Roger D. Clement (rclement@kipr.org)
Carol Goodgame (cgoodgame@kipr.org)
Steve Goodgame (sgoodgame@kipr.org)
Elementary Students Debugging Practical Text-based Code
KISS Institute for Practical Robotics

Elementary Students Debugging Practical Text-based Code

Abstract

The understanding that today's students need to acquire the skill of writing code to be competitive and successful in their future career is well understood. There are many options for learning to write code and because of the oft cited fear of frustrating teachers with no experience and their students with syntax, debugging and compile issues, graphical programs are often used. This paper looks at the ability of elementary educators and their students currently participating in the Junior Botball Challenge program to successfully debug text-based code. Educators, (n=231) were given pre and post surveys and students (n =640), were randomly assigned to debug programs that had one of 4 common syntax errors. The self-reported efficacy of the educators and the success rates of the student's debugging indicate that younger students and elementary educators can easily and successfully debug text-based code given proper motivation and experiences.

Elementary Students Debugging Practical Text-based Code

1 Introduction

With the spread of computer science focused programs such as code.org's Hour of Code and the work of former President Obama's *CSForAll* initiative, many educators and school administrators are now understanding the importance of teaching computational thinking skills to their students. This often gets translated into teaching their students to "code". There are numerous options to teach coding including the work by code.org, girls who code, and Khan Academy. There is also a large amount of robotics and other devices to help including; FLL, Junior Botball Challenge, Dash and Dot, and Fit Bit. Most of these programs rely on a graphical programming interface with the idea of making the entry point to programming easier for younger students, however it is well understood that at some point to advance in programming the switch to a text-based language must occur.
With this in mind, a heated and oft misplaced argument stands at the forefront of elementary computer education. At what age can students successfully and meaningfully code in a text-based language? It was this perceived floor that was the driving force for innovation of the Scratch graphical programming language, (Resnick, 2009) Recent published works done by Miller(2017) and anecdotal research completed by the KISS Institute for Practical Robotics seem to show a floor for children being able to learn to program in a text-based language being far lower than what some have perceived. It has been stated that a great stumbling block in beginning students and educators learning to code in a text-based language is due to the frustration in dealing with syntax and compilation errors (Berners-Lee,2017). It is our intent with this work to determine at what frequency children, grades 2-6, can debug common syntax and compiler errors after they are comfortable with a given text-based development environment.

2 Previous Work

The Junior Botball Challenge program is a sustainable Computer Science (CS) and engineering, design-focused STEM program produced by the KISS Institute for Practical Robotics a 501(c)(3) nonprofit organization. This program is designed to support and empower elementary and middle school educators in teaching their classroom students coding and computational thinking concepts coupled with engineering design concepts using robots as a manipulative. Currently the program is being implemented in 354 partner schools (47% in-class) impacting approximately 4,000 students and 800 educators in 13 states. International participation includes schools in Austria, Canada and China.

The JBC strategy revolves around three main components: educator friendly reusable robotics equipment, a standards-aligned curriculum and professional development. The inquiry-based curriculum has formative assessments, with accompanying evaluation rubrics that are performance-based and built into each of the lessons and accompanying activities. Summative assessments include the completion of performance-based challenges, where student groups demonstrate their mastery of the concept. Many times these are publicly held challenge day events, where completion buttons are awarded to students for each concept "challenge" mastered.

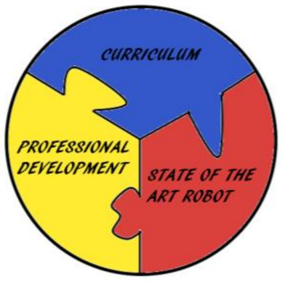
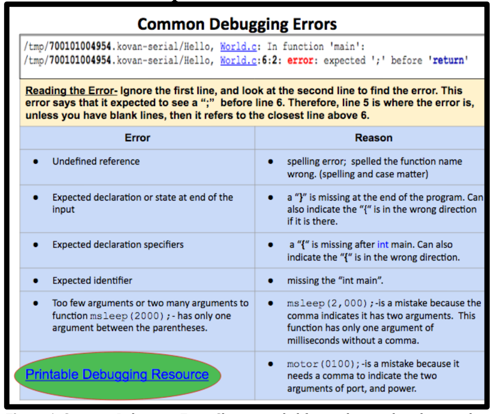


