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

Assignment 3 Specifications

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

FF Submission Opening Date: 26-10-2025

Due Date: 07-11-2025 at 23:59

Late Deadline: 08-11-2025 at 00:59

Total Marks: 410

Read the entire specification before starting with the practical.

Changes to the specification are indicated in orange.

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 assignment, you will implement a system in which a Printer maintains a collection of printable objects. This printer will be a singleton class, meaning only one instance of it may exist at any given time. This assignment will cover the following topics:

- Linked Lists
- Stacks
- Queues
- Templates

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.

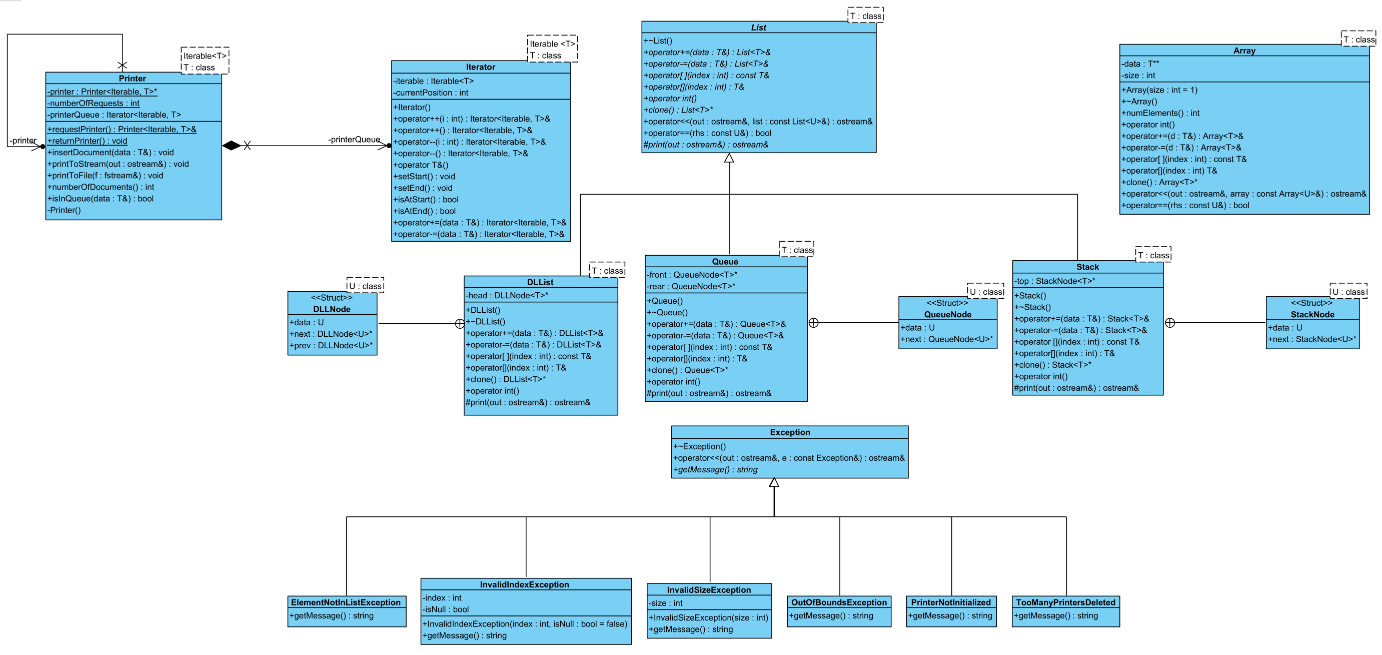
Printer
Iterable<T>
T : class
-printer : Printer<Iterable, T>*
-numberOfRequests : int
-printerQueue : Iterator<Iterable, T>
+requestPrinter() : Printer<Iterable, T>&
+returnPrinter() : void
+insertDocument(data : T&) : void
+printToStream(out : ostream&) : void
+printToFile(f : fstream&) : void
+numberOfDocuments() : int
+isInQueue(data : T&) : bool
-Printer()
-printer
-printerQueue
Iterator
Iterable <T>
T : class
-iterable : Iterable<T>
-currentPosition : int
+Iterator()
+operator++(i : int) : Iterator<Iterable, T>&
+operator++() : Iterator<Iterable, T>&
+operator--(i : int) : Iterator<Iterable, T>&
+operator--() : Iterator<Iterable, T>&
+operator T&()
+setStart() : void
+setEnd() : void
+isAtStart() : bool
+isAtEnd() : bool
+operator+=(data : T&) : Iterator<Iterable, T>&
+operator-=(data : T&) : Iterator<Iterable, T>&
List
T : class
+~List()
+operator+=(data : T&) : List<T>&
+operator-=(data : T&) : List<T>&
+operator[ ](index : int) : const T&
+operator[](index : int) : T&
+operator int()
+clone() : List<T>*
+operator<<(out : ostream&, list : const List<U>&) : ostream&
+operator==(rhs : const U&) : bool
#print(out : ostream&) : ostream&
Array
T : class
-data : T**
-size : int
+Array(size : int = 1)
+~Array()
+numElements() : int
+operator int()
+operator+=(d : T&) : Array<T>&
+operator-=(d : T&) : Array<T>&
+operator[ ](index : int) : const T&
+operator[](index : int) T&
+clone() : Array<T>*
+operator<<(out : ostream&, array : const Array<U>&) : ostream&
+operator==(rhs : const U&) : bool
<<Struct>>
DLLNode
U : class
+data : U
+next : DLLNode<U>*
+prev : DLLNode<U>*
DLList
T : class
-head : DLLNode<T>*
+DLList()
+~DLList()
+operator+=(data : T&) : DLList<T>&
+operator-=(data : T&) : DLList<T>&
+operator[ ](index : int) : const T&
+operator[](index : int) : T&
+clone() : DLList<T>*
+operator int()
#print(out : ostream&) : ostream&
Queue
T : class
-front : QueueNode<T>*
-rear : QueueNode<T>*
+Queue()
+~Queue()
+operator+=(data : T&) : Queue<T>&
+operator-=(data : T&) : Queue<T>&
+operator[ ](index : int) : const T&
+operator[](index : int) : T&
+clone() : Queue<T>*
+operator int()
#print(out : ostream&) : ostream&
<<Struct>>
QueueNode
U : class
+data : U
+next : QueueNode<U>*
Stack
T : class
-top : StackNode<T>*
+Stack()
+~Stack()
+operator+=(data : T&) : Stack<T>&
+operator-=(data : T&) : Stack<T>&
+operator [](index : int) : const T&
+operator[](index : int) : T&
+clone() : Stack<T>*
+operator int()
#print(out : ostream&) : ostream&
<<Struct>>
StackNode
U : class
+data : U
+next : StackNode<U>*
Exception
+~Exception()
+operator<<(out : ostream&, e : const Exception&) : ostream&
+getMessage() : string
ElementNotInListException
+getMessage() : string
InvalidIndexException
-index : int
-isNull : bool
+InvalidIndexException(index : int, isNull : bool = false)
+getMessage() : string
InvalidSizeException
-size : int
+InvalidSizeException(size : int)
+getMessage() : string
OutOfBoundsException
+getMessage() : string
PrinterNotInitialized
+getMessage() : string
TooManyPrintersDeleted
+getMessage() : string

