Teaching Coral before C++ in a CS1 Course

Joe Michael Allen, University of California, Riverside

Joe Michael Allen is a Ph.D. student in Computer Science at the University of California, Riverside. His current research focuses on finding ways to improve CS education, specifically focusing on introductory programming courses known as CS1. Joe Michael is actively researching the impact of using a many small programs (MSP) teaching approach in CS1 courses. His other interests include educational games for building skills for college-level computer science and mathematics.

Prof. Frank Vahid, University of California, Riverside

Frank Vahid is a Professor of Computer Science and Engineering at the Univ. of California, Riverside. His research interests include embedded systems design, and engineering education. He is a co-founder of zyBooks.com.

Teaching Coral before C++ in a CS1 Course

Abstract

Commercial languages like Python, Java, or C++, have syntactic, semantic, and compiler/interpreter issues that make them less-than-ideal as a CS1 language. The free Coral language, which uses ultra-simple statements, auto-derived flowcharts, and a web-based graphical educational simulator with clear error messages, was developed in 2017 to address such issues. Coral is designed to lead more directly into commercial languages than other educational languages like Scratch or Snap. Dozens of schools use Coral, often as the language in CS0 courses. In this work, we experimented with using Coral in CS1 to ease students into the commercial language C++. For one 80-student CS1 section, the term's first half used Coral to teach input/output, variables, expressions, branches, loops, arrays, and functions, thus focusing on program logic and problem solving rather than syntax and semantic details. The term's second half then retaught those constructs using C++. We found what we'd hoped: the Coral-to-C++ students did equally well on the identical C++ final exam and did equally well in the course. The results suggest that instructors can start a CS1 class with Coral to enable a smooth start and to teach using an educational simulator, without loss in learning outcomes or programming capability. We indicate ideas of how Coral's introduction can be improved, which may yield further improvements.

1. Introduction

CS1 courses are difficult and commonly have high rates of Ds, Fs, and withdrawals [1], [2], [3], [4], [5], [6]. One contributing factor is the set of technical challenges in the first several weeks, including nuances of commercial languages like Python, Java, and C++ [7]. Those languages were designed for professionals, not for learners. For example, Figure 1 shows an early input/output program in a popular Python textbook.

Fig. 1. Python source code for an introductory input/output program.

```
print('Enter wage:', end=' ')
wage = int(input())
wage = wage + 10
print('New wage: ')
print(wage)
```

While learning basic input, assignments, and output, students are also exposed to distracting/confusing details: function calls with parentheses as in print('text'), comma separators in argument lists, a strange end=' ' notation to prevent an output newline, the idea that a function can also return a value as in wage = input(), and types as well as type casting as in int(input()) to convert the input string to an integer. C++ and Java also have syntactic and semantic overhead, perhaps more.

One solution to avoid the overhead on learners of such commercial programming languages is the development of visual programming languages like Alice [8], Scratch [9], and Snap [10]. These languages use a block-based, drag-and-drop coding approach where many programming details are abstracted, allowing instructors and students to focus on desired functionality rather than the intricacies of a programming language. While showing great benefits [11], [12], [13], [14], such languages are primarily intended for students in elementary school, high school, and perhaps CS0 courses.

