

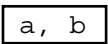
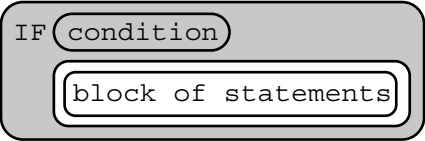
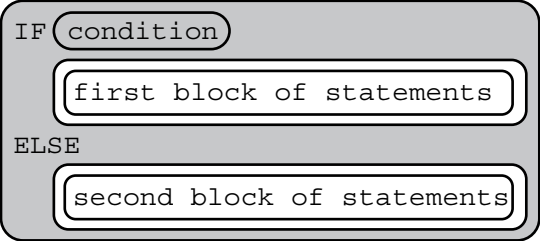
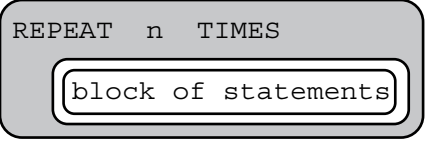

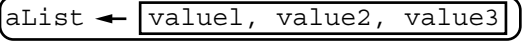
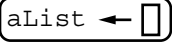
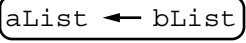

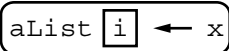
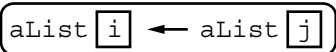
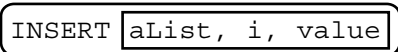


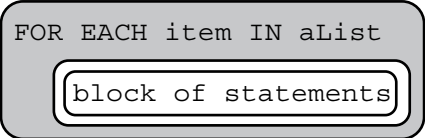


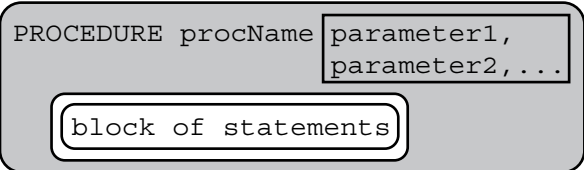
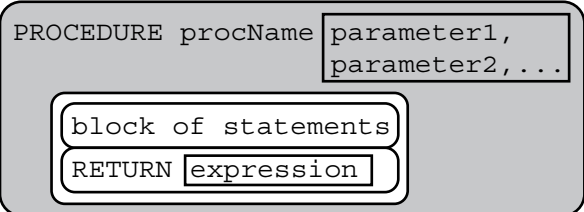


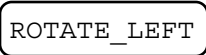
# CSP Exam Reference Sheet

Instruction	Explanation
<b>Assignment, Display, and Input</b>	
Text: <code>a ← expression</code>  Block: 	Evaluates <code>expression</code> and then assigns a copy of the result to the variable <code>a</code> .
Text: <code>DISPLAY(expression)</code>  Block: 	Displays the value of <code>expression</code> , followed by a space.
Text: <code>INPUT()</code>  Block: <code>INPUT</code>	Accepts a value from the user and returns the input value.
<b>Arithmetic Operators and Numeric Procedures</b>	
Text and Block: <code>a + b</code> <code>a - b</code> <code>a * b</code> <code>a / b</code>	The arithmetic operators <code>+</code> , <code>-</code> , <code>*</code> , and <code>/</code> are used to perform arithmetic on <code>a</code> and <code>b</code> .  For example, <code>17 / 5</code> evaluates to <code>3.4</code> .  The order of operations used in mathematics applies when evaluating expressions.
Text and Block: <code>a MOD b</code>	Evaluates to the remainder when <code>a</code> is divided by <code>b</code> . Assume that <code>a</code> is an integer greater than or equal to <code>0</code> and <code>b</code> is an integer greater than <code>0</code> .  For example, <code>17 MOD 5</code> evaluates to <code>2</code> .  The <code>MOD</code> operator has the same precedence as the <code>*</code> and <code>/</code> operators.
Text: <code>RANDOM(a, b)</code>  Block: <code>RANDOM</code> 	Generates and returns a random integer from <code>a</code> to <code>b</code> , including <code>a</code> and <code>b</code> . Each result is equally likely to occur.  For example, <code>RANDOM(1, 3)</code> could return <code>1</code> , <code>2</code> , or <code>3</code> .
<b>Relational and Boolean Operators</b>	
Text and Block: <code>a = b</code> <code>a ≠ b</code> <code>a &gt; b</code> <code>a &lt; b</code> <code>a ≥ b</code> <code>a ≤ b</code>	The relational operators <code>=</code> , <code>≠</code> , <code>&gt;</code> , <code>&lt;</code> , <code>≥</code> , and <code>≤</code> are used to test the relationship between two variables, expressions, or values. A comparison using relational operators evaluates to a Boolean value.  For example, <code>a = b</code> evaluates to <code>true</code> if <code>a</code> and <code>b</code> are equal; otherwise it evaluates to <code>false</code> .