Figure 1: Class diagrams

Note, the significance of the arrows between the classes is not important for COS 110 and will be expanded on in COS 214. However, you should take note of the **structs contained within classes**. Containment is represented using a small circle with a cross at the end connected to the containing class, and the other end connected to the nested struct. The nested structs in this Assignment are `private` to their associated classes.

All of the inheritance used in the assignment is `public`. Use the provided UML diagrams to determine when a function is `static` or `pure-virtual`.

3.1 Exception Hierarchy

3.2 Exception

This is the interface class for all exceptions that will be thrown in your program.

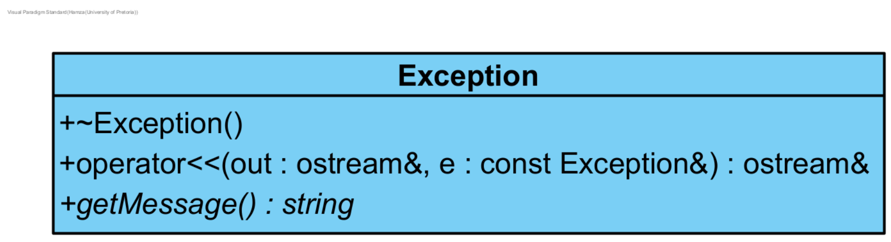
Visual Paradigm Standard(Hamza(University of Pretoria))
Exception
+~Exception()
+operator<<(out : ostream&, e : const Exception&) : ostream&
+getMessage() : string

Figure 2: UML for Exception

3.2.1 Functions

- Function 1
  - This is a virtual function.
  - This function deallocates any dynamically allocated memory if any was allocated.
- Function 2
  - This is a friend function.
  - This function should insert the result of the passed-in `Exception` object's `getMessage` function into the passed-in `ostream` parameter and then return the stream.
  - After inserting the message, insert an endline into the stream.
- Function 3
  - This is a constant function.
  - It will return an error message that explains the reason behind an exception being thrown.

3.3 InvalidIndexException

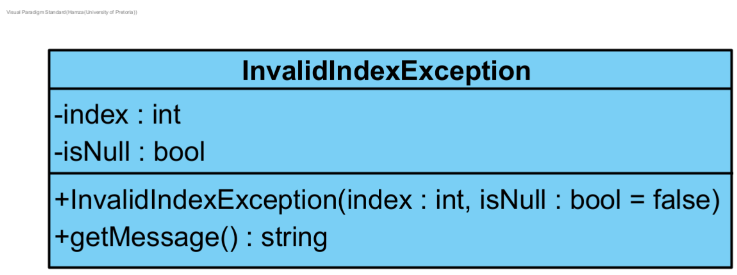

Figure 3: UML for InvalidIndexException

3.3.1 Members

- Member 1
  - This is the invalid index that caused the exception to be thrown after an attempted access.
- Member 2
  - This indicates whether the data at the invalid index is `NULL` or not.
  - The default value for this member variable (specified in your header file) is `false`.
  - This member variable will only be applicable to the `Array` class.

3.3.2 Functions

- Function 1
  - This function should initialize the exception's `index` and `isNull` member variables to the corresponding parameters.
  - `isNull` should have a default value of `false` in your class declaration.
- Function 2
  - This is a virtual function.
  - This is a constant function.
  - This function should return an error message.
  - if `isNull` is true, this function should return a string in the following format: `"Attempted access at invalid index of <INDEX> and index was null."`
  - Otherwise, the string should be of the following format: `"Attempted access at invalid index of <INDEX>."`
  - Below is an example with an actual number:

```
Attempted access at invalid index of 5.
```

3.4 InvalidSizeException

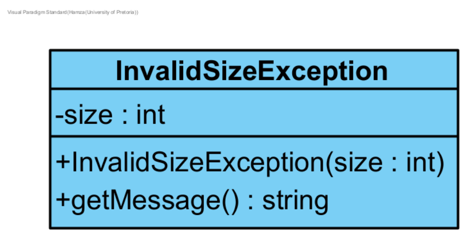

Figure 4: UML for InvalidSizeException

3.4.1 Members

- Member 1
  - This refers to the `size` value that caused the exception to the thrown.