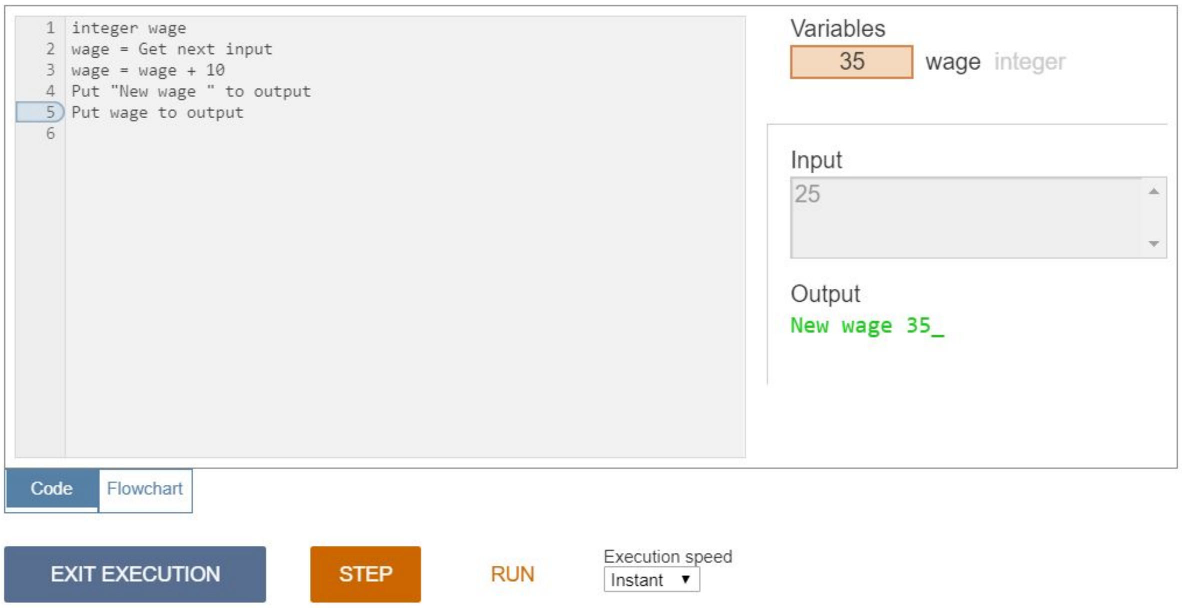
Instead, another solution to reduce language overhead was the development of the Coral programming language in 2017. Coral was created for college students and designed to look like common industry languages, but with ultra-simple syntax. Coral was created by computer scientists with learning and education in-mind from the beginning. The language looks like pseudocode. Below is the Python program from Figure 1, shown in Coral instead.

Fig. 2. Coral source code for the same input/output program shown in Python in Figure 1.

```
integer wage
wage = Get next input
wage = wage + 10
Put "New wage " to output
Put wage to output
```

Though both the Python and Coral code each have 5 lines, the Coral code is free of the above-listed distracting/confusing details. Though looking like pseudocode, Coral is executable, with a free online web-based visual simulator that can show statement-by-statement execution, variables in memory, inputs being consumed, and generated output, as seen in Figure 3.

Fig. 3. Online web-based Coral visual simulator with the previously shown wage example from Figure 2.



```
integer wage
wage = Get next input
wage = wage + 10
Put "New wage " to output
Put wage to output
```

