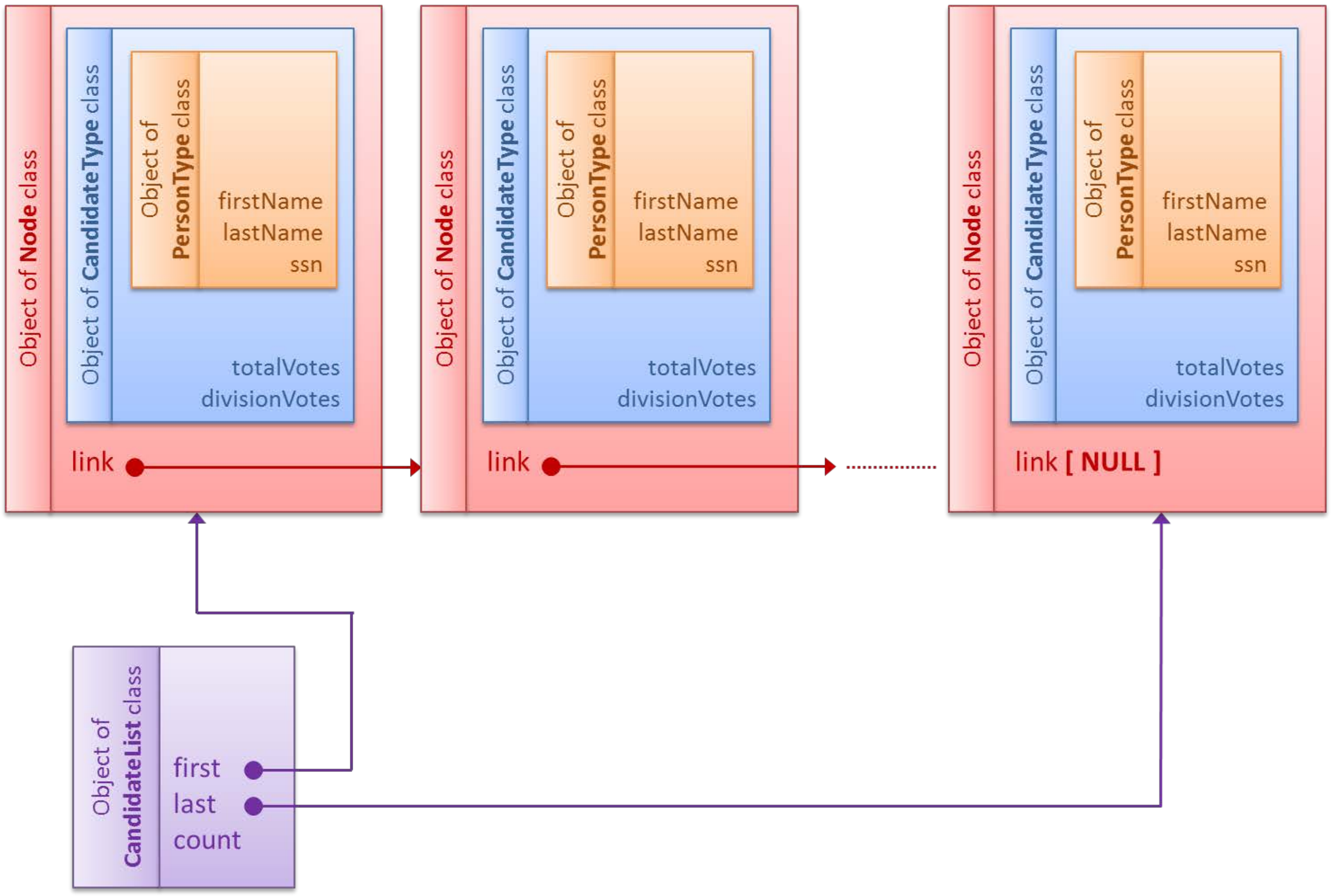
Project 1 (Part C) - Candidate List

For this part of the project, you will need to add to your project all the files in the **p1_c_candidate_list_files** folder. Directions on **how to import files into a project** are available on Canvas ➔Modules/References/How-to Files.

CANDIDATE LIST

You need to complete the class **CandidateList** that creates a **singly-linked list** of nodes containing objects of the class **CandidateType** and a pointer to the next node.



Import into your project the following files:

- CandidateList.h
- CandidateList.cpp
- candidate_data.txt
- InputHandler.h
- Main.cpp (this will replace your old Main.cpp file)

candidate_data.txt

The **candidate_data.txt** file should be placed in the **Resource Files** folder of your project. The file contains a list of candidates to add to the linked list your program creates. Each line contains a **social security number**, a first name, a last name and four integers indicating the votes by division (first integer for division 0, second integer for division 1, and so on):

```
        123456789 Donald Duck 89 34 45 5
```

The file ends with "**-999**" to stop the loop when reading the data.

