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 9 Specifications

Release Date: 27-10-2025 at 06:00

Due Date: 31-10-2025 at 23:59

Late Deadline: 01-11-2025 at 00:59

Total Marks: 103

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

Linked Lists are data structures which can be used to store data in a specific order. This is done using nodes which contain the data that is being stored and also a pointer to the next node in the list. A variation of a Linked List is a Circular Linked List. In a normal Linked List, the last node in the list points to NULL to indicate the end of the list. In a Circular Linked List, the last node points to the first node, thus all of the nodes are connected in a big circle.

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.

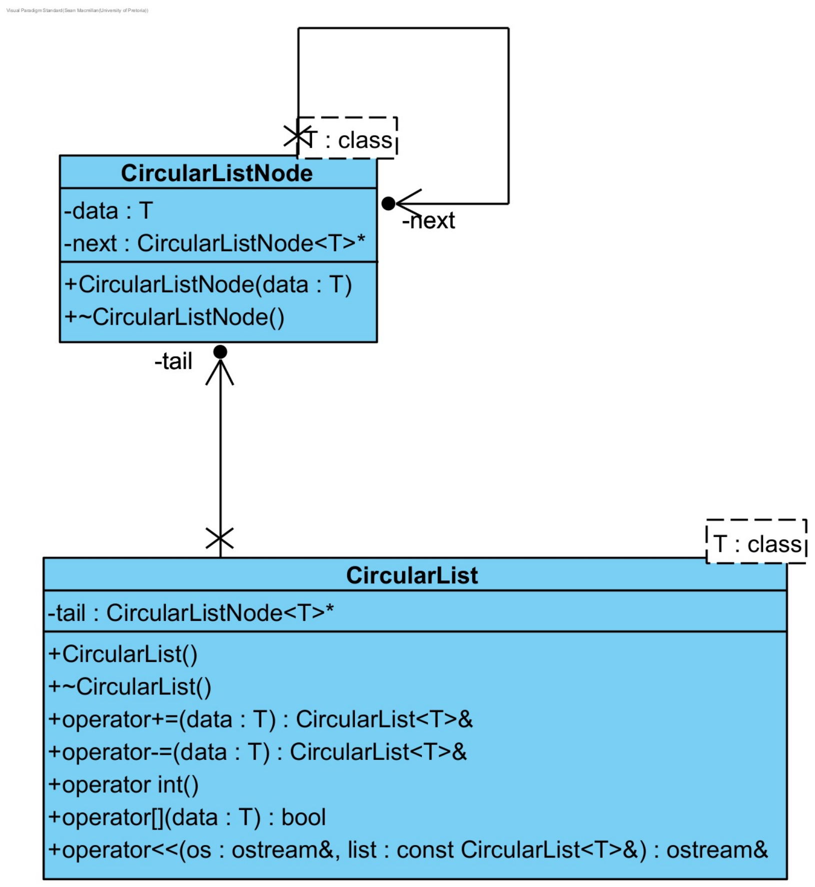
T : class
CircularListNode
-data : T
-next : CircularListNode<T>*
+CircularListNode(data : T)
+~CircularListNode()
-next
-tail
T : class
CircularList
-tail : CircularListNode<T>*
+CircularList()
+~CircularList()
+operator+=(data : T) : CircularList<T>&
+operator-=(data : T) : CircularList<T>&
+operator int()
+operator[](data : T) : bool
+operator<<(os : ostream&, list : const CircularList<T>&) : ostream&

Figure 1: UML

Note, the importance of the arrows between the classes are not important for COS 110 and will be expanded on in COS 214.

4 Template Compiling

There are many different approaches to compile and use templates in c++. For this practical you have to follow the following approach, as the header files will be overwritten on Fitchfork and will only work with this approach. The approach used is the approach on Slide 7 of Week 9 Lecture 3. This approach is:

- At the bottom of the header file (but before the #endif) include the implementation file.
- When you want to use a class in a different file, only include the header file.
- When compiling your code, don't compile the template classes.
- *Optional: In the implementation file you are allowed to include the header file at the top to allow auto complete to work for most IDE's.*

4.1 CircularListNode

This is a template class that will be used as the nodes in a Circular Linked List.

