Using Formative Assessments to Examine Student Understanding of Middle School Algorithms and Programming Concepts

Satabdi Basu, satabdi.basu@sri.com
Hui Yang, hui.yang@sri.com
Arif Rachmatullah, arif.rachmatuallah@sri.com
Carol Tate, carol.tate@sri.com
SRI
United States

Daisy Rutstein, dazerw@yahoo.com
EdCount LLC
United States

Abstract: The Computer Science Teachers Association (CSTA) K-12 Computer Science Standards identify 'Algorithms and Programming' as a key CS concept across all grade bands that encompasses sub-concepts such as algorithms, decomposition, variables, and control structures. Previous studies have shown that algorithms and programming concepts often pose challenges for novice programmers, and that instruction in these areas is often superficial. We developed formative assessment tasks to investigate middle school students' understanding of four CS standards related to algorithms and programming and collected responses from over 100 students associated with five different teachers. We found that students demonstrated a limited understanding of the standards. These findings contribute to the growing literature on middle school students' understanding of algorithms and programming, and provide insights that can inform CS teacher development, instruction, and curriculum design.

Keywords: Formative assessment, algorithms and programming, middle school, variables, control structures, decomposition

Introduction

The demand for computer science (CS) learning opportunities in K-12 is rapidly increasing as it becomes clear to policy makers, educators, and parents that an understanding of computing is essential to success in a technology and automation-rich society. The Computer Science Teachers Association (CSTA) K-12 CS Standards identify 'Algorithms and Programming' as a key CS concept across all grade bands. This broad concept strand includes standards for concepts related to algorithms, variables, control structures, decomposition, and procedures. A deep conceptual understanding of algorithms—defined as an ordered set of clear and specific steps aimed at solving a problem or achieving a desired outcome—is crucial, not only for coding, but also for problem-solving in various domains. Proficiency with algorithms is a core element of algorithmic thinking, a vital skill within the broader computational thinking framework (ISTE, 2016). Similarly, an understanding of programming concepts like variables and control structures prepares students for advanced CS topics, helping them build algorithmic literacy essential for designing and creating computing systems (Kafai & Burke, 2015; Ng et al., 2023; Wing, 2006).

Previous studies have shown that algorithms and programming concepts pose challenges for novice programmers, and that instruction on these concepts is often superficial. Insights into students' understanding of these concepts are essential to help teachers optimize their instruction. In addition to teachers' content knowledge, a deep understanding of how K-12 students think about algorithms and programming concepts would allow teachers to identify and address student challenges and advance student understanding. Formative assessment tasks that intentionally target individual aspects of different programming concepts can reveal useful information about student understanding and specific challenges on each of those aspects. The ability to use such formative assessments to measure and support student progress on CS concepts is articulated as part of the CS teacher standards (CSTA, 2020). However, there is currently limited literature on K-12 students' conceptualization of and challenges with algorithms and programming concepts, and many CS teachers report feeling underprepared to use formative assessments to monitor student learning (Banilower & Craven, 2020; Gordon & Heck, 2019).

In this paper, we describe formative assessments we developed to investigate middle school students' understanding of four middle school CS standards related to algorithms, variables, combined control structures, and decomposition. We analyzed responses from over 100 students across five teachers to evaluate their grasp of the concepts underlying these four standards. We believe the findings from this study will contribute to the limited literature on middle school students' understanding of algorithms and programming, providing insights that can inform more effective CS instruction.

Literature Review

Previous studies have shown that algorithms are typically introduced in K-12 education to communicate the idea that computers execute instructions exactly as written, rather than as intended (Basu et al., 2023a). However, these introductory activities often do not fully expose students to other important aspects of algorithms, such as algorithm representation (using flowcharts and pseudocode), interpreting and predicting algorithm behavior, algorithm comparison, testing algorithms with a wide variety of inputs, and debugging algorithms. As a result, much remains unexplored regarding students' understanding of algorithms and their ability to use them effectively. Recent research on assessing K-12 students' algorithmic thinking skills have typically been situated within broader computational thinking (CT) assessments (e.g., Basu et al., 2023b) and hence do not cover all aspects of the 'algorithms' concept. These research studies have noted middle school students' challenges with devising algorithms to solve real-world problems (e.g., Wong & Jiang, 2018) and comparing algorithms when the comparison is based on multiple criteria (Basu et al., 2023b). Some research on high school students' understanding of algorithms has revealed challenges with using flowcharts to create, call, and manage different sub-algorithms (Rahimi et al., 2018), as well as misconceptions related to the efficiency of algorithms, thinking that fewer lines of code and fewer variables characterize algorithm efficiency (Gal-Ezer & Zur, 2004).

Decomposition entails recognizing the elements of a complex problem and breaking them down into sub-problems that can be more easily solved. It is a divide-and-conquer technique that is fundamental to computational thinking and problem solving and applies both within and outside of CS. Hallmarks of efficient decomposition and common misconceptions are underspecified in the literature. Rich and colleagues (2019) have proposed a framework for measuring decomposition that includes categorizing and identifying elements of a problem, identifying relationships between the elements, employing strategies to execute a decomposition, and evaluating the utility of the strategy.

