CRYSTAL CLEAR
COMMUNICATIONS

Creators of the **Language Arts** technology

# New Employee Handbook and Technology Manual

# Table of Contents

## A Welcome From Our Founder

What a truly great time to be a part of crafting the future.

It is my pleasure to personally welcome you to Crystal Clear Communications. As we approach the 21st century, together we have the exciting opportunity to craft and utilize the state of the art in communications technology. Our company and our clients are counting on you to break down the barriers in communication across the globe and open the world to a new future of possibilities.

**Evan Moore**
President, Founder, and CEO

## Company Mission

The mission of Crystal Clear Communication is to fundamentally transform global corporate communications by developing technology that incorporates the latest research and tools and utilizing those tools to solve the communications needs of our clients.

## Our Values

1. Our client's needs are our needs. Their priorities are our priorities.

2. Communicating with clarity is our obsession.

3. Every communication problem can be fixed with clear reasoning and order.

4. Efficiency and reliability make us stand out from the crowd.

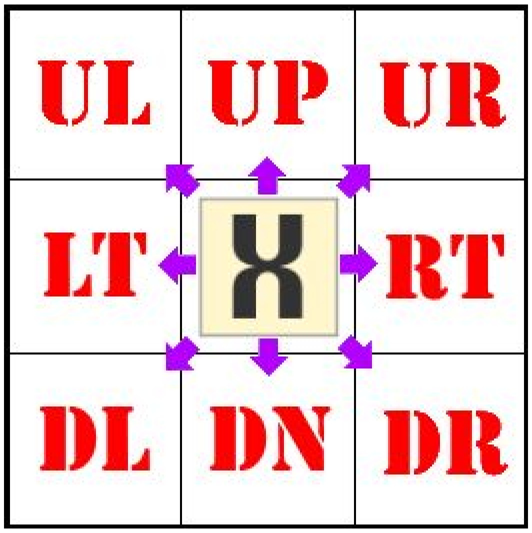
# Language Arts Manual

Language Arts is the new state of the art in communication management and editing software, and it is the foundational tool behind the inner workings at Crystal Clear Communications. In this manual you will learn how to interact with the program and create the rules that will power the communication of the future.

## Directions

Before learning anything else about Language Arts, it is important to know your directions. There are nine of them and each has a full name and abbreviation. They are:

- Up (UP), Down (DN), Left (LT), Right (RT)
- Up-Left (UL), Up-Right (UR), Down-Left (DL), Down-Right (DR)
- Here (HR)



The directions simply point from one grid space to the neighboring space indicated by the direction name. The Here direction indicates the current position in the grid.

Phil says

*Just because it isn't in the picture above, don't forget about the **Here** direction. It is often useful to compare your surroundings to yourself.*

## Rules

**Rules** are the building blocks of your solutions. In turn, rules are made up of their own building blocks, the most basic of which are the **Condition** and **Action**.

$$\text{Rule} = \text{Condition} + \text{Action}$$

To begin with, let’s stick with just one condition: `ALWAYS`.

### Basic Actions

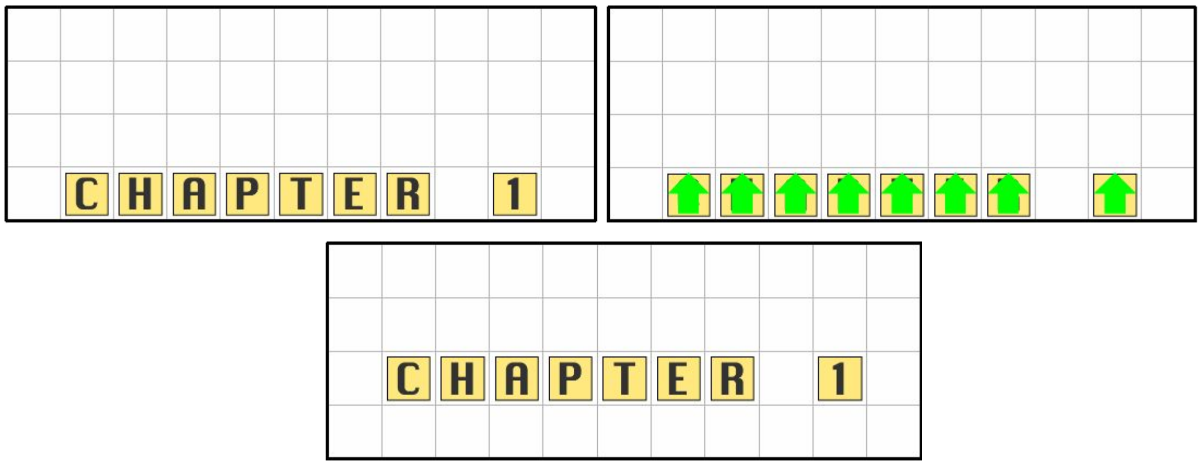
Some actions apply only in specific situations, but there are two action that can always apply to any character tile.

