**Software Development**

EduCode

A programming language designed towards younger students

Conference in Twin Falls

Idaho

2017

**Table of Contents**

1. Research
2. Description
3. Problem
4. Solution
5. Explanation of Value
6. Plan of Work Log
7. Documentation
8. Project Requirements
9. High Level Software Design
10. Testing
11. End User Product Documentation
12. Self-Evaluation
13. Future Prospects
14. References
15. Student Copyright Checklist
16. LEAP Resume

**Research**

Education has always been a large part of TSA and other groups. There are many forms of education, and these can be achieved through different methods. One contested topic is whether programming and development practices should be taught to K-8 children and encouraged. Various educational programming languages have been designed for children from Logo to Scratch. In order to replicate what these languages strived to do; research was done and analyzed on whether educational programming languages for children helped their educational learning and problem solving capabilities. A 1984 scholarly article by Douglas Clements and Dominic Gullo stated “Analyses revealed significant pre- to posttest differences on the Torrance Tests of Creative Thinking for the Logo group on fluency and originality as well as on the overall divergent thinking score, while no significant differences were found for the CAI group. Children in the Logo group may have increased their ability to produce original ideas and to produce creative ideas as compared to a normative group because the Logo programming facilitated divergent thinking within a figural context. Cliatt, Shaw, and Sherwood (1980) found increased scores on the verbal but not figural Torrance tests, probably owing to the treatment's emphasis on verbal thinking. The emphasis on figural thinking in the present study's Logo treatment may have accounted for the increase on the fluency and originality subtest scores on the Torrance figural test”. What this essentially means is that programming in Logo may have helped children 6 years old develop problem solving ideas and improve their problem solving abilities. It may have also increased cognitive ability and ability to think creatively. Learning to code is also going to be a key ability in the near future. With it also come endless possibilities for invention and innovation. A Guardian article states “Why is it so vital that we teach our children to code? We are already living in a world dominated by software. Your telephone calls go over software-controlled networks; your television is delivered over the internet; people don’t buy maps anymore, they use the web; we all shop online. The next generation’s world will be even more online and digital. Soon, your house will be controlled with software, some of your medical care will be delivered over the web and your car may even drive itself”. The importance of learning to program is growing every day, and it is up to the teachers of the past generations to educate the present. I myself have found in the past learning to program tedious, difficult, and overall frustrating with books and other materials; but at a certain age it all became much easier to understand in an educational environment. Last year I learned how to program in Java and C#, and this year I have expanded my horizon towards Python, C++, and JavaScript. While it was difficult to learn only the basics of these languages, it took concepts that seem too abstract for elementary and middle school children to understand. Thus lies the importance of development for educational coding languages. It simplifies the realm of programming to simple tasks and concepts for which any 3rd grader could easily understand. According to an Edutopia article, it brings up the importance of learning to code by stating “Seymour Papert (Schwarz, 1999), one of the bet known advocates for teaching computer science and programming in schools, sums this up well: ‘anyone who has witnessed a toddler using a computer has probably experienced a sense of awe at that child's facility with what for adults can be an infinitely frustrating gadget. It's one thing for a child to play a computer game; it's another thing altogether for a child to build his or her own game. And this, according to Papert, is where the computer's true power as an educational medium lies -- in the ability to facilitate and extend children's awesome natural ability and drive to construct, hypothesize, explore, experiment, evaluate, draw conclusions -- in short to learn -- all by themselves. It is this very drive, Papert contends, that is squelched by our current educational system.’”. All these sources show the importance of developing a programming language specified for children; simple, basic, and able to demonstrate concepts fundamental to all programming languages such as variables and output. Such importance is being ignored in the current educational environment and I hope my software can help contribute to leading to more awareness and development of the current educational system towards a coding focused learning base.

All sources are referenced under the “References” page.

