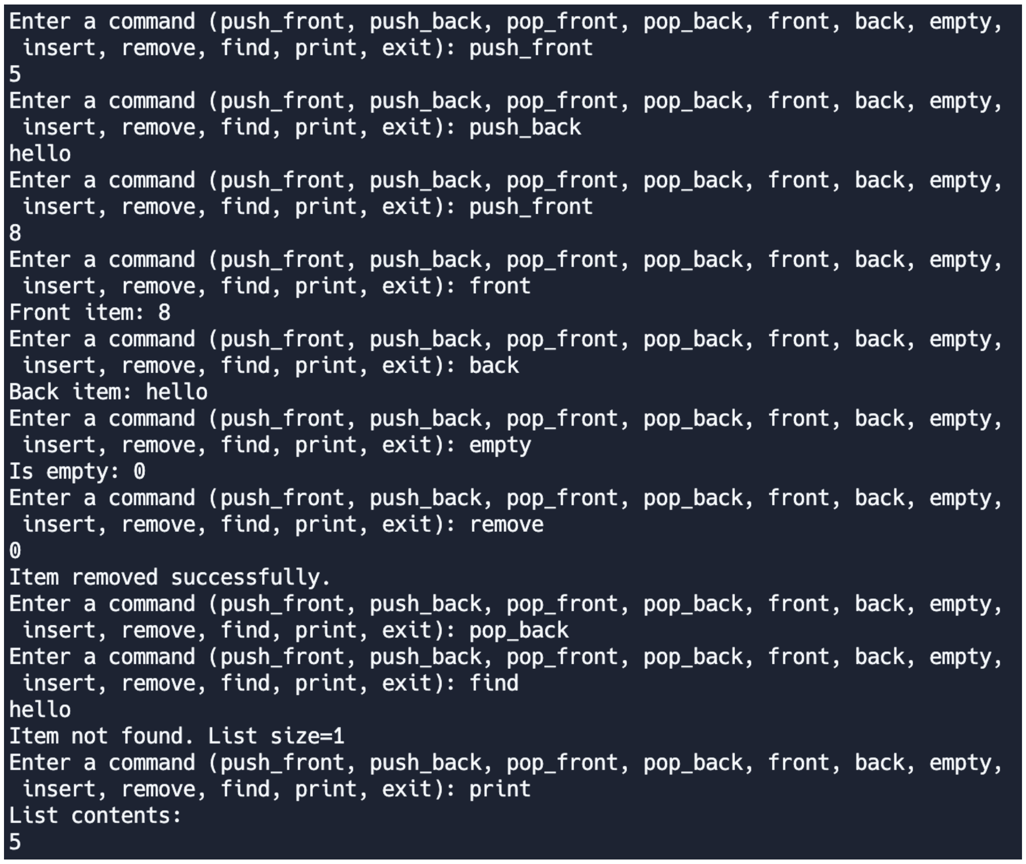
Q1 instructions:

1. Once the CPP and Header file are downloaded and in the same working directory, click "Run."
2. Next, type in any of the commands listed, (push_front, push_back, pop_front, pop_back), and click enter.
3. Next enter the data you want that command to edit the list with, for example, enter push_front, click enter, type 3, then click enter again.
4. If using any of the commands… (front, back, empty, print, exit), you will not need to enter a data value, as these commands will give output with no data value or index required.
5. When using insert, first type "insert", click enter, then type the index you'd like to insert at, click enter, and lastly type the data item you would like to insert there, and click enter.
6. When using remove, note that you are entering the index value, not the data item value to remove.
7. You can call any commands as much as you want, and you can always use the "print" command to see the current list's items.

