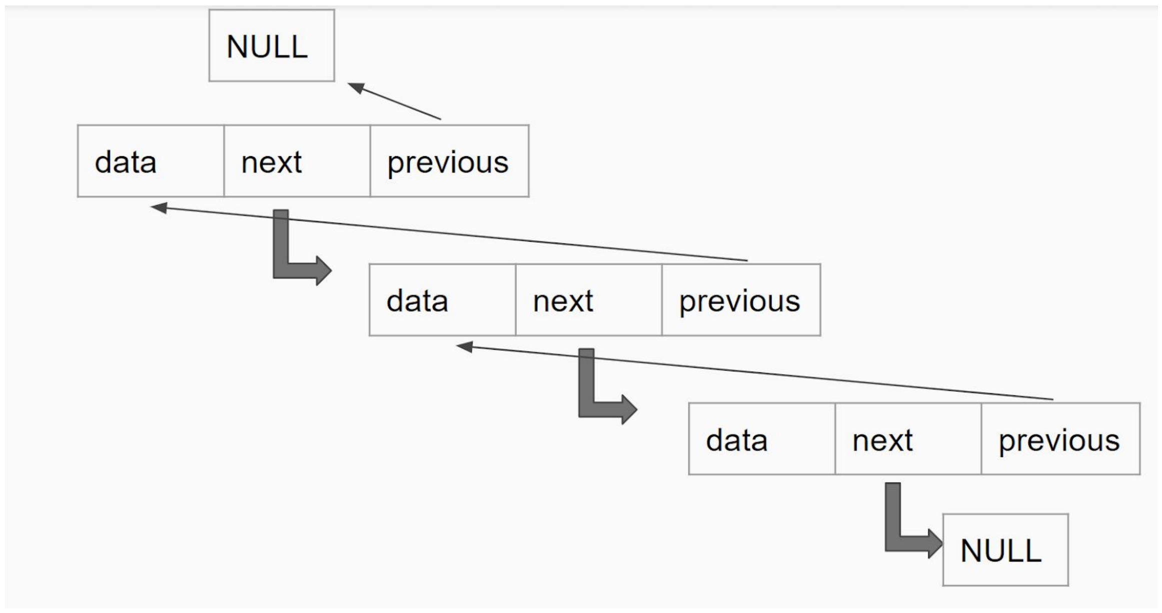
McGill University
School of Computer Science

COMP-206

Mini Assignment #5

Due: March 13, 2019 on myCourses at 23:30

Arrays have their limitations. Their size should be known in advance during compile time. Inserting a new element in an array is a costly operation because it involves the shifting of elements. Resizing of an array is also very costly operation because it involves the creation of a new array and the copying of all the elements to the new structure. These limitations can be avoided by using a Linked List data structure.
In this assignment you will be designing and implementing a Doubly Linked List structure.
A linked list data structure is a collection of elements called nodes. Each element can hold data along with a pointer to the next element. Doubly linked list structures hold two pointers in each element instead of one. The first pointer points to the next element, and the second one points to the previous element in the list (see following figure).



Let's start by designing and implementing a doubly linked list data structure step by step. Note that we will need to define one node structure to hold the data and pointers and we also need to define multiple utility functions that implement the doubly linked list logic and behavior.

Create a C program called **dllstructure.c** that implements the following:

1. Create a structure called Node having the following data members:
    a. Int data member to hold the data in each element
    b. A pointer to Node to hold the pointer to the next element
    c. A pointer to Node to hold the pointer to the previous element

2. Write a function that will loop over all the elements in the list and print their values to the screen. The signature of the function will be
   void print_dll(Node* head);

3. Write a function that will insert a new element in the linked list after the first occurence of the value provided by the second argument and having the value provided in the third argument. The signature of the function will be:
   void insert_after(Node* head, int valueToInsertAfter, int valueToBeInserted);

   if **valueToInsertAfter** was not found in the list, insert **valueToBeInserted** at the end of the list. The first argument is a pointer to the linked list.
   Remember that you have to update the previous and next pointers where appropriate to reflect the new list after inserting the new value.

