Project 1D – Candidate List: Code Check

This part is to check your implementation. Do **NOT** simply correct your code, but try and **understand WHY** it should be corrected (this is the reason why you are getting this file).

Name Header

The name header should be at the top of **EVERY file**, .h and .cpp, **except** the .txt file that contains the data. Check the instructions for part A to make sure the format is as required.

CharacterType.h

There are no includes or any other items added to the existing ones.
Your code should be only inside the class definition.

Functions are in this **order**:

1. Default constructor
2. Overloaded constructor
   (blank line)
3. setCharacterInfo
   (blank line)
4. getFirstName
5. getLastName
6. getID
   (blank line)
7. printName
8. printCharacterInfo
9. printID
   (blank line)
10. Destructor
11. private:
    All member variables listed as given.

All **string** parameters are passed by **reference** and as **const**.
All **int** parameters are passed by **value**.

Default Constructor: You should use the initializer list constructor to set the id to 0.
Do NOT initialize **firstName** and **lastName** to empty strings (“”). Why? If you are not sure, read **Review 1, slide 48**.

The following functions are **const** functions:

- getFirstName
- getLastName
- getID
- printName
- printCharacterInfo
- printID

CharacterType.cpp

Name header
#include/using as it was given.
Functions are in the same order as they appear in the .h file.
There is a **blank line** separating each function definition.

Overloaded constructor initializes all three member variables to the given values passed by the parameters.

Function **setCharacterInfo** initializes all three member variables to the given values passed by the parameters.

Function **getFirstName** has one statement only (return).

Function **getLastName** has one statement only (return).

Function **getID** has one statement only (return).

Function **printCharacterInfo** calls function **printID** and prints the person's name as shown in the instructions.

In this case, it is a good idea to call another function within the same class, because the function **printID** will print in a very specific format and the implementation does not need to be repeated in the **printCharacterInfo** function.

Function **printID** prints the ID number.
One statement only.

Destructor is empty.

CandidateType.h

Name header.
There are no includes or any other items added to the existing ones.
Your code should be only inside the class definition.
Class inherits from **CharacterType.**

Functions are in this **order**:

1. Default constructor
2. Copy constructor
   (blank line)
3. Copy assignment operator
   (blank line)
4. updateVotesByKingdom
   (blank line)
5. getTotalVotes
6. getVotesByKingdom
   (blank line)
7. printCandidateInfo

8. printCandidateTotalVotes
   (blank line)
9. Destructor
   (blank line)
10. private:
    All member variables listed as given.

All **object** parameters are passed by **reference** and as **const**.
All **int** parameters are passed by **value**.

The following functions are **const** functions:

- getTotalVotes
- getVotesByKingdom
- printCandidateInfo
- printCandidateTotalVotes

CandidateType.cpp

Name header
#include/using as it was given.
Functions are in the same order as they appear in the .h file.
There is a **blank line** separating each function definition.

Default constructor
- Initializes total votes to 0.
- Initializes numOfKingdoms to the const int provided in the .h file.
- Creates a dynamic array of capacity numOfKingdoms, where all indices are storing 0 (this can be done in two ways: Using a FOR loop or adding {0} at the end of the statement.

Copy constructor
- Syntax to pass values to the parent's constructor is as shown on the Inheritance lecture, slide 18 (values are retrieved using the accessor functions).
- Member variable totalVotes and numOfKingdoms are updated.
- A new dynamic array is created of same capacity as parameter object's array and a loop copies all values.

Copy assignment operator (also called "overloaded" assignment operator)
- Check if calling object and parameter object are the same. Make sure you are checking the addresses of the object as shown in the Big Three lecture.
- If they are the same, an error message is given using **cerr**.
- If they are different,
  - If array capacity is not the same
    - Delete parameter object's array
    - Create a dynamic array
  - Update numOfKingdoms
  - Use a loop to copy all values.

Function **updateVotesByKingdom** updates member variables with the values passed by the parameters.

Function **getTotalVotes** has one statement only (return).

Function **getVotesByKingdom** has one statement only (return).

Function **printCandidateInfo**
Calls **parent**'s functions **printID** and **printName** to print out the candidate's information.
(There should **NOT** be the class qualifier and scope resolution when calling the parent's functions. **Why?**)

Function **printCandidateTotalVotes** calls **parent**'s function **printName** and prints the rest with a **cout** statement.
(There should **NOT** be the class qualifier and scope resolution when calling the function **printName**. **Why?**)

Destructor should delete dynamic data and set the pointer to nullptr.
(Do not re-set automatic member variables.)

CandidateList.h

Name header.
There are no includes or any other items added to the existing ones.
Class **Node** is as it was given.
Your code should be only inside the class definition.

Functions are in this **order**:

1. Default constructor
   (blank line)
2. addCandidate
   (blank line)
3. getWinner
   (blank line)
4. isEmpty
5. searchCandidate
   (blank line)
6. printCandidateName
7. printAllCandidates
8. printKingdomVotes
9. printCandidateTotalVotes
   (blank line)
10. printFinalResults
11. clearList
12. Destructor
    (blank line)
13. private:
14. searchCandidate
    (blank line)
    All member variables listed as given.

The parameter **object** passed by the function **addCandidate** is passed by **reference** and as a **constant**.
All **int** parameters are passed by **value**.
The **pointer** in the **private searchCandidate** function is passed by **reference**.

Default Constructor: You should use the initializer list constructor to set the pointers to nullptr and the count to zero.