3.4.2 Functions

- Function 1
  - This function should initialize the `size` member variable to the passed-in parameter.
- Function 2
  - This is a virtual function.
  - This is a constant function.
  - This function should return a string in the following format:
    `"Invalid size of <SIZE> was passed in."`
  - Below is an example with an actual number:

```
Invalid size of 7 was passed in.
```

3.5 ElementNotInListException

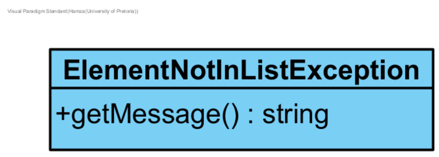

Figure 5: UML for ElementNotInListException

3.5.1 Functions

- Function 1

  - This is a virtual function.
  - This is a constant function.
  - This function should return the string:
    `"Element was not in the list."`

3.6 OutOfBoundsException

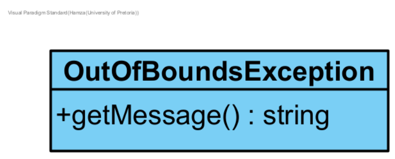

Figure 6: UML for OutOfBoundsException

3.6.1 Functions

- Function 1
  - This is a virtual function.
  - This is a constant function.
  - This function should return the string:
    `"The iterator went outside of the bounds of the iterable object."`

3.7 TooManyPrintersDeleted

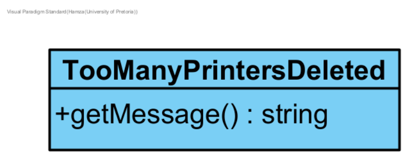

Figure 7: UML for TooManyPrintersDeleted

3.7.1 Functions

- Function 1
  - This is a virtual function.
  - This is a constant function.
  - This function should return the string:
    `"More printer references were returned than were requested."`

3.8 PrinterNotInitialized

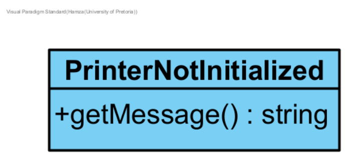

Figure 8: UML for PrinterNotInitialized

3.8.1 Functions

- Function 1
  - This is a virtual function.
  - This is a constant function.
  - This function should return the string:
    `"Attempted to print or add a document, but no printer has been requested yet."`

3.9 Array

This is an array wrapper class that is used to store values.

Type parameters:

T — is the template type that gets stored in the `Array`

U — is the template type that gets used for the `Array` in the stream insertion and equals operator.

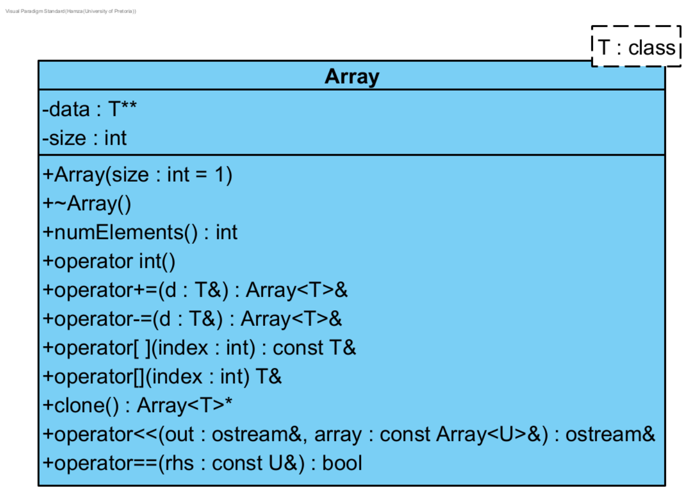
Visual Paradigm Standard(Hamza (University of Pretoria))
T : class
Array
-data : T**
-size : int
+Array(size : int = 1)
+~Array()
+numElements() : int
+operator int()
+operator+=(d : T&) : Array<T>&
+operator-=(d : T&) : Array<T>&
+operator[ ](index : int) : const T&
+operator[](index : int) T&
+clone() : Array<T>*
+operator<<(out : ostream&, array : const Array<U>&) : ostream&
+operator==(rhs : const U&) : bool

Figure 9: UML for Array

3.9.1 Members

- Member 1
  - This is a 1D dynamic array of dynamic T objects, and may contain `NULL`s.

- Member 2
  - This is the current size of the array.

3.9.2 Functions

- Function 1
  - This function should initialize the array to the passed-in `size`.
  - The `size` parameter should have a default value of 1 in your class declaration.
  - The `data` array must initially be filled with `NULL`s.
  - This function throws an `InvalidSizeException` if the `size` is less than or equal to zero.
- Function 2
  - This is a virtual function.
  - This function should deallocate any dynamically allocated memory.
- Function 3
  - This is a constant function.
  - This function should count and return the number of elements in the array.
- Function 4
  - This is a constant function.
  - This function should return the size of the array.
- Function 5
  - This is a virtual function.
  - This function should insert the passed-in `data` into the array at the first `NULL` position.
  - The new data must be inserted into the array using `T`'s copy constructor.
  - If the array is full, double its size and insert at the first `NULL` position.
  - This function should return the modified `Array` object.
- Function 6
  - This is a virtual function.
  - This function will remove the passed-in element from the array.
  - It throws an `ElementNotInListException` if the passed-in element is not in the array.
  - If the array is less than half-full after removing an element, the size of the array must be halved.

  - When halving the array, you should initially fill the new array that you create with **NULL**s. You should populate the new array in such a way that there are no gaps between objects in the array; so there will only be **NULLs** on the far end of the array.
  - This function should return the modified `Array` object.

- Function 7
  - This is a virtual function.
  - This is a constant function.
  - The function signature will be "`const T& operator[](int index)const`".
  - This function should return the value in the Array at the passed-in index.
  - If the index is invalid or the pointer at the index is `NULL`, this function should throw an `InvalidIndexException`.
  - Remember to set the second parameter correctly when throwing an `InvalidIndexException`.