- **Move**
  Move is always followed by a direction and indicates that the character will try to move in the direction indicated so long as nothing obstructs it.
  *Examples*: `MOVE LEFT` -or- `MOVE DR`
- **Pass**
  Pass indicates that no action occurs in the current cycle for the character in question.

So let’s combine our condition and actions to make a rule: `ALWAYS MOVE UP`

#### Example: Orientation Exercise 1

In the first exercise in your orientation, you can enter the above rule then press the ⏭ (Next button) to see how the grid responds:



The movement of all the tiles by one place is considered a **Cycle**. In the following cycle, the rule would be applied again and the character tiles would all move to the second row from the top, and so on.

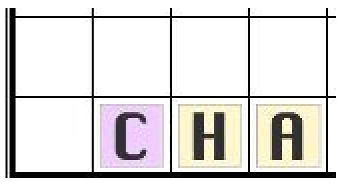
There are some other actions beyond `MOVE` and `PASS`, but we'll get back to those later.

## Conditions

A **Condition** is simply a statement that evaluates to either True or False. For each cycle evaluated as the solution plays, the first condition that evaluates to true will determine the action that will be taken for a particular character tile. If a rule's condition is false, the next rule in the rule set will be evaluated.

### Conditions with Equalities

Most conditions will begin with the word IF. The most common way of using a IF is to compare a direction with a value. For instance, see how the following rules are applied to the letter tile 'C' shown below:

| | | |
|---|---|---|
|  | ● `IF RT = H` | True |
| | ● `IF LT = H` | False |
| | ● `IF DN = WALL` | True |
| | ● `IF UP = EMPTY` | True |

### Example: Orientation Exercise 2

Let's take what we have learned so far and apply it to your second Orientation exercise. Here we want to move the `I` tiles to the left whenever there is an `E` to the left of the `I`. At the same time we want to move the `E` to the right if there is an `I` to the right of the `E`.

To do this we are going to need to use the **Logical Operator** AND. You can learn more about Logical Operators below. For now just know that AND will require the condition to its left as well as the one to it's right to be True in order for the entire condition to be True.

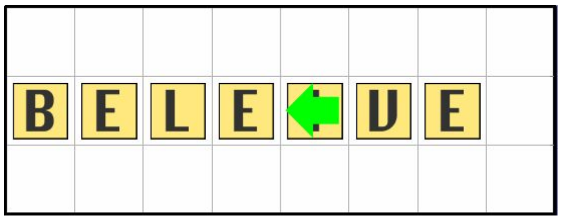
Let's start with the `I`. In the exercise enter the following:

```
IF HERE = I AND LEFT = E MOVE LEFT
```

Phil says

*When you add this rule you may notice that your rule has changed slightly. Language Arts automatically abbreviates directions and will also at times add or remove parentheses for clarity.*
*You should see:* `IF (HR = I AND LT = E) MOVE LT.`

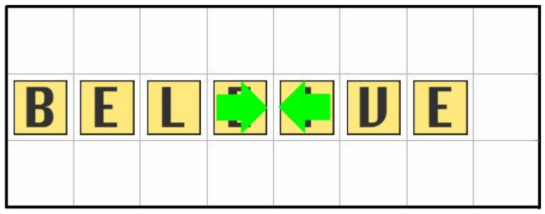
Using ⏭ on the solution so far will result in:

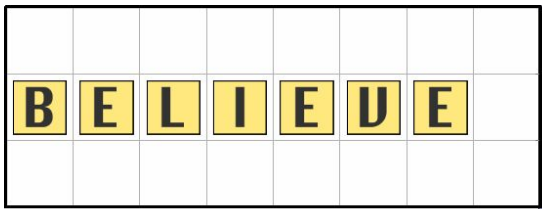


In this case, the `I` tile will not move. This is because the `E` tile is in the way. Tiles will only move into empty spaces. Fortunately, there is an exception to this where tiles will swap places with each other. So let's enter a second rule:

```
IF HERE = E AND RIGHT = I MOVE RIGHT
```

Restart the solution and try again. This time you will find that two tiles swap places.





