In the context of a class with several students, it is consequently imperative that every student be assigned a distinct username (such as student identification numbers).

Commands restrictions may only be enabled in the shareware version of LARP when plagiarism prevention functionalities are activated during software installation. Plagiarism prevention is not available in the freeware version of LARP.
3. Super user mode

3.1.1 Active username

This section of the guide is relevant exclusively to the shareware version of LARP. Furthermore, plagiarism prevention functionalities must be activated during software installation in order to have access to super user mode.

In order to prevent plagiarism through restrictions in functionalities in the shareware version of LARP, usernames are used to distinguish users in a group. A username is a character string uniquely identifying a user of LARP.

When LARP starts, the user identification window (see Figure 3-1) queries the user for a username. For users with no username, any character string will do (for instance his name or date of birth). The provided string of characters becomes the active username in LARP (as displayed in the status bar). Upon creation of LARP projects, the active username is automatically attached to the project in order to identify the user who created the project, in other words its author.

Configuring LARP

User identification

To prevent plagiarism, LARP requires each user to provide a username to serve as user ID for his projects. To each LARP project created by the user will be incorporated his username.

Since LARP do not allow someone to use the software without providing a username, you must provide one. According to circumstances, you may use as username:

- your student ID
- your employee number
- your date of birth
- you first or last name

Enter your username: 9704567

Use the above username as default username the next times LARP is run

Figure 3-1: Specifying a username

The user identification window provides the option of using the specified active username on subsequent start-ups. If the option Use the above username... is enabled, the specified username will automatically be selected as active username upon subsequent start-ups of LARP and the user will not be prompted for a username.

The active username can be changed at all times through the top menu command Options → Identification.... When changing the active username, no project file must be loaded into the development environment as such project file may not be accessible under the new active username. In such event, the user is informed to close the current project file prior to changing the active username.

If a super user key is present and the active username corresponds to the key's pre-programmed username, LARP's super user mode is activated. For information on the configuration LARP to recognize super user keys, see section Configuration of the super user mode.
The active username is always displayed in the status panel. Usernames usage is only available in the shareware version of LARP when plagiarism prevention functionalities are activated during software installation. Usernames are not available in the freeware version of LARP. Furthermore, project files may not be transferred back and forth between the shareware version and the freeware version of LARP in order to prevent users from using the freeware to exchange project file contents.

### 3.1.2 Username attached to project files

This section of the guide is relevant exclusively to the shareware version of LARP. Furthermore, plagiarism prevention functionalities must be activated during software installation in order to have access to super user mode.

Upon creating a new LARP project (through the top menu or the control panel), the active username (displayed in the status panel) is permanently embedded into the project file. No user (i.e. student) may thereafter change the username linked to the project. When a project is saved into a file, its attached username is saved along.

Prior to loading a project file, LARP first verifies that the username attached to the project (i.e. read from the file) corresponds to the active username displayed in the status panel. Such validation of usernames ensures that the sole user authorized to load a project file into LARP is the project's original author, unless one assumes the author's identity. Even if one assumes the identity of someone else by specifying their username as active username, such illegitimate user cannot modify the username embedded into the file and therefore pretend to be the author of the project file handed to an instructor.

This functionality ensures an instructor that no two students submit the same project file since a unique username is embedded into the project file and it cannot be changed unless in super user mode.

In order to load student project files into LARP without regards to their attached username, an instructor must first activate super user mode. When in super user mode, LARP allows loading any project file, regardless of its attached username.

When creating a project while in super user mode, no username is attached to the project file (i.e. the project has no author). LARP projects without attached username are considered public projects since they may be loaded by anyone without considerations for usernames. When a public project is loaded in LARP, the active username is automatically attached to the loaded project. As a result, when a student loads a public project, modifies it and then saves it back into its file or any other file, his username will hereafter attached to the project file and no one else may thereafter load it back into LARP to plagiarize its contents.

Note that usernames may only be used in the shareware version of LARP when plagiarism prevention functionalities are activated during software installation. Usernames are not available in the freeware version of LARP. Furthermore, project files with attached usernames may not be transferred from the shareware version to the freeware version of LARP in order to prevent users from using the freeware version to exchange project file contents.

### 3.1.3 Document encryption

This section of the guide is relevant exclusively to the shareware version of LARP. Furthermore, plagiarism prevention functionalities must be activated during software installation in order to have access to super user mode.

To prevent a user from accessing the contents of someone else's project file for extracting modules, the shareware version of LARP uses numerical encryption to cipher project files. When LARP saves a project into a file, its documents (i.e. its modules and input/output buffers) are...
encrypted using the active username as encryption key (the public and robust encryption algorithm Blowfish is used in LARP to encode documents).

Since LARP project files are encrypted, an illegitimate user is prevented from using a third party editor to access documents stored in a project file. In other words, the encryption of a project file using the author’s username as encryption key ensures the confidentiality of the LARP project, the author alone being the only one able to decipher its contents.

Furthermore, to prevent an illegitimate user from modifying a LARP project file using a third party editor, an integrity signature is inserted in every LARP project file. If the project file is somehow modified or corrupted, the alterations will be detected automatically by LARP when the project is subsequently loaded in the development environment.

The only mean for an illegitimate user to load a project file created by another user into LARP is to assume the author’s identity (i.e. specifying the author’s username as active username). However, since the user cannot change the username attached to the project, he will not be able to present the project file as his own.

Using the super user mode, an instructor may deactivate document encryption in order to load anyone’s project files (such as project submitted by students) in LARP’s development environment.

Note that usernames to encrypt project files are only used in the shareware version of LARP when plagiarism prevention functionalities are activated during software installation. Usernames are not available in the freeware version of LARP.

3.1.4 Cut and paste restrictions

This section of the guide is relevant exclusively to the shareware version of LARP. Furthermore, plagiarism prevention functionalities must be activated during software installation in order to have access to super user mode.

To prevent the use of the system’s clipboard to transfer modules from one user's LARP project to another user's project, the shareware version of LARP always encrypts module contents using the active username as encryption key prior to copying it into the clipboard. So commands in the development environment involving the clipboard (cut, copy and paste) can only be used to copy contents within modules of a same project or throughout projects of a single user. Consequently, encrypting clipboard contents prevents one from using the clipboard to copy modules from the project of one user to another one's project, therefore preventing plagiarism.

Clipboard contents encryption is only applied to modules (pseudo code and flowchart instructions). Input/output buffer contents are copied to the clipboard in clear text (i.e. not encrypted), therefore allowing the use of the clipboard to copy data from on user's project to another one's project. By definition, input/output buffers contain data or results, both of which are considered public.

By activating super user mode, an instructor can deactivate clipboard contents encryption, allowing the use of the clipboard to cut and paste module contents back and forth between LARP and other applications such as text editors and presentation software.

Note that usernames to encrypt clipboard contents are only used in the shareware version of LARP when plagiarism prevention functionalities are activated during software installation. Usernames are not available in the freeware version of LARP.
3.1.5 Printing restrictions

This section of the guide is relevant exclusively to the shareware version of LARP. Furthermore, plagiarism prevention functionalities must be activated during software installation in order to have access to super user mode.

To prevent a student from producing a paper copy of a LARP project and distribute it to his peers, printing modules is restricted to super user mode in the shareware version of LARP. Input/output buffers can however be printed by anyone, even when super user mode is not activated.

When the print command is invoked, a document selection window is displayed:

![Figure 3-2: Printing documents]

The user can select the module (when super user mode is activated) or input/output buffer to print, or even choose to print all documents within the project. Print options allow selecting printer and paper orientation. When super user mode is not activated, only input/output buffers may be printed.

Note that printing restrictions are only applied in the shareware version of LARP when plagiarism prevention functionalities are activated during software installation. The freeware version of LARP always allows modules printing, as well as the shareware version when plagiarism prevention functionalities are not activated.

3.2 Unlocking the development environment

This section of the guide is relevant exclusively to the shareware version of LARP. Furthermore, plagiarism prevention functionalities must be activated during software installation in order to have access to super user mode.

In a classroom context where students must submit LARP projects as part of the evaluation process, an instructor must be able to deactivate plagiarism prevention functionalities in the shareware version of LARP in order to distribute project files to students and to load projects submitted by students for evaluation purposes. The super user mode allows an instructor to deactivate plagiarism prevention functionalities in order to carry out these tasks.

To prevent students from activating super user mode, a super user key is required to activate it. LARP super user keys are peripherals (key chain sized dongles) to be connected into the computer's USB (Universal Serial Bus) or parallel port. Every super user key is sold pre-configured with a hard coded username.

![Figure 3-3: Super user key (for USB port)]
3. Super user mode

Figure 3-4: Super user key (for parallel port)

Upon startup, \textit{LARP} probes each USB and parallel port of the computer in order to detect the presence of a super user key. If such key is detected and its hard coded username corresponds to the \textit{active username}, super user mode is thereafter activated. When super user mode is activated, the letters \textbf{SU} (for \textit{Super User}) appear in the active username field of the \textit{status panel}:

![Super user mode indicator]

To acquire \textit{LARP} super user keys, see the section titled \textit{Ordering super user keys}.

\textbf{Notes}: the computer ports are probed for the presence of a super user key during the start up process of the shareware version of \textit{LARP} and whenever the active username is changed via the \textit{top menu}. At any other times the super user key may be removed from its port without deactivating super user mode in \textit{LARP}.

Note also that super user keys are only used in the shareware version of \textit{LARP} when plagiarism prevention functionalities are activated during software installation. The shareware version without plagiarism prevention functionalities and the \textit{freeware version} (which does not exploit plagiarism prevention functionalities) do not require super user keys.

3.2.1 Selecting a key technology

\textit{This section of the guide is relevant exclusively to the shareware version of LARP. Furthermore, plagiarism prevention functionalities must be activated during software installation in order to have access to super user mode.}

Since the \textit{shareware version} of \textit{LARP} supports several \textit{super user key} technologies, the software must be configured to use the proper technology according to keys being used. Consult the section titled \textit{Configuration of the super user mode} for more information on specifying a key technology.

3.2.2 Project statistics

When a user creates and works on a \textit{LARP} project, the following statistics are automatically gathered by \textit{LARP}:

- The project’s creation date.
- The total time the project was loaded in \textit{LARP}’s \textit{development environment}.
- The number of times the project was \textit{compiled} and run since its creation.
- The number of times the project was saved in its project file (excluding security backups) since its creation.
These statistics allow an instructor to identify students which may have submitted a LARP project copied from someone else’s work. Suspicious statistics usually suggest plagiarism. For instance, a functional LARP project made of several modules but edited for a few minutes or with few compilations could be considered suspect.

In shareware version of LARP with plagiarism prevention functionalities activated, statistics gathered on a LARP project are exclusively accessible when super user mode is activated, via the top menu. Statistics are always accessible when plagiarism prevention functionalities are disabled during installation or in the freeware version of LARP, with the exception of usernames not being displayed.

3.2.3 Converting flowcharts to pseudo code

LARP allows the user to convert project modules from flowchart to pseudo-code. When a flowchart module is displayed in the graphical editor, the top menu command View » Pseudo code... displays a new window listing the pseudo code equivalent of the edited flowchart:

The Copy to clipboard button allows copying the listed pseudo code into the clipboard in text form. This pseudo code may afterward be reintroduced into the project through cut and paste.
In the shareware version of LARP with plagiarism prevention functionalities activated, the conversion of flowcharts to pseudo code is restricted to super user mode in order for an instructor to impose the use of pseudo code among its students for formulating algorithms with LARP. Flowchart conversion is always accessible when plagiarism prevention functionalities are disabled during installation, as well as in the freeware version of LARP.

### 3.2.4 Public project files

This section of the guide is relevant exclusively to the shareware version of LARP. Furthermore, plagiarism prevention functionalities must be activated during software installation in order to have access to super user mode.

When a project is created in the shareware version of LARP with super user mode activated, the user’s username (i.e. the active username appearing in the status panel) gets permanently embedded into the project, and this user alone is later allowed to reload the project into LARP (see section Username attached to project files).

In such context, how can an instructor distribute a LARP project file to students?

When a project is created while super user mode is activated, no username is attached to the project or its project file. Furthermore, any project file with no attached username may be loaded into LARP whatever the active username (i.e. all users may load the project file), at which point the active username gets permanently attached to the project.

In other words, an instructor may distribute to students a project file created and saved while in super user mode. Such project is called a public project. Any student may then load a public project in LARP’s development environment and modify it in any way. When it gets saved back into its file, the student’s username is automatically attached to the project, therefore preventing another student from accessing the modified project file.

An instructor in super user mode can transform any project into a public project (i.e. with no attached username) via the Reinitialize button in the Statistics window.

Project files created with the shareware version of LARP with plagiarism prevention functionalities deactivated, as well as the ones created with the freeware version of LARP, are always public project files since usernames are not available. Furthermore, project files may not be transferred back and forth between the shareware and the freeware versions of LARP in order to prevent users from using the freeware version to exchange project file contents.
4  My first algorithm

This chapter presents LARP's pseudo code language and flowchart instructions. The language is flexible and intuitive, allowing one to easily formulate algorithms.

This first example illustrates the syntax of pseudo code instructions and flowchart instructions in LARP. It displays on screen the character string **Hello world!**

```
\ \ Very simple pseudo code!
START
  WRITE "Hello world!"
END
```

Pseudo code 4-1: Very simple pseudo code

![Flowchart 4-1: A very simple flowchart](image)

The instructions in the algorithm above are explained in the following sections.

4.1 Comments

As in most programming languages, comments may be inserted into LARP algorithms.

Comments are inserted into pseudo code by preceding text with `\` (two successive backslashes); all text up to the end of the line is then considered as comments and thus ignored during compilation of the algorithm. To extend a comment on several lines, each line must begin with `\`:

```
\ \ Example of a comment extending on
\ \ several lines
START
  WRITE "Hello world!" \ \ Comment at end of line
END
```

Pseudo code 4-2: Comments

Comments are inserted into LARP flowcharts (as depicted in Flowchart 4-1) using the *Comment* instruction (Figure 4-1), available through the template panel or through the graphical editor's contextual menus.
4.2 Begin point and end point of an algorithm

A LARP algorithm must start with the BEGIN instruction. This instruction indicates the starting point for the execution of algorithms. The instruction following BEGIN (in Pseudo code 4-1 and Flowchart 4-1, WRITE "Hello world!") is the first one processed when the algorithm is executed.

Reciprocally, the END instruction indicates where an algorithm ends. The execution of an algorithm stops when the END instruction is reached.

As explained in subsequent sections, a LARP algorithm may be divided into several modules where each module is a distinct pseudo code or flowchart. In such context, one and only one of these modules must have BEGIN and END instructions. These instructions indicate which module starts and ends the execution of the algorithm.

Since an algorithm may contain a single starting point of execution, only one module in the algorithm has a BEGIN instruction. Likewise, that same module is the only one in the algorithm ending with the END instruction. The module containing the BEGIN and END instructions is called the main module. If an algorithm is made of more than one module, the remaining modules are called auxiliary modules.

When a new project is created in LARP's development environment, START and END instructions are automatically inserted into the project’s main module.

4.3 Syntax of instructions

As in all programming languages, LARP provides a set of instructions allowing one to formulate algorithms as pseudo code and flowcharts. These instructions may accept one or more arguments and their syntax corresponds to one of the following formats:

\[ \text{instruction arg1, arg2, ...} \]

or

\[ \text{instruction (arg1, arg2, ...)} \]

The example below (Pseudo code 4-3 and Flowchart 4-2) use the WRITE instruction to display results in the execution console. A single WRITE instruction can produce several results:

```
\// Example of instructions
START
  WRITE "Maximum = ", \text{MAXIMUM}(12, 2, 9)
END
```

Pseudo code 4-3: Syntax of instructions

As noted in this example, WRITE instructions adopt the first syntax form while MAXIMUM instructions adopt the second.
Note also that LARP makes no distinction between uppercase and lowercase letters. Instructions WRITE, UNTIL and SELECTION may therefore be spelled Write, Until and Selection, or write, until and selection.

In LARP's online help, all instructions are formulated in uppercase letters. This convention allows one to quickly identify LARP reserved words in a pseudo code or a flowchart.

### 4.4 Separation of instructions

Instructions in a pseudo code module are generally written on separate lines. So, a change of line (i.e. the insertion of a carriage return at the end of a line) indicates the end of an instruction and the beginning of the following one. It is however possible to extend a long instruction to the following line by ending the first line with the $ symbol:

```
\ Less obvious pseudo code
START
  READ a, b, c
  WRITE "The maximum among the three values ", a, ", ", b, ", and ", $ c, ", is ", MAXIMUM(a, b, c)
END
```

Pseudo code 4-4: Separation of instructions

A long instruction can therefore be extended on several lines, every line except the last ending with $.

The same principle applies to flowcharts. A long instruction may be split into several ones by extending it into subsequent sequential instructions, all but the last one ending with $:
4.5 Creating a *LARP* project

Here are the minimal steps required to produce and run an algorithm with the shareware version of *LARP*. If the freeware version of *LARP* is being used, steps 2 and 3 may be ignored:

1. Start *LARP* and close the *welcome window* (its closes up automatically after a few seconds if the *Close* button is not pressed).

![Welcome window](image)

2. If the shareware is not registered, the *registration window* (Figure 4-3) presents instructions for registering *LARP*. Press the *Register later* button for now. For more information on the registering your *LARP* installation, consult the *Registration* section. Note that this window is not displayed if the shareware is registered or if you are using the freeware version of *LARP*. 

![Registration window](image)
3. When plagiarism prevention functionalities are activated, every user of LARP must be authenticated with a username (Figure 4-4). This username is used to identify authors of LARP projects in an educational environment (i.e. in a class of students). If you do not have a username such as one appointed by a teacher, enter any string of characters and/or digits. If you check the Use above username... box, LARP will automatically assume the specified username on subsequent start-ups:

The user identification window is not displayed when plagiarism prevention functionalities are not activated, which is always the case for the freeware version of LARP.
4. If your LARP installation is registered and your computer connected to the Internet, LARP's integrated update system attempts to determine if new updates are available for your installation. If so, you are prompted for the authorization to download and install the updates. For now you may refuse the updates; you will have the opportunity to install them later.

The integrated update system is not available in the freeware version of LARP.

5. Select the command New » File from LARP's top menu. This command creates a new project or document. Since LARP currently has no project within its development environment, the displayed New window (Figure 4-5) does not allow to create modules nor input/output buffers at this stage. Select the Pseudo code option in order to create a new project using pseudo code for its main module, then press the OK button.

6. The textual editor displays a minimal pseudo code in the new project's main module, delimited by BEGIN and END instructions. Insert a WRITE instruction such as the one presented in the following pseudo code:
To execute the new algorithm, select the **Project » Execute...** command from the **top menu**. This command runs the algorithm, redirecting all inputs and outputs to the **execution console** (Figure 4-7). As stated in the console, press any keyboard key to close the console and go back to the **LARP**'s development environment.

If executing the algorithm failed, it is most probably due to errors in the provided pseudo code instructions. In such cases the execution console is not displayed and appropriate error messages are listed in the **message panel** (Figure 4-8).
8. Note that the message panel shows information related to compilation and execution. If you entered erroneous pseudo code in the editor, errors will be identified in the message panel.

![Figure 4-8: Messages generated during compilation and execution](image)

9. To save your work in a project file, select the File » Save command in the top menu. You must specify a directory and a name for the project file:

![Figure 4-9: Saving the project](image)

That's it! You have just created, executed and saved your first LARP project. You are ready to explore LARP and develop more complex algorithms. LARP User's Guide contains all information necessary to produce a wide variety of algorithms.
5 Constants and variables

In LARP pseudo codes and flowcharts, a constant is a numerical or an alphanumeric value. For instance, 12.3 a numerical constant representing a fractional value, and "hello" is an alphanumeric constant representing a character string. LARP supports several constant types.

A variable in a LARP algorithm (as in most programming languages such as C++ and Java) is a memory location in the computer where data may be stored.

A variable can be seen as a box in which information may be stored and later recovered. Unlike the box which is empty once its content is removed, variables keep a "copy" of the data stored within by the algorithm (with assignment instructions). So, when the algorithm recovers data from a variable, it recovers in fact a copy of that data. The variable therefore retains its data (i.e. its content), which may be recovered repeatedly. In fact, a variable retains its data until other data is stored in the variable, overwriting its previous content, or until the variable is destroyed by the algorithm.

Since an algorithm may use several variables to manage data, each variable in the algorithm is given a unique name attributed by the programmer. The uniqueness of variable names allows the algorithm to identify precisely which variable to manipulate.

Unlike most traditional programming languages (such as C++ and Java), LARP is polymorphic contextual. This term means that a LARP algorithm does not have to explicitly define its variables prior to using them. This programming philosophy is frequently found in scripting languages such as Perl and Lisp. Since the type of a variable depends on its contents and this content may vary during the execution of the algorithm, so does its type.

5.1 Variable names

As in most programming languages, specific rules oversee the selection of variable names:

- A variable name must start with a letter (in Z, has in z) or the underline character (_).
- A variable name must consist of a combination of lowercase letters, uppercase letters, digits and the underline character.
- A variable name may not correspond to a LARP reserved word, such as WRITE, END, IF and PI.
- LARP does not differentiate lowercase and uppercase letters, which means that Name, name and NAME all refer to the same variable.
- LARP ignores accents, what means that akai and akaï refer to the same variable.

Here are examples of valid and invalid variable names:

<table>
<thead>
<tr>
<th>VALID</th>
<th>INVALID</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>101_French</td>
<td>Does not begin with a letter or _</td>
</tr>
<tr>
<td>DATE</td>
<td>Sale Price</td>
<td>The space is not allowed</td>
</tr>
<tr>
<td>_201</td>
<td>Pay&amp;Bonus</td>
<td>Special characters (others than _) are forbidden</td>
</tr>
<tr>
<td>Table10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sale_Price</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Akaï</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Storing values in variables is usually accomplished with the assignment operator (=):

```plaintext
\ Assignment operations
price = 10.20    \ assign a numeric value
name = "John Doe"   \ assign a character string
marks = [75, 56, 94, 69] \ assign a container
```

Pseudo code 5-1: Variable names

Note: containers are presented in a subsequent section.

5.2 Operations

*LARP* is a *polymorphic contextual* language, which means the type of a variable depends on its content. Consequently the type of a variable may vary during the execution of the algorithm.

Using the assignment, an algorithm can store a value of any type in a variable. Furthermore, values of different types may be combined in a single instruction and the conversion of values from a type to another (in order to execute the instruction) is usually transparent.

For example:

```plaintext
a = 1 + 2     \ Yields a = integer 3
a = "1" + 2   \ Also yields a = integer 3
a = "x" + 2   \ Yields a = character string "x2"
a = "1" + "2" \ Yields a = character string "12"
```

Pseudo code 5-2: The operators

In the above example, *LARP* converts character strings to numerical values in order to perform the required mathematical operations (ex: "1" + 2 results in 3). The result of some operations is therefore defined according to the type of values to which they are applied (ex: "a" + "b" results in "ab", while 1 + 2 results in 3).

Operations as depicted in Pseudo code 5-2 are generally found into flowcharts within sequential instructions. They may however also be found in other instructions such as conditions and arguments in module calls.

Flowchart 5-1: Sequential operations

5.3 Numerical values

There are two types of numerical values: *integers* and *floats*. Integer values do not have a fractional part while float values do.
Here are examples of numerical values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2345</td>
<td>Integer in decimal notation</td>
</tr>
<tr>
<td>1234.567</td>
<td>Float in decimal notation</td>
</tr>
<tr>
<td>1.23E-10</td>
<td>Float in scientific notation</td>
</tr>
</tbody>
</table>

**Pseudo code 5-3: Numerical values**

*Scientific notation* allows to express very small numbers (ex: `2.5E-201`) and very large ones (ex: `5E156`). The part after `E` (which can also be written in lower case, `e`) represents a power of `10` multiplying the number before `E`. For example, `2.1E7` is equivalent to `21000000` (i.e. `2.1x10^7`), and `2.1E-7` is equivalent to `0.00000021` (i.e. `2.1x10^-7`).

The range of values which may be manipulated by an algorithm are:

- integer values within `-2147483648` to `-2147483647`, and
- float values within `5.0E-324` to `1.7E308` (i.e. `5.0x10^-324` to `1.7x10^308`).

Any value exceeding these limits in an instruction (for example `1.7E308 * 12`) result in a fatal error during the execution of the algorithm.

### 5.4 Character strings

A character string can be assigned to a variable using the *assignment* operator. There are two equivalent representations of character strings in *LARP*:

<table>
<thead>
<tr>
<th>name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Antonio'</td>
<td>String within single quotes</td>
</tr>
<tr>
<td>&quot;Antonio&quot;</td>
<td>String within double quotes</td>
</tr>
</tbody>
</table>

**Pseudo code 5-4: Character strings**

The availability of these two representations allows the insertion of single or double quotes in character string constants:

<table>
<thead>
<tr>
<th>phrase</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Say &quot;Hello&quot;'</td>
<td>String containing double quotes</td>
</tr>
<tr>
<td>&quot;D'Acosta&quot;</td>
<td>String containing a single quote</td>
</tr>
</tbody>
</table>

**Pseudo code 5-5: Single and double quotes in character strings**

### 5.5 Escape sequences

*LARP* distinctively handles *backslashes* (`\`) found in character strings. A backslash indicates the start of an *escape sequence* representing a special character. An escape sequence consists of the character « `\` » followed by a specific letter.

Here are the escape sequences recognized by *LARP*:

<table>
<thead>
<tr>
<th>Escape sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\n</td>
<td>Carriage return</td>
</tr>
<tr>
<td>\a</td>
<td>Bell (« beep! »)</td>
</tr>
<tr>
<td>\b</td>
<td>Backspace one character</td>
</tr>
<tr>
<td>\ \</td>
<td>Represents the \</td>
</tr>
</tbody>
</table>

**Table 5-1: Escape sequences**
Note that escape sequence `\` must be used to represent the backslash in a character string since a backslash by itself is interpreted as the start of an escape sequence.