- Function 8
  - This is a virtual function.
  - This function should return the value in the Array at the passed-in index.
  - If the index is invalid or the pointer at the index is `NULL`, this function should throw an `InvalidIndexException`.
  - Remember to set the second parameter correctly when throwing an `InvalidIndexException`.

- Function 9
  - This is a virtual function.
  - This function should return a deep copy of the Array.

- Function 10
  - This is a friend function.
  - This function prints out the passed-in `Array` to the passed-in `ostream`, as a comma-delimited list.
  - Whenever a `NULL` is encountered in the list, the string value `"NULL"` should be appended to the output. Otherwise, the object in the array is appended.
  - Ensure that you do not have a trailing comma.
  - You should **not** include an `endline` and the end of your comma-delimited list.
  - This function should return the populated `ostream` object.

- Function 11
  - This is a constant function.

  - You may assume that the type parameter, “U”, will have the same public interface as the Array class.
  - This operator needs to determine if the passed-in object is equal to `this` array.
  - For the two objects to be equal, their sizes must match and each element at the same index must match.
  - It should return `true` if the two objects are equal, and `false` otherwise.
  - Remember that `operator[]` may throw an exception. In this case, you should simply return false.

3.10 List

This is an abstract class that defines the public interface of the List class hierarchy.

Type parameters:

T — is the template type that gets stored in the list.

U — is a type that can be assumed will have the same public interface as the List class.

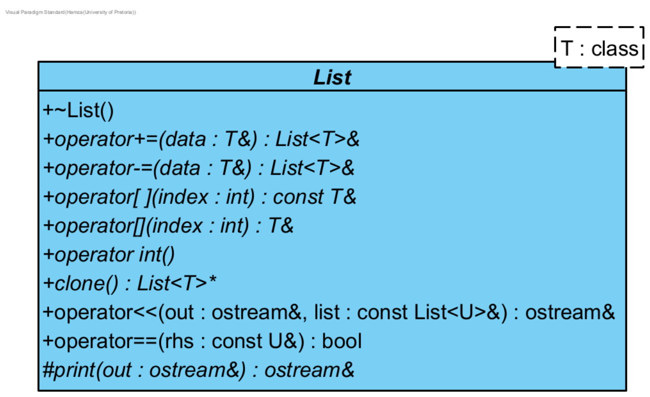
T : class
List
+~List()
+operator+=(data : T&) : List<T>&
+operator-=(data : T&) : List<T>&
+operator[ ](index : int) : const T&
+operator[](index : int) : T&
+operator int()
*+clone() : List<T>**
+operator<<(out : ostream&, list : const List<U>&) : ostream&
+operator==(rhs : const U&) : bool
#print(out : ostream&) : ostream&

Figure 10: UML for List

3.10.1 Functions

- Function 1
  - This is a virtual function.
  - This is the destructor for the class.
  - It should do nothing.
- Function 2
  - This function should add a node containing the passed-in `data` into the list and then return the modified list.
- Function 3
  - This function will remove the passed-in element from the list.

  - If an item was successfully removed, the modified version of the list is returned.
  - This function throws an `ElementNotInListException` if the passed-in element is not found in the list.

- Function 4
  - The function signature will be "`const T& operator[](int index)const`".
  - This function should return the value at the passed-in index.
  - If the index is invalid or if the pointer at the index is NULL,
    it should throw an `InvalidIndexException`.

- Function 5
  - This function should return the value at the passed-in index.
  - If the index is invalid or if the pointer at the index is NULL,
    it should throw an `InvalidIndexException`.

- Function 6
  - This function returns the size of the list.

- Function 7
  - This function returns a deep copy of the list.

- Function 8
  - This is a friend function.
  - This function prints out the passed-in `list` to the passed-in `ostream`, as a comma-delimited list.
  - Ensure that you do not have a trailing comma.
  - You should **not** include an `endline` and the end of your comma-delimited list.
  - This function should return the populated ostream object.

- Function 9
  - This is a constant function.
  - This operator needs to determine if the passed-in object is equal to the current list.
  - For the two objects to be equal, their sizes must match and each element at the same index must match.
  - It should return `true` if the two objects are equals, and false if they are not.
  - Remember that `operator[]` may throw an exception. In this case, you should simply return false.

- Function 10

– This function is responsible for populating the passed-in `ostream` object with the elements of *this* `List`.

– If the list is empty, the stream should only contain the text “EMPTY”. (without an endline)

– The `data` from each element of the `List` is inserted into the `ostream`, with each element’s data separated by a comma.

– You should **not** add an `endline` to the `ostream`.

– This function should return the populated `ostream` object

Example:
If the `List` contained nodes
→`[data=1]`→`[data=2]`→`[data=3]`→`[data=4]`
The expected result would be in the following format:

```
1,2,3,4
```

Note: the order in which elements are inserted into the `ostream` may differ among the `DLList`, `Queue` and `Stack` classes

3.11 DLList

This is a concrete class that implements the doubly linked list class. The data in this list is stored in ascending order.

Type parameter:

T - is the template type that gets stored in the doubly linked list.

