Design Document

Overview

For this project, I designed 4 classes:

MyList

MyList is a templated linked list that offers functionalities of a deque.

MyNode

MyNode is a private struct nested within MyList. It acts as a node that stores data in MyList.

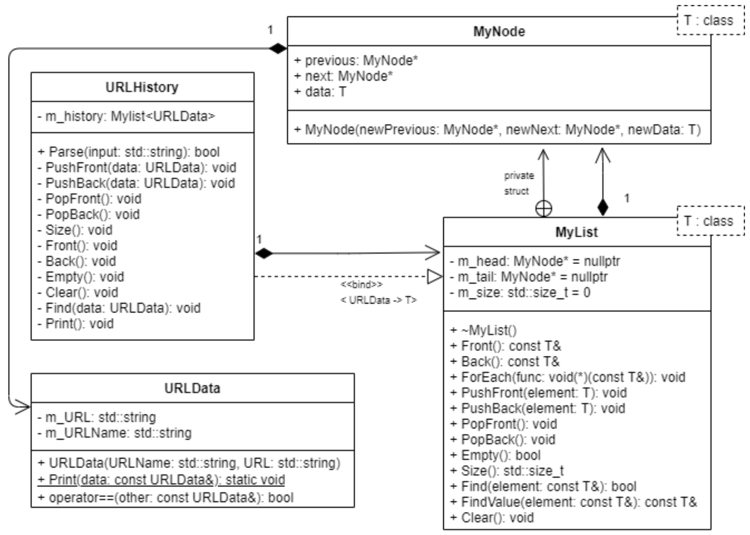
URLHistory

The URLHistory class contains a list of URLData (implemented using MyList), and acts as a driver for all functionalities related to this list (including parsing inputs and printing outputs).

URLData

The URLData class stores information of a single URL (URLName and URL) and decides how to compare URLs.

UML Class Diagram


1
MyNode
T : class
+ previous: MyNode*
+ next: MyNode*
+ data: T
+ MyNode(newPrevious: MyNode*, newNext: MyNode*, newData: T)
URLHistory
- m_history: Mylist<URLData>
+ Parse(input: std::string): bool
- PushFront(data: URLData): void
- PushBack(data: URLData): void
- PopFront(): void
- PopBack(): void
- Size(): void
- Front(): void
- Back(): void
- Empty(): void
- Clear(): void
- Find(data: URLData): void
- Print(): void
private
struct
1
1
MyList
T : class
- m_head: MyNode* = nullptr
- m_tail: MyNode* = nullptr
- m_size: std::size_t = 0
<<bind>>
< URLData -> T>
+ ~MyList()
+ Front(): const T&
+ Back(): const T&
+ ForEach(func: void(*)(const T&)): void
+ PushFront(element: T): void
+ PushBack(element: T): void
+ PopFront(): void
+ PopBack(): void
+ Empty(): bool
+ Size(): std::size_t
+ Find(element: const T&): bool
+ FindValue(element: const T&): const T&
+ Clear(): void
URLData
- m_URL: std::string
- m_URLName: std::string
+ URLData(URLName: std::string, URL: std::string)
+ Print(data: const URLData&): static void
+ operator==(other: const URLData&): bool

Design Decisions

MyList

This class uses the default constructor (members are initialized), and a destructor that calls MyList::Clear() to destroy any allocated MyNode instances. No operator overloads were necessary.

const T& Front()   const T& Back()   const T& FindValue(const T& element)

These three functions return a const reference to stored data, they throw an std::out_of_range exception if the list does not contain the expected element. Const references are used to save resources when T is a large class, since it is expected that the instance will not be modified.

bool Empty()   bool Find(const T& element)

These two functions return a Boolean that indicates whether the list is empty or if the target element exists. These two functions should be used before calling Front(), Back(), or FindValue() to prevent the exception from being thrown. The parameter is const reference for the same reason as above.

void PushFront(T element)   void PushBack(T element)

These functions create a node with the parameter passed in. The parameters are not const references to allow std::move().

void PopFront() void PopBack() void Clear()
These functions erase existing nodes. They do not return anything to prevent exceptions.

std::size_t Size()
This function returns the number of elements in MyList

void ForEach(void(*)(const T&) func)
This function applies to appointed function to all elements. Allows functionalities like Print(). Const reference is used to save resources, no modifications are done to the elements.

MyNode
This struct has a constructor that initializes all members, and a default destructor. No operator overloads or member functions were necessary.

URLHistory
This class uses the default constructor and destructor because there is no need for initialization and no dynamic allocations are used. No operator overloads were necessary.

bool Parse(std::string input)
This function returns false is the input is "exit" or if the end of file is reached, otherwise it returns true. The parameter is passed by value to allow std::move().

void Front() void Size() void Back() void Empty()
void PopFront() void Clear() void PopBack() void Print()
These functions perform the specified actions and prints the desired output. No return values are necessary since they are private member functions. Print() uses the function pointer from URLData::Print() to access MyList::ForEach().

void PushFront(URLData data) void PushBack(URLData data) void Find(URLData data)
These functions do the same as the functions above but require a parameter. The parameters are passed by value to allow std::move().

URLData
This class has a constructor that initializes all values, and a default destructor. The == operator was overridden to allow case insensitive comparisons for the URLName member only.

static void Print(const URLData& data)
This function prints the URLName and URL members of the parameter 'data' to the console. The parameter is const reference to save resources as no modifications are necessary.

Test Cases
- There is not a requirement to handle invalid commands, so all invalid commands should be ignored.
- The comparisons for URL names used in ***find*** should be case insensitive.
- The URL and URL names should be parsed using "", all characters should be allowed within the "".
- The program should handle all invalid access commands, such as using ***front***, ***back***, ***find***, ***pop_front***, ***pop_back***, on an empty list/non-existing element.

Performance Considerations
Let n = the number of URLs in the deque.

clear find print
These three commands have a runtime of Θ(n). In my implementation, these commands go through a number of constant time functions before using a function that walks through the linked list once. Find uses two separate functions that each walk through the linked list once, but Θ(2n) = Θ(n).

push_front push_back pop_front pop_back size front back empty exit
These commands have a runtime of Θ(1). In my implementation, all of these commands only use functions that do not depend on the number of URLs in the deque.