Image 1. The JBC 3-prong approach

The program engages classroom teachers that have no experience teaching programming and CT concepts. In a 2016-2017 school year post survey of participating Oklahoma City teachers (n = 231), 94.7% indicating they can easily debug student's programs that do not compile. This evidence supports the idea that elementary teachers with zero experience debugging can easily learn this skill and teach it to their students. This is not surprising, as the curriculum supports the early development of debugging skills by having educators lead the students in the creation of their own "Common Debugging Error" charts such as the one in Figure 1. Additionally, teachers are encouraged to engage in instructional strategies such as anchor charts (Figure 2).



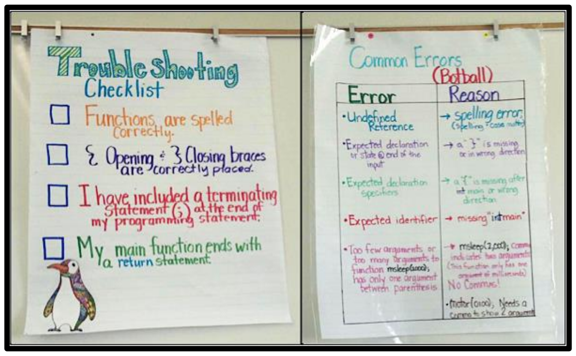
Common Debugging Errors

```
/tmp/700101004954.kovan-serial/Hello, World.c: In function 'main':
/tmp/700101004954.kovan-serial/Hello, World.c:6:2: error: expected ';' before 'return'
```

Reading the Error- Ignore the first line, and look at the second line to find the error. This error says that it expected to see a ";" before line 6. Therefore, line 5 is where the error is, unless you have blank lines, then it refers to the closest line above 6.

Error	Reason
• Undefined reference	• spelling error; spelled the function name wrong. (spelling and case matter)
• Expected declaration or state at end of the input	• a "}" is missing at the end of the program. Can also indicate the "{" is in the wrong direction if it is there.
• Expected declaration specifiers	• a "{" is missing after int main. Can also indicate the "{" is in the wrong direction.
• Expected identifier	• missing the "int main".
• Too few arguments or two many arguments to function `msleep(2000);` - has only one argument between the parentheses.	• `msleep(2,000);`-is a mistake because the comma indicates it has two arguments. This function has only one argument of milliseconds without a comma.
Printable Debugging Resource	• `motor(0100);`-is a mistake because it needs a comma to indicate the two arguments of port, and power.

Figure 1. Common Debugging Error Chart provided for teachers and students in the curriculum. Teachers are given strategies to help students help students generate their own guide


Trouble Shooting Checklist
Functions are spelled correctly.
{ Opening & } Closing braces are correctly placed.
I have included a terminating statement (;) at the end of my programming statement.
My main function ends with a return statement.
Common Errors (Botball)
Error
Reason
•Undefined Reference
→ spelling error
•Expected declaration or state @ end of the input
→ a "}" is missing or in wrong direction
•Expected declaration specifiers
•Expected identifier
→ missing "int main"
•Too few arguments or too many arguments to function msleep(2000); has only one argument between parenthesis
No Commas!
•motor(0100), Needs a comma to show 2 arguments
Figure 2. Anchor chart socially constructed in an Idaho classroom. A teacher with zero prior experience coding led the activity