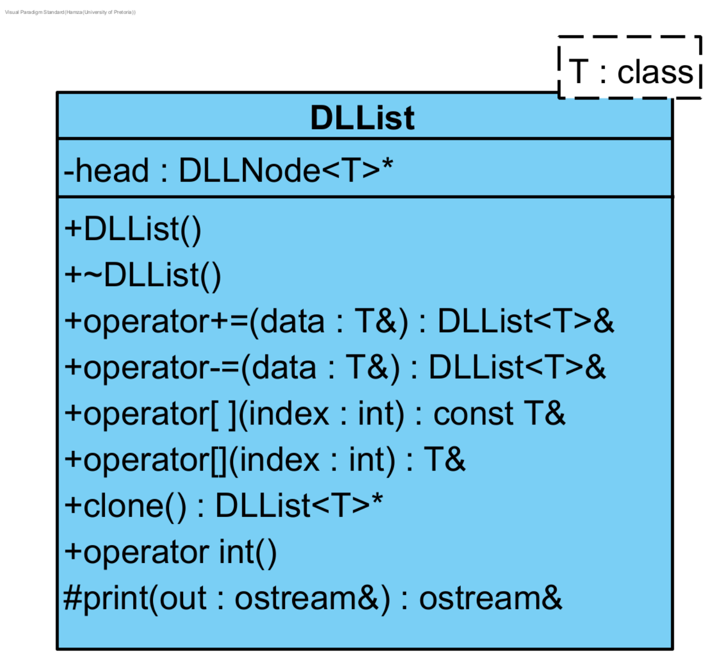
Visual Paradigm Standard(Hamza(University of Pretoria))
T : class
DLList
-head : DLLNode<T>*
+DLList()
+~DLList()
+operator+=(data : T&) : DLList<T>&
+operator-=(data : T&) : DLList<T>&
+operator[ ](index : int) : const T&
+operator[](index : int) : T&
+clone() : DLList<T>*
+operator int()
#print(out : ostream&) : ostream&

Figure 11: UML for DLList

3.11.1 DLLNode

This is the node struct for the dynamic linked list class.

Type parameter:

U - The template type that gets stored in the dynamic linked list.

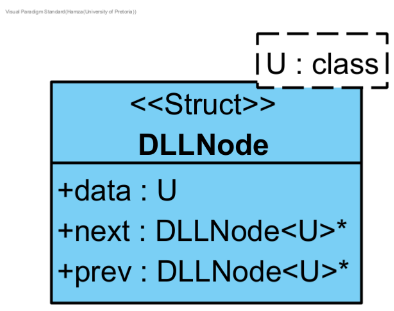

Figure 12: UML for DLLNode

- DLLNode Member 1
  - This is the data stored in the node.
- DLLNode Member 2
  - This is the node's next pointer.
- DLLNode Member 3
  - This is the node's previous pointer.

3.11.2 Members

- Member 1
  - This is the start of the doubly-linked list.

3.11.3 Functions

- Function 1
  - This function should initialize the `head` member variable to `NULL`.
- Function 2
  - This is a virtual function.
  - This function should deallocate the memory used by the entire list.
- Function 3
  - This function should create a node containing the passed-in `data`.

  - It will then add the new node into the list in **ascending order**, and then return the modified list.

- Function 4
  - This function should remove the first node with data matching the passed-in parameter from the list.
  - After this node is removed, the modified list should be returned.
  - If no node is found to contain `data` matching the passed-in parameter, the function throws an `ElementNotInListException`.
- Function 5
  - This is a constant function.
  - The function signature will be "`const T& operator[](int index)const`".
  - This function should return the value at the passed-in index.
  - If the index is invalid or the pointer at the index is `NULL`, throw an `InvalidIndexException`.
- Function 6
  - This function should return the value at the passed-in index.
  - If the index is invalid or the pointer at the index is `NULL`, throw an `InvalidIndexException`.
- Function 7
  - This function returns a deep copy of the list.
- Function 8
  - This function returns the size of the list.
- Function 9
  - This function is responsible for populating the passed-in `ostream` object with the elements of *this* `DLList`.
  - If the list is empty, the stream should only contain the text "EMPTY". (without an endline)
  - The `data` from each element of the `DLList` is inserted into the `ostream`, with each element's data separated by a comma.
  - You should **not** add an `endline` to the `ostream`.
  - This function should return the populated `ostream` object

3.12 Queue

This is a concrete class that implements a queue. The data in this queue is stored in first-in,first-out (FIFO) order.

Type Parameter:

T — is the template type that gets stored in the queue.

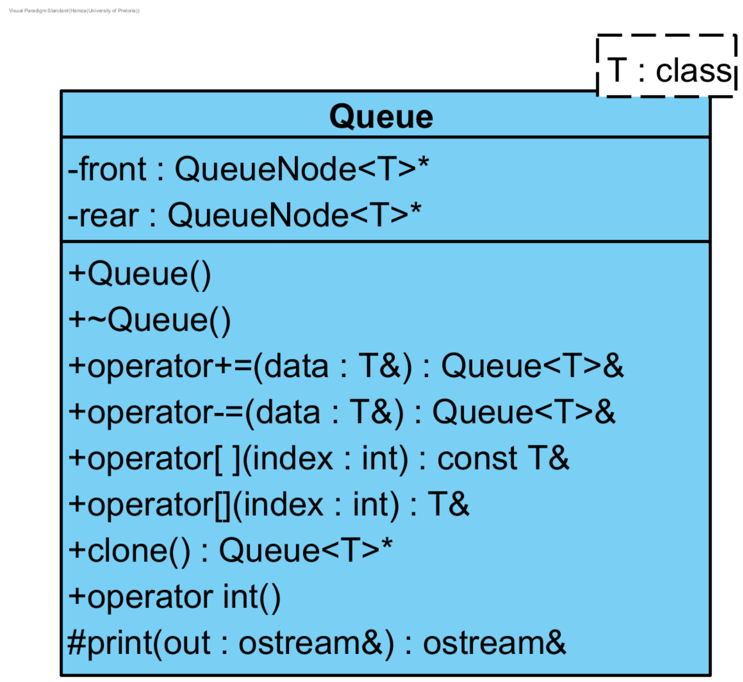
T : class
Queue
-front : QueueNode<T>*
-rear : QueueNode<T>*
+Queue()
+~Queue()
+operator+=(data : T&) : Queue<T>&
+operator-=(data : T&) : Queue<T>&
+operator[ ](index : int) : const T&
+operator[](index : int) : T&
+clone() : Queue<T>*
+operator int()
#print(out : ostream&) : ostream&

Figure 13: UML for Queue

3.12.1 QueueNode

This is the node struct for the queue class.

