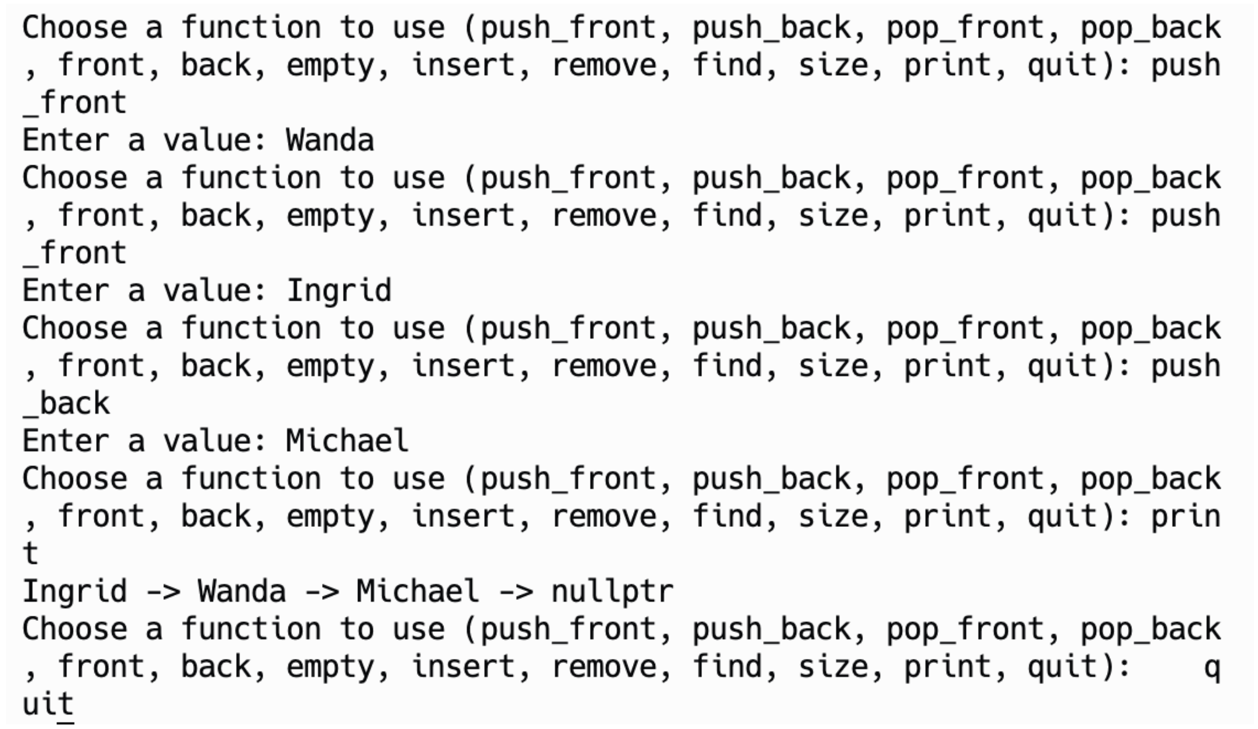
Abby Gerstner
CS 303 Assignment 2 Readme File

How to use Assignment 2 CS 303 Single Linked List:

1. Open SingleLinkedList.h and SLL_main.cpp files.
2. Run the program.
3. Type the function you want to use (push_front, push_back, pop_front, pop_back, front, back, empty, insert, remove, find, size, print, quit) in order to build/modify your list.
4. Press Enter
5. For push_front or push_back, type the value to be pushed onto the list either to the front or back. For pop_front or pop_back, simply press enter and the function will be completed. For front or back functions, press enter and the front or end of the list will be printed. Empty will return “Yes” if empty or “No” if not empty. Insert will ask for a value to be inserted and the index you want to insert to the list. Remove will ask for the index of a value to remove. Find will ask to type a value, and then look for that value in the list. Size will return the size of the list. Print will print the list.
6. The program will ask again which function you want to use.
7. When finished, type “quit” and press enter to quit the program.

Example output of building a list:



```
Choose a function to use (push_front, push_back, pop_front, pop_back
, front, back, empty, insert, remove, find, size, print, quit): push
_front
Enter a value: Wanda
Choose a function to use (push_front, push_back, pop_front, pop_back
, front, back, empty, insert, remove, find, size, print, quit): push
_front
Enter a value: Ingrid
Choose a function to use (push_front, push_back, pop_front, pop_back
, front, back, empty, insert, remove, find, size, print, quit): push
_back
Enter a value: Michael
Choose a function to use (push_front, push_back, pop_front, pop_back
, front, back, empty, insert, remove, find, size, print, quit): prin
t
Ingrid -> Wanda -> Michael -> nullptr
Choose a function to use (push_front, push_back, pop_front, pop_back
, front, back, empty, insert, remove, find, size, print, quit):    q
uit
```

Insert function example:

```
Kay -> Bea -> Michael -> Ingrid -> nullptr
Choose a function to use (push_front, push_back, pop_front, pop_back
, front, back, empty, insert, remove, find, size, print, quit): inse
rt
Enter a value: 4
Enter an index: 0
Choose a function to use (push_front, push_back, pop_front, pop_back
, front, back, empty, insert, remove, find, size, print, quit): prin
t
4 -> Kay -> Bea -> Michael -> Ingrid -> nullptr
```