3 Using the JBC Challenge as a data Collection Tool

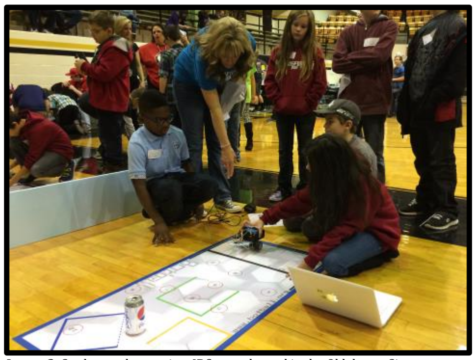
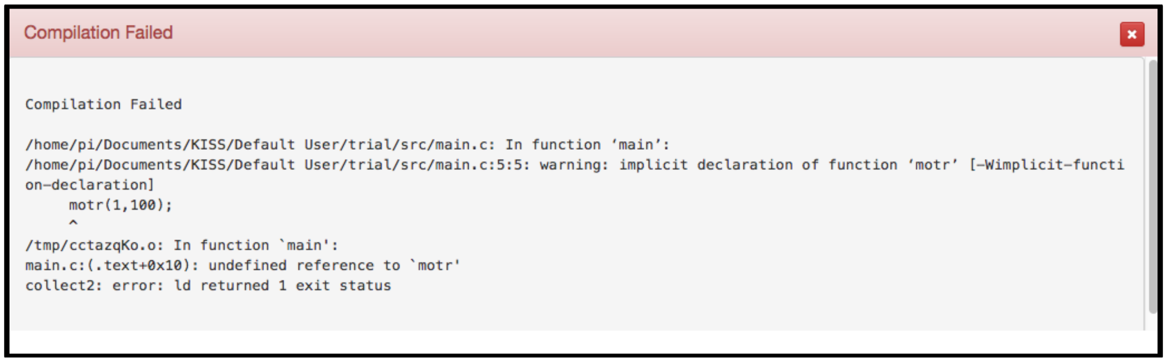


Image 2. Students take part in a JBC event hosted in the Oklahoma City metro area. Events have been as small as a single school and as large as 1,200 students.

Each JBC event that is hosted by the KISS Institute has one event that the coordinator calls a "mystery challenge". This challenge is unknown to teams and teachers prior to showing up the day of the challenge event. Typically these events are of moderate to higher difficulty and the student teams must solve the problem or complete the task if they would like the mystery challenge button. It was determined that an easy way to collect data on students fixing compiler errors would be to use five separate and individual stations set up as a mystery challenge at two of the larger events. The four most common errors reported by participating educators were then assigned a number (1-4) at random and a protocol and basic program was written that would allow these compiler errors to be observed individually, and as a whole. The KIPR Software Suite IDE provides information to the students when programs do not compile, which includes references to the line and column where the error occurred.



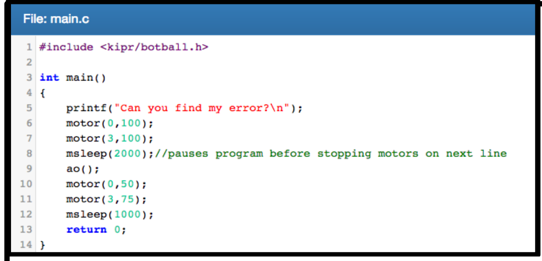
Compilation Failed

```
Compilation Failed

/home/pi/Documents/KISS/Default User/trial/src/main.c: In function ‘main’:
/home/pi/Documents/KISS/Default User/trial/src/main.c:5:5: warning: implicit declaration of function ‘motr’ [-Wimplicit-functi
on-declaration]
     motr(1,100);
     ^
/tmp/cctazqKo.o: In function `main':
main.c:(.text+0x10): undefined reference to `motr'
collect2: error: ld returned 1 exit status
```

Image 3. A screenshot taken of a common compilation error in the KIPR Software Suite IDE. This is a sample error message a student would receive if they were to misspell a function name.

The basic program and the intentional mistakes are outlined in Figure 3.