InputHandler.h

The **InputHandler.h** reads data from the **candidate_data.txt** file and inserts it in the list of candidates. It first checks if the file is available and the data can be read; if not, it will terminate the program.

The function **createCandidateList** creates objects of class **CandidateType** and it stores them in the list by calling the function **addCandidate** of the class **CandidateList**.

Although the implementation of this file is complete, do **NOT** dismiss it! Pay careful attention to the function **createCandidateList** to understand how everything is inserted in the list.

Main.cpp

The **Main.cpp** creates the menu and all selections associated with it, to allow the user to select one of the following:

1. Print all candidates
2. Print a candidate's division votes
3. Print a candidate total vote
4. Print winner
5. To exit

Although the implementation is completed, you should trace it to see what it does and how it connects everything.

IMPLEMENTATION

The **CandidateList interface** already has a **class Node** that creates **nodes** storing a **CandidateType object** and a **pointer link** that points to the next node. The file also includes the partial definition of the **class CandidateList**, which creates objects that contain a **pointer first** to point to the first node in the list, a **pointer last** to point to the last node in the list, and an **int count** to keep track of the number of nodes in the list.

You will need to implement the class **CandidateList** as indicated below.

CandidateList Class	
Member variables	A pointer that points to the first node. A pointer that points to the last node. An integer variable that stores the number of nodes.
Default constructor	Initializes all member variables.
Function **addCandidate**	**Parameters:** An object of the **CandidateType** class. Inserts nodes to the *back* of the list. You have a pointer pointing to the back of the list; therefore, there is **NO** need to traverse the list.
Function **getWinner**	Traverses the list to find the candidate who has the highest number of votes, and returns the social security number associated with that

	candidate. If the list is empty, output the error message " => List is empty." and return 0.
Function **searchCandidate**	**Parameters:** A social security number. Traverses the list to find the candidate with the given social security number and returns true if the candidate is found and false otherwise. Use a **while** loop so that you can **stop** the loop when the candidate is found ➔You are **NOT** allowed to use "break" or "continue" If the list is empty, output the error message " => List is empty." If the candidate was *not* found, output the error message " => SSN not in the list."
Function **printCandidateName**	**Parameters:** A social security number. Traverses the list to find the candidate with the given social security number and prints out the name using the **printName** function of the **PersonType** class. Use a **while** loop so that you can **stop** the loop when the candidate is found ➔You are **NOT** allowed to use "break" or "continue" If the list is empty, output the error message " => List is empty." If the candidate was *not* found, output the error message " => SSN not in the list."
Function **printAllCandidates**	Traverses the list to print all candidates using the **printCandidateInfo** function of the **CandidateType** class. If the list is empty, output the error message " => List is empty."
Function **printCandidateDivisionVotes**	**Parameters:** A social security number and a division number. Prints out all the division votes for a given candidate, using the **getVotesByDivision** function of the **CandidateType** class. Use a **while** loop so that you can **stop** the loop when the candidate is found ➔ You are **NOT** allowed to use "break" or "continue" If the list is empty, output the error message " => List is empty."
Function **printCandidateTotalVotes**	**Parameters:** A social security number. Traverses the list to find the candidate with the given social security number and prints out the total number of votes using the **getTotalVotes** function of the **CandidateType** class. Use a **while** loop so that you can **stop** the loop when the candidate is found ➔ You are **NOT** allowed to use "break" or "continue" If the list is empty, output the error message " => List is empty."
Function **destroyList**	Traverses the list to delete each node and reset all member variables to their default value.
Destructor	Calls the function **destroyList**.

The **Main.cpp** file reads from the **candidates_data.txt** file and calls the appropriate functions to insert each candidate in a list. It also displays a menu for the user to make selections.

ASSUMPTIONS

- Social security numbers are **unique**.
- **No** candidates have the same number of total votes; there are **no** ties.

EXPECTED OUTPUT

The **output.exe** file is your reference to compare results and format with the output of your own project.

TESTING

The **folder p1_c_output_exe** contains an **executable file** and a **data file** to compare the expected output to the output generated by your project.

Which **test cases** should you choose?

- Selection 1
- Selection 2
  - The first candidate in the text file
  - The last candidate in the text file
  - A candidate in between
  - A social security number not in the database
- Selection 3
  - (same as Selection 2)
- Selection 4
- Selection 5
- Any other integer that is not part of the selection menu.

Please note that **most of the grading** for **Project 1** will be heavily based on the **output**—an extra space, a missing space, a missing period, an extra line, a missing line, a typo, etc. will **each** count **1 point**. Your output **MUST** look **exactly** as the one generated by the **output.exe** file given.