Here is an example of use of escape sequences. The instruction rings the bell and shows two lines of text in the execution console:

```
WRITE '\line 1\n\line 2'
```

![Flowchart 5-2: Example of escape sequences](image)

### 5.6 Assignment

As mentioned earlier, the assignment operator (=) assigns a value to a variable. The name of the variable receiving the value is specified on left of the operator, while on right of the operator is specified an expression producing the value to be assigned to the variable:

```
a = 123       \ Variable a receives integer 123
b = "Hello"   \ Variable b receives a string
c = SINUS(10.2) + 1 \ Variable c receives 0.3001253124
```

![Pseudo code 5-6: Assignments](image)

Since LARP is polymorphic contextual it does not require variables to be "declared" prior to being used (as it is case in many programming languages such as C++ and Java). A variable is automatically created on its first value assignment (for instance with the assignment operator).

When no value is assigned to a variable, it is said indeterminate. If an algorithm attempts to display the value of an indeterminate variable in the execution console, the indicator #IND is displayed in red letters to underline the fact that the variable has no assigned value. A warning message is also displayed in the message panel to point out which manipulated variable is indeterminate.
6 Containers

As well as supporting integer values, float values and character strings, LARP also supports grouping data into containers. For those familiar with traditional programming languages such as C++ and Java, a container is a generalization of the array. Contrary to an array which may only contain elements of a common type (it is the case in C++), a LARP container can hold data of different types, including numerical values, character strings and even other containers.

6.1 Grouping values together

In LARP's syntax, a container is a structure able to contain several values at once. Every value stored in a container can be accessed, modified and/or retrieved from the container.

Container constants are expressed using brackets ([ and ]) inside which elements (i.e. values stored in the container) are enumerated, separated with commas (,). The following example creates two containers and assigns them to variables:

```
Days  = ["Mo","Tu","We","Th","Fr","Sa","Su"]   \ Container of strings
Marks = [45, 78, 56, 96, 35]                   \ Container of integer \ values
```

Pseudo code 6-1: Containers

Elements stored in a container may vary in type. For instance, the following container holds various data related to an employee (her name, her identification number, her wages, the year she was hired and the amount she received on her latest four pay checks):

```
Data = ["Jane Doe", 2013345, 56320.00, 1996, $
[1401.98, 1456.02, 1399.57, 1423.41]]
```

Pseudo code 6-2: Container in a container

A container may even contain other containers. In Pseudo code 6-3, the 5th element in variable Data is a container holding four elements.

6.2 Access to container elements

Container elements are accessible through their position in the container. The element's position is specified between brackets ([ and ]) following the name of the variable holding the container. The first element in a container is at position 1. In the following example, two elements of the container assigned to variable a are accessed (the second instruction references the first element of the container, and the last instruction references the third element):

```
a = [10, 2.3E-12, "Monday", -17, 0.234]   \ a is a container
b = a[1] - 3                             \ b = 7
с = a[3]                                 \ c = "Monday"
```

Pseudo code 6-3: Accessing elements in a container
The assignment is used to replace elements of a container. It may also be used to add elements to a container:

\[
\begin{align*}
\text{a} &= [10, 20, 30, 40] \\
\text{a}[2] &= 15 \\
\text{a}[5] &= \text{a}[1] + \text{a}[2] + \text{a}[3] + \text{a}[4] + \text{a}[5] \\
\end{align*}
\]

Flowchart 6-1: Modifying elements of a container

### 6.3 Retrieving container elements

Accessing a container element does not remove the element from the container. Container \( \text{a} \) remains unchanged in the following pseudo code:

```
WRITE a[4]
i = 2
b = a[i] + a[i+1]
WRITE b + a[i-1]
```

Pseudo code 6-4: Accessing container elements

The DESTROY instruction removes an element from a container. This instruction removes the element at the specified position within the given container but does not free its position. The resulting container is said to have an indeterminate element at this position. Indeterminate values are represented with the identifier #IND in the execution console:

```
a = [10, 20, 30, 40]  \\ a is a container
DESTROY a[3]  \\ a = [10, 20, , 40]
WRITE a  \\ displays [10 20 #IND 40]
a[3] = 30  \\ a = [10, 20, 30, 40]
```

Pseudo code 6-5: Indeterminate elements in a container
The **PACK** instruction eliminates indeterminate elements in a container:

```plaintext
a = [10, , 20, , 40]
```

LARP provides **predefined functions** for counting elements in containers:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZE</td>
<td>Returns the number of positions in a container, including those held by indeterminate elements.</td>
</tr>
<tr>
<td>COUNT</td>
<td>Returns the number of defined elements in a container (i.e. excluding those indeterminate).</td>
</tr>
</tbody>
</table>

Table 6-1: Functions for manipulating containers
7 Inputs and outputs

LARP provides instructions for reading data (READ) and for writing results (WRITE). These two instructions interact with the execution console for algorithms to read data from the keyboard and display results on screen. Input/output instructions can also interact with files and input/output buffers to store results and later retrieve it. LARP also provides an instruction to display a query and read the user's answer at once: the QUERY instruction.

As it does for pseudo code, LARP provides a flowchart instruction for handling input/output operations in flowcharts.

When multiple data are read or multiple results displayed in a single input/output instruction, a delimiter character, the separator, is used to separate values in the execution console.

Input/output instructions in LARP are presented in the following sections.

7.1 Input/output instruction for flowcharts

Inputs and outputs are represented in flowcharts with the input/output instruction:

![Figure 7-1: Input/output instruction for flowcharts](image)

The input/output flowchart instruction allows to formulate a read instruction, a write instruction or a query instruction, depending on instruction's attributes. These attributes are specified through the flowchart instruction editing window, displayed when the instruction is edited:

![Figure 7-2: Editing an input/output flowchart instruction](image)
When an attribute is specified in the Read field, the flowchart instruction performs a read operation. When an attribute is specified in the Write field, the flowchart instruction performs a write operation. If attributes are provided for both the Read and Write fields, the flowchart instruction performs a query. The Channel field is for specifying an input/output channel to redirect the operation to files or input/output buffers. Note that query operations may only be directed towards the execution console, and therefore cannot involve an input/output channel. For more information on input/output instructions, see the following sections.

7.2 Read instruction

The READ instruction allows to read one or more values (even containers) into variables. Reading is usually performed through the execution console using the keyboard.

In its basic form, READ is used to read a single value and to assign it to the specified variable:

```
READ a     \ Read a value into variable a
```

Pseudo code 7-1: Reading a value

When executing this instruction, the execution console displays a blinking cursor and waits for the user to input a value through the keyboard. Once the carriage return key is pressed, the entered value is assigned to the given variable (a in the above example). The type of value assigned depends on the format of the entered text:

<table>
<thead>
<tr>
<th>Format of entered text</th>
<th>Examples</th>
<th>Type of value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence of digits</td>
<td>234, -76</td>
<td>Integer</td>
</tr>
<tr>
<td>Sequence of digits with a decimal point or an exponent</td>
<td>2.1, -2E17</td>
<td>Float</td>
</tr>
<tr>
<td>Sequence of characters starting with something else than a digit</td>
<td>Hello, a234</td>
<td>String</td>
</tr>
</tbody>
</table>

Table 7-1: Interpretation of text read

A single READ instruction can read several values when a list of variables is provided:

```
READ a, b, c     \ Read a value for each variable
```

Pseudo code 7-2: Reading several values

While reading inputs, READ considers spaces and carriage returns as value separators. For example, when Pseudo code 7-2 is executed and the following text input in the execution console:

```
Hello 234 -14.78
```

Figure 7-3: Interpreting text entered on a READ instruction

the three provided values are respectively assigned, in order, to the corresponding variables (a = "Hello", b = 234 and c = -14.78).

If the user does not provide enough values for the number of variables to be read, READ waits for further inputs before completing its execution. If the user provides too many values prior to pressing the carriage return, all superfluous values are ignored.
The **Read** attribute in an input/output flowchart instruction is similar to the **READ** instruction in pseudo code:

![Edit (Input/Output)](image)

**Figure 7-4: Reading values in a flowchart**

### 7.3 Write instruction

The **WRITE** instruction writes one or several expression values to the **execution console**. Used alone (i.e. without values), its execution results in a carriage return (i.e. a change of line) written to the execution console. **WRITE** also accepts one or more expressions which values are to be displayed. When more than one expression is provided to **WRITE**, it shows them separated by spaces (the default **separator**).

Pseudo code 7-3 uses **WRITE** instructions to display the value of various expressions:

```plaintext
a = 100
WRITE "HELLO"  \ Write a character string
WRITE          \ Write an empty line
WRITE 1, 10+2, a * 2  \ Write three values
```

**Pseudo code 7-3: Writing the value of expressions**

When Pseudo code 7-3 is executed by **LARP**, the following results appear in the execution console:

```
HELLO
1 12 200
```

**Figure 7-5: Writing to the execution console**

Note that **WRITE** always produces a change of line after writing its last value. To display values without changing line after the last value, the **QUERY** instruction must be used.
The **Write** attribute of an input/output flowchart instruction is similar to the **WRITE** instruction in pseudo code:

![Flowchart diagram](image)

Figure 7-6: Writing expression values in a flowchart

### 7.4 Query instruction

The **WRITE** instruction appends a carriage return to the end of values it displays in the *execution console*. The resulting change of line can be a visual annoyance when there is a need to query the user for an input.

Consider following pseudo code:

```pseudo
WRITE "Enter a number: " \ Query
READ Number
```

**Pseudo code 7-4: Prompting the user for an input**

This pseudo code displays the query on one line, then reads the requested number on the following line since **WRITE** always ends its outputs with a carriage return:

![Query output](image)

Figure 7-7: An inelegant query

The **QUERY** instruction avoids this inconvenience by displaying a string and read values in a single instruction:

```pseudo
QUERY "Enter a number: ", Number \ Query and read
```

**Pseudo code 7-5: The QUERY instruction**
The above instruction displays the query and reads the requested number on the same line in the execution console:

```
Enter a number: 17
```

Figure 7-8: The QUERY instruction

As with the WRITE instruction, QUERY can obtain the query string from the evaluation of an expression. And like the READ instruction, QUERY can read more than one input:

```
Prompt = "Enter three numbers: 
QUERY Prompt, N1, N2, N3           \ \ Query obtained from a variable
```

Pseudo code 7-6: Querying for multiple inputs

Reading inputs is optional in QUERY. When no variable is listed after the query string in a QUERY instruction, the query string is displayed without any reading done afterward. This allows multiple QUERY instructions to display their character string consecutively on the same line.

The format of the Write and Read attributes of an input/output flowchart instruction is similar to the QUERY instruction in pseudo code:

```
QUERY "Enter a number: ", Number
```

Figure 7-9: Insert a query in a flowchart

### 7.5 Separator

By default, READ and QUERY instructions use spaces to distinguish values when reading in the execution console, and the WRITE instruction (as well as QUERY) also uses a space to separate values written to the execution console:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Default interpretation of spaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>Spaces allow to distinguish one value from the next.</td>
</tr>
<tr>
<td>WRITE</td>
<td>A space separates values produced by a single instruction. WRITE also uses the space to separate elements when displaying a container.</td>
</tr>
</tbody>
</table>

Table 7-2: Interpretation of space in READ and WRITE instructions
LARP provides an instruction to change the character to be used as separator:

```
WRITE 10, 20    \ Write two values separated by a space
SEPARATOR ","," \ Changing the separator to comma
WRITE 10, 20    \ Write two values separated by a comma
```

**Pseudo code 7-7: The SEPARATOR instruction**

 Executing the above pseudo code results in the following outputs in the execution console:

```
10 20
10,20
```

**Figure 7-10: Changing the separator**

The separator also has an impact on the READ instruction since the user must enter the active separator to distinguish entered values. For instance, let’s consider following pseudo code:

```
WRITE "Name?"
READ Name
WRITE "Name read = ", Name
SEPARATOR ","    \ Changing the separator
WRITE "Name?"
READ Name
WRITE "Name read = ", Name
```

**Pseudo code 7-8: Changing separator**

If the user inputs the name Jane Doe for both READ instructions during the execution of Pseudo code 7-8, the execution console will display the followings:

```
Name?
Jane Doe
Name read = Jane
Name?
Jane Doe
Name read = Jane Doe
```

**Figure 7-11: Using the separator to properly read data**

An algorithm may thus use an alternate character as separator in order to read character strings with spaces. Note that the carriage return is always considered as a separator, whatever character is identified as active separator with the SEPARATOR instruction.

In flowcharts, the SEPARATOR instruction must be inserted into a sequential instruction:

```
\[\text{SEPARATOR ","}\]
```

**Flowchart 7-1: Changing separator**
8 Operators and predefined functions

In order to support mathematical computations, LARP recognizes all basic arithmetical operations as well as many mathematical functions generally found in traditional programming languages such as Java and C++. LARP also provides several functions for manipulating character strings and containers.

8.1 Arithmetic operators

LARP supports all conventional arithmetic operators. These are presented in increasing order of priority in Table 8-1 (operators of equal priority are listed on the same line):

<table>
<thead>
<tr>
<th>Operators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+, -</td>
<td>Addition and subtraction</td>
</tr>
<tr>
<td>*, /, //, %</td>
<td>Multiplication, division, integer division and modulo</td>
</tr>
<tr>
<td>^</td>
<td>Power</td>
</tr>
<tr>
<td>-</td>
<td>Negation</td>
</tr>
</tbody>
</table>

Table 8-1: Arithmetic operators

The integer division is the division of an integer (i.e. a value with no decimal point) by another integer, yielding an integer result; any residual value resulting from an integer division is ignored. So, 17//5 results in 3, the rest (2) being dropped. The modulo operator (%) returns the residual of an integer division. For example, 17%5 yields 2, which is the residual of 17/5.

LARP evaluates expression elements according to operator priorities, i.e. negations (-) are first evaluated, then powers (^), then multiplications (*), divisions (/ and //) and modulos (%), and finally additions (+) and subtractions (-). Two successive operators of equal priority are evaluated from left to right. Expressions can also be grouped within parentheses in order to circumvent operator priorities:

```
WRITE 7/5     \ displays 1.4
WRITE 7//5    \ displays 1
WRITE 7%5     \ displays 2
WRITE 2^5     \ displays 32
WRITE 4+8/2+1 \ equivalent to 4+(8/2)+1 = 9
WRITE (4+8)/(2+1) \ displays 4
```

Pseudo code 8-1: Arithmetic operators

For educational purposes, animation may be used in step execution to visualize the impact of operator priorities on the evaluation process of arithmetic expressions.

Some arithmetic operators accept only specific type of values. For example both the integer division (/) and the modulo (%) require expressions on both sides of the operator to be integer values. When conversions into integers are required, the fractional part of floating values is eliminated. For instance, 14.8%5 is evaluated as 14%5, resulting in 4.

When an arithmetic expression involves incompatible types which cannot be converted to appropriate types for the targeted operator, LARP stops the execution of the algorithm and displays an error message indicating the incompatibility. Expressions involving indeterminate values also produce an error when evaluated during execution.
8.2 String operators

*LARP* offers one operator for handling character strings:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Concatenation (i.e. joining two strings)</td>
</tr>
</tbody>
</table>

**Table 8-2: Operators for character strings**

The + operator joins end to end (commonly called *concatenation*) two character strings. The example below demonstrates use of this operator (the resulting output is *To be or not to be*):

```
Flowchart 8-1: Concatenating character strings

Character strings involved in a concatenation operation remain unchanged. In the second assignment in the above example, the string in variable *a* is unchanged.

**Warning:** when the + operator involves a character string and a number (integer or float), *LARP* first attempts to convert the character string into a number in order to perform an addition. If conversion fails, the number is then converted into a character string and there is concatenation. Examples below demonstrate these conversions of type in action:

```
a = "12" + 10 \ a = integer 22  
b = 10 + "12" \ b = integer 22  
c = "12z" + 10 \ c = string "12z10"  
d = "12" + "10" \ d = string "1210"  
e = d + 2000 \ e = integer 3210  
f = "12" + "10" + 5 \ f = integer 1215, since the first +  
\ produces "1210"
```

**Pseudo code 8-2: Operator + applied to values of distinct types**
8.3 Container operators

LARP supports two operators for handling containers:

<table>
<thead>
<tr>
<th>Operators</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Concatenation (i.e. joining two containers).</td>
</tr>
<tr>
<td>-</td>
<td>Difference, eliminating from a container elements found in another container.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 8-3: Operators for container</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = [20, 30] + [30, 40] \ \ \ \ a = [20, 30, 30, 40]</td>
</tr>
<tr>
<td>b = 10 + a + 50 \ \ \ \ b = [10, 20, 30, 30, 40, 50]</td>
</tr>
</tbody>
</table>

Pseudo code 8-3: Joining containers

The + operator, commonly called concatenation, is used to join containers end to end or to add elements at the front or at the end of a container. The following example demonstrates the behaviour of this operator:

| a = [1, 2, 3, 2, 4] - [2, 5] \ \ \ \ a = [1, 3, 4] |
| b = a - 3 \ \ \ \ b = [1, 4] |

Pseudo code 8-4: Subtraction of containers

The - operator eliminates from the first container any element found in the second container:

Note that containers involved in the - operation remain unchanged. In the second line of Pseudo code 8-4, the container in variable a is unchanged.

Warning: when operator - eliminates all elements of a container, the returned value is indeterminate (#IND). The reserved word CONTAINER can be used in a test validation to determine if the operation produced a container. In the following example nothing will be displayed since the difference produces an indeterminate result (i.e. no resultant container):

Flowchart 8-2: CONTAINER type validation
8.4 Predefined functions

LARP's predefined functions are categorized according to the type of values on which they may be applied:

1. **mathematical functions**, applicable to numerical values,
2. **functions handling character strings**, and
3. **functions handling containers**.

The following sections present functions in each category. For more information or to get the invocation syntax of these functions, see Appendix C.

### 8.4.1 Predefined mathematical functions

Here are the mathematical constants and functions available in LARP:

<table>
<thead>
<tr>
<th>Functions</th>
<th>Descriptions</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSOLUTE</td>
<td>Returns the absolute (i.e. positive) of the given value.</td>
<td>ABSOLUTE(-6)</td>
</tr>
<tr>
<td>RANDOM</td>
<td>Returns a floating or integer number chosen at random (several versions available).</td>
<td>RANDOM RANDOM(11) RANDOM(2.3, 15.0)</td>
</tr>
<tr>
<td>ARCTANGENT</td>
<td>Returns ( \tan^{-1} ) for the given value in radians.</td>
<td>ARCTANGENT(0.0)</td>
</tr>
<tr>
<td>ROUND</td>
<td>Returns the given value rounded to the closest integer.</td>
<td>ROUND(12.6) returns 13</td>
</tr>
<tr>
<td>COSINUS</td>
<td>Returns the cosine of given value in radians.</td>
<td>COSINUS(1.5707963)</td>
</tr>
<tr>
<td>TOSTRING</td>
<td>Converts to given value to a character string.</td>
<td>TOSTRING(12.34)</td>
</tr>
<tr>
<td>EXP</td>
<td>Returns the basis for natural logarithm (( e )).</td>
<td>LOGE(EXP) returns 1</td>
</tr>
<tr>
<td>LOG10</td>
<td>Returns base 10 logarithm of the given value.</td>
<td>LOG10(100) returns 2</td>
</tr>
<tr>
<td>LOGE</td>
<td>Returns base ( e ) logarithm of the given value.</td>
<td>LOGE(2.1)</td>
</tr>
<tr>
<td>MAXIMUM</td>
<td>Returns the largest value among those given (two values or more).</td>
<td>MAXIMUM(11.1, 12, 7)</td>
</tr>
<tr>
<td>MINIMUM</td>
<td>Returns the smallest value among those given (two values or more).</td>
<td>MINIMUM(11.1, 12, 7)</td>
</tr>
<tr>
<td>PI</td>
<td>Returns the value of mathematical constant Pi.</td>
<td>Area = ( \pi \times r^2 )</td>
</tr>
<tr>
<td>CEILING</td>
<td>Returns the smallest integer greater or equal to the given value.</td>
<td>CEILING(12.1) returns 13</td>
</tr>
</tbody>
</table>
### Functions

<table>
<thead>
<tr>
<th>Functions</th>
<th>Descriptions</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FLOOR</strong></td>
<td>Return the largest integer smaller or equal to the given value.</td>
<td>FLOOR(12.1) (\text{returns}~12)</td>
</tr>
<tr>
<td><strong>SQUAREROOT</strong></td>
<td>Returns the square root of the given value.</td>
<td>SQUAREROOT(25) (\text{returns}~5)</td>
</tr>
<tr>
<td><strong>SINUS</strong></td>
<td>Returns the sinus of given value in radians.</td>
<td>SINUS(1.5707963)</td>
</tr>
</tbody>
</table>

Table 8-4: Predefined mathematical functions

#### 8.4.2 Predefined string functions

*LARP* offers various predefined functions for handling character strings. These functions provide the basic functionality for writing more sophisticated modules handling strings:

<table>
<thead>
<tr>
<th>Functions</th>
<th>Descriptions</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>COUNT</strong></td>
<td>Returns the number of characters in a string (synonym of LENGTH).</td>
<td>COUNT(&quot;Hello&quot;) (\text{returns}~5)</td>
</tr>
<tr>
<td><strong>TOCHARACTERS</strong></td>
<td>Converts a character string into a container where each character is an element.</td>
<td>TOCHARACTERS(&quot;Bye&quot;) (\text{returns}~['B', 'y', 'e'])</td>
</tr>
<tr>
<td><strong>FORMAT</strong></td>
<td>Returns a character string formatting a sequence of arguments according to a format string.</td>
<td>FORMAT(&quot;%5.2f&quot;, 3.1) (\text{returns}~&quot;3.10&quot;)</td>
</tr>
<tr>
<td><strong>LENGTH</strong></td>
<td>Returns the number of characters in a string (synonym of COUNT).</td>
<td>LENGTH(&quot;Hello&quot;) (\text{returns}~5)</td>
</tr>
<tr>
<td><strong>UPPERCASE</strong></td>
<td>Returns the given string with all lowercase letters converted into uppcercases.</td>
<td>UPPERCASE(&quot;Hello&quot;) (\text{returns}~&quot;HELLO&quot;)</td>
</tr>
<tr>
<td><strong>LOWERCASE</strong></td>
<td>Returns the given string with all uppercase letters converted into lowercases.</td>
<td>LOWERCASE(&quot;Hello&quot;) (\text{returns}~&quot;hello&quot;)</td>
</tr>
<tr>
<td><strong>POSITION</strong></td>
<td>Returns the position of the first string within the second string.</td>
<td>POSITION(&quot;cd&quot;,&quot;abcde&quot;) (\text{returns}~3)</td>
</tr>
<tr>
<td><strong>SUBSET</strong></td>
<td>Returns a subset of the given string (the 2nd parameter indicates the start position, and the 3rd indicates the number of characters to extract).</td>
<td>SUBSET(&quot;abcde&quot;, 2, 3) (\text{returns}~&quot;bcd&quot;)</td>
</tr>
</tbody>
</table>

Table 8-5: Predefined functions for handling character strings
8.4.3 Predefined container functions

*LARP* offers various predefined functions for handling containers:

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZE</td>
<td>Returns the number of determinate and indeterminate elements in a container (see DESTROY instruction).</td>
<td>SIZE([1, , 2, ]) returns 4</td>
</tr>
<tr>
<td>COUNT</td>
<td>Returns the number of determinate elements in a container.</td>
<td>COUNT([1, , 2, ]) returns 2</td>
</tr>
<tr>
<td>TOCHARACTERS</td>
<td>Recursively converts the given container into another container where each elements of the first are divided into separate characters in the second.</td>
<td>TOCHARACTERS([&quot;ab&quot;, 2]) returns ['a', 'b', '2']</td>
</tr>
<tr>
<td>TOSTRING</td>
<td>Converts the elements of a container into a single character string.</td>
<td>TOSTRING([10, 20]) returns &quot;1020&quot;</td>
</tr>
<tr>
<td>MAXIMUM</td>
<td>Returns the largest value among the elements of a container.</td>
<td>MAXIMUM([11, 12, 7])</td>
</tr>
<tr>
<td>MINIMUM</td>
<td>Returns the smallest value among the elements of a container.</td>
<td>MINIMUM([11, 12, 7])</td>
</tr>
<tr>
<td>POSITION</td>
<td>Returns the position of an element in the given container.</td>
<td>POSITION(8, [1, 8, 5]) returns 2</td>
</tr>
<tr>
<td>SUBSET</td>
<td>Returns a subset of the given container (the 2nd parameter indicates the start position, and the 3rd indicates the number of elements to extract).</td>
<td>SUBSET([1, 4, 9, 5, 11], 2, 3) returns [4, 9, 5]</td>
</tr>
</tbody>
</table>

Table 8-6: Predefined functions for handling containers
9 Conditional structures

Algorithms presented in previous sections consist in a list of instructions to be executed sequentially, from the Start instruction down to the End instruction.

Many problems solvable through programming require decisions to be made while executing their algorithmic solution. A conditional structure is an instruction which introduces alternative sequences of instructions in an algorithm.

LARP provides four conditional structures:

1. the IF structure,
2. the IF-ELSE structure,
3. the IF-ELSE-IF structure, and
4. the SELECTION structure.

In its simplest form (the IF structure), the pseudo code structure is composed of the reserved words IF, THEN and ENDIF, a condition and an instructions sequence to be executed when the condition is true. In a flowchart it consists of a conditional instruction containing the condition and the instructions sequence along the True branch of the structure, depicting that it is executed when the condition is true:

<table>
<thead>
<tr>
<th>IF conditional structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF condition THEN</td>
</tr>
<tr>
<td>Instructions sequence</td>
</tr>
<tr>
<td>ENDIF</td>
</tr>
</tbody>
</table>

Table 9-1: IF conditional structure

In the above pseudo code structure (at left in Table 9-1), the reserved word IF indicates the beginning of conditional structure and the reserved word ENDIF indicates where it ends. In the corresponding flowchart structure (at right in Table 9-1) the condition starts the conditional structure and the convergence of the True and False branches ends the structure. Note that LARP allows the True branch in a conditional structure to be positioned on right or left of the condition.

Conditional structures are based on the evaluation of the condition which yields a true or false result. The flow of execution depends on that result.
9.1 Conditions

A condition is a comparison. This phrase encompasses the essence of what is a condition. A simple condition is made of three elements:

1. the first value,
2. a comparison operator, and
3. the second value.

Values may be of any type (numericals, character strings or containers) and may be expressed explicitly as constants or implicitly as expressions to be evaluated. In order for a comparison to make sense, compared values must be of the same type or of comparable types.

The comparison operator in a simple condition is called a relational operator. These operators are used to compare the magnitude of two values.

A simple condition can also be a type validation, which are tests used to verify the type of a value. The tested value may be a constant or obtained from an expression.

Furthermore, simple conditions may be grouped together into a compound condition using logical operators.

9.2 Relational operators

Relational operators in LARP are used to compare two values:

<table>
<thead>
<tr>
<th>Operators</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Lower than: ( a &lt; b )</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Lower than or equal to: ( a \leq b )</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than: ( a &gt; b )</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to: ( a \geq b )</td>
</tr>
<tr>
<td>=</td>
<td>Equal to: ( a = b )</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to: ( a \neq b ). The equivalent symbol (&lt;&gt;) is also recognized by LARP.</td>
</tr>
</tbody>
</table>

Table 9-2: Relational operators

Relational operators compare two values of comparable types. By comparable types we mean values that can logically be compared. For example, while an integer value can be compared to a float value, it is illogical to compare a float value to a container.
Here are examples of simple conditions:

```pseudo
\ READ two values
WRITE "Enter two values: "
READ a, b

\ Indicate the smallest of values read
IF a < b THEN
  WRITE "Minimum = ", a
ENDIF
IF a >= b THEN
  WRITE "Minimum = ", b
ENDIF

\ Check if one or both values are zero
IF a*b = 0 THEN
  WRITE "At least one value is 0"
  WRITE "Please enter new values: 
  READ a, b
ENDIF
```

Pseudo code 9-1: Simple conditions

As shown in the examples above, a simple condition may include expressions, and the sequence of instructions within the conditional structure may consist of one or more instructions.

Note that relational operators may also be used to compare character strings or containers:

- Character strings are compared according to alphabetic order and ASCII coding. So, "abc" < "b", but "abc" > "B".

- The equality of containers is assessed according to their elements. Comparisons as recursives when container elements are themselves containers. Operators = and != are the only relational operators applicable to containers.

Warning: relational operators cannot be chained. For instance, the condition `5 < a < 10` is invalid in LARP (as in most programming languages). Logical operators must be used to express such conditions.

### 9.3 Type validation

It is sometimes necessary to validate the type of a value before processing it. It is such the case when a value entered by the user must be validated. LARP provides reserved words INTEGER, FLOAT, STRING and CONTAINER which, when used along with the reserved word IS, validate the type of the value obtained from evaluating a given expression:
Reserved words **DETERMINATE** and **INDETERMINATE** may also be used with reserved word **IS** to test if a variable or a container element is determinate or not:

```
IF Tab[1] IS INDETERMINATE THEN
  COMPRESS Tab
ENDIF
```

**Pseudo code 9-2: Condition testing if a value is indeterminate**

Note that **LARP** recognizes reserved words **DEFINED** and **UNDEFINED** as synonyms of **DETERMINATE** and **INDETERMINATE**, respectively.

### 9.4 Logical operators

**Logical operators** bring together **simple conditions** into « super-condition ». Grouping conditions is sometimes required to stipulate that multiple conditions must be satisfied in order to execute a sequence of instructions. For instance, a **compound condition** is needed to express conditions such as « the value must be greater than zero and lower than 100 » or « the color must be red or green ». Logical operators allow to formulate compound conditions by joining simple conditions.
Three logical operators are available in LARP:

<table>
<thead>
<tr>
<th>Operators</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Both conditions must be satisfied: a &gt; 0 AND a &lt; 100</td>
</tr>
<tr>
<td>OR</td>
<td>At least one of the conditions must be satisfied: a &lt; 1 OR a &gt; 99</td>
</tr>
<tr>
<td>!</td>
<td>Logical negation of a condition: !(a IS INTEGER). Reserved word NOT is equivalent.</td>
</tr>
</tbody>
</table>

Table 9-3: Logical operators

Note that the negation operator (!) is often used with parentheses to denote which condition is reversed.

Unlike relational operators, logical operators can be used to chain a sequence of conditions into a compound condition:

```plaintext
\\ Read and validate a color
WRITE "Enter a color: 
READ color
IF color="blue" OR color="white" OR color="red" THEN
   WRITE "Invalid color"
ENDIF
```

Pseudo code 9-3: Compound conditions

9.5 Priority of operators

As discussed in the section on arithmetic operators, all operators in LARP have a priority level. The priority of an operator determines the order of evaluation of components in an expression or a condition. Table 9-4 presents LARP operators in increasing levels of priority; operators with equal priority are listed on the same line:

<table>
<thead>
<tr>
<th>Operators</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>Logical or</td>
</tr>
<tr>
<td>AND</td>
<td>Logical and</td>
</tr>
<tr>
<td>NOT</td>
<td>Logical negation (symbol ! is equivalent).</td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;, &gt;=, =, !=</td>
<td>Relational operators</td>
</tr>
<tr>
<td>+, -</td>
<td>Addition and subtraction</td>
</tr>
<tr>
<td>*, /, //, %</td>
<td>Multiplication, division, integer division and modulo</td>
</tr>
<tr>
<td>^</td>
<td>Power</td>
</tr>
<tr>
<td>-</td>
<td>Arithmetic negation</td>
</tr>
</tbody>
</table>

Table 9-4: Priority of operators

Upon evaluating expressions or conditions, operator priorities may be circumvent using parentheses.
Here is an example of the order of evaluation of a compound condition according to the priority of involved operators:

\[
! a+2 < 30 \text{ OR } b-c\%2 = 28 \text{ AND } c^{11} > 2000\cdot c+1
\]

Figure 9-1: Compound condition

This condition is equivalent to the following:

\[
(! ((a+2) < 30)) \text{ OR } (((b-(c\%2)) = 28) \text{ AND } ((c^{11}) > ((2000\cdot c)+1)))
\]

Figure 9-2: Order of appreciation of the previous condition

Here is a graphical representation of the order in which operators are evaluated in the condition:

Figure 9-3: Graphical representation of the order in which operators are evaluated

For educational purposes, animation may be used in step execution to visualize the impact of operator priorities on the evaluation process of expressions and conditions.

IF and IF-ELSE structures

There are only two possible forms of IF structures: the structures on right of Table 9-5 is the complete form, while the ones on left are the simple forms.

<table>
<thead>
<tr>
<th>IF structure</th>
<th>IF-ELSE structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF condition</td>
<td>IF condition</td>
</tr>
<tr>
<td>THEN</td>
<td>THEN</td>
</tr>
<tr>
<td>Instructions</td>
<td>Instructions</td>
</tr>
<tr>
<td>sequence</td>
<td>sequence</td>
</tr>
<tr>
<td>#1</td>
<td>#1</td>
</tr>
<tr>
<td>ENDIF</td>
<td>ELSE</td>
</tr>
<tr>
<td></td>
<td>Instructions</td>
</tr>
<tr>
<td></td>
<td>sequence #2</td>
</tr>
<tr>
<td></td>
<td>ENDIF</td>
</tr>
</tbody>
</table>

Table 9-5: Conditional structures IF and IF-ELSE
A *condition* is an expression composed of *relational operators* (it may also contain *arithmetical operators* and *logical operators*) and its value is true or false. It can therefore be:

- a condition, or
- a type validation.

The two conditional structures presented in Table 9-5 are straightforward. When the flow of execution reaches the conditional structure (i.e. the pseudo code line `IF condition THEN`, or the flowchart *condition* box), LARP examines the value obtained from evaluating the *condition*. If the condition is true, the *Instructions sequence #1* is executed. This sequence of instructions may consist of any number and any type of instructions. Once *Instructions sequence #1* is executed, LARP exits the conditional structure. In other words:

- Once the flow of execution reaches the **ELSE** reserved word in pseudo code, LARP jumps directly to the next instruction after the **ENDIF**.

Once the flow of execution reaches the convergence point of branches in the flowchart conditional structure, LARP exits the structure.

On the other hand, if the *condition* is false, the flow of execution jumps over *Instructions sequence #1* to the **ELSE** reserved word in pseudo code, and executes *Instructions sequence #2*. In flowcharts, the flow of execution branches on the *False* edge of the structure.

```c
// IF structure
IF a < b THEN
    WRITE "Minimum = ", a
ENDIF

// IF-ELSE structure
IF a < b THEN
    WRITE "Minimum = ", a
ELSE
    WRITE "Minimum = ", b
ENDIF
```

**Pseudo code 9-4: Conditional structures IF and IF-ELSE**

The *IF* structure corresponds to the case when the false part of the structure is empty. Instead of stating « else there is nothing to do », we simply do not write anything (i.e. no **ELSE** and no *Instructions sequence #2*).
Since \textit{LARP} uses carriage returns to separate instructions in pseudo code, it is important to insert line changes at proper locations in conditional structures. The following pseudo code structures are invalid:

\begin{verbatim}
\<< Erroneous syntax (missing change of line after THEN)\>
IF a < b THEN WRITE "Minimum = ", a ENDIF
\end{verbatim}

\begin{verbatim}
\<< Erroneous syntax (extra change of line before THEN)\>
IF a < b THEN
  WRITE "Minimum = ", a
ENDIF
\end{verbatim}

\textbf{Pseudo code 9-5: Invalid IF structures}

Note that \textit{LARP}'s graphical editor prevents the user from constructing ill-shaped conditional structures.

\section{9.7 Embedded IF-ELSE structures}

Graphically, \textit{IF-ELSE} structures may be visualized as railway switches: it opens way to one of two different paths of instructions depending on a condition's value. There are however situations where two paths are not enough. For instance, an algorithm establishing the state of water according to its temperature may have to choose among three possible answers (solid, liquid or gaseous).

A first solution would be the following:

\begin{verbatim}
WRITE "Temperature of water? 
READ Temp
IF Temp <= 0 THEN \hskip 2cm \text{Is it ice?}
  WRITE "It's ice"
ENDIF
IF Temp > 0 AND Temp < 100 THEN \hskip 2cm \text{Is it liquid?}
  WRITE "It's liquid"
ENDIF
IF Temp >= 100 THEN \hskip 2cm \text{Is it vapor?}
  WRITE "It's vapor"
ENDIF
\end{verbatim}

\textbf{Pseudo code 9-6: Sequence of related IF structures}

Pseudo code 9-6 is rather laborious. Conditions are more or less alike, and the flow of execution must examine three successive conditions all on the same topic, the temperature (i.e. the value of variable \texttt{Temp}). It would be more rational to embed the conditional structures one within the other:
WRITE " Temperature of water? "
READ Temp
IF Temp <= 0 THEN \ Is it ice?
   WRITE " It's ice "
ELSE
   IF Temp > 0 AND Temp < 100 THEN \ Is it liquid?
      WRITE " It's liquid "
   ELSE
      WRITE " It's vapor " \ It's therefore vapor
   ENDIF
ENDIF

Pseudo code 9-7: Structures IF-ELSE embedded one in the other

Pseudo code 9-7 saves on the amount of source code: instead of having three conditions, we only have two conditions. Furthermore, we made savings in terms of execution time on the new algorithm: if the temperature is less than zero, Pseudo code 9-7 writes « It's ice » and the flow of execution jumps directly after the last ENDIF, without examining the other possibilities (which are evidently false).

This second version of the algorithm produces exactly the same results as Pseudo code 9-6, but more efficiently and in a more concise way. Embedded conditional structures are therefore an elegant and efficient approach to simplify and optimize algorithms.

Conditional structures may also be embedded in flowcharts:

It is important to emphasize the danger of embedding conditional structures in pseudo code: every IF...THEN must have a corresponding ENDIF. Missing or superfluous ENDIF keywords often
occur, but are easily spotted when pseudo code instructions are indented appropriately (i.e. pushed to the right inside structures, as in Pseudo code 9-7).

### 9.8 IF-ELSE-IF structure

As described in the previous section, embedded conditional structures reduce execution times since the flow of execution exits the structures as soon as one condition is satisfied and the execution of the corresponding sequence of instructions has completed.

The advantage of embedded conditional structures is however offset by the complexity of the algorithm when several conditions are involved. Pseudo code 9-8 is an example of embedded conditional structures making the algorithm difficult to understand:

```plaintext
WRITE "Temperature of water? 
READ Temp
IF Temp <= 0 THEN
    WRITE "It's frozen"
SINON
    IF Temp <= 12 THEN
        WRITE "It's cold"
    ELSE
        IF Temp <= 25 THEN
            WRITE "It's warm"
        ELSE
            IF Temp <= 75 THEN
                WRITE "It's hot"
            ELSE
                IF Temp <= 100 THEN
                    WRITE "It's very hot"
                ELSE
                    WRITE "It's burning"
                ENDIF
            ENDIF
        ENDIF
    ENDIF
ENDIF
ENDIF
ENDIF
```

**Pseudo code 9-8: Embedded conditional structures**

Note that such pseudo code is confusing. Authors often forget one or several ENDIF when writing algorithms with such exceedingly embedded structures.

Deeply embedded conditional structures are also confusing in flowcharts. In Flowchart 9-4 the embedded structure is very large and cannot fit within the visual area of the graphical editor unless the view is zoomed out so far as to make the text unreadable on screen.
IF-ELSE-IF conditional structures simplify the use of embedded conditional structures in a context where the flow of execution must leave the structure as soon as a condition is satisfied and the corresponding sequence of instructions executed:

**IF-ELSE-IF conditional structure**

```
IF condition #1 THEN
  Instructions sequence #1
ELSE IF condition #2 THEN
  Instructions sequence #2
ELSE IF condition #3 THEN
  Instructions sequence #3
ELSE IF ...
...
ELSE IF condition #n THEN
  Instructions sequence #n
ELSE
  Instructions sequence #n+1
ENDIF
```

Table 9-6: IF-ELSE-IF conditional structure
This conditional structure is used when one and only one sequence of instructions must be executed when a corresponding condition is true. This structure can be interpreted as follow:

- Execute *Instructions sequence #1* if and only if *condition #1* is true.
- Execute *Instructions sequence #2* if and only if *condition #1* is false and *condition #2* is true.
- Execute *Instructions sequence #3* if and only if *condition #1* and *condition #2* are false, but *condition #3* is true.

... 

- Execute *Instructions sequence #i* if and only if *condition #1* to *condition #i-1* are false, but *condition #i* is true.

- Finally, if none of the conditions in the structure is true and the structure has an **ELSE** part, *Instructions sequence #n+1* is executed.

**IF-ELSE-IF** structures are inserted into flowcharts using two *flowchart instructions*:

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="IF-ELSE-IF" /></td>
<td><strong>IF-ELSE-IF</strong> conditional structure: conditional structure consisting of one or more sequences of instructions, one of which is to be executed according to the value of a given conditions.</td>
</tr>
<tr>
<td><img src="image2" alt="Branching" /></td>
<td>Branching for conditional structures: allows to add alternate sequences of instructions in <strong>SELECT</strong> structure and <strong>IF-ELSE-IF</strong> conditional structure.</td>
</tr>
</tbody>
</table>

| Table 9-7: Flowchart instructions required to build an IF-ELSE-IF structure |

Here is the previous example rewritten using an **IF-ELSE-IF** conditional structure:

```
WRITE "Temperature of water? "
READ Temp
IF Temp <= 0 THEN
  WRITE "It's frozen"
ELSE IF Temp <= 12 THEN
  WRITE "It's cold"
ELSE IF Temp <= 25 THEN
  WRITE "It's warm"
ELSE IF Temp <= 75 THEN
  WRITE "It's hot"
ELSE IF Temp <= 100 THEN
  WRITE "It's very hot"
ELSE
  WRITE "It's burning"
ENDIF
```

**Pseudo code 9-9: IF-ELSE-IF conditional structure**

In a **IF-ELSE-IF** pseudo code structure, the last sequence of instructions (*ELSE Instructions sequence #n+1*) is optional. It is also the case in flowchart **IF-ELSE-IF** structures.
While both forms of conditional structures (embedded IF-ELSE and IF-ELSE-IF) are logically equivalent, the latter is preferred because it avoids deep indentation of pseudo code to the right. Such indentation often leaves most of the pseudo code line empty, forces the continuation of lines (using $\$\$\$\$ and diminishes the legibility of the algorithm. The flowchart IF-ELSE-IF structure is also preferred to its embedded counterpart because it’s more linear.

How LARP distinguish between IF-ELSE-IF conditional structures from embedded conditional structures in pseudo code? When reserved words ELSE and IF follow each other on the same line, then LARP assumes that they are part of a IF-ELSE-IF structure.

9.9 SELECT structure

An algorithm may contain a series of decisions in which a variable or an expression is separately tested for each of a set of potential values, and a distinct instructions sequence is consequently executed. LARP offers the SELECT structure for implementing such algorithmic structure.

The SELECT structure replaces the IF-ELSE-IF structure while making the algorithm more readable. Consider the following pseudo code which uses an IF-ELSE-IF structure:

```
READ Value1, Value2, Operator
IF Operator = '+' THEN
    Result = Value1 + Value2
ELSE IF Operator = '-' THEN
    Result = Value1 - Value2
ELSE IF Operator = '*' OR Operator = 'x' THEN
    Result = Value1 * Value2
ELSE IF Operator = '/' THEN
    Result = Value1 / Value2
ELSE
    WRITE "Bad operator"
ENDIF
WRITE Result
```

Pseudo code 9-10: IF-ELSE-IF structure testing a single variable
Now here is an equivalent pseudo code using a SELECT structure (using reserved words SELECT, ELSE and ENDSELECT):

```
READ Value1, Value2, Operator
SELECT Operator
  '+' : Result = Value1 + Value2
  '-' : Result = Value1 - Value2
  '*', 'x' : Result = Value1 * Value2
  '/' : Result = Value1 / Value2
ELSE
  Result = "Bad operator"
ENDSELECT
WRITE Result
```

**Pseudo code 9-11: SELECT structure**

In Pseudo code 9-11 the value of variable `Operator` is compared to each enumerated constant thereafter. If a constant corresponds to the value in `Operator`, the instructions sequence associated to this constant is executed.

If the value in `Operator` does not correspond to the first listed constant, it is compared with the following one, and so on. The ELSE section allows the algorithm to take appropriate measures when the tested variable (Operator) corresponds to none of the listed constants.

Note that

- the tested expression may only be compared with constants (ex: `+'`),
- more than one constant may be associated with the same instructions sequence (ex: `'*'`, `'x'`), and
- the ELSE section is optional.

The flowchart SELECT structure is similar to its IF-ELSE-IF counterpart, except for the tested expression which must be inserted at the entry point of the structure:

**Flowchart 9-6: Example of SELECT structure**
Here is the general form of a SELECT structure:

### SELECT structure

```plaintext
SELECT expression
  List of constants #1: Instructions sequence #1
  List of constants #2: Instructions sequence #2
  List of constants #3: Instructions sequence #3
  ...
  List of constants #n: Instructions sequence #n
ELSE
  Instructions sequence #n+1
ENDSELECT
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>List of constants #1</th>
<th>False</th>
<th>List of constants #2</th>
<th>False</th>
<th>List of constants #3</th>
<th>False</th>
<th>Instructions sequence #n+1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 9-8: SELECT structure

where

- *List of constants* consists of one or more constants, separated by commas; do not forget the colon (:) at the end of the list in pseudo code.

- Each *Instructions sequence* may contain any number of *LARP* instructions. In pseudo-code the first instruction may be positioned after the : symbol, on the same line as the corresponding *List of constants*.

- The *ELSE* section with its instructions sequence are optional. In flowcharts the *Instructions sequence #n+1* is also optional.

When executing a SELECT structure, *LARP* successively compares the value of *expression* with constants found in each *List of constants*, starting with the first list (*List of constant #1*). When a constant corresponding to the value of *expression* is found, its associated *Instructions sequence* is executed and the flow of execution leaves the SELECT structure afterwards. If no constant corresponding to the value of *expression* is found and the structure has an *ELSE* section, the corresponding instructions (*Instructions sequence #n+1*) is executed.

If more than one *List of constants* hold a constant corresponding to the value of *expression*, only the *Instructions sequence* corresponding to the first *List of constants* is executed since the flow of execution leaves the SELECT structure afterwards.
As with the IF-ELSE-IF structure, SELECT structures are inserted into flowcharts using the graphical editor with two flowchart instructions:

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="SELECT structure" /></td>
<td><strong>SELECT structure</strong>: conditional structure consisting of one or more sequences of instructions, one of which is to be executed according to the value of a given mathematical expression.</td>
</tr>
<tr>
<td><img src="image" alt="Branching for conditional structures" /></td>
<td><strong>Branching for conditional structures</strong>: allows to add alternate sequences of instructions in <strong>SELECT</strong> structure and <strong>IF-ELSE-IF</strong> conditional structure.</td>
</tr>
</tbody>
</table>
10 Iterative structures

Algorithms frequently have to repeat a sequence of instructions a given number of times to carry out their tasks; in fact, most algorithms exploit such repetitions. LARP provides three structures to execute a sequence of instructions repeatedly; they are generally referred to as iterative structures or loops:

1. WHILE structure
2. REPEAT-UNTIL structure
3. FOR structure

Iterative structures are based on the evaluation of a condition returning true or false. Repetition is decided according to the value of that condition.

10.1 WHILE structure

In his simplest form (the WHILE structure) as presented in Table 10-1, an iterative structure in pseudo code consists of reserved words WHILE, DO and ENDWHILE, a condition and a instructions sequence to be executed while the condition remains true.

In flowchart form (see Table 10-1), the WHILE structure includes a condition in a six-sided box followed by an instructions sequence along the edge labelled True. The False edge may be oriented towards left or right of the condition. Note that the True edge returns to the condition, illustrating the flow of execution returning to the condition once the execution of instructions sequence is completed.

<table>
<thead>
<tr>
<th>WHILE iterative structure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WHILE</strong> condition <strong>DO</strong></td>
</tr>
<tr>
<td>Instructions sequence</td>
</tr>
<tr>
<td><strong>ENDWHILE</strong></td>
</tr>
</tbody>
</table>

![Flowchart of WHILE iterative structure]

Table 10-1: WHILE iterative structure

In the pseudo code structure above, reserved word WHILE indicates where the iterative structure begins, and reserved word ENDWHILE indicates where it ends. In the flowchart counterparts, the condition box indicates where the structure starts while the bottom end of the False edge indicates its end.
Iterative structures are based on the evaluation of a condition, which result is true or false; the flow of execution is determined accordingly. In a WHILE structure, the instructions sequence is repeatedly executed as long as the condition is satisfied (i.e. it returns True).

The following example uses a WHILE structure to add up values until the sum reaches or exceeds 1000:

\[
\begin{align*}
\text{Sum} &= 0 \\
\text{WHILE} & \text{ Sum < 1000 DO} \\
& \text{READ Value} \\
& \text{Sum} = \text{Sum} + \text{Value} \\
\text{ENDWHILE}
\end{align*}
\]

Pseudo code 10-1: WHILE iterative structure

The condition in a WHILE structure is evaluated before each iteration (an iteration is one and only one execution of the instructions sequence within the loop). The execution of such structure may be so summed up as follows:

1. Evaluate condition
2. If it returns true then
   2.1. Execute instructions sequence
   2.2. Go back to step 1.

The condition must therefore be satisfied for the instructions sequence to be executed. As soon as the condition becomes false (i.e. not satisfied anymore), the flow of execution exits the WHILE structure and resume execution after the structure (passed the ENDWHILE reserved word in pseudo code).
A characteristic of the WHILE structure is the position of the condition at the beginning of the structure. As such, the instructions sequence may not be executed at all if the condition is found to be false upon entering the structure. In such scenario the flow of execution immediately exits the structure without performing a single iteration.

Since iterative structures repeat a sequence of instructions multiple times, they are sometimes referred to as repetitive structures. Both expressions are synonyms.

### 10.2 REPEAT-UNTIL structure

The REPEAT-UNTIL structure is similar to the WHILE structure. It involves executing repeatedly an instructions sequence according to the value of a condition. However, REPEAT-UNTIL and WHILE structures differ in two ways:

1. The WHILE structure executes the instructions sequence as long as the condition is satisfied, while the REPEAT-UNTIL structure executes the instructions sequence as long as the condition is not satisfied. In other words, the REPEAT-UNTIL structure iterates until the condition becomes true.

2. The WHILE structure checks the condition before each iteration while the REPEAT-UNTIL structure checks the condition after each iteration.

The main distinction between the two structures resides in the fact that, in the REPEAT-UNTIL structure, the instructions sequence is executed at least once, without regards to the value of the condition. This distinction is clearly highlighted by the location of the condition at the end of the structure, as opposed to the WHILE structure in which the condition is located at the entry point of the structure.