In computer programming, variables provide a way to label and store data for use later in the program. Regarding variables, prior studies found that middle school students exhibited a belief that a variable in CS is the same as a variable in mathematics, which caused them to think of single-letter variable names and not recognize when variables were updated (e.g., score=score+1) (Buffum et al., 2018). Sirkiä & Sorva (2012) found that novice programmers tend to interpret assignment statements as equations. Control structures, such as loops and conditionals that enable modification of the default sequential execution flow of programs, are another fundamental programming construct that all students need to learn from a young age (CSTA 2017). In middle school (grades 6-8), the focus shifts to combined control structures, such as loops within loops (nested loops), conditionals within conditionals (nested conditionals), and conditions combined using Boolean operators (compound conditionals). Prior research has shown that middle school students who understand combined control structures are better prepared to handle advanced CS topics and develop algorithmic literacy essential to designing and creating computing systems (Kafai et al., 2015; Ng et al., 2023; Wing, 2006). However, previous studies have also shown that novice programmers struggle as they progress from simple to combined control structures. When implementing loops, for example, students who do not understand the relationship between inner and outer loops might interpret the loops as sequential rather than nested (PD4CS, n.d.). Students also tend to find nested conditionals challenging, especially choosing when to use nested conditionals versus simple conditionals, as well as not recognizing redundancy in conditional statements (Rich et al., 2017). Overall, although researchers have started to explore students' challenges with key algorithms and programming concepts, there is still a need for research on middle school students' understanding of these concepts, using a variety of programming representations.

Methods

This work is part of a larger project in which we unpacked five middle school (grades 6-8) CS standards into granular learning targets (LTs) and developed 75 formative assessment tasks and rubrics aligned with these LTs to help teachers better understand their students' proficiency with CS concepts. The assessments were tested and refined through expert reviews, student cognitive interviews, and classroom studies. In this paper, we report on student responses to 17 formative assessment tasks collected from five middle school CS teachers.

The five teachers who participated in this study taught CS in middle schools in different parts of the U.S. and had diverse backgrounds. Dina (all teacher names are pseudonyms), a white female 6th-grade teacher, had 23 years of teaching experience, including 6 years in CS. Similarly, Rose, a white female, had 26 years of experience teaching secondary students, including 3 years in CS. Grace, also a white female, had been teaching CS to upper elementary and middle school students for 4 years. On the other hand, Brenda, a Hispanic female, had been teaching CS for 12 years, and Amber, a white female teacher, had over 20 years of CS teaching experience, though it was her first year teaching middle school when she administered our formative assessment tasks.

All teachers administered the tasks toward the end of the semester after teaching the relevant concepts. The number and types of tasks each teacher administered and the number of students who completed them varied. Table 1 describes the tasks each teacher used, the associated concepts, and the number of students who completed them.

Table 1. Tasks and Data Sources Associated with different Concepts and Learning Targets

Concept	Granular Learning Target (LT)	Task#	Data Source
Algorithms (2-AP-10 standard)	Knowledge of algorithms	A1	Dina: 24 students Grace: 109 students
	Trace an algorithm and describe its output for given inputs	A2	Grace: 109 students
	Compare algorithms based on given criteria	A3	Dina: 24 students Grace: 109 students
Variables (2-AP-11 standard)	Choose meaningful yet concise variable names.	V1	Amber: 15 students
	Initialize variables accurately in the right places in code	V2	Rose: 89 students Brenda: 8 students
	For complex variables (e.g. lists), assign values to variables, and perform operations on values of variables	V3	Brenda: 7 students
	Describe how a variable value changes within a loop or a conditional	V4 V5	Brenda: 5 students
Combined Control Structures (2-AP-12 standard)	Create/debug a nested loop for a given scenario	C1	Amber: 25 students
	Create/debug nested conditional statements for a given scenario	C2	Rose: 87 students
Decomposition (2-AP-13 standard)	Break up a problem into meaningful sub-problems when programming a story; Identify dependencies between solutions to sub-problems	D1	Amber: 13 students
	Break up a problem into meaningful sub-problems when creating an app	D2	Amber: 15 students
	Evaluate whether a decomposition is sufficient to address a bigger task completely and coherently.	D3	Dina: 24 students
	Solve a problem using solutions to existing sub-problems	D4	Amber: 23 students

Task formats included multiple choice questions, open-ended responses, Parsons problems, etc. For each task, we developed a rubric outlining possible student responses and their alignment with the LTs measured by the tasks. Teachers used these rubrics to evaluate student responses, gaining insights into students' understanding or difficulties across the four concepts. Additionally, two researchers independently scored each student response using the same rubric the teachers used. For open-ended responses, intercoder agreement ranged from 40% to 100%. The initially lower agreement occurred on few tasks with low responses (less than 15) and those open to interpretation, such as the decomposition task D2 (40% agreement) and the variable task V7 (43% agreement). To address discrepancies, a third researcher participated in a consensus meeting, discussing and resolving coding differences. Once coding was finalized, we conducted a descriptive analysis to explore middle school students' knowledge and challenges related to the four target Algorithm and Programming concepts for the middle grades.

