**Beteab Gebru**

**Mar 2 2017**

**CSCI 301 - Section 02**

**Doubly linked unordered lists**

**Project 5: Program Documentation**

**Introduction**

For this project of two parts, the requirement was to build a doubly linked list ADT with features such as pointers to its first item and the last item in the chain and ability to store, read, remove last node, check if list is currently empty.

The second part of the project requires us to use programs in the main function which will be able to test the features mentioned in first part. Reading inputs into the nodes allows and removing them out of the list without program crash will be the solution.

The test specified requires input of characters to be stores in the list nodes when input happens to be the ‘#’ character it is a code for delete last character.

**Data structures**

The ADT Unordered-Doubly-linked list declared as a class in the program has the following

**Data Members**

Node\*first;//will be a pointer to first node in the chain

Node\*last;//pointer to last node in chain

**And Functions;**

DL\_List();- constructor for class DL\_List

~DL\_List();- list destructor will nulify the list members

void Data\_Intake(New\_Item); //inserts new item in a node and chains it to the end bool is\_empty(); returns true if first-> is NULL

void remove\_Last(); removes the last node of the chain

void give\_Back(Node\*); //works recursively to deallocate memory deallocate memory **friend functions**

friend std::ostream& operator << (std::ostream& out\_s, DL\_List& Chain);

**Functions**

**The Constructor**

-The constructor in the program simply initializes the Data-members First=NULL, Last =NULL.

**~The Destructor**

The destructor calls the function give\_Back() to efficiently get rid of the data and memory allocation by deleting the nodes one by one.

**Give Back function**

-The function simply recursively calls it self in the program simply initializes the Data-members First=NULL, Last =NULL.

**Data Intake function**

-This function gets a parameter passed by a client program and it attaches the new entry into the linked list.

::Steps

1- declare a node struct to house the new entry, the node will have the links next and back

2-If list is empty make both first and last point to it.

3-If list !empty -> make new node’s **back** pointer point to last-node (new\_node->back=last)

->make the pointer **last** point at the new node (new last node)

**Is Empty? Function**

-This function simply returns the truth value of the logical statement (first==NULL)

**Remove last**

-This function is called when we need to delete the last node of list

::Steps

1- declare a temporary node pointer

2- If list !empty ->assign the last pointer to the temp pointer ->check if its only node(last==first)

-> assign the node to temp\* and make first && last =NULL

->if >1 nodes -> temp = last; last=last->back; and delete the temporary pointer

**Friend function**

-This is the overloaded output stream function that will be able to take an unordered-doubly-linked list object and display its data elements in a chosen format. In my case I have decided to output the elements in between ().

**The Programs:**

The second part of the project requires the exercising of the said functions and abilities of the Unordered-Doubly-Linked ADT. Declared an instance of the class ADT ->MyList.

I used a text file as the data source to test program.

The Client program

1.The input stream gathers a character using reader.get(T)) from file

2.It stores every character by plugging the characters in the linked list MyList.Data\_Intake(T)

3.if the char turns out to be ‘#’ we delete the preceding node

I have implemented the mechanisms to handle cases such as

|  |  |  |
| --- | --- | --- |
| INPUT TEST CHARCTERS | EXPECTED OUTPUT | ACTUAL OUTPUT |
| A#B#C#D#E#F#G#H#I#J# | Empty(deletes all of them) |  |
| ABC##DEFG#H# | ADEF |  |
| ###DEFG#H | DEFH |  |
| ABCDEFGH########## | Empty(Deletes more than input) |  |
| Empty input | ---------------------------------- |  |
| ABCDEF | ABCDEF |  |
|  |  |  |

**Conclusion**

The purpose of this project was to create and exercise an ADT with doubly linked nodes and pointers to the head and tail nodes. It is very efficient to store unordered list of items or to maintain queues.