#### REPEAT-UNTIL iterative structure

<table>
<thead>
<tr>
<th>REPEAT</th>
<th>Instructions sequence</th>
<th>UNTIL</th>
<th>condition</th>
</tr>
</thead>
</table>

![Flowchart of REPEAT-UNTIL iterative structure](image)

Table 10-2: REPEAT-UNTIL iterative structure

The REPEAT-UNTIL structure presented in Table 10-2 consists of reserved words REPEAT and UNTIL, a condition and an instructions sequence to be executed until the condition becomes true (in other words while it is false). In flowchart form the REPEAT-UNTIL structure includes a condition in a six-sided box preceded by an instructions sequence. The False edge, depicting the iteration, may be oriented on left or right of the condition. Note that the True edge exits the structure, illustrating that flow of execution stops executing the instructions sequence once the condition becomes true.
The following example uses a REPEAT-UNTIL structure to read and validate a positive value:

```
REPEAT
  WRITE "Positive number?"
  READ Number
UNTIL Number > 0
```

Pseudo code 10-2: REPEAT-UNTIL iterative structure

Since at least one read is required, the REPEAT-UNTIL structure is preferable to the WHILE
structure because it provides an initial iteration before validation with the condition. The REPEAT-
UNTIL structure is generally preferred to the WHILE structure when variables on which depends
the condition are initialized within the loop, therefore requiring at least one iteration.

Note however that a WHILE structure may replace any REPEAT-UNTIL structure, at the expense
of additional instructions (in Pseudo code 10-3 an extra read operation must precede the loop):

```
WRITE "Positive number?"
READ Number
WHILE Number <= 0 DO
  WRITE "Positive number?"
  READ Number
ENDWHILE
```

Pseudo code 10-3: WHILE structure equivalent to the loop in Pseudo code 10-2
10.3 FOR structure

The third iterative structure is more sophisticated than both previous ones (the WHILE and REPEAT-UNTIL structures). The FOR structure has the following syntax:

```
FOR variable = initial_value TO final_value STEP step_value DO
  Instructions sequence
ENDFOR
```

Here are more formal definitions of identifiers in Table 10-3:

- `variable`: variable which value varies from `initial_value` to `final_value`, updated by `step_value` units after each iteration.
- `initial_value`: value of `variable` on its first iteration.
- `final_value`: value of `variable` on its last iteration.
- `step_value`: increment applied to `variable` after each iteration.
- `Instructions sequence`: instructions executed on each iteration. `Variable` may be used in these instructions.

The step value may be omitted in a FOR structure. In such case, a default step value of 1 is applied to `variable`. The variable used to control a FOR structure is commonly called an iteration variable.
Here is a pseudo code example of a FOR structure. Suppose we want to display temperatures in Fahrenheit corresponding to the first 20 Celsius values on a thermometer. Such listing of temperatures may be produced using a WHILE structure:

```
\ Display temperatures from 0C up to 19C en Fahrenheit
Celsius = 0
WHILE Celsius <= 19 DO
  Fahrenheit = Celsius * 9/5 + 32
  WRITE Celsius, ' = ', Fahrenheit
  Celsius = Celsius + 1
ENDWHILE
```

**Pseudo code 10-4: Iterate from one value to the next with a WHILE structure**

The above structure performs exactly 20 iterations, augmenting by 1 the value of variable `Celsius` at the end of each iteration.

The FOR structure offers a more natural and compact syntax to express iterative structures involving an iteration variable such as `Celsius` in Pseudo code 10-4:

```
\ Display temperatures from 0C up to 19C in Fahrenheit
FOR Celsius = 0 TO 19 DO
  Fahrenheit = Celsius * 9/5 + 32
  WRITE Celsius, ' = ', Fahrenheit
ENDFOR
```

**Pseudo code 10-5: FOR iterative structure**

In this structure, variable `Celsius` is initialized to 0 prior to the first iteration, then successively incremented by 1 (since no step value is provided) at the end of each iteration. Variable incremental is implicit (i.e. there is no need to explicitly increment the variable using an instruction such as `Celsius = Celsius + 1`). When variable `Celsius` reaches 20, the flow of execution exits the FOR structure and continue executing pseudo code after the `ENDFOR`.

Flowchart 10-3 presents the same algorithm as Pseudo code 10-5. The condition in the FOR structure is significantly different from ones found in WHILE and REPEAT-UNTIL structures. All condition elements found in a pseudo code FOR structure are represented in the condition symbol of the flowchart FOR structure: initialization of the iteration variable upon entry (`Celsius = 0`), testing whether to proceed with the next iteration (`Celsius in [0...19]`) and updating the iteration variable at the end of each iteration (`Celsius = Celsius + 1`). Edges within the flowchart FOR structure clearly highlight the flow of execution within the structure:

1. Upon entering the structure, the initialization instruction (`Celsius = 0`) is executed once and only once.
2. The iteration variable is validated according to the range of values it must successively take (`Celsius in [0...19]`). If the variable’s value is within this range then
   2.1 The two instructions within the loop are executed.
   2.2 The iteration variable is updated (`Celsius = Celsius + 1`).
   2.3 The flow of execution returns to step 2 in order to determine whether or not to proceed with another iteration.
The option of specifying a step value within a FOR structure allows one to impose an increment other than the default (which is 1). The following algorithm only converts even valued temperatures from Celsius to Fahrenheit:

\[
\text{Display temperatures 0C, 2C, 4C, 6C ... up to 18C in Fahrenheit}
\]

\[
\text{FOR Celsius = 0 TO 18 STEP 2 DO}
\]

\[
\text{Fahrenheit = Celsius * 9/5 + 32}
\]

\[
\text{WRITE Celsius, ' = ', Fahrenheit}
\]

\[
\text{ENDFOR}
\]

**Pseudo-code 10-6: FOR structure with step value other than 1**

By default a FOR structure increments the value of its iteration variable by one unit after each iteration. However when the given \textit{initial\_value} is superior to \textit{final\_value}, the iteration variable will be automatically reduced by 1 at each iteration:

\[
\text{Display temperatures from 19C down to 0C in Fahrenheit}
\]

\[
\text{FOR Celsius = 19 TO 0 DO}
\]

\[
\text{Fahrenheit = Celsius * 9/5 + 32}
\]

\[
\text{WRITE Celsius, ' = ', Fahrenheit}
\]

\[
\text{ENDFOR}
\]

**Pseudo code 10-7: Backward iterations in a FOR structure**
A negative step value may also be specified to reduce the iteration variable by more than one unit at each iteration:

```markdown
\ Display temperatures 18C, 16C, 14C ... down to 0C in Fahrenheit
FOR Celsius = 18 TO 0 STEP -2 DO
  Fahrenheit = Celsius * 9/5 + 32
  WRITE Celsius, ' = ', Fahrenheit
ENDFOR
```

**Pseudo-code 10-8: Backward iterations in a FOR structure with negative step value**

The value of the iteration variable (Celsius in previous examples) may be used in instructions within the FOR structure, but may not be modified by these instructions. In the following example instruction `READ i` is not allowed by LARP because it attempts to change the value of the iteration variable i. Instruction `WRITE Log(i * 100)` is however permitted since it does not modify variable i. It is also the case with `Fahrenheit = Celsius * 9/5 + 32` in Pseudo-code 10-8: it does not alter iteration variable Celsius.

```markdown
FOR i = 0 TO 10 DO
  WRITE Log(i * 100)
  READ i \ not allowed
ENDFOR
```

**Pseudo code 10-9: Illegal FOR structure**

Even though it is very practical, the FOR structure is not mandatory to programming languages such as LARP. All loops in algorithms may be expressed with a WHILE structure. The key reasons to use a FOR structure is to alleviate the job of programmers by automatically managing iteration variables, as well as to produce more comprehensible algorithms (for example Pseudo code 10-5 is easier to understand than its equivalent, Pseudo code 10-4). In other words, the FOR structure is a special case of the WHILE structure.

Conceptually, the FOR structure is said to be an unconditional loop since the number of iterations it will perform is predetermined and does not depend on instructions within the loop (since they cannot modify the iteration variable). For example in Pseudo code 10-5 the number of iterations to be performed is fixed (20 iterations), as well as in Pseudo-code 10-6 (10 iterations) and in Pseudo code 10-7 (20 iterations). WHILE and REPEAT-UNTIL structures are called conditional loops since the number of iterations such loop perform depends on the instructions within the structure and cannot therefore be determined in advance. It is the case in Pseudo code 10-2 and Pseudo code 10-3 where the value in variable Number is set within the loops (it is not therefore an iteration variable).

Most traditional programming languages offer an iterative structure equivalent to LARP’s FOR loop. For example C++ and Java have `for` loops which resemble LARP’s FOR structure, though exhibiting a more complex syntax.
11 Modules

Modular programming is a technique used to create complex algorithms. It consists in splitting a complex algorithm into smaller ones. Each small algorithm is called a module and carries out a simple task. All modules within an algorithm cooperate in order to perform the algorithm’s overall tasks.

Here are examples of tasks carried out by modules:

- Display a menu of options
- Show formatted results
- Compute data average
- Validate entered data
- Sort data

A module is identified by an unique name and consists of an instructions sequence, starting with reserved word ENTER and ending with reserved word RETURN. The instructions sequence in a module is executed when the name of module is encountered during execution of other modules. The module is said to be invoked.

LARP supports three types of modules:

- Simple modules are not parametrized, accepting no argument during their invocation.
- Parameterized modules accept arguments during their invocation, allowing their execution to be controlled by given parameter values.
- Modules with return value, when invoked, return a result to the invoking module. As parameterized modules, they may accept arguments.

Modules help to structure complex algorithms, allowing to build modular and reusable sequences of instructions. Modules are essentially « reusable pieces of algorithms ».

Managing modules in a LARP project is performed through the document browser and the top menu. They provide commands to create and/or delete modules within a project.

11.1 Module names

Modules in a LARP project must be named according to the following rules:

- The name of module must start with a letter (A to Z, a to z) or the underline character (\_).
- The name of module can be made of lowercase letters, uppercase letters, digits (0 to 9) and the underline character (\_).
- The name of module should not correspond to a reserved word of LARP, such as WRITE, END, IF and PI.
- LARP does not distinguish lowercase letters from its uppercase counterparts, so MODULE_1, Module_1 and module_1 all refer to the same module.
A new module can be added to a project in three ways:

- through the top menu using the command **New » File**, then selecting the type of the new module,
- through the control panel button or
- through the document browser’s contextual menu.

Prior to adding the new module to the project, **LARP** displays the following window which queries the user for the type of module (pseudo code or flowchart) and its name:

![New module dialog box](image)

**Figure 11-1: Creating an auxiliary module**

Note that no two modules within a single project may have the same name.

### 11.2 Main module

When an algorithm is split into several modules, one of them must be the **main module**. As opposed to other modules (usually called **auxiliary modules**) which start with reserved word **ENTER** and ends with **RETURN**, the main module starts with word reserved **START** and ends with **END**:

```
\ Main module
START
  WRITE "Hello"
END
```

**Pseudo code 11-1: Main module**

A **LARP** project must contain one and only one main module since **LARP** starts executing an algorithm at instruction **START** and ends its execution when it reaches the instruction **END**. Therefore only the main module may contain instructions **START** and **END**. On the other hand, a
project may contain none to several auxiliary modules, which must all start with reserved word \texttt{ENTER} and end with reserved word \texttt{RETURN}.

Flowchart 11-1: Main module

When creating a new project, \textit{LARP} automatically creates the project's main module (named \texttt{MAIN}) with its \texttt{START} and \texttt{END} instructions.

11.3 Auxiliary modules

\textit{Simple modules} (i.e. without \textit{parameters}) are used to accomplish straightforward tasks such as displaying menus for the user. Such modules consist of an instructions sequence between reserved words \texttt{ENTER} and \texttt{RETURN}.

In a project, all modules other than the \textit{main module} are called \textit{auxiliary modules}. They usually perform tasks requested by the main module, and sometimes by other auxiliary modules.

Here is an example of an auxiliary module displaying a menu:

\begin{verbatim}
\Auxiliary module Menu
\ENTER
  WRITE "The menu is"
  WRITE " 1 - Read the record"
  WRITE " 2 - Save the record"
  WRITE " 3 - Display the record"
  WRITE " 4 - Modify the record"
  WRITE " 5 - Quit"
\RETURN
\end{verbatim}

Pseudo code 11-2: Auxiliary module

The above module, called \texttt{Menu}, executes its instructions sequentially, from \texttt{ENTER} through \texttt{RETURN}. To execute it, a module (usually another module in the project) invoke the targeted module by its name, preceded by the reserved word \texttt{CALL}:

\begin{verbatim}
\Main module
\START
  \REPEAT \Display the menu
    \CALL Menu
    \QUERY "Command? ", Command
  \UNTIL Command = 5
\END
\end{verbatim}

Pseudo code 11-3: Invoking an auxiliary module
In the main module above, the simple module **Menu** is called (i.e. invoked) to display the menu at every iteration of the **REPEAT-UNTIL** loop. All instructions in **Menu** are executed every time the module is invoked. The following **REQUEST** instruction is executed after each call to **Menu**.

The resulting outputs in the **execution console** are displayed in Figure 11-2, where the user inputs values 1 and 5 for the main module’s **REQUEST** instruction:

```
The menu is
  1 - Read the record
  2 - Save the record
  3 - Display the record
  4 - Modify the record
  5 - Quit
Command? 1
The menu is
  1 - Read the record
  2 - Save the record
  3 - Display the record
  4 - Modify the record
  5 - Quit
Command? 5
```

**Figure 11-2: Outputs from auxiliary module Menu**

Obviously the main module in Pseudo code 11-3 is incomplete since no action is undertaken when the user inputs commands 1 to 5.

In **LARP**, auxiliary modules may also be built as flowcharts. The following module displays the current date formatted as **MM/DD/YYYY**:

```
ENTER

WRITE FORMAT("%d/%d/%d", DATE[2], DATE[3], DATE[1])

RETURN
```

**Flowchart 11-2: Simple auxiliary module as flowchart**

The auxiliary module in Flowchart 11-2 may be invoked from a main module written as pseudo code or as flowchart. Note that there is a flowchart instruction specially defined for invoking auxiliary modules:
11.4 Local variables

A module can use its own variables to perform its tasks. These variables belong exclusively to module and are not shared with other modules in the project. They are called local variables since they are locally accessible by a single module.

Local variables are accessible anywhere between reserved words ENTER and RETURN of an auxiliary module, or START and END of the main module. When two or more modules use the same name for a local variable, these variables are distinct. The following example illustrates this independence:

```
\ Main module
START
  Value = 1
  CALL Module_A
  WRITE Value \ Display 1 as result
END

\ Auxiliary module Module_A
ENTER
  Value = 2
RETURN
```

Even if both modules use a variable named Value, the two variables are distinct. The variable Value in the main module is not modified when auxiliary module Module_A is invoked.

The only way for two modules to share data through variables is by using parameters.

11.5 Auxiliary module parameters

Auxiliary modules may receive values, called parameters, provided when they are invoked (i.e. called) by other modules. Parameters allow the invoking module (i.e. module containing the CALL instruction) to provide data to the invoked module (i.e. module targeted by the CALL instruction).

The parameters of an auxiliary module allow the invoking module to « configure » the execution of the invoked module according to one or several values.
LARP offers two types of parameters:

1. **Value parameters**: the invoking module may transmit values to the invoked module through these parameters, but the invoked module cannot transmit results to the invoking module through these parameters. Auxiliary module parameters are value parameters by default.

2. **Reference parameters**: the invoking module may transmit values to the invoked module through these parameters, and the invoked module can transmit back results to the invoking module through these same parameters. A parameter is identified as reference parameter by preceding its name with reserved word `REFERENCE` in the auxiliary module header.

The following figure illustrates the transfer of data through each type of parameters between two modules, one module (Module_A) invoking the other (Module_B):

![Figure 11-3: Transfer of data through parameters](image)

### 11.5.1 Parameter declarations in module header

Parameters of an auxiliary module are variables listed to the right of reserved word `ENTER` in the module’s first instruction. When more than one parameter are listed, they are separated with commas. In the following example (Pseudo code 11-5 and Flowchart 11-4) V1 and V2 are parameters of the module **Addition**:

```pseudo
\ AUXILIARY MODULE Addition
ENTER V1, V2
  Result = V1 + V2
  WRITE "Sum of", V1, "and", V2, "is", Result
RETURN
```

Pseudo code 11-5: Auxiliary module with parameters
Parameters in an auxiliary module are variables which are to receive values provided when the module is invoked. Values provided through module calls are called *arguments* and must be listed within parentheses following the module’s name in the **CALL** instruction:

```
\// Main module
START
  QUERY "Enter two values: ", N1, N2
  CALL Addition(N1, N2)
END
```

**Pseudo code 11-6: Invoking an auxiliary module with arguments**

When module **Addition** is invoked in Pseudo code 11-6 and Flowchart 11-5, values in arguments **N1** and **N2** are respectively copied into the invoked module’s parameters **V1** and **V2** (value of **N1**...
is copied in \texttt{V1} and value of \texttt{N2} is copied in \texttt{V2}). Since parameters are to receive values provided by arguments, auxiliary module parameters must be variables.

By default, parameters enumerated in a module’s \texttt{ENTER} instruction are said to be value parameters since they are to receive values from arguments. Alternatively, a parameter may be designated as reference parameter by preceding its name with reserved word \texttt{REFERENCE} in the \texttt{ENTER} instruction:

\begin{verbatim}
\%% Auxiliary module Swap
\texttt{ENTER REFERENCE V1, REFERENCE V2}
\hspace{1em} Temp = V1
\hspace{1em} V1 = V2
\hspace{1em} V2 = Temp
\texttt{RETURN}
\end{verbatim}

Pseudo code 11-7: Defining reference parameters

The \texttt{REFERENCE} designator only applies to the single parameter following the reserved word. It must therefore be repeated before each reference parameter in \texttt{ENTER} instructions.

Reference parameters allow the invoked module to return results to the invoking module through the corresponding arguments. For example when a module executes \texttt{CALL Swap(N1, N2)}, the \texttt{Swap} module (Pseudo code 11-7) actually transposes values in arguments \texttt{N1} and \texttt{N2}. For more information on reference parameters see section 11.5.3.

Note that main modules cannot have parameters.

\subsection{11.5.2 Value parameters}

When arguments in a module call are variables, the invoked module works with a copy of each variable provided as argument, even if both the argument variable and the parameter variable have the same name. The argument variable is therefore not modified by the module. In other words, the invoked module may modify its parameter variable but the corresponding argument variable will not be changed accordingly.

The following example clearly illustrates this independence of arguments from their corresponding parameter in a module call:

\begin{verbatim}
\%% Auxiliary module WithoutChange
\texttt{ENTER Letter, Number}
\hspace{1em} \texttt{WRITE Letter, Number} \hspace{1em} \%% Display B and 12 (see call)
\hspace{1em} Letter = 'A'
\hspace{1em} Number = 32
\hspace{1em} \texttt{WRITE Letter, Number} \hspace{1em} \%% Display A and 22
\texttt{RETURN}
\end{verbatim}
In the above main module, variables Letter and Number are not changed by the call to auxiliary module WithoutChange, even if the arguments and their corresponding parameter have identical names.

When an invoked module receives values provided through arguments but cannot change the content of these arguments, its parameters are said to be value parameters. In technical literature the process of providing values to a module through its parameters is named call by value. By default module parameters are value parameters in LARP.

Since an argument is independent from its corresponding value parameter, it may be something else than a variable, such as a constant or the result of an expression:

![Flowchart 11-7: Call by value through value parameters](image)

In this example the first argument is a constant and the second is an expression. As opposed to value parameters which may correspond to variables or expressions as arguments, reference parameters may only correspond to arguments which are variables.

### 11.5.3 Reference parameters

When a parameter listed in an auxiliary module’s ENTER instruction is preceded with reserved word REFERENCE, it is said to be a reference parameter. As opposed to the value parameter which receives the value of its corresponding argument from the CALL instruction, the reference parameter refers to the variable provided as argument in the CALL instruction, even if the argument variable is named differently than the parameter variable.
This distinction between value and reference parameters is best explained through an example. Consider the two auxiliary modules presented in Pseudo code 11-9: both modules have identical pseudo code instructions except for parameter declarations: Swap_1 has value parameters while Swap_2 has reference parameters.

```pseudo
\ \ Auxiliary module Swap_1
ENTER V1, V2
Temp = V1
V1 = V2
V2 = Temp
RETURN

\ \ Auxiliary module Swap_2
ENTER REFERENCE V1, REFERENCE V2
Temp = V1
V1 = V2
V2 = Temp
RETURN
```

Pseudo code 11-9: Declaring value and reference parameters

The following main module calls both auxiliary modules presented in Pseudo code 11-9. While the call to Swap_1 does not modify the content of arguments N1 and N2, the call to Swap_2 does modify both variables (their values are actually swapped):

```pseudo
\ \ Main module
START
N1 = 10
N2 = 20

CALL Swap_1(N1, N2)
WRITE N1, N2 \ \ Displays 10 20

CALL Swap_2(N1, N2)
WRITE N1, N2 \ \ Displays 20 10
END
```

Pseudo code 11-10: Main module

Since auxiliary module Swap_2 uses reference parameters, V1 and V2 are in fact synonyms of argument variables N1 and N2, respectively. Thus when Swap_2 assigns a new value to V1, it actually modifies the value of variable N1 in the main module. The case also applies respectively to reference parameter V2 and argument variable N2. This relationship between Swap_2’s reference parameters (V1 and V2) and the variables provided as arguments when the module is invoked (N1 and N2) does not exist when Swap_1 is invoked since the latter uses value parameters: the values of arguments (in this case values in N1 and N2) are copied into value parameters V1 and V2; when module Swap_1 assign new values to its parameters V1 and V2, their corresponding argument variables (N1 and N2) are not affected.

This relationship between reference parameters and their corresponding arguments imposes a major restriction on arguments: the argument corresponding to a reference parameter must be a variable. In fact, since the reference parameter is a variable referring to its corresponding argument in the module call, that argument must also be a variable in order to receive values potentially assigned to the reference parameter by the invoked module. This restriction does not apply to value parameters since they only receive argument values provided through calls but do not maintain a relationship with these arguments while the auxiliary module is executed.
Reference parameters are identified with reserved word **REFERENCE** in an auxiliary module header instruction **ENTER**, as illustrated in **Swap_2** (Pseudo code 11-9). The same applies to flowchart auxiliary modules:

![Flowchart 11-8: Declaring reference parameters](image)

In technical literature the relationship between reference parameters and their corresponding arguments is named **call by reference**.

### 11.6 Modules with a return value

**Auxiliary modules** usually accept data through their parameters during execution. Modules with a **return value** may also accept arguments, but have the additional feature of returning a result through their **RETURN** instruction when invoked. Modules with return value are used when complex computations must return a result, such as complex mathematical operations or reading a selection from a displayed menu.

The auxiliary module in Flowchart 11-9 calculates and returns through its return value the factorial of its parameter. The factorial of value \( n \) is defined as the product of sequence \( 1 \times 2 \times 3 \times \ldots \times n \):
Flowchart 11-9: Auxiliary module with return value

The return value of a module is specified at the end of its `RETURN` instruction. The value may be the result of an expression: `RETURN I + 1`. There must however be only one return value: `RETURN I, I + 1` is incorrect, but `RETURN [I, I+1]` isn’t since a single container is returned.

Here is another example of auxiliary module with return value. This module displays a menu and reads the user’s selection until a valid command is entered, then finally returns that command:

```
\ \ Auxiliary module Menu
ENTER
REPEAT
  WRITE "The menu is"
  WRITE " 1 - Factorial"
  WRITE " 2 - Addition"
  WRITE " 3 - Quit"
  READ Command
UNTIL Command >= 1 AND Command <= 3
RETURN Command
```

Pseudo code 11-11: Module reading a command based on a menu
When the above module is invoked, the returned value may be recovered in a variable of the calling module through assignment. It may also be directed to another LARP instruction, as depicted in the following main module:

```
\ Main module
START
Operation = CALL Menu
IF Operation = 1 THEN
    READ N
    WRITE CALL Factorial(N)
ELSE IF Operation = 2 THEN
    READ N1, N2
    WRITE N1+N2
ENDIF
END
```

Pseudo code 11-12: Invoking a module with return value

An auxiliary module may only return one result through its return value. However multiple results may be returned by a module through reference parameters.

### 11.7 Alternate call syntax

An auxiliary module is usually invoked with the CALL instruction. However LARP also allows to invoke a module without reserved word CALL, as shown in Pseudo code 11-3:

```
\ Main module
START
Operation = Menu // Invoking Menu module
IF Operation = 1 THEN
    READ N
    WRITE Factorial(N) // Invoking Factorial module
ELSE IF Operation = 2 THEN
    READ N1, N2
    WRITE N1+N2
ENDIF
END
```

Pseudo code 11-13: Alternate syntax for invoking auxiliary modules

To improve legibility it is recommended to use the reserved word CALL when invoking a module without return value, but to avoid using it when invoking a module returning a result. In the following example (Flowchart 11-10) the second call constitutes a more « elegant » instruction than the first one:
Furthermore, the second call syntax corresponds to how LARP’s predefined functions are invoked, such as SQUAREROOT:

```
\ Main module
START
  READ N
  Result = SQUAREROOT(N) / Factorial(N)
END
```

Pseudo code 11-14: Invoking predefined functions
12 Files and input/output buffers

By default, an algorithm reads data through the keyboard and produces results in the execution console (i.e. the screen) when executing. The keyboard (for inputs) and the execution console (for outputs) are said to be standard input/output interfaces in LARP.

In some situations however, an algorithm must process data from sources others than the keyboard. This data is generally stored in documents external to the algorithm. In other cases, the algorithm must store results in a permanent external document where they will not be lost as it is the case when results are displayed in the execution console, which is closed once the execution ends.

LARP supports two external sources of information:

- **Input/output buffers**: these documents are data storage integrated to a LARP project. They allow an algorithm to read and/or write directly in a document inserted into LARP project. As for modules, input/output buffers are identified by an unique name and listed in the document browser.

- **Files**: files are data storage generally located on the computer’s hard disk (or on any hardware device accessible through the computer’s file system). Files are identified with a unique name in the file system of the computer running LARP.