Phil says

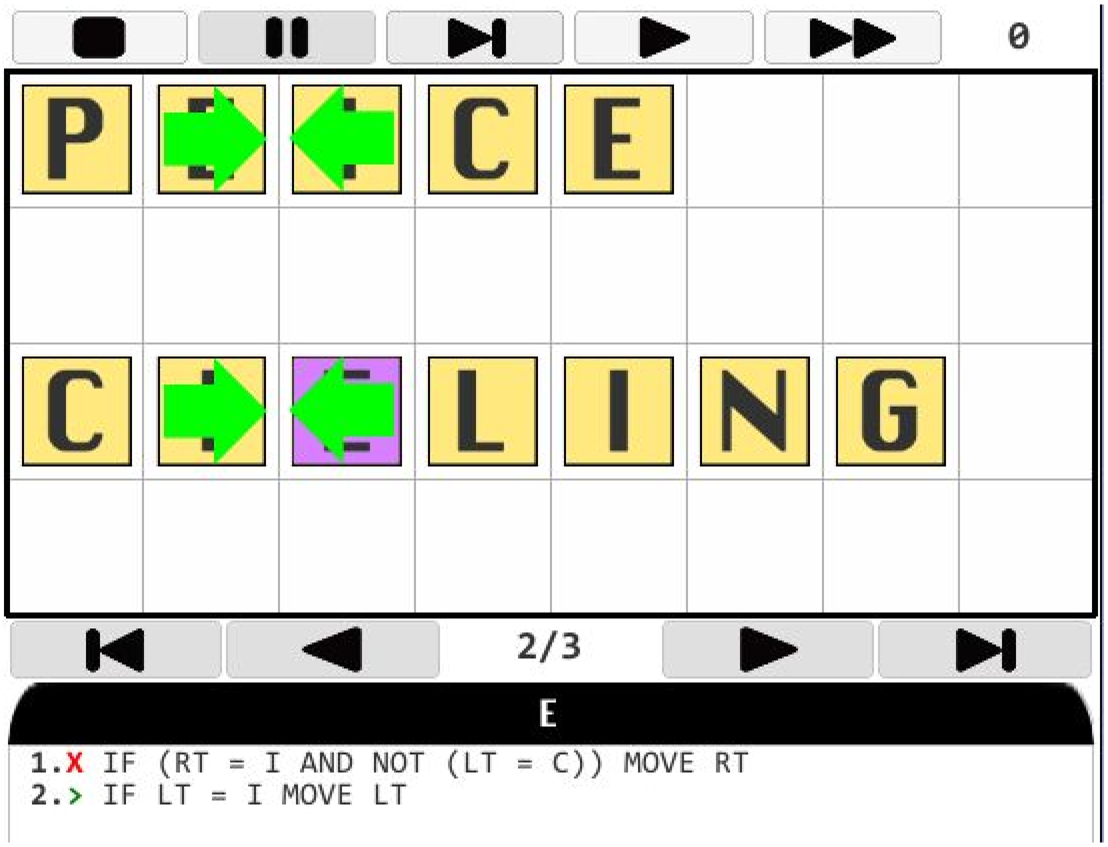
*Don't be afraid to ask for help. You can get help from Language Arts by entering **HELP**. You might even be able to ask your co-workers for a **HINT** or two.*

## Rule Sets

When you entered your rules into Language Arts, you probably saw that they were placed in a list in the bottom right of your screen under the word 'ALL'. This list is called a **Rule Set**.

Each cycle, every character tile will find if it has an action to take by evaluating the rules from top to bottom in its particular rule set. The first one whose condition evaluates to True will determine the action that tile will attempt to take on the given cycle.