Variables

35	wage	integer

Input

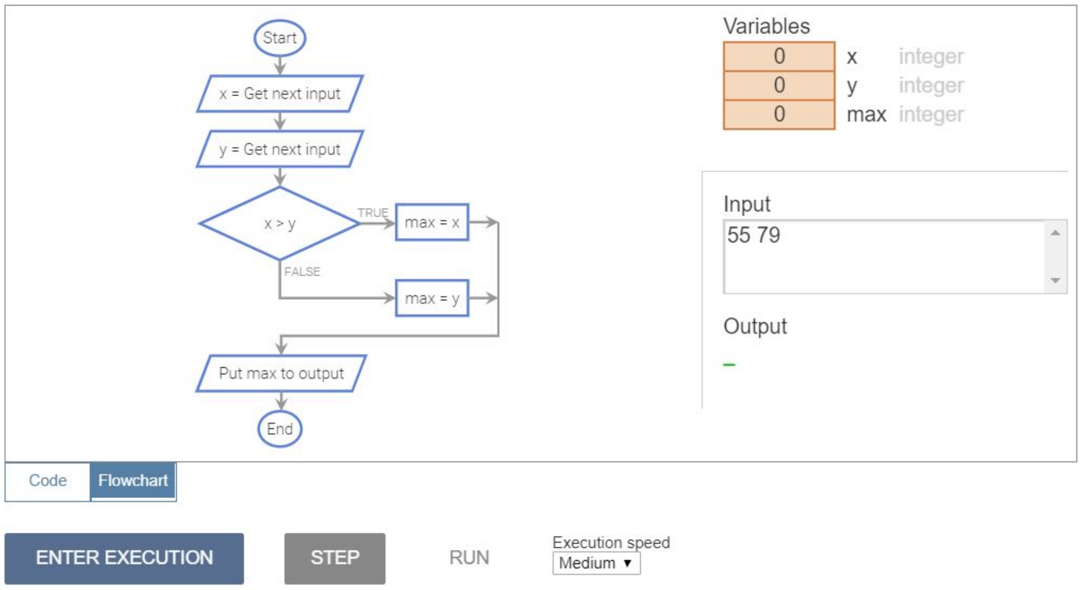
```
25
```

Output

```
New wage 35_
```

Coral has an equivalent flowchart language, with the web-based simulator automatically deriving the flowchart from the Coral textual code. The flowchart is generated to look as close to the code as possible. For example, branch or loop sub-statement graphical nodes appear "indented" just like code. Figure 4 shows a different Coral example, having a branch, displayed in flowchart form.

Fig. 4. Online web-based Coral flowchart visual simulator for a branch example.



Variables

0	x	integer
0	y	integer
0	max	integer

Input

```
55 79
```

Output

```
_
```

Coral has several features that ease learning the language, such as:

- Coral has only 7 kinds of statements.
- Coral only allows exactly one statement per line, and requires 3-space indents for sub-statements.
- Coral only supports integer and float data types, which are sufficient to learn programming fundamentals by processing data.
- Coral requires no surrounding main() or include/use directives.

While simple, Coral is designed to lead learners directly into programming in commercial languages. For example, Table 1 shows how each Coral statement is converted into C++.

Table 1. Coral statements converted into C++ statements.

Type	Coral	C++

Variable declaration	integer numCats
	float wallHeight
	integer array(5) numSequence
	float array(5) wallAreas
		int numCats;
		double wallHeight;
		vector<int> numSequence(5);
		vector<double> wallAreas(5);

Input statement	numCats = Get next input	cin >> numCats;
Output statement	Put numCats to output	cout << numCats;
Assignment statement	numCats = numCats + 1	numCats = numCats + 1;

Branch statements	if numCats > 5
	   Put "too many" to output
	elseif numCats > 1
	   Put "sufficient" to output
	else
	   Put "need more" to output
		if (numCats > 5) {
		   cout << "too many";
		}
		else if (numCats > 1) {
		   cout << "sufficient";
		}
		else {
		   cout << "need more";
		}

While loop statement	while wallHeight <= 0
	   wallHeight = Get next input
		while (wallHeight <= 0) {
		   cin >> wallHeight;
		}

For loop statement	for i = 0; i < 5; i = i + 1
	   Put i to output
		for (i = 0; i < 5; i = i + 1) {
		   cout << i;
		}

And/or/not	(a < b and a < c) or (not(a > d))	(a < b && a < c) || (!(a > d))

The language features listed above eliminate many common errors by beginning programmers, and enable the Coral simulator to provide clear helpful feedback on syntax errors.

The web-based Coral simulator, language specification, and tutorial are freely available at corallanguage.org [15]. For classroom and further learning, the developers of Coral published an introductory paper in 2019 [16]. Since the release of Coral in 2017, the language and simulator have been used at 20+ universities (likely more) and by thousands of students.

