CS170L – High-level Programming II
Lab 2: Linked List (continued) & Doxygen Documentation

Deadline:	Before the next lab starts
Topics:	1. Additional functionality implemented for linked lists. 2. Source code documentation using **Doxygen**.
Objectives:	To demonstrate an ability to: 1. Develop algorithm for solving linked-list problems. 2. Use Doxygen style comments and documentation generation.
Header files allowed:	**`iostream, iomanip, string, list.h`** No other header files can be included in your source code.
Deliverables:	Put the following 4 files in the submission folder **plus the signed checklist**: 1. **`list.cpp`** 2. **`makefile.gnu`**, **`makefile.ms`** – the Makefiles for GNU and Microsoft compilers. 3. **Doxygen** CHM file. All files should be in put in a folder named: `cs170_<your Digipen login id>_2,` Then the folder should be put in a zipped file named: `cs170_<your Digipen login id>_2.zip.` Finally, upload the zipped file using the submission link on Moodle. Please see CS 170 page for coding and submission standards.

1. Programming Assignment

1.1. Your Programming Task

This lab will provide additional functionalities to the linked list data type. Your task is to define the following functions:

- **`void AddToEnd(Node **pList, int value);`**
- **`void AddToFront(Node **pList, int value);`**
- **`void FreeList(Node *list);`**
- **`Node *Find(Node *list, int value);`**
- **`void Insert(Node **pList, int value, int position);`**
- **`void Delete(Node **pList, int value);`**
- **`void Concat(Node **pList1, Node **pList2);`**

The first three should have been implemented in the previous assignment. They should still work this week. Fix any bugs you might otherwise had. The last four are the new functions you need to implement. The specifications for them are described in the **`list.h`** file and examples of their usage in the **`main.cpp`** driver file.)

In this lab, you'll need to create two lists **`list1`** and **`list2`**. Your program needs to read in a sequence of numbers from the user input, 4 numbers at a time. Your program ends when the numbered entered are **`0 0 0 0`**. This is also the only time the 1st number of them is **`0`**. Otherwise, the 1st number indicates the user's choice: **`Add`**, **`Find`**, **`Insert`**, **`Delete`** items or **`Concatenate`** two lists. The 2nd number indicates the list. The 3rd number indicates the value of the item. When applicable, the 4th number is the position in the list where the user wants to insert the new value.

More details

When the 1st number is between 1 and 5, then the list to be used is determined by the 2nd number: 1 for **`list1`**, 2 for **`list2`**. If the 1st number is:

- **1**: add the 3rd number to the front of the considered list (**`AddToFront`**). Ignore the 4th number.
- **2**: add the 3rd number to the end of the considered list (**`AddToEnd`**). Ignore the 4th number.
- **3**: find the 3rd number in the considered list (**`Find`**). Ignore the 4th number.
- **4**: delete the 3rd number from the considered list (**`Delete`**). Ignore the 4th number.
- **5**: insert the 3rd number at position indicated by the 4th number in the considered list (**`Insert`**). **6**: concatenate **`list2`** to **`list1`**. You can ignore other numbers in this case.

Similar to the first lab, you are given a sample test case in **`input.txt`** with the corresponding output in **`output.txt`**. Note that this test case is not exhaustive. You should think of other possible cases to test your program' correctness.

2. Grading Scheme

The grader will evaluate your submitted files, including **`list.cpp`**, **`makefile.gnu`**, **`makefile.ms`** and the **Doxygen** CHM file.

The entire lab is scored over 100 points. The distribution of points is as follows:

- 20 points for successful build and execution using each of the 2 Makefiles:
  - 5 points for build:
    - 0 point if the executable cannot be created.
    - 3 points if the build succeeds but results in some warnings.
    - 5 points if the executable can be created without any warnings.
  - 15 points for execution:
    - 5 points for an exact match with the sample output given for each compiler.
    - 10 points for 5 hidden test cases not given in the driver.
- 10 points for correct implementation of each of the 4 linked list functions in **`list.cpp`**. Note that there should be no memory leak. As a rule of thumb in our lab, the call to the **`new`** and **`delete`** operators should match and the program should not crash. For each function, memory leak will results in at least 5 points deducted.
- 10 points for programming style: Clearly named variables, defining variables before their first use, appropriate comments and consistent indentation (no mixing between spaces and tabs).
- 10 points for proper **Doxygen** CHM file. Do not forget the file header and function header comments. Every file should have a proper file header comment that specifies the purpose of the file (**`list.cpp`** in this case). For each function, there should also be a proper function header comment that specifies the purpose/return value of the function, and the role of each parameter/argument of the function.

It is also very important to note that:

- Any plagiarism detected results in 0 marks for the entire lab.
- If you don’t submit the signed programming checklist, we’ll grade your assignment but record a score of 0.
- Failure to comply with the submission conventions (Page 1) results in 10 points deducted.
- **The file header comment has to be the first thing in a submitted source file**. A sample file header comment you ought to use is given in **`CustomDate.cpp`** in the **`Doxygen_Guide`**. Failure to conform to these stipulations will result in all **10 points deducted** for programming style.
- Submissions after the deadline will receive 0 mark even if it is accepted by Moodle.