You can see how this works in Exercise 2 by clicking on (and thus selecting) one of the letter tiles, then clicking the Next button. Try this for the various letter tiles in the exercise. You will see a red **X** next to rules that evaluated to False. You will see a green **>** next to the rule that evaluated to True. Rules underneath the one that evaluated to True will have no symbol next to them because evaluation of the rules stopped once the first rule came back as True.



```
1.X IF (RT = I AND NOT (LT = C)) MOVE RT
2.> IF LT = I MOVE LT
```

## Using Multiple Rule Sets - Revisiting Orientation Exercise 2

In our example for Exercise 2 we wanted different behavior for the `I` and `E` characters, so we wrote rules that first checked if we were one of those characters. We can get around having to do this by using multiple rule sets. In the case of Exercise 2 we want a rule set for the letter `I` and a rule set for the letter `E`.

To create a new rule set or to switch the display to one we have already created, you can enter `I:` or `E:` in your command prompt. The displayed rule set in the lower right of the screen should update once you do this.

Character tiles will look to see if a rule set has been created for its particular value. If so, it will use the rules in that rule set each cycle. The ‘ALL’ rule set is used as a fallback if no rule set has been created for the character value of a tile on the grid.

Knowing this we can create a new solution for Exercise 2 that looks like this:

```
I:
IF LEFT = E MOVE LEFT

E:
IF RIGHT = I MOVE RIGHT
```

## Advanced Topic: Movement

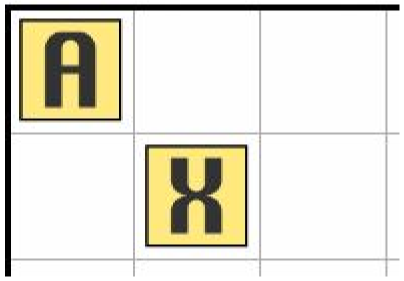
***Note:*** First time users of Language Arts may want to skip this section and revisit it once you have gained some additional experience.

Tiles move according to priority. Character tiles that come first in reading order (left to right, then top to bottom) get a higher priority when considering a move.

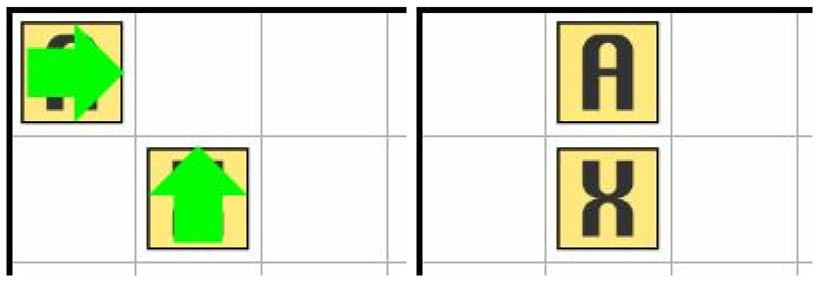
Let’s look at this in an example. Consider the following rules and grid:

```
ALL:
ALWAYS MOVE RIGHT

X:
ALWAYS MOVE UP
```



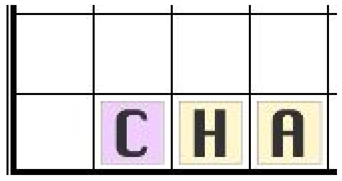
Both A and X want to occupy the same spot, but since A comes before X in priority order, A is going to take the new spot when movement begins.



## Conditions with Inequalities

When making comparisons with directions, you may also use a variety of operators other than equals =. These are: greater than >, less than <, greater than or equal to >=, and less than or equal to <=. These additional comparison operators may only be used when comparing a direction to an individual character or when comparing two directions to each other.

Using our previous example for rules for the letter C:



| | |
|---|---|
| `IF RT > H` | False |
| `IF RT <= H` | True |
| `IF RT > HR` | True |
| `IF RT > LT` | False |
| `IF UR > WALL` | Invalid Rule |

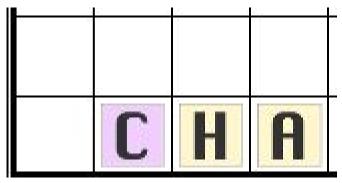
***Note***: Characters cannot see through walls. If in the above example a wall showed between C and H, only checking for `RT = WALL` would return true. Characters cannot even see diagonally if a wall is in the way. For instance C would not be able to see what is in the cell to the upper right if there was a wall either up or to the right of it.