**Description**

1. The problem stated under the “Research” section was one of lack of encouragement towards programming for younger children and how current languages are too advanced for an elementary schooler to grasp.
2. Thus, the solution: To create a programming language oriented towards elementary and middle schoolers to teach the basics and fundamentals of programming which include structure, formatting, and modification, and output. EduCode strived to deliver on these concepts with its functionality. EduCode is a structured top down programming language that contains concepts such as “variables” and “printing” to mimic more complex programming languages. Such concepts are used very often in these more complex languages such as Java, C++, and C#. EduCode was developed using C# and can compile to C# and also be executed through the Compiler Console which is an .exe. The file extension for EduCode files is .edc, and compiles down to .cs.
3. The educational value of this software piece is to help teach children K-8 basic programming ability and to help prepare them for future more advanced programming opportunities. The social value of this software piece is to help spread awareness towards increasing focus on teaching programming for children K-8 and to help inspire more education oriented programming languages and programs. While the language itself is not ideal, it helps to spread the dream of an ideal educational software and programming language to help gear children towards high school, graduate, and post-graduate programming.

**Plan of Work Log**

**TODO**

**Documentation**

1. **Project Requirements:**

* The project requires an Integrated Development Environment (IDE) to program the compiler in and a text editor to modify the .edc files. Nothing else was required to develop this software. Other software was used to help documentation (Microsoft Word) and presentation (Microsoft PowerPoint). The IDE used was Visual Studio 2015 and the text editor used was Visual Studio Code, both free to use and developed by Microsoft. The project itself consists of a compiler, which takes the raw input from the .edc file, and interprets it in order to execute and/or compile it into C# code, which then can be run independently through a C# compiler. The compiler is required to link into a console which can then display any necessary output. For debugging or general purposes, additional files are generated which help show the processes of the software piece.