Type Parameter:

U — The template type that gets stored in the queue.

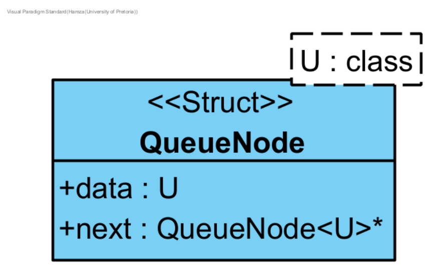

Figure 14: UML for QueueNode

- QueueNode Member 1
  - This is the data stored in the node.

- QueueNode Member 2
  - This is the node's next pointer.

3.12.2 Members

- Member 1
  - This is the front of the queue.
- Member 2
  - This is the rear of the queue.

3.12.3 Functions

- Function 1
  - This constructor should initialize the `front` and `rear` member variables to NULL.
- Function 2
  - This is a virtual function.
  - This function should deallocate the entire list.
- Function 3
  - This function should create a node containing the passed-in `data`.
  - It will then add the new node to the rear of the list, and then return the modified list.
- Function 4
  - This function will remove the front element and store it in the passed-in reference.
  - If the list is empty, this function should return `*this`.
  - Otherwise, the function should return the modified list (which has one less element).
- Function 5
  - The function signature will be "`const T& operator[](int index)const`".
  - This function should return the value at the passed-in index.
  - If the index is invalid or the pointer at the index is `NULL`, throw an `InvalidIndexException`.
- Function 6
  - This function should return the value at the passed-in index.
  - If the index is invalid or the pointer at the index is `NULL`, throw an `InvalidIndexException`.
- Function 7

  - This function returns a deep copy of the list.

- Function 8
  - This function returns the size of the list.

- Function 9
  - This function is responsible for populating the passed-in `ostream` object with the elements of *this* `Queue`.
  - If the list is empty, the stream should only contain the text "EMPTY". (without an endline)
  - The `data` from each element of the `List` is inserted into the `ostream`, with each element's data separated by a comma.
  - You should **not** add an `endline` to the `ostream`.
  - This function should return the populated `ostream` object

3.13 Stack

This is a concrete class that implements the stack class. The data in this stack is stored in last-in,first-out LIFO order.

Type Parameter:

T — is the template type that gets stored in the stack.

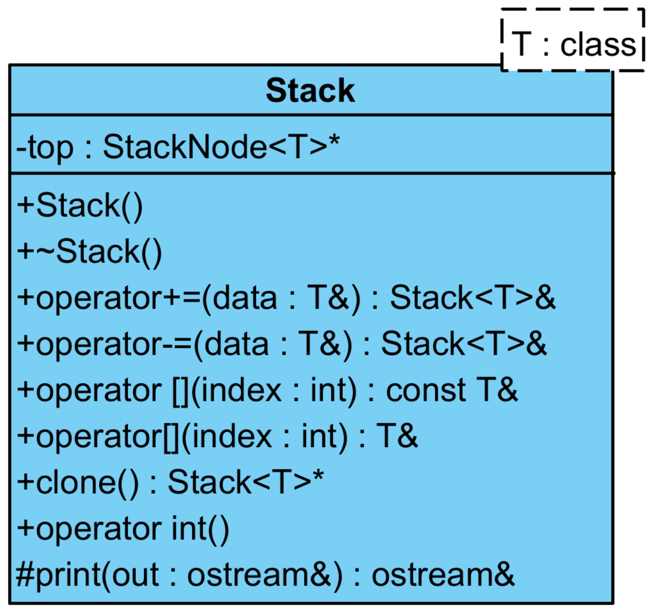
T : class
Stack
-top : StackNode<T>*
+Stack()
+~Stack()
+operator+=(data : T&) : Stack<T>&
+operator-=(data : T&) : Stack<T>&
+operator [](index : int) : const T&
+operator[](index : int) : T&
+clone() : Stack<T>*
+operator int()
#print(out : ostream&) : ostream&

Figure 15: UML for Stack

3.13.1 StackNode

This is the node struct for the stack class.

Type Parameter:

U — The template type that gets stored in the stack.

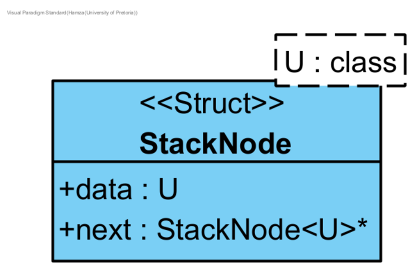

Figure 16: UML for StackNode

- StackNode Member 1
  - This is the data stored in the node.
- StackNode Member 2
  - This is the node's next pointer.

3.13.2 Members

- Member 1
  - This is the top of the stack.

3.13.3 Functions

- Function 1
  - This constructor should initialize the `head` member variable to `NULL`.
- Function 2
  - This is a virtual function.
  - This function should deallocate the entire list.
- Function 3
  - This function will add a `StackNode` containing the passed-in `data` to the list and then return the modified list.
- Function 4
  - This function will remove the top element and store it in the passed-in reference parameter.
  - If the stack is empty, the function should return **this* and not affect the value of the passed-in parameter.
  - Otherwise, the function will return the modified list (with one less element).
- Function 5

  - The function signature will be "`const T& operator[](int index)const`".
  - This function should return the value at the passed-in index.
  - If the index is invalid or the pointer at the index is `NULL`, throw an `InvalidIndexException`.