Results

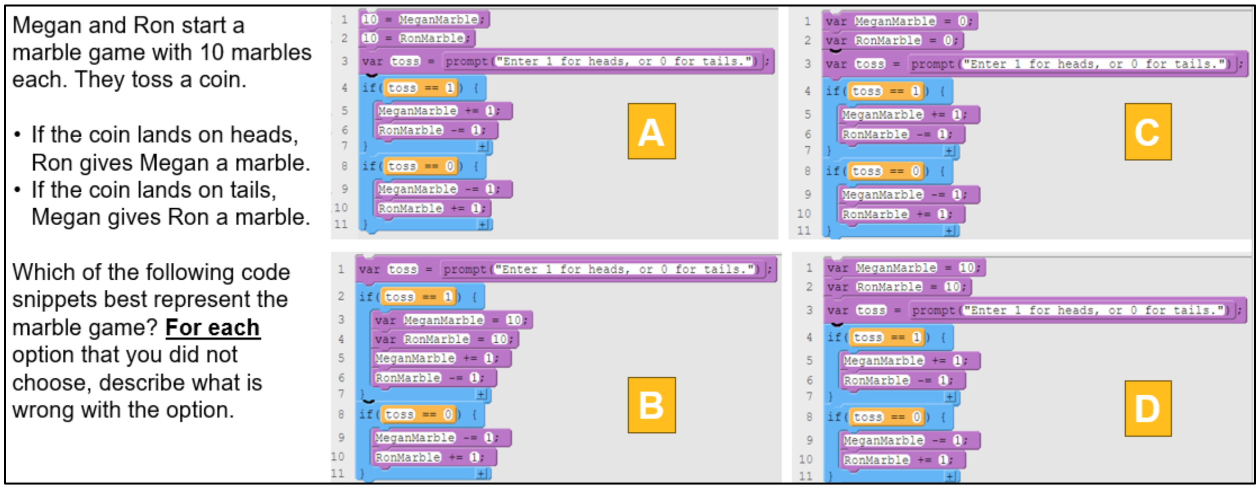
Below, we summarize our findings on students' understanding of and challenges with the four concepts.

Algorithms

Understanding what an algorithm is. Task A1 assessed students' understanding of algorithms through a combination of multiple-choice and open-ended questions. Students were asked to identify algorithms for making hot chocolate and explain their selections. They were given four options: an unordered procedure list that does not fulfill the goal of making hot chocolate, a flowchart for making hot chocolate, a static diagram depicting needed objects, and a procedure list for making hot chocolate. Of the 104 students who responded to A1 (29 students left blank responses), most (*n* = 98) were able to recognize at least one algorithm for making hot chocolate, and 85 students recognized at least one algorithm without choosing an additional incorrect option. Of these 85 students, 46 identified one algorithm while 39 students identified both. Among students who recognized only one algorithm, most could recognize only the procedure form (33 out of 46 students), while only 13 students could recognize the flowchart form. Among the other students, four selected the static diagram depicting objects. This indicates some students' challenges with understanding that an algorithm is a process or ordered set of steps. When explaining why their selected options were algorithms for making hot chocolate, only 2 out of the 104 students who responded provided an explanation that included all three characteristics of algorithms: clear and specific instructions, logical ordering of steps, and ending in a goal state. Fifty-one student responses were unable to recognize that an algorithm needs to include a set of clear and specific instructions, 35 students struggled to see that the algorithm must achieve its goal, and similarly, 35 students had difficulties understanding that an algorithm requires instructions to be ordered in a logical sequence. Many students demonstrated a combination of these challenges. Additionally, some explanations (*n* = 19) did not include any characteristics of an algorithm (e.g., "it is an algorithm for making hot chocolate is because it is the correct way to make it.").

Tracing a flowchart and describing its behavior. Task A2 asked students to predict the output of a flowchart given a specific input and to identify the input needed to achieve a desired output. We found that less than half of the students (45 out of 109 of Grace's students) were able to correctly answer A2 and predict the correct output. Many students (23 of 109) selected an option that indicated challenges with interpreting two conditionals in a row involving a "greater than" operator. Some student responses (16 out of 109) indicated their inability to interpret a conditional statement with a "less than" operator. Additionally, 19 students thought there wasn't enough information to determine the flowchart output, perhaps due to an ability to interpret the comparison statements or to recognize what the variable "x" denoted in the algorithm. Students found the second part of the task harder where they had to determine which input would result in a given output; only 37 of 109 students could answer this part. Overall, student responses to A2 demonstrate that students find it easier to predict the output given an input, compared to predicting the input that generates a given output. Also, students' ability to trace flowcharts is dependent on the CS concepts included in the flowcharts, for example variables and logical operators, in this case.