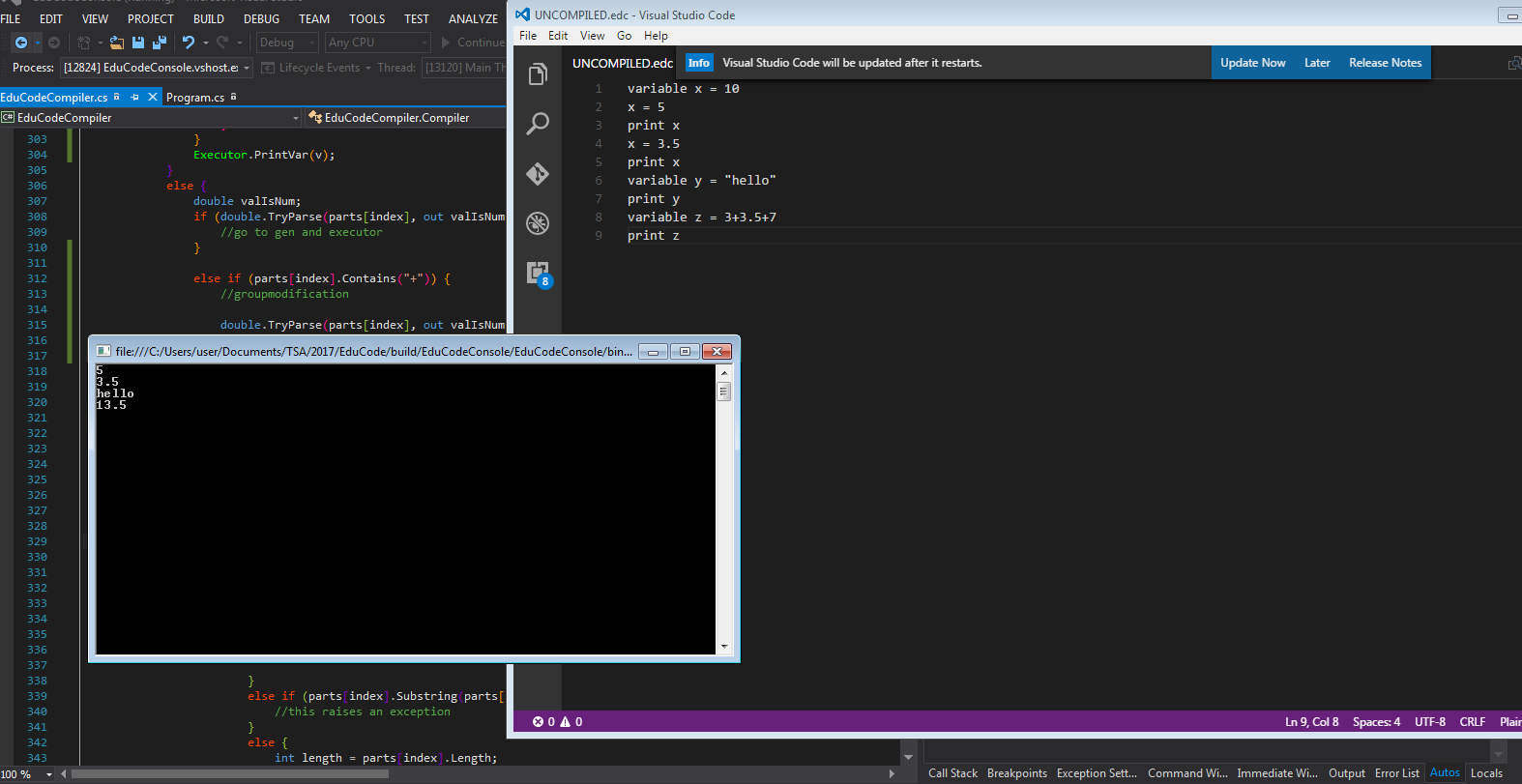
