Design Document

- Overview of Class:

  Class: Node

  Class Node defines the previous, and next node using Node pointer type, and a string type data that stores the URL and URL name added to the program. The previous and next nodes are the reference to the data’s address.

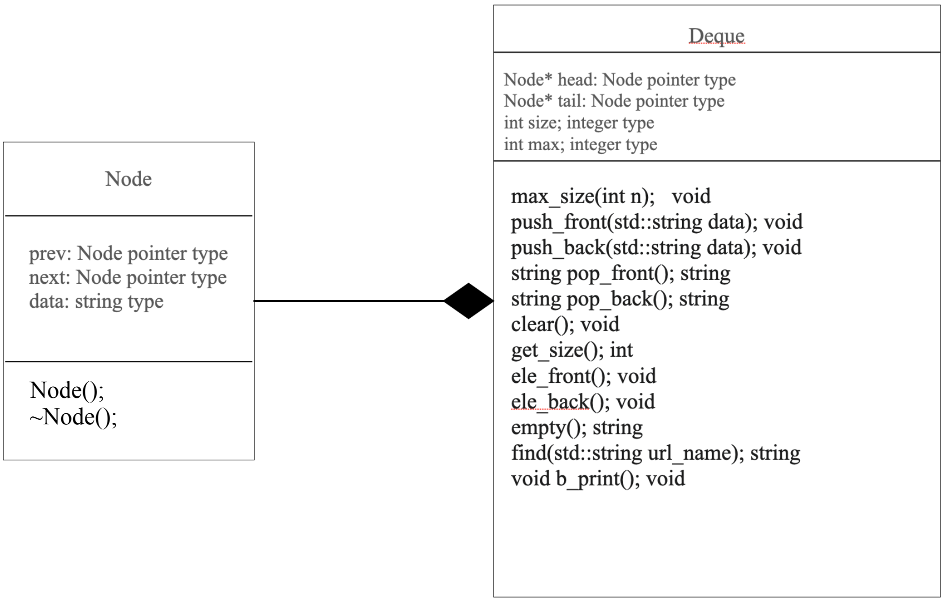
  Class: Deque:

  Class Deque creates a dynamic dequeue using a doubly linked list. Deque class stores data using the Node class. Besides, it can insert and delete elements at the front and the back of the dequeue. It also prints, clears and gets the size of the dequeue if needed.

  Relate:

  Class Node is used in the class Deque to store data, and provide previous, and next node reference to all elements in the linked list.

- UML diagram:



- Details on design decision

  Class Node:

  One constructor with no parameter and one destructor is created.
  Reason: initialize nodes prev and next to nullptr;

  Class Deuce:

  One constructor with no parameter and one destructor is created.
  Reason: initialize nodes head and tail to nullptr; set the initial value for max, and size to 0.
  Provide your rationale for any operators that you need or decide to override
  “=“ is used to assign values to the variable.
  “->” is used to access the class variable such as prev, next, and data with pointers

  Function parameter:

  Function without parameter: Deque(); ~Deque(); std::string pop_front(); void clear(); int get_size(); void ele_front(); void ele_back(); std::string empty(); void b_print(); std::string pop_back();

  Function with parameter(s):

  void max_size(int n);

The parameter n is passed by reference since users need to decide the size of the dequeue by inputting the size.

void push_front(std::string data);

The parameter data is passed by reference since users need to input the URL and the name they want to push to the front of the dequeue.

void push_back(std::string data);

The parameter data is passed by reference since users need to input the URL and the name they want to push to the end of the dequeue.

std::string find(std::string url_name);

The parameter data is passed by reference since users need to input the URL and the name they want to find in the dequeue.

- Test Cases

Class Node:

Since Node is a composition of Deque, I decide to test functions in class Deque to see whether the nodes work

Class Deque:

When finishing each member function in class Deque, I test the function.
In push_front and push_back: check whether the data is pushed to the correct position and whether the last (first) data will be deleted when the linked list is full while adding data.
pop_front and pop_back: check whether data in the front and end will be deleted. Can also be checked using push_front and push_back.
Print and clear: check whether print can print all the data from the back. Print can also check whether the linked list is empty after using the clear() function. If it is empty, then it shows that the clear function works.
Find: check whether the output is correct or not when the URL to be found is(not) in the list.
Exit: check whether the program will end when the command “exit” is typed in the terminal. I used a while(cmd != “exit”) to accomplish the exit command.

- Performance Consideration

The find, clear, and print functions have an upper bound of O(n).

In the find function, I used a for loop to traverse the linked list, and check whether the data in each element is the same as the input data. With a linked list of size n, the time needed to do a for loop is n. Therefore, The upper bound of the find is O(n).
In the clear function, I use a while loop to delete the linked list from the head, then set the next node to be the head. A dequeue of size n will be deleted n times. Therefore, the runtime is O(n).
In the print function, I use a for loop to print the data in the deque from the back. A dequeue with size n has n elements to be printed out. Therefore, the upper bound will be O(n).

The push_front, push_back, pop_front, pop_back, max_size, get_size, ele_front, ele_back, empty functions have a upper bound of O(1).

There are no loops in these functions and each step takes a time of 1. Therefore, the upper bound is O(1).
push_front, push_back: Check whether the head is nullptr using if-else. Each step use T = 1. assign two nodes to each element and data to the element each with T = 1.
pop_front, pop_back: use if-else, and delete data with T=1.
max_size: assign the member variable max with n T =1.
get_size: return the member variable size.
ele_front, and ele_back functions return the first and last element data using T = 1.
The empty function checks whether the member variable size is 0. It returns a different string when the size is 0 or not 0. Therefore, the time used is O(1);