Project 01 - Linked Lists

May 21, 2016

Introduction

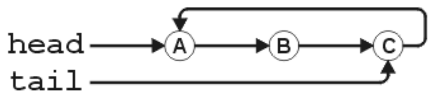
The purpose of this project is to make you used to handling linked lists. You are required to implement two different types of linked lists, namely, cyclic linked list, and doubly linked list. You are also required to create UML diagrams for each of the classes that you create. Finally, you have to provide the means to test your system by developing a menu program that allows the user to manipulate your linked lists.

Deliverables

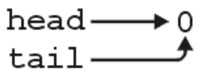
- An implementation of a cyclic linked list data structure.
- An implementation of a doubly linked list data structure.
- A menu program to test the implemented data structures.

1 Cyclic Linked List

In this part of the project, you need to implement two classes, and create their respective UML diagrams. Figure 1 shows examples of cyclic linked lists.



(a) Singly linked list with three nodes.



(b) An empty singly linked list.

Figure 1: Examples of singly linked lists.

1.1 Deliverables

- Singly linked node class with UML diagram.
- Cyclic linked list class with UML diagram.

1.2 SingleNode Class

1.2.1 Description

A class that stores an element of any type and an pointer to the next node in a list.

1.2.2 Data Members

1. A *Type* variable that contains the *data* information of the node.
2. A pointer to a SingleNode object, referred to as the *next* pointer.

1.2.3 Member Functions

Constructor

SingleNode(Type const &, SingleNode *)

This constructor takes two arguments: a constant reference to a Type (by default, an integer) and a pointer to a SingleNode (by default nullptr). These are assigned to the member variables, respectively.

Destructor

This class uses the default destructor.

Accessors

Type getData() const Returns the *data* element of the node.

SingleNode *getNext() const Returns the *next* pointer.

Mutators

This class has no mutators.

Friends

This CyclicLinkedList< *Type* > is a friend of this class.

1.3 CyclicLinkedList Class

1.3.1 Description

This class stores a finite list of n (zero or more) elements stored in singly linked nodes. If there are zero elements in the list, the list is said to be empty. Each element is stored in an instance of the SingleNode$<$ *Type* $>$ class. If the list is empty, the head pointer is assigned nullptr. Otherwise, head pointer points to the first node, the next pointer of the *ith* node ($1 \leq i < n$) points to the $(i+1)$st node, and the next pointer of the nth node points to the first node.

1.3.2 Data Members

1. A pointer to the head of the list, referred to as the *head* pointer.
2. A pointer to the tail of the list, referred to as the *tail* pointer.
3. An integer that holds the *size* of the linked list. The number of elements in the list.

1.3.3 Member Functions

Constructors

CyclicLinkedList() This constructor sets all member variables to 0 or nullptr, as appropriate.

Destructor

The destructor must delete each of the nodes in the linked list.

Accessors

int size() const; returns the number of items in the list.

bool empty() const; Returns true if the list is empty, false otherwise.

Type front() const; Retrieves the object stored in the node pointed to by the head pointer. This function throws a underflow if the list is empty.

Type back() const; Retrieves the object stored in the node pointed to by the tail pointer. This function throws a underflow if the list is empty.

SingleNode$<$ *Type* $>$ ***head() const;** Returns the head pointer.

SingleNode$<$ *Type* $>$ ***tail() const;** Returns the tail pointer.

int count(Type const &) const; Returns the number of nodes in the linked list storing a value equal to the argument.

Mutators

void push_front(Type const &); Creates a new SingleNode< *Type* > storing the argument, the next pointer of which is set to the current head pointer. The head pointer is set to this new node. If the list was originally empty, the tail pointer is set to point to the new node.

void push_back(Type const &); Similar to push_front, this places a new node at the back of the list.

Type pop_front(); Delete the node at the front of the linked list and, as necessary, update the head and tail pointers. Return the object stored in the node being popped. Throw an underflow exception if the list is empty.

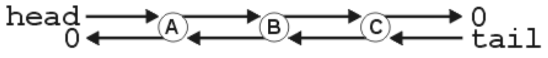
int erase(Type const &); Delete the node(s) (from the front) in the linked list that contains the element equal to the argument (use == to to test for equality with the retrieved element). As necessary, update the head and tail pointers and the next pointer of any other node within the list. Return the number of nodes that were deleted.