The following functions are **const** functions:

- getWinner
- isEmpty
- searchCandidate (both functions)
- printCandidateName
- printAllCandidates
- printKingdomVotes
- printCandidateTotalVotes
- printFinalResults

CandidateList.cpp

Name header
#include/using as it was given.
Functions are in the same order as they appear in the .h file.
There is a **blank line** separating each function definition.

Function **addCandidate**

- Adds to the back of the list.
- Creates a node containing information passed by the parameter.
- Does **NOT** traverse the list, because it uses the pointer **last** to add to the back of the list.
- Updates count.

Function **getWinner**

- If list is empty use cerr to print statement as shown in the instructions.
- Else traverse the list to look for the highest total votes.
- **Both** blocks (IF/ELSE) **must** have a return value, or there must be a return **OUTSIDE** the **IF/ELSE** block. If you did not do so, then you should have a **Warning**. Check it, because you need to learn to pay attention to warnings. What is a **warning**? It VS telling you that this is working right now, but it might not work under other conditions.

Function **isEmpty** should be one statement only (return).

Function **searchCandidate (public function)**

- **Two** statements only:
  - Declares and initializes a **pointer** to a **Node** object (**ALWAYS** initialize your variables when you declare them).
  - Function returns a Boolean value: This value is given by a function call to the **private** function **searchCandidate**.

Function **printCandidateName**

- Declares and initializes a **pointer** to a **Node** object (**ALWAYS** initialize your variables when you declare them).
- **IF** (call to **private** function **searchCandidate** to retrieve the node address)
    - Using the updated address, calls to function **printName** to print.

Function **printAllCandidates**

IF block:

- Checks if the list is empty.
- Uses **cerr** to print an error message as described in the instructions.

ELSE block:

- Declares a pointer to traverse the list
- **while** loop to traverse the list and print information

Function **printKingdomVotes**

- Declares and initializes a **pointer** to a **Node** object (**ALWAYS** initialize your variables when you declare them).
- **IF** (call to **private** function **searchCandidate** to retrieve the address of the node)
    - Calls function **getVotesByKingdom** to print the votes and prints the kingdom names.

Function **printCandidateTotalVotes**

- Declares and initializes a **pointer** to a **Node** object (**ALWAYS** initialize your variables when you declare them).
- **IF** (call to **private** function **searchCandidate** to retrieve the address of the node)
    - Call to function **getTotalVotes** to print the votes.

Function **printFinalResults**

IF block:

- Checks if the list is empty.
- Uses **cerr** to print an error message as described in the instructions.

ELSE block:

- Prints table header and each candidate in order of highest to lowest votes.
- What you need to check:
    - Function **setw** was used (**not** spaces or tabs).
    - List was **NOT** sorted (list should be in the same order as it was when it was populated).
    - Highest vote is set to the vote of the first candidate vote, **NOT** some random high number or a MAX number.
    - There is **NO** call to searchCandidate or getWinner. All search should be within the function implemented with loops.
    - You did not create a candidate anywhere in the function.
    - Once the winner is found, the rest of the candidates should be found according to this criteria:
        - You look for the next highest vote making sure that you do not exceed the previous highest vote – 1. **Why?** We assumed that there are **no** ties. Therefore, if the highest vote is 78, the next highest vote cannot exceed 77, which means that your loop should stop if highest vote – 1 is reached.
        - Here is an **example**. Let's assume the total votes are like this:

45 89 36 88 35 72

- First loop finds 89 (this loop goes all the way through the list), which is the winner.
- Remember that there are no ties.
- If you let the second loop go all the way through, it will find 88 as the highest. This is inefficient. If 89 was the highest in the previous loop, the next highest cannot be higher than 88; therefore, if you find 88 before getting to the end of the list, you should stop (and for this reason you need a **WHILE** loop, because you should not break a FOR loop).

Next semester you will learn efficient algorithms to sort lists. ☺

Function **clearList**

- Declares a pointer to traverse the list.
- A **loop** deletes each node in the list.
- **Outside** the loop:
  - The **count** is updated.
  - The **first** and **last** pointers are set to **nullptr.**

Destructor: Calls function **clearList**

Function **searchCandidate** (**private function)**

The function returns a **Boolean** value and passes two parameters, an **int** storing an ID number and a **pointer** to retrieve the location of the candidate. **NOTE that the pointer is passed as reference.**

IF block:
- Checks if the list is empty.
- Uses cerr to output empty list message

ELSE block:
- Uses the **parameter pointer** to traverse the list.
- Uses a **while** loop to traverse the list and print information.
  - The loop ends (or returns) when it reaches the end of the list OR when the ID is found.
- If the ID is not found, prints an error message as indicated in the instructions using **cerr.**

Both blocks (IF/ELSE) **must** have a return value, or there must be a return **OUTSIDE** the **IF/ELSE** block. If you did not do so, then you should have a **Warning**. Check it, because you need to learn to pay attention to warnings. What is a **warning**? It VS telling you that this is working right now, but it might not work under other conditions.

Main.cpp and **InputHandler.h**

Nothing has been modified in these files.

OTHER

There is **NO horizontal scrolling**. All statements are **at most** 70 **col** (NOT ch).
All code is properly indented.
All **variables** and **objects** have a **descriptive** identifier.
All **variables** and **objects** use the **camelCase** convention.

All **variables** have been **initialized**.
No **unnecessary** variables were created.