This paper is not about the Coral language itself, which was introduced by others in an earlier publication [16], but rather is about an experiment to answer a question:

> *Can a CS1 course start by using Coral, then switch to a commercial language, and still have students attain the same proficiency in the commercial language?*

CS1 usage was not the main original intended use of Coral; instead, Coral was intended to serve an entire CS0 course, or an AP CS Principles course. But, using Coral in CS1, as stated above, could have the immediate benefit of a smoother and more inviting first several weeks, due to fewer language issues. Such use would also allow an instructor to teach using the Coral simulator, which can be an enjoyable and illustrative aid to teaching. Note that this paper only seeks to address the above question, which is necessary to answer before embarking on the broader/harder questions of whether introducing Coral in CS1 reduces withdrawals, Ds, and Fs, and whether such an introduction can improve the amount of learning in CS1. We hope that we and others will address those questions in future work.

Section 2 describes our methodology. Section 3 discusses the results of this study. Section 4 provides a discussion of this work. Section 5 concludes.

2. Methodology

2.1. Course

We conducted a study in our CS1 course in Fall 2019. The course serves 300-500 students per 10-week quarter (fall, winter, and spring, plus another 100 students in the summer). The course has about half computing/engineering majors and half nonmajor students from disciplines like biology, business, mathematics, chemistry, and more. The course normally teaches input/output, variables, branches, loops, functions, and vectors, all in C++. The course is taught in sections of 80-100 students, and each section is taught the same. All sections take the same midterm exam in the 5th week and the same final exam after the 10th week, with each exam being half multiple choice and half coding questions. For all sections, each week has (1) the same required online reading with auto-graded short question activities, (2) online small auto-graded coding homework activities, and (3) 5-7 small/medium programming assignment activities requiring about 3 hours. All 3 items are provided via a single C++ zyBook [17], configured to have one chapter per week.

2.2. Data collection

For this work, we focus on data from three sources. First, we analyzed student grades from weekly reading activities, challenge activities, and programming assignments, from both the midterm and final exam, and from the overall course grade. Second, we collected data from surveys given to students throughout the quarter. Students were typically given 2-3 anonymous surveys each week containing questions that pertain to class information and logistics, their experience at the university, personal lifestyle questions, feedback for the class, and more. We include responses spanning all surveys, however we primarily focus on survey responses from two "stress surveys" given to students during weeks 4 and 8. The stress survey questions are in the form of a 6-point Likert scale: Strongly agree (6), Agree (5), Slightly agree (4), Slightly disagree (2), Disagree (1), and Strongly disagree (0). Finally, we examined the university's anonymous end-of-quarter course/instructor evaluations.

2.3. Experiment details -- Teaching one section using Coral before C++

During the Fall 2019 quarter, for one course section of 80 students, we introduced a Coral-to-C++ teaching approach to our CS1 class. We combined the C++ zyBook with a Coral zyBook [18], and configured the book so that the first 5 chapters/weeks were taught using Coral, and the second 5 chapters/weeks were taught using C++. This configuration meant that the second half of the class was largely redundant conceptually, though the C++ did add content on strings and vectors. Many of the second half's C++ activities were repeats of Coral activities. In lecture, we taught the first half using the online web-based educational Coral simulator, walking students through example programs coded live in class, showing flowchart representations, and code execution with statement tracing and memory updates. All student homework and programming assignments were done in Coral using the simulator, with a Coral auto-grader in the zyBook. While the C++ midterm taken by students in the other two class sections covered input/output, assignments, branches, and loops, the Coral midterm additionally covered functions and arrays. However, all class sections took the exact same final exam, in C++, covering input/output, assignments, branches, loops, functions, strings, and vectors.

2.4. Class demographics and background

Based on class demographics provided by our university, the class section that used Coral had 16% (13) of students who were computing majors (Computer Science, Computer Engineering, or Computer Science with Business Applications) while 84% (67) were non-major students (Biology, Math, Physics, etc.). Furthermore, most of the students (60%), were freshmen.