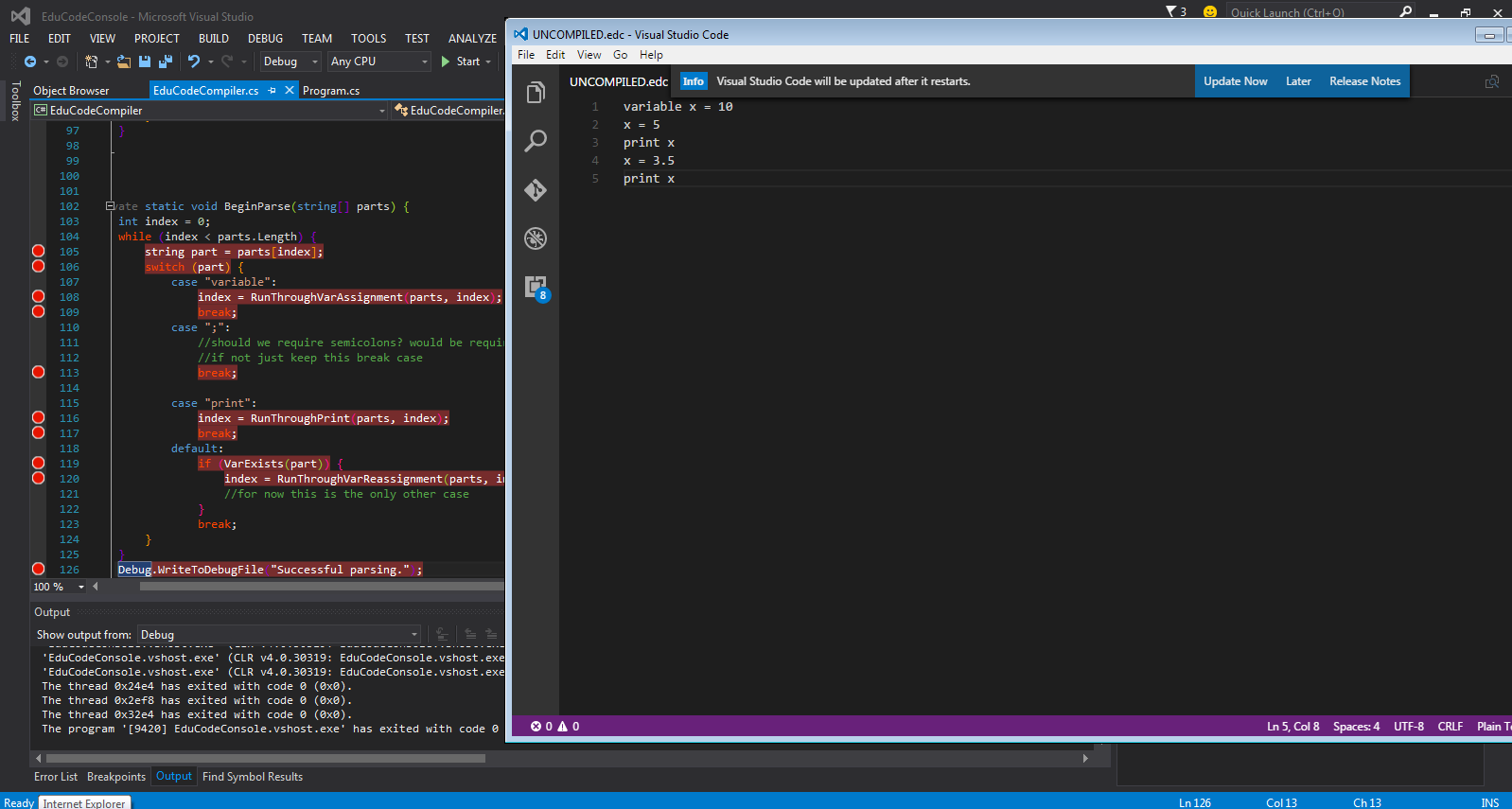
1. **High Level Software Design:**

