Community College of Philadelphia

CSCI211 – Data Structures and Algorithms
Storm List 1

Overview

Using C++, Java, or Python, design a program that creates a singly linked list, where each node contains an object with information about a certain tropical cyclone between the years 1900-2010.

Provided with the assignment is a CSV file containing entries for the thirty costliest tropical cyclones to impact the United States mainland during this time period. Each line/entry in this CSV file contains the storm's name, year formed, category upon landfall, and damage (in billions of US dollars; unadjusted for inflation). Two notes about this file:

- The first line is a header line that contains the titles of the four fields. This should be ignored when your program reads and parses this file.
- "Hurricanes" have categories 1 through 5 (strongest); There is one record with a category of 0, indicating it was a "Tropical Storm" upon landfall.

Your program must read in the data contained in this CSV file to set the fields/attributes of the objects stored in your linked list's nodes. Using this data, your program will instantiate Storm objects and append (in one case, insert) these objects into the nodes of the linked list.

See the following pages of these instructions for details about the necessary classes/objects and an outline of what will happen in your main program.

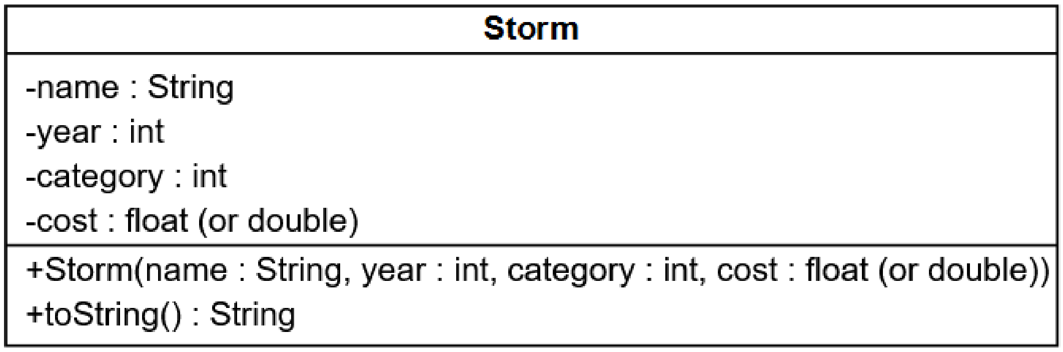
If you need a refresher on reading files and tokenizing Strings, here are some resources:

- C++
  - Reading/Writing CSV Files | Tokenizing Strings
- Java
  - Reading/Writing CSV Files
- Python
  - Reading/Writing CSV Files

Since this is text data, it will be parsed as strings so be sure to typecast numeric data to a numeric (int/float/double) datatype where necessary.

You can use the above resources or other resources for parsing the CSV file, but be sure to leave a comment crediting the source if you do. *How* your program reads the data from the CSV file into your objects isn't being graded, just that the program *is* reading the data from the CSV file into your objects.

Storm Object



Storm

-name : String
-year : int
-category : int
-cost : float (or double)

+Storm(name : String, year : int, category : int, cost : float (or double))
+toString() : String

The class for a **Storm** (can be implemented as either a struct (C++) or a class) must contain:

- Four private fields:
  - **name** (String) – The name of the storm.
  - **year** (int) – The year the storm formed.
  - **category** (int) – The storm's category.
  - **cost** (float or double) – The cost of the storm's damage in billions of dollars.
- Two public methods:
  - A **constructor** that accepts four arguments, one for each of the four fields.
  - A **toString** method that returns a string in one of the following formats, depending on the category:

    Category 1 through 5 (Hurricanes):

