**Z+- Interpreter – Java and C++**

For this problem, we used two imperative languages, Java and C++. While the overall structure of the code was similar, the implementation of the methods was different.

In both implementations, I kept track of all the variables and their values from the source code we were interpreting. In Java, I represented this as a HashMap, which is a dictionary: HashMap<String, String> vars. I used the variable name as a string for the key and the variable’s value as a string for the dictionary value. The issue here was that I needed a way to tell the variable type from its value, where the type could either be a string or an int. However, I could only store one type in the dictionary. I chose to just store the value as a string, and if the variable type was a string I would keep the quotes, and if it was an int there would be no quotes. This became somewhat annoying, since anytime I wanted to use the variable’s value I would need to check for quotes to determine the type and remove them before using the value.

|  |
| --- |
| **C++ Variable Table for Z+-** |
| std::map<std::string, Data> varTable;  enum Type {typeString, typeInt, typeVar};  struct Data  {  Type typeFlag;  union  {  char\* s;  int i;  };  }; |

In C++, I tried a different solution to the issue of storing multiple different types in one dictionary. I created an enumerable named Type for having a readable way to denote if something is an int, string, or variable. Then, I used a C++ dictionary (std::map) for storing the string variable name with its value. To represent the value, I created a discriminated union. The union has a flag for the type of thing it is storing (either a typeString or typeInt), and then it can either store a string (char\*) or int. I liked this way much better than the Java implementation because it was more readable and easier to use. For example, to determine the type of a variable, all I had to do was check the typeFlag. I did not have to check to see if it had quotation marks or remove them before use.