Interpreting and comparing algorithms. Task A3 focused on comparing algorithms based on multiple criteria. Through four multiple-choice questions, students compared three algorithms for moving a robot through a grid based on criteria such as cost and time. We found that 66 out of 133 students were able to identify the algorithms that would take the robot from the "Start" to the "Finish" square. The remaining 67 students either struggled with understanding what the individual steps meant or had trouble following the sequence of the steps. In the second part of the task, 57 of 133 students were able to compare the algorithms based on a given time criterion to determine which algorithm gets the robot to the Finish square in the least amount of time. For the third and fourth parts of the task, only Grace's students (*n* = 109) worked on these parts. Fewer than half of the students (44 of 109) were able to compare the algorithms based on a given cost criterion for the third part of the task. Finally, for the fourth part, where students had to compare algorithms based on multiple criteria related to time and cost (the robot must complete the route within 10 minutes and incur a cost of $5 or less), only 29 of 109 students were able to respond accurately. These findings align with existing literature about students' ability to compare algorithms and how students find the process more challenging as the number of comparison criteria increases (Basu et al., 2023b).


Megan and Ron start a marble game with 10 marbles each. They toss a coin.

- If the coin lands on heads, Ron gives Megan a marble.
- If the coin lands on tails, Megan gives Ron a marble.

Which of the following code snippets best represent the marble game? **For each** option that you did not choose, describe what is wrong with the option.

A
```
10 = MeganMarble;
10 = RonMarble;
var toss = prompt("Enter 1 for heads, or 0 for tails.");
if(toss == 1) {
  MeganMarble += 1;
  RonMarble -= 1;
}
if(toss == 0) {
  MeganMarble -= 1;
  RonMarble += 1;
}
```

C
```
var MeganMarble = 0;
var RonMarble = 0;
var toss = prompt("Enter 1 for heads, or 0 for tails.");
if(toss == 1) {
  MeganMarble += 1;
  RonMarble -= 1;
}
if(toss == 0) {
  MeganMarble -= 1;
  RonMarble += 1;
}
```

B
```
var toss = prompt("Enter 1 for heads, or 0 for tails.");
if(toss == 1) {
  var MeganMarble = 10;
  var RonMarble = 10;
  MeganMarble += 1;
  RonMarble -= 1;
}
if(toss == 0) {
  MeganMarble -= 1;
  RonMarble += 1;
}
```

D
```
var MeganMarble = 10;
var RonMarble = 10;
var toss = prompt("Enter 1 for heads, or 0 for tails.");
if(toss == 1) {
  MeganMarble += 1;
  RonMarble -= 1;
}
if(toss == 0) {
  MeganMarble -= 1;
  RonMarble += 1;
}
```

Figure 1. Task V2 assessing student understanding of variable initialization

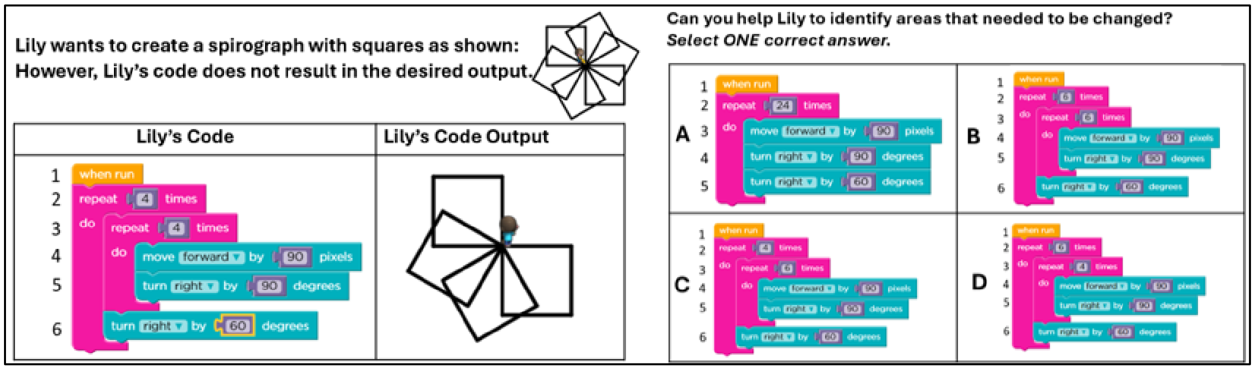
Variables

Choosing meaningful yet concise variable names. In task V1, students were asked to evaluate variable names in a given program about a ball's movement, provide reasoning for their decisions, and suggest new variable names when they considered the existing ones inappropriate. Only two of 15 students from Rose's class answered this correctly. Only 7 students realized that "Up" was an ambiguous variable name and not descriptive enough; others thought it was a good name because it was short. However, most students realized "BallColor" was a good variable name and that "RandomDirectiontheBallMovesAtAnyGivenTime" was too long a variable name. Even when students could evaluate variable names correctly, they were generally unable to suggest better alternate names. This suggests a need for explicitly teaching students about variable naming mechanisms, something often skipped in middle school CS curricula.

Initializing variables accurately in the right places in code. In task V2 (see Figure 1), students were given a description of a marble game and four code snippets, and asked to describe why each code snippet represented the game or did not. One option initialized the marbles to an incorrect value, one initialized them in an inappropriate place in the code, and one option switched the variable and values in the assignment statement. Overall, only 16 out of 97 students correctly identified the appropriate code and explained why it was correct while the other three options were not. Most students (61 of 97) struggled to recognize that the variable name should appear before the = sign. About half the students (46 of 97) had difficulty matching the marble game description to the code and missed the fact that variables were initialized to 0 instead of 10. Similarly, 45 students did not understand that initialization should occur outside an 'if' statement. Even those students who identified the errors could not always describe them.