File: main.c

```
1  #include <kipr/botball.h>
2
3  int main()
4  {
5      printf("Can you find my error?\n");
6      motor(0,100);
7      motor(3,100);
8      msleep(2000);//pauses program before stopping motors on next line
9      ao();
10     motor(0,50);
11     motor(3,75);
12     msleep(1000);
13     return 0;
14 }
```

The common code for each of the stations:
The error that will correspond to each station is as follows:
Station 1: The semicolon will be removed from the end of line 8
Station 2: “msleep” in line 8 will be replaced with “mslep”
Station 3: An argument bracket will be missing from the end of line 8
Station 4: A comma will be placed between the 2 and the 0 in line 8.

Students will be in a line and will enter the first open station when they come to the front of the line.(hence, possibility to end up with a different number of responses on each trial).

Proctor will escort student to the open station where the student will begin.

When a student is completed, whether successful or not, the instructor will record their information.

Figure 3. Protocol and information sheet for administration of the “Don’t Bug Me”

4 Data

Error to be debugged	Attempts by students (n=)	Correct responses	rf as % Correct
1	61	51	83.6
2	89	81	91.0
3	55	47	85.5
4	74	69	93.2
Sum	279	248	88.9

Table 1: Data from the “Don’t Bug Me” station at the OKC Dell JBC Event on April 15th, 2017

Error to be debugged	Attempts by students (n=)	Correct responses	rf as % Correct
1	95	84	88.4
2	115	110	95.7
3	75	68	90.7
4	76	70	92.1
Sum	361	332	92.0

Table 2: Data from the “Don’t Bug Me” station at the MCN Invitational Event on April 22nd, 2017

It is important to note several small differences in these two trial groups:

The moderator for the first event did not put a time limit on any of the student participants. Additionally, these students were told that there was a compiling error that they would need to fix and then successfully compile.
The moderator for the second event put a 5-minute time limit on each of the students (kept by a stopwatch). Students in this event were told that there was "one error in the program" that they needed to fix.

5 Analysis

The (n=) value for each of these independent studies gives us a relatively large sample size of elementary students who have taken part in the JBC program. There is a minimal amount of statistical difference in mean % correct (88.44% and 91.78% respectively). This difference can be attributed to differences in moderator presentation of the test (informing students that there was only 1 error) or differences in the amount of in-class implementation of the material. It is our speculation, based on conversation and observation, that a larger percentage of teachers in attendance at the MCN Invitational were doing an in-class implementation of the program. Contrary to one of our unstated research hypothesis, there is no significant difference in student correctness based on the type of debugging error. Both sets of data showed that students were able to debug the misspelled function name the fastest and were slowest in correcting the error presented as a missing argument bracket. Exact times required to complete the debugging were not taken for this data.

6 Conclusions

The data provides evidence that elementary students, given the proper scaffolding and developmentally appropriate instruction, can easily debug basic compiling errors in a text-based language. The attention to detail and persistence on task (Common Core Math Practices) required to code and debug in a text-based language are excellent skills for a student to acquire. These cognitive processes may not be as much of a hindrance to learning to program in a text-based language at an early age as many people have indicated.

7 Implications for future work

The findings of this study and statistical verification of the authors' previous observations have opened the door to more research questions. Among these questions include a determination if there is a correlation between a student's ability to debug increasingly complex errors and the total time spent coding in a classroom setting. Additionally, the authors would like to develop practical solutions to aid teachers in their quest to maximize students' abilities in this area in the least amount of time.

References

D.P. Miller, R. Clement, C. Goodgame, and S. Goodgame. Elementary students programming in C to make their robots do their bidding. In Submitted to the CLAWAR Workshop on Education and Robots: available now http://www.amerobotics.ou.edu/IRL/Papers/ dpm-clawar17-sub.pdf, September 2017.

M. Resnick, J. Maloney, A. Monroy-Hernandez, N. Rusk, E. Eastmond, K. Brennan, A. Millner, ´ E. Rosenbaum, J. Silver, B. Silverman, et al. Scratch: programming for all. Communications of the ACM, 52(11):60–67, 2009.

Berners-Lee, Tim. Learnable Programming: Blocks and Beyond. https://vimeo.com/216045469