Instruction	Explanation
<b>Relational and Boolean Operators (continued)</b>	
Text: NOT condition  Block: NOT (condition)	Evaluates to true if condition is false; otherwise evaluates to false.
Text: condition1 AND condition2  Block: (condition1) AND (condition2)	Evaluates to true if both condition1 and condition2 are true; otherwise evaluates to false.
Text: condition1 OR condition2  Block: (condition1) OR (condition2)	Evaluates to true if condition1 is true or if condition2 is true or if both condition1 and condition2 are true; otherwise evaluates to false.
<b>Selection</b>	
Text: IF(condition) { <block of statements> }  Block: 	The code in block of statements is executed if the Boolean expression condition evaluates to true; no action is taken if condition evaluates to false.
Text: IF(condition) { <first block of statements> } ELSE { <second block of statements> }  Block: 	The code in first block of statements is executed if the Boolean expression condition evaluates to true; otherwise the code in second block of statements is executed.

Instruction	Explanation
<b>Iteration</b>	
<p>Text:  REPEAT n TIMES  {  &lt;block of statements&gt;  }  Block:</p> 	<p>The code in <b>block of statements</b> is executed n times.</p>
<p>Text:  REPEAT UNTIL(condition)  {  &lt;block of statements&gt;  }  Block:</p> 	<p>The code in <b>block of statements</b> is repeated until the Boolean expression <b>condition</b> evaluates to true.</p>
<b>List Operations</b>	
<p>For all list operations, if a list index is less than 1 or greater than the length of the list, an error message is produced and the program terminates.</p>	
<p>Text:  aList ← [value1, value2, value3, ...]  Block:</p> 	<p>Creates a new list that contains the values <b>value1</b>, <b>value2</b>, <b>value3</b>, and <b>...</b> at indices 1, 2, 3, and <b>...</b> respectively and assigns it to <b>aList</b>.</p>
<p>Text:  aList ← []  Block:</p> 	<p>Creates an empty list and assigns it to <b>aList</b>.</p>
<p>Text:  aList ← bList  Block:</p> 	<p>Assigns a copy of the list <b>bList</b> to the list <b>aList</b>.</p> <p>For example, if <b>bList</b> contains [20, 40, 60], then <b>aList</b> will also contain [20, 40, 60] after the assignment.</p>
<p>Text:  aList[i]  Block:  aList [i]</p>	<p>Accesses the element of <b>aList</b> at index <b>i</b>. The first element of <b>aList</b> is at index 1 and is accessed using the notation <b>aList[1]</b>.</p>

