# Week Two Assignment

| **First Name** | Ryan |
| --- | --- |
| **Last Name** | Veitenheimer |
| **ID#** | 041372295 |
| **Email Address** | r.veitenheimer2295@student.nu.edu |

Table of Contents

[How to submit your Assignment 2](#_Toc15038261)

[Assignment 2](#_Toc15038262)

# How to submit your Assignment

After filling all the parts in this file, please follow the following steps.

1. Add your name and ID to the first page.
2. Save the file in the original format (Docx or Doc)

(please do not convert to other file formats e.g. PDF, ZIP, RAR, …).

1. Rename the file as

CSC335 *– HW2 - ID – YOUR Last Name - YOUR First Name.docx*

**Example:** CSC355 *– HW2 -* 234566435 - Smith - John.docx

1. Upload the file and submit it (only using Blackboard)

# Assignment

**CSC335 – Week 2 Assignment: Due on Saturday of week four, no later than the Midnight**

1. Consider the Linked list discussed in the class (must use the code discussed in class, provided in the textbook any changes made to it must be documented and explained).

template<typename T>

class Node

{

public:

Node();

Node(T value, Node<T>\* nextNode);

T data;

Node<T>\* next;

};

template<typename T>

class LinkedList

{

public:

LinkedList();

Node<T>\* getHead() const; // return the head pointer.

Node<T>\* getTail() const; // returns the tail pointer.

void ListAppend(T value); // inserts an element at the end of the list.

void listPrepend(T value); //inserts a n element at the head of the list.

void insertAfter(Node<T>\* curNode, T value); // Insert value after the curNode

void removeAfter(Node<T>\* curNode); // remove Node after CurNode

void removeHead(); // Removes the first element.

void removeTail(); // removes the last element.

void printList() const; // print the elements of the linked list.

private:

Node<T>\* head;

Node<T>\* tail;

};

Add the following functions to the class LinkedList (No need to run your code, unless you want to. The code must be syntactically correct.)

1. **getLength():** returns the number of items in the LinkedList.
2. **search(T element)** : searches for the **element** in the LinkedList, returns a pointer to the Node containing the element if found, otherwise returns NULL.
3. Define the **Node** class for a Doubly linked list. Define a **doubly linked list class** that uses this Node. Implement the node as well as the Doubly Linked list classes. In the doubly linked list class include a **default constructor** and functions **append**, **prepend** and **print** that displays all the elements of the doubly linked list. Implement your classes and write a main function in which you populate a doubly linked list by appending and prepending then print the list.
4. Write a program to compute the number of collisions when inserting a sequence of 20 random integers between 0 and 100. Insertion should be done once using the linear probing and then separately with quadratic probing for collision resolution. Use a table size of 31. Output must display the list of integers inserted and the hash tables after insertions. The program must display the number of collisions in each collision resolution strategy as well.
5. Insert the sequence of numbers 4371, 1323, 6173, 4199, 4344, 9679, 1989 in a hash table of size 10 using the hash function Show the hash tables after insertions using the following collision resolution strategies. Clearly show calculations and indicate cases where a number cannot be inserted in the table.
   1. Separate chaining hash table
   2. Hash table using linear probing
   3. Hash table using quadratic probing
   4. Hash table with double hashing. Use the second hash function