An algorithm reading data or writing results in a file or an input/output buffer is said to perform input-output operations. Such operations are handled through input/output channels. In LARP, information processed through an input/output channel is presented in textual form (i.e. a sequence of characters) in the targeted document. An input/output channel may therefore be viewed as a sequence of characters (such as the keyboard which produces characters and the execution console which displays characters). Input/output channels are manipulated in an algorithm through channel numbers.

12.1 Input/output buffers

A LARP project has a single main module and perhaps one or several auxiliary modules. LARP also allows a project to have « data modules », commonly called input/output buffers. Input/output buffers are accessible through the document browser (see Figure 12-1) in LARP’s development environment.

Input/output buffers are created the same way auxiliary modules are:

- through the File » New... or Project » New... commands in the top menu, or

- through the document browser’s contextual menu, displayed when clicking the right mouse button on the browser.

Rules governing how input/output buffers are named are the same as for module.

Once an input/output buffer is created, the user can insert data in it using the textual editor. The algorithm may then read and write data in a buffer using READ and WRITE instructions. QUERY instructions cannot however be used to access input/output buffers or files.

**Important**: input/output buffers can only be created or deleted by the user through LARP’s development environment. It is not possible for an algorithm to create a new buffer in the project or destroy an existing one during execution.
12. Files and input/output buffers

12.2 Files

As with input/output buffers, LARP can read data or write results using files managed by the computer’s operating system. Here are the distinctions between input/output buffers and files:

- Input/output buffers are managed by LARP while files are managed by the operating system of the computer running LARP. This explains why files are not listed in LARP’s document browser along with input/output buffers.

- In an algorithm, an input/output buffer is identified by its name. Along with its name, a file must also be identified by the path leading to the directory where it is located within the computer’s file system. To open a file an algorithm must therefore provide both its name and the path to its directory.

- Any input/output buffer used in an algorithm must be created prior to the algorithm’s execution through LARP’s development environment. A file can be implicitly created by an algorithm during its execution.

- An input/output buffer is not accessible outside the development environment of LARP. Files are however accessible with any software able to read text files. Therefore a file created with another software may be read by a LARP algorithm, and a file created by a LARP algorithm may be read by other software. This is not case with input/output buffers. Note however that LARP’s development environment can convert input/output buffers into files (with the Project » I/O Buffers » Export command accessible through the top menu), and vice versa (through the Project » I/O Buffers » Import command).

Another major distinction between files and input/output buffers is in how each is opened by the algorithm using the OPEN instruction.
12.3 Input/output channels

Referencing a document (i.e. an input/output buffer or a file) in a LARP algorithm requires using input/output channels. LARP associates a number to every opened document. During its execution an algorithm uses this number to refer to the document. These associated numbers are called input/output channels.

LARP provides 256 input/output channels, numbered 1 through 256. An algorithm can therefore access up to 256 documents simultaneously, input/output buffers and/or files. An input/output channel is associated to a document using an OPEN instruction; this association is terminated when the document is closed (using the CLOSE instruction). An input/output channel cannot be associated simultaneously to more than one opened document. Likewise, two input/output channels cannot be associated simultaneously to a single document. Any violation in these restrictions automatically terminates the execution of the algorithm.

The use of input/output channels in algorithms is well illustrated in the following sections.

12.3.1 Opening a document

The OPEN instruction opens a document so it can be accessed by the algorithm during execution. This instruction allows:

- to assign an input/output channel to a file or an input/output buffer, and
- to specify the access mode to the input/output channel.

An input/output buffer or a file can be associated with no more than one channel at a time, and it remains associated with the channel until it is closed.

Except for assigning a channel to a document (with the OPEN instruction), LARP makes no distinction between accessing an input/output buffer or a file since instructions manipulating channels in LARP (READ, WRITE, CLOSE, ...) use the input/output channel as reference to the opened document.

12.3.2 Opening and input/output buffer

By default, the OPEN instruction associates an existing input/output buffer to the specified input/output channel:

```
\ Open an input/output buffer in read mode
OPEN "DATA" ON 3 IN READMODE
```

Pseudo code 12-1: Opening an input/output buffer

The above instruction assigns channel 3 to the input/output buffer called DATA in order to read its contents.

LARP assumes an OPEN instruction targets an input/output buffer. The specified document may be explicitly designated as input/output buffer using reserved word BUFFER. The following flowchart instruction performs the same operation as the instruction in Pseudo code 12-1:
12. Files and input/output buffers

Flowchart 12-1: Using reserved word BUFFER

Note that a sequential flowchart instruction is used to open documents (input/output buffers and files) in flowchart modules.

As mentioned previously, multiple channels may not be simultaneously assigned to a single buffer. Reciprocally, multiple input/output buffers may not be simultaneously associated to the same channel:

```
\ Open an input/output buffer in read mode
OPEN "DATA" ON 3 IN READMODE \ Error: document already
OPEN "DATA" ON 4 IN READMODE \ Error: channel 3 not available
OPEN "DATA_2" ON 3 IN READMODE \ Error: channel 3 not available
```

Pseudo code 12-2: Invalid OPEN instructions

It is important to remember that an algorithm cannot create input/output buffers during its execution. A LARP project must therefore have all its required input/output buffers created prior to the execution of its algorithm. The development environment provides commands to manage input/output buffers within a project.

All input/output buffers created in a LARP project, as well as their contents, are saved along with the modules in the project file whenever the project is saved.

12.3.3 Opening a file

Accessing files in an algorithm is similar to accessing input/output buffers, using reserved word FILE replacing BUFFER in the OPEN instruction:

```
\ Open a file in read mode
OPEN FILE "C:\DATA.TXT" ON 3 IN READMODE
```

Pseudo code 12-3: Opening a file

The above pseudo code opens for reading the file named DATA.TXT, located in the directory C:\. The location of a file in the computer’s file system is indicated with the path to its directory, using backslashes (\) to walk through the hierarchy of directories in the file system. The double backslash is the escape sequence representing a single backslash (\).

If no directory path is provided along with the file name, LARP assumes the file is located in the current directory (usually the directory where the current LARP project is saved). However, since there are exceptions in this rule, it is recommended to always precede the name of a file with the complete path to its directory.

LARP may fail to open a file for multiple reasons:

- **The file does not exist**: the algorithm attempts to open a nonexistent file for reading.

- **The file is already opened**: the algorithm attempts to open a file which is already opened by the algorithm or by another running application.
• **The input/output channel is not available:** the algorithm attempts to open a file on an input/output channel already associated with another opened document.

• **The file name is invalid:** the specified file name is invalid (the specified directory may not exist, the file name may contain characters unsupported by the Windows® file system, or the supporting hardware media may malfunction or be disabled).

When a file cannot be opened, *LARP* interrupts execution of the algorithm and displays an explanatory error message.

### 12.3.4 Access modes

When opening an input/output buffer or a file, an access mode must be specified:

```
OPEN "DATA" ON 3 IN READMODE  
OPEN "RESULTS" ON 5 IN WRITEMODE
```

Flowchart 12-2: Specifying access mode to a document

Three access modes are supported in *LARP*:

- **READMODE**: allows to read the document contents using READ instructions. If the document does not exist, the algorithm’s execution is interrupted.

- **WRITEMODE**: allows to write results to the document using WRITE instructions. Any previous contents in the targeted document is erased. If the document is a file which does not exist, it is created. If the document is a nonexistent input/output buffer, the algorithm’s execution is interrupted.

- **APPENDMODE**: allows to write results at the end of the document using WRITE instructions. Any previous contents in the targeted document is retained. If the document is a file which does not exist, it is created. If the document is a nonexistent input/output buffer, the execution of the algorithm is interrupted.

The main distinction between **WRITEMODE** and **APPENDMODE** is related to prior contents of documents:

- Opening an input/output buffer or a file in WRITEMODE automatically clears any previous contents of that document (i.e. if the document already exists, its content is erased).

- Opening an input/output buffer or a file in APPENDMODE preserves previous contents, any subsequent WRITE instruction appending outputs at the end of that contents.

Read instructions (**READ**) are exclusively allowed on input/output channels associated to documents opened in **READMODE**. Similarly, write instructions (**WRITE**) are exclusively allowed on input/output channels associated to documents opened in **WRITEMODE** or **APPENDMODE**. Any invalid read or write instruction on an input/output channel automatically stops execution of the algorithm with an appropriate error message displayed.
12.3.5 Closing an input/output channel

Every input/output buffer or file opened by the algorithm must be closed before execution ends. The instruction for closing documents associated to input/output channels is CLOSE:

```
OPEN "DATA" ON 3 IN READMODE
CLOSE 3
```

Pseudo code 12-4: Closing an input/output buffer or a file

Here are the rules related to closing documents in an algorithm:

- Any opened document must be closed. If an algorithm reaches the end of its execution without all opened input/output buffers and files having been closed with the CLOSE instruction, a warning message is displayed in the message panel and all remaining opened documents are automatically closed by LARP.

- The CLOSE instruction makes no distinction among access modes. Channels opened in READMODE, WRITEMODE and APPENDMODE are all closed the same way.

- An opened document must only be closed once. A second CLOSE instruction applied to the same channel interrupts the execution of the algorithm.

- A CLOSE instruction involving a disabled input/output channel (for example a channel not associated with an input/output buffer or a file) causes the algorithm to stop executing.

A CLOSE instruction may close multiple input/output channels at once. They must be enumerated using commas:

```
CLOSE 3, 5, 6
```

Flowchart 12-3: Closing multiple input/output channels at once

12.4 Inputs through input/output channels

Reading data from an input/output buffer or a file is accomplished through the input/output channel associated with the document upon opening:

```
OPEN "DATA" ON 3 IN READMODE
READ Name, Number, Salary FROM 3
```

Pseudo code 12-5: Reading from a document using an input/output channel

The syntax for the READ instruction using an input/output channel is analogous to its syntax for reading from the keyboard; however the targeted input/output channel must be appended to the instruction using reserved word FROM in pseudo code, or specified in the designated field in the flowchart instruction edit window:
12. Files and input/output buffers

Before attempting to read a document's contents, an input/output channel must be associated to the document opened in READMODE. Any attempt to read from an input/output channel associated to a document opened in WRITEMODE or APPENDMODE immediately interrupts execution of the algorithm.

Similarly to reading from the keyboard, READ instructions using input/output channels are subject to the active separator.

12.5 Outputs through input/output channels

Writing data to an input/output buffer or a file is accomplished through the input/output channel associated with the document upon opening:

```
OPEN "RESULTS" ON 5 IN WRITEMODE
WRITE Name, Number, Salary IN 5
```

Flowchart 12-4: Writing in a document using an input/output channel

The syntax for the WRITE instruction using an input/output channel is analogous in its syntax for writing to the execution console; however the targeted input/output channel must be appended to the instruction using reserved word IN in pseudo code, or specified in the designated field in the flowchart instruction edit window.
Before writing to a document, the input/output channel must be associated with the document opened in WRITEMODE or APPENDMODE. Any attempt to write to an input/output channel associated with a document opened in READMODE immediately interrupts execution of the algorithm.

Similarly to writing to the execution console, WRITE instructions using input/output channels are subject to the active separator.

### 12.6 Detecting end of content through input/output channels

Since input/output buffers and files contain limited amounts of data, algorithms often need to verify if an input operation has reached the end of the document. The predefined function ENDOFCONTENT, when applied to an input/output channel in the condition of a conditional structure or an iterative structure, returns true when the end of the document is reached:

```pseudo
code
Sum = 0
OPEN "DATA" ON 3 IN READMODE
REPEAT
  READ Value FROM 3
  Sum = Sum + Value
UNTIL ENDOFCONTENT(3)
CLOSE 3
WRITE Sum
```

*Pseudo code 12-6: Detecting the end of a document when reading*

The predefined function ENDOFCONTENT may only be applied to input/output channels accessible in READMODE. Invoking ENDOFCONTENT on an input/output channel associated with a document opened in WRITEMODE or APPENDMODE immediately interrupts execution of the algorithm.

ENDOFCONTENT is not applicable to the keyboard since no input/output channel may be associated with it. The keyboard cannot obviously run out of content!
Appendix A - Number coding

Computers do arithmetic calculations differently than humans. While we calculate with decimal numbers (base 10, with digits 0 through 9), computers calculate with binary numbers (base 2, with digits 0 and 1).

A.1 Why are computers binary?

Although computers are able to deal with text, play music and videos, or play computer games, in fact they are only capable of doing a single thing: process numbers. Computers are only very efficient adding devices.

Computers represent and manipulate text, sound, pictures and video as numbers. We say they manipulate binary information. Binary information is an information unit that can only have two states: active/inactive, or on/off. This representation is imposed by the physical devices storing and processing information. Since these devices work with electricity, the presence or absence of current in a transistor is used to represent a unit of information.

Computers represent binary information, whatever the physical support, as 1’s and 0’s. They therefore use digits 0 and 1 as mathematical base (contrary to humans who use digits 0 through 9). In spite of this restriction, computers are able of perform computations as complex as the ones humans can do. In fact, the computing speed of computers widely compensates for their limitations in terms of representation. That’s why computers outperform humans on that level, even if they can only handle zeros and ones.

A.2 Decimal representation of numbers

Humans use decimal base numbering to represent numbers. The decimal base provides an alphabet of ten digits (0 to 9). When we write a number, the order of its digits is significant. For example, 3498 is in not the same number as 8439, although they share the same digits. The ordering of digits is the second characteristic of our numerical system notation: its decimal base.

A number such as 3498 can be broken down according to the position of every digit in it:

\[ 3498 = 3 \times 1000 + 4 \times 100 + 9 \times 10 + 8 \times 1 \]

Trailing zeroes appearing in this decomposition come from a multiplying factor positioning every digit in the number:

\[
\begin{align*}
3000 & \text{ is } 3 \times 1000 \\
400 & \text{ is } 4 \times 100 \\
90 & \text{ is } 6 \times 10 \\
8 & \text{ is } 8 \times 1
\end{align*}
\]

Number 3498 may therefore be written in many forms:

\[ 3498 = (3 \times 10 \times 10 \times 10) + (4 \times 10 \times 10) + (9 \times 10) + (8 \times 1) \]

or:

\[ 3498 = 3 \times 10^3 + 4 \times 10^2 + 9 \times 10^1 + 8 \times 10^0 \]

The last decomposition highlights the general method of representation in decimal base numbering: the digits in number 3498 are positioned according to their associated power of 10,
A.3 Binary representation of numbers

The technology used to store and manipulate information in a computer is rudimentary; data is stored in binary form using packets of 0 and 1. By convention, the size of these packets is 8 binary units. A binary unit (represented by 0 or 1) is called a bit. A group of eight bits is called a byte.

How many states can a byte have? Let’s first go back to the decimal base. How many numbers can be represented with three digits in decimal base? In fact, three digits in base 10 allow to represent \(10^3 = 1000\) numbers (i.e. from 0 to 999). Similarly, a byte containing 8 bits in base 2 can represent \(2^8 = 256\) numbers. A byte may therefore encode in binary form numbers 0 through 255 (or -127 through +128 if negative numbers are represented). To represent larger numbers, multiple bytes are required: two bytes (i.e. 16 bits) can represent \(2^{16} = 65536\) numbers, three bytes can represent \(2^{24} = 16777216\) numbers, and so on.

Since computers must handle various types of information, bytes are also used to encode information other than numbers, such as text. Since there are only 26 letters in the alphabet, even when considering uppercase and lowercase letters, accented letters (as in French and Spanish), digits and punctuation marks, the complete set of characters to encode still represents less than 256 elements. A single byte can therefore be used to represent most characters found in English texts.

There is an international standard used to encode letters, digits and punctuation marks using binary form: ASCII (American Standard Code for Information Interchange). ASCII coding establishes a correspondence between frequently used symbols of the alphabet (as well as many other symbols commonly used in typography) and binary numbers 0 to 255. Table 12-1 presents ASCII codes corresponding to characters frequently used in programming.

To understand how numbers are represented in binary, let’s consider encoding numbers into a byte (i.e. 8 bits). The conversion of a binary number to its corresponding decimal number (i.e. base 10) is achieved by applying the decomposition process introduced in decimal base (see the previous section), but this time using a binary base. First we select a binary number (8 bits) at random:

\[10011101\]

This number may be decomposed into base 2 factors with power corresponding to the bit’s position within the byte (from right to left):

\[
1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1 \times 128 + 1 \times 16 + 1 \times 8 + 1 \times 4 + 1 \times 1 = 128 + 16 + 8 + 4 + 1 = 157
\]
The binary number 10011101 therefore represents the unsigned number 157. On the other hand, to convert a decimal number into its binary equivalent, successive powers of 2 must be found within the binary number, starting with the largest possible power down to 0. For example, 157 is converted using successive powers $2^7$ down to $2^0$:

157 contains $1 \times 128 \left(2^7\right)$, remaining 29.
29 contains $0 \times 64 \left(2^6\right)$, still remaining 29.
29 contains $0 \times 32 \left(2^5\right)$, still remaining 29.
29 contains $1 \times 16 \left(2^4\right)$, remaining 13.
13 contains $1 \times 8 \left(2^3\right)$, remaining 5.
5 contains $1 \times 4 \left(2^2\right)$, remaining 1.
1 contains $0 \times 2 \left(2^1\right)$, still remaining 1.
1 contains $1 \times 1 \left(2^0\right)$, and finally nothing remains.

By placing the corresponding power coefficients in order of resolution, we get 10011101.
A.4 Hexadecimal representation of numbers

Since humans are generally not familiar with binary coding, it is difficult to « read » a binary number, i.e. to mentally convert a binary number into its corresponding decimal counterpart. For instance, converting binary 10011101 into decimal 157 requires complex computations which are difficult to carry out mentally. To alleviate binary reading for humans, base 16 coding, commonly called hexadecimal coding, is frequently used in computer science.

Hexadecimal coding sees a byte not as a packet of 8 bits but as two packets of 4 bits each (four bits on left and four bits on right). Four bits allow to encode \(2^4 = 16\) different numbers. A single hexadecimal digit may thus represent 16 different numbers (just as in base 10, one digit allows to represent 10 numbers, from 0 to 9).

In hexadecimal coding, to the first ten digits of decimal coding (0 to 9) are added the first 6 letters of the alphabet (A to F). By convention, A represents 10, B is 11, C is 12, D is 13, E is 14 and F is 15. Digits used to represent numbers in hexadecimal coding are therefore 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.

The following table lists decimal and binary correspondences for the hexadecimal digits:

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>A</td>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
<td>1100</td>
</tr>
<tr>
<td>D</td>
<td>13</td>
<td>1101</td>
</tr>
<tr>
<td>E</td>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>F</td>
<td>15</td>
<td>1111</td>
</tr>
</tbody>
</table>

Table 12-2: Correspondences between hexadecimal, decimal and binary coding

Since every hexadecimal digit can represent 16 numbers or 4 bits \(2^4 = 16\), two hexadecimal digits can represent a byte. For instance, decimal number 157 is represented in binary as follows:

\[10011101\]

This byte can be represented in hexadecimal form by dividing its bits into two packets of 4 bits each, then converting each packet to the corresponding hexadecimal digit:

\[1001\quad 1101\]

\[9\quad D\]

So binary number 10011101 is represented in hexadecimal by 9D, which is 157 in decimal.

Hexadecimal coding is commonly used to represent individual bytes since the value of a byte may easily be read as two digits (9D) in hexadecimal form rather than eight zeros and ones (10011101) in binary. Furthermore, when the hexadecimal to binary conversion table (Table 12-2)
is memorized, it is easy for a programmer to mentally convert hexadecimal numbers to binary numbers and vice versa.
Appendix B - Recursion

In algorithms, modules sometimes adopt a strategy often used in mathematics: recursion. Recursive programming is presented in this appendix using the mathematical factorial as example.

The formula for computing the factorial of an integer \( n \), designated \( n! \), is the following:

\[ n! = 1 \times 2 \times 3 \times \ldots \times n \]

This equation can be reformulated in recursive form:

\[ n! = (n-1)! \times n \]

The latter equation is said recursive because its defines the factorial of a number as that number multiplied by the factorial of the previous number. In other words the factorial is defined using the factorial. Although it seems to circle to nothing, it actually makes sense because \((n-1)!\) must first be computed in order to get \( n! \). Since \((n-1)! = (n-2)! \times (n-1)\), \((n-2)!\) must be computed beforehand, and so on. Since by definition \(0! = 1\), the process of computing the factorial of the previous number stops at \( n = 0 \), which is 1 by definition.

As in most programming languages, LARP allows to write recursive modules. The following module (Pseudo-code B-1) computes the factorial of a given parameter by recursion. The function multiplies the given number by the factorial of the previous number. The factorial of the previous number is in turn computed using the Factorial module. In other words, the module calls itself a certain number of times. This is recursion.

Here is a pseudo code module using recursion to compute the factorial of its parameter \( n \):

```plaintext
\{ Factorial module
ENTER n
IF n = 0 THEN
    nFac = 1
ELSE
    nFac = n * Factorial(n-1)
ENDIF
RETURN nFac
\}
```

Pseudo code B-1: Computing the factorial (recursive module)

In this module, the line of pseudo code recursively calling back the module is preceded by a condition ensuring that the recursive calls will eventually stop (when \( n \) reaches 0). Without such condition, recursion would continue indefinitely. This is a basic characteristic of recursive programming: there must be a condition which will eventually stop the recursive calls.

Recursion may sometimes be considered as an alternative to iterative structures. For instance the Factorial module defined previously may be rewritten in iterative form, without recursion, as shown in the next pseudo code (Pseudo-code B-2).

This is another characteristic of recursive modules: it is always possible to write an equivalent iterative module.
To conclude, here are three important observations on recursion:

- Recursive programming is very intuitive when solving specific problems; it often allows to write clear and straightforward modules which would otherwise be complex if written in iterative form.

- Recursion is very expensive in terms of computing resources. Every recursive call in a module consumes computer memory. Every variable in the module must be temporarily stored in memory prior to proceeding with a recursive call. As recursive calls pile up, memory consumption follows.

- Any problem solved using recursion can always be solved in iterative form, without recursion. In fact recursion is not essential to programming. However it often provides elegant solutions!
Appendix C - Predefined functions

This section provides a detailed description of every predefined function in LARP.

Available predefined functions are:

<table>
<thead>
<tr>
<th>Name</th>
<th>EXP</th>
<th>MAXIMUM</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSOLUTE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARCTANGENT</td>
<td>FLOOR</td>
<td>MINIMUM</td>
<td>SUBSET</td>
</tr>
<tr>
<td>CEILING</td>
<td>FORMAT</td>
<td>PI</td>
<td></td>
</tr>
<tr>
<td>COSINUS</td>
<td>LENGTH</td>
<td>POSITION</td>
<td>TIME</td>
</tr>
<tr>
<td>COUNT</td>
<td>LOG10</td>
<td>RANDOM</td>
<td>TOCHARACTERS</td>
</tr>
<tr>
<td>DATE</td>
<td>LOGE</td>
<td>ROUND</td>
<td>TOSTRING</td>
</tr>
<tr>
<td>ENDOFCOMMENT</td>
<td>LOWERCASE</td>
<td>SINUS</td>
<td>UPPERCASE</td>
</tr>
</tbody>
</table>

Note that:

- In the following function descriptions, brackets ([ and ]) are used to indicate optional syntax elements which may be omitted when invoking the functions.
- The syntax elements presented in italics are descriptive elements not part of the function's syntax.

### ABSOLUTE

<table>
<thead>
<tr>
<th>Name:</th>
<th>ABSOLUTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>ABS</td>
</tr>
<tr>
<td>Return type:</td>
<td>Numerical</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1</td>
</tr>
<tr>
<td>Description:</td>
<td>ABSOLUTE returns the absolute value (i.e. positive) of the number given in parameter.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>ABSOLUTE(numeral)</td>
</tr>
</tbody>
</table>

This function returns the absolute value of the numerical value provided as argument. The type of value returned corresponds to the argument's type (ex: if the argument is a float, the value returned is also a float).

If the given argument is a character string, an attempt is made to convert it into a numerical value.

#### Examples

```
WRITE ABSOLUTE(-3.2)
WRITE ABS(17+4)
WRITE ABSOLUTE("-4.1")
```

Pseudo code C-1: Examples invoking ABSOLUTE
Here are results displayed in the execution console when the above instructions are executed:

```
3.2
21
4.1
```

Figure C-1: Results of invoking ABSOLUTE

---

**ARCTANGENT**

<table>
<thead>
<tr>
<th>Name:</th>
<th>ARCTANGENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>ARCTAN</td>
</tr>
<tr>
<td>Return type:</td>
<td>Numerical (float)</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1</td>
</tr>
<tr>
<td>Description:</td>
<td>ARCTANGENT returns the trigonometric inverse function $\tan^{-1}$ of the angle given as parameter (in radians).</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>ARCTANGENT(numeral)</td>
</tr>
</tbody>
</table>

This function returns the inverse tangent ($\tan^{-1}$) of the numerical value (angle in radians) provided as parameter. The returned value is a float.

If the given argument is a character string, an attempt is made to convert it into a numerical value.

Note that various trigonometric functions can be calculated using predefined functions SINOUS, COSINUS and ARCTANGENT:

\[
\tan(x) = \frac{\sin(x)}{\cos(x)}
\]

\[
\sin^{-1}(x) = \arctan\left(x / \sqrt{1 - (x \times x)}\right)
\]

\[
\cos^{-1}(x) = \arctan\left(\frac{\sqrt{1 - (x \times x)}}{x}\right)
\]

**Examples**

```
WRITE ARCTANGENT(1.2)
WRITE ARCTAN("-1.7")
```

Pseudo code C-2: Examples invoking ARCTANGENT

Here are results displayed in the execution console when the above instructions are executed:

```
0.876058050598193
-1.03907225953609
```

Figure C-2: Results of invoking ARCTANGENT
CEILING

<table>
<thead>
<tr>
<th>Name</th>
<th>CEILING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>CEIL</td>
</tr>
<tr>
<td>Return type:</td>
<td>Numerical (integer)</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1</td>
</tr>
<tr>
<td>Description:</td>
<td>CEILING returns the smallest integer value larger or equal to the given value.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>CEILING(numeral)</td>
</tr>
</tbody>
</table>

This function returns the smallest integer larger or equal to its parameter. If the parameter is already integer, the parameter value is returned unchanged.

