Criteria B – Computational Thinking

**Developing the Product**

Creating a full-stack Reverse Polish Notation (RPN) calculator that is functional, beautiful, and responsive is a complex problem that requires a thorough understanding of the stack data structure and algorithms. In RPN, an operator is encountered, the two most recent operands from the stack are ‘popped’ and the relevant operation is to be performed between them. The result is then pushed back onto the stack.

A tool that was used throughout the development process was the Vue JavaScript Framework, which was implemented through [technique] harnessing the data-binding features of the lightweight Vue instance as can be seen below.

A picture containing text, screenshot, font

Description automatically generated

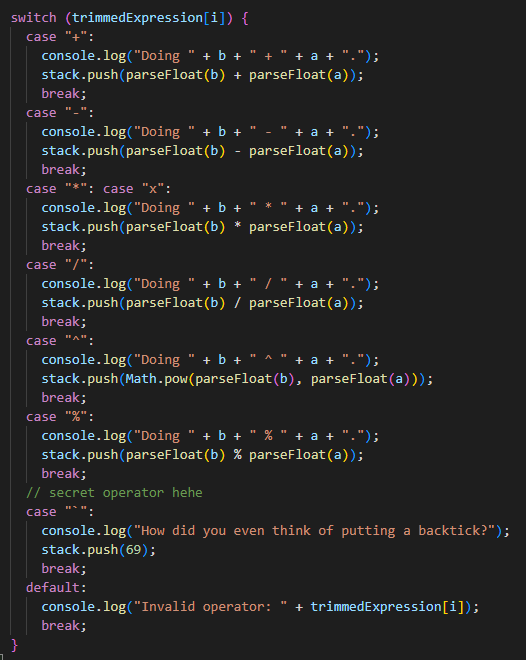
Vue, being an up-to-date and popular framework, is also widely compatible cross-browser, on all modern versions of Chromium, Firefox, and Safari (Success Criteria #7) making it a popular choice among many developers. Taking advantage of this useful property, values from the frontend are able to easily traverse to the backend in order to be processed. Additionally, with conditional rendering out-of-the-box (), the product aligns with Success Criteria #3, meaning that unneeded UI clutter is eliminated, and the product maintains a clean and minimalistic interface. The clear documentation and active community within the Vue ecosystem allowed for fast debugging of the product accelerated completion. By also utilizing Vue for input and event handling, excessive boilerplate JavaScript was omitted from the codebase, thus streamlining development, and increasing the amount of time able to be spent on both frontend and backend. An example of this is in implementing Success Criteria #3, where a keypress was to trigger the computation of a formula. Where in pure JavaScript this could be done like so:



In Vue, a single inline HTML flag is suffice (@keydown.enter):



Moving to the backend, the stack data structure [tool] was chosen as the optimal type for this specific product [technique]. However, the justification of this has been detailed in the Criteria A document. Therefore, the frequent usage of the switch statement [tool] within backend RPN evaluation will be the main subject of focus.



A notable appearance [technique] of the switch statement within the backend appears when a token within the spliced expression is deemed an operator. While both the switch and if-else statements are both conditional statements (selection) which execute different blocks of code based on different conditions, the switch statement has certain advantages over if-else blocks in terms of readability, maintainability, and performance.

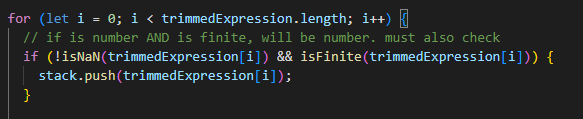
In implementing programmatical logic, the main focus besides functionality is future maintenance and potential expansion (Success Criteria #10). A switch statement achieves this by drastically simplifying and standardizing its condition syntax, essentially substituting condition **statements** for *cases*. Multiple cases can be chained to allow for alternate expressions of the same operator as seen here:



Due to its modular nature, a switch statement can eliminate the need for complex if-else trees which compromise readability and lower maintainability over time.

Switch statements are inherently implemented at a low level and therefore fundamentally begin with an advantage over if-else blocks in terms of performance (Success Criteria #6). Adding onto this is the fact that the JavaScript parser creates a jump table for execution at compilation, mapping each code block to its corresponding case. Therefore, no comparisons are required to be done on the fly which drastically reduces required computational or resource overhead.