4. Write a function that will delete the first occurence of the value provided in the second argument. The signature of the function will be:
   void delete_element(Node* head, int valueToBeDeleted);

   The first argument is a pointer to the linked list. Remember that you have to update the previous and next pointers where appropriate to reflect the new list after deleting the node. If the value to be deleted was not found, do nothing.

5. Write a function that will sort the elements in ascending order. The signature of the function will be:
   void sort_dll(Node* head);
   Use bubble sort algorithm. This sorting algorithm has the reputation of being the least efficient one, however it is the simplest to implement. Let's ignore efficiency for the sake of simplicity. The algorithm is explained in this wikipedia article: https://en.wikipedia.org/wiki/Bubble_sort

6. What other function you should be adding to your code to make sure that we don't have any memory leaks? Please provide a full implementation.

The main() function that will be used to test your functions is provided as follow, please use it as the main function of your code:

```
int main()
{
  int array[5] = {11, 2, 7,  22, 4};
  Node* head;

  /* Question 1 */
  head = create_dll_from_array(array, 5); //size of array is 5

  /* Question 2 */
  print_dll(head);

  /* Question 3 */
  // To insert 13 after the first occurence of 7
  insert_after(head, 7, 13);
  // To insert 29 after the first occurence of 21
  insert_after(head, 21, 29);
  print_dll(head);

  /* Question 4 */
  delete_element(head, 22);
  print_dll(head);
  delete_element(head, 11);
  print_dll(head);

  /* Question 5 */
  sort_dll(head);
  print_dll(head);

  /* Question 6 */
  // add the call to your function here

  return 0;
}
```

WHAT TO HAND IN

Everything must be submitted to My Courses before the due date. Remember that you can hand in your assignment up to two days late but there will be a penalty of 5% each day. After that, your assignment will not be accepted. Please hand in the following:

- A single zip file having:
  - **your c code file dllstructure.c**
  - **a shell script tester.sh that compiles your c code and runs the executable**

HOW IT WILL BE GRADED

The assignment is worth a total of 20 points.

- 20 points dllstructure program implementation as follow:
  - 2 points for creating the proper structure
  - 3 points for print_dll function
  - 4 points for insert_after function
    - 1 point for finding the first occurence
    - 1 point for successfully inserting the elements and updating the pointers
    - 1 point for detecting if no occurrence had been found
    - 1 point for inserting at the end in case no occurrence had been found
  - 4 points for delete_element function
    - 1 point for finding the element to be deleted
    - 1 point for detecting if no occurrence had been found
    - 1 point for properly updating the pointers
    - 1 point for dealing with cases such as deleting the head element
  - 3 points for sorting the elements using bubble sort algorithm
  - 4 points for question 6
    - 2 points for knowing what should be done
    - 2 points for providing a working implementation

Please note that the shell script will not be graded, however a penalty of -3 points will be applied to the overall grade if the shell script does not run properly.

GRADING RULES

The following rules are followed by the TA when grading assignments:

- A program must run in order to get a grade (even if it does not run well). If it does not run (does not compile) it will receive a zero. (Make sure to run your programs from Trottier – they sometimes do not run the same from home when logging in using putty.)
- The TA will grade using the mimi.cs.mcgill.ca server.
- All questions are graded proportionally (assuming the program runs at all). This means that if 40% of the question is correct, you will receive 40% of the grade.
- The TA will run the script and observe the execution of your program.
- The TA will check whether the linked list structure is created and the content is printed out to the screen properly. Numbers should be separated by a tabulation.
- The TA will check whether the insert_after function is behaving properly under all possible use cases.
- The TA will check whether the delete_element function is behaving properly under all possible use cases.
- The TA will check whether the sort_dll function is behaving properly and the numbers are printed out to the screen in ascending order.
- The TA will inspect your code for memory leaks and will check whether you succeeded on providing an implementation for that purpose in question 6.