## Type Checking Conditions

In addition to single character values, the values supported to check in a condition are:

- **WALL** - Indicated by thicker lines in the grid and impassable during movement
- **EMPTY** - A grid cell that has nothing occupying it
- **TRASH** - The trash can that destroys characters moved to it
- **CHARACTER** or **CHAR-** Any character occupant: numbers, letters, or punctuation.
- **NUMBER** - Any of the numeric digits 0 through 9
- **LETTER** - Any of the letters A through Z

- SYMBOL - Any character that is neither a number nor a letter



| | |
|---|---|
| `IF RT = LETTER` | True |
| `IF LT = EMPTY` | True |
| `IF DN = WALL` | True |
| `IF RT = NUMBER` | False |

Phil says

*As rules get longer, you might find yourself making little typos here and there. Don't sweat it. Language Arts keeps track of what you have typed. If you want to make a little change to what you just entered, just use the up and down arrow keys to bring up your input history and go from there.*

## Logical Operators

You may also apply **logical operators** to the condition.

- **NOT**
  Changes a false to true or a true to false.
  *Examples*: `NOT (LT = WALL)`, `NOT (DN = Z)`
- **AND**
  Expressions to the left and right must both be true for the whole expression to be true.
  *Examples*: `LT = WALL AND RT = WALL`, `LT > A AND LT < J`
- **OR**
  At least one expression to the left or the right must be true for the whole expression to be true.
  *Examples*: `LT = WALL OR RT = WALL`, `LT > J OR LT = NUMBER`

To avoid confusion, it is recommended that you use parentheses with logical operators. Conditions within parentheses are evaluated in the order of how deeply nested they are. In the case of the NOT operator, the parentheses are required.
*Examples*: `IF NOT (LT = WALL)`, `IF ((UP = WALL OR UP > J) AND DN = EMPTY)`

Phil says

*There are an awful lot of parentheses showing up. Expressions deeper in nested parentheses get evaluated first - just like in your math classes. When you enter a rule, Language Arts will simplify it as best as possible and also add missing parentheses to show you the order it will be checking things.*

### Example: Orientation Exercise 3

Exercise 3 is a lot like Exercise 2, but now we want `I` to move right if it has a `C` to its left as well as an `E` to it's right. The logical operator `AND` allows us to do this:

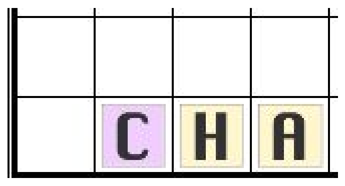
```
I:
IF (LEFT = C AND RIGHT = E) MOVE RIGHT
IF LEFT = E MOVE LEFT
```

Similarly, we want `E` to move left right when there is an I to the right, but we don't want this to happen if there is `C` to the left of the `E`. We can add a `NOT` operator to do this:

```
E:
1. IF (RIGHT = I AND NOT(LEFT = C)) MOVE RIGHT
2. IF LEFT = I MOVE LEFT
```

## Comparing Directions

The final feature of conditions is that directions can be used on both sides of the equality or inequality operator.
*Examples*: `IF LT = UP`, `IF DOWN < RIGHT`



| | |
|---|---|
| • `IF UP = EMPTY OR DN = EMPTY` | **True** |
| • `IF NOT (RT = LETTER)` | **False** |
| • `IF RT = EMPTY AND RT = UP` | **False** |
| • `IF RT = H AND NOT (LT = WALL)` | **True** |

Comparisons using >, >=, <, and <= will only return true when comparing two characters. The = operator will check not only if the characters are the same value but if the occupants are the same. For instance, are both directions empty? do they both have a wall or a trash? Etc.

Phil says

*Did you know that it is ok for your character tiles to still want to move after you have them all in the correct place? Language Arts knows what it wants the grid to look like in the end and it will stop the solution the first time it gets there. You just need to show it how to get there. You might first notice this happening in Orientation Exercise 4.*

### Advanced Actions

There are two additional actions that only have an effect on digit characters (i.e. the numbers from 0 to 9).

- **Add**
  Add may be followed by either a single digit number or by a direction. If a number is specified that number is added to the current digit when the rule is triggered. If a direction is specified, the occupant in the direction indicated is examined. If it is a number, that number is added to the current digit.
  *Examples*: `ADD 5`, `ADD UP`
- **Subtract**
  Subtract works like Add, only the number is subtracted from the current number. The subtract action can also be abbreviated using Sub.
  *Examples*: `SUBTRACT 5`, `SUB LT`

***Note***: Adding or subtracting is capped by the range of 0 through 9. Subtracting below 0 will change the number to 0 and adding above 9 will change the number to 9. Trying to add or subtract using directions when there is no number in the direction indicated or when a wall blocks visibility will leave the number unchanged.

