UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science

Faculty of Engineering, Built Environment & IT

University of Pretoria

COS110 - Program Design: Introduction

Practical 8 Specifications

Release Date: 20-10-2025 at 06:00

Due Date: 24-10-2025 at 23:59

Late Deadline: 25-10-2025 at 00:59

Total Marks: 48

Read the entire specification before starting with the practical.

Contents

1 General Instructions

- *Read the entire assignment thoroughly before you begin coding.*
- This assignment should be completed individually.
- **Every submission will be inspected with the help of dedicated plagiarism detection software.**
- Be ready to upload your assignment well before the deadline. There is a late deadline which is 1 hour after the initial deadline which has a penalty of 20% of your achieved mark. **No extensions will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or structure).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `https://portal.cs.up.ac.za/files/departmental-guide/`.
- Unless otherwise stated, the usage of C++11 or additional libraries outside of those indicated in the assignment, will **not** be allowed. Some of the appropriate files that you have submit will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use C++98**
- All functions should be implemented in the corresponding `cpp` file. No inline implementation in the header file apart from the provided functions.
- The usage of ChatGPT and other AI-Related software to generate submitted code is strictly forbidden and will be considered as plagiarism.

2 Overview

In this practical, you are tasked with implementing a doubly-linked list with **no head or tail pointer being stored**.

3 Your Task:

You are required to implement the following class diagram illustrated in Figure 1. Pay close attention to the function signatures as the `h` files will be overwritten, thus failure to comply with the UML, will result in a mark of 0.

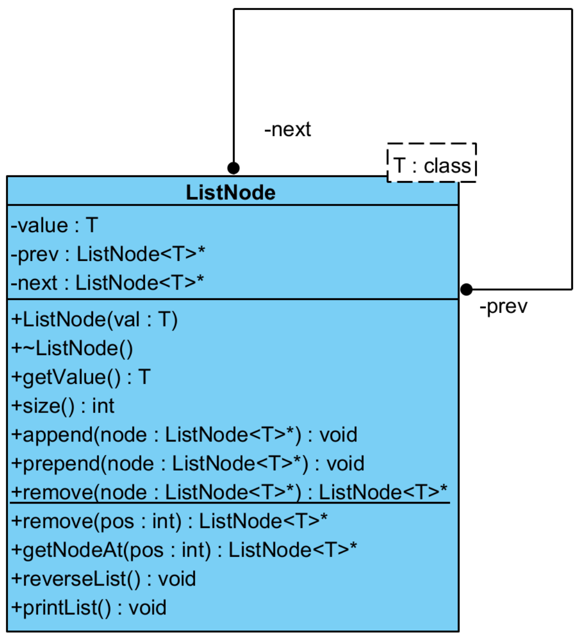
-next
T : class
-prev
ListNode
-value : T
-prev : ListNode<T>*
-next : ListNode<T>*
+ListNode(val : T)
+~ListNode()
+getValue() : T
+size() : int
+append(node : ListNode<T>*) : void
+prepend(node : ListNode<T>*) : void
+remove(node : ListNode<T>*) : ListNode<T>*
+remove(pos : int) : ListNode<T>*
+getNodeAt(pos : int) : ListNode<T>*
+reverseList() : void
+printList() : void

Figure 1: Class diagram

3.1 ListNode

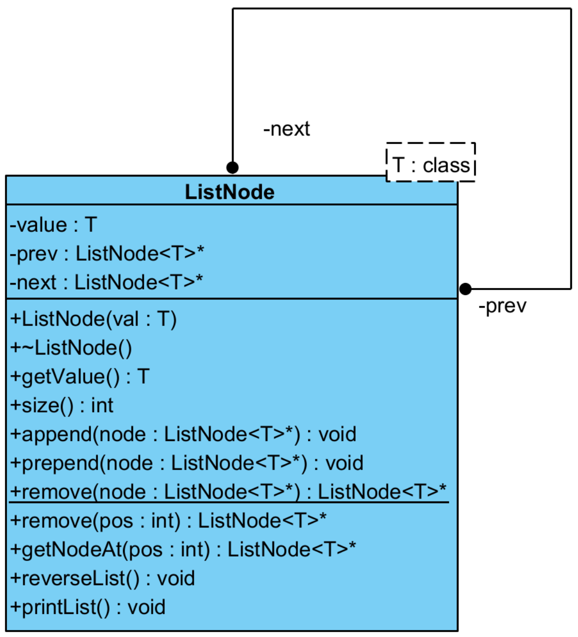
-next
T : class
ListNode
-value : T
-prev : ListNode<T>*
-next : ListNode<T>*
+ListNode(val : T)
+~ListNode()
+getValue() : T
+size() : int
+append(node : ListNode<T>*) : void
+prepend(node : ListNode<T>*) : void
+remove(node : ListNode<T>*) : ListNode<T>*
+remove(pos : int) : ListNode<T>*
+getNodeAt(pos : int) : ListNode<T>*
+reverseList() : void
+printList() : void
-prev

Figure 2: UML Diagram for ListNode Class

3.1.1 Members

- ListNode Member 1
  - This is the value of a node. It could be of any data type.
- ListNode Member 2
  - The pointer to the previous node
- ListNode Member 3
  - The pointer to the next node

3.1.2 Functions

- ListNode Function 1
  - This function should set the node's value appropriately.
  - The next and previous pointer must be set to *NULL*
- ListNode Function 2
  - This function should deallocate the memory used by a node and all other nodes that it is linked to.
  - For example: If we have four nodes that are linked together, deleting a single node should deallocate the memory used by all four nodes.

