**Project 2: STL Candidate List**

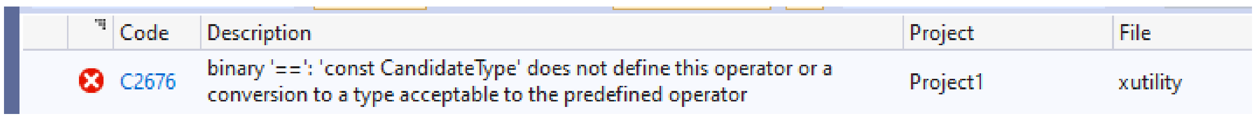
For this project, you will change the implementation of **Project 1** so that the list of candidates is an **STL list** instead of a linked list.

Follow the instructions below to modify the code.

| **CandidateList.h** | |
|---|---|
| **#include** | Include the **STL <list>** class. |
| Class **Node** | Delete the class **Node**. |
| **Member variables** | You will **not** need the *first* and *last* pointer any longer, and you will **not** need the variable count either. Delete all member variables and declare an **STL list** of **CandidateType** named **candidates**. |
| **clearList** | You have no dynamic variables; therefore, you can remove this function. |
| **NOTE:** Before you move onto modifying the implementation of the class, make sure you keep in mind the following: <br>• Function **size** of the **STL list** class does **NOT** return an **int**; therefore, you will need to cast the function using **static_cast OR** you will need to use either type **size_t** or **unsigned_int**. <br>• You will be eliminating several **parentheses**. If you get errors you cannot identify, check for **unnecessary parentheses** on the function where you get the error and the function **above** as well. <br>• You may use **auto** for iterators (**not** for other variables), but remember that you will **not** be able to use auto for parameters. Also, make sure you **consider** whether the iterator should be a **const iterator**. | |
| **CandidateList.cpp** | |
| **clearList** | Remove the definition of the function **clearList.** |
| **Destructor** | Since there are no dynamic variables, the destructor will be empty. |
| **Constructor** | It will be empty, because the **STL list** has its own constructor that initializes to an empty list. |
| *** | Remove all error statements that indicate whether the list is empty. You will be using the **isEmpty** function in the **Main.cpp** file to verify whether there are elements in the list. <br><br>From now on, you will always **assume** that the **list** is **non-empty**. |
| **addCandidate** | Remove all code and simply use the function **push_back** of the **STL list** to insert a new candidate. <br>**Review:** Why it is not necessary to resize an **STL list** when using |

| | |
|---|---|
| | **push_back,** but it is necessary to do so for an **STL vector**? |
| **isEmpty** | Modify the implementation so that it works with the **STL list**.<br><br>There should be one statement only. |
| **searchCandidate (public)** | Declare an **STL list iterator** and initialize it to the first object in the list (you will be modifying the code so that it is assumed that the **list is non-empty**).<br><br>Call the **private** function **searchCandidate**, passing the id parameter and the iterator you just created. |
| **searchCandidate (private)** | **Parameters**: Replace the pointer parameter with an **STL list iterator**. (Make sure you change the function declaration as well.)<br><br>Remove the **IF** statement that checks whether the list is empty. You will be using the **isEmpty** function in the **Main.cpp** file to verify whether there are elements in the list.<br><br>You will **NOT** need a loop, and you will **NOT** need the Boolean variable either. Use instead the **STL algorithm find**. |
| **getWinner**<br>**printCandidateName**<br>**printAllCandidates**<br>**printKingdomVotes**<br>**printCandidateTotalVotes**<br>**printFinalResults** | Most of the code will stay the same, but you will need to make a few syntax modifications because you are not using a customized linked list any longer.  To traverse the **STL list**, you are **NOT** going to use pointers, but you will need to use **iterators**. If the function is a **const** function, you **MUST** use a **const iterator**—you can use **auto**, if you prefer. |

Run your program. You should get an error similar to the one shown below (if you get other errors, then you need to fix your code first).



| | Code | Description | Project | File |
|---|---|---|---|---|
| ❌ | C2676 | binary '==': 'const CandidateType' does not define this operator or a conversion to a type acceptable to the predefined operator | Project1 | xutility |