- **Do not** make any changes to this class. It is given to you, and will be overwritten on Fitchfork.
- It has a basic constructor, and an empty destructor.
- All of the members are private, but the CircularList class is declared as a friend to this class, so the CircularList class can access the private members of this class.

4.2 CircularList

This is a template class that stores a Circular Linked List.

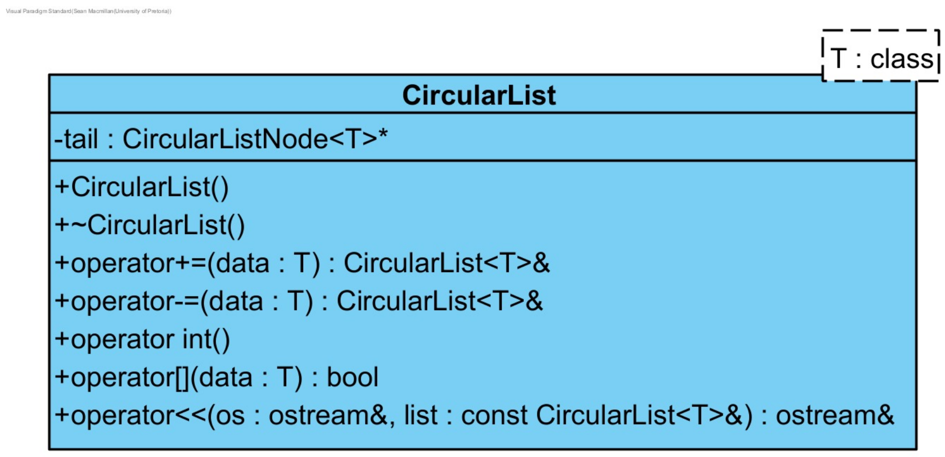
Visual Paradigm Standard(Sean Macmillan(University of Pretoria))
T : class
CircularList
-tail : CircularListNode<T>*
+CircularList()
+~CircularList()
+operator+=(data : T) : CircularList<T>&
+operator-=(data : T) : CircularList<T>&
+operator int()
+operator[](data : T) : bool
+operator<<(os : ostream&, list : const CircularList<T>&) : ostream&

Figure 2: CircularList

4.2.1 Members

- Member 1
  - This will store the tail of the CircularList.
  - If the list is empty this will be NULL.

4.2.2 Functions

- Function 1
  - This is the default constructor. Initiliase the list to an empty list.
- Function 2
  - This is the destructor. This should deallocate the whole list.
- Function 3
  - This should create a new node using the passed-in data.
  - This operator should allow chaining.
  - The node should then be placed inside the list.
  - The list must remain in ascending order at all times. Thus insert the node at the correct position to ensure this is true.
  - Duplicate values are allowed.
- Function 4
  - This should remove a node with the same data as the passed-in value.
  - This operator should allow chaining.
  - If there are multiple nodes with this data, then only delete the first one.
  - The node should be unlinked from the list, and the memory for this node should be freed.
- Function 5
  - This should return the number of nodes in the list.
- Function 6
  - This will act as a search function. Return true if the passed-in data is inside the list, and return false if the passed-in data is not in the list.

- Function 7
  - This function is a friend to both CircularListNode and CircularList to get access to the private members of both classes.
  - This operator will be used to print out a string representation of the list.
  - This operator should allow chaining.
  - If the list is empty then print "Empty List".
  - If the list is not empty then print out all of the data in the list, starting with the head, and ending with the tail.
  - There should be an arrow (->) between the data of each node.
  - There should not be any spaces in the output.
  - There should be an endline at the end of the output.

5 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.

6 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.

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

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.

7 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
- You are supplied with a **trivial** main demonstrating the basic functionality of this assessment.

8 Upload Checklist

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

- CircularList.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.

9 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 9 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!**