- ListNode Function 3
  - This is a constant function.
  - This function should return a node's value
- ListNode Function 4
  - This is a constant function.
  - This function should calculate the total number of nodes in the doubly-linked list that the node is a part of.
- ListNode Function 5
  - This function should add the new node to the very end of the doubly-linked list that *this* node is a part of.
  - If the passed-in parameter is equal to *this*, the function should do nothing.
  - If the passed-in parameter is already contained within the linked list, the function should also do nothing.
- ListNode Function 6
  - This function should add the new node to the very beginning of the doubly-linked list that *this* node is a part of.
  - If the passed-in parameter is equal to *this*, the function should do nothing.
  - If the passed-in parameter is already contained within the linked list, the function should also do nothing.
- ListNode Function 7
  - This is a static function
  - The function should remove a node from its linked list.
  - This is done by setting the node's *previous* and *next* neighbours' pointers correctly.
  - Remember to set the removed node's *prevprev* and *next* pointers to *NULL*.
  - The removed node should then be returned by the function.
  - If no node was removed, the function should return *NULL*.
- ListNode Function 8
  - This function should remove a node from the doubly-linked list at a relative position to *this* node.
  - If the passed-in parameter is zero, the function should do nothing.
  - If the passed-in parameter is negative, the function should remove the node that is *pos* steps to the left of *this* node.

  - If the passed-in parameter is positive, the function should remove the node that is *pos* steps to the right of *this* node.
  - Additionally, if the position is out of bounds, the function should do nothing.
  - The removed node should then be returned by the function.
  - If no node was removed, the function should return *NULL*.

- ListNode Function 9
  - This function should return the node at the passed-in position relative to *this* node.
  - If the passed-in parameter is zero, the function should return *this*.
  - If the passed-in parameter is negative, the function should return the node that is *pos* steps to the left of *this* node.
  - If the passed-in parameter is positive, the function should return the node that is *pos* steps to the right of *this* node.
  - Additionally, if the position is out of bounds, the function should do nothing.
  - If the position is out of bounds, the function should return *NULL*.

- ListNode Function 10
  - This function should reverse all the pointers in the doubly-linked list.
  - Example: If the list contained nodes with values 'a','b' and 'c' respectively, after the function call; the list should contain the nodes in the order 'c','b','a'.

- ListNode Function 11
  - This is a constant function.
  - This function should output a string containing all the node values in the doubly-linked list, printed from the start to the end.
  - Each value should be separated by a single whitespace character.
  - The output string must not end with a trailing whitespace.

4 Memory Management

As memory management is a core part of COS110 and C++, each task on FitchFork will allocate approximately 10% of the marks to memory management. The following command is used:

```
valgrind --leak-check=full ./main
```

Please ensure, at all times, that your code *correctly* de-allocates *all* the memory that was allocated.

5 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, 10% of the assignment marks will be allocated to your testing skills. To do this, you will need to submit a testing main (inside the `main.cpp` file) that will be used to test an Instructor Provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing the gcov [1] tool, specifically the following version *gcov (Debian 8.3.0-6) 8.3.0*, will be used. The following set of commands will be used to run gcov:

```
g++ --coverage *.cpp -o main
./main
gcov -f -m -r -j ${files}
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

We will scale this ration based on class size.

The mark you will receive for the testing coverage task is determined using Table 1:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 1: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the functions stipulated in this specification will be considered to determine your testing mark. Remember that your main will be testing the Instructor Provided code and as such, it can only be assumed that the functions outlined in this specification are defined and implemented.

As you will be receiving marks for your testing main, we will also be doing plagiarism checks on your testing main.

[1] For more information on gcov please see `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`

6 Implementation Details

- You must implement the functions in the header files exactly as stipulated in this specification. Failure to do so will result in compilation errors on FitchFork.
- You may only use **c++98**.
- You may only utilize the specified libraries. Failure to do so will result in compilation errors on FitchFork.
- You may only use the following libraries:
  - iostream
  - sstream
  - string
  - cmath
- You are supplied with a **trivial** main demonstrating the basic functionality of this assessment.

7 Upload Checklist

The following c++ files should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- ListNode.cpp
- `main.cpp`
- Any textfiles used by your `main.cpp`
- `testingFramework.h` and `testingFramework.cpp` if you used these files.

The files should be in the root directory of your zip file. In other words, when you open your zip file you should immediately see your files. They should not be inside another folder.

8 Submission

You need to submit your source files on the FitchFork website (`https://ff.cs.up.ac.za/`). All methods need to be implemented (or at least stubbed) before submission. Your code should be able to be compiled with the following command:

```
g++ -Werror -Wall *.cpp -o main
```

and run with the following command:

```
./main
```

Remember your `h` file will be overwritten, so ensure you do not alter the provided `h` files.
You have 10 submissions and your best mark will be your final mark. Upload your archive to the Practical 8 slot on the FitchFork website. If you submit after the deadline but before the late deadline, a 20% mark deduction will be applied. **No submissions after the late deadline will be accepted!**