Q1 Outputs:



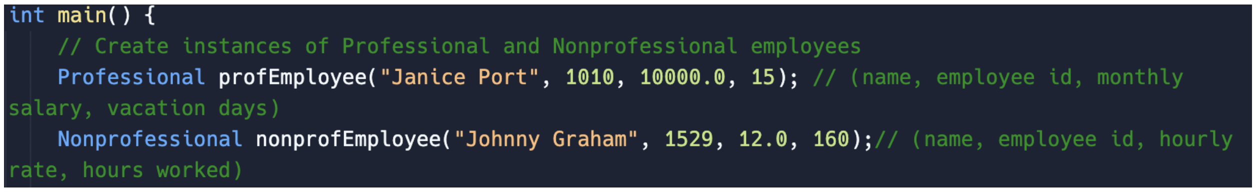
```
Enter a command (push_front, push_back, pop_front, pop_back, front, back, empty,
 insert, remove, find, print, exit): push_front
5
Enter a command (push_front, push_back, pop_front, pop_back, front, back, empty,
 insert, remove, find, print, exit): push_back
hello
Enter a command (push_front, push_back, pop_front, pop_back, front, back, empty,
 insert, remove, find, print, exit): push_front
8
Enter a command (push_front, push_back, pop_front, pop_back, front, back, empty,
 insert, remove, find, print, exit): front
Front item: 8
Enter a command (push_front, push_back, pop_front, pop_back, front, back, empty,
 insert, remove, find, print, exit): back
Back item: hello
Enter a command (push_front, push_back, pop_front, pop_back, front, back, empty,
 insert, remove, find, print, exit): empty
Is empty: 0
Enter a command (push_front, push_back, pop_front, pop_back, front, back, empty,
 insert, remove, find, print, exit): remove
0
Item removed successfully.
Enter a command (push_front, push_back, pop_front, pop_back, front, back, empty,
 insert, remove, find, print, exit): pop_back
Enter a command (push_front, push_back, pop_front, pop_back, front, back, empty,
 insert, remove, find, print, exit): find
hello
Item not found. List size=1
Enter a command (push_front, push_back, pop_front, pop_back, front, back, empty,
 insert, remove, find, print, exit): print
List contents:
5
```

Q1 Assumptions:

1. Valid Inputs: The code assumes that users will provide valid inputs when interacting with the program. For example, when specifying an index for inserting or removing an item, it assumes that the index is within a valid range.
2. String Inputs: The code assumes that item inputs are of type std::string. It does not handle other data types.
3. Positive Indexes: The code assumes that index values for inserting and removing items are non-negative integers. Negative index values are not handled.
4. Menu-Driven Interaction: The code assumes that users will follow the menu-driven interaction by entering specific commands as instructed in the program's menu.
5. Numeric Size: When the program reports the size of the list, it assumes that the size can be represented by a positive integer and does not handle exceptionally large sizes.
6. No Duplicate Items: The code does not handle duplicate items in the list. If you insert an item that already exists, it will not be removed automatically.
7. Empty String as Default: When calling front() and back() functions on an empty list, the code assumes that an empty string "" is an appropriate default return value.
8. Static Default Values: The code uses static default values for certain error cases, such as an empty list when calling front() and back().
9. Memory Management: The code assumes that memory allocation and deallocation for nodes in the linked list will be successful. It does not handle memory allocation failures.
10. Command Validity: The code assumes that users will only enter valid commands from the menu and will not attempt to execute invalid or unsupported commands.
11. Single-Linked List: This code is specifically designed for a single-linked list. It does not handle double-linked lists or other data structures.
12. Exit Command: The code assumes that users will use the "exit" command to gracefully exit the program.

Q2 instructions:

1. Once the CPP and all 3 Header files are downloaded and in the same working directory, in the CPP file, navigate to the main function.
2. At the top of the main function, create instances of profEmployee and nonprofEmployee as you wish. Below, 2 examples are provided.