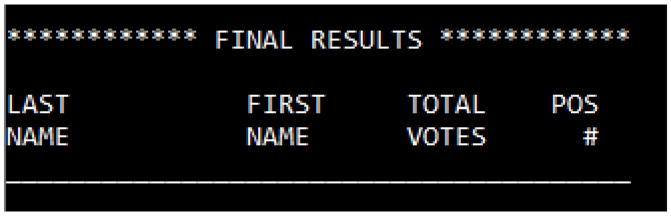
We talked about this in class. The **STL algorithm function find** will try to compare elements in the **STL list**. If the elements in the list were primitive types, such as **int** or **double** or similar, then there would be no problem, but because the elements in the **STL list** are objects of type **CandidateType**, function **find** cannot access the ID in the candidate object.

**So, what should you do?**

The **STL algorithm function find** will go through the list, checking each candidate, trying to compare an ID to a candidate. What do you *really* want to compare? You want to compare the given ID to the ID *inside* the **CandidateType** object. Therefore, since function find uses a comparison operator (==), you need to overload the comparison operator to compare the given ID to the ID of the object in the list.

**Cleaning up function printFinalResults…**

A good coding practice is to keep functions short, performing one task only. The **printFinalResults** function is quite long and does too many things. You will extract some of the code to clean up the function.

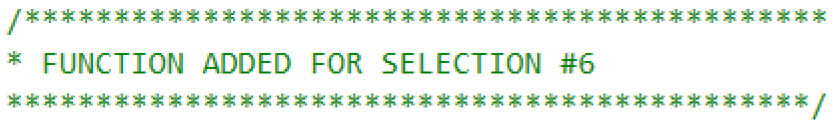
| **CandidateList.cpp (adding more functions)** | |
|---|---|
| **printHeader** | This is a **private** function. Why? This is a function that will be used *only* by a member function; therefore, there is no need to make it public.<br><br>Remove from the **printFinalResults function** all the code that prints the header of the table, and paste it in the **printHeader** function. Add a function call in place of the code that was removed.<br><br><br>`************ FINAL RESULTS ************`<br>`LAST            FIRST      TOTAL    POS`<br>`NAME            NAME       VOTES      #`<br>`_________________________________________` |
| **printCandidate** | This is a **private** function.<br><br>**Parameters:** An iterator pointing to a candidate in the list and an **int** storing the ranking position of the candidate.<br><br>The function should print the candidate to which the iterator parameter is pointing. You will use this for **ALL** candidates, including the winner.<br><br>Of course, you will need to add the appropriate function calls in the **printFinalResults** function. |

**Add a selection of your choice**

For this part, you need to add an **additional selection**, which will become **#6**, moving the **exit** selection to #7. This can be **anything you choose** that would **enhance the menu**. You will obviously need to:

- **Add** the **selection** to the **menu**.
- **Modify** the **switch** statement in the **processChoice** function.

- **Maintain** the **same format** as the other choices (spacing, indentation, and other details).
- **Test** it.
- Depending on what you are adding, you might need to create more than one function in different section of the program. **LABEL** your function(s) by adding the comment shown below **BEFORE** (1) the function **declaration** and (2) the function **definition**.



```
/*********************************************
* FUNCTION ADDED FOR SELECTION #6
*********************************************/
```

**Optional:** You may change the background and/or text color, if you wish. This is how to do it:

**system("Color xy");**

// x : background color
// y : text color

**Possible colors:**

| | |
|---|---|
| 0 = Black | 8 = Gray |
| 1 = Blue | 9 = Light Blue |
| 2 = Green | A = Light Green |
| 3 = Aqua | B = Light Aqua |
| 4 = Red | C = Light Red |
| 5 = Purple | D = Light Purple |
| 6 = Yellow | E = Light Yellow |
| 7 = White | F = Bright White |

**Examples:**

"Color 12" = Blue background/Green text
"Color 4F" = Red background/Bright white text
"Color B1" = Light aqua background/Blue text