By giving in-class surveys, we also got a sense of the students' prior programming experience. We asked "Do you have prior programming experience? (Don't worry; none is expected)" where the responses had a range of 0 (No experience) to 5 (Lots of experience). The average response was 0.84, indicating that most students had no prior programming experience. We also asked "If you had prior programming experience, where did you get it and in what language?" Responses varied from "Java from working at an internship," "Small dabble in Python via Codecademy," and "C++ from highschool."

We could not obtain background information regarding race, gender, or some other demographics.

3. Results

3.1 Grade performance

Our main question was whether the Coral-to-C++ students would learn C++ as proficiently as the C++-only students. The main measure was the students' performance on the C++ final exam, which was half code-writing problems and half multiple choice questions. Table 2 summarizes our results. For completeness, we also show results for the midterm exam and for the final course grade.

The Coral-to-C++ students performed about the same on the final exam as the C++-only students did. In particular, both groups' final exam coding scores were very similar; the small difference was not statistically significant (the p-value is greater than 0.05).

Table 2. Coral-to-C++ students performed roughly the same on the final exam in C++ as the traditional C++-only students.

	Coral-to-C++ average	C++-only average	p-value
Midterm	82.35	79.32	0.0762
Midterm MC	84.73	82.46	0.0944
Midterm Code	79.97	76.19	0.0921
Final	77.60	79.72	0.1753
Final MC	80.33	82.57	0.1081
Final Code	**74.86**	**76.86**	**0.2476**
Total Class	82.89	85.65	0.0782

3.2 Stress survey

For several years, we have given our CS1 students a brief anonymous survey each week. In week 8 (of 10 total), we give students what we internally call a "stress survey" that seeks to determine how a student is "feeling" about the class -- are students happy or stressed, are they confident or nervous, etc. We had some concern that teaching two languages, namely Coral followed by C++, might cause extra stress. We thus investigated whether the Coral-to-C++ students showed any negative signs on the stress survey compared to the C++-only students.

Below, each question's answers were a Likert scale: Strongly agree (6), Agree (5), Slightly agree (4), Slightly disagree (2), Disagree (1), Strongly disagree (0). The questions above the black line are questions for which higher scores are better, while below the line are questions where lower scores are better. (On the actual survey, such questions were intermixed).

On most questions, the Coral-to-C++ students' responses were about the same as the C++-only students; differences were not statistically significant (p-values were greater than 0.05). Thus, overall, we conclude that the Coral-to-C++ students were generally not significantly worse off from a stress perspective. However, two questions, related specifically to the programming assignments ("zyLabs"), did show a significant difference. Based on comments from students, this appears to be due to the Coral-to-C++ students having to do more C++ labs in the latter weeks than the C++-only students, since the C++ subjects were being packed into 5 weeks rather than spread over 10 weeks.

Table 3. Results from a "stress survey" given to students during week 8 of the class, showed that overall the Coral-to-C++ students were mostly the same as the C++-only students. The two questions with a statistically-significant difference (p-value < 0.05) are likely due to the Coral-to-C++ students having to do more C++ programs in the latter weeks.

Question	Coral/C++	C++-only	p-value
I enjoy the class.	4.95	4.54	0.0620
This class is an appropriate amount of work per week for the number of units.	4.06	4.41	0.0571
I was prepared for the midterm exam.	4.25	4.04	0.2840
I feel prepared for the final exam.	2.80	3.06	0.1767
The weekly zyLab programming assignments were enjoyable.	3.71	3.91	0.1744
The weekly zyLab programming assignments contributed to my success in the course.	4.74	4.58	0.3004
I learned a lot from the weekly zyLab programming assignments.	4.88	4.68	0.2299
I feel confident in my ability to write a small (< 50 line) useful program.	3.91	3.95	0.3926
I spend a lot of time in the class figuring out system issues rather than learning programming.	2.00	2.38	0.0864

The number of tools and websites for this class are somewhat overwhelming.	2.23	2.29	0.3749
I felt anxious before the midterm exam.	4.22	4.25	0.3986
I feel anxious about the final exam.	4.95	4.72	0.1851
The weekly zyLab programming assignments were stressful.	3.82	3.11	0.0089*
The weekly zyLab programming assignments were frustrating.	3.98	3.31	0.0114*
I am often anxious about the class.	3.17	3.05	0.3855

*Significant under the 0.05 probability level.