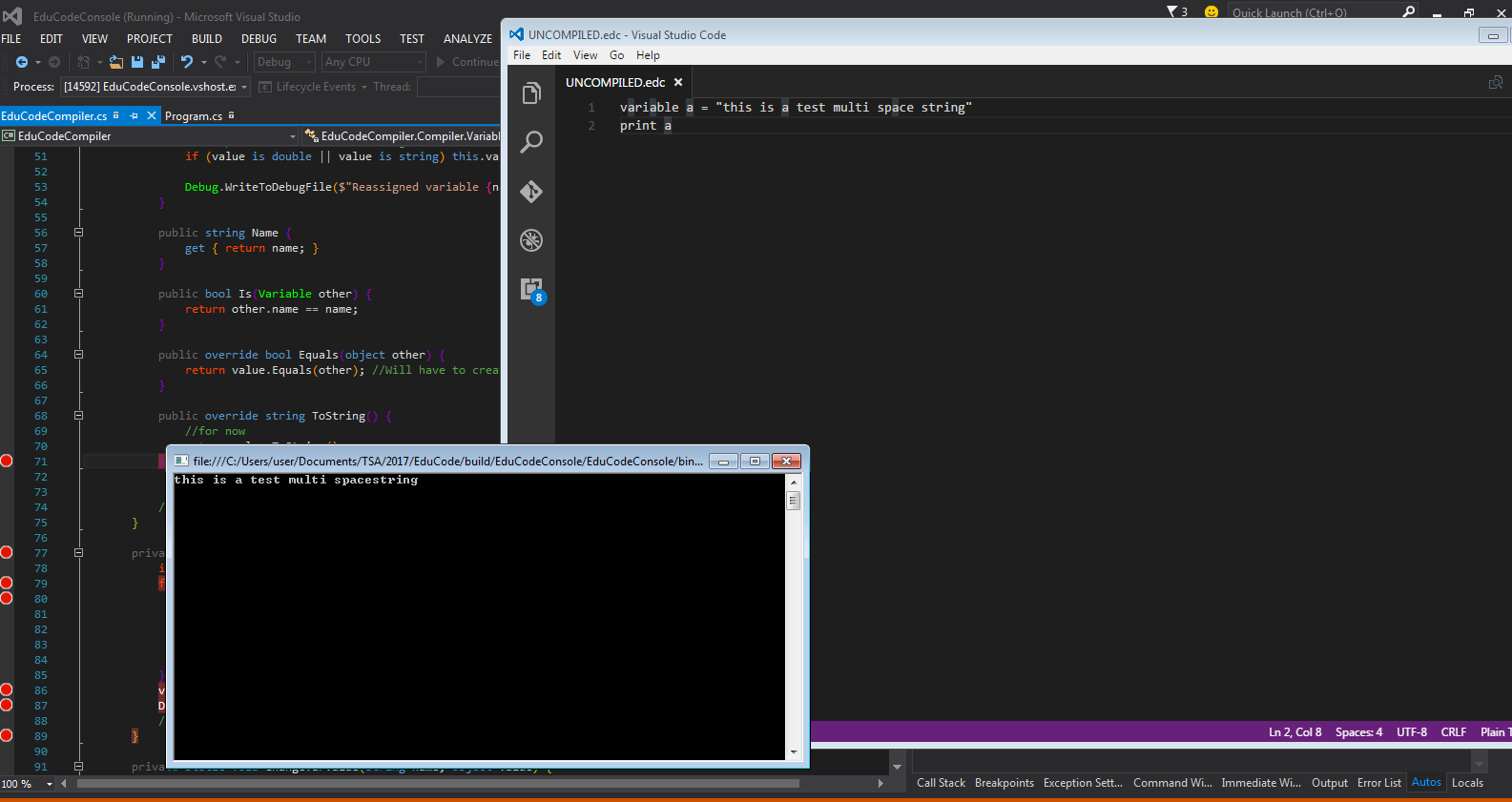
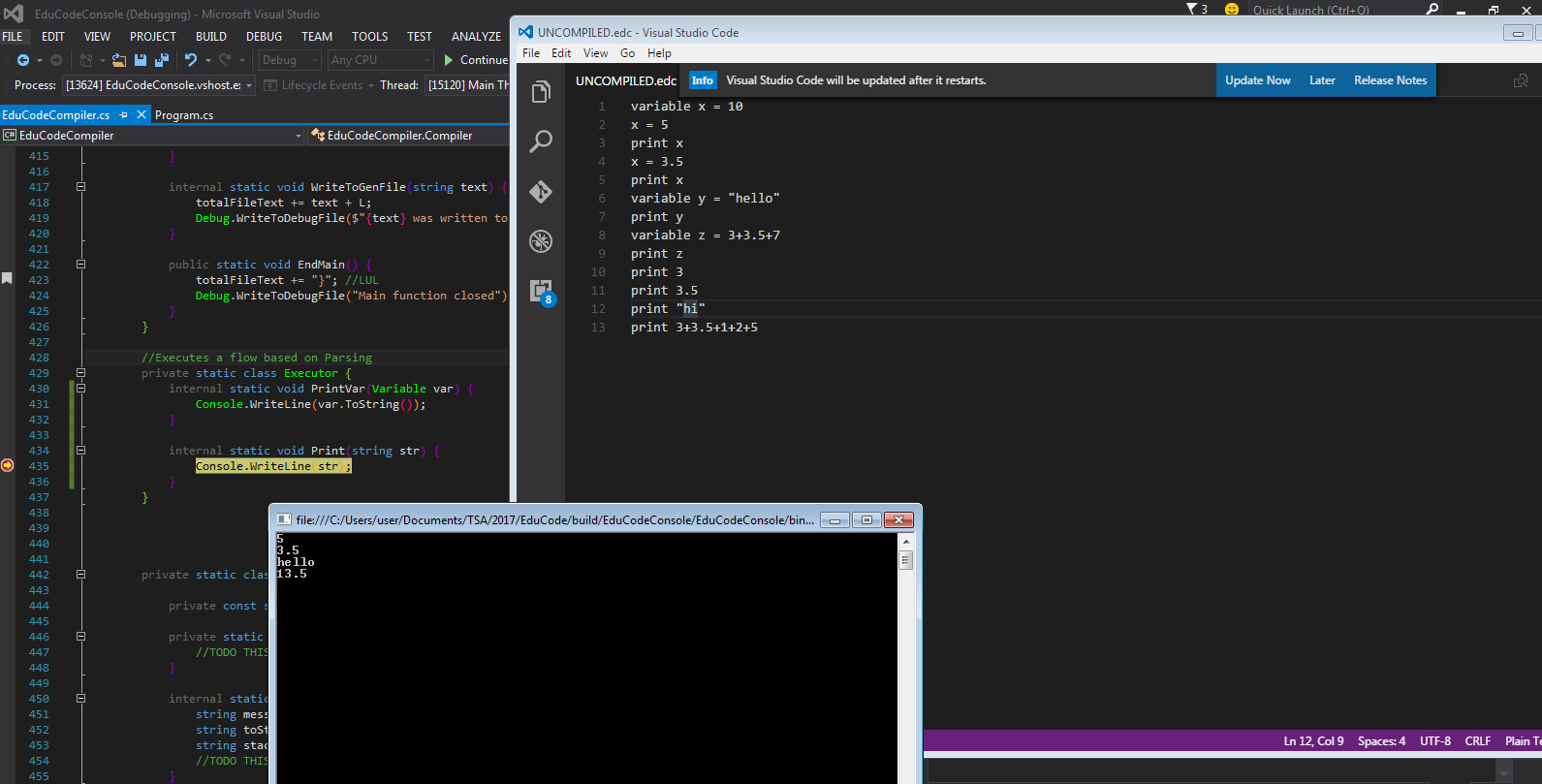
* The project was developed and run in Visual Studio 2015, and the Compiler itself is not a program, but rather a Dynamic Linked Library (DLL) which is referenced and used as an API to compile the EduCode program. The .edc file was simply edited and saved in order to test, and then the program would run to check if the desired outcome was achieved. After a feature or fix was implemented, the DLL was rebuilt and the Console application was run to see if the desired outcome was produced. Debugging was done using breakpoints, which stop the program after reaching a certain line of code. GitHub was also used as a version control system and also a documenting tool. Abstraction through methods was used in order to simplify a complex load on the system. Additionally, for debugging purposes geared towards the end user, a debug file is generated after compilation or during an error in compilation that delivers a general response as to what is the problem. File Input Output (I/O) is also used in order to open, read, and store the text of the .edc file and to generate and modify the .cs and debug file. Programming language interpretation was also researched and was used to be able to understand exactly how certain programming languages work. While it was not the complex and entangled way that many compilers do it, the program parses each text snippet with the spaces cut out, which contributes to problems towards programming style and interpretation of certain parts of the EduCode program.