- Function 6
  - This function should return the value at the passed-in index.
  - If the index is invalid or the pointer at the index is `NULL`, throw an `InvalidIndexException`.

- Function 7
  - This function returns a deep copy of the list.

- Function 8
  - This function returns the size of the list.

- Function 9
  - This function is responsible for populating the passed-in `ostream` object with the elements of *this* `Stack`.
  - If the list is empty, the stream should only contain the text "EMPTY". (without an endline)
  - The `data` from each element of the `List` is inserted into the `ostream`, with each element's data separated by a comma.
  - You should **not** add an `endline` to the `ostream`.
  - This function should return the populated `ostream` object

3.14 Iterator

This is an iterator class that loops through the passed-in template type named Iterable.
Type Parameters:
Iterable — this is the type of the container that the Iterator class will move through. It must be a template class that takes one type parameter
T — This is the template type that gets stored in the Iterable type.
Hint: The template preamble for this class should be:

```
template<template <class> class Iterable, class T>
```

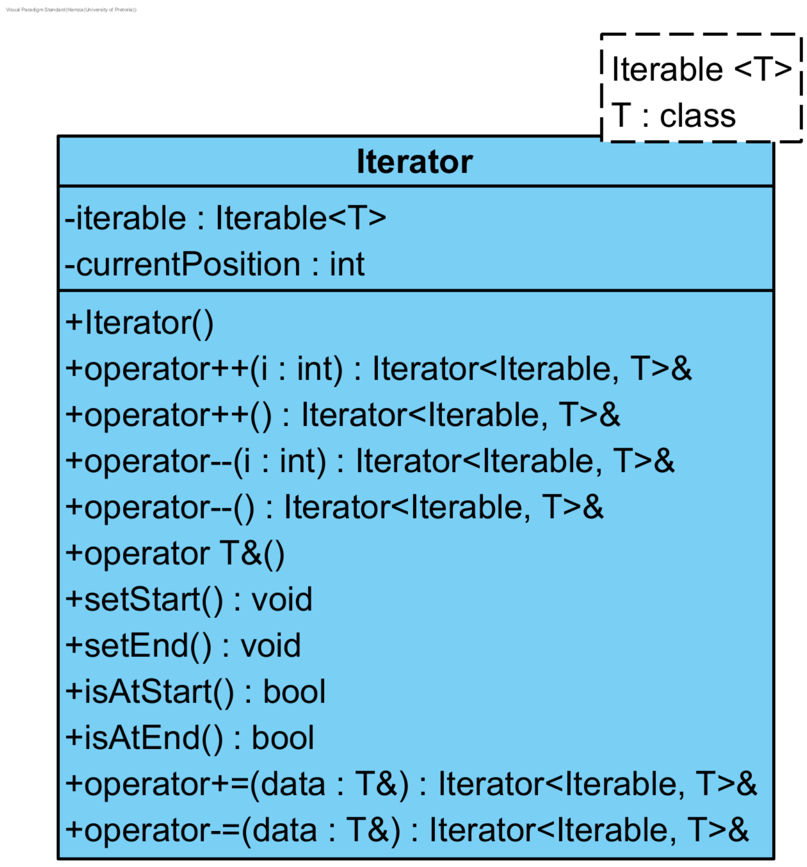
Visual Paradigm Standard(Hamza(University of Pretoria))
Iterable <T>
T : class
Iterator
-iterable : Iterable<T>
-currentPosition : int
+Iterator()
+operator++(i : int) : Iterator<Iterable, T>&
+operator++() : Iterator<Iterable, T>&
+operator--(i : int) : Iterator<Iterable, T>&
+operator--() : Iterator<Iterable, T>&
+operator T&()
+setStart() : void
+setEnd() : void
+isAtStart() : bool
+isAtEnd() : bool
+operator+=(data : T&) : Iterator<Iterable, T>&
+operator-=(data : T&) : Iterator<Iterable, T>&

Figure 17: UML for Iterator

3.14.1 Members

- Member 1
  - This is the member variable that will be iterated through.
- Member 2
  - This is the current position in the iterable member variable.

3.14.2 Functions

- Function 1
  - This constructor should set the `currentPosition` to 0.
- Function 2
  - This function should move the `currentPosition` up by 1 and return the modified *this.
  - You should perform bounds checking before increasing the `currentPosition`
  - If the `currentPosition` is too large for the `iterable`, this function should throw an `OutOfBoundsException`.
- Function 3

  - This function should move the `currentPosition` up by 1 and return the modified *this.
  - You should perform bounds checking before increasing the `currentPosition`
  - If the `currentPosition` is too large for the `iterable`, this function should throw an `OutOfBoundsException`.

- Function 4
  - This function should move the `currentPosition` down by 1 and return the modified *this.
  - You should perform bounds checking before decreasing the `currentPosition`
  - If the `currentPosition` is negative, this function should throw an `OutOfBoundsException`.
  - This function should return the modified *this.

- Function 5
  - This function should move the `currentPosition` down by 1 and return the modified *this.
  - You should perform bounds checking before decreasing the `currentPosition`
  - If the `currentPosition` is negative, this function should throw an `OutOfBoundsException`.
  - This function should return the modified *this.

- Function 6
  - This function should return the value stored at the `currentPosition` index in the iterable member variable.

- Function 7
  - This function should set the `currentPosition` to 0.

- Function 8
  - This function should set the `currentPosition` to the size of the iterable member variable

- Function 9
  - This function should determine if the `currentPosition` is at the start of the iterable member variable.
  - If the `currentPosition` is at the start, this function should return true. Otherwise, it should return false.

- Function 10
  - This function should determine if the `currentPosition` has the value of the size of the iterable member variable.
  - If the `currentPosition` is at the size of the iterable, this function should return true. Otherwise, it should return false.

