1 Project Description

The project is a Vector calculator that can perform various calculation on 2D and 3D vectors. The operations it can perform:

- Addition → Takes two vectors to add to them to each other

- Multiplication (Not working-Don't use) → Takes number and vector to multiply the vector with the number and get the scalar product

- Length → Takes one vector and gets its length

- Distance → Takes two vectors and get the distance between them

- Dot → Takes two vectors and get the dot product for them

- Rotation (Not working-Don't use) → Takes an angle in radian and a vector and returns the vector after rotating

To use the program your input should be in the form 'Command (x y) (m n)' in case of 2D and 'Command (x y z) (m n l)' in case of 3D, where command is represented by any of the operations above. So an addition example would be 'Addition (3 4 2) (5 6 7)' This should return back the answer 'ThreeD (8 10 9)'

2 Architectural Choices

- Data Vector → The data type that stores the vectors that will be used throughout the program to perform operations on. Has only two types: Two dimensional and Three dimensional vectors. I am aware that vectors can theoretically have an infinite amount of dimensions but for my program i decided to limit it to only Two and Three dimensions. This will be part of the improvements i would do if i had more time/more skill in Haskell.

  The data type is capable of taking any type in either the 2D or 3D versions but we will only be using double as some functions like vectorDistance for example that uses square root requires a floating point number.

  I derive both show and read to be able to convert strings that will be parsed to the data type and to be able to show it in terminal.

- Data Mode → The data type that specifies what each command that can be performed on the vector needs. It includes the operation as its first constructor and either two vectors, one vector or a vector and a scalar value as its second/third constructors, those would be what the operations are performed on.

- Parseing → I made the user input the whole string including the command and the vectors as i thought it would be easier for the user to do that rather than me asking for input for the operation, first vector and possibly a second vector separately.The parserHelper will take the the input from the user and returns a mode which will include all the information needed to

perform the operation. I chose the format i mentioned before mainly as it will give me places to stop and end the parsing like the brackets before and after vector for example.

- Operations → For the vector operations i wanted to use a maybe or either function to make them total functions and not partial functions. I ended up scrapping that idea though as it would have added more complexity to a project where i started with no knowledge.

3 Libraries

3.1 Dependency Listing

- Haskeline - Used as the main IO, where is helped me loop through the operations and helped with the input.
- Parsec - The main library used as i used it to parse the user input to something useful my functions can use.
- Containers - Includes many useful libraries like lists

4 Personal Experience

The program helped me understand how to deal with parsing, which is one of the harder things to do when dealing with user input. Mainly because we have to make sure the user input the information in the correct format and that the user will receive the correct output if the format was correct. Formatting from string to my own made data structure proved challenging to me especially as a beginner to functional programming. My parsing is by no means perfect of course, some of my vector operations use Integers as an input and not two vectors, figuring out how to generalize parsing is still a problem that i will need to figure out in future projects that include parsing, for example two possible operations would have been Multiplication and Rotation, but i ended up commenting both of them out as they take a number and a vector, while the parser was made to only accept vectors. I used the data structure mode, which i took the inspiration of from the ExactMatch in our hurdle coursework, where the thing we need to compare/use is in the same data type. The coursework helped familiarise me with parsec which is one of the more popular libraries for IO in Haskell. It made me look into several other libraries previously unknown to me and made me use them like haskeline for example, also a few libraries that i didn't end up using like Data.Map.Strict for example, where i was going to map my operations where the key would be the string and the value would be the Mode a. I used monads for the first time in a full program and implemented it to an extent which made it work. It also helped me understand the vector topic in CS131 better since i had to directly make algorithms that will help do operations on vectors.

5 Resource List

5.1 General Resources

- Parsec Hackage documentation
- Haskeline Hackage documentation
- Learn you a Haskellopera

5.2 Credits

Credit: Musab Guma'a for report template and for helping me understand parsec functions.