Performing operations on values of complex variables. In task V3, students were asked to work with a list of ice cream flavors in a program. Though we have limited data from a few students on this task, it was evident that students found the concept of complex variables challenging. Only 2 out of 7 students correctly understood what the len/length function is and how it stores the number of elements in a list. Two other students were the only ones who answered the second part of V3 correctly, where students were asked to debug a code snippet that referenced certain elements of the list using incorrect indices.

Describing how a variable value changes within a loop or a conditional. In this task, Brenda used two different tasks, each with 5 students, to understand student proficiency with this aspect of the variables concept. We found that variables, when combined with control structures like loops and conditionals, become more challenging for students. For task V4 where students had to predict the value of a 'score' variable which was dependent on the value of another variable, only 1 out of 5 students could answer the task correctly. For task V5, students were asked to identify the output of code containing a variable that updates within a loop but is also repeatedly re-initialized within the loop. None of Brenda's five students who completed this task could correctly identify the output. Four of them struggled with understanding how indexing works in a loop and may have made an "off by 1" error. Two students didn't recognize that the variable is re-initialized and reset to 0 repeatedly within the loop, and one student failed to recognize that the variable is incremented before it is printed.


Lily wants to create a spirograph with squares as shown:
However, Lily's code does not result in the desired output.

Lily's Code

```
when run
repeat 4 times
do  repeat 4 times
    do  move forward by 90 pixels
        turn right by 90 degrees
    turn right by 60 degrees
```

Lily's Code Output

Can you help Lily to identify areas that needed to be changed?
Select ONE correct answer.

A

```
when run
repeat 24 times
do  move forward by 90 pixels
    turn right by 90 degrees
    turn right by 60 degrees
```

B

```
when run
repeat 6 times
do  repeat 6 times
    do  move forward by 90 pixels
        turn right by 90 degrees
    turn right by 60 degrees
```

C

```
when run
repeat 4 times
do  repeat 6 times
    do  move forward by 90 pixels
        turn right by 90 degrees
    turn right by 60 degrees
```

D

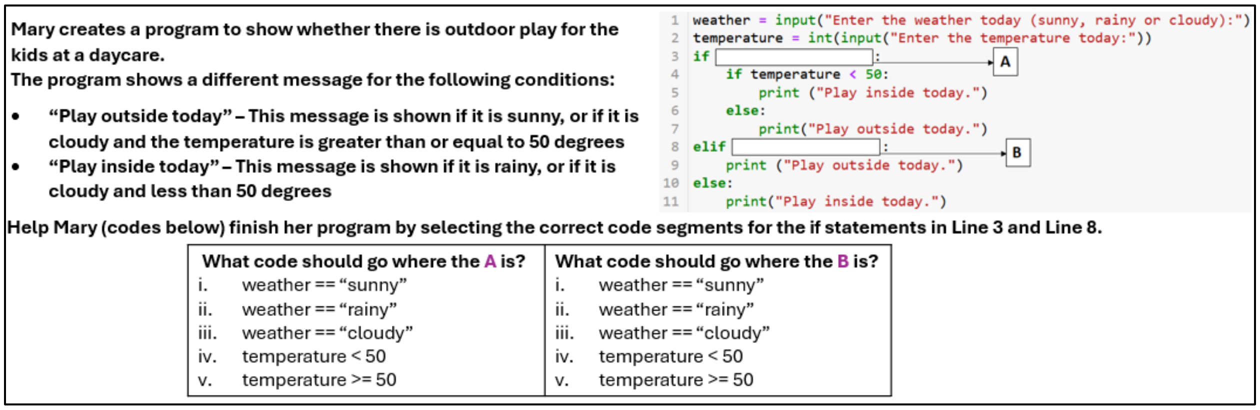
```
when run
repeat 6 times
do  repeat 4 times
    do  move forward by 90 pixels
        turn right by 90 degrees
    turn right by 60 degrees
```

Figure 2. Task C1 checking understanding of nested loops

Combined Control Structures

Understanding nested loops. For task C1 (Figure 2), we found that less than half of the students in Dina's class could answer this question correctly (option D) by recognizing how many times the inner and outer loop should repeat to draw the spirograph. All students realized that nested loops would be required to draw the given image and no student selected Option A (simple loop). But only 11 of 25 students recognized that there are 6 squares in the spirograph, and the outer loop should be set to 6 to create those 6 squares. The majority of students still had **challenges recognizing the number of repeats for the inner and outer loops**. This reflected that they might not know which part of the spirograph each loop is responsible for drawing. For example, 8 students said that the outer and inner loop should both repeat 6 times (option B) perhaps not realizing that the inner loop should draw a square and does not require 6 repeats. Five other students chose C as the correct answer, possibly because they switched the number of repeats for the inner and outer loops without realizing which part of the spirograph each loop is responsible for drawing.

