

TALLER IA 1 – SANTIAGO DÍAZ OBANDO

Para este taller, en el que se solicitaba implementar una función a través de un LLM o IA como ChatGPT, utilicé los siguientes prompts. En el primer prompt cargué los archivos base (`list.h` y `main.cpp`) de los que debía partir la implementación. Hice uso del inglés para interactuar con el modelo. Antes que nada, en el prompt no hago explícito que únicamente quiero la implementación, por lo que empieza a explicar un poco acerca del código y las clases que van a permitir lograr la tarea, expandiéndose bastante.

PROMPT 1

- Files:
 - `list.h`
 - `main.cpp`

In the `main.cpp` file you will find a list of newline separated strings, and a function that generates a `DataList` object that is a linked list with the values in the newline separated string. Your task is to implement a function that is called `insert_aggregated_data()` that takes the list, the value you are going to aggregate and a step amount, The function has to insert that value at that every step amount of positions in said linked list In `list.h` you will find the implementation of the linked list, so you know which methods to use.

ChatGPT

To implement the `insert_aggregated_data()` function in `main.cpp`, I will first need to examine the `list.h` file to understand the linked list implementation and the methods available. Then, I'll review `main.cpp` to see how the linked list is created and populated with data.

Let me begin by checking the content of both files. I'll start with `list.h` to see the linked list implementation.

The `list.h` file defines a templated doubly linked list class named `List`. The class contains a nested `Node` structure representing individual elements in the list, and a `const_iterator` class for traversing the list. The key components include:

Node Structure:

- Each node contains `data`, a `prev` pointer to the previous node, and a `next` pointer to the next node.
- Two constructors are provided: one for copying data and one for moving data.

Const_iterator Class:

- Allows traversal of the list with operators for incrementing (`++`) and decrementing (`--`).
- `operator*` returns the data of the current node.

Given this, we can likely insert elements by navigating to specific nodes and modifying their `next` and `prev` pointers.

Now, let's review the content of `main.cpp` to understand how the linked list is utilized and where the `insert_aggregated_data` function needs to be implemented.

The `main.cpp` file includes a function `load_data_from_csv` that converts a newline-separated string into a `List<float>` object, representing a linked list of floats. The file also contains a prototype for the function `insert_aggregated_data` that needs to be implemented.

Implementation Plan for `insert_aggregated_data`

The goal is to insert an aggregated value into the list at every specified `step` interval. Here's how we can approach this:

Initialization:

- Start from the head of the list.
- Initialize a counter to track the position within the list.

Traversal and Insertion:

- Traverse the list using a loop.
- Every time the counter reaches the `step` value, insert the `aggregated_value` at the current position.

Inserting a Node:

- To insert a new node, update the `prev` and `next` pointers of the neighboring nodes to accommodate the new node.

Continue Traversal:

- After inserting a new node, reset the counter and continue traversing until the end of the list.

I will now implement the `insert_aggregated_data` function based on the above plan.

Here is the implementation of the `insert_aggregated_data` function:

```
cpp
void insert_aggregated_data(List<float> &list, float aggregated_value,
int step) {
    auto it = list.begin();
    int counter = 0;

    while (it != list.end()) {
        // Advance the counter and iterator
        ++counter;
        ++it;

        // If the counter matches the step, insert the aggregated value
        if (counter == step) {
            // Insert the aggregated_value before the current position
            list.insert(it, aggregated_value);
            // Reset counter after insertion
            counter = 0;
        }
    }
}
```

How to Use

- Parameters:
 - `list`: The linked list where the value will be inserted.
 - `aggregated_value`: The value to be inserted at each step interval.
 - `step`: The number of positions between each insertion.

Integration

- Add this function to your `main.cpp` file, and call it with the appropriate parameters where necessary.

If you need further assistance or modifications, feel free to ask!

A pesar de que la implementación era correcta, no me fijé en el comentario que se hizo dentro de la misma clase, el cual era que el promedio a insertar era el de los anteriores `step` datos, por lo que le pedí que realizara la implementación teniendo en cuenta este cambio. Cabe mencionar que en el prompt hago énfasis en que no debe usar ninguna clase o implementación distinta a la que ya había sido anteriormente dada. También hago mención del proceso que debe tener para obtener el promedio.

PROMPT 2

Now, instead of inserting a constant value, make it be the average of the previous step values, do not use the vector class, instead sum all the values as the list is traversed.

ChatGPT

To modify the `insert_aggregated_data` function so that it inserts the average of the previous `step - 1` values, we need to accumulate the sum of these values as we traverse the list. Here's how we can approach this:

Implementation Plan

- **Initialization:**
 - Initialize a counter to track the position within the list.
 - Use a `float` variable to accumulate the sum of the values over the interval of `step - 1`.
- **Traversal and Accumulation:**
 - As we traverse the list, add each value to the running sum.
 - Once the counter reaches `step - 1`, calculate the average and insert it into the list.
- **Reset After Insertion:**

- After inserting the average, reset the counter and sum, then continue traversing the list.

Implementation

Here's the updated implementation of `insert_aggregated_data`:

```
cpp
void insert_aggregated_data(List<float> &list, int step) {
    auto it = list.begin();
    int counter = 0;
    float sum = 0.0;

    while (it != list.end()) {
        // Advance the counter and add the current value to the sum
        ++counter;
        sum += *it;
        ++it;

        // If the counter reaches step, calculate the average and insert
        if (counter == step) {
            float average = sum / (step);
            list.insert(it, average);
            // Reset counter and sum after insertion
            counter = 0;
            sum = 0.0;
        }
    }
}
```

Explanation

- **Sum Accumulation:** As the list is traversed, the sum of the current `step` elements is accumulated.
- **Average Calculation:** When the counter reaches `step`, the average is calculated and inserted into the list.
- **List Insertion:** The average value is inserted before the current position in the list.

This function assumes that the list has at least `step` elements before the first insertion. If the list is shorter, no insertion occurs until enough elements have been traversed.

You can add this function to your `main.cpp` file and replace the previous version of `insert_aggregated_data` with it.

If you need further modifications or explanations, feel free to ask!

Este prompt y la implementación de la función que me proporcionaron dieron solución al problema propuesto. Está bien observar que el modelo de lenguaje también hace un poco de explicación acerca de cómo funciona dicha implementación, que con comprensión previa del código y del problema resulta en mayor facilidad al momento de verificar si dicha función sirve para resolver el problema.

Conclusión

Teniendo las instrucciones y el objetivo claro, se hace fácil implementar una función con ayuda de una LLM o IA. Resulta bastante útil para hacer tareas específicas como esta, pero siento que si hubiera tratado de hacer una implementación de la lista enlazada, este proceso no hubiera sido tan fructífero.