16

LESSON NAME:

# Relay Programming

Lesson time: 45–60 Minutes : 5 minutes

## Main Goal: Emphasize the importance of checking your work and writing programs in proper sequence.

### OVERVIEW:

This lesson will take an idea from a previous activity (4. Graph Paper Programming) and use those skills to emphasize the importance of completing programs that operate in sequence and frequently checking programs for "bugs."

### OBJECTIVE:

Students will—

- Learn to check their work as well as the work of others
- Think about sequence
- Practice imagining expected outcomes
- Practice completing "thinking tasks" under pressure

### MATERIALS:

- Sample Drawings/Algorithms Kit from lesson 4
- Programming Instructions Card
- Large grid graph paper
- Blank notecards or pieces of paper
- Markers, pens, or pencils (two or three colors)

### PREPARATION:

This works best if the class has already learned the details of Graph Paper Programming.

Print out a Drawings/Algorithms Kit for each group.

Print Programming Instructions Card for each group.

Print several drawing grids based on which version of the exercise you do.

### VOCABULARY:

***Bugs***—Problems with your code

***Debugging***—Fixing problems in your code

***Sequence***—The order in which things are done

## REVIEW:

This review segment is intended to get the class thinking back to the last lesson. If you are covering these activities out of order, please substitute your own review subjects here.

### Class Participation Questions:

- What did we do in our last lesson?
- Do you remember what a parameter is?
- Is a parameter also a variable? Why or why not?

**This lesson emphasizes the importance of frequently checking your programs for "bugs."**

## INTRODUCE:

The activity is the key here. Most of the skills needed to play the game in this lesson were explained in lesson 4. The learning experience in this activity comes from the way those skills are put to the test.

Computer scientists are always facing deadlines. The tighter the time crunch, the more tempted a programmer might be to skip important quality checking steps, or push forward without reviewing what has already been done. To simulate the pressure of working in these situations, this lesson is structured as a team-based relay.

Break your class into groups of 4 to 6 and line them in relay queues on one side of the room (playing outdoors allows for more distance/speed/excitement). On the other side of the room (or yard) place one of the graph drawings across from each relay queue. Put a blank notecard or piece of paper very near each image.

The rules are simple. Each team sends over the first student in line to look at the graph paper image and draw the first programming symbol on the blank piece of paper nearby. The student then returns to the queue and touches the next student's hand. The next student then goes up to the papers, looks at the image, reviews the programming of the previous students, and adds a symbol. If a student finds a bug in the group's program, the student should use their turn to fix the already written code instead of adding another symbol. This process repeats until the group is confident that they've programmed the entire image correctly. The speed and energy of the game depends on whether it is being played outdoors or inside a classroom. If space is limited, you can require each student to walk, or even pass the papers around as students sit at their desks. The running version is perhaps more impressive, since the students have less time to process and communicate during transitions.

A winner is declared when the entire team believes they are done, and the teacher checks the validity of the algorithm in recreating the original drawing. This game can be played again with multiple drawings or multiple adjustments.

When the game is over, gather the students and ask them about what they learned.

- **Was it easy to create perfect code when you were working so quickly?**
- **How easy/hard was it to read the code that the group had already written?**
- **Did you find any bugs? How did you know they were bugs?**
- **Was is more simple or more complicated to have several people involved in the creation at different times?**
- **Are there any tricks that you can think of to make the work easier for the person who follows you?**
- **What do you wish the person before you would have done to help you be faster? Or what did they do to help you be faster/more accurate?**

## ADJUSTMENTS:

### GRADE:

**K-3:** You could have very young students try to recreate a simple shape out of large building blocks. In a hurried atmosphere, this may be a more manageable task. It is less about the "algorithm" but still gets all important ideas across.

**4-6:** The activity should run more or less as written.

**7-8:** Add more complexity as you go with larger drawings or additional colors. Decide whether or not you want to allow the use of the "functions" that you created in lesson 4.

### SPEED:

**Fast:** The relay race described is chaotic and hilarious. Give it a shot if you have wide open spaces.

**Medium:** This is good for a classroom environment. Have students walk across the room. You can even add extra requirements that distract the students while trying to complete their task, like walking backward, requiring at least one hand on a desk at all times, or having to read the whole program aloud before they can begin writing.

**Slower:** Have students remain seated (in rows, circles, or small groups) and pass the image and notecard from desk to desk. Don't allow students to provide hints...only the one with the notecard should be allowed to talk.

### DIFFICULTY:

**Harder:** Have two drawings at each station, and make the students determine which one is being coded as they work on the algorithm.

**Easier:** Have the person who just programmed stay with the notecard for one turn to help the person who is coming to do the next step. This adds a continuity that keeps students from getting lost as easily.