## General Commands

While creating solutions in the Language Arts program, you may enter either a *Command* or a *Rule*. Commands are executed immediately by the program, while Rules are added to the solution to be run later.

Language Arts checks first to see if you have entered a command before checking to see if you have entered a rule. Some commands interact with the program itself, while others manipulate your solution.

When getting started, the most important command to familiarize with is **`HELP`**. You can use **`HELP`** to learn about each of these commands while you are using Language Arts.

These are the commands available in the program:

| **Command** | **Description** |
|---|---|
| `Help` | `Help` used by itself explains how to use the Help command. Using Help combined with other commands, such as `Help Edit`, will provide details about those commands |
| `Swap` | If the currently displayed rule set has more than one rule in it, you can use Swap to change the position of two rules within that set. For instance if you want Rule 1 and Rule 3 to trade places, you would enter `Swap 1 3`. |
| `Edit` | Entering a rule (`ALWAYS MOVE UP`) places it at the end of the currently displayed Rule Set. You can replace an existing rule by first specifying the number of the rule to replace (`3. ALWAYS MOVE UP`). If you want to make minor changes to the rule you want to replace, you can use the Edit command before replacing the rule (`Edit 3`). |
| `Insert` | Creates a rule at the index that you provide (`Insert 3`). The rule inserted will be one that never triggers. The rule can be edited and the Insert command will also call `Edit` on the new rule. |
| `List` | Characters follow the rule set associated with their character. If one is not defined, they use the ALL rule set instead. To know if a character has a rule set defined, you may want to see a list of all of the rule sets. Entering the `List` command will show you this information, letting you know if a particular character will be affected by the ALL rule set or not. |
| `Delete` | `Delete` can be used either to remove a single rule in the active rule set or it can be used to delete the rule set entirely. To delete a single rule, simply use the number for that rule (`Delete 5`). To delete the entire displayed rule set enter `Delete Rule Set`. Note that you cannot entirely delete the ALL rule set. In this case `Delete Rule Set` will clear all of the existing rules from the set. |
| `Exit` | The `Exit` command causes the program to leave the level editor interface and return to the level selection interface. When you use `Exit`, your work on the current solution should be saved. |

There are two special commands that don’t start with a keyword:

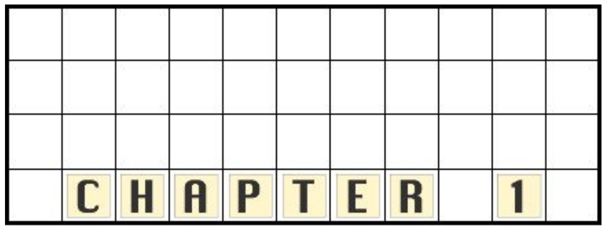
- Rule Set Change/Create
  This command is executed by entering the name of the rule set followed by a colon. If the rule set for that name exists, it will move to the displayed rule set. If it does not exist, it will be created then moved to the displayed rule set.
  *Examples*: `A:`, `Z:`, `ALL:`
- Existing Rule Edit
  This works the same way as entering a rule (described below), but in this case will replace an existing rule with the one entered rather than create a new one. This is done by placing the rule number followed by a period then the rule.
  *Example*: `1. ALWAYS MOVE DOWN`

# Appendix I. Orientation Walkthrough

While we recognize that the job we do for our clients can be complicated, we want your initial experience as a Communications Engineer to be as smooth as possible. We hope that you will try to solve the exercises in our Orientation on your own. If that ever becomes difficult, we provide this walkthrough as way to help you learn how to interact with Language Arts.

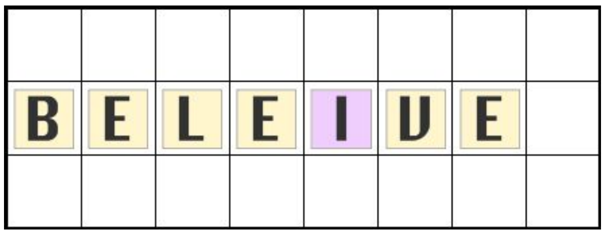
Once you fully enter the workplace, of course, you will need to create your own solutions for the jobs that you will be asked to perform.

