Design Document

Overview

The tree data collection application, is a simple to use program for both collecting and exploring data input by a user. Using dynamic arrays and several link lists, it is capable of tracking tree data from a neighborhood and exploring through a "street view" perspective. For this CLI utility, the structure of the program is broken into 5 modules and 2 supporting utilities:

Modules

1. Input Mode - Handles the input of data from the user through the CLI
2. Exploration Mode - Allows the user to navigate the data using the CLI
3. Import Data - Imports data from a previous session using a DAT file
4. Export Data - Exports data collected during the input mode for later exploration
5. Main Menu - Entry and exit point of the application

Utilities

1. Linked List - A doubly linked list utility for storing critical tree data
2. Utilities - Quality of life functions for managing a CLI

Design

During the initial stages of this project, brainstorming revealed that the most efficiant way to achieve to goal of tracking shade trees, would be to work backwards from the stretch goals to the projects required deliverables.

Doubly Linked List

To begin, the doubly linked list class was formed to store data such as its associated street, and a node containing block level data. This was found to be the best method, as it kept a key of the longitudinal street, and as a doubly linked list, was integrated to support both head and tail navigation. (Front to Back or Back to Front). Several helper functions were also included to allow the appending of a node at the end, inserting a node at any point, deleting a node, copying the entire list, and deleting both a node or the whole list. Due to the nature of the project, the use of vectors to store the collection of linked lists was decided, due to its ability to keep an order of the longitudinal streets. A dictionary was considered for a short period of time, but was ultimatly decided against, due to its unordered nature. Pending future iterations to allow sorting of the longitudinal streets in the proper order provieded in a look up table.

Nodes

Inside the doubly linked list are the node structs which contain block level data including: the approaching cross street, the leaving cross street, and a dynamic array (vector) of tree data. Within the vector, is another struct containing individual tree data including: east distance between the next tree, west distance between the next tree, and tree type. In future iterations, this struct can be directly updated to include more data that would want to be stored. It is due to this reason that the node has been structured the way it is, to allow for quick and easy access to data in the code.

Input Mode

Outside of data architecture, the input mode was created as a way to interface between the user and the program, allowing them to input tree data directly. It utilizes link list navigation and several menus for quick CLI access to add a tree to a block, add a block, or add a longitudinal street. From there, the data is then exported to the results folder and stored for later use. JSON was initially considered as a human readable and easily integrated data file, but was ultimatly vetoed due to its more complicated

integration with the application. This is something that will be revisited in future iterations, to allow form multiple file format uploads

Exploration Mode

Due to the creation of the input mode first, the exploration mode was found to be easier to impliment. By reversing the export data function, the import data function allows for a user to reload past data. Upon its completion, several menus were constructed to dynamically list all streets and blocks. This utilizes an unordered map (dictionary) to store string or struct data and recall it with a key, input during the menu selection process. Future improvements include a sorting algorithm to sort each linked list, ensuring a continuous and geographic accurate map of city blocks.

Summary

This high level design documention serves as a general overivew to the structure of this application and reasoning behind choices made during development. For an indepth analysis of each feature and their supporting functions, please review the **Specifications Document.** An example DAT file will be provied in results for a demonstration. **results/logan_tree_data.dat**

How To Run

To run the application, simply run `Make`. This will provide the user with a menu to select input mode or output mode.

To run the unit tests, simply run `Make test` and the program will go through each of the unit tests in the **Tests/** directory.

Use of AI

As a tool for development, AI was utilized throughout the application process to development time. Both the doubly linked list implementation and utilites packages were devoid of AI influence, other sections such as the exploration mode and and import/export did use AI. Specifically the functions would be prototyped and later optimized by generative AI to fix glaring bugs, add edge case error handling, and

simplify the overall code. It was also instrumental in the initial design of the application, ensuring packages worked together and overall logic was sound.