If the given argument is a character string, an attempt is made to convert it into a numerical value.

Use the predefined function FLOOR to get the largest integer smaller or equal to the given value.

Examples

```
WRITE CEILING(11.32)
WRITE CEIL("-1.5")
```

Pseudo code C-3: Examples invoking CEILING

Here are results displayed in the execution console when the above instructions are executed:

```
12
-1
```

Figure C-3: Results of invoking CEILING

COSINUS

<table>
<thead>
<tr>
<th>Name</th>
<th>COSINUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>COS</td>
</tr>
<tr>
<td>Return type:</td>
<td>Numerical (float)</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1</td>
</tr>
<tr>
<td>Description:</td>
<td>COSINUS returns the value of trigonometric function Cos applied to the angle given as parameter (in radians).</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>COSINUS(numeral)</td>
</tr>
</tbody>
</table>

This function returns the cosine (Cos) of the numerical value (angle in radians) provided as parameter. The returned value is a float.

If the given argument is a character string, an attempt is made to convert it into a numerical value.
Appendix C - Predefined functions

Examples

```
WRITE COSINUS(1 + 0.2)
WRITE COS("-1.7")
```

Pseudo code C-4: Examples invoking COSINUS

Here are results displayed in the execution console when the above instructions are executed:

```
0.362357754476674
-0.128844494295525
```

Figure C-4: Results of invoking COSINUS

COUNT

<table>
<thead>
<tr>
<th>Name:</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>LENGTH</td>
</tr>
<tr>
<td>Return type:</td>
<td>Numerical (integer)</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1</td>
</tr>
<tr>
<td>Description:</td>
<td>COUNT returns the number of elements defined in a container, or the number of characters in a character string.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>COUNT(container)</td>
</tr>
<tr>
<td></td>
<td>COUNT(character string)</td>
</tr>
</tbody>
</table>

If the parameter is a character string, this function counts individual characters in it.

If the parameter is a container, this function counts the number of defined elements in the container, ignoring indeterminate elements. Counting is not applied recursively to inner dimensions of a multidimensional container; only elements found within the first dimension of the container are counted.

Contrary to the COUNT function, SIZE returns the number of determinate and indeterminate elements in a container.

Examples

```
WRITE COUNT([1, , 3, ])
a = [1, [2, 3], 4]
WRITE COUNT(a)
WRITE LENGTH("Hello world!")
```

Pseudo code C-5: Examples invoking COUNT
Here are results displayed in the execution console when the above instructions are executed:

```
2
3
12
```

Figure C-5: Results of invoking COUNT

## DATE

<table>
<thead>
<tr>
<th>Name:</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>None</td>
</tr>
<tr>
<td>Return type:</td>
<td>Container</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>0</td>
</tr>
<tr>
<td>Description:</td>
<td>DATE returns a container with four values: the current year, the current month (1 to 12), the current day (1 to 31) and the day of week (1 to 7).</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>DATE</td>
</tr>
</tbody>
</table>

This function returns the current date in a container. The four values contained in this container are:

- `DATE[1]` is the current year,
- `DATE[2]` is the current month (1 = January to 12 = December),
- `DATE[3]` is the current day, and
- `DATE[4]` is the day of week (1 = Sunday up to 7 = Saturday).

### Examples

```
WRITE DATE
WRITE "Year =", DATE[1]
```

Pseudo code 12-7: Example invoking DATE

Here are results displayed in the execution console when the above instructions are executed:

```
[2006 3 13 2]
2006
```

Figure C-6: Results of invoking DATE
ENDOFCOMMENT

<table>
<thead>
<tr>
<th>Name:</th>
<th>ENDOFCOMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>EOC</td>
</tr>
<tr>
<td>Return type:</td>
<td>True or false</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1</td>
</tr>
<tr>
<td>Description:</td>
<td>ENDOFCOMMENT returns true if the given input/output channel has reached the end of the associated file or input/output buffer while reading its contents.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>ENDOFCOMMENT(input/output channel)</td>
</tr>
</tbody>
</table>

Predefined function ENDOFCOMMENT applied to an input/output channel allows to determine within a condition whether or not the end of a document read (file or input/output buffer) has been reached.

The function is only applicable to channels opened in READMODE. Any call to this function on an input/output channel opened in WRITEMODE or APPENDMODE causes an interruption in the execution of the algorithm.

ENDOFCOMMENT is not applicable to the keyboard since it is not possible to associate it to an input/output channel.

**Examples**

```plaintext
OPEN "DATA" ON 3 IN READMODE
REPEAT
   READ Value FROM 3
   WRITE Value
UNTIL ENDOFCOMMENT(3) OR Value < 0

IF EOC(3) THEN
   WRITE "All data have been read"
ENDIF
```

Pseudo code C-6: Examples invoking ENDOFCOMMENT

EXP

<table>
<thead>
<tr>
<th>Name:</th>
<th>EXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>None</td>
</tr>
<tr>
<td>Return type:</td>
<td>Numerical (float)</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>0</td>
</tr>
<tr>
<td>Description:</td>
<td>EXP returns the base for natural logarithms.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>EXP</td>
</tr>
</tbody>
</table>

This function returns $e$ (2.71828182845905), the base for the natural logarithm (see also predefined function LOGE).
Examples

```
WRITE EXP
WRITE LOGE(EXP)
```

Pseudo code C-7: Examples invoking EXP

Here are results displayed in the execution console when the above instructions are executed:

```
2.71828182845905
1
```

Figure C-7: Results of invoking EXP

### FLOOR

<table>
<thead>
<tr>
<th>Name:</th>
<th>FLOOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>None</td>
</tr>
<tr>
<td>Return type:</td>
<td>Numerical (integer)</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1</td>
</tr>
<tr>
<td>Description:</td>
<td>FLOOR returns the largest integer value smaller or equal to the given value.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>FLOOR(numeral)</td>
</tr>
</tbody>
</table>

This function returns the largest integer smaller or equal to its parameter. If the parameter is already integer, the parameter value is returned unchanged.

If the given argument is a character string, an attempt is made to convert it into a numerical value.

Use the predefined function **CEILING** to get the smallest integer larger or equal to the given value.

Examples

```
WRITE FLOOR(11.32)
WRITE FLOOR("-1.5")
```

Pseudo code C-8: Examples invoking FLOOR

Here are results displayed in the execution console when the above instructions are executed:

```
11
-2
```

Figure C-8: Results of invoking FLOOR
FORMAT

<table>
<thead>
<tr>
<th>Name:</th>
<th>FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>None</td>
</tr>
<tr>
<td>Return type:</td>
<td>Character string</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1 or more</td>
</tr>
<tr>
<td>Description:</td>
<td>FORMAT returns a character string made from a format string and optional arguments.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>FORMAT(format string, argument sequence)</td>
</tr>
</tbody>
</table>

This function formats a sequence of arguments according to formatting directives provided by a format string. This function may be used to format results prior to displaying them in the execution console. The returned value is a character string containing the formatted arguments.

Function **TOSTRING** can be used to convert a single argument into character string without format specifications.

Format string

Format strings passed to **FORMAT** contain two types of elements: plain characters and format specifiers. Plain characters are copied verbatim to the resulting string. Format specifiers fetch arguments from the argument list and apply formatting to them.

Format specifiers have the following form (brackets [ and ] indicates optional fields and are not part of specifier syntax):

    %[-][width][.precision]type

A format specifier begins with a % character. After the % come the following, in this order:

- An optional left justification indicator, -
- An optional width specifier, width
- An optional precision specifier, .precision
- The conversion type character, type

The following table summarizes the possible values for type:

<table>
<thead>
<tr>
<th>Type</th>
<th>Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Decimal</td>
<td>The argument must be an integer value. The value is converted to a string of decimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has less digits, the resulting string is left-padded with zeros.</td>
</tr>
<tr>
<td>Type</td>
<td>Designation</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>e</td>
<td>Scientific</td>
<td>The argument must be a floating-point value. The value is converted to a string of the form &quot;-d.ddd...E+ddd&quot;. The resulting string starts with a minus sign if the number is negative. One digit always precedes the decimal point. The total number of digits in the resulting string (including the one before the decimal point) is given by the precision specifier in the format string--a default precision of 15 is assumed if no precision specifier is present. The &quot;E&quot; exponent character in the resulting string is always followed by a plus or minus sign and at least three digits.</td>
</tr>
<tr>
<td>f</td>
<td>Fixed</td>
<td>The argument must be a floating-point value. The value is converted to a string of the form &quot;-ddd.ddd...&quot;. The resulting string starts with a minus sign if the number is negative. The number of digits after the decimal point is given by the precision specifier in the format string--a default of 2 decimal digits is assumed if no precision specifier is present.</td>
</tr>
<tr>
<td>g</td>
<td>General</td>
<td>The argument must be a floating-point value. The value is converted to the shortest possible decimal string using fixed or scientific format. The number of significant digits in the resulting string is given by the precision specifier in the format string--a default precision of 15 is assumed if no precision specifier is present. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format.</td>
</tr>
<tr>
<td>n</td>
<td>Number</td>
<td>The argument must be a floating-point value. The value is converted to a string of the form &quot;-d,ddd,ddd.ddd...&quot;. The &quot;n&quot; format corresponds to the &quot;f&quot; format, except that the resulting string contains thousand separators.</td>
</tr>
<tr>
<td>m</td>
<td>Money</td>
<td>The argument must be a floating-point value. The value is converted to a string that represents a currency amount. The conversion is controlled by Windows® configuration and adjustable through the Regional Settings of the Windows® Control Panel. If the format string contains a precision specifier, it overrides the default precision of 2 digits.</td>
</tr>
<tr>
<td>s</td>
<td>String</td>
<td>The argument must be a character string. The string or character is inserted in place of the format specifier. The precision specifier, if present in the format string, specifies the maximum length of the resulting string. If the argument is a string that is longer than this maximum, the string is truncated.</td>
</tr>
<tr>
<td>x</td>
<td>Hexadecimal</td>
<td>The argument must be an integer value. The value is converted to a string of hexadecimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has fewer digits, the resulting string is left-padded with zeros.</td>
</tr>
</tbody>
</table>
Since character % the beginning of a format specifier, it cannot be used directly to insert the % character in the resulting string. The % specifier overcomes this difficulty. Any occurrence of % in the format string is therefore translated into a single % in the resulting string.

Conversion characters may be specified in uppercase as well as in lowercase; both produce the same results. For all floating-point formats, the actual characters used as decimal and thousand separators are obtained from Windows® (See Regional Settings in Windows® Control Panel).

Width and precision specifiers must be provided in integer form (for example, %8.3f). A width specifier sets the minimum field width for a conversion. If the resulting string is shorter than the minimum field width, it is padded with blanks to increase the field width. The default is to right-justify the result by adding blanks in front of the value, but if the format specifier contains a left-justification indicator (a "-" character preceding the width specifier), the result is left-justified by adding blanks after the value.

Examples

Consider the following instructions which use FORMAT to convert various values into character strings:

```
\ Examples with one argument
WRITE '................................'
WRITE FORMAT('1. ***%8d***', 999)
WRITE FORMAT('2. ***%8.7d***', 999)
WRITE FORMAT('3. ***-%8d***', 999)
WRITE FORMAT('4. ***%e***', -999.99)
WRITE FORMAT('5. ***%14.5e***', -999.99)
WRITE FORMAT('6. ***%f***', -999.0)
WRITE FORMAT('7. ***%f***', -999.99)
WRITE FORMAT('8. ***%14.5f***', -999.99)
WRITE FORMAT('9. ***%g***', -999.0)
WRITE FORMAT('10. ***%g***', -999.99)
WRITE FORMAT('11. ***%14.5g***', -999.99)
WRITE FORMAT('12. ***%n***', 99999.99)
WRITE FORMAT('13. ***%m***', 99999.99)
WRITE FORMAT('14. ***%s***', "Hello")
WRITE FORMAT('15. ***%10s***', "Hello")
WRITE FORMAT('16. ***-%10s***', "Hello")
WRITE FORMAT('17. ***%x***', 123)

\ Example with multiple arguments
WRITE FORMAT("\n18. V=%d \n19. %s(V)=%8.4f", V, "Sin", $ Sinus(V-4.5))
```

Here are results displayed in the execution console when the above instructions are executed:

```
.........................
1. *** 999***
2. *** 0000999***
3. ***999***
4. ***-9.99990000000000E+002***
5. *** -9.9999E+002***
6. ***-999***
7. ***-999.99***
8. *** -999.99000***
```
9. ***-999***
10. ***-999.99***
11. ***     -999.99***
12. ***99,999.99***
13. ***$99,999.99***
14. ***Hello***
15. ***     Hello***
16. ***Hello     ***
17. ***7B***
18. V=10
19. Sin(V)= -0.7055

**Figure C-9: Results of invoking FORMAT**

### LOG10

<table>
<thead>
<tr>
<th>Name:</th>
<th>LOG10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>None</td>
</tr>
<tr>
<td>Return type:</td>
<td>Numerical (float)</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1</td>
</tr>
<tr>
<td>Description:</td>
<td>LOG10 returns base 10 logarithm of the value given in parameter.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>LOG10(numeral)</td>
</tr>
</tbody>
</table>

This function returns base 10 logarithm (i.e. $\log_{10}$) of the value given in parameter.

If the given argument is a character string, an attempt is made to convert it into a numerical value.

**Examples**

```
WRITE LOG10(100)
WRITE LOG10("5.5")
```

**Pseudo code C-10: Examples invoking LOG10**

Here are results displayed in the execution console when the above instructions are executed:

```
2
0.740362689494244
```

**Figure C-10: Results of invoking LOG10**
LOGE

<table>
<thead>
<tr>
<th>Name:</th>
<th>LOGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>None</td>
</tr>
<tr>
<td>Return type:</td>
<td>Numerical (float)</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1</td>
</tr>
<tr>
<td>Description:</td>
<td>LOGE returns the natural logarithm of the value given in parameter.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>LOGE(numeral)</td>
</tr>
</tbody>
</table>

This function returns the natural logarithm (i.e. base $e$) of the value given in parameter.

If the given argument is a character string, an attempt is made to convert it into a numerical value.

The predefined function EXP returns the natural base constant ($e$) used by LOGE.

**Examples**

```
WRITE LOGE(100)
WRITE LOGE(EXP)
```

Pseudo code C-11: Examples invoking LOGE

Here are results displayed in the execution console when the above instructions are executed:

```
4.60517018598809
1
```

Figure C-11: Results of invoking LOGE

LOWERCASE

<table>
<thead>
<tr>
<th>Name:</th>
<th>LOWERCASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>None</td>
</tr>
<tr>
<td>Return type:</td>
<td>Character string</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1</td>
</tr>
<tr>
<td>Description:</td>
<td>LOWERCASE returns the given character string with all uppercase letters converted to lowercase letters.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>LOWERCASE(character string)</td>
</tr>
</tbody>
</table>

This function returns the character string given in parameter with all its uppercase letters converted into corresponding lowercase letters. Other characters in the string remain unchanged.

The predefined function UPPERCASE transforms lowercase letters into uppercase letters.
Examples

```plaintext
WRITE LOWERCASE("Hello World!")
WRITE LOWERCASE("Joe 99")
```

Pseudo code C-12: Examples invoking LOWERCASE

Here are results displayed in the execution console when the above instructions are executed:

```
hello world!
joe 99
```

Figure C-12: Results of invoking LOWERCASE

MAXIMUM

<table>
<thead>
<tr>
<th>Name:</th>
<th>MAXIMUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>MAX</td>
</tr>
<tr>
<td>Return type:</td>
<td>Sequence of numerical values and/or containers</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1 or more</td>
</tr>
<tr>
<td>Description:</td>
<td>MAXIMUM returns the largest value among those provided as parameters.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>MAXIMUM(argument, argument, argument, ...)</td>
</tr>
</tbody>
</table>

This function accepts a variable number of parameters and returns the largest value among these parameters. If a parameter is a container, the function walks recursively through the container to identify the largest numerical value within.

If a parameter is a character string, an attempt is made to convert it into a numerical value.

The predefined function MINIMUM returns the smallest value among those provided as parameters.

Examples

```plaintext
WRITE MAXIMUM(10, 100, 20)
WRITE MAX(10, [100, [20, 200], "50"], 150))
```

Pseudo code C-13: Examples invoking MAXIMUM

Here are results displayed in the execution console when the above instructions are executed:

```
100
200
```

Figure C-13: Results of invoking MAXIMUM
MINIMUM

<table>
<thead>
<tr>
<th>Name:</th>
<th>MINIMUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>MIN</td>
</tr>
<tr>
<td>Return type:</td>
<td>Sequence of numerical values and/or containers</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1 or more</td>
</tr>
<tr>
<td>Description:</td>
<td>MINIMUM returns the smallest value among those provided as parameters.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>MINIMUM(argument, argument, argument, ...)</td>
</tr>
</tbody>
</table>

This function accepts a variable number of parameters and returns the smallest value among these parameters. If a parameter is a container, the function walks recursively through the container to identify the smallest numerical value within.

If a parameter is a character string, an attempt is made to convert it into a numerical value.

The predefined function **MAXIMUM** returns the largest value among those provided as parameters.

**Examples**

```
WRITE MINIMUM(10, 100, 20)
WRITE MIN(40, [100, [20, "12"], 50], 150))
```

Pseudo code C-14: Examples invoking MINIMUM

Here are results displayed in the execution console when the above instructions are executed:

```
10
12
```

Figure C-14: Results of invoking MINIMUM

**PI**

<table>
<thead>
<tr>
<th>Name:</th>
<th>PI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>None</td>
</tr>
<tr>
<td>Return type:</td>
<td>Numerical (float)</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>0</td>
</tr>
<tr>
<td>Description:</td>
<td>PI returns the value of the trigonometric constant π.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>PI</td>
</tr>
</tbody>
</table>

This function returns the value of the trigonometric constant π, in radians (3.14159265358979).
Examples

```
WRITE PI
WRITE SINUS(PI) + COSINUS(PI)
```

Pseudo code C-15: Examples invoking PI

Here are results displayed in the execution console when the above instructions are executed:

```
3.14159265358979
-1
```

Figure C-15: Results of invoking PI

### POSITION

<table>
<thead>
<tr>
<th>Name:</th>
<th>POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>POS</td>
</tr>
<tr>
<td>Return type:</td>
<td>Numerical (integer)</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>2</td>
</tr>
<tr>
<td>Description:</td>
<td>POSITION returns the position of a character string in another character string, or the position of an element in a container.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>POSITION(character string, character string) POSITION(element, container)</td>
</tr>
</tbody>
</table>

This function searches for the position (i.e. the index) of an element in a character string or in a container. The type of search depends on the second parameter:

- If the second parameter is a character string, the first parameter must also be a character string or a single character (otherwise an attempt is made to convert the first parameter into a character string). The function returns the position of the first occurrence of the first string into the second. Position 0 is returned when the first string is not found within the second. The returned positions are based on 1 as starting index (i.e. index of the first character in a string or the first element in a container).

- If the second parameter is a container, the first parameter may be of any type. POSITION searches for the first occurrence of the first parameter within the container. If the specified element is not found in the container, position 0 is returned.

Note that searching a container is not recursive. Only the first dimension of a multidimensional container is searched.

Examples

```
WRITE POSITION("the", "Hello the world!")
WRITE POSITION(3, [1, 2, [3, 4], 3, 5])
WRITE POSITION(4, [1, 2, [3, 4], 3, 5])
WRITE POS(([3, 4], [1, 2, [3, 4], 3, 5]))
```

Pseudo code C-16: Examples invoking POSITION
Here are results displayed in the execution console when the above instructions are executed:

```
7
4
0
3
```

Figure C-16: Results of invoking POSITION

**RANDOM**

<table>
<thead>
<tr>
<th>Name:</th>
<th>RANDOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>RAND</td>
</tr>
<tr>
<td>Return type:</td>
<td>Numeric</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>0, 1 or 2</td>
</tr>
<tr>
<td>Description:</td>
<td>RANDOM returns a float value or integer value selected randomly (several versions of the function are available).</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>RANDOM, RANDOM(numeral), RANDOM(numeral, numeral)</td>
</tr>
</tbody>
</table>

This function returns a numerical value selected randomly. The returned value depends on provided parameters:

- **No parameter**: RANDOM returns a float value \( x \) in range \( 0 < x < 1 \).

- **One parameter**: RANDOM(\( p \)) returns a value \( x \) in range \( 0 < x < p \). The type of \( x \) corresponds to the type of \( p \).

- **Two parameters**: RANDOM(\( p, q \)) returns a value \( x \) in range \( p < x < q \). The type of \( x \) corresponds to types of \( p \) and \( q \) (if \( p \) or \( q \) is a float, the returned value \( x \) is a float; if \( p \) and \( q \) are integer, the returned value is integer).

If a given argument is a character string, an attempt is made to convert it into a numerical value.

**Examples**

```
FOR i = 1 UNTIL 5 DO
  WRITE "RAND=", RANDOM
  WRITE "RAND(4)=", RANDOM(4)
  WRITE "RAND(1,5.0)=", RAND(1, 5.0)
ENDFOR
```

Pseudo code C-17: Examples invoking RANDOM
Here are results displayed in the execution console when the above instructions are executed:

```
RAND= 0.232930421130732
RAND(4)= 3
RAND(1,5.0)= 3.8551177335903
RAND= 0.438837686553597
RAND(4)= 3
RAND(1,5.0)= 1.61782418005168
RAND= 0.0393810363020748
RAND(4)= 0
RAND(1,5.0)= 1.03878780175
RAND= 0.378834913019091
RAND(4)= 1
RAND(1,5.0)= 2.45533445477486
RAND= 0.0822982566896826
RAND(4)= 1
RAND(1,5.0)= 3.0188071122393
```

Figure C-17: Results of invoking RANDOM

### ROUND

<table>
<thead>
<tr>
<th>Name:</th>
<th>ROUND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>RND</td>
</tr>
<tr>
<td>Return type:</td>
<td>Numerical (integer)</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1</td>
</tr>
<tr>
<td>Description:</td>
<td>ROUND returns its parameter rounded to the nearest integer value.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>ROUND(numeral)</td>
</tr>
</tbody>
</table>

This function returns its parameter rounded to the nearest integer value. If the given value is already integer, it is returned as is. On the other hand if the parameter is float, its closest integer value is returned. If the parameter is exactly midway between two integers (i.e. \( x + 0.5 \)), it is rounded up to the nearest larger integer in absolute magnitude (for example, 7.5 is rounded up to 8, while \(-7.5\) is rounded down to \(-8\)).

If the given argument is a character string, an attempt is made to convert it into a numerical value.

**Examples**

```
WRITE ROUND(11.32)
WRITE RND("-1.5")
```

Pseudo code C-18: Examples invoking ROUND
Here are results displayed in the execution console when the above instructions are executed:

```
11
-2
```

Figure C-18: Results of invoking ROUND

**SINUS**

<table>
<thead>
<tr>
<th>Name:</th>
<th>SINUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>SIN</td>
</tr>
<tr>
<td>Return type:</td>
<td>Numeric (float)</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1</td>
</tr>
<tr>
<td>Description:</td>
<td>SINUS returns the value of trigonometric function $\sin$ applied to the angle given as parameter (in radians).</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>SINUS($\text{numeric}$)</td>
</tr>
</tbody>
</table>

This function returns the $\sin$ ($\text{Sin}$) of the numerical value (angle in radians) provided as parameter. The returned value is a float.

If the given argument is a character string, an attempt is made to convert it into a numerical value.

**Examples**

```
WRITE SINUS(1.2)
WRITE SIN (1 - 2.7)
```

Pseudo code C-19: Examples invoking SINUS

Here are results displayed in the execution console when the above instructions are executed:

```
0.932039085967226
-0.991664810452469
```

Figure C-19: Results of invoking SINUS

**SIZE**

<table>
<thead>
<tr>
<th>Name:</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>None</td>
</tr>
<tr>
<td>Return type:</td>
<td>Numerical (integer)</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1</td>
</tr>
<tr>
<td>Description:</td>
<td>SIZE returns the number of determinate and indeterminate elements in a container.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>SIZE($\text{container}$)</td>
</tr>
</tbody>
</table>

This function counts the total number of positions (both determinate and indeterminate elements) in the container provided as parameter.
**SIZE** is not applied recursively to the inner dimensions of a multidimensional container, it only counts elements found within the first dimension of the container.

Contrary to the **SIZE** function, the **COUNT** function only counts determinate elements in a container.

### Examples

```plaintext
WRITE SIZE([1, , 3, ])
a = [1, [2, 3], 4]
WRITE SIZE(a)
```

Here are results displayed in the execution console when the above instructions are executed:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

**Figure C-20: Results of invoking SIZE**

### SQUAREROOT

<table>
<thead>
<tr>
<th>Name:</th>
<th>SQUAREROOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>SQRT</td>
</tr>
<tr>
<td>Return type:</td>
<td>Numerical (float)</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1</td>
</tr>
<tr>
<td>Description:</td>
<td>SQUAREROOT returns the square root of the given parameter value.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>SQUAREROOT(numeral)</td>
</tr>
</tbody>
</table>

This function returns the square root of the value given in parameter.

If the given argument is a character string, an attempt is made to convert it into a numerical value.