3.3 Student survey feedback on the Coral/C++ experiment

At the end of the course (week 10), we gave the Coral-to-C++ students an anonymous survey asking for their views on the experiment of starting with Coral and transitioning to C++. 64 students (80% of students) completed the survey. Each question's choices were a Likert scale: Strongly disagree (0), Slightly disagree (1), Neutral (2), Slightly agree (3), and Strongly agree (4). Questions above the black line are those where higher scores are better, while below the line are those where lower scores are better. We were concerned that students would find the Coral approach to be stressful due to the transition, or perhaps boring due to the ease of use or redundant learning of C++. Table 4 shows that overall, the students had positive responses related to the Coral-to-C++ approach.

Table 4. Results from an anonymous survey given to the Coral-to-C++ students regarding the experiment of transitioning from Coral to C++ in a CS1 course.

Question	Answer (0-4)
I enjoyed learning Coral	3.11
I understood Coral compiler errors	3.38
I was able to capture my programs in Coral without too much difficulty	3.11
I was able to successfully debug my Coral programs	3.28
I found the Coral simulator helpful in learning concepts	3.23
I found the Coral simulator helpful when debugging my programs	3.19
I enjoyed learning C++	3.03
I understood C++ compiler errors	2.28
I was able to capture my programs in C++ without too much difficulty	2.22
I was able to successfully debug my C++ programs	2.34
I think learning programming concepts (like loops or functions) was easier in Coral than it would have been in C++	2.86

I think learning Coral first made C++ easier to learn	2.75
I think my C++ code is neater/cleaner because I learned Coral first	2.52
I struggled with Coral's syntax	1.31
I struggled with C++'s syntax	2.41
I would have preferred to learn C++ through the whole course, without Coral	1.97
I think learning Coral first caused me to struggle more with C++ syntax than I would have otherwise	1.55

3.4 Student comments

On the survey mentioned in Section 3.3, we also asked a few open-ended questions for students to provide more insight on their experiences with Coral. In this section, we share some responses from the question "Please comment on the approach of learning Coral before C++ (pros, cons, experiences, etc)." The comments indicate most students found the Coral approach enjoyable and inviting. Even the occasional student who was indifferent to Coral, or found it a bit slow, seemed to understand the value that Coral offers.

"I was scared of CS1 so I did short online class (SoloLearn) on C++ before the quarter started. Coral completely brainwashed everything I learned during the summer, but I LOVE Coral. It made things easier to understand and I believe the flowcharts helped many of my classmates. When we switched to C++, I automatically connected to Coral concepts instead of the stuff I learned on SoloLearn. It just takes a bit of practice to get used to the syntax, but we should be practicing anyways. Overall wonderful experience."

"Learning coral Pros: it was easy to learn and understand. easy to follow and straight forward. transitioning to C++ was easier and I felt it was much more efficient than Coral. Organizing and saving lines of code was more evident with C++. Learning the concepts we learned in Coral in C++ was like extra review. (was easier to understand and make a connection with)."

"Coral provided a good foundation for students to build on. Coral taught that structure and readability is important. Coral has an amazing simulator that can step through the code like a debugger. Very good for beginners."

"Although I believe that I would have liked to learn C++ from the beginning (because I would have liked to dive into the intricacies of the language), I think that Coral has its value from the reasons stated above."

"made it a lot easier to grasp the concepts (especially the FLOWCHARTS) so by the time i got to C++ it was easy to get the hang of"

"i personally enjoyed learning coral before going into c++ because coral acted as a way to "test the waters" before we make the deep dive in c++. i found it beneficial because coral is a much more beginner friendly language and has really helpful ways to help both beginning and experienced programmers solidify a good programming base. Learning coral first really helped me incrementally understand the logic and reasoning that many programming programs have and helped me apply what i learned to those problems (even in the real world). I personally did not find any "cons" with using coral but im sure other students have a difference experience with it than i had."