Understanding nested conditionals. For Task C2 (Figure 3), a very small group of Amber's students (18 of 87; 8 sixth graders and 10 seventh graders) could correctly complete both the missing conditions in the nested conditional statements. About half of the student (41 of 87) were able to correctly complete Part B, while 27 students were able to correctly complete Part A. Student challenges included **inability to follow the flow of control in code with nested conditionals** and **interpreting nested conditionals as if they were combined using the OR operator instead of the AND operator**. For example, for Part A (the correct answer is weather == "cloudy"), 30 of the 87 students who selected the (weather == "rainy") option might have difficulty knowing when and how to nest conditionals and might interpret a nested conditional as combined via an OR instead of an AND operator. For Part B (the correct answer is weather == "sunny"), 24 of the 87 students selected (temperature >= 50), perhaps not realizing that the temperature condition always needs to be combined with the weather condition. For Part B, 10 students chose (weather == "rainy"), indicating challenges with simple if-else statements.


Mary creates a program to show whether there is outdoor play for the kids at a daycare.
The program shows a different message for the following conditions:

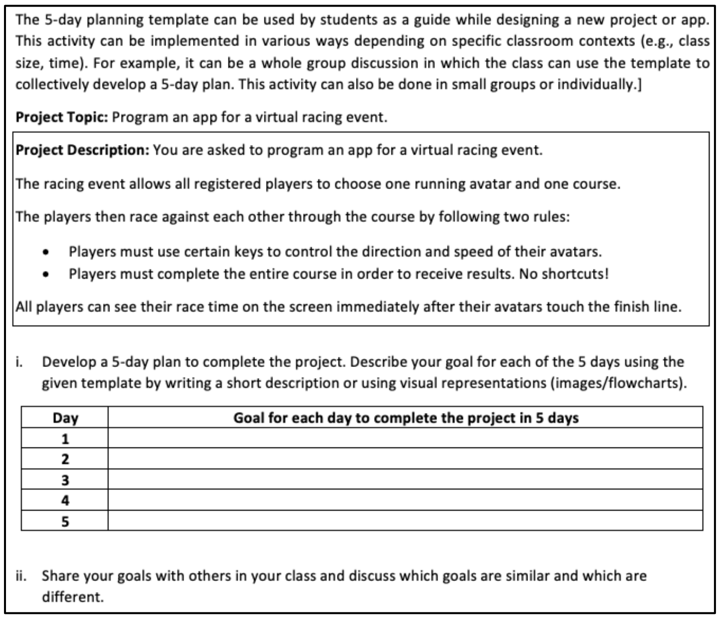
- **"Play outside today" – This message is shown if it is sunny, or if it is cloudy and the temperature is greater than or equal to 50 degrees**
- **"Play inside today" – This message is shown if it is rainy, or if it is cloudy and less than 50 degrees**

```
1  weather = input("Enter the weather today (sunny, rainy or cloudy):")
2  temperature = int(input("Enter the temperature today:"))
3  if [                ]:  ──▶ A
4      if temperature < 50:
5          print ("Play inside today.")
6      else:
7          print("Play outside today.")
8  elif [                ]:  ──▶ B
9      print ("Play outside today.")
10 else:
11     print("Play inside today.")
```

Help Mary (codes below) finish her program by selecting the correct code segments for the if statements in Line 3 and Line 8.

What code should go where the A is?	What code should go where the B is?
i. weather == "sunny"	i. weather == "sunny"
ii. weather == "rainy"	ii. weather == "rainy"
iii. weather == "cloudy"	iii. weather == "cloudy"
iv. temperature < 50	iv. temperature < 50
v. temperature >= 50	v. temperature >= 50

Figure 3. Task C2 measuring understanding of nested conditionals


The 5-day planning template can be used by students as a guide while designing a new project or app. This activity can be implemented in various ways depending on specific classroom contexts (e.g., class size, time). For example, it can be a whole group discussion in which the class can use the template to collectively develop a 5-day plan. This activity can also be done in small groups or individually.]

Project Topic: Program an app for a virtual racing event.

Project Description: You are asked to program an app for a virtual racing event.

The racing event allows all registered players to choose one running avatar and one course.

The players then race against each other through the course by following two rules:

- Players must use certain keys to control the direction and speed of their avatars.
- Players must complete the entire course in order to receive results. No shortcuts!

All players can see their race time on the screen immediately after their avatars touch the finish line.

i. Develop a 5-day plan to complete the project. Describe your goal for each of the 5 days using the given template by writing a short description or using visual representations (images/flowcharts).

Day	Goal for each day to complete the project in 5 days
1	
2	
3	
4	
5	

ii. Share your goals with others in your class and discuss which goals are similar and which are different.

Figure 4. Task D2 – An example task for Decomposition

Decomposition

Breaking up a problem into meaningful sub-problems. Task D1 asked students to decide how to program the Turtle and the Hare story by thinking of what the different scenes should be and what characters and backgrounds they would have. Then, it asked students to divide the programming task among three different group members and identify what information should be shared among them. We found that half of the students (7 out of 15) were able to identify the characters in the story, while the remaining students struggled with following the storyline and some decided to embellish the storyline on their own. When asked to decompose the story into distinct scenes, only one student provided a complete and accurate breakdown. Four other students had complete breakdowns, but they included inaccuracies. Some students outlined scenes that did not cover the whole story (incomplete breakdown), some added irrelevant details, while others described scenes where there was a mismatch between the characters and the backgrounds. For the part about decomposing programming tasks among a group of three classmates, six students provided a decomposition that was complete and would cover all the aspects of programming the story. However, only one student effectively distributed the work among the group members. The others either assigned overlapping tasks to different group members or created an uneven distribution of work among group members. Finally, when asked to identify at least two pieces of information that group members should share, only 3 of 15 students responded appropriately, describing relevant information like variable names and features of story characters.