- Function 11

    - This function should insert the data into the iterable member variable and then return the modified **this*.

- Function 12
    - This function should remove the data from the iterable member variable using the iterable member variable's `operator-=` operation.
    - Then the function should return the modified **this*.

3.15 Printer

The `Printer` class is able to output documents of any data type. The documents can be printed in a different order depending on the type of `List` class used as a template parameter.

This class is based on the singleton design pattern, which ensures that only one instance of a class will ever exist throughout the program's execution. Having one printer shared throughout your program is more efficient than having several printers.

The details on the Singleton pattern will be covered in COS214. For enrichment, you can read more about the pattern here: `https://refactoring.guru/design-patterns/singleton`

Type Parameters:

Iterable — This is the container class that stores all the objects of type T that needs to be printed.

T — this is the type that will be printed.

Hint: The template preamble for this class must be:

```
template<template <class> class Iterable, class T>
```

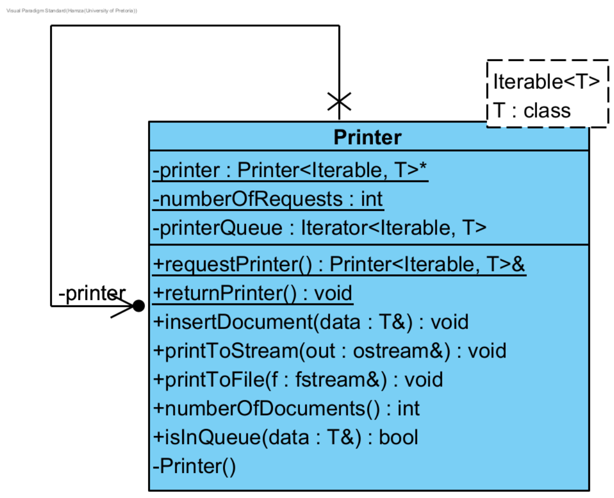
Visual Paradigm Standard(Hamza(University of Pretoria))
Iterable<T>
T : class
Printer
-printer : Printer<Iterable, T>*
-numberOfRequests : int
-printerQueue : Iterator<Iterable, T>
+requestPrinter() : Printer<Iterable, T>&
+returnPrinter() : void
+insertDocument(data : T&) : void
+printToStream(out : ostream&) : void
+printToFile(f : fstream&) : void
+numberOfDocuments() : int
+isInQueue(data : T&) : bool
-Printer()
-printer

Figure 18: UML for Printer

3.15.1 Members

- Printer Member 1

- This is the printer pointer that is used to print out values
- It should be initialized to NULL.

- Printer Member 2
  - This is the number of printer reference requests that are outstanding.
  - It should be initialized to 0.
- Printer Member 3
  - This is the iterator that is responsible for the moving through the queue of documents.

3.15.2 Functions

- Function 1
  - This function requests a printer reference and should make a printer if the pointer is NULL.
  - It should also increase the `numberOfRequests`.
- Function 2
  - This function should decrement the `numberOfRequests`.
  - If the `numberOfRequests` is 0, deallocate the printer pointer.
  - Otherwise, if the numberOfRequests becomes negative while calling this function, it should throw a `TooManyPrintersDeleted` exception.
- Function 3
  - This function should insert the data into the printer queue.
  - If the `printer` member variable is `NULL`, the function should throw a `PrinterNotInitialized` exception.
- Function 4
  - This function should iterate through the printer's queue, from start to end, and place the data from the queue into the `out` reference parameter.
  - The passed-in parameter, `out`, should be populated in the same order in which values are removed from the queue.
  - Each element from `printerQueue` should be on a new line when inserted into `out`. Thus, include an endline after each insertion.
  - If an `InvalidIndexException` is caught in this function, your function should output the text "NULL", terminated with a newline.
- Function 5

  - This function should iterate through the printer's queue, from start to end, and place the data from the queue into the `f` reference parameter.
  - The passed-in parameter, `f`, should be populated in the same order in which values are removed from the queue.
  - Each element from `printerQueue` should be on a new line when inserted into `f`. Thus, include an endline after each insertion.

- Function 6
  - This function should return the size of the printerQueue.
- Function 7
  - This function should determine if the passed-in `data` is in the queue.
  - It should return `true` if the `data` is contained in the queue, and `false` if it is not.
- Function 8
  - This is an empty constructor.

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}
```

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

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.

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:
  - fstream
  - sstream
  - string
  - iostream

7 Upload Checklist

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

- `Exception.cpp`
- `InvalidIndexException.cpp`
- `InvalidSizeException.cpp`
- `ElementNotInListException.cpp`
- `OutOfBoundsException.cpp`
- `TooManyPrintersDeleted.cpp`
- `PrinterNotInitialized.cpp`
- `Array.cpp`
- `List.cpp`
- `DLList.cpp`
- `Stack.cpp`
- `Queue.cpp`
- `iterator.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++ -g -std=c++98 -Werror -Wall main.cpp \
    Exception.cpp \
    InvalidIndexException.cpp \
    InvalidSizeException.cpp \
    ElementNotInListException.cpp \
    OutOfBoundsException.cpp \
    TooManyPrintersDeleted.cpp \
    PrinterNotInitialized.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 5 submissions and your best mark will be your final mark. Upload your archive to the Assignment 3 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!**

9 Change Log

Change	Date
Updated UML for List Hierarchy, Array, Printer and Iterator for include missing object conversion functions	20/10/2025
Improved explanations and descriptions throughout the specification	21/10/2025
Added example output additional functions and provided more detailed explanations for List functions Updated UML to explicitly show all Template Parameters and added inheritance type	22/10/2025
Added explicit Return False when exception thrown in operator==	23/10/2025