### Examples

```plaintext
WRITE SQUAREROOT(100)
WRITE SQRT("5.5")
```

Here are results displayed in the execution console when the above instructions are executed:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2.34520787991171</td>
</tr>
</tbody>
</table>

**Figure C-21: Results of invoking SQUAREROOT**
SUBSET

<table>
<thead>
<tr>
<th>Name:</th>
<th>SUBSET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>None</td>
</tr>
<tr>
<td>Return type:</td>
<td>Character string or container</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>3</td>
</tr>
<tr>
<td>Description:</td>
<td>SUBSET returns a subset of the first parameter.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>SUBSET(character string, start, length) SUBSET(container, start, length)</td>
</tr>
</tbody>
</table>

This function returns a partial copy of the first parameter, which must be a character string or a container. The second parameter (start) indicates the index where to begin extracting characters or elements from the first parameter. The third parameter (length) indicates the number of characters or elements to copy. The type of the returned value depends on the type of the first parameter:

- If the first parameter is a character string, the function returns a character string composed of length characters from the first parameter, starting with the character located at start index of the given string.

- If the first parameter is a container, the function returns a container composed of length elements from the first parameter, starting with the element located at start index of the given container.

If the second or the third argument is a character string, an attempt is made to convert it into a numerical value.

Examples

```
WRITE SUBSET("Hello world!", 1, 5)
WRITE SUBSET("Hello world!", 7, 5)
WRITE SUBSET([10, 20, 30, 40, 50], 2, "3")
```

Pseudo code C-22: Examples invoking SUBSET

Here are results displayed in the execution console when the above instructions are executed:

```
Hello
world
[20 30 40]
```

Figure C-22: Results of invoking SUBSET
TIME

<table>
<thead>
<tr>
<th>Name:</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>None</td>
</tr>
<tr>
<td>Return type:</td>
<td>Container</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>0</td>
</tr>
<tr>
<td>Description:</td>
<td>TIME returns a container with four values giving the current time: hours, minutes, seconds and milliseconds.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>TIME</td>
</tr>
</tbody>
</table>

This function returns the current time in a container. The four values returned in this container are:

- \( \text{TIME}[1] \) are hours since the start of day (0 to 23),
- \( \text{TIME}[2] \) are minutes since the start of current hour (0 to 59),
- \( \text{TIME}[3] \) are seconds since the start of current minute (0 to 59), and
- \( \text{TIME}[4] \) are milliseconds since the start of current second (0 to 999).

**Examples**

```
WRITE TIME
WRITE "Hour =", \text{TIME}[1]
```

Pseudo code C-23: Examples invoking TIME

Here are results displayed in the execution console when the above instructions are executed:

```
[15 23 54 921]
15
```

Figure C-23: Results of invoking TIME

**TOCHARACTERS**

<table>
<thead>
<tr>
<th>Name:</th>
<th>TOCHARACTERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>TOCHARS</td>
</tr>
<tr>
<td>Return type:</td>
<td>Container</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1</td>
</tr>
<tr>
<td>Description:</td>
<td>TOCHARACTERS converts the given parameter to a container with individual characters as elements.</td>
</tr>
</tbody>
</table>
| Invoke format(s): | TOCHARACTERS \((\text{character string})\)  
|                 | TOCHARACTERS \((\text{numeral})\)  
|                 | TOCHARACTERS \((\text{container})\) |

This function returns a container holding each individual character found in the given parameter. The conversion to characters is performed according to parameter type.
• If the parameter is a character string, the elements of the returned container are individual characters found in the given character string.

• If the parameter is a numerical value, it is first converted into a character string before extracting the individual characters.

• If the parameter is a container, TOCHARACTERS converts every element of that container into individual characters, regrouping all resulting characters within a single container. If the parameter is a multidimensional container, the extraction of characters is applied recursively.

Examples

| SEPARATOR "", |
| WRITE TOCHARACTERS("Bravo!") |
| WRITE TOCHARACTERS(132.4) |
| WRITE TOCHARS([1, ["Bye", 3], 4]) |

Pseudo code C-24: Examples invoking TOCHARACTERS

Here are results displayed in the execution console when the above instructions are executed:

[ B, r, a, v, o, ! ]
[ 1, 3, 2, . . 4 ]
[ 1, B, y, e, 3, 4 ]

Figure C-24: Results of invoking TOCHARACTERS

Note that every element of the resulting containers is a character. So container [1, B, y, e, 3, 4] contains elements '1', 'B', 'y', 'e', '3' and '4'.

TOSTRING

| Name: | TOSTRING |
| Synonyms: | TOSTR |
| Return type: | Character string |
| Number of arguments: | 1 |
| Description: | TOSTRING converts the given parameter into a character string. |
| Invoke format(s): | TOSTRING(numeral) TOSTRING(container) |

This function returns a character string containing the parameter as it appears when written to the execution console. The parameter can be a numerical value, a container or even another character string (in which case no conversion occurs).

The predefined function FORMAT can be used to convert one or multiple arguments according to given format directives.
Examples

WRITE TOSTRING("Bravo!")
WRITE TOSTRING(132.4)
WRITE TOSTR([1, ["Bye", 3], 4])

Pseudo code C-25: Examples invoking TOSTRING

Here are results displayed in the execution console when the above instructions are executed:

Bravo!
132.4
1Bye34

Figure C-25: Results of invoking TOSTRING

In the above example instruction TOSTRING(132.4) returns the string "132.4".

UPPERCASE

<table>
<thead>
<tr>
<th>Name:</th>
<th>UPPERCASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonyms:</td>
<td>None</td>
</tr>
<tr>
<td>Return type:</td>
<td>Character string</td>
</tr>
<tr>
<td>Number of arguments:</td>
<td>1</td>
</tr>
<tr>
<td>Description:</td>
<td>UPPERCASE returns the given character string with all lowercase letters converted to uppercase letters.</td>
</tr>
<tr>
<td>Invoke format(s):</td>
<td>UPPERCASE(character string)</td>
</tr>
</tbody>
</table>

This function returns the character string given in parameter with all its lowercase letters converted into corresponding uppercase letters. All other characters in the string remain unchanged.

The predefined function LOWERCASE transforms uppercase letters into lowercase letters.

Examples

WRITE UPPERCASE("Hello world!")
WRITE UPPERCASE("Joe 99")

Pseudo code C-26: Examples invoking UPPERCASE

Here are results displayed in the execution console when the above instructions are executed:

HELLO WORLD!
JOE 99

Figure C-26: Results of invoking UPPERCASE
Appendix D - *LARP*'s syntax elements

This section presents *LARP*'s pseudo code and flowchart syntax. In the following tables, reserved words are presented in bold and uppercase letters (for example, **START**), elements to expand in italics (for example, *instructions sequence*) and optional elements within brackets (for example, *[parameter lists]*)).

For more information on a specific syntax element, consult the corresponding section.

### D.1 Modules

<table>
<thead>
<tr>
<th>Main module</th>
<th>Auxiliary module</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>START</strong></td>
<td><strong>ENTER</strong></td>
</tr>
<tr>
<td><em>instructions sequence</em></td>
<td><em>parameter list</em></td>
</tr>
<tr>
<td><strong>END</strong></td>
<td><em>instruction sequence</em></td>
</tr>
<tr>
<td></td>
<td><strong>RETURN</strong></td>
</tr>
<tr>
<td></td>
<td><em>return value</em></td>
</tr>
</tbody>
</table>

![Flowchart diagrams for modules](image)

where

- *instruction sequence* is a sequence of *LARP* instructions other than module definitions (a module definition cannot be embedded within another module).
- *parameter list*, optional, consists of one or several variables separated by commas.
- *return value*, optional, is an expression returning an integer value, a float value, a character string or a container.

### D.2 Conditional structures

<table>
<thead>
<tr>
<th>IF structure</th>
<th>IF-ELSE structure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IF</strong> condition <strong>THEN</strong></td>
<td><strong>IF</strong> condition <strong>THEN</strong></td>
</tr>
<tr>
<td><em>instructions sequence</em></td>
<td>*instructions sequence #1</td>
</tr>
<tr>
<td><strong>ENDIF</strong></td>
<td><strong>ELSE</strong></td>
</tr>
<tr>
<td></td>
<td>*instructions sequence #2</td>
</tr>
<tr>
<td></td>
<td><strong>ENDIF</strong></td>
</tr>
</tbody>
</table>

![Flowchart diagrams for conditional structures](image)
### IF structure

- **condition** is a boolean expression made of relational operators, logical operators and/or type validations.

- **instructions sequence #** are sequences of LARP instructions other than module definitions (a module definition cannot be embedded within another module).

### IF-ELSE structure

```
IF condition #1 THEN
  instructions sequence #1
ELSE IF condition #2 THEN
  instructions sequence #2
ELSE IF condition #3 THEN
  instructions sequence #3
  ...
ELSE IF condition #n THEN
  instructions sequence #n
[ELSE
  instructions sequence #n+1 ]
ENDIF
```

### IF-ELSE-IF structure

```
IF condition #1 THEN
  instructions sequence #1
ELSE IF condition #2 THEN
  instructions sequence #2
ELSE IF condition #3 THEN
  instructions sequence #3
  ...
ELSE IF condition #n THEN
  instructions sequence #n
[ELSE
  instructions sequence #n+1 ]
ENDIF
```

### SELECT structure

```
SELECT expression
  list of constants #1 [:] instructions sequence #1
  list of constants #2 [:] instructions sequence #2
  list of constants #3 [:] instructions sequence #3
  ...
  list of constants #n [:] instructions sequence #n
  [ELSE
    instructions sequence #n+1 ]
ENDSELECT
```

---

Appendix D - LARP’s syntax

LARP Users Guide

190

Copyright © 2004-2008 Marco Lavoie
SELECT structure

where

- **condition #** are boolean expressions made of relational operators, logical operators and/or type validations.
- **expression** is an expression returning an integer value, a float value, a character string or a container.
- **instructions sequence #** are sequences of LARP instructions other than module definitions (a module definition cannot be embedded within another module).
- **list of constants** are lists of integer values, float values, character strings and/or containers separated by commas. An optional colon (:) may be inserted between a list of constants and its instructions sequence.
- The last **ELSE** section is optional and may thus be omitted when not required.

### D.3 Iterative structures

<table>
<thead>
<tr>
<th>WHILE structure</th>
<th>REPEAT-UNTIL structure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WHILE</strong> <strong>condition</strong> <strong>DO</strong></td>
<td><strong>REPEAT</strong></td>
</tr>
<tr>
<td><strong>instructions sequence</strong></td>
<td><strong>instructions sequence</strong></td>
</tr>
<tr>
<td><strong>ENDWHILE</strong></td>
<td><strong>UNTIL</strong> <strong>condition</strong></td>
</tr>
</tbody>
</table>

![Diagram of WHILE and REPEAT-UNTIL structures]
where

- **condition** is a boolean expression made of relational operators, logical operators and/or type validations.
- **instructions sequence** is a sequence of LARP instructions other than module definitions (a module definition cannot be embedded within another module).

FOR structure

```plaintext
FOR variable = initial value TO final value [ STEP step value ] DO
    instructions sequence
ENDFOR
```

where

- **variable** is a variable name.
- **initial value, final value** and **step value** (optional) are expressions returning an integer value.
- **instructions sequence** is a sequence of LARP instructions other than module definitions (a module definition cannot be embedded within another module).

### D.4 Files and input/output buffers

```plaintext
OPEN input/output buffers

OPEN [ BUFFER ] buffer name ON channel number IN access mode
```

where

- **buffer name** is a buffer name.
- **channel number** is a channel number.
- **access mode** is an access mode.
where

- **buffer name** is the name of the input/output buffer to open (the buffer must already be created and listed in the document browser).
- **channel number** is an expression returning an integer value from 1 to 256.
- **access mode** is one of the following reserved words: READMODE, WRITEMODE or APPENDMODE.
- Reserved word BUFFER is optional.

### OPEN a file

**OPEN FILE**  
`file name ON channel number IN access mode`

where

- **file name** is the name of the file to open.
- **channel number** is an expression returning an integer value from 1 to 256.
- **access mode** is one of the following reserved words: READMODE, WRITEMODE or APPENDMODE.

### CLOSE input/output buffers

**CLOSE**  
`list of channel numbers`

where

- **list of channel numbers** is a list of one or more expressions separated with commas, each returning an integer value from 1 to 256.
D.5 Inputs and outputs

Instructions for reading and writing

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>READ</strong> list of variables [ FROM channel number ]</td>
<td>Read list of variables from a specified channel.</td>
</tr>
<tr>
<td><strong>WRITE</strong> list of expressions [ IN channel number ]</td>
<td>Write list of expressions to a specified channel.</td>
</tr>
<tr>
<td><strong>QUERY</strong> prompt [ , list of variables ]</td>
<td>Query with a prompt and list of variables.</td>
</tr>
<tr>
<td><strong>SEPARATOR</strong> character</td>
<td>Separate output with a character.</td>
</tr>
</tbody>
</table>

where

- **list of variables** is made of one or more variables separated by commas.
- **channel number**, optional, is an expression returning an integer value from 1 to 256.
- **list of expressions** is a list of one or more expressions, each returning an integer value, a float value, a character string or a container.
- **prompt** is an expression returning a character string.
- **character** is an expression returning a character string containing a single character.
- if no **channel number** is provided, inputs and outputs are directed to the execution console.
Appendix E - Warnings and errors

Messages displayed in LARP are classified in two categories:

1. **Messages associated with the development environment**: messages resulting from erroneous actions or commands on behalf of users.

2. **Messages associated with algorithms**: messages resulting from syntax errors in modules and/or illegal operations performed by algorithms during their execution.

### E.1 Messages related to the development environment

The following messages are usually displayed in a popup window whenever LARP’s development environment refuses operations requested by the user or whenever more information is required to perform requested operations.

The messages are presented in alphabetical order, each with a brief description.

**A backup copy of a project has been generated following a crash. Do you want to reload that project?**

At regular time intervals as well as when an algorithm is executed, a backup copy of the current project file is automatically stored in a temporary file on the computer. This backup copy is automatically destroyed when the edited project is saved in its own file or when the execution of the algorithm ends.

If a fatal software error occurs and causes LARP to crash, the backup copy of the edited project is not destroyed. Upon the next startup of LARP, the backup copy will be detected and optionally recovered. In other words if LARP crashes before you had the opportunity to save modifications to your project, you can recover your work when LARP is restarted.

Consult the section on security backups for more information.

**Warning**: if you refuse to recover a backup project file upon starting up LARP, the backup copy will be irremediably destroyed.

**A document named «document name» already exists in project.**

Your attempt to create a new document (module or input/output buffer) failed because the project already has a document with the specified name. Choose a different name for the new document.

Note that LARP may have shortened the provided document name by eliminating illegal characters.

**A project’s main module cannot be deleted.**

Your attempt to destroy a project’s main module failed. All LARP projects must have one and only one main module since it is the starting point of execution of algorithms. If you want to change the main module of a project, you must edit the contents of the existing main module.
Basic passwords error.

The username provided to activate super user mode does not correspond to the super user key currently connected to the computer. Make sure the super user key is properly connected and your username corresponds to the connected key.

Close SparKey error.

An error occurred during access to the super user key currently connected to the computer. Make sure the super user key is properly connected and the specified username corresponds to the key.

Console must first be closed.

During the execution of an algorithm, some commands commonly available in the development environment are temporarily deactivated until the execution ends and the execution console is closed.

You must first close the execution console (by completing or interrupting the algorithm’s execution) before requesting the desired operation.

Do you really want to erase all above data associated with the project?

As super user you may erase statistics as well as usernames attached to project files. Once erased, this information is not recoverable.

Help files are not accessible.

It seems some files related to LARP’s online help are missing. Make sure LARP is properly installed and help files are accessible. If it is not the case, reinstall the most recent version of LARP. If reinstalling LARP does not resolve the situation, contact technical support.

«Identifier» is not a valid I/O buffer name.

The name you wish to give to an input/output buffer is invalid. Buffer names must be made of letters, digits and/or the underline character (_). Furthermore the first character of buffer names may not be a digit and spaces are not allowed within the names.

For more information, consult the section on input/output buffers.

«Identifier» is not a valid module name.

The name you wish to give to a module is invalid. Module names must be made of letters, digits and/or the underline character (_). Furthermore the first character of module names may not be a digit and spaces are not allowed within the names.

For more information, consult the section on modules.

Invalid dongle configuration.

The configuration of the super user key currently connected to the computer is erroneous. Such error is usually caused by defective keys or keys not issued by LARP’s publisher. Contact technical support to have your super user key replaced.
Invalid key format.

The username provided to activate super user mode is incompatible with the super user key currently connected to the computer. Make sure your username corresponds to the key connected to the computer.

Invalid username.

The username provided upon LARP’s start-up or when changing user is invalid. Read the section on super user mode for more information.

No help is available for this type of errors.

LARP’s online help cannot provide further information on this error.

Only one file can be dragged in LARP.

LARP’s development environment accepts dropped files (i.e. by drag and drop), but only one file at a time.

Open SparKey error.

An error occurred upon detection and/or validation of a super user key currently connected to the computer. Make sure the super user key is properly connected, the specified username corresponds to the key and the key is not defective. If everything seems in order, contact technical support to have your super user key replaced.

Read data error.

An error occurred during access to the super user key currently connected to the computer. Make sure the super user key is properly connected and the key is not defective. If everything seems in order, contact technical support to have your super user key replaced.

Read ID error.

An error occurred upon detection and/or validation of a super user key currently connected to the computer. Make sure the super user key is properly connected, the specified username corresponds to the key and the key is not defective. If everything seems in order, contact technical support to have your super user key replaced.

Read length or Start address error.

An error occurred during access to the super user key currently connected to the computer. Make sure the super user key is properly connected, the specified username corresponds to the key and the key is not defective. If everything seems in order, contact technical support to have your super user key replaced.

SparKey not found.

An error occurred upon detection and/or validation of a super user key currently connected to the computer. Make sure the super user key is properly connected, the specified username corresponds to the key and the key is not defective. If everything seems in order, contact technical support to have your super user key replaced.
The project has no I/O buffer to print.

Printing modules is restricted to super user mode. However printing input/output buffers is allowed for every user. You tried to print a project with no input/output buffer while super user mode was deactivated.

Unknown error code.

An unknown error (not anticipated by the developers of LARP) occurred. Please contact technical support for help.

You must close the current project prior to changing user.

During the execution of an algorithm, some commands usually available in the development environment are temporarily deactivated until the execution ends and the execution console is closed.

You must first close the execution console (by completing or interrupting the execution of algorithm) before changing the active username.

E.2 Messages related to the execution of algorithms

Messages described in this section are displayed by LARP when an error occurs during the compilation and execution of algorithms. These messages are displayed in the message panel and, when problem occur during the execution of an algorithm, in a popup window.

Warning messages usually indicate potential errors in an algorithm, but these errors are not fatal and LARP is able to continue executing the algorithm.

Error messages are usually displayed:

- during the compilation of an algorithm: compilation usually continues in order to validate the syntax of the whole algorithm, but its execution is not possible.

- during the execution of an algorithm: most errors occurring during the execution of an algorithm are fatal and interrupt the execution.

Some messages may identify errors found in the LARP software itself. LARP’s technical support must be informed of such bugs so they may be fixed in the next version of the software.

E1001 An auxiliary module must start with ENTER command

You have an auxiliary module starting with an instruction other than ENTER (which may have optional parameters).

If the module in error starts with the instruction START, remember that only the main module may have START as first instruction.

Consult the section on modules for more information.
E1002  The main module must start with START command

   The project’s main module starts with an instruction other than START.

   If the module in error starts with the instruction ENTER, remember that only auxiliary modules may start with ENTER.

   Consult the section on modules for more information.

E1003  Invalid module header

   The instruction starting the erroneous module does not correspond to the format imposed by LARP.

   Consult the section on modules for more information.

E1004  An auxiliary module must end with RETURN command

   You have an auxiliary module ending with an instruction other than RETURN (with optional return value).

   If the module in error ends with an END instruction, remember that only the main module may end with END.

   Consult the section on modules for more information.

E1005  The main module must end with END command

   The project’s main module ends with an instruction other than END.

   If the module in error ends with the instruction RETURN (with optional return value), remember that only auxiliary modules may end with RETURN.

   Consult the section on modules for more information.

E1006  A module must end with RETURN or END

   The module in error does not end with an instruction appropriate to the type of module.

   Consult the section on modules for more information.

E1007  Variables must be separated by commas

   Some LARP instructions require a list of variables (for example, READ, QUERY and ENTER). When multiple variables are listed, they must be separated by commas (,) in the list.

   For instance, the instruction READ A B C is invalid; it should be written READ A, B, C.

E1008  The identifier «variable name» is a module name

   You try to use the name of a module as variable (for instance in an assignment). The name in question corresponds to one of the modules defined in the project, but this module is not invoked appropriately.

   Consult the section on modules for more information.
E1009  The keyword «reserved word» cannot be used in this context

You try to use a reserved word of LARP in a context other than those to which applies the word in question.

The most common mistake is attempting to use a reserved word as variable name.

E1010  I do not understand this statement

You do not respect the syntax of LARP and it is not possible to provide further information on the detected error.

See the online help corresponding to the instruction which causes the error and make sure to use proper syntax as imposed in LARP pseudo codes and flowcharts.

E1011  Template fields must be replaced by valid pseudo code

You have dropped a template from the template panel into your pseudo code, but you forgot to replace template fields with valid pseudo code. Template fields are identified within curly brackets ( { and } ) and must be replaced (along with their brackets) with valid pseudo code.

For example, when dropping a READ template into a pseudo code module, the line READ {variable_list} appears. You must replace the field {variable_list} with one or more variables.

E1012  The condition in corresponding conditional or iterative structure is invalid

There is an error in the formulation of a condition within a conditional structure or an iterative structure.

Consult the corresponding sections for more information on formulating conditions.

E1999  Unknown error; contact technical support

An unexpected error occurred. This error was not anticipated by LARP nor its developers.

Please contact LARP’s technical support from help.

E2001  Not enough arguments provided in module call

A call to an auxiliary module does not provide enough arguments. The number of arguments provided in a module call must correspond to the number of parameters listed in module’s ENTER instruction.

Consult the section on auxiliary module parameters for more information.

E2002  Too many arguments provided in module call

A call to an auxiliary module provides more arguments than required. The number of arguments provided in a module call must correspond to the number of parameters listed in module’s ENTER instruction.

Consult the section on auxiliary module parameters for more information.

E2003  Invalid value type

A numerical value, an expression or a variable containing a numerical value of inappropriate type is exploited in an instruction.
The most frequent reason for this error is using a variable containing a value which is inappropriate for the LARP instruction in which it is used (for example, providing an inappropriate value as argument to a predefined function).

**E2004 Invalid index value**

The index provided to reference an element of a container is invalid. Container indexes must be integer values.

For instance, the instruction `WRITE a[1.2]` is erroneous since the specified index is not integer.

For more information, consult the section describing access to container elements.

**E2005 Index value out of bounds**

The index provided to reference an element of a container is inferior to the allowed minimum index value or superior to the number of elements in the container.

For more information, consult the section describing access to container elements.

**E2006 The reference is not a character string**

The instruction requires a character string but the provided value is of another type (for example a number or a container).

**E2007 The reference is not a container**

The instruction requires a container but the provided value is of another type (for example a number or a character string).

**E2008 Unable to set container dimensions**

The container cannot be dimensioned according to the instruction’s requirements. This error is generally caused by excessive use of the computer’s memory. To overcome this problem, use smaller containers.

For configuring the maximum size of containers in LARP algorithms, consult the section describing configuration of the execution console.

**E2009 The container is empty**

The specified container contains no element.

**E2010 Invalid file or I/O buffer name**

The file name provided to an OPEN instruction is invalid.

When opening a file, the provided file name is invalid if it does not respect Windows® rules for naming files or if the file to be opened for reading does not exist.

When opening an input/output buffer, the provided buffer name is invalid if it does not correspond to a buffer defined in the project.

Consult the section on Files and input/output buffers for more information.
E2011  Invalid channel number

The channel number provided to an OPEN instruction is invalid. Input/output channels available in LARP are numbered from 1 to 256.

Consult the section on Input/output channels for more information.

E2012  Channel already allocated to another file or I/O buffer

The channel number provided to an OPEN instruction is already associated with another opened file or input/output buffer. It is not allowed to associate a single input/output channel to multiple documents opened simultaneously.

Consult the section on Input/output channels for more information.

E2013  Attempting to access a non allocated channel

The channel number provided to the instruction is not associated with any file nor any input/output buffer.

Before using an input/output channel for reading or writing, the channel must first be associated with a file or an input/output buffer through the OPEN instruction.

Consult the section on Input/output channels for more information.

E2014  Invalid access to specified channel

The channel number provided to a READ or WRITE instruction does not allow the given input/output operation.

When opening of a file or an input/output buffer using the OPEN instruction, an access mode must be provided for accessing the document’s contents, indicating whether the document is to be accessed for reading data from it or for writing results to it.

This error occurs when an algorithm attempts to read in from document (through its channel) opened in write mode, or to write to a document opened in read mode.

Consult sections on Input/output channels and Opening a document for more information.

E2015  File or I/O buffer already opened on another channel

The file or input/output buffer specified in an OPEN instruction is already associated with another input/output channel.

It is not allowed to open the same document more than once simultaneously, even on different channels. To circumvent the error, close the channel linked to the document involved in the error (with the CLOSE instruction) and re-open the document on another channel.

Consult sections on Input/output channels and Opening a document for more information.

E2016  Unable to open specified file or I/O buffer

LARP is unable to open the file or the input/output buffer specified in an OPEN instruction.

If the document opened is a file, the error may be due to the impossibility of accessing the media (hard disc, memory stick or others), a malfunction of this media or to a corrupt file.
Make sure the requested file exists and is accessible if it is to be read, or that it can be overwritten if it already exists and must be replaced.

Errors may also be caused by difficulties in accessing the temporary files directory. This directory is used to store files automatically created by LARP to manage input/output buffers.

For more information on temporary files and selecting an alternate directory where to store them, consult the section on configuration of the execution console.

**E2017 Unable to open a temporary file**

When opening an input/output buffer, LARP creates a temporary file to handle accesses to the buffer’s contents. This temporary file is stored in a directory provided by Windows®. This error means that Windows® is unable to provide such directory to LARP.