In the D2 task (Figure 4), students were evaluated on similar aspects of the decomposition standard, but in the context of developing an app instead of programming a story. Students were asked to create a 5-day plan with intermediate goals and milestones to develop a racing game based on a given app description. Each student's plan was evaluated on three criteria: completeness, logical ordering, and efficiency. Five of the 15 students who responded to this task successfully met all three criteria, meaning their plans covered all aspects of the game, included a logical sequence of daily tasks (where each day's plan did not rely on the completion of future tasks), and distributed work somewhat evenly across days without overlapping tasks. All students' decomposition covered at least half of the game's requirements, and six students' day-by-day plans followed a logical ordering. Students struggled the most in decomposing tasks efficiently (9 of 15 students) – they either outlined an imbalanced distribution of work across the five days or defined overlapping tasks or both.

Evaluating whether a decomposition is complete. In task D3, students were told the Little Red Riding Hood story with examples of characters, actions and backgrounds. Then, students were provided with a decomposition of the story into scenes and asked to determine whether the decomposition fully covered the entire story or missed certain scenes. We found that out of 24 students, none correctly identified whether the decomposition programmed the entire story or left out certain elements. Nearly all students (23) faced challenges in recognizing the roles of multiple characters within single scenes and/or spotting missing elements in scenes. Additionally, few students (4) could not recognize certain parts of the story were already programmed and pointed them out as missing.

Solving a problem using solutions to existing sub-problems. In Task D4, students were asked to design a webpage by reusing a given webpage which is similar in some but not all aspects. For part 1 of this task where students had to identify two elements that should be the same in the new webpage and the existing webpage, only four students answered correctly. Many students (11 of 23) struggled to separate style from content, and identify how the style or content could be reused for different parts of the webpage. For part 2, students needed to identify two aspects that should change between the webpages. Similar to part 1, only five students responded correctly, demonstrating students' challenges with reusing solutions to subproblems to solve a given problem.

Discussion

In this paper, we describe formative assessment tasks that target specific aspects of important CS concepts covered by the algorithms and programming standards for middle school. The results reveal useful information about how middle school students understand each of these concepts, as well as the challenges they face with the concepts. Although our findings are limited in scope, they align with and add to a growing literature on how novices learn CS concepts. While we worked with five different teachers from different parts of the country, they all used popular middle school curricula such as code.org's CS Discoveries, CMU's CS1, and the MyCS Scratch curriculum. These curricula use different programming representations; hence our formative assessments used pseudocode or were customized for each teacher based on the programming representation their students would be familiar with. The student challenges we observed were not specific to a particular teacher or use of a certain curriculum or programming representation. Our findings underscore the need for current CS curricula to further emphasize CS concepts in depth. For example, curricula need to cover the nuanced aspects of working with algorithms and not merely the idea that algorithms need to include clear and specific instructions. Decomposition needs to be covered explicitly in curricula to help students improve on this standard and should include aspects such as completeness and efficiency of decompositions. With control structures, our tasks revealed that many students demonstrated challenges with using combined control structures, including understanding when and how to nest conditionals, and how to distinguish between inner and outer loops in nested loops. We also found that students struggle with complex variables and variables used alongside control structures such as loops and conditionals.

Currently, CS teacher professional development is typically tied to a specific curriculum using a specific programming language. Formative assessment practices and knowledge about the CS standards or CS concepts themselves are often not a focus of teacher professional learning (Basu et al., 2022). Teachers' lack of deep CS knowledge contributes to difficulty with diagnosing the source of students' struggles and determining instructional strategies for helping students who are stuck. Tasks such as those presented in this paper can be a valuable tool for teachers charged with supporting student progress on CS concepts because they yield valuable information to guide instruction. This information is essential for teachers to develop the ability to measure and support student progress on CS concepts as articulated in the CS teacher standards (CSTA, 2020).

This paper highlights the need for topic or content-focused professional development for middle school CS teachers to deepen their understanding of the CS concepts they teach. Content-focused PD has been widely recommended in other STEM education contexts as a successful model, particularly for teachers who lack a strong background in a specific discipline (Loucks-Horsley & Matsumoto, 1999). This is especially relevant for middle school CS teachers, many of whom do not have formal training in CS (Basu et al., 2022; Yadav et al., 2016). In content-focused PD, teachers learn details about the concepts they will teach, as well as how students are likely to learn those concepts (Loucks-Horsley & Matsumoto, 1999). This approach can be followed by practicing the use of formative assessments with sample student responses to identify students' conceptual understanding and challenges. When combined with effective PD practices, such as sustained engagement over time, content-focused PD has consistently been linked to improved teacher knowledge in various disciplines (Desimone, 2009; Hill et al., 2005). However, more research is needed to evaluate the effectiveness of this type of PD on CS teachers' pedagogical content knowledge (PCK) and its impact on students' CS learning. Furthermore, content-focused PD grounded in standards can support a variety of teachers teaching different CS curricula. Teachers can learn the concepts before learning how to apply the concepts using a specific representation (programming language) aligned with their CS curriculum. This adaptability will equip teachers to address diverse teaching needs.