## Exercise 1. Heading Up



```
ALL:
1. ALWAYS MOVE UP
```

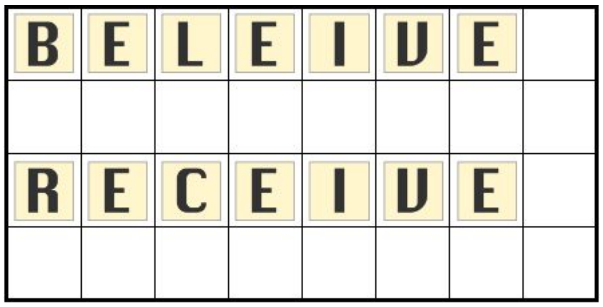
## Exercise 2. I Before E



```
I:
1. IF LT = E MOVE LT

E:
1. IF RT = I MOVE RT
```

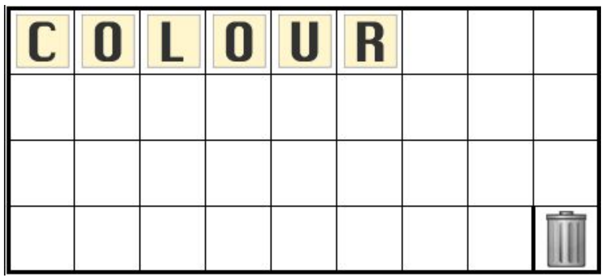
## Exercise 3. Except After C



```
I:
1. IF LT = E MOVE LT
2. IF (LT = C AND RT = E) MOVE RT

E:
1. IF (RT = I AND NOT(LT = C)) MOVE RT
2. IF LT = I MOVE LT
```

## Exercise 4. Brexit



```
R:
1. IF (LT = EMPTY AND NOT (DN = U)) MOVE LT

U:
1. IF (LT = O AND RT = R) MOVE DR
2. IF UR = CHARACTER MOVE UL
3. IF (LT = EMPTY AND RT = EMPTY) MOVE RT
4. IF RT = WALL MOVE DN
```

# Appendix II. Creating Your Own Puzzles

Crystal Clear Communications encourages the personal career development of all it's employees. To that end, we have included in the Language Arts program a means for creating and sharing your own communication puzzles.

## File Locations

Each user created puzzle is defined in its own file. These names of these files need to end in file extension ".lvl" and need to be placed in a specific directory on your computer. Once loaded, these puzzles will be shown in the Language Arts progrom under the **CCC Creators Club** grouping.

For Microsoft Windows users this location is:
`C:\Users\[YourUserName]\AppData\LocalLow\BustedThumbs\LanguageArts\`

For Macintosh users this location is:
`~/Library/Application Support/BustedThumbs/LanguageArts/`

These folders may be hidden to you by default in your OS. Consult the guides for your OS to learn how to add files to these locations.

## Sample Puzzle Level File

The following is a sample user defined puzzle:

```
Name:Sample User Puzzle
Goal:Swap the position of characters in all pairs of I before E.
Rows:3
Columns:8
Colored:No
Hint:Anne,Solving the puzzle is easier than creating it.
Hint:Lawrence,You have to be very careful not to have a typo or you will get an error.
Hint:Scott,I think that is enough talking for now.
Intro:Anne,Welcome to the sample user puzzle. Is this your first one?
Intro:Lawrence,Yes it is.
Intro:Anne,Lawrence, I wasn't talking to you. I was talking to _PlayerName_.
Outro:Evan,Nicely done. We should consider recruiting whoever put this puzzle
together.
```

```
Grids:3
CharactersStart1
+=+=+=+=+=+=+=+=+
/ | | | | | | | /
+-+=+=+=+=+=+=+-+
+B|E|L|E|I|V|E| /
/-+-+-+-+-+-+-+-+
/ / / / / / / /~/
+=+=+=+=+=+=+=+=+
CharactersEnd1
+=+=+=+=+=+=+=+=+
/ | | | | | | | /
+-+=+=+=+=+=+=+-+
+B|E|L|I|E|V|E| /
/-+-+-+-+-+-+-+-+
/ / / / / / / /~/
+=+=+=+=+=+=+=+=+
CharactersStart2
+=+=+=+=+=+=+=+=+
/ | | | | | | | /
+-+=+=+=+=+=+=+-+
+F|R|E|I|N|D| | /
/-+-+-+-+-+-+-+-+
/ / / / / / / /~/
+=+=+=+=+=+=+=+=+
CharactersEnd2
+=+=+=+=+=+=+=+=+
/ | | | | | | | /
+-+=+=+=+=+=+=+-+
+F|R|I|E|N|D| | /
/-+-+-+-+-+-+-+-+
/ / / / / / / /~/
+=+=+=+=+=+=+=+=+
CharactersStart3
+=+=+=+=+=+=+=+=+
/ | | | | | | | /
+-+=+=+=+=+=+=+-+
+ |T|H|E|I|F| | /
/-+-+-+-+-+-+-+-+
/ / / / / / / /~/
+=+=+=+=+=+=+=+=+
CharactersEnd3
+=+=+=+=+=+=+=+=+
/ | | | | | | | /
+-+=+=+=+=+=+=+-+
+ |T|H|I|E|F| | /
/-+-+-+-+-+-+-+-+
/ / / / / / / /~/
+=+=+=+=+=+=+=+=+
```