"I loved learning coral first. The only problem I have is how long we spent on coral. I wish we would have had more time to work with C++ before taking the final. Especially since we have gone from exclusively typing our code to writing it on paper."

"I like that Coral was the first experience to programming because it allows for a comfortable learning experience without the stress of actual programming (without the support system and guide)"

"I think that Coral is super effective in learning to code, especially for C++, because of the similarities in syntax. For these reasons, I think that Coral should be more commonly integrated into other introductory coding classes."

"Coral was more of learning the basics and understanding how to code. C++ was applying it almost into the real world. Coral was useful in the sense that you get a feel for it and C++ was easier to understand after using coral"

"Coral definitely made the transition to C++ smooth rather than jumping straight into C++ with little background in programming."

"Coral I feel helped me understand the material in a more simple way and it wasn't too much of a challenge to translate it to c++. Most of us (including myself) had never programmed before so I feel like using coral really did help out a lot."

"Learning Coral first was AMAZING. It made learning concepts so much easier since the language and form was intuitive! Then C++ concepts were easy, and we could focus on mastering language specifics."

"Since I already had experience programming before it didn't make too much of a difference to learn Coral before C++ but I could see why it would be easier for people who had no coding experience to learn Coral before C++."

"Its good for absolute beginners (which is what this course is intended for) but for me it felt like going backwards. Coral was as close to pseudo code as one could get and that is really good for beginners."

"Honestly this course was such a breeze when we learned c++ after coral, most of the concepts are the same just with c++ having added features like strings. This course would have been much harder if we just learned c++ for 10 weeks."

4. Discussion

Overall, the question of whether a Coral-to-C++ approach leads to equally-proficient learning of C++ seems to be answered as *yes*. And, our concern about stressing students with the Coral-to-C++ transition, or of boring students who want to start in C++, seem not to be problematic issues. In contrast, the student comments indicate that they enjoyed learning Coral first.

Furthermore, we as the instructors found the Coral approach to be *extremely enjoyable to teach*, eliminating nearly all syntax/semantics issues in the first weeks of the quarter and letting us focus on the logic of programming and on problem solving. The online web-based simulator also was a pleasure to use during lectures, supporting code stepping, displaying variable updates, and depicting flowchart views of code. The students indicated they enjoyed using the simulator as well, and we had no reports of students having trouble accessing or using the simulator.

However, we believe we got a few things wrong in the initial design of the Coral-to-C++ approach. Based on the experience, we plan to make several changes in our next offering in Spring 2020 (ongoing at the time of this paper's writing):

- We taught Coral for 5 weeks, covering input/output, assignments, branches, loops, functions, and arrays. As we got into functions and arrays in weeks 4 and 5, we started to feel it was time to switch to C++, but could not because the syllabus was already defined. Some students made the same point in the end-of-quarter survey. Thus, next time, we intend to cover up to loops in Coral, and then start teaching C++ in week 4.
- We taught C++ using the same content and sequence as in the 10-week C++ section, but packed those sections into the last 5 weeks. That was likely a mistake. First, there were too many sections for students to cover each week in those latter weeks. Second, the C++

input/output and assignments labs were trivially-simple for students who were already doing problem solving using branches and loops in Coral. The students made this point in the end-of-quarter survey. Next time, in week 4, we will simply show how the already-learned Coral statements can be rewritten in C++: input/output, assignments, branches, and loops. We'll then immediately have students work on C++ programs starting at the level of loops. Then, we will introduce functions and vectors in weeks 6-9, which we can do at a slower pace since we'll have more weeks in the latter half of the quarter.

