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

Department of Computer Science
COS110 - Program Design: Introduction
Practical 8

1 Introduction

Deadline: 19th of November, 19:30

1.1 Objectives and Outcomes

The objective of this practical is to test your understanding of the programming concepts covered in the theory classes. In particular, this practical will test your understanding of circular doubly-linked lists.

1.2 Plagiarism

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying textual material from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to **http://www.ais.up.ac.za/plagiarism/index.htm** (from the main page of the University of Pretoria site, follow the *Library* quick link, and then click the *Plagiarism* link). If you have questions regarding this, please ask one of the lecturers, to avoid any misunderstanding.

1.3 Implementation Guidelines

Follow the specifications of the practical precisely. For each practical, you will be required to create your own makefile so pay attention to the names of the files you will be asked to create. If the practical requires you to submit additional files of your own, follow the file structure and format exactly. Incorrect submissions will use up your uploads and no extensions will be given. In terms of C++, unless otherwise stated, the usage of C++11 or additional libraries outside of those indicated in the practical, will not be allowed. Some of the appropriate files that you submit will be overwritten during marking to ensure compliance to these requirements. If the specification makes use of text files, for providing input information, be sure to include blank text files with the specified names.

2 Practical

Please consult the UML diagram below and read the specification fully before coding.

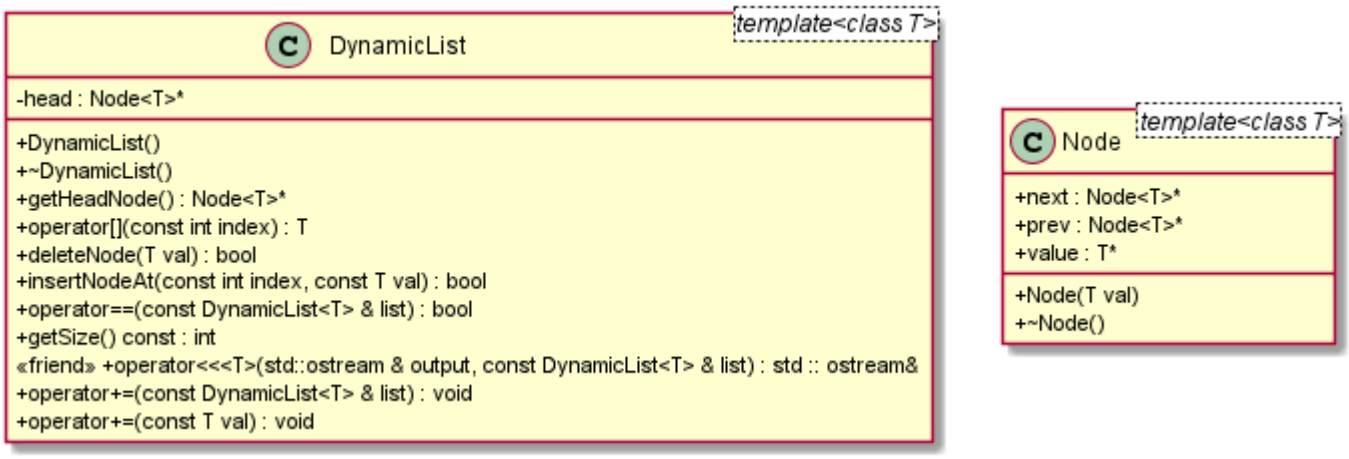

Figure 1: UML Diagrams for the two classes to implement.

Node

Each node has a pointer to the `next` node in the list as well as the `prev` node in the list. Each node also stores a pointer to a value of type `T`. Take note of the following:

- The constructor of this class sets `next` and `prev` to `NULL`. The constructor also creates a new pointer to a value of type `T` based on the value passed in as a parameter.
- The destructor deletes the `value` stored in this class.

DynamicList

This class is a **Circular Doubly-Linked List**. Each node in this list has a `next` pointer and a `prev` pointer (which makes it doubly linked). The head of the list has a `prev` pointer to the tail of the list, and the `tail` of the list has a `next` pointer to the `head` of the list (making the list circular). Here are some notes:

- The constructor of this class sets the `head` initially to `NULL`.
- The destructor of this class deletes each node in the list.
- `getSize` returns the number of nodes in the list. If the list is empty return zero.
- `getHeadNode` returns the `head` node member variable.
- `operator+=(const T)` is an operator overload which creates a new node with the value passed in as a parameter and appends this new node to the list. If the list is empty then the new node created becomes the `head` node of the list.
- `operator+=(const DynamicList<T>& list)` is an operator overload which appends the list passed as a parameter to `this` list. A deep copy should be created of each node in the list passed as a parameter before it is appended to the current list. If `this` list is empty, this function essentially clones the provided `list` parameter into `this` list (deep copy). If the `list` parameter is an empty list, do nothing.