Instruction	Explanation
<b>List Operations (continued)</b>	
Text: <code>x ← aList[i]</code> Block: 	Assigns the value of <code>aList[i]</code> to the variable <code>x</code> .
Text: <code>aList[i] ← x</code> Block: 	Assigns the value of <code>x</code> to <code>aList[i]</code> .
Text: <code>aList[i] ← aList[j]</code> Block: 	Assigns the value of <code>aList[j]</code> to <code>aList[i]</code> .
Text: <code>INSERT(aList, i, value)</code> Block: 	Any values in <code>aList</code> at indices greater than or equal to <code>i</code> are shifted one position to the right. The length of the list is increased by 1, and <code>value</code> is placed at index <code>i</code> in <code>aList</code> .
Text: <code>APPEND(aList, value)</code> Block: 	The length of <code>aList</code> is increased by 1, and <code>value</code> is placed at the end of <code>aList</code> .
Text: <code>REMOVE(aList, i)</code> Block: 	Removes the item at index <code>i</code> in <code>aList</code> and shifts to the left any values at indices greater than <code>i</code> . The length of <code>aList</code> is decreased by 1.
Text: <code>LENGTH(aList)</code> Block: <code>LENGTH aList</code>	Evaluates to the number of elements in <code>aList</code> .
Text: <code>FOR EACH item IN aList</code> <code>{</code> <code>&lt;block of statements&gt;</code> <code>}</code> Block: 	The variable <code>item</code> is assigned the value of each element of <code>aList</code> sequentially, in order, from the first element to the last element. The code in <code>block of statements</code> is executed once for each assignment of <code>item</code> .

Instruction	Explanation
<b>Procedures and Procedure Calls</b>	
<p>Text:</p> <pre>PROCEDURE procName(parameter1,                     parameter2, ...)</pre> <pre>{   &lt;block of statements&gt; }</pre> <p>Block:</p> 	<p>Defines <code>procName</code> as a procedure that takes zero or more arguments. The procedure contains <code>block of statements</code>.</p> <p>The procedure <code>procName</code> can be called using the following notation, where <code>arg1</code> is assigned to <code>parameter1</code>, <code>arg2</code> is assigned to <code>parameter2</code>, etc:</p> <pre>procName(arg1, arg2, ...)</pre>
<p>Text:</p> <pre>PROCEDURE procName(parameter1,                     parameter2, ...)</pre> <pre>{   &lt;block of statements&gt;   RETURN(expression) }</pre> <p>Block:</p> 	<p>Defines <code>procName</code> as a procedure that takes zero or more arguments. The procedure contains <code>block of statements</code> and returns the value of <code>expression</code>. The <code>RETURN</code> statement may appear at any point inside the procedure and causes an immediate return from the procedure back to the calling statement.</p> <p>The value returned by the procedure <code>procName</code> can be assigned to the variable <code>result</code> using the following notation:</p> <pre>result ← procName(arg1, arg2, ...)</pre>
<p>Text:</p> <pre>RETURN(expression)</pre> <p>Block:</p> 	<p>Returns the flow of control to the point where the procedure was called and returns the value of <code>expression</code>.</p>
<b>Robot</b>	
<p>If the robot attempts to move to a square that is not open or is beyond the edge of the grid, the robot will stay in its current location and the program will terminate.</p>	
<p>Text:</p> <pre>MOVE_FORWARD ( )</pre> <p>Block:</p> 	<p>The robot moves one square forward in the direction it is facing.</p>
<p>Text:</p> <pre>ROTATE_LEFT ( )</pre> <p>Block:</p> 	<p>The robot rotates in place 90 degrees counterclockwise (i.e., makes an in-place left turn).</p>

Instruction	Explanation
<p>Text: ROTATE_RIGHT ( )</p> <p>Block:</p> <div style="border: 1px solid black; border-radius: 10px; padding: 5px; width: fit-content; margin: 5px auto;">ROTATE_RIGHT</div>	<p style="text-align: center;"><b>Robot</b></p> <p>The robot rotates in place 90 degrees clockwise (i.e., makes an in-place right turn).</p>
<p>Text: CAN_MOVE (direction)</p> <p>Block:</p> <div style="border: 1px solid black; padding: 2px; display: inline-block;">CAN_MOVE <span style="border: 1px solid black; padding: 2px 10px;">direction</span></div>	<p>Evaluates to <code>true</code> if there is an open square one square in the direction relative to where the robot is facing; otherwise evaluates to <code>false</code>. The value of <code>direction</code> can be <code>left</code>, <code>right</code>, <code>forward</code>, or <code>backward</code>.</p>