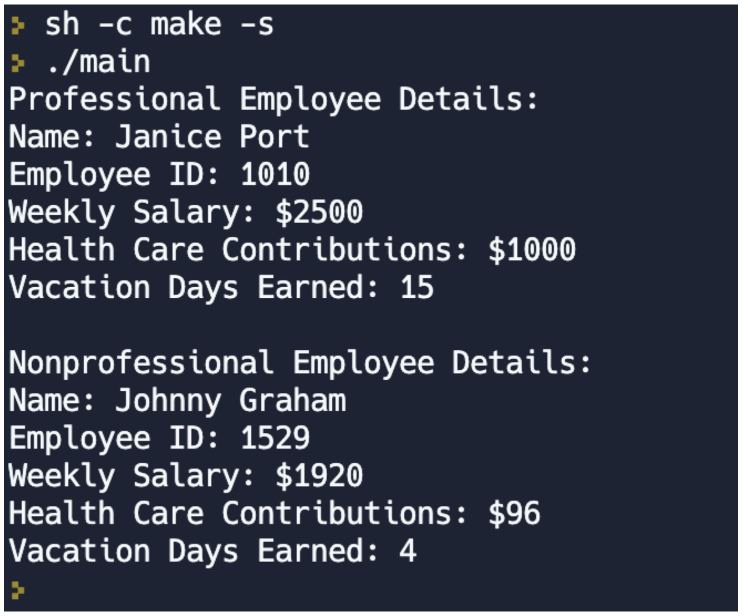


```
int main() {
    // Create instances of Professional and Nonprofessional employees
    Professional profEmployee("Janice Port", 1010, 10000.0, 15); // (name, employee id, monthly
salary, vacation days)
    Nonprofessional nonprofEmployee("Johnny Graham", 1529, 12.0, 160);// (name, employee id, hourly
rate, hours worked)
```

   - Note that profEmployee class instances take arguments (name, employee id, monthly salary, vacation days) while nonprofEmployee class instances take arguments (name, employee id, hourly rate, hours worked).
3. Once you have created desired objects, click run.

4. Upon clicking run, outputs for each data object will be output.

Q2 Outputs:



```
› sh -c make -s
› ./main
Professional Employee Details:
Name: Janice Port
Employee ID: 1010
Weekly Salary: $2500
Health Care Contributions: $1000
Vacation Days Earned: 15

Nonprofessional Employee Details:
Name: Johnny Graham
Employee ID: 1529
Weekly Salary: $1920
Health Care Contributions: $96
Vacation Days Earned: 4
›
```

Q2 Assumptions:

1. Valid Inputs: The code assumes that users will provide valid inputs when creating instances of Professional and Nonprofessional employees. Invalid inputs for attributes such as monthly salary, hourly rate, vacation days, and hours worked are not handled.
2. Monthly Salary Assumption: For Professional employees, it assumes that the monthly salary is provided, and the weekly salary is calculated based on a month having 4 weeks. It doesn't account for variations in the number of days in a month or leap years.
3. Health Care Contribution Calculation: For both Professional and Nonprofessional employees, the code assumes a fixed percentage (10% for professionals and 5% for nonprofessionals) for health care contributions. This percentage is not configurable or dynamic.
4. Vacation Days Calculation: For both types of employees, the code assumes a simple calculation for vacation days (e.g., 1 vacation hour for every 40 hours worked for nonprofessionals). It does not handle more complex vacation policies.
5. Assumption of 4 Weeks in a Month: The code assumes that there are always 4 weeks in a month when calculating weekly salaries for professional employees. This might not be accurate for all situations.
6. Fixed Hourly Rate for Nonprofessionals: For nonprofessional employees, the code assumes a fixed hourly rate. It does not handle scenarios where the hourly rate may change over time.
7. Default Constructor Values: The code assumes that the default constructor for the Employee, Professional, and Nonprofessional classes sets reasonable initial values for attributes such as name and employee ID. It does not explicitly handle default values.

8. Static Attribute Values: The code does not allow dynamic changes to employee attributes after object creation. Employee attributes are set during object creation and assumed to remain constant.
9. No Input Validation: The code does not perform extensive input validation or error checking for user inputs. It assumes that users will provide valid inputs.
10. No Exception Handling: The code does not handle exceptions or errors that may occur during execution, such as memory allocation errors.
11. Hard-Coded Data: The code uses hard-coded data for creating instances of Professional and Nonprofessional employees. It does not interactively accept user inputs for employee data.
12. Instance size: This code assumes that one and only one instance of each subclass, profEmployee and nonprofEmployee, is defined at a time. Multiple instances can be tracked, however only the first of each subclass will be displayed.