The most probable causes of such error are the unavailability of space on the target media (hard disc for example), the inability to write to the media or a malfunction of this media.

Errors may also be caused by difficulties in accessing the temporary files directory. This directory is used to store files automatically created by LARP to manage input/output buffers.

For more information on temporary files and selecting an alternate directory where to store them, consult the section on configuration of the execution console.

**E2018 Unable to access specified file or I/O buffer**

The specified file or input/output buffer has been opened successfully with an OPEN instruction but an error occurred upon reading from the document or writing to the document through the associated input/output channel.

Reasons for such errors are numerous, among which running out of space on the media (hard disk or memory stick for example), a sudden unavailability the media (for example, unplugging a memory stick) or a malfunction of this media.

**E2019 End of file or I/O buffer reached**

An attempt was made to read from a file or an input/output buffer while the end of the document was reached (i.e. there is no more data to be read).

Use the predefined function ENDOFCONTENT to detect when the end of a document is reached.

**E2020 Invalid format string**

Invalid format specifications are provided to the predefined function FORMAT.

Consult the documentation on this function to determine how to resolve the problem.

**E2021 Call stack overflow (maybe due to infinite recursion)**

The algorithm caused infinite recursion during its execution. When a module calls itself or two modules mutually call themselves, the computer’s memory will eventually run out if this invoke process continues forever. LARP detected such situation.

Consult the section on recursion for more information.
E2022 Variable value cannot be modified

Depending on circumstances, a variable may sometimes be locked so that its value cannot be changed. An example of such situation is an iteration variable which cannot be changed inside its repetitive FOR structure.

The algorithm explicitly attempted to modify the value of a locked variable.

E2023 Iteration variable «variable name» cannot be modified within the loop

An iteration variable cannot be changed by instructions inside its repetitive FOR structure. This variable is implicitly and exclusively updated by the FOR instruction at each iteration. The error occurred because the algorithm explicitly attempted to modify the value of an iteration variable.

Consult the section on the FOR structure for more information.

E2024 Variable «variable name» contains an invalid file or I/O buffer name

The document name provided to an OPEN instruction through the specified variable is invalid.

Make sure the specified variable contains the name of a file or an input/output buffer to be opened. If what is specified in the OPEN instruction is not a variable but the name of the document to be opened, you probably forgot to put this name within quotes for it to be considered a character string.

When opening a file the provided file name is invalid if it does not conform to file names allowed by Windows®, or if the file to be opened for reading does not exist. When opening an input/output buffer the buffer name is invalid if it does not correspond to any existing buffer in the project.

Consult the section on Files and input/output buffers for more information.

E2025 Invalid file or I/O buffer name (did you forget quotes?)

The name provided to an OPEN instruction is invalid. You perhaps forgot to specify this name within quotes for it to be considered a character string.

When opening a file the provided file name is invalid if it does not conform to file names allowed by Windows®, or if the file to be opened for reading does not exist. When opening an input/output buffer the buffer name is invalid if it does not correspond to any existing buffer in the project.

Consult the section on Files and input/output buffers for more information.

E2026 Infinite loop caused by sign of step value opposite to the direction of iterations

A FOR structure forces an iteration variable to step in opposite direction of the starting and ending values for that variable. An example of such error is an iteration variable which must iterate through values 1 to 10 while the provided step value is negative (FOR i = 1 TO 10 STEP -1 DO).

Make sure the provided step value is signed according to the starting and ending values imposed on the iteration variable (in the previous example, since i varies from 1 through 10, the iteration variable must be incremented at each iteration).
Consult the section on the FOR structure for more information.

E2027 Number of arguments provided in module call does not correspond to number of parameters in module header

A call to an auxiliary module defined in the project does not provide an appropriate number of arguments. Either the call does not have enough arguments or it has too many.

For more information on restrictions imposed on arguments and module parameters, see Auxiliary module parameters.

E2028 A module reference parameter does not have a variable as corresponding argument in module call

A module call does not provide a variable name as argument to a corresponding module reference parameter. When a module header includes a reference parameter, any call to this module must provide as corresponding argument the name of a variable able to receive any value assigned to its corresponding reference parameter by the called module.

For more information on reference parameters, see Reference parameters.

E2029 Invoking an undefined module

The algorithm calls an module not defined within the project.

Consult the section on auxiliary modules for more information.

E2101 Error #«error code» in software (address «address») - contact technical support

This error message indicates that an unexpected error occurred during the execution of an algorithm.

Contact LARP's technical support and provide the information included in the error message (the error code and address where the error occurred in LARP).

E2102 Error («description») in software (address «address») - contact technical support

This error message indicates that an unexpected error occurred during the execution of an algorithm.

Contact LARP's technical support and provide the information included in the error message (the description and address where the error occurred in LARP).

E2103 The processor detected an invalid arithmetic operation

An error occurred in the computer’s processor while processing arithmetic operations. This error is probably due to a bug in LARP, such as an arithmetical operation producing a result too large or too small to be manipulated by the computer.

This error should rarely occur in LARP. Please contact LARP's technical support.

E2104 Floating point value exceeds processor capacity (overflow)

An arithmetic operation in the algorithm produced a float result too large to be manipulated by the computer.

For more information on float value limits, consult the section on numerical values.
E2105  Floating point value exceeds processor capacity (underflow)

An arithmetic operation in the algorithm produced a float result too small to be manipulated by the computer.

For more information on float value limits, consult the section on numerical values.

E2106  Attempt to divide by zero

An arithmetic operation in the algorithm involved a division in which the denominator is 0.

E2107  Unknown mathematical error implicating a floating point value

An arithmetic operation involving float values failed to be evaluated during execution, even though it is syntactically correct.

In other words, the computer's processor was unable to compute the requested arithmetic operation.

E2108  Unknown mathematical error implicating an integer value

An arithmetic operation involving integer values failed to be evaluated during execution, even though it is syntactically correct.

In other words, the computer's processor was unable to compute the requested arithmetic operation.

E2109  Integer value too large (overflow) or invalid index value

An arithmetic operation within an algorithm produced a result too large to be manipulated by the computer.

For more information on integer value limits, consult the section on numerical values.

The error may also have been caused by an invalid access to a container element.

E2110  Integer value too large (overflow)

An arithmetic operation within an algorithm produced a result too large to be manipulated by the computer.

For more information on integer value limits, consult the section on numerical values.

E2111  Out of random access memory (RAM)

There is not enough random access memory (RAM) in the computer to execute the algorithm. Among causes for such error are oversized containers or too many applications running simultaneously along with LARP.

To resolve the problem, close all non essential applications and, if the algorithm uses containers, reduce their size.

E2112  File not found

The file name provided to an OPEN instruction for access in READMODE is invalid. Most probably the file does not exist or, if it does, it is inaccessible (may be locked by another application).
E2113 Invalid file name

A file name provided to an OPEN instruction is invalid. File names must abide by file naming rules imposed by Windows ®.

Consult the section on files and input/output buffers for more information.

E2114 Too many files opened simultaneously

Windows ® allows a limited number of files to be opened simultaneously. An attempt to open a file (either explicitly in an algorithm with the OPEN instruction or implicitly upon opening a temporary file) failed because this limit was reached.

To resolve the issue, consult Windows ® documentation to determine how many files can be opened simultaneously and change your algorithm in order not to open more files or input/output buffers than permitted.

E2115 Access to file refused

The file name provided to an OPEN instruction corresponds to a file to which LARP cannot have access.

Such error may be due to an attempt to open a file in WRITEMODE or APPENDMODE while no modification is allowed to the file. It may also be possible that the targeted file is temporarily locked by another application.

To resolve the issue, make sure the targeted file exists or is accessible according to the desired access mode.

The error may also be caused by an invalid access to the temporary files directory. This directory is used for storing files created by LARP to manage input/output buffers.

For more information on temporary files and their directory, consult the section on configuring the execution console.

E2116 End of file reached

An attempt to read the content of a file failed because the end of the file was reached (there was no more data to read).

Use the predefined function ENDOFCONTENT to detect end of file accessed in READMODE.

E2117 Out of disk space

The space available on the data storage media (disc, diskette or others) where are stored the files manipulated by the algorithm is exhausted. This error may also occur when input/output buffers are manipulated since they are implicitly linked to temporary files.

To resolve the issue, free up storage space on the media or use a larger storage media.

E2118 Invalid data read

A READ instruction involving an input/output channel associated to a file opened in READMODE failed. The most probable cause of such failure is an attempt to read data with invalid format. It might also be that the file or its content is corrupted.
For more information, consult sections on files and input/output buffers and reading through an input/output channel.

**E2119 Unknown file error**

An unexpected error occurred while executing a file operation (either opening the file, reading from it, writing to it or closing the file). The error may be also be caused by corrupted file contents.

Consult the section on files and input/output buffers for more information.

**E2120 Out of stack space (may be due to infinite recursion)**

The algorithm caused infinite recursion during its execution. When a module calls itself or two modules mutually call themselves, the computer’s memory will eventually run out if this calling process continues forever. LARP detected such situation.

Consult the section on recursion for more information. The call stack size may also be increased (see configuring the execution console). If the error is not caused by infinite recursion, contact technical support to inform them of the problem.

**E2121 Operation involving incompatible data types**

Some types of variables and numerical values cannot be combined in arithmetic expressions. One such example is to divide a number by a character string (for example, 10/"Hello").

LARP interrupts the execution of the algorithm and displays this error message when it encounters such flawed expressions.

**E2122 Access error to a file («description»)**

This message indicates an error which occurred during an access to a file or an input/output buffer. An additional error description provided by Windows® accompanies the displayed message.

The most probable cause for this error is a failure to access a file which suddenly became unavailable. Make sure the storage media (disc, diskette, etc) containing the file is accessible and operational.

The error may also occur during access to the temporary files directory. This directory is used to store files transparently created by LARP when handling input/output buffers during algorithm execution. For more information on the temporary files and their directory, consult the section on Configuration of the execution console.

**E2999 Unknown error; contact technical support**

An unexpected error (not anticipated by the developers of LARP) occurred. Please contact LARP’s technical support for help.

**E3001 Opened I/O buffers or files have not been closed**

The algorithm in execution opened files or input/output buffers (with OPEN instructions) but did not close them upon termination.

Consult online help on the CLOSE instruction for more information.
E3002 Variable «variable name» without a value

The algorithm refers to the value of a variable while no value has previously been assigned to this variable.

Make sure to assign a value to the variable before using it in expressions and LARP instructions. You may also have misspelled the name of a variable, making it distinct from another variable to which a value has been assigned.

When an algorithm refers to a variable with no assigned value, the variable is said to be indeterminate. Indeterminate variables often cause fatal errors during the execution of algorithms.

For more information, consult the section on assignment.

E3003 Variable «variable name» without a value (maybe confusing with variable «variable name»?)

The algorithm refers to the value of a variable while no value has previously been assigned to this variable.

Make sure to assign a value to the variable before using it in expressions and LARP instructions. You may also have misspelled the name of a variable, making it distinct from another variable to which a value has been assigned.

When an algorithm refers to a variable with no assigned value, the variable is said to be indeterminate. Indeterminate variables often cause fatal errors during the execution of algorithms.

For more information, consult the section on assignment.

E3004 Container element «container name» without a value

The algorithm refers to the value of a nonexistent container element.

Make sure to assign a value to the container position before using the element in expressions and LARP instructions. You may also have misspelled the name of a container, making it distinct from another one to which values have properly been assigned. Finally, make sure of the index of the referred element is valid.

When an algorithm refers to a container element with no assigned value, the element is said to be indeterminate. Indeterminate container elements and variables often cause fatal errors during the execution of algorithms.

For more information, consult the section on assignment.

E3005 Container element «container name» without a value (maybe confusing with variable «variable name»?)

The algorithm refers to the value of a nonexistent container element.

Make sure to assign a value to the container position before using the element in expressions and LARP instructions. You may also have misspelled the name of a container, making it distinct from another one to which values have properly been assigned; LARP identified another container or variable having an assigned value and a name closely resembling the erroneous container. Finally, make sure of the index of the referred element is valid.
When an algorithm refers to a container element with no assigned value, the element is said to be *indeterminate*. Indeterminate container elements and variables often cause fatal errors during the execution of algorithms.

For more information, consult the section on assignment.

**E3006  Calling a module which do not return a value**

A module which does not return a value is invoked as though it does return a value.

For more information, consult the section on modules with a return value.

**E3007  Access to indeterminate container element**

The algorithm refers to the value of a nonexistent container element.

Make sure to assign a value to the container position before using the element in expressions and LARP instructions. You may also have misspelled the name of a container, making it distinct from another one to which values have properly been assigned. Finally, make sure of the index of the referred element is valid.

When an algorithm refers to a container element with no assigned value, the element is said to be *indeterminate*. Indeterminate container elements and variables often cause fatal errors during the execution of algorithms.

For more information, consult the section on assignment.

**E3008  The FOR loops in this project's flowcharts must be validated since they may not respect the new format of unconditional iterative structures**

The format of a flowchart FOR structure had to be modified in order to make its syntax match its pseudo code counterpart.

You opened a project file which was created with an earlier version of LARP, and your current version of LARP had to convert flowchart FOR structures within the project’s algorithms to the new version of the iterative structure. Since such conversions are not infallible, you should verify all flowchart FOR structures within the project to validate their components (the iteration variable, start and end values, and the step value).

For more information on FOR loops and their syntax, see the section on FOR structures.

**E9999  Unknown error; contact technical support**

An unexpected error (not anticipated by the developers of LARP) occurred. Please contact LARP's technical support for help.
Index

= (assignment) ............................................ See Assignment
>  Priority of operators .................................. 118
  Relational operators ................................ 114
>= Priority of operators ................................ 118
  Relational operators ................................ 114

A
ABS ..................................................... See ABSOLUTE
ABSOLUTE ............................................. 108, 171
Accelerator keys
  Graphical editor commands accessible through the
  keyboard .................................................. 57
  Textual editor commands accessible through the
  keyboard .................................................. 48
  Top menu .............................................. 35
Access mode
  Detecting end of content through input/output
  channels .................................................. 161
ENDOFCONTENT ......................................... 176
  Opening a document ................................... 155
Access modes ........................................... 157
Active username ....................................... 71
Addition .................................................. 71
  See +
  Algorithm Introduction ................................ 17
AND
  Logical operators ....................................... 117
  Priority of operators ................................ 118
  Priority of operators ................................ 118
  Animation .................................................. 62
  Syntax (files) ......................................... 199
  Syntax (input/output buffers) ........................ 199
  ARCTAN ............................................... See ARCTANGENT
  ARCTANGENT ......................................... 108, 172
Argument
  Parameter declarations in module header .......... 145
  Reference parameters ................................ 148
  Syntax of instructions ................................ 80
  Value parameters ..................................... 146
Arithmetic operator ................................... 105
  Priority of operators ................................ 117
Array .................................................... See Container
ASCII
  Auxiliary module ....................................... 141
  Alternate call syntax .................................. 151
  Modules with a return value ........................ 151
ASCII ........................................................ 164
  Access to container elements ........................ 94
  Access to container elements ........................ 94
  Modules with a return value ........................ 151
  Operations ............................................ 88
  Read instruction ....................................... 98
  Variable names ......................................... 88
Assignment .............................................. 90
  Access to container elements ........................ 94
  Access to container elements ........................ 94
  Modules with a return value ........................ 151
  Operations ............................................ 88
  Read instruction ....................................... 98
  Variable names ......................................... 88
Auxiliary module ....................................... 141
Base 10 .................................................. See Binary coding
LOG10 .................................................... 181
REPEAT-UNTIL structure ............................................. 133
Type validation .................................................. 116
WHILE structure .................................................. 131
Conditional instruction
Conditional structures ........................................... 113
Conditional loop
FOR structure .................................................. 138
Conditional structure ......................................... 113
Detecting end of content through input/output
channels .......................................................... 161
Embedded IF-ELSE structures ............................ 121
IF-ELSE-IF structure ............................................ 123
Conditional structures
IF and IF-ELSE structures .................................. 119
SELECT structure ............................................... 127
Configuration ...................................................... 64
Color selection .................................................... 68
Colors in the editors ............................................ 69
Colors in the execution console ......................... 69
Configuration of editors ..................................... 65
Configuration of the execution console ............... 65
Configuration of the super user mode and the
updating system .................................................. 67
Graphical editor configuration ............................ 55
Constant ............................................................ 87
Arithmetic operators .......................................... 105
EXP (e) ............................................................ 177
Numeric ............................................................. 89
PI (π) ............................................................... 184
SELECT structure ............................................... 127
Container .......................................................... 93
Access to container elements ........................... 93
Configuration of the execution console .......... 66
Container operators .......................................... 107
COUNT .......................................................... 174
Grouping values together .................................. 93
MAXIMUM ..................................................... 183
MINIMUM ...................................................... 184
POSITION ....................................................... 185
Predefined container functions ......................... 110
Relational operators .......................................... 115
Retrieving container elements .......................... 94
SIZE .............................................................. 188
SUBSET .......................................................... 190
TOCHARACTERS .............................................. 192
TOSTRING ........................................................ 193
CONTAINER
Type validation .................................................. 116
Container element .......................................... 93
Access to container elements .......................... 93
Container functions
COUNT .......................................................... 174
DATE ............................................................ 175
MAXIMUM ..................................................... 183
MINIMUM ...................................................... 184
POSITION ....................................................... 185
SIZE .............................................................. 188
SUBSET .......................................................... 190
TIME .............................................................. 191
TOCHARACTERS .............................................. 192
TOSTRING ........................................................ 193
Contextual menu
Document browser ......................................... 39
Graphical editor commands accessible through
menus .......................................................... 56
Graphical editor’s edit commands .................. 55
Inserting, moving and deleting flowchart instructions
................................................................. 53
Mouse control in the textual editor .................. 49, 57
Textual editor commands accessible through menus
................................................................. 47
Textual editor's edit commands ......................... 47
Warnings and errors ........................................... 63
Contextual Menu
Editing a textual document .................................. 45
Control panel ................................................... 37
Editing a flowchart ............................................ 51
Editing a textual document .................................. 45
Graphical editor commands accessible through
menus .......................................................... 56
Graphical editor functionalities ........................ 49
Graphical editor’s edit commands .................. 56
Interface elements ............................................ 34
Step execution interface ................................... 59
Textual editor commands accessible through menus
................................................................. 48
Textual editor's edit commands ......................... 47
Conversion
Converting flowcharts to pseudo code ............... 76
Conversion of type
Arithmetic operators ...................................... 106
String operators .............................................. 106
Copyright ........................................................ 20
COS ............................................................. 109
COSINUS ....................................................... 109
ARCTANGENT .................................................. 172
COUNT ........................................................ 109, 110, 174
SIZE ............................................................. 189
Crash of LARP .................................................. 28
Bug reports ....................................................... 28
Security backups .............................................. 63
Cut
Inserting, moving and deleting flowchart instructions
................................................................. 54
Cut and paste
Converting flowcharts to pseudo code ............... 77
Cut and paste restrictions .................................. 73
D
DATE ............................................................. 175
Decimal ................................................................ 20
Decimal coding ................................................ 163
DEFINED
Type validation .................................................. 117
Delete instructions
Inserting, moving and deleting flowchart instructions
................................................................. 54
DESTROY ......................................................... 94
DETERMINE
Type validation .................................................. 116
Development environment .................................. 33
General configuration ....................................... 64
Input/output buffers .......................................... 153
Interface elements .......................................... 33
Introduction ..................................................... 17
Messages related to the development environment
................................................................. 201
Online help ....................................................... 28
Difference
Container operators .......................................... 107
Directory
Opening a file .................................................. 156
Distribution ...................................................... 20
Division .......................................................... 57
FOR structure .................................................. 135
Syntax (FOR structure) ...................................... 198

Copyright © 2004-2008 Marco Lavoie
Index

LARP

Introduction ...................................................... 17
LARP release ...................................................... 19
LARP's splash window also indicates which version is
currently running. License ................................. 20
LARP's author ...................................................... 21
LENGTH ................................................................ 110 See COUNT
License agreement .................................................. 18
Registration ............................................................ 23
License number ...................................................... 25
Lisp ....................................................................... 27
Constants and variables ........................................... 87
Local variables ....................................................... 143
LOG10 .................................................................. 109, 181
LOGE .................................................................. 109, 182
EXP ..................................................................... 177
Logging ................................................................... 29
Logical operator ..................................................... 117
Conditions ............................................................ 114
Priority of operators .............................................. 117
Relational operators ................................................. 116
Loop........................................................................ 25
See Iterative structure
Lowercase ................................................................ 87
Variable names ...................................................... 87
LOWERCASE ..................................................... 110, 182

M

Main module .......................................................... 140
Parameter declarations in module header .................. 146
Mathematical functions ........................................... 146
ABSOLUTE ............................................................ 171
CEILING ............................................................. 173
EXP ................................................................. 177
FLOOR ............................................................... 177
LOG10 ............................................................... 181
LOGE ............................................................... 182
MAXIMUM .......................................................... 183
MINIMUM .......................................................... 184
RANDOM ............................................................ 186
ROUND ............................................................. 187
SQUAREROOT .................................................... 189
MAX ................................................................. 189
MAXIMUM .......................................................... 183
See MAXIMUM
MINIMUM .......................................................... 184
MINIMUM .......................................................... 184
MINIMUM .......................................................... 184
OPERATIONS .................................................... 187
QUAD ............................................................... 189
REPEAT-UNTIL ..................................................... 132
REPEAT-UNTIL structure ......................................... 133
Recursion ............................................................. 119
Relation operators .................................................. 116
Loops .................................................................... 116
Loop ..................................................................... 131
Iterative structure .................................................... 131
Detecting end of content through input/output
channels ................................................................ 161
FOR structure ....................................................... 135
Recursion ............................................................. 169
REPEAT-UNTIL structure ......................................... 133
WHILE structure .................................................... 131

J

Java ....................................................................... 27
Assignment .............................................................. 90
Constants and variables ........................................... 87
Containers ............................................................. 93
FOR structure ....................................................... 138
Introduction ........................................................... 138
Operators and predefined functions ......................... 105

K

Keyboard .................................................................. 42
Detecting end of content through input/output
channels .................................................................. 161
Files and input/output buffers .................................. 153
Graphical editor's edit commands ......................... 156
Read instruction ...................................................... 98
Textual editor commands accessible through the
keyboard .............................................................. 48
Textual editor's edit commands ............................... 47

E1001 .................................................................. 204
E1002 .................................................................. 205
E1003 .................................................................. 205
E1004 .................................................................. 205
E1005 .................................................................. 205
E1006 .................................................................. 205
E1007 .................................................................. 205
E1008 .................................................................. 205
E1009 .................................................................. 206
E1010 .................................................................. 206

Copyright © 2004-2008 Marco Lavoie
LARP Users Guide

Index

Uninstallation .......................................................... 22
License agreement .................................................. 19
Registration .......................................................... 23
Unlock
Ordering super user keys ........................................... 25
Super user mode ..................................................... 71
Unlock key
Configuration of the super user mode and the updating system ............................................. 67
Unlock key manager .................................................. 67
Unlocking .............................................................. 74
Configuration of the super user mode and the updating system ............................................. 67
UNTIL
REPEAT-UNTIL structure ......................................... 134
Syntax (REPEAT-UNTIL structure) .......................... 197
Updates .................................................................. 26
License agreement .................................................. 18
Uppercase
Variable names ....................................................... 87
UPPERCASE ............................................................ 110, 194
Uppercase and lowercase
Separation of instructions ....................................... 81
Uppercases and lowercases
Conversion ............................................................ 182, 194
Username
Active username .................................................... 71
Configuration of the super user mode and the updating system ............................................. 67
Creating a LARP project .......................................... 83
Document encryption ............................................... 73
Project statistics ...................................................... 76
Public project files .................................................. 77
Super user mode ..................................................... 71
Unlocking the development environment ................... 75
Username attached to project files ............................. 72
Users guide ........................................................... 33

V

Value limits .............................................................. 89
Value parameter
Auxiliary module parameters .................................. 144
Parameter declarations in module header ................. 146
Value parameter ..................................................... 146
Variable .............................................................. 87
Access to container elements .................................. 93
Arithmetic operators ................................................. 105
Assignment .......................................................... 90
Local variables ....................................................... 143
Operations ............................................................ 88
Parameter declarations in module header ................. 144
Step execution interface ......................................... 60
Value parameters ................................................... 146
Variable names ...................................................... 87
Variables inspection ............................................... 62
Variable name ....................................................... 87

W

Walking
Step execution modes ................................................. 61
Warnings ............................................................ 63, 201
Compilation and execution .................................... 58
Online help ........................................................... 28
Web
bugs ................................................................. 31
Cleaning ............................................................. 28
Configuration of the super user mode and the updating system ............................................. 68

Unconditional loop
FOR structure ....................................................... 138
UNDEFINED
Type validation ..................................................... 117
Undo operations ..................................................... 65
LARP updates ....................................................... 26
LARP’s author ....................................................... 21
Ordering super user keys ...................................... 26
Registration procedure ........................................ 24
Welcome window ....................................................... 82
WHILE ..................................................................... 197
FOR structure ...................................................... 135
REPEAT-UNTIL structure .................................... 133
WHILE structure .................................................. 131
WHILE structure .................................................. See Iterative structures
WRITE ..................................................................... 99, 200
Access modes ..................................................... 157
Colors in the execution console ....................... 69
Execution console ................................................. 43
Outputs through input/output channels .......... 160
Query instruction ................................................. 100
WRITEMODE .......................................................... 157
Outputs through input/output channels .......... 160
Syntax (files) ........................................................ 199
Syntax (input/output buffers) ........................... 199
Writing ................................................................. 99
Query instruction ................................................. 100
Separator ............................................................ 102

Z

Zoom
Graphical editor commands accessible through the keyboard .............................................. 56
Status panel ........................................................ 42
Zoom factor
Zooming the display ........................................... 55
Zooming ............................................................... 55