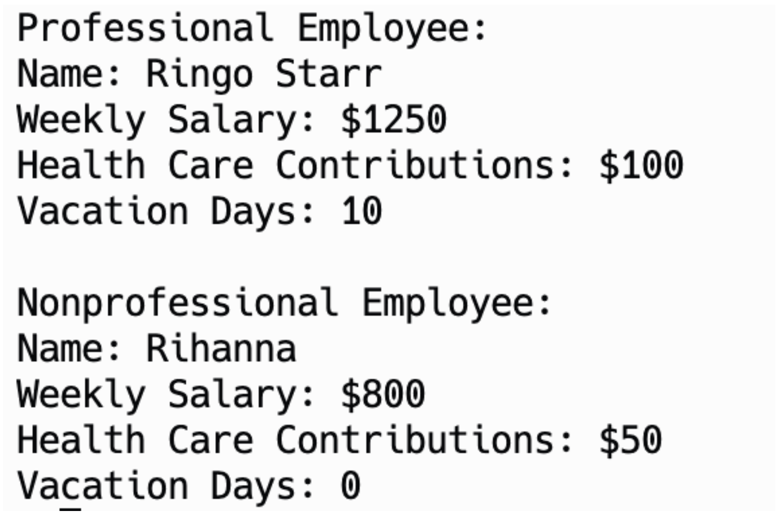
Assumptions:

1. String Data: The code assumes that the data stored in the linked list is of type std::string. It is not designed to handle other data types.
2. Valid Inputs: It assumes that users will provide valid inputs when interacting with the linked list through the console. For example, it expects users to enter valid strings and indices when inserting or removing elements.
3. No Memory Constraints: The code assumes that there are no severe memory constraints. It uses dynamic memory allocation (e.g., new and delete) for nodes without checking for memory allocation failures.
4. Indexing Starts at 0: It assumes that the indexing for the linked list starts at 0, where the first element is at index 0, the second at index 1, and so on.
5. Destruction of Nodes: The code assumes that the user is responsible for managing the destruction of nodes. The destructor (~LinkedList) is provided to clean up the memory used by the linked list, but it's the user's responsibility to ensure proper cleanup.
6. Valid Choice in Main: In the main function, it assumes that the user will enter a valid choice when prompted for an action (e.g., "push_front," "pop_back," etc.). Invalid choices will result in an error message.
7. Positive Indices: When using the insert and remove functions, the code assumes that users will provide positive indices. Negative indices are not supported.
8. No Duplicate Values: The code does not explicitly handle duplicate values in the linked list. It may find the first occurrence of a value when searching with the find function.
9. Size of LinkedList: The size function returns the number of items in the linked list but does not check for integer overflow. It also counts nullptr as an index of the list.
10. No Memory Leaks: It assumes that the user will not intentionally create memory leaks by not properly destroying the linked list or nodes.
11. User Input Limits: The code does not enforce any specific limits on user input, such as the maximum length of strings or the maximum number of elements in the linked list. It assumes reasonable input sizes.
12. No Exception Handling in Main: While the code contains exception handling for certain errors, it assumes that the main function does not encounter exceptions beyond the provided error handling.

How to use Assignment 2 CS 303 Employee:

1. Open employee.h and employee_main.cpp files.
2. Run the program.
3. The console will display one case of the professional class and one case of the nonprofessional class.

Output:



```
Professional Employee:
Name: Ringo Starr
Weekly Salary: $1250
Health Care Contributions: $100
Vacation Days: 10

Nonprofessional Employee:
Name: Rihanna
Weekly Salary: $800
Health Care Contributions: $50
Vacation Days: 0
```

Assumptions:

1. Employee Class Construction: The code assumes that the "Employee" class can be constructed by providing a single parameter, which is a string for the employee's name. It doesn't handle scenarios where the name parameter is missing or of an incorrect data type.
2. Monthly Salary and Vacation Days: In the "Professional" class, it is assumed that the provided monthly salary is non-negative, and the number of vacation days is a non-negative integer. The code doesn't explicitly check for invalid or negative values.
3. Fixed Health Care Contributions: The code assumes a fixed health care contribution value of $100 for professional employees and $50 for nonprofessional employees. It does not account for scenarios where these values may change or be subject to variations.
4. Vacation Hours Earned: In the "Nonprofessional" class, it is assumed that vacation hours earned are calculated based on the rule of 1 vacation hour for every 10 hours worked.
5. Assumption of 4 Weeks in a Month: The "Professional" class assumes that a month consists of four weeks when calculating the weekly salary. This assumption may not hold true in all cases.
6. Assumption of 8 Hours per Day: The "Nonprofessional" class calculates the number of vacation days by assuming that 8 hours make a day. This assumption may not be suitable for all work contexts, where a workday may have different hour durations.
7. No Input Validation: The code assumes that the provided values for employee attributes, such as salary, vacation days, hourly rate, and hours worked, are valid and do not need further validation. It does not handle input validation or error-checking.

8. Namespace Assumption: The code assumes that the "std" namespace is in use for input and output operations since it includes <iostream> and <string> without using the "std::" prefix.
9. Compilation and Runtime Assumption: The code assumes that the program will be compiled and executed successfully without any compilation or runtime errors. It does not include error handling for potential issues.
10. Header File Assumptions: The code assumes that the "employee.h" header file is available and properly designed, containing class declarations and necessary function prototypes.
11. No Memory Management Constraints: The code assumes that there are no severe memory constraints. It uses dynamic memory allocation (e.g., new and delete) for objects without checking for memory allocation failures or implementing memory management best practices.