## Field Definitions

Each field in the file is required unless otherwise noted and the order the fields are defined is important. None of the fields may have empty values. The header fields include:

- **Name**
  The name shown in the puzzle selection view. This name must be unique across all other puzzle level files in the directory. Maximum Length: 20.
- **Goal**
  The text presented to the user describing the end state that the solution needs to achieve. Maximum Length: 240
- **Rows**
  Integer value indicating the number of rows in the grid. Minimum: 1 / Maximum: 12
- **Columns**
  Integer value indicating the number of columns in the grid. Minimum: 3 / Maximum: 20
- **Colored**
  Indication as to whether the puzzle supports coloring the background of cells. Acceptable values: Yes or No
- **Hint**
  Dialog text providing hints to the user when requested. This is an optional field. As many as 5 Hint lines may be provided. See below for Dialog format.
- **Intro**
  Dialog text sequence presented when the level is first opened by the user. This is an optional field. As many as 8 Intro lines may be provided. See below for Dialog format.
- **Outro**
  Dialog text sequence presented when the level is first solved by the user. This is an optional field. As many as 4 Outro lines may be provided. See below for Dialog format.
- **Grids**
  Integer value indicating the number of grids the user will be required to solve with their solution. The number provided here will dictate the grid data following this line. Minimum: 1 / Maximum: 9

Fields requiring a dialog format need to be formatted as follows:

```
[Character Name],[Dialog Text]
```

The total combined length of this field cannot exceed 200. Accepted character names are: Anne, Lawrence, Scott, Phil, Betsy, Evan. If you would like to refer to the user, _PlayerName_ will be replaced in the dialog with the name the player gave at the start of the game.

Following the Grids number, the grids themselves must be defined. In a non-color enabled level you must provide a CharactersStart grid and CharactersEnd followed by the index number (counting from 1) for each grid based on the Grids number you provided.

A 2x3 grid might look like:

```
+=+=+=+
/ | | /
+-+=+-+
/ | / /
+=+=+=+
```

Horizontal cell separations are marked with = for walls and - for non-walls.
Vertical cell separations are marked with / for walls and | for non-walls.
Note that all outside border walls are required. The game will assume them, even if you change the values. The + characters simply provide proper spacing in the definition file.

Placing a letter, number, or punctuation character in the cells indicates that a letter tile should be placed in the cell with the given value. The ~ character is a special character. In CharactersStart definitions it indicates that a Trash can should be placed in that cell. In the CharactersEnd definition it indicates that any character is allowed in that position in the end state.

Color enabled grids must also provide grid definitions for the initial and end colors. These should be done in the following order: CharactersStart#, ColorsStart#, CharactersEnd#, ColorsEnd#.

Characters placed in the cells of these grid definitions represent the starting cell background colors and the puzzle end desired cell background colors. The colors are represented as: Black (B), White (W), Red (R), Blue (E), Green (G), Yellow (Y), Orange

(O), Brown (N), Purple (P). Again the ~ may be used to represent any color in the ColorsEnd definition.

## Appendix III. Puzzle Solution States

The content in the linked document contains some **Spoilers**.

A walkthrough of each level is not provided. There are many ways to solve each puzzle. However, if you find that you are having trouble understanding the text of the Goal of a particular level, you can refer to the follow images to hopefully better understand the intent of the level.

The Orientation puzzles are not shown in the document as there is a walkthrough for those puzzles in Appendix I in this document.

Link to Puzzle Solution States