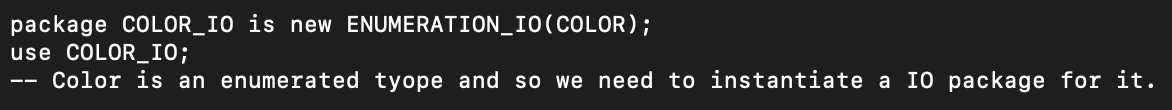
Matt Burton

CSC-4510

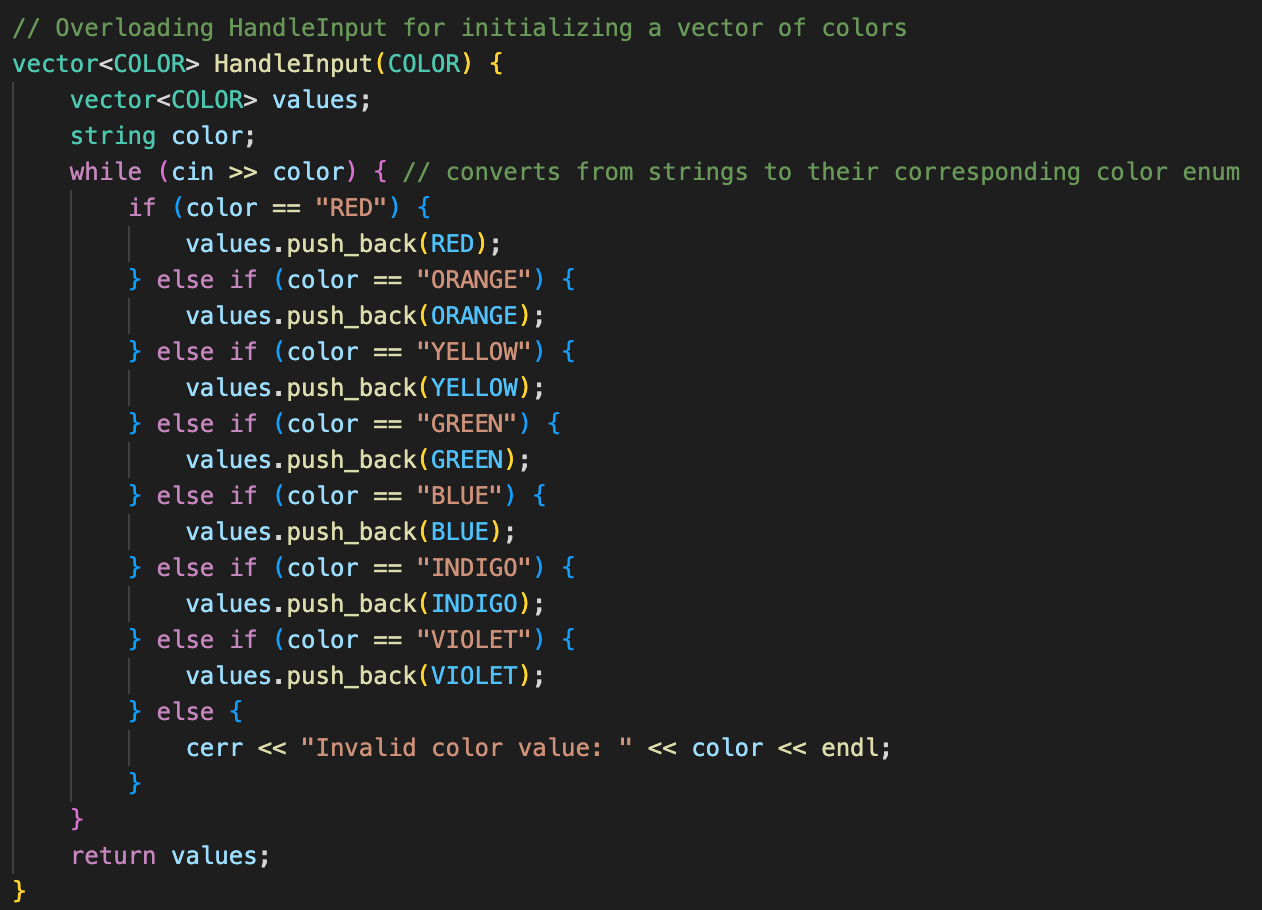
9/22/23

**Assignment 3**

Ada and C++’s ability for generics/templating offers increased flexibility when writing methods for processing different data types similarly. Templates in C++ are simple to implement and easy to understand, and they only require one additional line to identify a method as a template. The main strength of templates is being able to write one method to process any kind of data, which removes the need to write specialized functions for other primitive data types. A significant weakness in C++ is that data types are reduced to integers behind the scenes, so when using templates with enumerated types, the method couldn’t distinguish between the enum and a standard integer. Unlike Dr.Oudshoonrn’s Ada example, C++ does have a library for enumerated type IO as seen below:



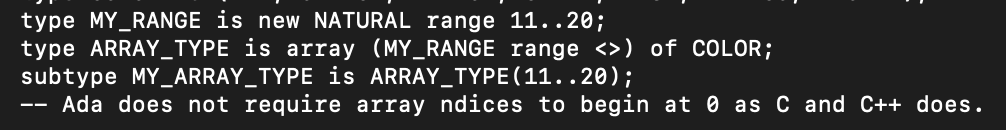
Instead, colors are read as strings and must be converted to their enumerated equivalent through the use of an overloaded input handling method.



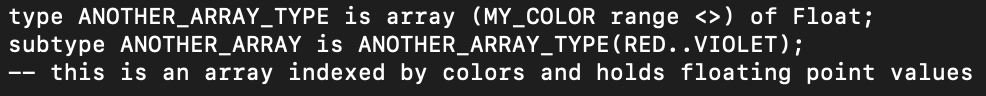
During sorting the data gets converted to integers, so I needed another overloaded function to convert from integers back to strings for the output.



Generics are great for generalizing methods and reducing code. However, the lack of support for dealing with enumerated types and C++ storing data as integers makes it difficult for the compiler to understand the type of data it’s handling in a templated function.

While exploring this topic, the provided Ada code was somewhat confusing, as I'm still learning Ada’s features. Also, some of the variables seemed to be necessary declarations for different parts of a similar goal, so they were named similarly but had distinctively different setups and applications which weren’t distinguished in the comments. For example, a comment explaining why and what you were doing with types and subtypes would have been helpful:  


I understand that this creates an array of colors, but I don't get the significance of starting at a custom range, nor do I get why the variables have the suffix \_TYPE when it seems like they are the arrays themselves and not an abstract description of array types. Additionally, when declaring the second array, indexing by color was interesting but it’s not clear to me if it's a necessity or a showcase of a unique feature of Ada.



Not being able to distinguish between the core functionality from the general Ada tips made deciphering the essential code for writing a generic function in Ada more confusing. Overall, I could understand the gist of what this program did, but it didn’t really help me write my templated sorting program in C++. Quick research led me to the syntax for templates in C++, but I spent most of my time working out scripts to automate testing, and methods to handle the automation instead of working with predefined arrays. Moving away from hardcoded arrays meant that the data type being passed in needed to be determined before storing the data because you can’t create an array of a template type. This was another weakness of C++ that made working with templates more challenging than I expected it to be. In the future, I’ll likely only use templates when necessary, as they may overcomplicate an otherwise simple solution.