Friends

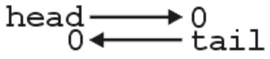
This class has no friends.

2 Doubly Linked List

In this part of the project, you need to implement two classes, and create their respective UML diagrams. Figure 2 shows examples of doubly linked lists.



(a) Doubly linked list with three nodes.



(b) An empty doubly linked list.

Figure 2: Examples of doubly linked lists.

2.1 Deliverables

- Doubly linked node class with UML diagram.
- Doubly linked list class with UML diagram.

2.2 DoubleNode Class

2.2.1 Description

A class which stores an object, a pointer to the next node in a linked list, and a pointer to the previous node in the linked list.

2.2.2 Data Members

1. A *Type* variable that contains the *data* information of the node.
2. A pointer to a DoubleNode object, referred to as the *next* pointer.
3. A pointer to a DoubleNode object, referred to as the *previous* pointer.

2.2.3 Member Functions

Constructor

DoubleNode(Type const &, DoubleNode *, DoubleNode *)

This constructor takes three arguments: a constant reference to an Type (by default, an integer) and two pointers to a DoubleNode (each by default nullptr). These are assigned to the member variables, respectively.

Destructor

This class uses the default destructor.

Accessors

Type getData() const Returns the element of the node.

DoubleNode *getNext() const Returns the next pointer.

DoubleNode *getPrevious() const Returns the previous pointer.

Mutators

This class has no mutators.

Friends

This DoublyLinkedList< *Type* > is a friend of this class.

2.3 DoublyLinkedList Class

2.3.1 Description

This class stores a finite list of n (zero or more) elements stored in doubly linked nodes. If there are zero elements in the list, the list is said to be empty. Each element is stored in an instance of the DoubleNode< *Type* > class. If the list is empty, the head and tail pointers are assigned nullptr. Otherwise, the head pointer points to the first node, the tail pointer points to the nth node, the next pointer of the *ith* node ($1 \leq i < n$) points to the $(i+1)$st node, the next pointer of the nth is assigned nullptr, the previous pointer of the *ith* node ($2 \leq i \leq n$) points to the $(i-1)$st node, and the previous pointer of the first node is assigned nullptr.

2.3.2 Data Members

1. A pointer to the head of the list, referred to as the *head* pointer.
2. A pointer to the tail of the list, referred to as the *tail* pointer.
3. An integer that holds the *size* of the linked list. The number of elements in the list.

2.3.3 Member Functions

Constructors

DoublyLinkedList() This constructor sets all member variables to 0 or nullptr, as appropriate.

Destructor

The destructor must delete each of the nodes in the linked list.

Accessors

int size() const; returns the number of items in the list.

bool empty() const; Returns true if the list is empty, false otherwise.

Type front() const; Retrieves the object stored in the node pointed to by the head pointer. This function throws a underflow if the list is empty.

Type back() const; Retrieves the object stored in the node pointed to by the tail pointer. This function throws a underflow if the list is empty.

DoubleNode< *Type* > ***head() const;** Returns the head pointer.

DoubleNode< *Type* > ***tail() const;** Returns the tail pointer.

int count(Type const &) const; Returns the number of nodes in the linked list storing a value equal to the argument.

Mutators

void push_front(Type const &); Creates a new DoubleNode< *Type* > storing the argument, the next pointer of which is set to the current head pointer. The head pointer is set to this new node. If the list was originally empty, the tail pointer is set to point to the new node.

void push_back(Type const &); Similar to push_front, this places a new node at the back of the list.

Type pop_front(); Delete the node at the front of the linked list and, as necessary, update the head and tail pointers. Return the object stored in the node being popped. Throw an underflow exception if the list is empty.

int erase(Type const &); Delete the node(s) (from the front) in the linked list that contains the element equal to the argument (use == to to test for equality with the retrieved element). As necessary, update the head and tail pointers and the next pointer of any other node within the list. Return the number of nodes that were deleted.

Friends

This class has no friends.

3 The Menu Program

In order to test your program, you are required to implement a menu program that provides the means to run each of the functions in your classes. Please choose the *double* simple data type to create the linked lists in your menu program. So for example, when asked to create a linked list, its nodes should hold double elements in their data fields. The TA will choose one member of your group to defend the demo, and the same grade will be assigned to all of the members of the group.

4 Acknowledgment

This project was created based on the work shared by the University of Waterloo.