- `operator[]` is an overload of the subscript operator. This function should return the value of the `Node` at the index passed in as a parameter. If the index is larger than the size of the list, the list should be traversed in a circular fashion until that index is found (because the list is circular). If the index is negative or the list is empty return `-1` (Note that this makes some assumptions about the template class types that will be used). Please see the example in Figure 2 for an example.

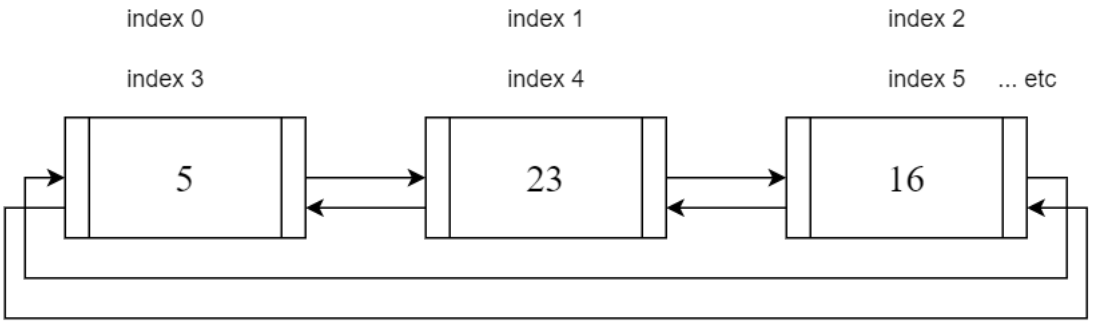

Figure 2: If index 5 is requested, the value `16` should be returned. If the index 6 is requested, the value `5` should be returned.

- `insertNodeAt` receives an index as well as a value of type `T` as a parameter. If the index is valid, a new node containing the value of type `T` should be inserted at the position in the list corresponding to the index. The index is valid if it is in the range $0 \leq index \leq size$ where *size* is the number of elements in the list. If the index is zero, then the new node will be the new head of the list. If the index is equal to *size*, then the new node will be the new tail of the list. Any index in-between zero and *size* will mean the new node is inserted in-between other nodes. If a new node was inserted successfully return `true`, otherwise return `false`.

- `deleteNode` receives a value of type `T` as a parameter. The first node to contain the value passed in as a parameter should be removed from the list. If a successful deletion occurs return `true` else return `false`.

- `operator==` is an operator overload that compares `this` list to the list passed as a parameter. If the lists contain the same values (in the same order) and are the same size, return `true`; otherwise, return `false`. If both lists are empty, they are considered to be equivalent.

- `operator<< <T>` is a template overloaded `ostream` operator which is a `friend` function. This function prints each node in the list in the format `A[B,C]` where `A` is the value of the current node, `B` is the value of the `prev` node and `C` is the value of the `next` node. Each node is separated by an arrow `->` with a space before and after the arrow. The last node does not have an arrow after it. **No** `endl` should be output after the list and there should be no whitespace after the last node is printed. If the list is empty, don't print anything. Here is an example print of the list in Figure 2, note there should be no `endl` after the list is printed:
  `5[16,23] -> 23[5,16] -> 16[23,5]`

  You may need to forward declare this operator. For an explanation of how friend templates work for operator overloads, consult this source: https://en.cppreference.com/w/cpp/language/friend

3 Includes

You should use `namespace std` in your `.cpp` files and not your `.h` files. You may have the following includes:

- cstddef
- iostream
- `X.h` or `X.cpp` where `X` is a class defined in this specification.

4 Submission

You need to submit your source files on the Fitch Fork website (https://ff.cs.up.ac.za/). Place all of your `.h` files and your `.cpp` files in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number. Also place your **makefile** in this archive. There is no need to include any other files in your submission. **Do not put any folders in the .zip archive**. You have 10 submissions and your best mark will be your final mark. Do not use Fitch Fork to test your code because you have limited marking opportunities. Upload your archive to the Practical 8 slot on the Fitch Fork website for COS110. Submit your work before the deadline. No late submissions will be accepted!