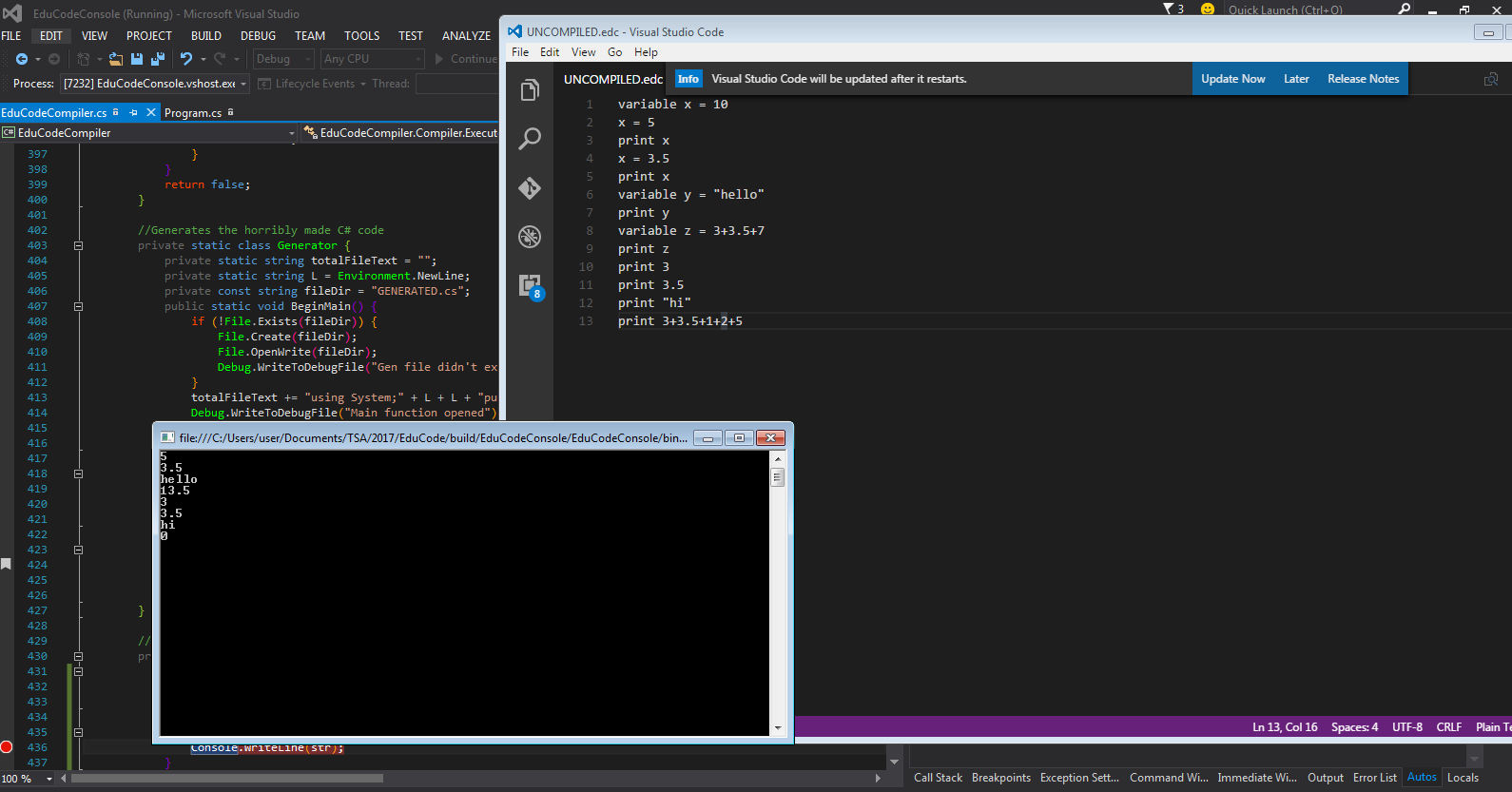
1. **Testing:**

* Testing consisted of changing the input from the .edc file, and checking the output generated by the .cs file and printed by the console. Line by line interpretation using breakpoints was also used if an input and output didn’t match. If satisfactory results were achieved, more features and fixes were worked on.
* **Screenshots:**

**Examples of Testing:**

****

**Examples of Debugging: **

****

1. **End User Documentation:**

* EduCode, developed by Noah Guidara for TSA State Competition in Twin Falls, Idaho. Produced in C#.
* Included separately should be a Syntax Cheat Sheet.

**Self-Evaluation**

Overall, the work put into this program was well worth it. I learned very much about how programming languages work and the benefits of learning them. The program could have had much more development time, but due to a busy schedule, lack of an idea, lack of confidence, and procrastination, it was delayed until mid-February and early March. While there was a lot of development time squandered, the core idea was implemented almost perfectly to achieve the goal I desired. Overall, the main idea of having a variable, changing it, and outputting it was achieved. While more time and work could have meant more time behind the keyboard, there is no guarantee I could be up to the task of creating a functional programming language, which usually requires years to make and a large team and large amount of resources. Overall, as a program gets bigger, its upkeep expands exponentially, meaning each new function almost doubles the amount of work to do each time. All this aside, the time put into this program when it was put in, was not wasted and was go time for me. I felt motivated, inspired, and ready to tackle the current challenge or implement the latest feature. I feel truly proud of my achievements and progress. The drudgery of figuring out exactly what went wrong was mind numbing and all the typing on the keyboard and editing lead to much stress. Even under pressure and having a lack of sleep, I succeeded in delivering on most of my goals I set for myself.

1. Future Prospects:

* I see myself coming back to this program in the future, to help develop it and update it on GitHub just as a hobby, and maybe to actually spread awareness of the problems we face with lack of education in programming. I see a very bright future, if upheld, for my program and maybe even a qualification for Nationals and a high score there. Overall, the future prospects for my program include a full launch in many years, a handoff to another individual or group, an open source, or future personal development towards national competition in TSA.

**References**

**TODO**

**Student Copyright Checklist**

(Print this out and fill it out)

Included is the Student Copyright Checklist sheet:

**LEAP Resume**