A central goal of CS for All is to create pathways into CS for students from groups that are currently underrepresented in the field. Motivating students to see themselves as potential computer scientists will require the ability to pitch instruction to a level that will present an appropriate challenge. This ability depends on deep understanding of both the subject matter and how students learn it. In this sense, formative assessments like those discussed in this paper form an important tool for driving equitable access to CS learning. Further, content standards are one tool to ensure that all students receive high quality, cognitively demanding instruction. The results we report in this paper indicate that current curricula and classroom practice may be disconnected from the current standards, making both standards and curricula worth revisiting for better alignment.

References

Banilower, E., & Craven, L. (2020, February). Factors associated with high-quality computer science instruction: Data from a nationally representative sample of high school teachers. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 360-365).

Basu, S., Rutstein, D., Tate, C., Rachmatullah, A., Yang, H., & Ortiz, C. (2023a). Exploring Middle School Students' Understanding of Algorithms Using Standards-Aligned Formative Assessments: Teacher and Researcher Perspectives. In *Proceedings of the 17th International Conference of the Learning Sciences-ICLS 2023, pp. 114-121*. International Society of the Learning Sciences.

Basu, S., Rutstein, D. W., Xu, Y., Wang, H., & Shear, L. (2023b). A principled approach to designing computational thinking concepts and practices assessments for upper elementary grades. In Assessing Computational Thinking (pp. 57-86). Routledge.

Basu, S., Rutstein, D., Tate, C., Rachmatullah, A., & Yang, H. (2022, February). Standards-Aligned Instructional Supports to Promote Computer Science Teachers' Pedagogical Content Knowledge. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1* (pp. 404-410).

Buffum, P. S., Ying, K. M., Zheng, X., Boyer, K. E., Wiebe, E. N., Mott, B. W., ... & Lester, J. C. (2018, February). Introducing the computer science concept of variables in middle school science classrooms. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (pp. 906-911).

Computer Science Teachers Association. (2017). CSTA K-12 computer science standards. Revised 2017. http://www.csteachers.org/standards

Desimone, L. M. (2009). Improving impact studies of teachers' professional development: Toward better conceptualizations and measures. *Educational Researcher*, *38*(3), 181-199.

Gordon, E. M., & Heck, D. J. (2019). 2018 NSSME+: Status of High School Computer Science.

Hill, H. C., Rowan, B., & Ball, D. L. (2005). Effects of Teachers' Mathematical Knowledge for Teaching on Student Achievement. *American Educational Research Journal, 42*(2), 371–406. https://doi.org/10.3102/00028312042002371

International Society for Technology in Education (ISTE). (2016). ISTE Standards: Students. https://www.iste.org/standards/iste-standards-for-students Accessed: 2024-10-10

Kafai, Y. B., & Burke, Q. (2015). Constructionist gaming: Understanding the benefits of making games for learning. *Educational psychologist*, *50*(4), 313-334.

Loucks-Horsley, S., & Matsumoto, C. (1999). Research on professional development for teachers of mathematics and science: The state of the scene. *School science and mathematics*, *99*(5), 258-271.

Ng, D. T. K., Lee, M., Tan, R. J. Y., Hu, X., Downie, J. S., & Chu, S. K. W. (2023). A review of AI teaching and learning from 2000 to 2020. *Education and Information Technologies*, *28*(7), 8445-8501.

Rich, K. M., Strickland, C., Binkowski, T. A., Moran, C., & Franklin, D. (2017, August). K-8 learning trajectories derived from research literature: Sequence, repetition, conditionals. In *Proceedings of the 2017 ACM conference on international computing education research* (pp. 182-190).PD4CS. (n.d.). Retrieved from http://www.pd4cs.org/. Accessed: October 8, 2024.

Rich, P. J., Egan, G., & Ellsworth, J. (2019, July). A framework for decomposition in computational thinking. In Proceedings of the 2019 ACM conference on innovation and technology in computer science education (pp. 416-421).

Sirkiä, T., & Sorva, J. (2012, November). Exploring programming misconceptions: an analysis of student mistakes in visual program simulation exercises. In *Proceedings of the 12th Koli calling international conference on computing education research* (pp. 19-28).

Wing, J. (2006). Computational Thinking. Communications of the ACM, 49 (3), 33–35.

Wong, G. K., & Jiang, S. (2018, December). Computational thinking education for children: Algorithmic thinking and debugging. In *2018 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)* (pp. 328-334). IEEE.

Yadav, A., Gretter, S., Hambrusch, S., & Sands, P. (2016). Expanding computer science education in schools: understanding teacher experiences and challenges. *Computer science education*, *26*(4), 235-254.

Acknowledgements

This paper is based upon work supported by the National Science Foundation under Grant No. DRL-2010591. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.