5. Conclusion

The details and intricacies of commercial languages like Python, C++, or Java can magnify fears and struggles of many students when first learning programming. We wished to investigate whether starting a CS1 course using the free and ultra-simple Coral language, and then transitioning to C++, could lead to equally-proficient learning of C++. Our experiment in Fall 2019 demonstrated that indeed Coral could be used as such. We also found that the approach did not introduce significant stress or other negative consequences. Instead, both students and instructors indicated they very much liked the approach, saying it was enjoyable, lessened fears, and helped the course run smoothly. Looking ahead, we intend to adjust how we introduce Coral to further improve the student experience, and we hope others will experiment as well. We believe using Coral at the start of CS1 classes, then transitioning to C++, Java, Python, C, etc., may ultimately improve both the learning and teaching experience.

References

[1] J. Bennedsen and M. E. Casperson, "Failure Rates in Introductory Programming," in ACM SIGCSE Bulletin, Volume 39 Issue 2 (pp. 32-36), 2007.

[2] J. Bennedsen and M. E. Casperson, "Failure rates in introductory programming: 12 years later," in ACM Inroads, 2019.

[3] S. Bergin and R. G. Reilly, "The Influence of Motivation and Comfort-Level on Learning to Program," in PPIG, 2005.

[4] T. Jenkins, "On the Difficulty of Learning to Program," In Proceedings for the 3rd Annual conference of the LTSN Centre for Information and Computer Sciences , Loughborough, UK August 27 - 29, 2002.

[5] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz, "A Multi-national, Multi-institutional Study of Assessment of Programming Skills of First-year CS Students," in ACM SIGCSE Bulletin, 33(4), 125 – 180, 2001.

[6] C. Watson and F. W. B. Li, "Failure rates in introductory programming revisited," in Proceedings of the 2014 conference on Innovation technology in computer science education (ITiCSE '14). New York: Association for Computing Machinery (ACM), pp. 39-44, 2014.

[7] E. Lahtinen, K. Ala-Mutka, and H. Jarvinen, "A study of the difficulties of novice programmers," in SIGCSE Bull. 37, 3, 14–18, 2005.

[8] Alice. https://www.alice.org/. Accessed January 2020.

[9] Scratch. https://scratch.mit.edu/. Accessed January 2020.

[10] Snap. https://snap.berkeley.edu/. Accessed January 2020.

[11] K. Kahn, R. Megasari, E. Piantari, and E. Junaeti, "AI programming by children using snap! block programming in a developing country," in ECTEL Practitioner Proceedings 2018: 13th European Conference On Technology Enhanced Learning, Leeds, UK, September 3-6, 2018., 2018.

[12] S. Lopez, J. M., Gonzalez, M. R., & Cano, E. V., "Visual programming languages integrated across the curriculum in elementary school: A two year case study using “scratch” in five schools," in Computers & Education, 2016.

[13] S. Mishra, S. Balan, S. Iyer, and S. Murthy, "Effect of a 2-week Scratch Intervention in CS1 on Learners with Varying Prior Knowledge," in Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education, ser. ITiCSE ’14. New York, NY, USA: ACM, 2014, pp. 45–50, 2014

[14] K. Powers, S. Ecott, and L. Hirshfield, "Through the looking glass: teaching CS0 with Alice," in Proceedings of the 38th SIGCSE technical symposium on Computer Science Education. (Covington, Kentucky, USA), 2007.

[15] The Coral Language for Learning Programming. https://corallanguage.org/. Accessed: January, 2020

[16] A. Edgecomb, F. Vahid, and R. Lysecky, "Coral: An Ultra-Simple Language For Learning to Program," in ASEE Annual Conference & Exposition, Tampa, Florida, 2019.

[17] zyBooks - Programming in C++. https://www.zybooks.com/catalog/programming-c-plus-plus/. Accessed: January, 2020

[18] zyBooks - Fundamental Programming Concepts. https://www.zybooks.com/catalog/fundamental-programming-concepts/. Accessed: January, 2020