The solution that I created was complex in that multiple different technologies, such as JavaScript, Vue, HTML, and Firebase (for deployment, although this is irrelevant) were utilized in order to create the solution and bring it to a usable state. Through in-class discussions with the teacher and student body, the structure of my implementation differed slightly in approach [ingenuity]. While most solutions included creating an array with each operand/operator as an index *then* operating on all values, my solution creates a persistent stack that will only ever contain a total of one calculation if the formula is valid. This can be seen below.





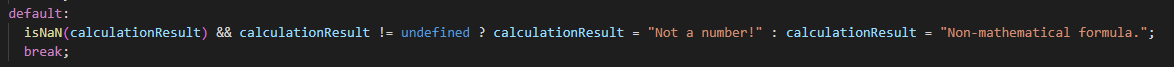
By forgoing the operator switch statement entirely when a number is added to the stack, unnecessary resources are not consumed, and the calculation environment is inherently isolated and not exposed. This prevents code leaking from other parts of the program resulting in unwanted behaviour and contributes to the reduction of runtime or logic errors.

In addition, in order to fulfil the requirements of Success Criteria #5, a specific breakdown of each error had to be given to the user. While division by zero cases and invalid formulas were easy to detect, the switch block does not implicitly convert expressions provided to it and cannot match inputted arguments against targets such as NaN, shown in the image below. Therefore, differentiating NaN cases and non-mathematical formulas (random letter spam or sentences) required an alternate solution [ingenuity].

A picture containing text, font, screenshot

Description automatically generated

By implementing a ternary operator, we can gracefully resolve the issue in one line by checking whether the result truly is Not a Number, and cross-checking that result with whether it is undefined (letters). If both conditions are met, we return NaN. If not, it is a non-mathematical formula. This solution and the implementation of such can be seen below.



Taking inspiration from other UI elements designed by developers worldwide, a smooth and aesthetically pleasing interface that contributes to the user experience was achieved (Success Criteria #3) taking inspiration from Google’s Material design language. A palette generator [tool] (see references and bibliography) was used in tandem with custom buttons in order to create a colour scheme that highlights the most prominent features of the website, these being the title, input, and calculation buttons. The minimalistic and intuitive interface is designed to significantly decrease the amount of time spent getting familiar with the product and to maximize productivity, evident through the monotonic colour background free of distractions.

Thus, given the above tools, techniques, complexity, and ingenuity used and exercised throughout the creation of the solution, the final deployed RPN calculator can be accessed [here](https://bruh-calculator.web.app), hosted securely with SSL traffic encryption at Firebase. It is a single page application and successfully fulfills all design specifications defined previously in Criteria A, being a viable and realistic solution to the given brief.

A screenshot of a computer

Description automatically generated

If you are interested in checking out the latest source code, it can be found [on my GitHub](https://github.com/woflydev/).

**Sources List (Bibliography)**

1. *Code Studio*. (2023). Codingninjas.com. <https://www.codingninjas.com/codestudio/library/time-and-space-complexity-of-linear-data-structures>. Last accessed 27/04.
2. *Coolors - The super-fast colour palettes generator!* (2023). Coolors.co. <https://coolors.co/>. Last Accessed 27/04.
3. *Form Input Bindings | Vue.js*. (2023). Vuejs.org. <https://vuejs.org/guide/essentials/forms.html#text>. Last accessed 28/04.
4. Holman, M. (2012, July 25). *How do you have a NaN case in a switch statement?* Stack Overflow. [https://stackoverflow.com/questions/11658439/how-do-you-have-a-nan-case-in-a-switch-statement. Last accessed 27/04](https://stackoverflow.com/questions/11658439/how-do-you-have-a-nan-case-in-a-switch-statement.%20Last%20accessed%2027/04).
5. Packtpub (2022). Packtpub.com. <https://subscription.packtpub.com/book/business-and-other/9781800206625/1/ch01lvl1sec04/understanding-the-vue-instance>. Last accessed 27/04.*193 CSS Buttons*. (2023). Free Frontend. <https://freefrontend.com/css-buttons>. Last accessed 28/04.