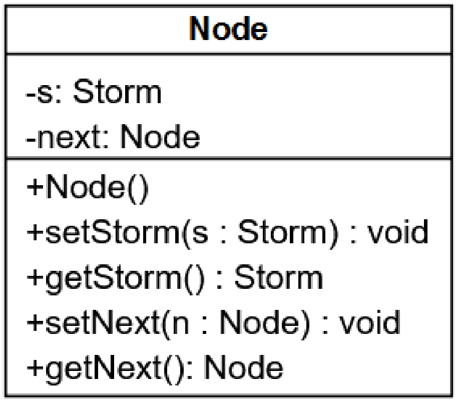
    `Hurricane` **`name`** `- Year:` **`year`**`; Category:` **`category`**`; Cost: $`**`cost`**`B`

    Category 0 (Tropical Storms):

    `Tropical Storm` **`name`** `- Year:` **`year`**`; Cost: $`**`cost`**`B`

Neither the program's output nor the string's text need to be in bold; The bold font used in the examples above are only to emphasize where the fields should be in the string that your toString() method returns.

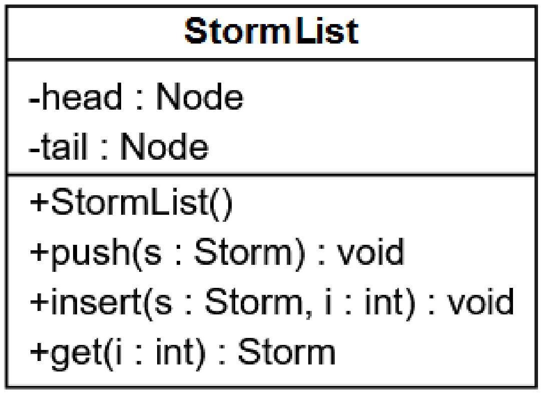
Node Object



The class for a **Node** (can be implemented as either a struct (C++) or a class) must contain:

- Two private fields:
  - **s** (Storm) – A Storm object.
  - **next** (Node) – A pointer/reference to the next Node in the list.
- Five public methods:
  - A no-argument **constructor** that initializes both fields to **null**.
  - "Setter" and "getter" methods for both fields.

<u>The Linked List (StormList Object)</u>



The class for your **StormList** linked list must contain:

- Two private fields:
  - **head** (Node) – A pointer/reference to the head Node
  - **tail** (Node) – A pointer/reference to the tail Node
- Four public methods:
  - A no-argument **constructor** that initializes both fields to **null**.
  - A **push** method that appends a Storm object to a new Node at the end of the list.
  - An **insert** method that stores a Storm object to a new Node at a specific position/index in the list.
  - A **get** method that retrieves a Storm object from an existing Node at a specific position/index in the list.

Main Program

In a main function, complete the following:

1. Instantiate a new instance of your linked list (StormList).
2. Have the program read the CSV file's data:
    a. Parse the four fields from the line/record.
    b. Instantiate an instance of a new Storm object, providing its constructor with the data parsed from the current line/record.
    c. Append the newly created Storm object to the end of the list using the StormList's **push** method.
    d. Repeat for the first 29 storms.
    e. For the last (30th) record (storm named "GLORIA"), insert this Storm object to **position/index 14** using the StormList's **insert** method.
3. Iterate through the linked list:
    a. Retrieve each Storm object from the StormList using it's **get** method.
    b. Using the Storm object's **toString**() method, print the storm's information.
    c. Repeat for all Storm objects stored in the linked list.

You can do all the above in the main method or break it up into separate functions that the main method calls.

Other Items

- **Important: You may not use a List, ArrayList, Vector, or similar data structure built-in to your chosen programming language in place of the StormList class. You must implement a linked list and its necessary components (Node and Storm classes) yourself.**

- Your Storm, Node, and StormList classes/structs must closely match the UML class diagrams provided in these instructions. Slight deviations in field, method, parameter names are OK, but there is no need to include additional methods or fields in these structs/classes; Access modifiers (public/private) must be as they are seen in the UML class diagrams.
  See here if you need a refresher on UML class diagrams

- **Do not change or modify the CSV file**. I will be using the provided CSV file when I test your program.

- Be sure to use comments to adequately document your source code.

Grading

See Assignment Rubric in Canvas.